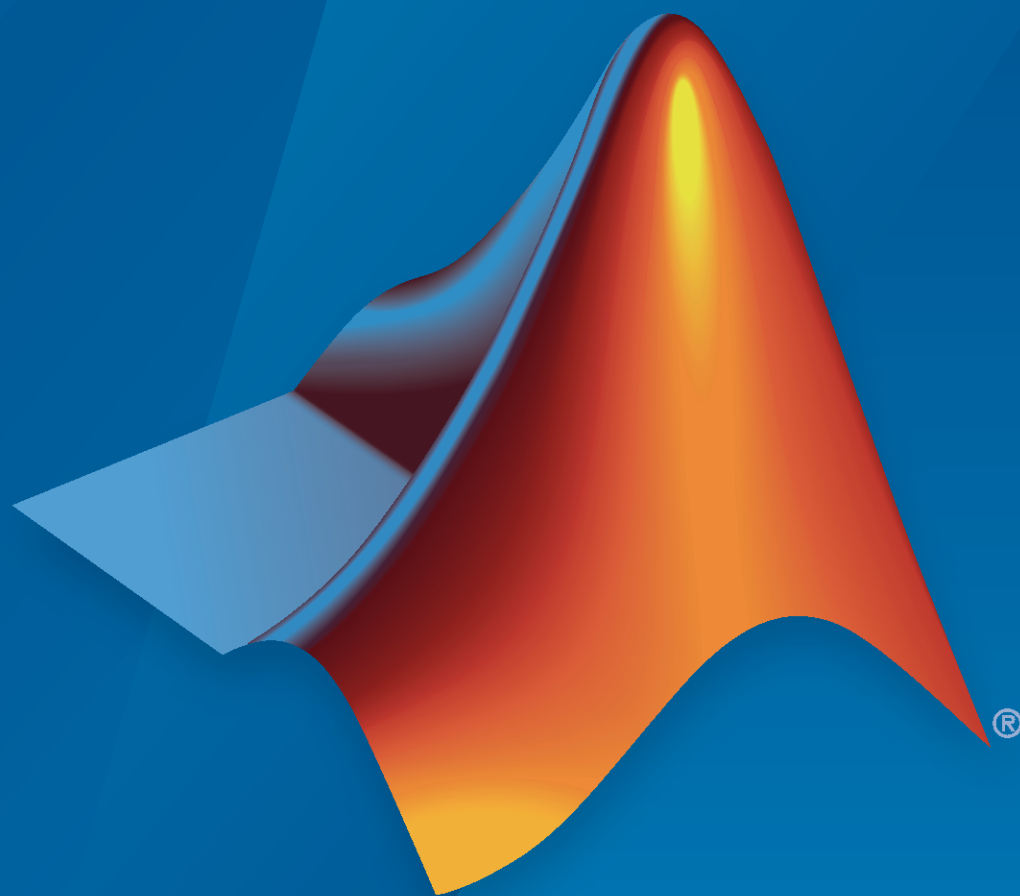# Mapping Toolbox™

Reference

MATLAB®

MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

Technical support: www.mathworks.com/support/contact_us

Phone: 508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

Contents

Functions

Map Viewer

View and explore data in map coordinates

Description

The Map Viewer app enables you to view geospatial data in map (*x-y*) coordinates. The Map Viewer works with vector, image, and raster data grids in a map coordinate system. You can pan and zoom on the map, specify the map scale of your screen display, and control the order, visibility, and symbolization of map layers. In addition, you can add annotations to your map and click to learn more about individual vector features.
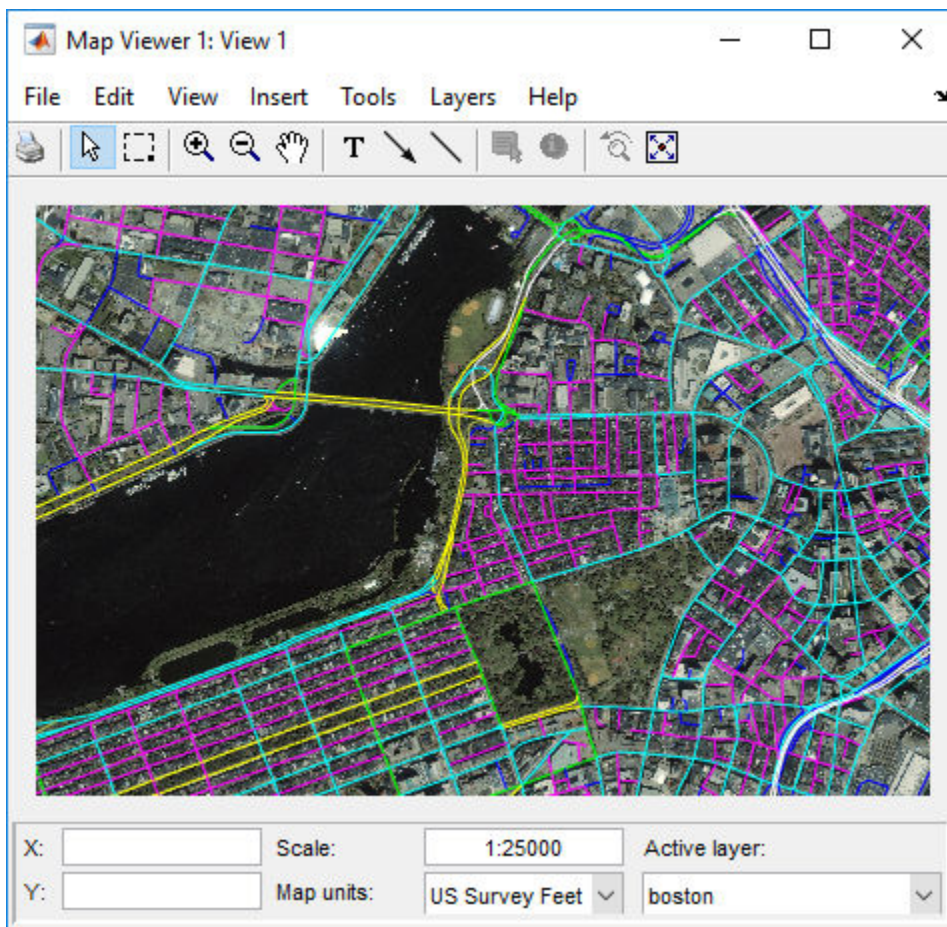
Open the Map Viewer App

- MATLAB® Toolstrip: Open the **Apps** tab, under **Image Processing and Computer Vision**, click the app icon.
- MATLAB command prompt: Enter `mapview`.

Examples

*   "Tour Boston with the Map Viewer App"

Limitations

The **Select area** tool is not supported in MATLAB Online™. To view a particular region on the map, use the **Zoom in**, **Zoom out**, and **Pan** tools instead.

See Also

Functions
mapshow

Topics
"Tour Boston with the Map Viewer App"

Introduced before R2006a

addCustomBasemap

Add custom basemap

Syntax

```
addCustomBasemap(basemapName,URL)
addCustomBasemap( ___ ,Name,Value)
```

Description

addCustomBasemap(basemapName,URL) adds the custom basemap specified by URL to the list of basemaps available for use with mapping functions. basemapName is the name you choose to call the custom basemap. Added basemaps remain available for use in future MATLAB sessions.

You can use custom basemaps with several types of map displays, for example, web maps created using the webmap function, geographic globes created using the geoglobe function, and geographic axes created using the geoaxes function.

addCustomBasemap(___ ,Name,Value) specifies name-value pairs that set additional parameters of the basemap.

Examples

Add Basemap from OpenStreetMap

Display locations on a geographic bubble chart using a basemap from OpenStreetMap®.

Define the name that you will use to specify your custom basemap.

```
name = 'openstreetmap';
```

Specify the website that provides the map data. The first character of the URL indicates which server to use to get the data. For load balancing, the provider has three servers that you can use: a, b, or c.

```
url = 'a.tile.openstreetmap.org';
```

Create an attribution to display on the map that gives credit to the provider of the map data. Web map providers might define specific requirements for the attribution.

```
copyright = char(uint8(169));
attribution = copyright + "OpenStreetMap contributors";
```

Add the custom basemap to the list of basemap layers available.

```
addCustomBasemap(name,url,'Attribution',attribution)
```

Plot the locations over the map using a geographic bubble chart. You can specify your custom basemap when you create the geographic bubble chart.

```
lat = [42.3501 42.3515 42.3598 42.3584 42.3529];
lon = [-71.0870 -71.0926 -71.0662 -71.0598 -71.0662];
```

```
gb = geobubble(lat,lon,'Basemap','openstreetmap');
gb.BubbleWidthRange = 25;
gb.MapLayout = 'maximized';
gb.ZoomLevel = 14;
```
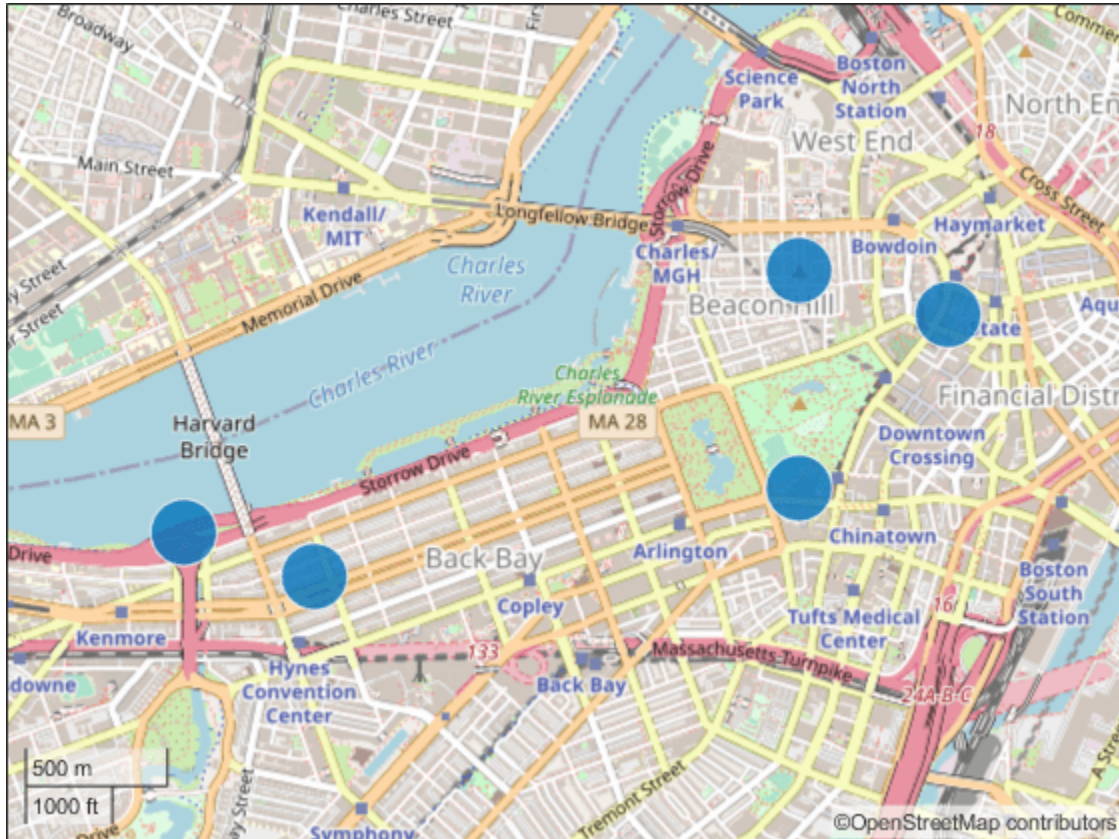

Add Basemap from USGS National Map

Display the route of a glider in 2-D and 3-D using a topographic basemap from the USGS National Map.

Before adding the basemap, specify the location of the map tiles. To do this, specify the URL of the National Map ArcGIS REST Services Directory. Then, add the path to the map tiles from the USGS Topo basemap service.

```
url = "https://basemap.nationalmap.gov/ArcGIS/rest/services";
fullurl = url + "/USGSTopo/MapServer/tile/${z}/${y}/${x}";
```

Specify a name for the basemap and attribution text to display with it.

```
nm = 'usgstopo';
att = 'Credit: US Geological Survey';
```

Finally, add the USGS Topo basemap.

```
addCustomBasemap(nm,fullurl,'Attribution',att)
```

Plot the path of a glider over the basemap. To do this, import latitude, longitude, and geoid height values.

```
trk = gpxread('sample_mixed','FeatureType','track');
lat = trk.Latitude;
lon = trk.Longitude;
h = trk.Elevation;
```

Display the path in 2-D using geographic axes. Specify the basemap using the `geobasemap` function and the name of the basemap. Use the basemap name given when you created it. Call `hold on` before plotting the line to prevent the basemap from resetting.

```
geoplot(lat,lon)
geobasemap('usgstopo')
hold on
geoplot(lat,lon,'r')
```
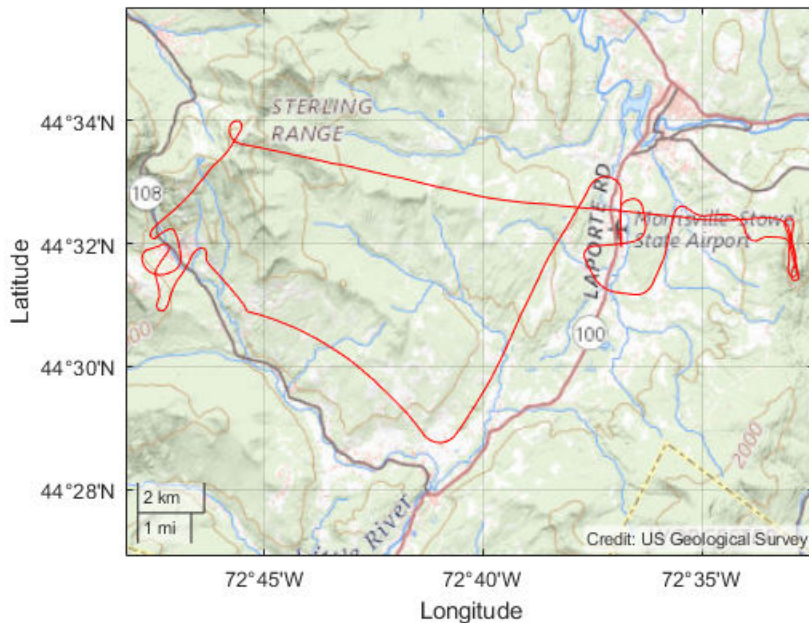

Display the path in 3-D using a geographic globe. Specify the basemap using the `'Basemap'` name-value pair argument. By default, the view is directly above the path. Tilt the view by holding **Ctrl** and dragging.

```
uif = uifigure;
g = geoglobe(uif,'Basemap','usgstopo');
hold(g,'on')
geoplot3(g,lat,lon,h,'r')
```

Add Basemap from OpenTopoMap

Display the route of a glider on a web map using a basemap from OpenTopoMap.

Define the name that you will use to specify your custom basemap.

```
name = 'opentopomap';
```

Specify the website that provides the map data. The first character of the URL indicates which server to use to get the data. For load balancing, the provider has three servers that you can use: a, b, or c.

```
url = 'a.tile.opentopomap.org';
```

Create an attribution to display on the map that gives credit to the provider of the map data. Web map providers might define specific requirements for the attribution.

```
copyright = char(uint8(169));
attribution = [ ...
     "map data:  " + copyright + "OpenStreetMap contributors,SRTM", ...
     "map style: " + copyright + "OpenTopoMap (CC-BY-SA)"];
```

Define the name that will appear in the Layer Manager to represent your custom basemap.

```
displayName = 'Open Topo Map';
```

Add the custom basemap to the list of available basemap layers.

```
addCustomBasemap(name,url,'Attribution',attribution, ...
                      'DisplayName',displayName)
```

Open a web map. Specify the custom basemap using the name you defined when you added it. Then, read glider data into the workspace and plot it over the basemap.

```
webmap opentopomap
trk = gpxread('sample_mixed.gpx','FeatureType','track');
wmline(trk,'LineWidth',2)
```
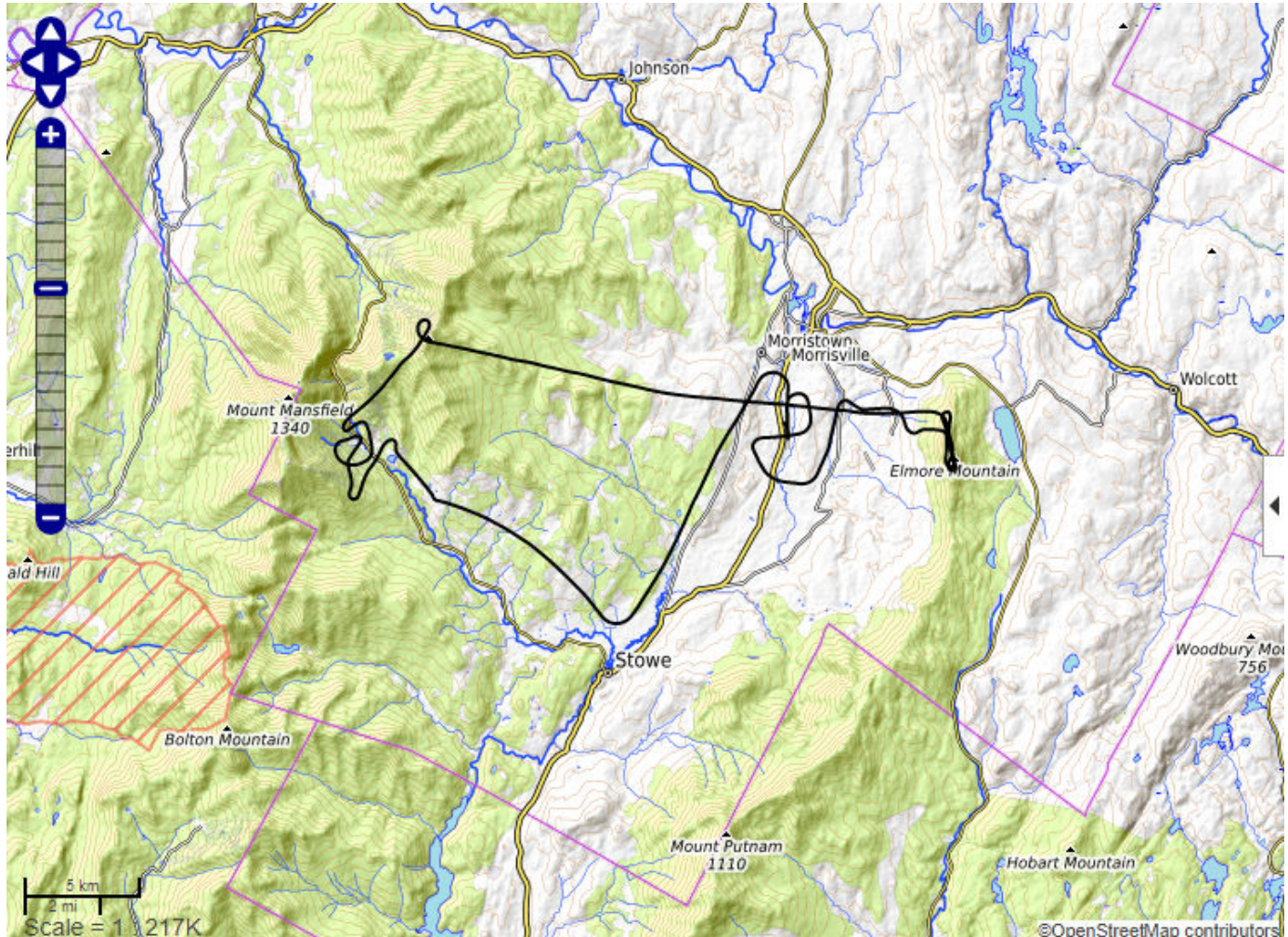

Add Several Basemaps from USGS National Map

Plot the path of a glider over a variety of maps that are available from the USGS National Map site. Viewing data over several maps can help you determine which map provides the best background.

Read in the glider path track data.

```
trk = gpxread('sample_mixed.gpx','FeatureType','track');
```

Specify the custom basemap URL. The USGS National Map supports several tiled web maps. For this example, insert the word BASEMAP into the URL string. In this way, you can replace the word BASEMAP with the name of one of the maps supported by the USGS National Map.

```
baseURL = "https://basemap.nationalmap.gov/ArcGIS/rest/services";
usgsURL = baseURL + "/BASEMAP/MapServer/tile/${z}/${y}/${x}";
```

Specify a list of the names of USGS basemaps that you want to use. These names will be inserted into the URL in place of BASEMAP.

```matlab
basemaps = ["USGSImageryOnly" "USGSImageryTopo" "USGSTopo" "USGSHydroCached"];
```

Specify a list of display names that you can use with each map. Use display names that are the same as those used by webmap so that webmap does not contain duplicate maps.

```matlab
displayNames = ["USGS Imagery" "USGS Topographic Imagery" ...
        "USGS Shaded Topographic Map" "USGS Hydrography"];
maxZoomLevel = 16;
```

Create a map attribution to give credit to the provider of the map data.

```matlab
attribution = "Credit: U.S. Geological Survey";
```

Create a loop in which you use each basemap with the geoplot function, plotting the glider data on each of the USGS basemaps.

```matlab
for k =1:length(basemaps)
    basemap = basemaps(k);
    name = lower(basemap);
    url = replace(usgsURL,"BASEMAP",basemap);
    displayName = displayNames(k);
    addCustomBasemap(name,url,"Attribution",attribution, ...
            "DisplayName",displayName,"MaxZoomLevel",maxZoomLevel)
    figure
    geoplot(trk.Latitude,trk.Longitude,"r","LineWidth",5);
    geobasemap(basemap)
    title(displayName)
end
```

USGS Imagery

USGS Topographic Imagery
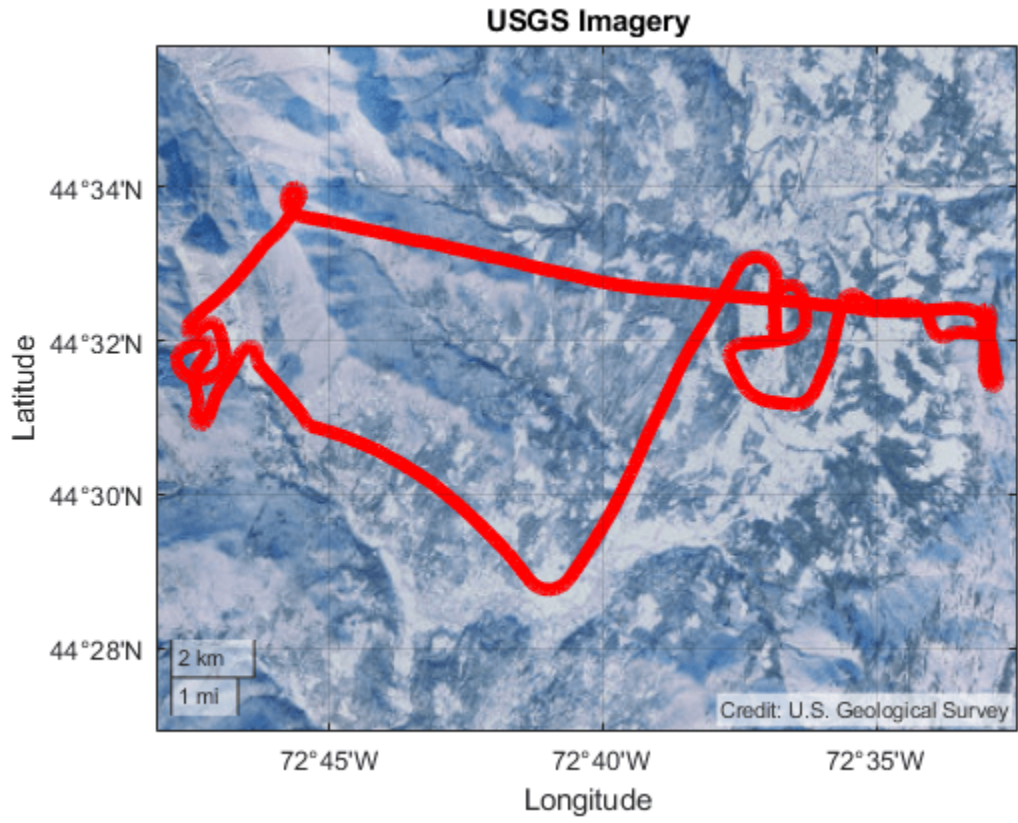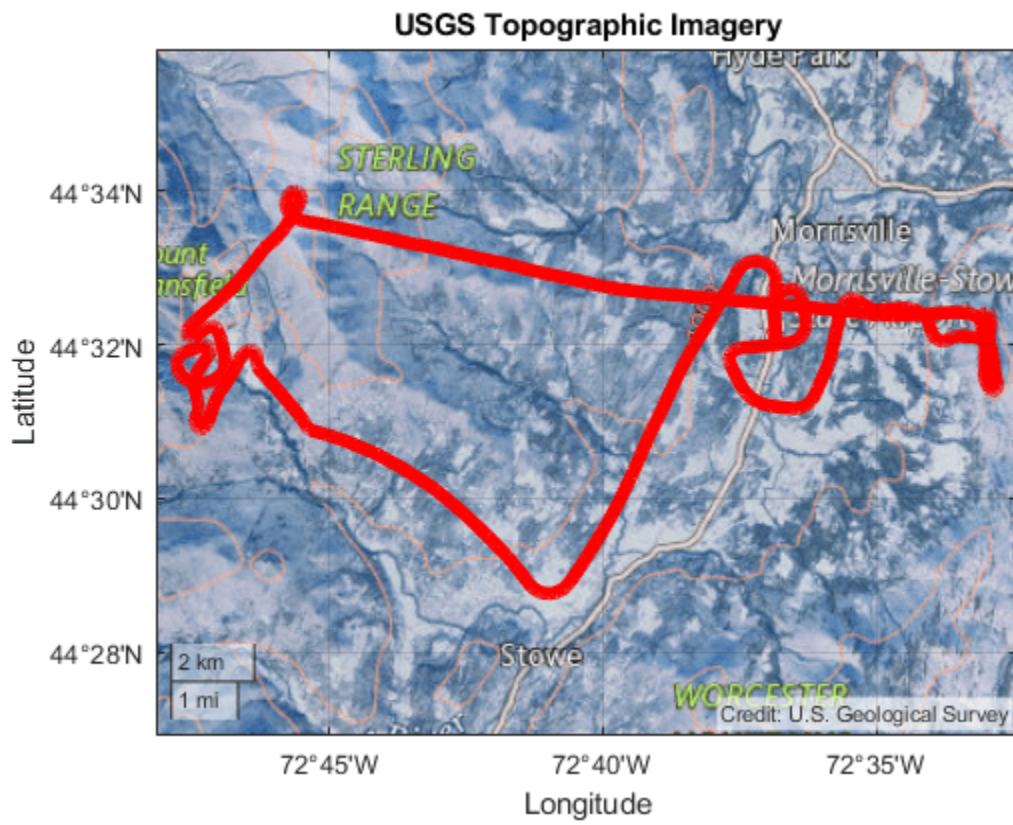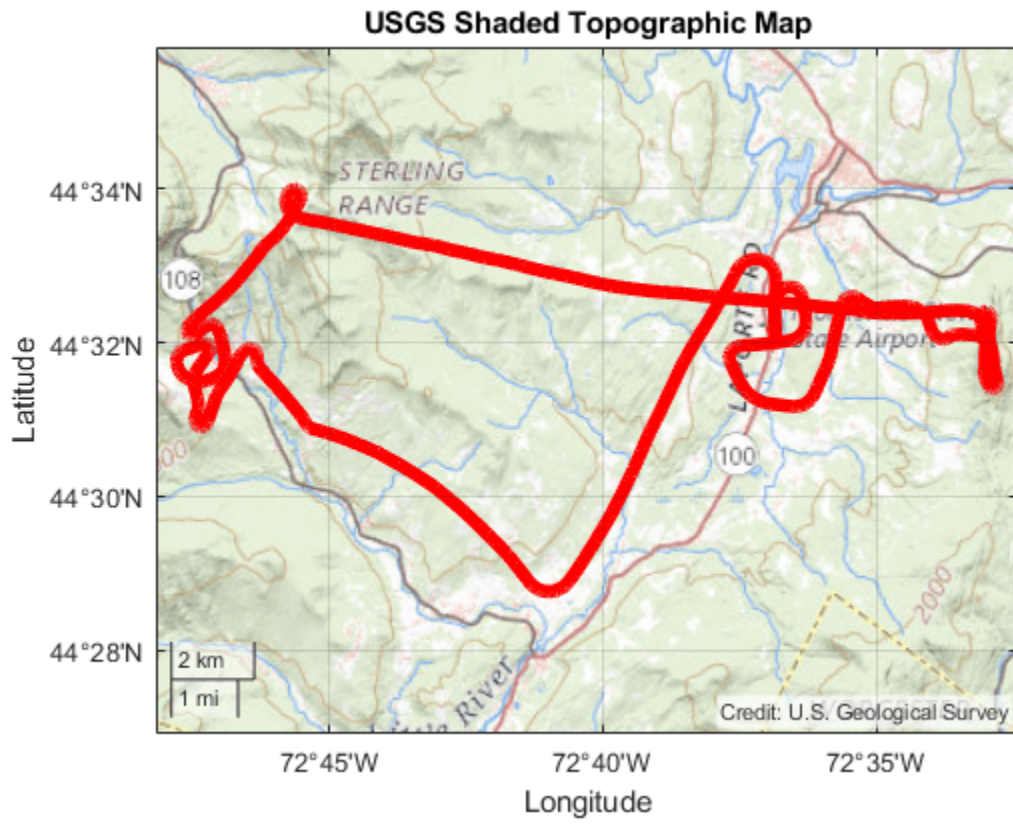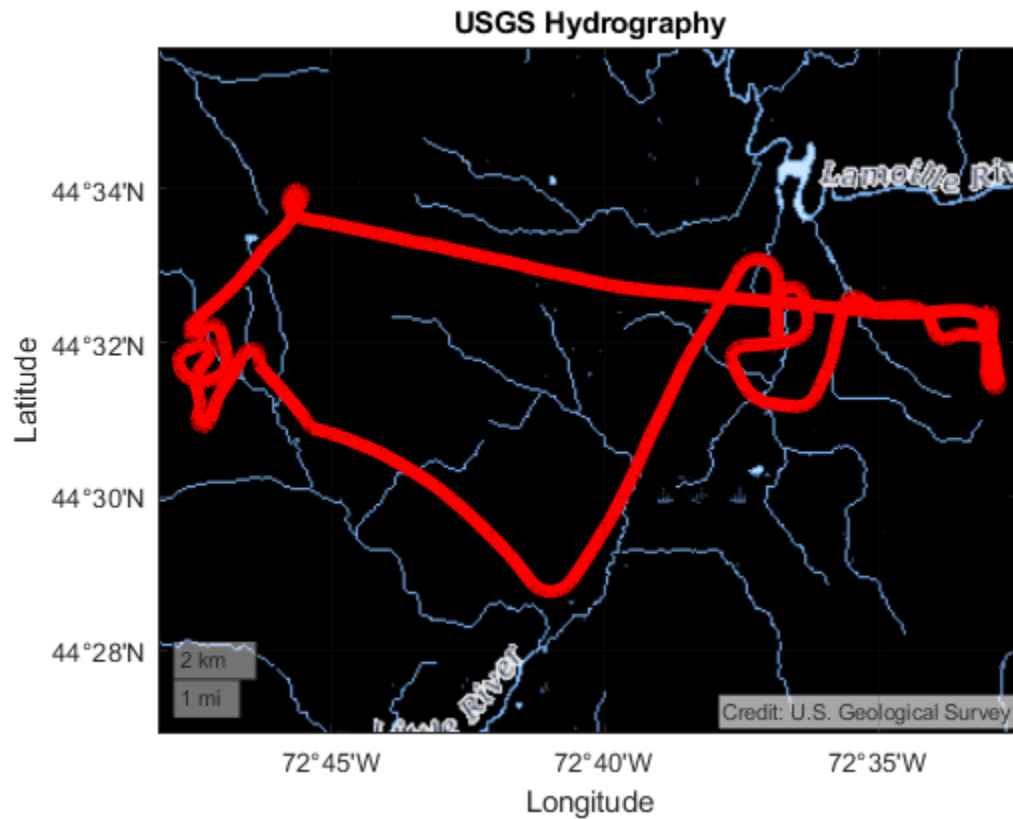
Input Arguments

basemapName — Name used to identify basemap programmatically
string scalar | character vector

Name used to identify basemap programmatically, specified as a string scalar or character vector.

Example: `'openstreetmap'`

Data Types: `string` | `char`

URL — Parameterized map URL
string scalar | character vector

Parameterized map URL, specified as a string scalar or character vector. A parameterized URL is an index of the map tiles, formatted as `${z}/${x}/${y}.png` or `{z}/{x}/{y}.png`, where:

- `${z}` or `{z}` is the tile zoom level.
- `${x}` or `{x}` is the tile column index.
- `${y}` or `{y}` is the tile row index.

Example: `'https://hostname/${z}/${x}/${y}.png'`

Data Types: `string` | `char`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `addCustomBasemap(basemapName,URL,'Attribution',attribution)`

Attribution — Attribution of custom basemap
`'Tiles courtesy of *DOMAIN_NAME_OF_URL*'` (default) | string scalar | string array | character vector | cell array of character vectors

Attribution of custom basemap, specified as the comma-separated pair consisting of `'Attribution'` and a string scalar, string array, character vector, or cell array of character vectors. If the host is `'localhost'`, or if URL contains only IP numbers, specify an empty value (`''`). To create a multiline attribution, specify a string array or nonscalar cell array of character vectors.

If you do not specify an attribution, the default attribution is `'Tiles courtesy of *DOMAIN_NAME_OF_URL*'`, where the `addCustomBasemap` function obtains the domain name from the URL input argument.

Example: `'Credit: U.S. Geological Survey'`

Data Types: `string` | `char` | `cell`

DisplayName — Display name of custom basemap
string scalar | character vector

Display name of the custom basemap, specified as the comma-separated pair consisting of `'DisplayName'` and a string scalar or character vector.

The `webmap` function uses this name in the Layer Manager. The `addToolbarMapButton` function uses this name in the basemap picker.

Example: `'OpenStreetMap'`

Data Types: `string` | `char`

MaxZoomLevel — Maximum zoom level of basemap
18 (default) | integer in the range [0, 25]

Maximum zoom level of the basemap, specified as the comma-separated pair consisting of `'MaxZoomLevel'` and an integer in the range [0, 25].

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

IsDeployable — Map is deployable using MATLAB Compiler™
`false` (default) | `true`

Map is deployable using MATLAB Compiler, specified as the comma-separated pair consisting of `'IsDeployable'` and `false` or `true`.

If you are deploying a map application and want users to have access to the added basemap, set `'IsDeployable'` to `true`. For more information about deployment, see "Tips" on page 1-15.

Data Types: `logical`

Tips

- You can find tiled web maps from various vendors, such as OpenStreetMap®, the USGS National Map, Mapbox, DigitalGlobe, Esri® ArcGIS Online, the Geospatial Information Authority of Japan (GSI), and HERE Technologies. Abide by the map vendors terms-of-service agreement and include accurate attribution with the maps you use.

- To use a custom basemap in a deployed application, when you call `addCustomBasemap`, you must set the value of the `'IsDeployable'` name-value pair to `true`. You must set this name-value pair whether you call `addCustomBasemap` in your application or outside of your application.

- To access a list of available basemaps, press **Tab** before specifying the basemap in your plotting function.
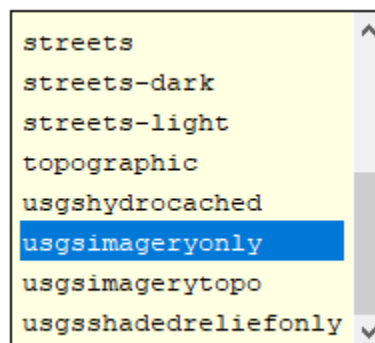
See Also

geoaxes | geobasemap | geobubble | geoglobe | webmap | addToolbarMapButton | removeCustomBasemap

Topics
"Specify a Custom Base Layer"

Introduced in R2018b

addCustomTerrain

Add custom terrain data

Syntax

```
addCustomTerrain(terrainName,files)
addCustomTerrain( ___ ,Name,Value)
```

Description

addCustomTerrain(terrainName,files) adds terrain data specified by files for use with
geographic plotting functions such as geoglobe. The terrain is named terrainName, and you can
specify it by this name when calling a plotting function. Custom terrain data is available for current
and future sessions of MATLAB, until you call removeCustomTerrain.

addCustomTerrain(___ ,Name,Value) adds custom terrain data with additional options specified
by one or more name-value pairs.

Examples

Display Custom Terrain Using Geographic Globe

Display a line from the surface of Gross Reservoir to a point above South Boulder Peak using custom
terrain.

First, add terrain for an area around South Boulder Peak by calling addCustomTerrain and
specifying a DTED file. The DTED file is from the SRTM Void Filled data set, available from the US
Geological Survey. Name the terrain 'southboulderpeak'.

```
addCustomTerrain('southboulderpeak','n39_w106_3arc_v2.dt1')
```

Create a geographic globe. Specify the terrain by name, using the 'Terrain' argument of the
geoglobe function. Preserve the terrain after plotting by calling the hold function. Then, plot the
line. Tilt the view by holding **Ctrl** and dragging.

```
uif = uifigure;
g = geoglobe(uif,'Terrain','southboulderpeak');
hold(g,'on')

lat = [39.95384 39.95];
lon = [-105.29916 -105.3608];
hTerrain = [10 0];
geoplot3(g,lat,lon,hTerrain,'y','HeightReference','Terrain', ...
    'LineWidth',3)
```

Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Close the geographic globe and remove the custom terrain.

```
close(uif)
removeCustomTerrain('southboulderpeak')
```

Input Arguments

`terrainName` — User-defined identifier for terrain data
string scalar | character vector

User-defined identifier for terrain data, specified as a string scalar or a character vector.

Data Types: `char` | `string`

`files` — List of DTED files
string scalar | character vector | cell array of character vectors

List of DTED files, specified as a string scalar, a character vector or a cell array of character vectors.

Note If you specify multiple files, they must combine to define a complete rectangular geographic region. If not, you must set the name-value pair `'FillMissing'` to `'true'`.

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'FillMissing',true`

Attribution — Attribution of custom terrain data
character vector | string scalar

Attribution of custom terrain data, specified as a character vector or a string scalar. Attributions display on geographic plots that use the custom terrain. By default, the attribution is empty.

Data Types: `char` | `string`

FillMissing — Fill data of missing files with value 0
`false` (default) | `true`

Fill data of missing files with value 0, specified as `true` or `false`. Missing file values are required to complete a rectangular geographic region with the input `files`.

Data Types: `logical`

WriteLocation — Name of folder to write extracted terrain files to
character vector | string scalar

Name of folder to write extracted terrain files to, specified as a character vector or a string scalar. The folder must exist and have write permissions. By default, `addCustomTerrain` writes extracted terrain files to a temporary folder that it generates using the `tempname` function.

Data Types: `char` | `string`

Tips

- To deploy an application with custom terrain using MATLAB Compiler, call `addCustomTerrain` in the application and include the DTED files in the deployed application package.
- You can find and download DTED files by using EarthExplorer, a data portal provided by the US Geological Survey (USGS). From the list of data sets, search for DTED files by selecting **Digital Elevation**, **SRTM**, and then **SRTM 1 Arc-Second Global** and **SRTM Void Filled**.

See Also

`removeCustomTerrain` | `geoglobe` | `addCustomBasemap`

Topics
"Access Basemaps and Terrain for Geographic Globe"

Introduced in R2020a

addToolbarMapButton

Add map button to toolbar

Syntax

```
addToolbarMapButton(tb)
addToolbarMapButton(tb,"basemap")
addToolbarMapButton(tb,"basemap",Name=Value)
```

Description

addToolbarMapButton(tb) adds all map buttons to the axes toolbar tb. The axes toolbar must be in a GeographicAxes object and the GeographicAxes object must be in a figure created by using the uifigure function.

addToolbarMapButton(tb,"basemap") creates a basemap picker and adds it to the axes toolbar. Open the basemap picker by selecting ⬚ **Pick Basemap** from the axes toolbar. The basemap picker appears near the upper-right corner of the map.

addToolbarMapButton(tb,"basemap",Name=Value) specifies basemap options using name-value arguments. For example, specify basemaps to include in the basemap picker by using the BasemapNames name-value argument.

Examples

Add and Remove Map Buttons from Toolbar

Create a figure by using the uifigure function. Place a geographic axes in the figure.

```
uif = uifigure;
gx = geoaxes(uif);
```

Create a custom axes toolbar. Add all map buttons to the toolbar.

```
tb = axtoolbar(gx,"default");
addToolbarMapButton(tb)
```

Remove all map buttons from the toolbar.

```
removeToolbarMapButton(tb)
```
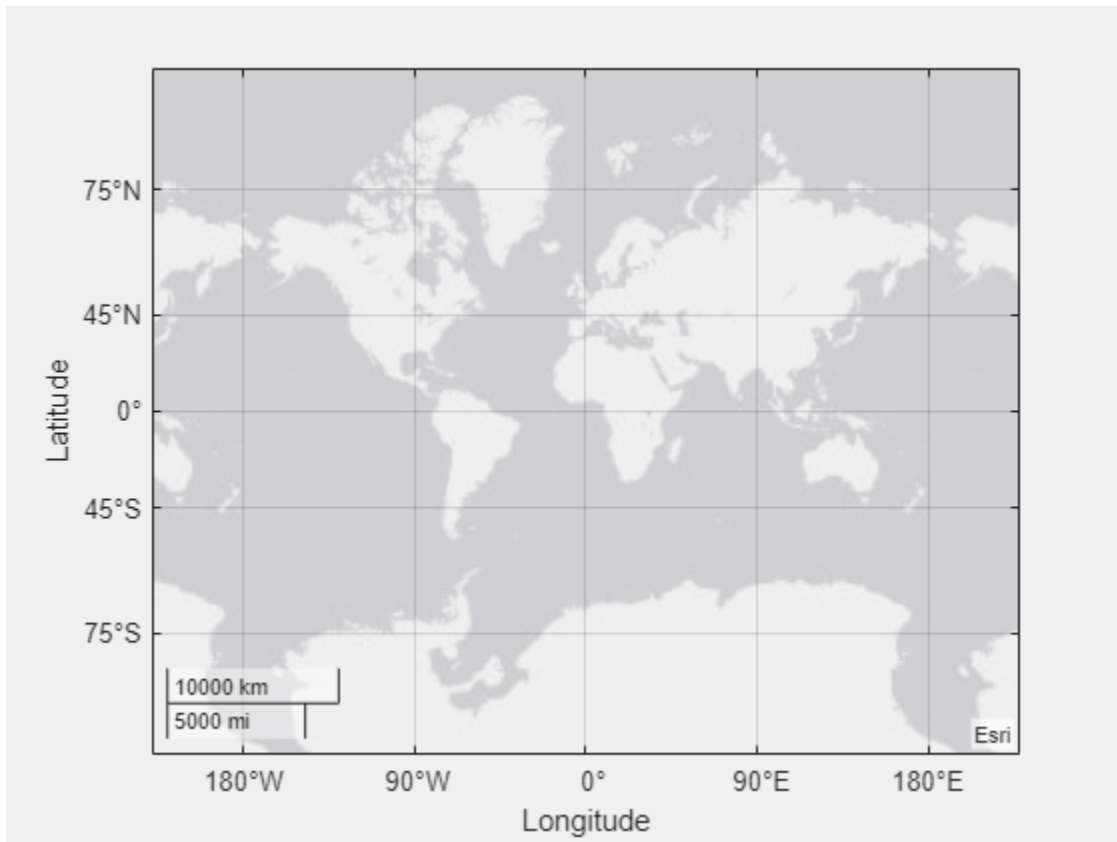
Add Basemap Picker to Geographic Plot

Read latitude and longitude data from a shapefile into a table by using the `readgeotable` function.

```
T = readgeotable("tsunamis.shp",CoordinateSystemType="geographic");
lat = T.Shape.Latitude;
lon = T.Shape.Longitude;
```

Display the latitude and longitude data on a map. To add a basemap picker to the toolbar, you must create a figure by using the `uifigure` function and a geographic axes within the figure by using the `geoaxes` function. By default, geographic plots use the `"streets-light"` basemap.

```
uif = uifigure;
gx = geoaxes(uif);
geoscatter(gx,lat,lon,"r+")
geolimits(gx,[-25.8016 27.4532],[96.9154 166.8728])
```

Preserve the specified toolbar, limits, and plotted data by using the `hold` function.

```
hold(gx,"on")
```

Create a custom toolbar by using the `axtoolbar` function. Add the basemap picker to the toolbar.

```
tb = axtoolbar(gx,"default");
addToolbarMapButton(tb,"basemap")
```
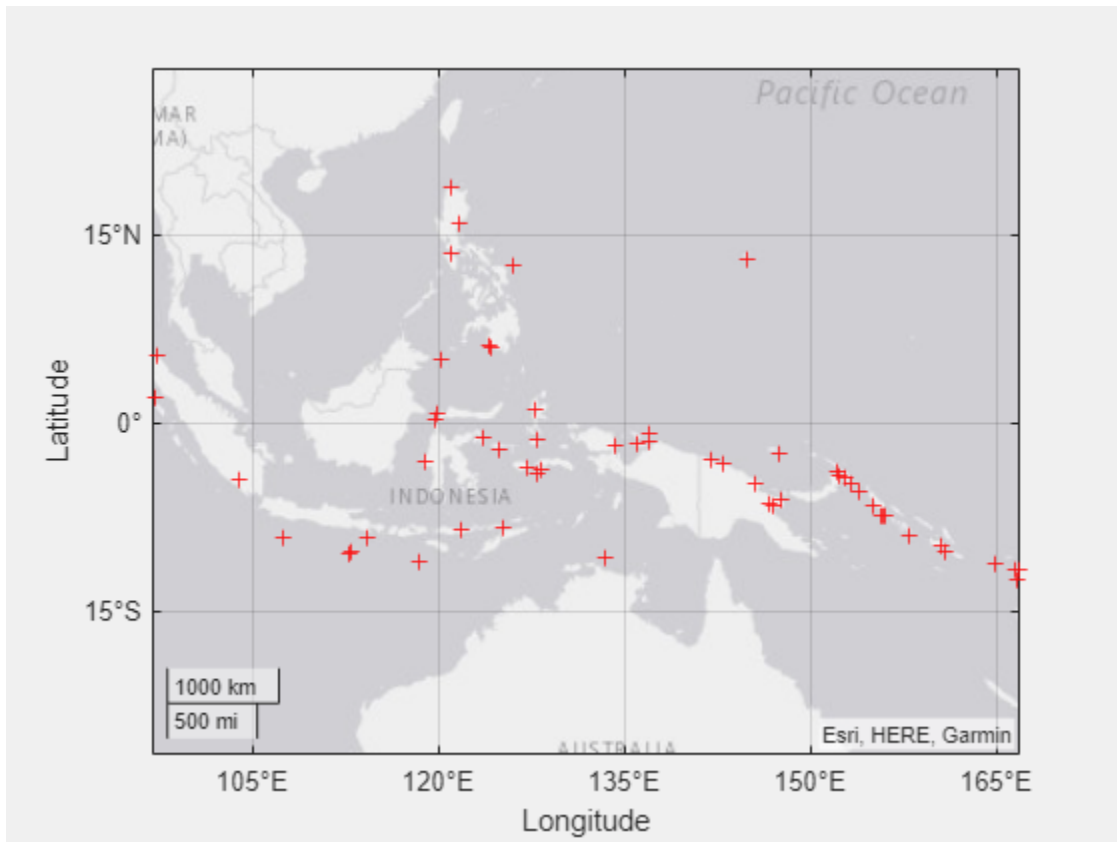
Open the toolbar and select a basemap.
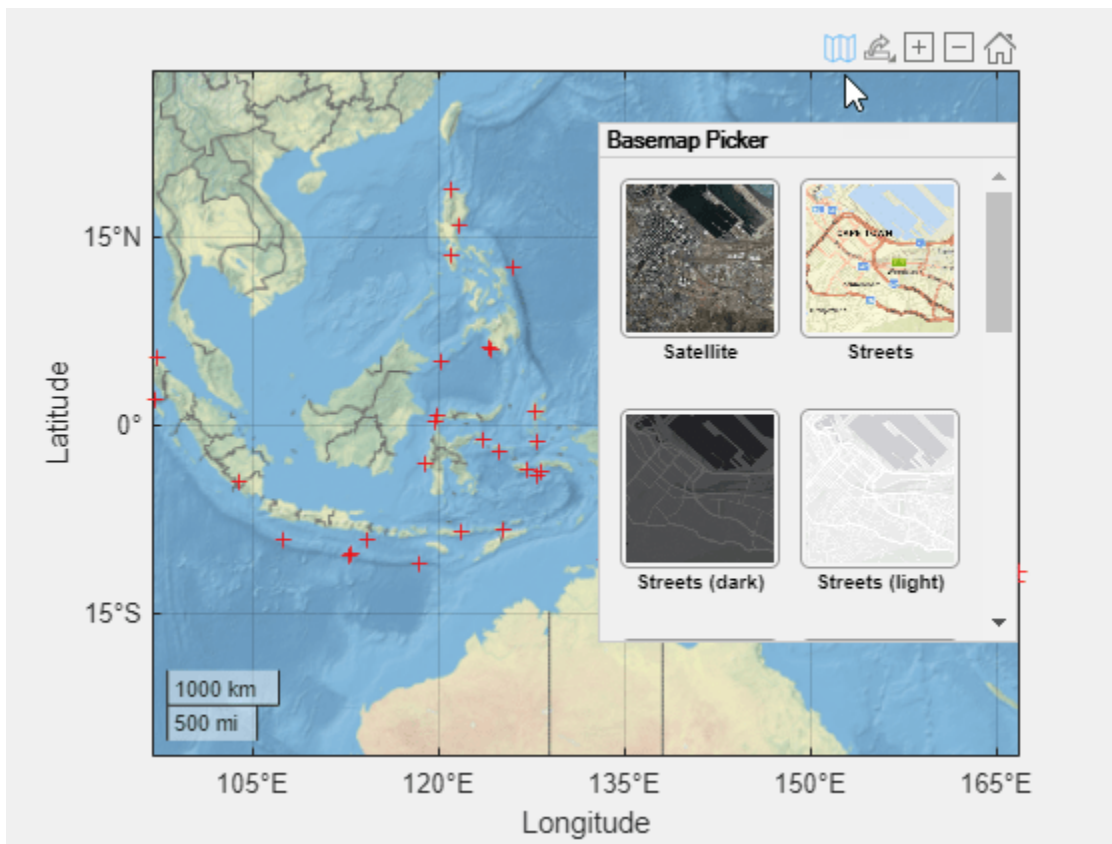
Add Custom Basemap to Basemap Picker

Add the OpenStreetMap© basemap to the list of basemaps available for use with mapping functions. After you add the basemap, you do not need to add it again in future sessions. The basemap picker uses the display name you specify when you add the basemap.

```
name = "openstreetmap";
url = "https://a.tile.openstreetmap.org/${z}/${x}/${y}.png";
copyright = char(uint8(169));
displayName = "OpenStreetMap";
attribution = copyright + "OpenStreetMap contributors";
addCustomBasemap(name,url,Attribution=attribution,DisplayName=displayName)
```

Read latitude and longitude data from a GPX file into a table.

```
T = readgeotable("sample_route.gpx");
lat = T.Shape.Latitude;
lon = T.Shape.Longitude;
```

Display the latitude and longitude data on a map.

```
uif = uifigure;
gx = geoaxes(uif);
geoplot(gx,lat,lon,LineWidth=2)
hold(gx,"on")
```

Add a basemap picker to the axes toolbar. Include the OpenStreetMap basemap and the high-zoom-level basemaps provided by Esri®.

```
tb = axtoolbar(gx,"default");
bmaps = ["openstreetmap","streets","streets-light","streets-dark","topographic","satellite"];
addToolbarMapButton(tb,"basemap",BasemapNames=bmaps)
```
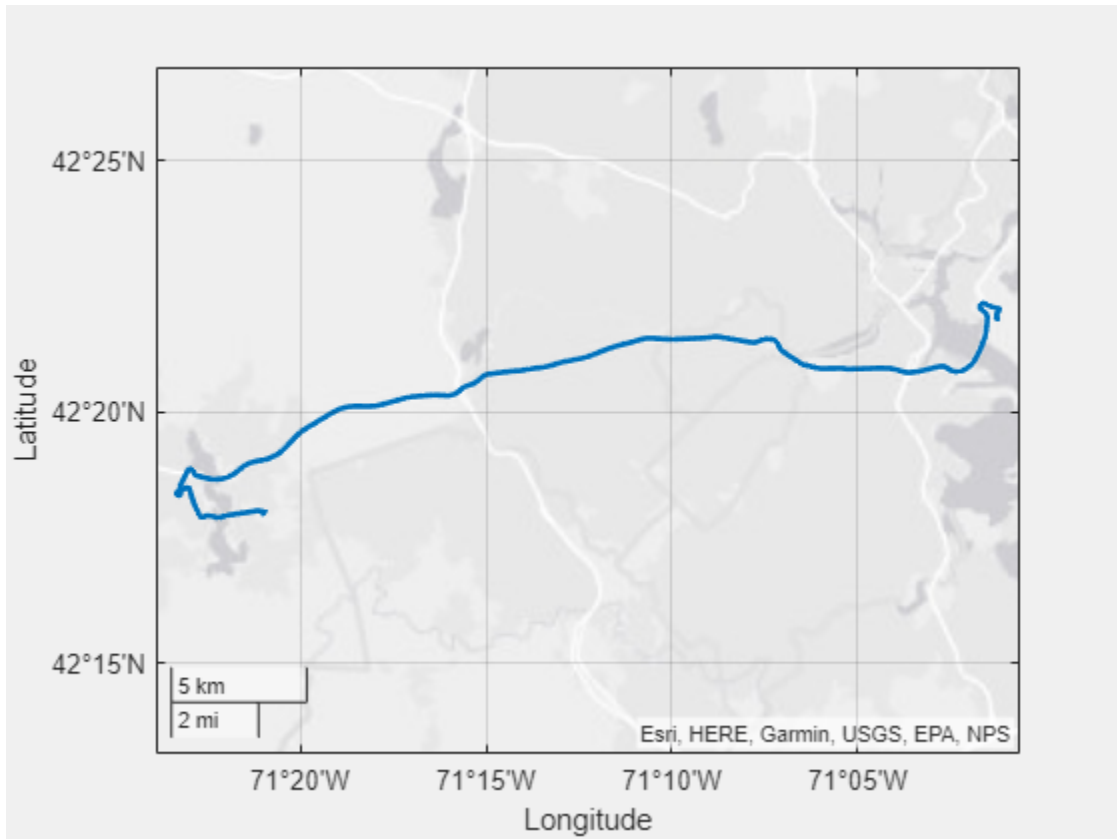

Open the basemap picker and select the OpenStreetMap basemap.

Add Custom Icons to Basemap Picker

By default, the basemap picker uses the Level 1, North America tile as the icon for custom basemaps. Create a basemap picker with custom icons.

Add the OpenStreetMap basemap to the list of basemaps available for use with mapping functions. After you add the basemap, you do not need to add it again in future sessions.

```
name = "openstreetmap";
url = "https://a.tile.openstreetmap.org/${z}/${x}/${y}.png";
copyright = char(uint8(169));
displayName = "OpenStreetMap";
attribution = copyright + "OpenStreetMap contributors";
addCustomBasemap(name,url,Attribution=attribution,DisplayName=displayName)
```

Read latitude and longitude data from a GPX file into a table.

```
T = readgeotable("sample_route.gpx");
lat = T.Shape.Latitude;
lon = T.Shape.Longitude;
```

Display the latitude and longitude data on a map.

```
uif = uifigure;
gx = geoaxes(uif);
```

```
geoplot(gx,lat,lon,LineWidth=2)
hold(gx,'on')
```

Specify the locations of the icons. For the custom basemap, use an icon that shows an area around Cape Town, South Africa. For the streets basemap, use the default icon.

```
osm_filename = "openstreetmap_256.png";
streets_filename = fullfile(matlabroot,"toolbox","shared","basemaps", ...
    "resources","icons","streets.png");
```

Create a basemap picker that includes the custom basemap, the "streets" basemap, and the custom icon.

```
tb = axtoolbar(gx,"default");
```
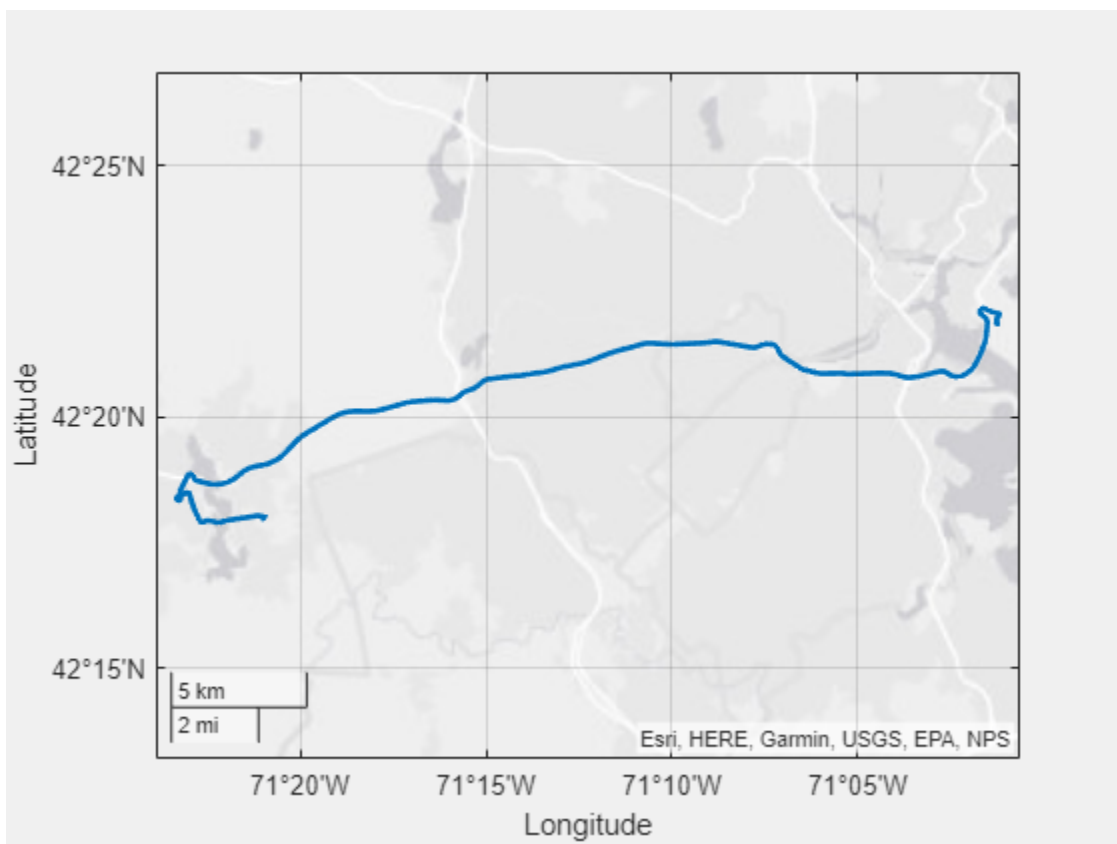


```
addToolbarMapButton(tb,"basemap",BasemapNames=["openstreetmap","streets"], ...
    BasemapIcons=[osm_filename,streets_filename])
```

Open the basemap picker and view the icons.
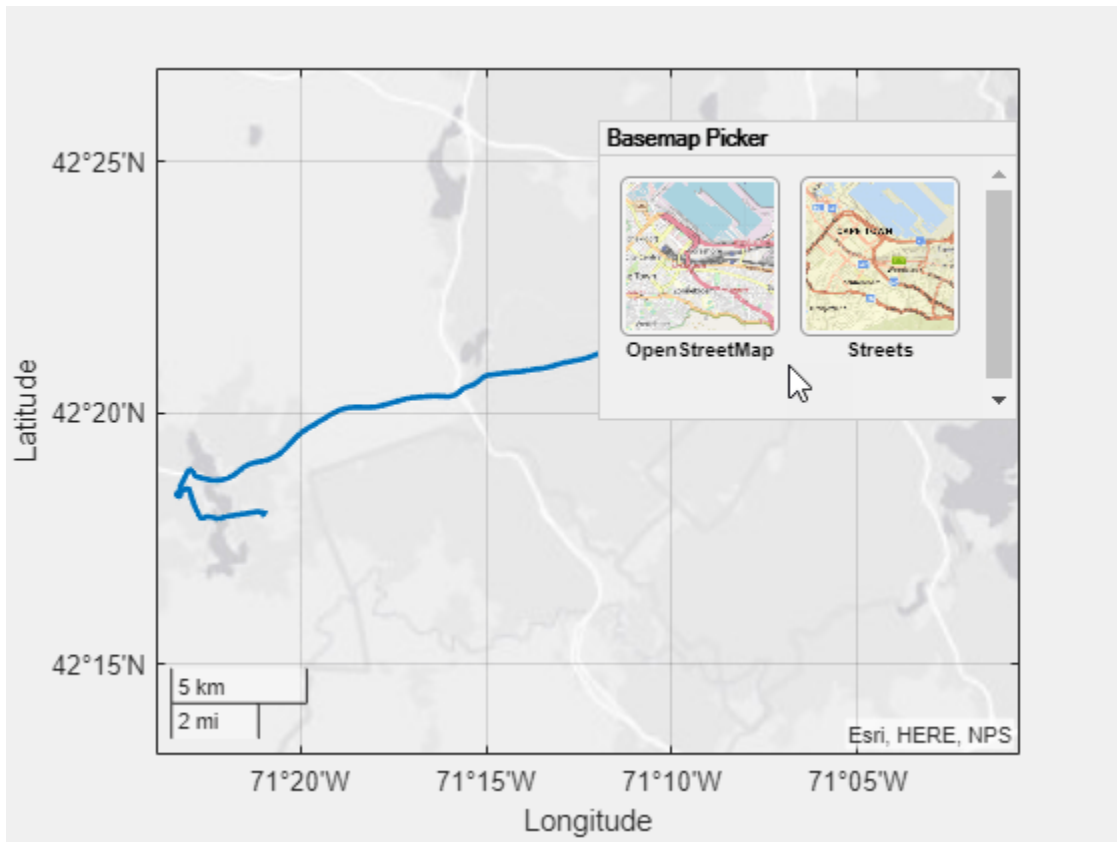
Input Arguments

tb — Axes toolbar
AxesToolbar object | array of AxesToolbar objects

Axes toolbar, specified as an AxesToolbar object or an array of AxesToolbar objects. The axes toolbar must be in a GeographicAxes object. Create the toolbar by using the axtoolbar function.

You can add only one of each type of map button to the axes toolbar. Adding a new button replaces any existing icons of the same type.

Name-Value Pair Arguments

Specify optional pairs of arguments as Name1=Value1,...,NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.
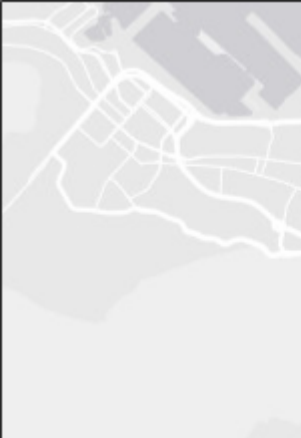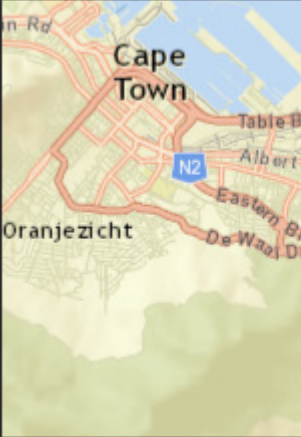
Example: addToolbarMapButton(tb,"basemap",BasemapNames=["streets","satellite"],NumColumns=1) specifies a basemap picker with one column and the "streets" and "satellite" basemaps.

BasemapNames — Basemaps to include in basemap picker
string array | cell array of character vectors

Basemaps to include in the basemap picker, specified as a string array or a cell array of character vectors. You can specify the basemaps shown in the table and basemaps added using the `addCustomBasemap` function. Five basemaps in the table are high-zoom-level maps hosted by Esri. The other six basemaps are tiled data sets created using Natural Earth. By default, the basemap picker includes all available basemaps, including custom basemaps, except `"none"`.

	`"streets-light"` (default) Map designed to provide geographic context while highlighting user data on a light background. Hosted by Esri.		`"streets-dark"` Map designed to provide geographic context while highlighting user data on a dark background. Hosted by Esri.
	`"streets"` General-purpose road map that emphasizes accurate, legible styling of roads and transit networks. Hosted by Esri.		`"satellite"` Full global basemap composed of high-resolution satellite imagery. Hosted by Esri.
	`"topographic"` General-purpose map with styling to depict topographic features. Hosted by Esri.		`"landcover"` Map that combines satellite-derived land cover data, shaded relief, and ocean-bottom relief. The light, natural palette is suitable for thematic and reference maps. Created using Natural Earth.

"colorterrain" Shaded relief map blended with a land cover palette. Humid lowlands are green and arid lowlands are brown. Created using Natural Earth.	**"grayterrain"** Terrain map in shades of gray. Shaded relief emphasizes both high mountains and micro-terrain found in lowlands. Created using Natural Earth.
"bluegreen" Two-tone, land-ocean map with light green land areas and light blue water areas. Created using Natural Earth.	**"grayland"** Two-tone, land-ocean map with gray land areas and white water areas. Created using Natural Earth.
"darkwater" Two-tone, land-ocean map with light gray land areas and dark gray water areas. This basemap is installed with MATLAB. Created using Natural Earth.	N/A **"none"** Blank background that plots your data with a latitude-longitude grid, ticks, and labels.

All basemaps except `"darkwater"` require Internet access. The `"darkwater"` basemap is included with MATLAB.

If you do not have consistent access to the Internet, you can download the basemaps created using Natural Earth onto your local system by using the Add-On Explorer. The five high-zoom-level maps are not available for download. For more information about downloading basemaps and changing the default basemap on your local system, see "Access Basemaps for Geographic Axes and Charts".

The basemaps hosted by Esri update periodically. As a result, you might see differences in your visualizations over time.

Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks®.

If you specify any combination of `BasemapNames`, `DisplayNames`, and `BasemapIcons`, they must all be the same size. The elements of `BasemapNames` correspond with the elements of `DisplayNames` and `BasemapIcons`.

Example: BasemapNames=["streets","satellite"]

Data Types: char | string

DisplayNames — Basemap labels
string array | cell array of character vectors

Basemap labels, specified as a string array or a cell array of character vectors. When a label is longer than the width of a column, MATLAB breaks the label into separate lines so that each line fits within the width of the column, and avoids breaking words when possible. Basemap labels display up to three lines. MATLAB truncates labels that are longer than three lines.

The default display name for each basemap is:

- "satellite" — **Satellite**
- "streets" — **Streets**
- "streets-dark" — **Streets (dark)**
- "streets-light" — **Streets (light)**
- "topographic" — **Topographic**
- "darkwater" — **Dark Water**
- "bluegreen" — **Blue Green**
- "grayterrain" — **Gray Terrain**
- "colorterrain" — **Color Terrain**
- "landcover" — **Land Cover**
- "none" — **none**
- Custom basemap — Display name specified by the `addCustomBasemap` function. If you do not specify a display name when you add the custom basemap, then the basemap picker uses the basemap name.

Create a basemap picker with no display names by specifying `DisplayNames` as `""`.

If you specify any combination of `BasemapNames`, `DisplayNames`, and `BasemapIcons`, they must all be the same size. The elements of `DisplayNames` correspond with the elements of `BasemapNames` and `BasemapIcons`.

Example: DisplayNames=["Road Map","Satellite Imagery"]

Data Types: char | string

BasemapIcons — Icon file names
string array | cell array of character vectors

Icon file names, specified as a string array or a cell array of character vectors. Supported image formats include JPEG, PNG, GIF, and SVG.

Specify file names based on the locations of your files.

- If a file is in your current folder or in a folder on the MATLAB path, then specify the name of the file only, such as `"myIcon.png"`.
- If a file is not in the current folder or in a folder on the MATLAB path, then specify the full or relative path name, such as `"C:\myfolder\myFile.png"` or `"dataDir\myFile.png"`.

The default basemap icons are in the folder *matlabroot*/`toolbox/shared/basemaps/resources/icons`. When you add a custom basemap using the `addCustomBasemap` function, the default icon is the Level 1, North America tile.

When an icon has a side length greater than 80 pixels, MATLAB scales the icon to fit.

Create a basemap picker with no icon images by specifying `BasemapIcons` as `""`. In this case, the icons display the basemap names instead of images.

If you specify any combination of `BasemapNames`, `DisplayNames`, and `BasemapIcons`, they must all be the same size. The elements of `BasemapIcons` correspond with the elements of `BasemapNames` and `DisplayNames`.

Example: `BasemapIcons=["myIcon1.png","myIcon2.png"]`

Data Types: `char` | `string`

NumColumns — Number of basemap picker columns
2 (default) | positive integer

Number of basemap picker columns, specified as a positive integer.

Example: `NumColumns=3`

Data Types: `single` | `double`

Limitations

- When the `Scrollable` property of the figure is `true`, the basemap picker may not appear, or may appear in a different location than the upper-right corner.

See Also

Functions
`axtoolbar` | `geoaxes` | `geobasemap` | `geoplot` | `geoscatter` | `removeToolbarMapButton`

Topics
"Access Basemaps for Geographic Axes and Charts"

Introduced in R2021b

aer2ecef

Transform local spherical coordinates to geocentric Earth-centered Earth-fixed

Syntax

```
[X,Y,Z] = aer2ecef(az,elev,slantRange,lat0,lon0,h0,spheroid)
[ ___ ] = aer2ecef( ___ ,angleUnit)
```

Description

`[X,Y,Z] = aer2ecef(az,elev,slantRange,lat0,lon0,h0,spheroid)` transforms the local azimuth-elevation-range (AER) spherical coordinates specified by `az`, `elev`, and `slantRange` to the geocentric Earth-centered Earth-fixed (ECEF) Cartesian coordinates specified by X, Y, and Z. Specify the origin of the local AER system with the geodetic coordinates `lat0`, `lon0`, and `h0`. Each coordinate input argument must match the others in size or be scalar. Specify `spheroid` as the reference spheroid for the geodetic coordinates.

`[ ___ ] = aer2ecef( ___ ,angleUnit)` specifies the units for azimuth, elevation, latitude, and longitude. Specify `angleUnit` as `'degrees'` (the default) or `'radians'`.

Examples

Calculate ECEF Coordinates from AER Coordinates

Find the ECEF coordinates of a satellite, using the AER coordinates of the satellite relative to the geodetic coordinates of a satellite dish.

First, specify the reference spheroid as WGS84 with length units measured in kilometers. For more information about WGS84, see "Reference Spheroids". The units for the ellipsoidal height, slant range, and ECEF coordinates must match the units specified by the `LengthUnit` property of the reference spheroid.

```
wgs84 = wgs84Ellipsoid('kilometers');
```

Specify the geodetic coordinates of the local origin. In this example, the local origin is the satellite dish. Specify `h0` as ellipsoidal height in kilometers.

```
lat0 = 42.3221;
lon0 = -71.3576;
h0 = 0.0847;
```

Specify the AER coordinates of the point of interest. In this example, the point of interest is the satellite. Specify the slant range in kilometers.

```
az = 24.8012;
elev = 14.6185;
slantRange = 36271.6327;
```

Then, calculate the ECEF coordinates of the satellite. In this example, the results display in scientific notation.

```
[x,y,z] = aer2ecef(az,elev,slantRange,lat0,lon0,h0,wgs84)

x = 1.0766e+04

y = 1.4144e+04

z = 3.3992e+04
```

Reverse the transformation using the `ecef2aer` function. In this example, `slantRange` displays in scientific notation.

```
[az,elev,slantRange] = ecef2aer(x,y,z,lat0,lon0,h0,wgs84)

az = 24.8012

elev = 14.6185

slantRange = 3.6272e+04
```

Input Arguments

az — Azimuth angles
scalar | vector | matrix | N-D array

Azimuth angles of one or more points in the local AER system, specified as a scalar, vector, matrix, or N-D array. Azimuths are measured clockwise from north. Specify values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

elev — Elevation angles
scalar | vector | matrix | N-D array

Elevation angles of one or more points in the local AER system, specified as a scalar, vector, matrix, or N-D array. Specify elevations with respect to a plane that is perpendicular to the normal of the spheroid surface. If the local origin is on the surface of the spheroid (`h0 = 0`), then the plane is tangent to the spheroid.

Specify values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

slantRange — Distances from local origin
scalar | vector | matrix | N-D array

Distances from the local origin, specified as a scalar, vector, matrix, or N-D array. Specify each distance as along a straight, 3-D, Cartesian line. Specify values in units that match the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Data Types: `single` | `double`

lat0 — Geodetic latitude of local origin
scalar | vector | matrix | N-D array

Geodetic latitude of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

lon0 — Geodetic longitude of local origin
scalar | vector | matrix | N-D array

Geodetic longitude of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

h0 — Ellipsoidal height of local origin
scalar | vector | matrix | N-D array

Ellipsoidal height of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify values in units that match the `LengthUnit` property of the `spheroid` object. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

For more information about ellipsoidal height, see "Find Ellipsoidal Height from Orthometric Height".

Data Types: `single` | `double`

spheroid — Reference spheroid
`referenceEllipsoid` object | `oblateSpheroid` object | `referenceSphere` object

Reference spheroid, specified as a `referenceEllipsoid` object, `oblateSpheroid` object, or `referenceSphere` object. The term reference spheroid is used synonymously with reference ellipsoid. To create a reference spheroid, use the creation function for the object. To specify the reference ellipsoid for WGS84, use the `wgs84Ellipsoid` function.

For more information about reference spheroids, see "Reference Spheroids".

Example: `spheroid = referenceEllipsoid('GRS 80');`

angleUnit — Angle units
`'degrees'` (default) | `'radians'`

Angle units, specified as `'degrees'` (the default) or `'radians'`.

Output Arguments

X — ECEF *x*-coordinates
scalar | vector | matrix | N-D array

ECEF *x*-coordinates of one or more points in the geocentric ECEF system, returned as a scalar, vector, matrix, or N-D array. Units are specified by the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Y — ECEF *y*-coordinates
scalar | vector | matrix | N-D array

ECEF *y*-coordinates of one or more points in the geocentric ECEF system, returned as a scalar, vector, matrix, or N-D array. Units are specified by the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Z — ECEF *z*-coordinates
scalar | vector | matrix | N-D array

ECEF *z*-coordinates of one or more points in the geocentric ECEF system, returned as a scalar, vector, matrix, or N-D array. Units are specified by the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also
`aer2geodetic` | `ecef2aer` | `enu2ecef` | `ned2ecef`

Topics
"Choose a 3-D Coordinate System"

Introduced in R2012b

aer2enu

Transform local spherical coordinates to local east-north-up

Syntax

```
[xEast,yNorth,zUp] = aer2enu(az,elev,slantRange)
[ ___ ] = aer2enu( ___ ,angleUnit)
```

Description

`[xEast,yNorth,zUp] = aer2enu(az,elev,slantRange)` transforms the local azimuth-elevation-range (AER) spherical coordinates specified by `az`, `elev`, and `slantRange` to the local east-north-up (ENU) Cartesian coordinates specified by `xEast`, `yNorth`, and `zUp`. Both coordinate systems use the same local origin. Each input argument must match the others in size or be scalar.

`[ ___ ] = aer2enu( ___ ,angleUnit)` specifies the units for azimuth and elevation. Specify `angleUnit` as `'degrees'` (the default) or `'radians'`.

Examples

Calculate ENU Coordinates from AER Coordinates

A sensor captures the AER coordinates of a nearby ground vehicle. Find the ENU coordinates of the vehicle with respect to the sensor, using the AER coordinates of the vehicle with respect to the same sensor.

First, specify the AER coordinates of the vehicle. Specify the azimuth and elevation in degrees. For this example, specify the slant range in meters.

```
az = 34.1160;
elev = 4.1931;
slantRange = 15.1070;
```

Then, calculate the ENU coordinates of the vehicle. The units for the ENU coordinates match the units specified by the slant range. Thus, the ENU coordinates are specified in meters.

```
[xEast,yNorth,zUp] = aer2enu(az,elev,slantRange)
```

```
xEast = 8.4504
```

```
yNorth = 12.4737
```

```
zUp = 1.1046
```

Reverse the transformation using the `enu2aer` function.

```
[az,elev,slantRange] = enu2aer(xEast,yNorth,zUp)
```

```
az = 34.1160
```

```
elev = 4.1931
```

```
slantRange = 15.1070
```

Input Arguments

az — Azimuth angles
scalar | vector | matrix | N-D array

Azimuth angles of one or more points in the local AER system, specified as a scalar, vector, matrix, or N-D array. Azimuths are measured clockwise from north. Specify values in degrees. To use values in radians, specify the angleUnit argument as 'radians'.

Data Types: single | double

elev — Elevation angles
scalar | vector | matrix | N-D array

Elevation angles of one or more points in the local AER system, specified as a scalar, vector, matrix, or N-D array. Specify elevations with respect to the xEast-yNorth plane that contains the local origin. If the local origin is on the surface of the spheroid, then the xEast-yNorth plane is tangent to the spheroid.

Specify values in degrees. To use values in radians, specify the angleUnit argument as 'radians'.

Data Types: single | double

slantRange — Distances from local origin
scalar value | vector | matrix | N-D array

Distances from the local origin, specified as a scalar, vector, matrix, or N-D array. Specify each distance as along a straight, 3-D, Cartesian line.

Data Types: single | double

angleUnit — Angle units
'degrees' (default) | 'radians'

Angle units, specified as 'degrees' (the default) or 'radians'.

Output Arguments

xEast — ENU *x*-coordinates
scalar | vector | matrix | N-D array

ENU *x*-coordinates of one or more points in the local ENU system, returned as a scalar, vector, matrix, or N-D array. Values are returned in the same units as the slantRange argument.

yNorth — ENU *y*-coordinates
scalar | vector | matrix | N-D array

ENU *y*-coordinates of one or more points in the local ENU system, returned as a scalar, vector, matrix, or N-D array. Values are returned in the same units as the slantRange argument.

zUp — ENU *z*-coordinates
scalar | vector | matrix | N-D array

ENU *z*-coordinates of one or more points in the local ENU system, returned as a scalar, vector, matrix, or N-D array. Values are returned in the same units as the `slantRange` argument.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`aer2ned` | `enu2aer` | `aer2ecef` | `aer2geodetic`

Topics
"Choose a 3-D Coordinate System"

Introduced in R2012b

aer2geodetic

Transform local spherical coordinates to geodetic

Syntax

```
[lat,lon,h] = aer2geodetic(az,elev,slantRange,lat0,lon0,h0,spheroid)
[ ___ ] = aer2geodetic( ___ ,angleUnit)
```

Description

`[lat,lon,h] = aer2geodetic(az,elev,slantRange,lat0,lon0,h0,spheroid)` transforms the local azimuth-elevation-range (AER) spherical coordinates specified by `az`, `elev`, and `slantRange` to the geodetic coordinates specified by `lat`, `lon`, and `h`. Specify the origin of the local AER system with the geodetic coordinates `lat0`, `lon0`, and `h0`. Each coordinate input argument must match the others in size or be scalar. Specify `spheroid` as the reference spheroid for the geodetic coordinates.

`[ ___ ] = aer2geodetic( ___ ,angleUnit)` specifies the units for azimuth, elevation, latitude, and longitude. Specify `angleUnit` as `'degrees'` (the default) or `'radians'`.

Examples

Calculate Geodetic Coordinates from AER Coordinates

Find the geodetic coordinates of the Matterhorn, using the AER coordinates of the Matterhorn with respect to the geodetic coordinates of Zermatt, Switzerland.

First, specify the reference spheroid as WGS 84. For more information about WGS 84, see "Reference Spheroids". The units for the ellipsoidal height and slant range must match the units specified by the `LengthUnit` property of the reference spheroid. The default length unit for the reference spheroid created by `wgs84Ellipsoid` is `'meter'`.

```
wgs84 = wgs84Ellipsoid;
```

Specify the geodetic coordinates of the local origin. In this example, the local origin is Zermatt. Specify `h0` as ellipsoidal height in meters.

```
lat0 = 46.017;
lon0 = 7.750;
h0 = 1673;
```

Specify the AER coordinates of the point of interest. In this example, the point of interest is the Matterhorn. Specify the slant range in meters.

```
az = 238.08;
elev = 18.744;
slantRange = 8876.8;
```

Then, calculate the geodetic coordinates of the Matterhorn. The result h is the ellipsoidal height of the Matterhorn in meters. To view the results in standard notation, specify the display format as shortG.

```
format shortG
[lat,lon,h] = aer2geodetic(az,elev,slantRange,lat0,lon0,h0,wgs84)

lat =
      45.977


lon =
       7.658


h =
        4531

```

Reverse the transformation using the geodetic2aer function.

```
[az,elev,slantRange] = geodetic2aer(lat,lon,h,lat0,lon0,h0,wgs84)

az =
      238.08

elev =
      18.744

slantRange =
      8876.8
```

Input Arguments

az — Azimuth angles
scalar | vector | matrix | N-D array

Azimuth angles of one or more points in the local AER system, specified as a scalar, vector, matrix, or N-D array. Azimuths are measured clockwise from north. Specify values in degrees. To use values in radians, specify the angleUnit argument as 'radians'.

Data Types: single | double

elev — Elevation angles
scalar | vector | matrix | N-D array

Elevation angles of one or more points in the local AER system, specified as a scalar, vector, matrix, or N-D array. Specify elevations with respect to a plane that is perpendicular to the normal of the spheroid surface. If the local origin is on the surface of the spheroid (h0 = 0), then the plane is tangent to the spheroid.

Specify values in degrees. To use values in radians, specify the angleUnit argument as 'radians'.

Data Types: single | double

slantRange — Distances from local origin
scalar | vector | matrix | N-D array

Distances from the local origin, specified as a scalar, vector, matrix, or N-D array. Specify each distance as along a straight, 3-D, Cartesian line. Specify values in units that match the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Data Types: `single` | `double`

lat0 — Geodetic latitude of local origin
scalar | vector | matrix | N-D array

Geodetic latitude of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

lon0 — Geodetic longitude of local origin
scalar | vector | matrix | N-D array

Geodetic longitude of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

h0 — Ellipsoidal height of local origin
scalar | vector | matrix | N-D array

Ellipsoidal height of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify values in units that match the `LengthUnit` property of the `spheroid` object. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

For more information about ellipsoidal height, see "Find Ellipsoidal Height from Orthometric Height".

Data Types: `single` | `double`

spheroid — Reference spheroid
`referenceEllipsoid` object | `oblateSpheroid` object | `referenceSphere` object

Reference spheroid, specified as a `referenceEllipsoid` object, `oblateSpheroid` object, or `referenceSphere` object. The term reference spheroid is used synonymously with reference ellipsoid. To create a reference spheroid, use the creation function for the object. To specify the reference ellipsoid for WGS84, use the `wgs84Ellipsoid` function.

For more information about reference spheroids, see "Reference Spheroids".

Example: `spheroid = referenceEllipsoid('GRS 80');`

angleUnit — Angle units
`'degrees'` (default) | `'radians'`

Angle units, specified as `'degrees'` (the default) or `'radians'`.

Output Arguments

`lat` — Geodetic latitude
scalar | vector | matrix | N-D array

Geodetic latitude of one or more points, returned as a scalar, vector, matrix, or N-D array. Values are specified in degrees within the interval [-90 90]. To use values in radians, specify the `angleUnit` argument as `'radians'`.

`lon` — Geodetic longitude
scalar | vector | matrix | N-D array

Geodetic longitude of one or more points, returned as a scalar, vector, matrix, or N-D array. Values are specified in degrees within the interval [-180 180]. To use values in radians, specify the `angleUnit` argument as `'radians'`.

`h` — Ellipsoidal height
scalar | vector | matrix | N-D array

Ellipsoidal height of one or more points, returned as a scalar, vector, matrix, or N-D array. Values are specified in units that match the `LengthUnit` property of the `spheroid` object. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

For more information about ellipsoidal height, see "Find Ellipsoidal Height from Orthometric Height".

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`aer2ecef` | `enu2geodetic` | `geodetic2aer` | `ned2geodetic`

Topics
"Choose a 3-D Coordinate System"

Introduced in R2012b

aer2ned

Transform local spherical coordinates to local north-east-down

Syntax

```
[xNorth,yEast,zDown] = aer2ned(az,elev,slantRange)
[ ___ ] = aer2ned( ___ ,angleUnit)
```

Description

`[xNorth,yEast,zDown] = aer2ned(az,elev,slantRange)` transforms the local spherical azimuth-elevation-range (AER) coordinates specified by `az`, `elev`, and `slantRange` to the local north-east-down (NED) coordinates specified by `xNorth`, `yEast`, and `zDown`. Both coordinate systems use the same local origin. Each input argument must match the others in size or be scalar.

`[ ___ ] = aer2ned( ___ ,angleUnit)` specifies the units for azimuth and elevation. Specify `angleUnit` as `'degrees'` (the default) or `'radians'`.

Examples

Calculate NED Coordinates from AER Coordinates

A sensor in an aircraft measures the AER coordinates of a nearby landmark. Find the NED coordinates of the landmark with respect to the aircraft, using the AER coordinates of the landmark with respect to the same aircraft.

First, specify the AER coordinates of the landmark. Specify the azimuth and elevation in degrees. For this example, specify the slant range in kilometers.

```
az = 155.427;
elev = -23.161;
slantRange = 10.885;
```

Then, calculate the NED coordinates of the landmark. The units for the NED coordinates match the units specified by the slant range. Thus, the NED coordinates are specified in kilometers.

```
[xNorth,yEast,zDown] = aer2ned(az,elev,slantRange)
```

```
xNorth = -9.1013
```

```
yEast = 4.1617
```

```
zDown = 4.2812
```

Reverse the transformation using the `ned2aer` function.

```
[az,elev,slantRange] = ned2aer(xNorth,yEast,zDown)
```

```
az = 155.4270
```

```
elev = -23.1610
```

```
slantRange = 10.8850
```

Input Arguments

az — Azimuth angles
scalar | vector | matrix | N-D array

Azimuth angles of one or more points in the local AER system, specified as a scalar, vector, matrix, or N-D array. Azimuths are measured clockwise from north. Specify values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

elev — Elevation angles
scalar | vector | matrix | N-D array

Elevation angles of one or more points in the local AER system, specified as a scalar, vector, matrix, or N-D array. Specify elevations with respect to the `xNorth-yEast` plane that contains the local origin. If the local origin is on the surface of the spheroid, then the `xNorth-yEast` plane is tangent to the spheroid.

Specify values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

slantRange — Distances from local origin
scalar value | vector | matrix | N-D array

Distances from the local origin, specified as a scalar, vector, matrix, or N-D array. Specify each distance as along a straight, 3-D, Cartesian line.

Data Types: `single` | `double`

angleUnit — Angle units
`'degrees'` (default) | `'radians'`

Angle units, specified as `'degrees'` (the default) or `'radians'`.

Output Arguments

xNorth — NED *x*-coordinates
scalar | vector | matrix | N-D array

NED *x*-coordinates of one or more points in the local NED system, returned as a scalar, vector, matrix, or N-D array. Values are returned in the same units as the `slantRange` argument.

yEast — NED *y*-coordinates
scalar | vector | matrix | N-D array

NED *y*-coordinates of one or more points in the local NED system, returned as a scalar, vector, matrix, or N-D array. Values are returned in the same units as the `slantRange` argument.

zDown — NED *z*-coordinates
scalar | vector | matrix | N-D array

NED *z*-coordinates of one or more points in the local NED system, returned as a scalar, vector, matrix, or N-D array. Values are returned in the same units as the `slantRange` argument.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`aer2enu` | `ned2aer` | `aer2ecef` | `aer2geodetic`

Topics
"Choose a 3-D Coordinate System"

Introduced in R2012b

almanac

Parameters for Earth, planets, Sun, and Moon

Note almanac is not recommended. Use earthRadius, referenceEllipsoid, referenceSphere, or wgs84Ellipsoid instead.

Syntax

```
almanac
almanac(body)
data = almanac(body,parameter)
data = almanac(body,parameter,units)
data = almanac(parameter,units,referencebody)
```

Description

almanac displays the names of the celestial objects available in the almanac.

almanac(*body*) lists the options, or parameters, available for any of the following celestial bodies:

```
'earth'       'pluto'
'jupiter'     'saturn'
'mars'        'sun'
'mercury'     'uranus'
'moon'        'venus'
'neptune'
```

data = almanac(body,parameter) returns the value of the requested parameter for the celestial body specified by body.

Valid parameter values are 'radius' for the planetary radius, 'ellipsoid' or 'geoid' for the two-element ellipsoid vector, 'surfarea' for the surface area, and 'volume' for the planetary volume.

For the Earth, parameter can also be any of the following valid predefined ellipsoid values. In this case, the two-element ellipsoid vector for that ellipsoid model is returned. Valid ellipsoid definitions for the Earth are:

'everest'	1830 Everest ellipsoid
'bessel'	1841 Bessel ellipsoid
'airy'	1830 Airy ellipsoid
'clarke66'	1866 Clarke ellipsoid
'clarke80'	1880 Clarke ellipsoid
'international'	1924 International ellipsoid
'krasovsky'	1940 Krasovsky ellipsoid
'wgs60'	1960 World Geodetic System ellipsoid

`'iau65'`	1965 International Astronomical Union ellipsoid
`'wgs66'`	1966 World Geodetic System ellipsoid
`'iau68'`	1968 International Astronomical Union ellipsoid
`'wgs72'`	1972 World Geodetic System ellipsoid
`'grs80'`	1980 Geodetic Reference System ellipsoid
`'wgs84'`	1984 World Geodetic System ellipsoid

For the Earth, the parameter values `'ellipsoid'` and `'geoid'` are equivalent to `'grs80'`.

`data = almanac(body,parameter,units)` specifies the units to be used for the output measurement, where units is any valid distance `units` . Note that these are linear units, but the result for surface area is in square units, and for volume is in cubic units. The default units are `'kilometers'`.

`data = almanac(parameter,units,referencebody)` specifies the source of the information. This sets the assumptions about the shape of the celestial body used in the calculation of volumes and surface areas. A reference body of `'actual'` returns a tabulated value rather than one dependent upon a ellipsoid model assumption. Other possible `referencebody` s are `'sphere'` for a spherical assumption and `'ellipsoid'` for the default ellipsoid model. The default reference body is `'sphere'`.

For the Earth, any of the preceding predefined ellipsoid definition s can also be entered as a reference body.

For Mercury, Pluto, Venus, the Sun, and the Moon, the eccentricity of the ellipsoid model is zero, that is, the `'ellipsoid'` reference body is actually a sphere.

Tips

Take care when using angular arc length units for distance measurements. All planets have a radius of 1 radian, for example, and an area unit of *square degrees* indicates unit squares, 1 degree of arc length on a side, not 1-degree-by-1-degree quadrangles.

See Also
distance | earthRadius | referenceEllipsoid | referenceSphere | wgs84Ellipsoid

Introduced before R2006a

angl2str

Convert angles to character array

Syntax

```
str = angl2str(angle)
str = angl2str(angle,signcode)
str = angl2str(angle,signcode,units)
str = angl2str(angle,signcode,units,n)
```

Description

`str = angl2str(angle)` converts a numerical vector of angles in degrees to a character array. The purpose of this function is to make angular-valued variables into character vectors suitable for map display.

`str = angl2str(angle,signcode)` specifies the method for indicating that a given angle is positive or negative, where `signcode` is one of the following:

`'ew'`	east/west notation; trailing 'e' (positive longitudes) or 'w' (negative longitudes)
`'ns'`	north/south notation; trailing 'n' (positive latitudes) or 's' (negative latitudes)
`'pm'`	plus/minus notation; leading '+' (positive angles) or '-' (negative angles)
`'none'`	blank/minus notation; leading '-' for negative angles or sign omitted for positive angles (the default value)

`str = angl2str(angle,signcode,units)` specifies the units and the output format of the returned angle, using the following values:

Units	Units of Angle	Output Format
`'degrees'`	degrees	decimal degrees
`'degrees2dm'`	degrees	degrees + decimal minutes
`'degrees2dms'`	degrees	degrees + minutes + decimal seconds
`'radians'`	radians	decimal radians

`str = angl2str(angle,signcode,units,n)` uses the integer `n` to control the number of significant digits provided in the output. `n` is the power of 10 representing the last place of significance in the number of degrees, minutes, seconds, or radians -- for `units` of `'degrees'`, `'degrees2dm'`, `'degrees2dms'`, and `'radians'`, respectively. For example, if `n = -2` (the default), `angl2str` rounds to the nearest hundredth. If `n = -0`, `angl2str` rounds to the nearest integer. And if `n = 1`, `angl2str` rounds to the tens place, although positive values of `n` are of little practical use. Note that this sign convention for `n` is opposite to the one used by the MATLAB `round` function.

Examples

Convert Numeric Angles to String Matrix

Create a series of values for angles.

```
a = -3:1.5:3;
```

Convert the numeric values in DMS units, using the north-south format.

```
str = angl2str(a,'ns','degrees2dms',-3)

str = 5x25 char array
    ' 3^{\circ} 00' 00.000" S '
    ' 1^{\circ} 30' 00.000" S '
    ' 0^{\circ} 00' 00.000"   '
    ' 1^{\circ} 30' 00.000" N '
    ' 3^{\circ} 00' 00.000" N '
```

These LaTeX strings are displayed (using the `text` function) as follows:

```
x = [.1 .1 .1 .1 .1];
y = [.1 .2 .3 .4 .5];
text(x,y,str)
```
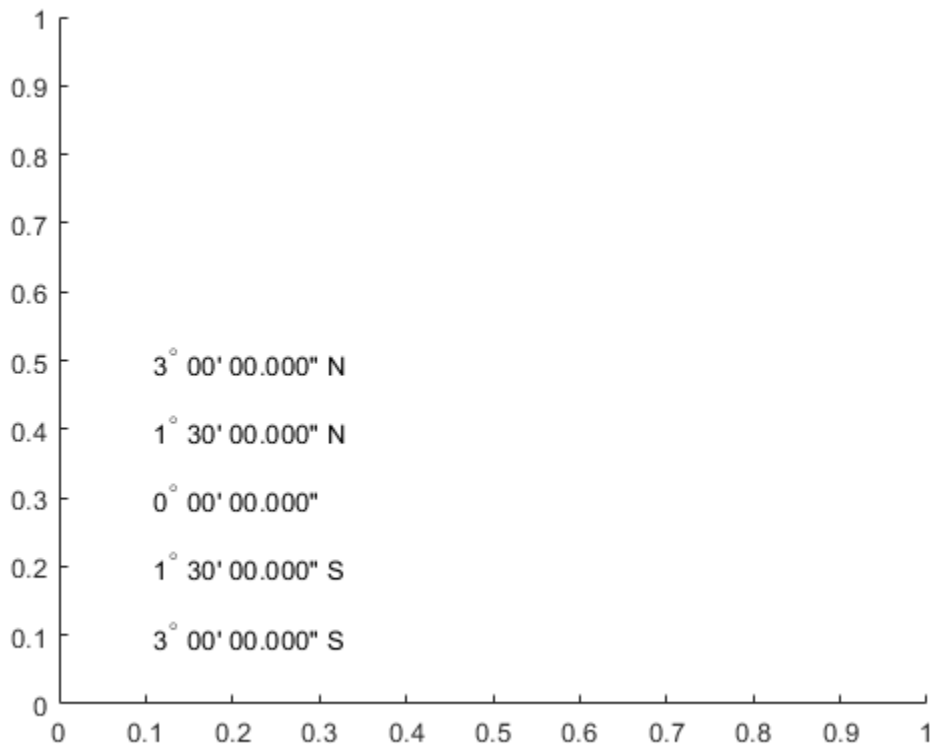

See Also
str2angle | dist2str

Introduced before R2006a

angledim

Convert angles units

Note

Note The `angledim` function has been replaced by four, more specific, functions: `fromRadians`, `fromDegrees`, `toRadians`, and `toDegrees`. However, `angledim` will be maintained for backward compatibility. The functions `deg2rad`, `rad2deg`, and `unitsratio` provide additional alternatives.

Syntax

angleOut = angledim(angleIn,*from*,*to*)

Description

angleOut = angledim(angleIn,*from*,*to*) returns the value of the input angle `angleIn`, which is in units specified by *from*, in the desired units given by *to*. Angle units are `'degrees'` for "decimal" degrees or `'radians'` for radians.

Examples

Convert from degrees to radians:

angledim(23.45134,'degrees','radians')

ans =
    0.4093

See Also
degrees2dms | deg2rad | fromDegrees | fromRadians | toDegrees | toRadians | rad2deg | unitsratio

Introduced before R2006a

antipode

Point on opposite side of globe

Syntax

```
[newlat,newlon] = antipode(lat,lon)
[newlat,newlon] = antipode(lat,lon,angleunits)
```

Description

`[newlat,newlon] = antipode(lat,lon)` returns the geographic coordinates of the points exactly opposite on the globe from the input points given by `lat` and `lon`. All angles are in degrees.

`[newlat,newlon] = antipode(lat,lon,angleunits)` where `angleunits` specifies the input and output units as either `'degrees'` or `'radians'`. It can be abbreviated and is case-insensitive.

Examples

Find Antipode of Given Point

Given a point (43ºN, 15ºE), find its antipode:

```
[newlat,newlong] = antipode(43,15)
newlat =
        -43
newlong =
        -165
```

or (43ºS, 165ºW).

Find Antipode of North and South Poles

Perhaps the most obvious antipodal points are the North and South Poles. The function `antipode` demonstrates this:

```
[newlat,newlong] = antipode(90,0,'degrees')
newlat =
        -90
newlong =
        180
```

Note that in this case longitudes are irrelevant because all meridians converge at the poles.

Find Antipode of MathWorks Headquarters

This example shows how to find the antipode of the location of the MathWorks corporate headquarters in Natick, Massachusetts. The example maps the headquarters location and its antipode in an orthographic projection.

Specify latitude and longitude as degree-minutes-seconds and then convert to decimal degrees.

```
mwlat = dms2degrees([ 42 18 2.5])
```

```
mwlat = 42.3007

mwlon = dms2degrees([-71 21 7.9])

mwlon = -71.3522
```

Find the antipode.

```
[amwlat amwlon] = antipode(mwlat,mwlon)

amwlat = -42.3007

amwlon = 108.6478
```

Prove that these points are antipodes. The `distance` function shows them to be 180 degrees apart.

```
dist = distance(mwlat,mwlon,amwlat,amwlon)

dist = 180.0000
```

Generate a map centered on the original point and then another map centered on the antipode.

```
figure
subplot(1,2,1)
axesm ('MapProjection','ortho','origin',[mwlat mwlon],...
       'frame','on','grid','on')
load coastlines
geoshow(coastlat,coastlon,'displaytype','polygon')
geoshow(mwlat,mwlon,'Marker','o','Color','red')
title(sprintf('Looking down at\n(%s,%s)', ...
    angl2str(mwlat,'ns'), angl2str(mwlon,'ew')))

subplot(1,2,2)
axesm ('MapProjection','ortho','origin',[amwlat amwlon],...
       'frame','on','grid','on')
geoshow(coastlat,coastlon,'displaytype','polygon')
geoshow(amwlat,amwlon,'Marker','o','Color','red')
title(sprintf('Looking down at\n(%s,%s)', ...
    angl2str(amwlat,'ns'), angl2str(amwlon,'ew')))
```

Looking down at
(42.30° N , 71.35° W)
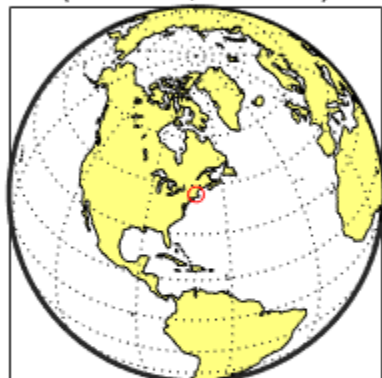
Looking down at
(42.30° S , 108.65° E)
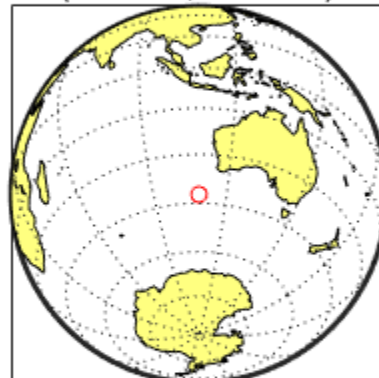
Introduced before R2006a

append

Append features to geographic or planar vector

Syntax

```
vout = append(vin,lat,lon)
vout = append(vin,x,y)
vout = append( ___ ,field,value)
```

Description

`vout = append(vin,lat,lon)` appends the latitude and longitude values in `lat` and `lon` to the `Latitude` and `Longitude` properties of the geographic vector `vin`. `vin` is either a `geopoint` or a `geoshape` object.

`vout = append(vin,x,y)` appends the planar *x*- and *y*-coordinates in `x` and `y` to the X and Y properties of the planar vector `vin`. `vin` is either a `mappoint` or a `mapshape` object.

`vout = append( ___ ,field,value)` appends the values specified in `value` to the corresponding dynamic property, `field`. If the property does not exist, `append` adds the dynamic property to the object using the value of `field` for the name and assigning the field the value specified in `value`. You can specify multiple field-value pairs. Enclose each field name in single quotes.

Examples

Append Points to Geopoint Vector

Create a geopoint vector.

```
p = geopoint(42,-110)

p =
 1x1 geopoint vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
 Feature properties:
     Latitude: 42
    Longitude: -110
```

Append the vector with three new geographic points.

```
lat1 = [42.1 44 44.1];
lon1 = [-101 -120 -121];
p = append(p,lat1,lon1)

p =
 4x1 geopoint vector with properties:
```

```
Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
Feature properties:
     Latitude: [42 42.1000 44 44.1000]
    Longitude: [-110 -101 -120 -121]
```

The length of the geopoint vector increases by three, as expected, and the `'Latitude'` and `'Longitude'` feature properties list the new coordinates.

Append Points to Mapshape Vector

Create a mapshape vector, designating a dynamic `'Temperature'` property. This vector has one feature with three vertices.

```
s = mapshape(42:44,30:32,'Temperature', {65:67})

s =
 1x1 mapshape vector with properties:

 Collection properties:
      Geometry: 'line'
      Metadata: [1x1 struct]
 Vertex properties:
             X: [42 43 44]
             Y: [30 31 32]
   Temperature: [65 66 67]
```

Append the vector with two new planar points. The points are stored as a new feature with two vertices.

```
x1 = [44.5 45];
y1 = [32.5 33];
s = append(s,x1,y1)

s =
 2x1 mapshape vector with properties:

 Collection properties:
      Geometry: 'line'
      Metadata: [1x1 struct]
 Vertex properties:
  (2 features concatenated with 1 delimiter)
             X: [42 43 44 NaN 44.5000 45]
             Y: [30 31 32 NaN 32.5000 33]
   Temperature: [65 66 67 NaN 0 0]
```

The features are separated by NaN. The `'Temperature'` value of the new points is set to 0 by default, since no value was specified during the call to append. The mapshape vector grew from 1x1 to 2x1 in length because the number of features increased.

Append Point with New Property to Mappoint Vector

Create a mappoint vector with a dynamic property `Temperature`.

```
mp = mappoint(42,-110,'Temperature',65)

mp =
 1x1 mappoint vector with properties:

 Collection properties:
      Geometry: 'point'
      Metadata: [1x1 struct]
 Feature properties:
             X: 42
             Y: -110
   Temperature: 65
```

Add a point to this vector. The point contains a new dynamic property, `Pressure`.

```
mp = append(mp,42.2,-110.5,'Temperature',65.6,'Pressure', 100.0)

mp =
 2x1 mappoint vector with properties:

 Collection properties:
      Geometry: 'point'
      Metadata: [1x1 struct]
 Feature properties:
             X: [42 42.2000]
             Y: [-110 -110.5000]
   Temperature: [65 65.6000]
      Pressure: [0 100]
```

A default `Pressure` value of 0 is automatically added to the first point.

Append Points with New Properties to Geoshape Vector

Create a geoshape vector, designating a dynamic `'Temperature'` property. The `'Temperature'` values are input as a cell array so that they belong to a single feature. This vector has one feature with three vertices.

```
lat1 = [42, 42.2, 43];
lon1 = [-110, -110.3, -110.5];
temp1 = {[65, 65.1, 65.2]};
s = geoshape(lat1,lon1,'Temperature',temp1)

s =
 1x1 geoshape vector with properties:

 Collection properties:
      Geometry: 'line'
      Metadata: [1x1 struct]
 Vertex properties:
      Latitude: [42 42.2000 43]
      Longitude: [-110 -110.3000 -110.5000]
```

```
        Temperature: [65 65.1000 65.2000]
```

Add three points to the geoshape vector, including a two new dynamic properties `'Precipitation'` and `'CloudCover'`. The latitude and longitude values are added as a two-element cell array, so two features are added to the geoshape vector. Note that the `'Temperature'` and `'Precipitation'` values are specified as two-element vectors, while the new `'CloudCover'` values are specified as a one-element cell array.

```
lat2 = {[50 50.2],60};
lon2 = {[-120 -121],-130};
temp2 = [60.2 60.4];
precip = [0.07 0.19];
cloud = {[20,80]};
s2 = append(s,lat2,lon2,'Temperature',temp2,'Precipitation',precip,'CloudCover',cloud)

s2 =
 3x1 geoshape vector with properties:

 Collection properties:
        Geometry: 'line'
        Metadata: [1x1 struct]
 Vertex properties:
  (3 features concatenated with 2 delimiters)
        Latitude: [42 42.2000 43 NaN 50 50.2000 NaN 60]
       Longitude: [-110 -110.3000 -110.5000 NaN -120 -121 NaN -130]
     Temperature: [65 65.1000 65.2000 NaN 60.2000 0 NaN 60.4000]
       CloudCover: [0 0 0 NaN 20 80 NaN 0]
 Feature properties:
    Precipitation: [0 0.0700 0.1900]
```

This appended vector `s2` now has three features, separated by `NaN`, with some Vertex properties and some Feature properties. The two cells of the latitude and longitude cell arrays form the two newly-added features, one with two vertices and the other with one vertex. Since `'Temperature'` had previously been designated as a Vertex property in `s`, the new `'Temperature'` values are added as Vertex properties. The `'Temperature'` value of one new point has not been assigned, so it is set to the default value of 0.

However, the new `'Precipitation'` and `'CloudCover'` properties are designated as a Vertex or Feature property, whichever is more appropriate for the value format. `'Precipitation'` values are provided as a two-element vector, so they are assigned as Feature properties, where each element of `precip2` belongs to a separate feature of the geoshape vector. Since `'CloudCover'` values are provided as a cell array, the values must belong to the same feature, so they are set as Vertex properties corresponding to the first added feature. No `'CloudCover'` values have been specified for the second added feature, so vertices in the second feature are assigned the default value 0. Finally, `'Precipitation'` and `'CloudCover'` values for the original feature are set to the default value of 0.

Input Arguments

`vin` — Input geographic or planar vector
`geopoint`, `geoshape`, `mappoint`, or `mapshape` object

Input geographic or planar vector, specified as a `geopoint`, `geoshape`, `mappoint`, or `mapshape` object.

lat — Latitude values
numeric scalar or vector

Latitude values, specified as a numeric scalar or vector.

Data Types: `single` | `double`

lon — Longitude values
numeric scalar or vector

Longitude values, specified as a numeric scalar or vector.

Data Types: `single` | `double`

x — Planar *x*-coordinates
numeric scalar or vector

Planar *x*-coordinates, specified as a numeric scalar or vector.

Data Types: `single` | `double`

y — Planar *y*-coordinates
numeric scalar or vector

Planar *y*-coordinates, specified as a numeric scalar or vector.

Data Types: `single` | `double`

field — Field name
string scalar | character vector

Field name, specified as a string scalar or character vector. `Field` can specify the name of an existing property in the vector data, or the name you want assigned to a new property that you want to add to the vector data.

Data Types: `char` | `string`

value — Value you want to assign to the property specified by `field`
cell array | scalar | vector

Value you want to assign to the property specified by `field`, specified as a cell array, or a scalar or vector of any numeric class or logical.

- When `value` is a cell array, `append` adds the property as a Vertex property.
- When `value` is a numeric array, `append` adds the property as a Feature property.
- When `value` is empty, `append` removes the property.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical` | `char` | `string` | `cell`

Output Arguments

vout — Output geographic or planar vector
`geopoint`, `geoshape`, `mappoint`, or `mapshape` object

Output geographic or planar vector, returned as a `geopoint`, `geoshape`, `mappoint`, or `mapshape` object. The object type of `vout` matches the object type of `vin`.

See Also

`cat`

Introduced in R2012a

arcgridread

(Not recommended) Read gridded data set in ArcGrid ASCII or GridFloat format

Note `arcgridread` is not recommended. Use `readgeoraster` instead. For more information, see "Compatibility Considerations".

Syntax

```
[Z,R] = arcgridread(filename)
[Z,R] = arcgridread(filename,coordinateSystemType)
```

Description

`[Z,R] = arcgridread(filename)` imports a grid in either ArcGrid ASCII or GridFloat format from the file specified by `filename`. Returns Z, a 2-D array containing the data values, and raster referencing information in R. If the input file is accompanied by a projection file (with extension `.prj` or `.PRJ`), then R is a raster reference object whose type matches the coordinate reference system defined in the projection file. Otherwise R is a referencing matrix.

`[Z,R] = arcgridread(filename,coordinateSystemType)` returns R as a raster reference object whose type is consistent with the value specified by `coordinateSystemType`. This optional input argument can be helpful in the absence of a projection file. The function throws an error if a projection file is present and `coordinateSystemType` contradicts the type of coordinate reference system defined in the projection file.

Examples

Load and View Mount Washington Terrain Elevation Data

Read the data.

```
[Z,R] = arcgridread('MtWashington-ft.grd');
```

Display the data as a surface. Add two labels to the display and set the colormap.

```
mapshow(Z,R,'DisplayType','surface')
xlabel('x (easting in meters)')
ylabel('y (northing in meters)')
demcmap(Z)
```
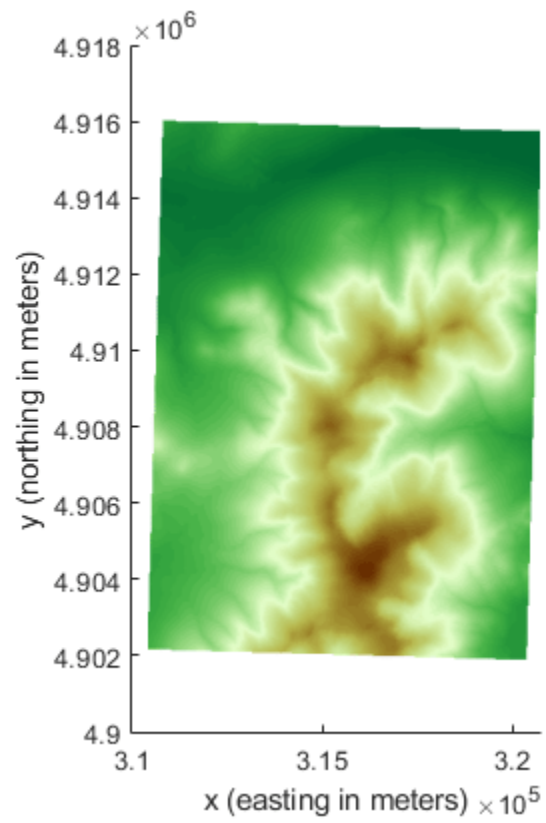
View the terrain in 3-D.

```
axis normal
view(3)
axis equal
grid on
zlabel('elevation in feet')
```
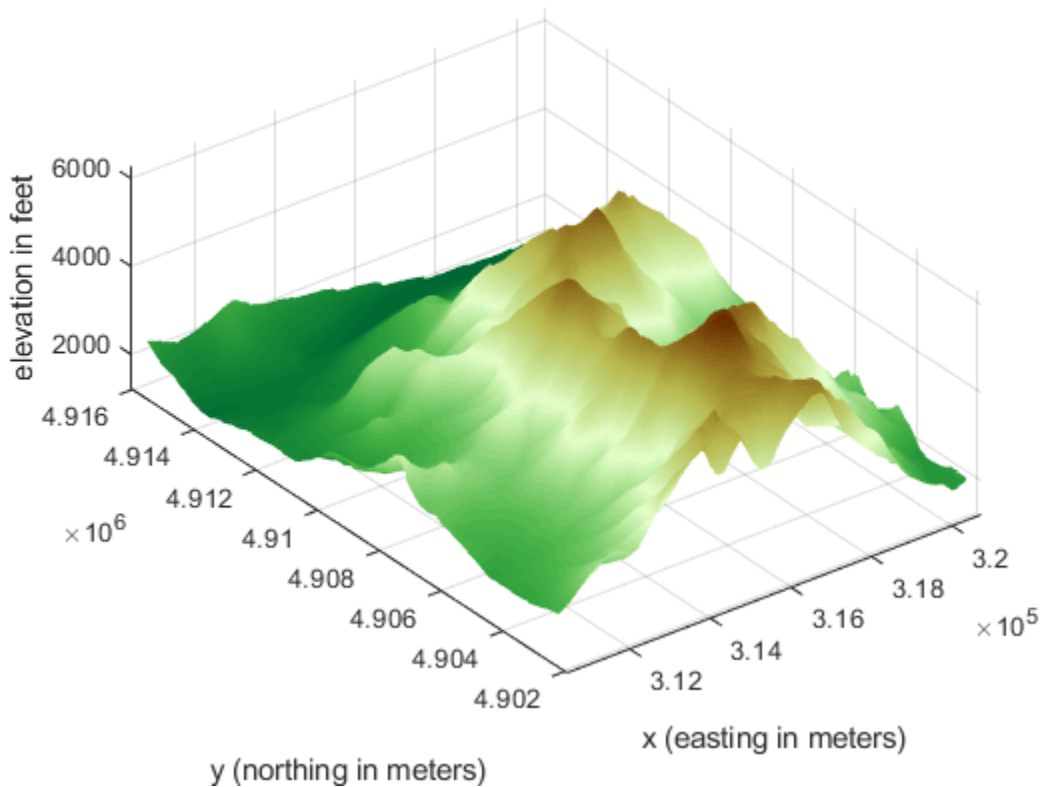
Input Arguments

`filename` — Name of file containing the grid
character vector

Name of file containing the grid, specified as a character vector. The `arcgridread` function supports the following formats.

Format	Description
ArcGrid ASCII	In this format, created by the ArcGIS GRIDASCII command, the data and header information are in a single text file. `arcgridread` will also read a `.prj` file, if one is present. This format is also known as Arc ASCII Grid or Esri ASCII Raster format.
GridFloat	In this format, created by the ArcGIS GRIDFLOAT command, data and header information are in separate files (`.flt` and `.hdr`). Specify the name of the `.flt` file (including the file extension). `arcgridread` will also read a `.prj` file, if one is present. This format is also known as Esri GridFloat.

Data Types: `char`

`coordinateSystemType` — Coordinate system type identifier
`'auto'` (default) | `'geographic'` | `'planar'`

Coordinate system type identifier, specified as one of the following values.

Type Identifier	Description
`'geographic'`	Returns a geographic cells reference object appropriate to a latitude/longitude system.
`'planar'`	Returns a map cells reference object appropriate to a projected map coordinate system.
`'auto'`	Type of raster reference object determined by the file contents.

Data Types: `char`

Output Arguments

Z — Gridded data set
2-D array

Gridded data set, returned as a 2-D array. The class of the array depends on the format of the data, described in the following table. `arcgridread` assigns `NaN` to elements of Z corresponding to null data values in the grid file.

Format	Class of Returned Gridded Data
ArcGrid ASCII	2-D array of class `double`.
GridFloat	2-D array of class `single`.

R — Raster referencing information
raster reference object | referencing matrix

Raster referencing information, returned as a raster reference object whose type matches the coordinates reference system defined in the projection file. If no projection file exists and you do not specify the `coordinateSystemType` parameter, R is a referencing matrix.

Tips

- The `arcgridread` function does not import data in the ArcGrid Binary format (also known as ArcGrid, Arc/INFO Grid, and Esri ArcInfo Grid). ArcGIS uses this format internally and it uses multiple files in a folder with standard names such as `hdr.adf` and `w001001.adf`.

Compatibility Considerations

arcgridread is not recommended
Not recommended starting in R2020a

`arcgridread` is not recommended. Use `readgeoraster` instead. There are no plans to remove `arcgridread`.

Unlike `arcgridread`, which returns a referencing matrix in some cases, the `readgeoraster` function returns a raster reference object. Reference objects have several advantages over referencing matrices.

- Unlike referencing matrices, reference objects have properties that document the size of the associated raster, its limits, and the direction of its rows and columns. For examples of reference object properties, see `GeographicCellsReference` and `MapPostingsReference`.

- You can manipulate the limits of rasters associated with reference objects using the `geocrop` or `mapcrop` functions.
- You can manipulate the size and resolution of rasters associated with reference objects using the `georesize` or `mapresize` functions.

This table shows some typical usages of `arcgridread` and how to update your code to use `readgeoraster` instead.

Not Recommended	Recommended
`[Z,R] = arcgridread(filename);`	`[Z,R] = readgeoraster(filename);`
`[Z,R] = arcgridread(filename,cst);`	`[Z,R] = readgeoraster(filename, ...` `                'CoordinateSystemType',cst);`

The `readgeoraster` function does not automatically replace missing data with `NaN` values. If your data set uses large negative numbers to indicate missing data, replace instances with `NaN` values using the `standardizeMissing` function.

```
[Z,R] = readgeoraster('MtWashington-ft.grd');
info = georasterinfo('MtWashington-ft.grd');
m = info.MissingDataIndicator;
Z = standardizeMissing(Z,m);
```

See Also
worldfileread | readgeoraster

Introduced before R2006a

areaint

Surface area of polygon on sphere or ellipsoid

Syntax

```
area = areaint(lat,lon)
area = areaint(lat,lon,ellipsoid)
area = areaint(lat,lon,units)
area = areaint(lat,lon,ellipsoid,units)
```

Description

`area = areaint(lat,lon)` calculates the spherical surface area of the polygon specified by the input vectors `lat` and `lon`. The calculation uses a line integral approach. The output, `area`, is the fraction of surface area covered by the polygon on a unit sphere. To supply multiple polygons, separate the polygons by NaNs in the input vectors. Accuracy of the integration method is inversely proportional to the distance between lat/lon points.

`area = areaint(lat,lon,ellipsoid)` calculates the surface area of the polygon on the ellipsoid or sphere defined by the input `ellipsoid`, which can be a `referenceSphere`, `referenceEllipsoid`, or `oblateSpheroid` object, or a vector of the form `[semimajor_axis eccentricity]`. The output, `area`, is in squares units corresponding to the units of `ellipsoid`.

`area = areaint(lat,lon,units)` uses the units defined by `units`, the string scalar or character vector `'degrees'` or `'radians'`. If omitted, default units of degrees are assumed.

`area = areaint(lat,lon,ellipsoid,units)` uses both the inputs `ellipsoid` and `units` in the calculation.

Examples

Find Approximate Area of Quadrangle

Consider the area enclosed by a 30º lune from pole to pole and bounded by the prime meridian and 30ºE. You can use the function `areaquad` to get an exact solution:

```
area = areaquad(90,0,-90,30)
```

```
area =
    0.0833
```

This is 1/12 the spherical area. The more points used to define this polygon, the more integration steps `areaint` takes, improving the estimate. This first attempt takes a point every 30º of latitude:

```
lats = [-90:30:90,60:-30:-60]';
lons = [zeros(1,7), 30*ones(1,5)]';
area = areaint(lats,lons)
```

```
area =
    0.0792
```

Now, calculate a better estimate, with one point every 1º of latitude:

```
lats = [-90:1:90,89:-1:-89]';
lons = [zeros(1,181), 30*ones(1,179)]';
area = areaint(lats,lons)
```

```
area =
    0.0833
```

Calculate Area of Geographic Polygons

Load the latitude and longitude coordinates of the perimeter of the conterminous United States. Display the coordinates on a map as a polygon.

```
load('conus.mat','uslat','uslon')
figure
usamap([min(uslat) max(uslat)],[min(uslon) max(uslon)])
geoshow(uslat,uslon,'DisplayType','polygon')
```
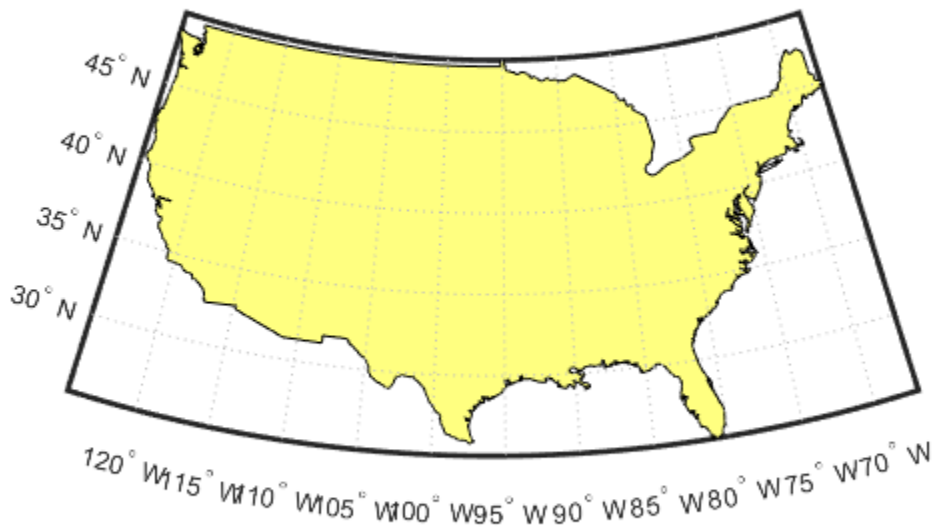

Calculate the area of the conterminous United States in square kilometers. In this case, the `areaint` function returns three areas. The largest area is for the polygon representing the conterminous United States. The other two areas are for the polygons representing Long Island and Martha's Vineyard.

```
rKilometers = earthRadius('km');
a = areaint(uslat,uslon,rKilometers)
```

```
a = 3×1
10⁶ ×

    7.9256
    0.0035
    0.0004
```

Load the latitude and longitude coordinates of the perimeter of the Great Lakes. Display the coordinates on a map as a polygon.

```
load('conus.mat','gtlakelat','gtlakelon')
figure
usamap([min(gtlakelat) max(gtlakelat)],[min(gtlakelon) max(gtlakelon)])
geoshow(gtlakelat,gtlakelon,'DisplayType','polygon')
```
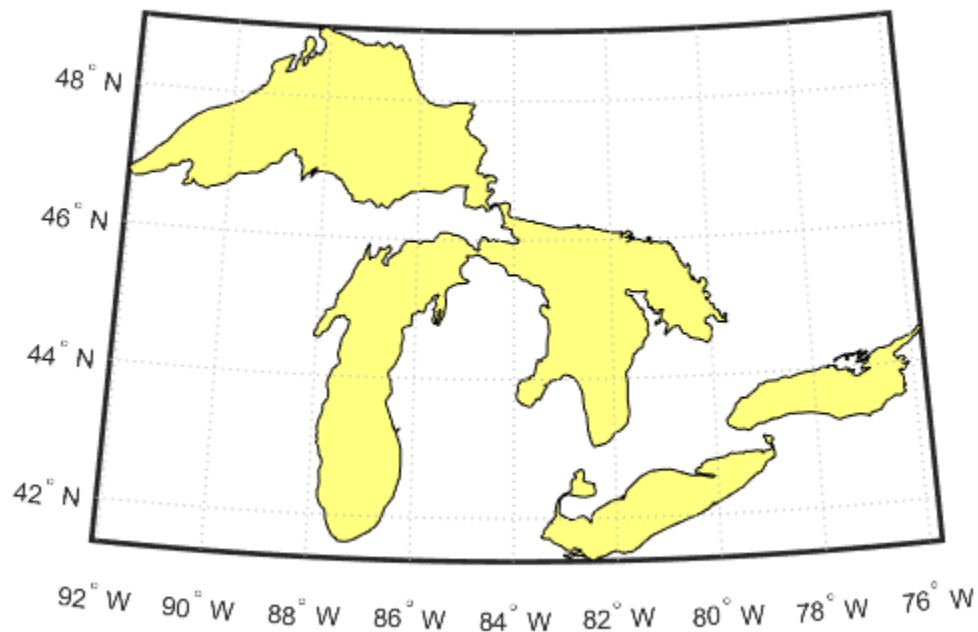

Calculate the areas of the Great Lakes in square miles. In this case, the `areaint` function returns three areas. The largest area is for the polygon representing Lake Superior, Lake Michigan, and Lake Huron. The other two areas are for the polygons representing Lake Erie and Lake Ontario.

```
rMiles = earthRadius('mi');
area = areaint(gtlakelat,gtlakelon,rMiles)
```

```
area = 3×1
10⁴ ×
```

```
8.0120
1.0382
0.7634
```

Tips

Regardless of the polygon vertex order, the values returned by `areaint` are positive.

Algorithms

This function enables the measurement of areas enclosed by arbitrary polygons. This is a numerical estimate, using a line integral based on Green's Theorem. As such, it is limited by the accuracy and resolution of the input data.

Areas are computed for arbitrary polygons on the ellipsoid or sphere

An area is returned for each NaN-separated polygon

Given sufficient data, the `areaint` function is the best method for determining the areas of complex polygons, such as continents, cloud cover, and other natural or derived features. The calculations in this function employ a spherical Earth assumption. For nonspherical ellipsoids, the latitude data is converted to the auxiliary authalic sphere.

See Also
almanac | areamat | areaquad

Topics
"Create and Display Polygons"

Introduced before R2006a

areamat

Surface area covered by nonzero values in binary data grid

Syntax

```
A = areamat(BW,R)
A = areamat(BW,R,ellipsoid)
[A, cellarea] = areamat(...)
```

Description

`A = areamat(BW,R)` returns the surface area covered by the elements of the binary regular data grid BW, which contain the value 1 (`true`). BW can be the result of a logical expression such as `BW = (topo60c > 0)`. R can be a geographic raster reference object, a referencing vector, or a referencing matrix.

If R is a geographic raster reference object, its `RasterSize` property must be consistent with `size(BW)` and its `RasterInterpretation` must be `'cells'`.

If R is a referencing vector, it must be a 1-by-3 with elements:

`[cells/degree northern_latitude_limit western_longitude_limit]`

If R is a referencing matrix, it must be 3-by-2 and transform raster row and column indices to or from geographic coordinates according to:

`[lon lat] = [row col 1] * R`

If R is a referencing matrix, it must define a (non-rotational, non-skewed) relationship in which each column of the data grid falls along a meridian and each row falls along a parallel. Nearest-neighbor interpolation is used by default. NaN is returned for points outside the grid limits or for which `lat` or `lon` contain NaN. All angles are in units of degrees.

The output A expresses surface area as a fraction of the surface area of the unit sphere (4*pi), so the result ranges from 0 to 1.

`A = areamat(BW,R,ellipsoid)` calculates the surface area on the ellipsoid or sphere defined by the input `ellipsoid`, which can be a `referenceSphere`, `referenceEllipsoid`, or `oblateSpheroid` object, or a vector of the form `[semimajor_axis eccentricity]`. The units of the output, A, are the square of the length units in which the semimajor axis is provided. For example, if `ellipsoid` is replaced with `wgs84Ellipsoid('kilometers')`, then A is in square kilometers. If you do not specify `ellipsoid` and R is a reference object with a non-empty `GeographicCRS` property, then `areamat` uses the ellipsoid contained in the `Spheroid` property of the `geocrs` object in the `GeographicCRS` property of R.

`[A, cellarea] = areamat(...)` returns a vector, `cellarea`, describing the area covered by the data cells in BW. Because all the cells in a given row are exactly the same size, only one value is needed per row. Therefore `cellarea` has size `M-by-1`, where `M = size(BW,1)` is the number of rows in BW.

Examples

Find Surface Area in Normalized Units

Find the surface area in normalized units of the part of Earth's terrain that is above sea level.

First, load elevation raster data and a geographic cells reference object. The raster contains terrain heights relative to mean sea level. Then, create a logical array representing the terrain above sea level.

```
load topo60c
topoASL = topo60c > 0;
```

Find the surface area in normalized units of the elements of the array that contain `true`.

```
areamat(topoASL,topo60cR)
```

```
ans = 0.2890
```

The result means 28.9% of Earth's terrain is above sea level.

Find Surface Area in Kilometers

Find the surface area in kilometers of the part of Earth's terrain that is above sea level.

First, load elevation raster data and a geographic cells reference object. The raster contains terrain heights relative to mean sea level. Then, create a reference sphere for Earth and specify its units as kilometers.

```
load topo60c
s = referenceSphere('earth','km');
```

Create a logical array representing the terrain above sea level.

```
topoASL = topo60c > 0;
```

Find the surface area in kilometers of the elements of the array that contain `true`.

```
areamat(topoASL,topo60cR,s)
```

```
ans = 1.4739e+08
```

The result means that approximately 147 million square kilometers of Earth's terrain is above sea level.

Tips

Given a regular data grid that is a logical 0-1 matrix, the `areamat` function returns the area corresponding to the true, or 1, elements. The input data grid can be a logical statement, such as (`topo60c > 0`), which is 1 everywhere that `topo60c` is greater than 0 meters, and 0 everywhere else. This is an illustration of that matrix:

This calculation is based on the `areaquad` function and is therefore limited only by the granularity of the cellular data.

See Also

areaint | areaquad

Introduced before R2006a

areaquad

Surface area of latitude-longitude quadrangle

Syntax

```
area = areaquad(lat1,lon1,lat2,lon2)
area = areaquad(lat1,lon1,lat2,lon2,ellipsoid)
area = areaquad(lat1,lon1,lat2,lon2,ellipsoid,units)
```

Description

`area = areaquad(lat1,lon1,lat2,lon2)` returns the surface area bounded by the parallels `lat1` and `lat2` and the meridians `lon1` and `lon2`. The output `area` is a fraction of the unit sphere's area of 4π, so the result ranges from 0 to 1.

`area = areaquad(lat1,lon1,lat2,lon2,ellipsoid)` allows the specification of the ellipsoid model with `ellipsoid`. `ellipsoid` is a `referenceSphere`, `referenceEllipsoid`, or `oblateSpheroid` object, or a vector of the form `[semimajor_axis eccentricity]`. When `ellipsoid` is input, the resulting `area` is given in terms of the (squared) units of the ellipsoid. For example, if the ellipsoid `referenceEllipsoid('grs80','kilometers')` is used, the resulting area is in km^2.

`area = areaquad(lat1,lon1,lat2,lon2,ellipsoid,units)` where `units` specifies the units of the inputs. The default is `'degrees'`.

Examples

Find the fraction of the Earth's surface that lies between 30ºN and 45ºN, and also between 25ºW and 60ºE:

```
area = areaquad(30,-25,45,60)

area =
    0.0245
```

Assuming a spherical ellipsoid, find the surface area of the Earth in square kilometers.

```
earthellipsoid = referenceSphere('earth','km');
area = areaquad(-90,-180,90,180,earthellipsoid)

area =
   5.1006e+08
```

For comparison,

```
earthellipsoid.SurfaceArea

ans =
   5.1006e+08
```

More About

Latitude-Longitude Quadrangle

A latitude-longitude quadrangle is a region bounded by two meridians and two parallels. In spherical geometry, it is the intersection of a *lune* (a section bounded by two meridians) and a *zone* (a section bounded by two parallels).
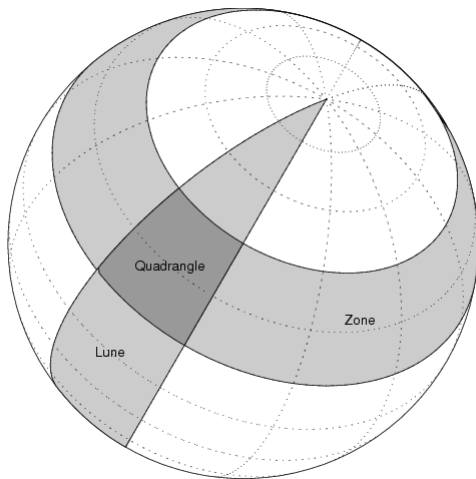
Algorithms

The `areaquad` calculation is exact, being based on simple spherical geometry. For nonspherical ellipsoids, the data is converted to the auxiliary authalic sphere.

See Also
almanac | areaint | areamat

Introduced before R2006a

map.geodesy.AuthalicLatitudeConverter

Convert between geodetic and authalic latitudes

Description

An `AuthalicLatitudeConverter` object provides conversion methods between geodetic and authalic latitudes for an ellipsoid with a given eccentricity.

The authalic latitude maps an ellipsoid (oblate spheroid) to a sphere while preserving surface area. Use authalic latitudes when implementing equal area map projections on the ellipsoid.

Creation

Syntax

```
converter = map.geodesy.AuthalicLatitudeConverter
converter = map.geodesy.AuthalicLatitudeConverter(spheroid)
```

Description

`converter = map.geodesy.AuthalicLatitudeConverter` creates an `AuthalicLatitudeConverter` object for a sphere and sets the `Eccentricity` to `0`.

`converter = map.geodesy.AuthalicLatitudeConverter(spheroid)` creates an authalic latitude converter object and sets the `Eccentricity` property to match the specified spheroid object.

Input Arguments

spheroid — Reference spheroid
`referenceEllipsoid` object | `oblateSpheroid` object | `referenceSphere` object

Reference spheroid, specified as a `referenceEllipsoid` object, `oblateSpheroid` object, or `referenceSphere` object. The term reference spheroid is used synonymously with reference ellipsoid. To create a reference spheroid, use the creation function for the object. To specify the reference ellipsoid for WGS84, use the `wgs84Ellipsoid` function.

For more information about reference spheroids, see "Reference Spheroids".

Example: `spheroid = referenceEllipsoid('GRS 80');`

Properties

Eccentricity — Ellipsoid eccentricity
`0` | numeric scalar

Ellipsoid eccentricity, specified as a numeric scalar. `Eccentricity` is in the interval [0, 0.5]. Eccentricities larger than 0.5 are possible in theory, but do not occur in practice and are not supported.

Data Types: `double`

Object Functions

forward   Convert geodetic latitude to authalic, conformal, isometric, or rectifying latitude
inverse   Convert authalic, conformal, isometric, or rectifying latitude to geodetic latitude

Examples

Create an Authalic Converter

Create a Geodetic Reference System 1980 (grs80) reference ellipsoid.

```
grs80 = referenceEllipsoid('GRS 80');
```

Create an authalic converter object and set the value of the `Eccentricity` property.

```
conv1 = map.geodesy.AuthalicLatitudeConverter;
conv1.Eccentricity = grs80.Eccentricity
```

```
conv1 =

  AuthalicLatitudeConverter with properties:

    Eccentricity: 0.0818
```

Create an Authalic Converter Specifying a Spheroid

Create a reference ellipsoid.

```
grs80 = referenceEllipsoid('GRS 80');
```

Create an authalic latitude converter object based on the ellipsoid.

```
conv2 = map.geodesy.AuthalicLatitudeConverter(grs80)
```

```
conv1 =

  AuthalicLatitudeConverter with properties:

    Eccentricity: 0.0818
```

See Also

Functions
geocentricLatitude | parametricLatitude

Objects
ConformalLatitudeConverter | IsometricLatitudeConverter | RectifyingLatitudeConverter

Introduced in R2013a

avhrrgoode

Read AVHRR data product stored in Goode Projection

Syntax

```
[latgrat,longrat,z] = avhrrgoode(region,filename)
[...] = avhrrgoode(region,filename,scalefactor)
[...] = avhrrgoode(region,filename,scalefactor,latlim,lonlim)
[...] = avhrrgoode(region,filename,scalefactor,latlim,lonlim,gsize)
[...] = avhrrgoode(region,filename,scalefactor,latlim,lonlim,gsize,...
nrows,ncols)
[...] = avhrrgoode(region,filename,scalefactor,latlim,lonlim,gsize,...
nrows,ncols,resolution)
[...] = avhrrgoode(region,filename,scalefactor,latlim,lonlim,gsize,...
nrows,ncols,resolution,precision)
```

Description

`[latgrat,longrat,z] = avhrrgoode(region,filename)` reads data from an Advanced Very High Resolution Radiometer (AVHRR) data set with a nominal resolution of 1 km that is stored in the Goode projection. Data in this format includes a nondimensional vegetation index (NDVI) and Global Land Cover Characteristics (GLCC) data sets. `region` specifies the geographic coverage of the file, using the following values:

- `'g'` or `'global'`
- `'af'` or `'africa'`
- `'ap'` or `'australia/pacific'`
- `'ea'` or `'eurasia'`
- `'na'` or `'north america'`
- `'sa'` or `'south america'`

`filename` is a string scalar or character vector specifying the name of the data file. Output `Z` is a geolocated data grid with coordinates `latgrat` and `longrat` in units of degrees. `Z`, `latgrat`, and `longrat` are of class double. Projected coordinates that lie within the interrupted areas of the projection are set to NaN. A scale factor of 100 is applied to the original data set, so that `Z` contains every 100th point in both X and Y directions.

`[...] = avhrrgoode(region,filename,scalefactor)` uses the integer `scalefactor` to downsample the data. A scale factor of 1 returns every point. A scale factor of 10 returns every 10th point. The default value is 100.

`[...] = avhrrgoode(region,filename,scalefactor,latlim,lonlim)` returns data for the specified region. The returned data can extend somewhat beyond the requested area. Limits are two-element vectors in units of degrees, with `latlim` in the range `[-90 90]` and `lonlim` in the range `[-180 180]`. `latlim` and `lonlim` must be ascending. If `latlim` and `lonlim` are empty, the entire area covered by the data file is returned. If the quadrangle defined by `latlim` and `lonlim` (when projected to form a polygon in the appropriate Goode projection) fails to intersect the bounding box of the data in the projected coordinates, then `Z`, `latgrat`, and `longrat` are returned as empty.

[...] = avhrrgoode(region,filename,scalefactor,latlim,lonlim,gsize) controls the size of the graticule matrices. gsize is a two-element vector containing the number of rows and columns desired. By default, latgrat, and longrat have the same size as Z.

[...] = avhrrgoode(region,filename,scalefactor,latlim,lonlim,gsize,... nrows,ncols) overrides the dimensions for the standard file format for the selected region. This syntax is useful for data stored on CD-ROM, which may have been truncated to fit. Some global data sets were distributed with 16347 rows and 40031 columns of data on CD-ROMs. The default size for global data sets is 17347 rows and 40031 columns of data.

[...] = avhrrgoode(region,filename,scalefactor,latlim,lonlim,gsize,... nrows,ncols,*resolution*)reads a data set with the spatial resolution specified in meters. Specify resolution as either 1000 or 8000 (meters). If empty, the full resolution of 1000 meters is assumed. Data is also available at 8000-meter resolution. Nondimensional vegetation index data at 8-km spatial resolution has 2168 rows and 5004 columns.

[...] = avhrrgoode(region,filename,scalefactor,latlim,lonlim,gsize,... nrows,ncols,*resolution*,*precision*) reads a data set expecting the integer precision specified. If empty, 'uint8' is assumed. 'uint16' is appropriate for some files. Check the metadata (.txt or README) file in the GLCC ftp folder for specification of the file format and contents. In either case, Z is converted to class double.

Background

The United States maintains a family of satellite-based sensors to measure climate change under the Earth Observing System (EOS) program. The precursors to the EOS data are the data sets produced by NOAA and NASA under the Pathfinder program. These are data derived from the Advanced High Resolution Radiometer sensor flown on the NOAA Polar Orbiter satellites, NOAA-7, -9, and -11, and have spatial resolutions of about 1 km. The data from the AVHRR sensor is processed into separate land, sea, and atmospheric indices. Land area data is processed to a nondimensional vegetation index (NDVI) or land cover classification and stored in binary files in the Plate Carrée, Goode, and Lambert projections. Sea data is processed to surface temperatures and stored in HDF formats. avhrrgoode reads land data saved in the Goode projection with global and continental coverage at 1 km. It can also read 8 km data with global coverage.

Limitations

Most files store the data in scaled integers. Though this function returns the data as double, the scaling from integer to float is not performed. Check the data's README file for the appropriate scaling parameters.

Examples

Example 1 — Downsampled Classified Global GLCC Coverage

Read and display every 50th point from the Global Land Cover Characteristics (GLCC) file covering the entire globe with the USGS classification scheme, named gusgs2_0g.img. (To run the example, you must first download the file.)

```
[latgrat, longrat, Z] = avhrrgoode('global', ...
    'gusgs2_0g.img',50);
```

```
% Convert the geolocated data grid to an geolocated image.
uniqueClasses = unique(Z);
RGB = ind2rgb8(uint8(Z), jet(numel(uniqueClasses)));

% Display the data as an image using the Goode projection.
origin = [0 0 0];
ellipsoid = [6370997 0];
figure
axesm('MapProjection', 'goode', 'Origin', origin, ...
    'Geoid', ellipsoid)
geoshow(latgrat, longrat, RGB, 'DisplayType', 'image');
axis image off

% Plot the coastlines.
hold on
load coastlines
plotm(coastlat,coastlon)
```
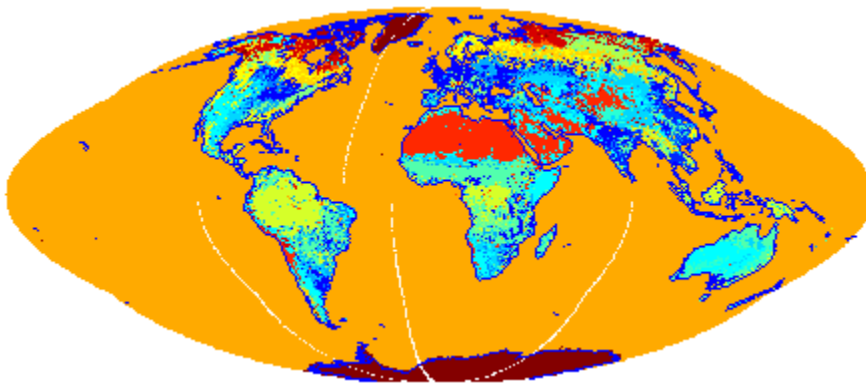

Example 2 — Classified GLCC Data for California

Read and display every point from the Global Land Cover Characteristics (GLCC) file covering California with the USGS classification scheme, named `nausgs1_2g.img`. You must first download the file to run this example.

```
figure
usamap california
mstruct = gcm;
latlim = mstruct.maplatlimit;
lonlim = mstruct.maplonlimit;
scalefactor = 1;
[latgrat, longrat, Z] = ...
    avhrrgoode('na', 'nausgs1_2g.img', scalefactor, latlim, lonlim);
geoshow(latgrat, longrat, Z, 'DisplayType', 'texturemap');

% Overlay vector data from usastatehi.shp.
california = shaperead('usastatehi', 'UseGeoCoords', true,...
    'BoundingBox', [lonlim;latlim]);
geoshow([california.Lat], [california.Lon], 'Color', 'black');
```
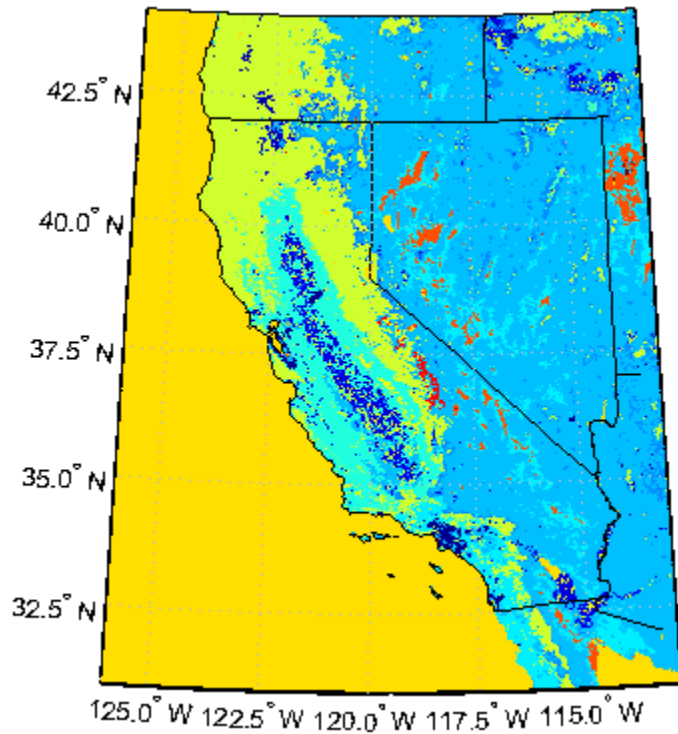
Tips

This function reads the binary files as is. You should not use byte-swapping software on these files.

The AVHRR project and data sets are described in and provided by various U.S. Government Web sites. See the entry for Global Land Cover Characteristics (GLCC) in the tech note referred to below.

Note For details on locating map data for download over the Internet, see the following documentation at the MathWorks Web site: "Find Geospatial Data Online".

See Also
avhrrlambert

Introduced before R2006a

avhrrlambert

Read AVHRR data product stored in `eqaazim` projection

Syntax

```
[latgrat,longrat,Z] = avhrrlambert(region,filename)
[...] = avhrrlambert(region,filename, scalefactor)
[...] = avhrrlambert(region,filename, scalefactor, latlim, lonlim)
[...] = avhrrlambert(region,filename, scalefactor, latlim, lonlim, gsize)
[...] = avhrrlambert(region,filename, scalefactor, latlim, lonlim,
gsize,precision)
```

Description

`[latgrat,longrat,Z] = avhrrlambert(region,filename)` reads data from an Advanced Very High Resolution Radiometer (AVHRR) data set with a nominal resolution of 1 km that is stored in the Lambert Equal Area Azimuthal projection. Data of this type includes the Global Land Cover Characteristics (GLCC). `region` specifies the coverage of the file. Valid regions are listed in the following table. `filename` is a string specifying the name of the data file. `Z` is a geolocated data grid with coordinates `latgrat` and `longrat` in units of degrees. A scale factor of 100 is applied to the original data set such that `Z` contains every 100th point in both X and Y.

Region Specifiers
`'a'` or `'asia'`
`'af'` or `'africa'`
`'ap'` or `'australia/pacific'`
`'e'` or `'europe'`
`'na'` or `'north america'`
`'sa'` or `'south america'`

`[...] = avhrrlambert(region,filename, scalefactor)` uses the integer `scalefactor` to downsample the data. A scale factor of 1 returns every point. A scale factor of 10 returns every 10th point. The default value is 100.

`[...] = avhrrlambert(region,filename, scalefactor, latlim, lonlim)` returns data for the specified region. The result may extend somewhat beyond the requested area. The limits are two-element vectors in units of degrees, with `latlim` in the range `[-90 90]` and `lonlim` in the range `[-180 180]`. If `latlim` and `lonlim` are empty, the entire area covered by the data file is returned. If the quadrangle defined by `latlim` and `lonlim` (when projected to form a polygon in the appropriate Lambert Equal Area Azimuthal projection) fails to intersect the bounding box of the data in the projected coordinates, then `latgrat`, `longrat`, and `Z` are empty.

`[...] = avhrrlambert(region,filename, scalefactor, latlim, lonlim, gsize)` controls the size of the graticule matrices. `gsize` is a two-element vector containing the number of rows and columns desired. If omitted or empty, a graticule the size of the grid is returned.

`[...] = avhrrlambert(region,filename, scalefactor, latlim, lonlim, gsize,precision)` reads a data set with the integer `precision` specified. If omitted, `'uint8'` is

assumed. `'uint16'` is appropriate for some files. Check the metadata (`.txt` or README) file in the ftp folder for specification of the file format and contents.

Background

The United States plans to build a family of satellite-based sensors to measure climate change under the Earth Observing System (EOS) program. Early precursors to the EOS data are the data sets produced by NOAA and NASA under the Pathfinder program. These are data derived from the Advanced High Resolution Radiometer sensor flown on the NOAA Polar Orbiter satellites, NOAA-7, -9, and -11 with a spatial resolution of about 1 km. The data from the AVHRR sensor is processed into separate land, sea, and atmospheric indices. Land area data is processed to a nondimensional vegetation index or land cover classification and stored in binary files in the Plate Carrée, Goode, and Lambert Equal Area Azimuthal projections. Sea data is processed to surface temperatures and stored in HDF formats. This function reads land cover data for the continents saved in the Lambert Equal Area Azimuthal projection at 1 km.

Examples

Example 1

Read and display every 100th point from the Global Land Cover Characteristics (GLCC) file covering North America with the USGS classification scheme, named `nausgs1_2l.img`. To run this example, you must first download the file.

```
[latgrat, longrat, Z] = avhrrlambert('na','nausgs1_2l.img');
```

Display the data using the Lambert Equal Area Azimuthal projection.

```
origin = [50 -100 0];
ellipsoid = [6370997 0];
figure
axesm('MapProjection', 'eqaazim', 'Origin', ...
   origin, 'Geoid', ellipsoid)
geoshow(latgrat, longrat, Z, 'DisplayType', 'texturemap');
```
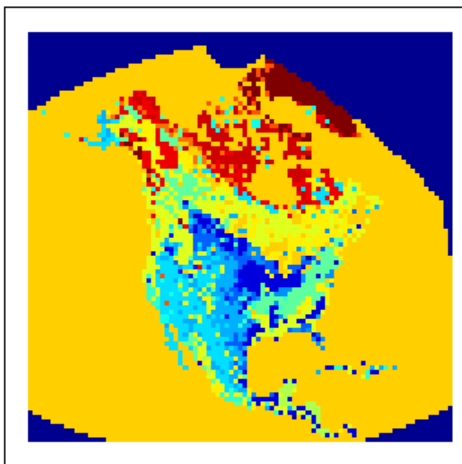
Example 2

Read and display every other point from the Global Land Cover Characteristics (GLCC) file covering Europe with the USGS classification scheme, named `eausgs1_2le.img`. To run this example, you must first download the file.

```
figure
worldmap france
mstruct = gcm;
latlim = mstruct.maplatlimit;
lonlim = mstruct.maplonlimit;
scalefactor = 2;
[latgrat, longrat, Z] = avhrrlambert('e', 'eausgs1_2le.img', ...
    scalefactor, latlim, lonlim);
geoshow(latgrat, longrat, Z, 'DisplayType', 'texturemap');
geoshow('landareas.shp','FaceColor','none','EdgeColor','black')
```
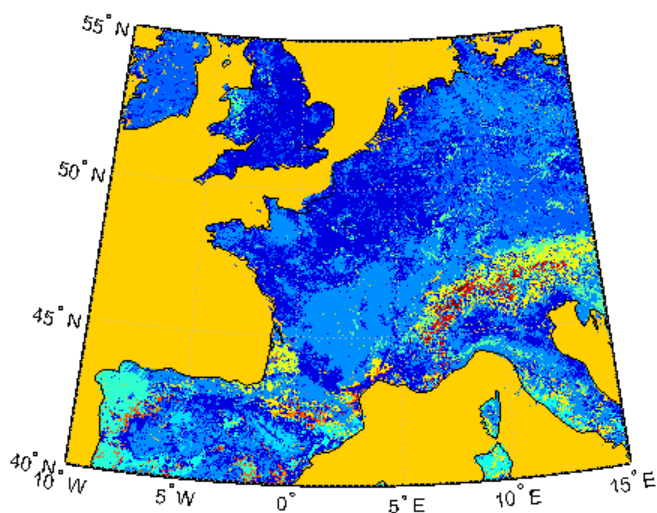

Tips

This function reads the binary files as is. You should not use byte-swapping software on these files.

The AVHRR project and data sets are described in and provided by various U.S. Government Web sites.

Note For details on locating map data for download over the Internet, see the following documentation at the MathWorks Web site: "Find Geospatial Data Online".

See Also
`avhrrgoode`

Introduced before R2006a

axes2ecc

Eccentricity of ellipse from axes lengths

Note Support for nonscalar input, including the syntax `ecc = axes2ecc(vec)`, will be removed in a future release.

Syntax

```
ecc = axes2ecc(semimajor,semiminor)
ecc = axes2ecc(vec)
```

Description

`ecc = axes2ecc(semimajor,semiminor)` computes the eccentricity of an ellipse (or ellipsoid of revolution) given the semimajor and semiminor axes. The input data can be scalar or matrices of equal dimensions.

`ecc = axes2ecc(vec)` assumes a 2 element vector (`vec`) is supplied, where `vec = [semimajor semiminor]`.

See Also
`ecc2flat` | `ecc2n` | `majaxis` | `minaxis`

Introduced before R2006a

axesm

Create map axes

Syntax

```
axesm
axesm(Name,Value)
axesm(projid,Name,Value)
```

Description

The `axesm` function creates a map axes into which both vector and raster geographic data can be projected using functions such as `plotm` and `geoshow`. Properties specific to map axes can be assigned upon creation with `axesm`, and for an existing map axes they can be queried and changed using `getm` and `setm`. Use the standard `get` and `set` methods to query and control the standard MATLAB axes properties of a map axes.

Map axes are standard MATLAB axes with different default settings for some properties and a MATLAB structure for storing projection parameters and other data. The main differences in default settings are:

- Axes properties `XGrid`, `YGrid`, `XTick`, `YTick` are set to `'off'`.
- The `hold` mode is `'on'`.

The map projection structure stores the map axes properties, which, in addition to the special standard axes settings, allow Mapping Toolbox functions to recognize an axes or an opened FIG-file as a map axes. See Map Axes Properties for descriptions of the map axes properties.

`axesm` with no input arguments, initiates the `axesmui` map axes graphical user interface, which can be used to set map axes properties. This is detailed on the `axesmui` reference page.

`axesm(Name,Value)` creates a map axes and modifies the map axes appearance using name-value pairs to set properties. You can specify multiple name-value pairs. Enclose each property name in quotes. For example, `'FontSize',14` sets the font size for the map axes text. Properties may be specified in any order, but the `MapProjection` property must be included. For a full list of properties, see Map Axes Properties.

`axesm(projid,Name,Value)` specifies which map projection to use. `projid` should match one of the entries in the last column displayed by the `maps` function. You can also find these listed in "Summary and Guide to Projections".

Examples

Create Map Axes for Mercator Projection

Create map axes for a Mercator projection, with selected latitude limits:

```
axesm('MapProjection','mercator','MapLatLimit',[-70 80])
```

All properties not explicitly addressed in the call are set to either fixed or calculated defaults. The file mercator.m defines a projection function, so the same result could have been achieved with the function.

```
axesm('mercator','MapLatLimit',[-70 80])
```

Each projection function includes default values for all properties. Any following property name/property value pairs are treated as overrides.

In either of the above examples, data displayed in the given map axes is in a Mercator projection. Any data falling outside the prescribed limits is not displayed.

Access and Change Map Axes Properties

Similar to accessing and manipulating standard axes properties by using the get and set functions, you can access and manipulate map axes properties by using the getm and setm functions.

Use the axesm function only to *create* a map axes object. Use the setm function to *modify* an existing map axes.

Create a map axes object that contains no map data. Note that you specify MapProjection ID values in lowercase.

```
axesm('MapProjection','miller','Frame','on')
```
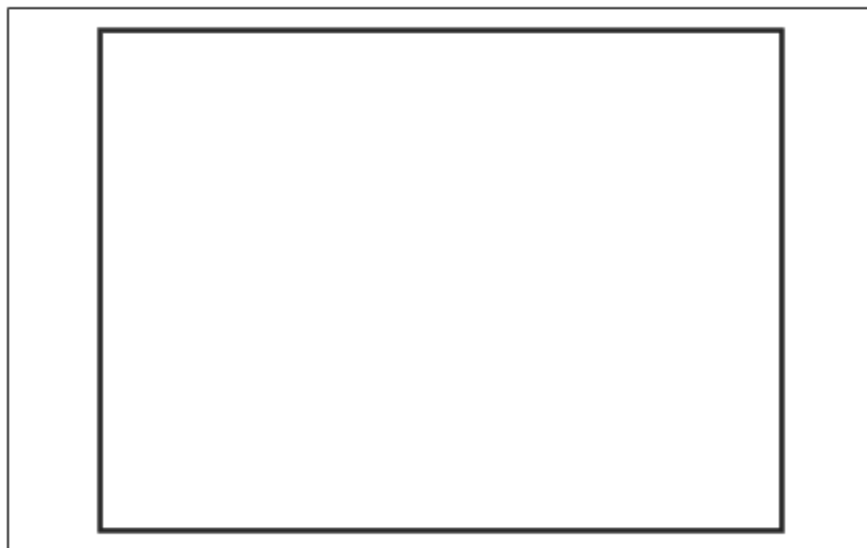
At this point you can begin to customize the map. For example, you might decide to make the frame lines bordering the map thicker. First, you need to identify the current line width of the frame, which you do by querying the current axes, identified as `gca`.

```
getm(gca,'FLineWidth')
```

```
ans = 2
```

Now reset the line width to four points. The default `fontunits` value for axes is `points`. You can set `fontunits` to be `points`, `normalized`, `inches`, `centimeters`, or `pixels`.

```
setm(gca,'FLineWidth',4)
```


You can set any number of properties simultaneously with `setm`. Continue by reducing the line width, changing the projection to equidistant cylindrical, and verify the changes.

```
setm(gca,'FLineWidth',3,'Grid','on','MapProjection','robinson')
```
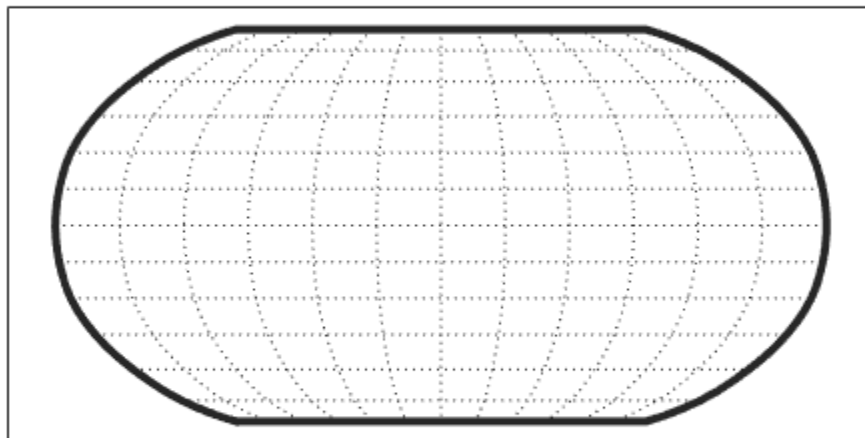
```
getm(gca,'FLineWidth')

ans = 3

getm(gca,'MapProjection')

ans =
'robinson'
```

Inspect the entire set of map axes properties at their current settings. Note that the list of properties includes both those particular to map axes and general ones that apply to all MATLAB® axes.

```
getm(gca)

ans = struct with fields:
    mapprojection: 'robinson'
             zone: []
       angleunits: 'degrees'
           aspect: 'normal'
     falsenorthing: 0
      falseeasting: 0
       fixedorient: []
             geoid: [1 0]
        maplatlimit: [-90 90]
        maplonlimit: [-180 180]
      mapparallels: 38
        nparallels: 0
            origin: [0 0 0]
```

```
        scalefactor: 1
            trimlat: [-90 90]
            trimlon: [-180 180]
              frame: 'on'
              ffill: 100
          fedgecolor: [0.1500 0.1500 0.1500]
          ffacecolor: 'none'
           flatlimit: [-90 90]
          flinewidth: 3
           flonlimit: [-180 180]
               grid: 'on'
           galtitude: Inf
             gcolor: [0.1500 0.1500 0.1500]
          glinestyle: ':'
          glinewidth: 0.5000
       mlineexception: []
           mlinefill: 100
          mlinelimit: []
        mlinelocation: 30
        mlinevisible: 'on'
       plineexception: []
           plinefill: 100
          plinelimit: []
        plinelocation: 15
        plinevisible: 'on'
           fontangle: 'normal'
           fontcolor: [0.1500 0.1500 0.1500]
            fontname: 'Helvetica'
            fontsize: 10
           fontunits: 'points'
          fontweight: 'normal'
         labelformat: 'compass'
        labelrotation: 'off'
          labelunits: 'degrees'
       meridianlabel: 'off'
       mlabellocation: 30
       mlabelparallel: 90
          mlabelround: 0
         parallellabel: 'off'
       plabellocation: 15
       plabelmeridian: -180
          plabelround: 0
```

Similarly, use the `setm` function alone to display the set of properties, their enumerated values, and defaults.

`setm(gca)`

```
AngleUnits                [ {degrees} | radians ]
Aspect                    [ {normal} | transverse ]
FalseEasting
FalseNorthing
FixedOrient               FixedOrient is a read-only property
Geoid
MapLatLimit
MapLonLimit
MapParallels
```

```
MapProjection
NParallels                 NParallels is a read-only property
Origin
ScaleFactor
TrimLat                    TrimLat is a read-only property
TrimLon                    TrimLon is a read-only property
Zone
Frame                      [ on | {off} ]
FEdgeColor
FFaceColor
FFill
FLatLimit
FLineWidth
FLonLimit
Grid                       [ on | {off} ]
GAltitude
GColor
GLineStyle                 [ - | -- | -. | {:} ]
GLineWidth
MLineException
MLineFill
MLineLimit
MLineLocation
MLineVisible               [ {on} | off ]
PLineException
PLineFill
PLineLimit
PLineLocation
PLineVisible               [ {on} | off ]
FontAngle                  [ {normal} | italic | oblique ]
FontColor
FontName
FontSize
FontUnits                  [ inches | centimeters | normalized | {points} | pixels ]
FontWeight                 [ {normal} | bold ]
LabelFormat                [ {compass} | signed | none ]
LabelRotation              [ on | {off} ]
LabelUnits                 [ {degrees} | radians ]
MeridianLabel              [ on | {off} ]
MLabelLocation
MLabelParallel
MLabelRound
ParallelLabel              [ on | {off} ]
PLabelLocation
PLabelMeridian
PLabelRound
```

Many, but not all, property choices and defaults can also be displayed individually.

```
setm(gca,'FontUnits')

FontUnits                  [ inches | centimeters | normalized | {points} | pixels ]

setm(gca,'MapProjection')

An axes's "MapProjection" property does not have a fixed set of property values.

setm(gca,'Frame')
```

```
Frame                        [ on | {off} ]
```

```
setm(gca,'FixedOrient')
```

```
FixedOrient              FixedOrient is a read-only property
```

In the same way, `getm` displays the current value of any axes property.

```
getm(gca,'FontUnits')
```

```
ans =
'points'
```

```
getm(gca,'MapProjection')
```

```
ans =
'robinson'
```

```
getm(gca,'Frame')
```

```
ans =
'on'
```

```
getm(gca,'FixedOrient')
```

```
ans =

    []
```

To identify which properties apply to a given map projection, see the reference page for that projection.

Input Arguments

`projid` — Map projection ID
character vector | string scalar

Map projection ID, specified as a string scalar or character vector. `projid` should match one of the entries in the last column displayed by the `maps` function. You can also find these listed in "Summary and Guide to Projections".

Note The names of projection files are case sensitive. The projection files included in Mapping Toolbox software use only lowercase letters and Arabic numerals.

Example: `'eqdcylin'`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `axesm('MapProjection','pcarree','Frame','on')` creates a map axes with a Plate Carree projection, and makes the map frame visible.

Note The properties listed here are only a subset. For a full list, see Map Axes Properties.

MapProjection — Map projection
character vector | string scalar

Map projection, specified as a string scalar or character vector. `MapProjection` sets the projection, and hence all transformation calculations, for the map axes object. It must be a member of the recognized projection set, which you can list by typing `getm('MapProjection')` or `maps`. For more information on projections, see the *Mapping Toolbox User's Guide*. Some projections set their own defaults for other properties, such as parallels and trim limits.

MapLatLimit — Geographic latitude limits of the display area
two-element vector

Geographic latitude limits of the display area, specified as a two-element vector of the form `[southern_limit northern_limit]`. This property can be set for many typical projections and geometries, but cannot be used with oblique projections or with `globe`, for example.

When applicable, the `MapLatLimit` property may affect the origin latitude if the `Origin` property is not set explicitly when calling `axesm`. It may also determine the value used for `FLatLimit`.

MapLonLimit — Geographic longitude limits of the display area
two-element vector

Geographic longitude limits of the display area, specified as a two-element vector of the form `[western_limit eastern_limit]`. This property can be set for many typical projections and geometries, but cannot be used with oblique projections or with `globe`, for example.

When applicable, the `MapLonLimit` property may affect the origin longitude if the `Origin` property is not set explicitly when calling `axesm`. It may also determine the value used for `FLonLimit`.

Tips

- In general, after re-opening a saved figure that contains a map axes, you should not attempt to modify the projection properties of that map axes.

- When you create a map axes with `axesm` and right click in the axes, a context menu appears. If you do not need the menu or it interferes with your application, you can disable it by resetting the `'ButtonDownFcn'` property of the axes:

```
ax = axesm('mercator');      % Right-clicking brings up context menu.
set(ax,'ButtonDownFcn',[])   % Context menu has been disabled.
```

- By default, `axesm` does not clip graticules or labels that occur outside the boundaries of the axes. Enable clipping by setting the `'Clipping'` property of these objects.

```
objects = [handlem('grid'); handlem('mlabel'); handlem('plabel')];
set(objects,'Clipping','on');
```

See Also

Properties
Map Axes Properties

Functions
axes | gcm | getm | setm

Topics
"Introduction to Mapping Graphics"
"The Map Axes"
"The Map Frame"
"Map Limit Properties"
"The Map Grid"

Introduced before R2006a

Map Axes Properties

Control axes appearance and behavior

Description

Map axes properties control the appearance and behavior of an `axesm` object. By changing property values, you can modify certain aspects of the map axes.

Properties

Properties That Control the Map Projection

AngleUnits — Angular unit of measure
`'degrees'` (default) | `'radians'`

Angular unit of measure — Controls the units of measure used for angles (including latitudes and longitudes) in the map axes. All input data are assumed to be in the given units; `'degrees'` is the default. For more information on angle units, see "Angle Representations and Angular Units" in the *Mapping Toolbox User's Guide*.

Aspect — Display aspect
`'normal'` (default) | `'transverse'`

Display aspect — Controls the orientation of the base projection of the map. When the aspect is `'normal'` (the default), *north* in the base projection is up. In a `transverse` aspect, north is to the right. A cylindrical projection of the whole world would look like a *landscape* display under a `'normal'` aspect, and like a *portrait* under a `'transverse'` aspect. Note that this property is not the same as projection aspect, which is controlled by the `Origin` property vector discussed later.

FalseEasting — Coordinate shift for projection calculations
`0` (default) | numeric scalar

Coordinate shift for projection calculations — Modifies the position of the map within the axes. The projected coordinates are shifted in the *x*-direction by the amount of `FalseEasting`. The `FalseEasting` is in the same units as the projected coordinates, that is, the units of the first element of the `Geoid` map axes property. False eastings and northings are sometimes used to ensure nonnegative values of the projected coordinates. For example, the Universal Transverse Mercator uses a false easting of 500,000 meters.

FalseNorthing — Coordinate shift for projection calculations
`0` (default) | numeric scalar

Coordinate shift for projection calculations — Modifies the position of the map within the axes. The projected coordinates are shifted in the *y*-direction by the amount of `FalseNorthing`. The `FalseNorthing` is in the same units as the projected coordinates, that is, the units of the first element of the `Geoid` map axes property. False eastings and northings are sometimes used to ensure nonnegative values of the projected coordinates. For example, the Universal Transverse Mercator uses a false northing of 0 in the northern hemisphere and 10,000,000 meters in the southern.

FixedOrient — Projection-based orientation
[] (default) | numeric scalar

This property is read-only.

Projection-based orientation — This read-only property fixes the orientation of certain projections (such as the Cassini and Wetch). When empty, which is true for most projections, the user can alter the orientation of the projection using the third element of the `Origin` property. When fixed, the fixed orientation is always used.

Geoid — Reference spheroid definition
[1 0] (default) | referenceSphere object | referenceEllipsoid object | oblateSpheroid object | [semimajor_axis eccentricity]

Reference spheroid definition — The spheroid (ellipsoid or sphere) for calculating the projections of any displayed map objects. It can be an `referenceSphere`, `referenceEllipsoid`, or `oblateSpheroid` object, or a two-element vector of the form `[semimajor_axis eccentricity]`. The default value is an ellipsoid vector representing the unit sphere: `[1 0]`.

MapLatLimit — Geographic latitude limits of the display area
[southern_limit northern_limit]

Geographic latitude limits of the display area — Expressed as a two-element vector of the form `[southern_limit northern_limit]`. This property can be set for many typical projections and geometries, but cannot be used with oblique projections or with `globe`, for example. When applicable, the `MapLatLimit` property may affect the origin latitude if the `Origin` property is not set explicitly when calling `axesm`. It may also determine the value used for `FLatLimit`.

MapLonLimit — Geographic longitude limits of the display area
[western_limit eastern_limit]

Geographic longitude limits of the display area — Expressed as a two-element vector of the form `[western_limit eastern_limit]`. This property can be set for many typical projections and geometries, but cannot be used with oblique projections or with `globe`, for example. When applicable, the `MapLonLimit` property may affect the origin longitude if the `Origin` property is not set explicitly when calling `axesm`. It may also determine the value used for `FLonLimit`.

MapParallels — Projection standard parallels
[lat] | [lat1 lat2]

Projection standard parallels — Sets the standard parallels of projection. It can be an empty, one-, or two-element vector, depending upon the projection. The elements are in the same units as the map axes `AngleUnits`. Many projections have specific, defining standard parallels. When a map axes object is based upon one of these projections, the parallels are set to the appropriate defaults. For conic projections, the default standard parallels are set to 15ºN and 75ºN, which biases the projection toward the northern hemisphere.

For projections with one defined standard parallel, setting the parallels to an empty vector forces recalculation of the parallel to the middle of the map latitude limits. For projections requiring two standard parallels, setting the parallels to an empty vector forces recalculation of the parallels to one-sixth the distance from the latitude limits (e.g., if the map latitude limits correspond to the northern hemisphere `[0 90]`, the standard parallels for a conic projection are set to `[15 75]`). For azimuthal projections, the `MapParallels` property always contains an empty vector and cannot be altered.

See the *Mapping Toolbox User's Guide* for more information on standard parallels.

MapProjection — Map projection
character vector

Map projection — Sets the projection, and hence all transformation calculations, for the map axes object. It is required in the creation of map axes. It must be a member of the recognized projection set, which you can list by typing `getm('MapProjection')` or `maps`. For more information on projections, see the *Mapping Toolbox User's Guide*. Some projections set their own defaults for other properties, such as parallels and trim limits.

Origin — Origin and orientation for projection calculations
`[latitude longitude orientation]`

Origin and orientation for projection calculations — Sets the map origin for all projection calculations. The latitude, longitude, and orientation should be in the map axes `AngleUnits`. Latitude and longitude refer to the coordinates of the map origin; orientation refers to an angle of skewness or rotation about the axis running through the origin point and the center of the earth. The default origin is 0º latitude and a longitude centered between the map longitude limits. If a scalar is entered, it is assumed to refer to the longitude; if a two-element vector is entered, the default orientation is 0º, a normal projection. If an empty origin vector is entered, the origin is centered on the map longitude limits. For more information on the origin, see the *Mapping Toolbox User's Guide*.

Parallels — Number of standard parallels
`0 | 1 | 2`

This property is read-only.

Number of standard parallels — This read-only property contains the number of standard parallels associated with the projection. See the *Mapping Toolbox User's Guide* for more information on standard parallels.

ScaleFactor — Scale factor for projection calculations
1 (default) | scalar

Scale factor for projection calculations — Modifies the size of the map in projected coordinates. The geographic coordinates are transformed to Cartesian coordinates by the map projection equations and multiplied by the scale factor. Scale factors are sometimes used to minimize the scale distortion in a map projection. For example, the Universal Transverse Mercator uses a scale factor of 0.996 to shift the line of zero scale distortion to two lines on either side of the central meridian.

Zone — Zone for certain projections
`[]` or `31N` (default) | `ZoneSpec`

Zone for certain projections — Specifies the zone for certain projections. A zone is a region on the globe that has a special set of projection parameters. In the Universal Transverse Mercator Projection, the world is divided into quadrangles that are generally 6 degrees wide and 8 degrees tall. The number in the zone designation refers to the longitude range, while the letter refers to the latitude range. Most projections use the same parameters for the entire globe, and do not require a zone.

Properties That Control the Frame

Frame — Frame visibility
`'off'` (default) | `'on'`

Frame visibility — Controls the visibility of the display frame box. When the frame is `'off'` (the default), the frame is not displayed. When the frame is `'on'`, an enclosing frame is visible. The frame

is a patch that is plotted as the lowest layer of displayed map objects. Regardless of its display status, the frame always operates in terms of trimming map data.

FFill — Frame plotting precision
100 (default) | scalar plotting point density

Frame plotting precision — Sets the number of points to be used in plotting the frame for display. The default value is 100, which for a rectangular frame results in a plot with 100 points for each side, or a total of 400 points. The number of points required for a reasonable display varies with the projection. Cylindrical projections such as the Miller require very few. Projections resulting in more complex frames, such as the Werner, look better with higher densities. The default value is generally sufficient.

FEdgeColor — Color of displayed frame edge
[0 0 0] (default) | RGB triplet | hexadecimal color code | 'r' | 'g' | 'b' | ...

Color of displayed frame edge — Specifies the color used for the displayed frame. You can specify an RGB triplet, a hexadecimal color code, a color name, or a short name. By default, the frame edge is displayed in black.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1]; for example, [0.4 0.6 0.7].
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (#) followed by three or six hexadecimal digits, which can range from 0 to F. The values are not case sensitive. Thus, the color codes '#FF8800', '#ff8800', '#F80', and '#f80' are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
'red'	'r'	[1 0 0]	'#FF0000'	
'green'	'g'	[0 1 0]	'#00FF00'	
'blue'	'b'	[0 0 1]	'#0000FF'	
'cyan'	'c'	[0 1 1]	'#00FFFF'	
'magenta'	'm'	[1 0 1]	'#FF00FF'	
'yellow'	'y'	[1 1 0]	'#FFFF00'	
'black'	'k'	[0 0 0]	'#000000'	
'white'	'w'	[1 1 1]	'#FFFFFF'	
'none'	Not applicable	Not applicable	Not applicable	No color

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
[0 0.4470 0.7410]	'#0072BD'	
[0.8500 0.3250 0.0980]	'#D95319'	
[0.9290 0.6940 0.1250]	'#EDB120'	
[0.4940 0.1840 0.5560]	'#7E2F8E'	
[0.4660 0.6740 0.1880]	'#77AC30'	
[0.3010 0.7450 0.9330]	'#4DBEEE'	
[0.6350 0.0780 0.1840]	'#A2142F'	

FFaceColor — Color of displayed frame face
'none' (default) | RGB triplet | hexadecimal color code | 'r' | 'g' | 'b' | ...
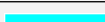
Color of displayed frame face — Specifies the color used for the displayed frame face. You can specify an RGB triplet, a hexadecimal color code, a color name, or a short name. The default, 'none', means no face color is filled in.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1]; for example, [0.4 0.6 0.7].
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (#) followed by three or six hexadecimal digits, which can range from 0 to F. The values are not case sensitive. Thus, the color codes '#FF8800', '#ff8800', '#F80', and '#f80' are equivalent.
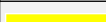
Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
'red'	'r'	[1 0 0]	'#FF0000'	
'green'	'g'	[0 1 0]	'#00FF00'	
'blue'	'b'	[0 0 1]	'#0000FF'	
'cyan'	'c'	[0 1 1]	'#00FFFF'	
'magenta'	'm'	[1 0 1]	'#FF00FF'	
'yellow'	'y'	[1 1 0]	'#FFFF00'	
'black'	'k'	[0 0 0]	'#000000'	
'white'	'w'	[1 1 1]	'#FFFFFF'	
'none'	Not applicable	Not applicable	Not applicable	No color

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
[0 0.4470 0.7410]	'#0072BD'	
[0.8500 0.3250 0.0980]	'#D95319'	
[0.9290 0.6940 0.1250]	'#EDB120'	
[0.4940 0.1840 0.5560]	'#7E2F8E'	
[0.4660 0.6740 0.1880]	'#77AC30'	
[0.3010 0.7450 0.9330]	'#4DBEEE'	
[0.6350 0.0780 0.1840]	'#A2142F'	

FLatLimit — Latitude limits of map frame relative to projection origin
[southern_limit northern_limit]

Latitude limits of map frame relative to projection origin — The map frame encloses the area in which data and graticule lines are plotted and beyond which they are trimmed. For non-oblique and non-azimuthal projections, which have quadrangular frames, this property controls the north-south extent of the frame. If a projection is made oblique by the inclusion of a non-zero rotation angle (the third element of the Origin vector), FLatLimit still applies, but in the rotated latitude-longitude system rather than in the geographic system. In the case of azimuthal projections, which have circular frames, FLatLimit takes the special form [-Inf radius] where radius is the spherical distance (in degrees or radians, depending on the AngleUnits property of the projection) from the projection origin to the edge of the frame.

Note In most common situations, including non-oblique cylindrical and conic projections and polar azimuthal projections, there is no need to set FLatLimit; use MapLatLimit instead.

FLineWidth — Frame edge line width
2 (default) | scalar

Frame edge line width — Sets the line width of the displayed frame edge. The value is a scalar representing points, which is 2 by default.

FLonLimit — Longitude limits of map frame relative to projection origin
[western_limit eastern_limit]

Longitude limits of map frame relative to projection origin — The map frame encloses the area in which data and graticule lines are plotted and beyond which they are trimmed. For non-oblique and non-azimuthal projections, which have quadrangular frames, this property controls the east-west extent of the frame. If a projection is made oblique by the inclusion of a non-zero rotation angle (the third element of the Origin vector), FLonLimit still applies, but in the rotated latitude-longitude system rather than in the geographic system. The FLonLimit property is ignored for azimuthal projections.

Note In most common situations, including non-oblique cylindrical and conic projections, there is no need to set FLonLimit; use MapLonLimit instead.

TrimLat — Bounds on FLatLimit
[southern_limit northern_limit]

This property is read-only.

Bounds on FLatLimit — This read-only property sets bounds on the values that `axesm` and `setm` will accept for the `MapLatLimit` and `FLatLimit` properties, which is necessary because some map projections cannot display the entire globe without extending to infinity. For example, `TrimLat` is `[-90 90]` degrees for most cylindrical projections and `[-86 86]` degrees for the Mercator projection because the north-south scale becomes infinite as one approaches either pole.

TrimLon — Bounds on FLonLimit
[western_limit eastern_limit]

This property is read-only.

Bounds on FLonLimit — This read-only property sets bounds on the values that `axesm` and `setm` will accept for the `MapLonLimit` and `FLonLimit` properties, which is necessary because some map projections cannot display the entire globe without extending to infinity. For example, `TrimLon` is `[-135 135]` degrees for most conic projections.

Properties That Control the Grid

Grid — Grid visibility
'off' (default) | 'on'

Grid visibility — Controls the visibility of the display grid. When the grid is `'off'` (the default), the grid is not displayed. When the grid is `'on'`, meridians and parallels are visible. The grid is plotted as a set of line objects.

GAltitude — Grid *z*-axis setting
Inf (default) | scalar

Grid z-axis setting — Sets the *z*-axis location for the grid when displayed. Its default value is infinity, which is displayed above all other map objects. However, you can set this to some other value for stacking objects above the grid, if desired.

GColor — Color of the displayed grid
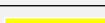[0 0 0] (default) | RGB triplet | hexadecimal color code | 'r' | 'g' | 'b' | ...
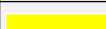
Color of the displayed grid — Specifies the color used for the displayed grid. You can specify an RGB triplet, a hexadecimal color code, a color name, or a short name. By default, the map grid is displayed in black.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (#) followed by three or six hexadecimal digits, which can range from 0 to F. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
'red'	'r'	[1 0 0]	'#FF0000'	
'green'	'g'	[0 1 0]	'#00FF00'	
'blue'	'b'	[0 0 1]	'#0000FF'	
'cyan'	'c'	[0 1 1]	'#00FFFF'	
'magenta'	'm'	[1 0 1]	'#FF00FF'	
'yellow'	'y'	[1 1 0]	'#FFFF00'	
'black'	'k'	[0 0 0]	'#000000'	
'white'	'w'	[1 1 1]	'#FFFFFF'	
'none'	Not applicable	Not applicable	Not applicable	No color

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
[0 0.4470 0.7410]	'#0072BD'	
[0.8500 0.3250 0.0980]	'#D95319'	
[0.9290 0.6940 0.1250]	'#EDB120'	
[0.4940 0.1840 0.5560]	'#7E2F8E'	
[0.4660 0.6740 0.1880]	'#77AC30'	
[0.3010 0.7450 0.9330]	'#4DBEEE'	
[0.6350 0.0780 0.1840]	'#A2142F'	

GLineStyle — Grid line style
: (default) | LineStyle

Grid line style — Determines the style of line used when the grid is displayed. You can specify any line style supported by the MATLAB line function. The default line style is a dotted line (that is, ':').

GLineWidth — Grid line width
0.5 (default) | scalar

Grid line width — Sets the line width of the displayed grid. The value is a scalar representing points, which is 0.5 by default.

MLineException — Exceptions to grid meridian limits
[] (default) | vector

Exceptions to grid meridian limits — Allows specific meridians of the displayed grid to extend beyond the grid meridian limits to the poles. The value must be a vector of longitudes in the appropriate angle units. For longitudes so specified, grid lines extend from pole to pole regardless of the existence of any grid meridian limits. This vector is empty by default.

MLineFill — Grid meridian plotting precision
100 (default) | scalar plotting point density

Grid meridian plotting precision — Sets the number of points to be used in plotting the grid meridians. The default value is 100 points. The number of points required for a reasonable display varies with the projection. Cylindrical projections such as the Miller require very few. Projections resulting in more complex shapes, such as the Werner, look better with higher densities. The default value is generally sufficient.

MLineLimit — Grid meridian limits
[] (default) | [north south] | [south north]

Grid meridian limits — Establishes latitudes beyond which displayed grid meridians do not extend. By default, this property is empty, so the meridians extend to the poles. There are two exceptions to the meridian limits. No meridian extends beyond the map latitude limits, and exceptions to the meridian limits for selected meridians are allowed (see above).

MLineLocation — Grid meridian interval or specific locations
30 (default) | scalar | vector

Grid meridian interval or specific locations — Establishes the interval between displayed grid meridians. When a scalar interval is entered in the map axes MLineLocation, meridians are displayed, starting at 0º longitude and repeating every interval in both directions, which by default is 30º. Alternatively, you can enter a vector of longitudes, in which case a meridian is displayed for each element of the vector.

PLineException — Exceptions to grid parallel limits
[] (default) | vector

Exceptions to grid parallel limits — Allows specific parallels of the displayed grid to extend beyond the grid parallel limits to the International Date Line. The value must be a vector of latitudes in the appropriate angle units. For latitudes so specified, grid lines extend from the western to the eastern map limit, regardless of the existence of any grid parallel limits. This vector is empty by default.

PLineFill — Grid parallel plotting precision
100 (default) | scalar plotting point density

Grid parallel plotting precision — Sets the number of points to be used in plotting the grid parallels. The default value is 100. The number of points required for a reasonable display varies with the projection. Cylindrical projections such as the Miller require very few. Projections resulting in more complex shapes, such as the Bonne, look better with higher densities. The default value is generally sufficient.

PLineLimit — Grid parallel limits
[] (default) | [east west] | [west east]

Grid parallel limits — Establishes longitudes beyond which displayed grid parallels do not extend. By default, this property is empty, so the parallels extend to the date line. There are two exceptions to the parallel limits. No parallel extends beyond the map longitude limits, and exceptions to the parallel limits for selected parallels are allowed (see above).

PLineLocation — Grid parallel interval or specific locations
15 (default) | scalar | vector

Grid parallel interval or specific locations — Establishes the interval between displayed grid parallels. When a scalar interval is entered in the map axes PLineLocation, parallels are displayed, starting at 0º latitude and repeating every interval in both directions, which by default is 15º. Alternatively,

you can enter a vector of latitudes, in which case a parallel is displayed for each element of the vector.

Properties That Control Grid Labeling

FontAngle — Select italic or normal font for all grid labels
`'normal'` (default) | `'italic'` | `'oblique'`

Select italic or normal font for all grid labels — Selects the character slant for all displayed grid labels. `'normal'` specifies nonitalic font. `'italic'` and `'oblique'` specify italic font.

FontColor — Text color for all grid labels
`'black'` (default) | RGB triplet | hexadecimal color code | `'r'` | `'g'` | `'b'` | ...
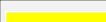
Text color for all grid labels — Sets the color of all displayed grid labels. You can specify an RGB triplet, a hexadecimal color code, a color name, or a short name.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (`#`) followed by three or six hexadecimal digi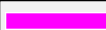ts, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`'red'`	`'r'`	`[1 0 0]`	`'#FF0000'`	
`'green'`	`'g'`	`[0 1 0]`	`'#00FF00'`	
`'blue'`	`'b'`	`[0 0 1]`	`'#0000FF'`	
`'cyan'`	`'c'`	`[0 1 1]`	`'#00FFFF'`	
`'magenta'`	`'m'`	`[1 0 1]`	`'#FF00FF'`	
`'yellow'`	`'y'`	`[1 1 0]`	`'#FFFF00'`	
`'black'`	`'k'`	`[0 0 0]`	`'#000000'`	
`'white'`	`'w'`	`[1 1 1]`	`'#FFFFFF'`	
`'none'`	Not applicable	Not applicable	Not applicable	No color

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
`[0 0.4470 0.7410]`	`'#0072BD'`	
`[0.8500 0.3250 0.0980]`	`'#D95319'`	

RGB Triplet	Hexadecimal Color Code	Appearance
[0.9290 0.6940 0.1250]	'#EDB120'	
[0.4940 0.1840 0.5560]	'#7E2F8E'	
[0.4660 0.6740 0.1880]	'#77AC30'	
[0.3010 0.7450 0.9330]	'#4DBEEE'	
[0.6350 0.0780 0.1840]	'#A2142F'	

FontName — Font family name for all grid labels
'helvetica' (default) | 'courier' | 'symbol' | 'times'

Font family name for all grid labels — Sets the font for all displayed grid labels. To display and print properly, FontName must be a font that your system supports.

FontSize — Font size
0 (default) | scalar

Font size — An integer specifying the font size to use for all displayed grid labels, in units specified by the FontUnits property. The default point size is 9.

FontUnits — Units used to interpret the FontSize property
'points' (default) | 'normalized' | 'inches' | 'centimeters' | 'pixels'

Units used to interpret the FontSize property — When set to normalized, the toolbox interprets the value of FontSize as a fraction of the height of the axes. For example, a normalized FontSize of 0.1 sets the text characters to a font whose height is one-tenth of the axes' height. The default units ('points') are equal to 1/72 of an inch.

FontWeight — Select bold or normal font
'normal' (default) | 'bold'

Select bold or normal font — The character weight for all displayed grid labels.

LabelFormat — Labeling format for grid
'compass' (default) | 'signed' | 'none'

Labeling format for grid — Specifies the format of the grid labels. If 'compass' is employed (the default), meridian labels are suffixed with an "E" for east and a "W" for west, and parallel labels are suffixed with an "N" for north and an "S" for south. If 'signed' is used, meridian labels are prefixed with a "+" for east and a "-" for west, and parallel labels are suffixed with a "+" for north and a "-" for south. If 'none' is selected, straight latitude and longitude numerical values are employed, so western meridian labels and southern parallel labels will have a "-", but no symbol precedes eastern and northern (positive) labels.

LabelRotation — Label Rotation
'off' (default) | 'on'

Label rotation — Determines whether the meridian and parallel labels are displayed without rotation (the default) or rotated to align to the graticule. This option is not available for the Globe display.

LabelUnits — Specify units and formatting for grid labels
'degrees' (default) | 'dm' | 'dms' | 'radians'

Specify units and formatting for grid labels — The display of meridian and parallel labels is controlled by the map axes `LabelUnits` property, as described in the following table.

LabelUnits value	Label format
`'degrees'`	decimal degrees
`'dm'`	degrees/decimal minutes
`'dms'`	degrees/minutes/decimal seconds
`'radians'`	decimal radians

`LabelUnits` does not have a default of its own; instead it defaults to the value of `AngleUnits` at the time the map axes is constructed, which itself defaults to degrees. Although you can specify `'dm'` and `'dms'` for `LabelUnits`, these values are not accepted when setting `AngleUnits`.

MeridianLabel — Toggle display of meridian labels
`'off'` (default) | `'on'`

Toggle display of meridian labels — Specifies whether the meridian labels are visible or not.

MLabelLocation — Specify meridians for labeling
scalar | vector

Specify meridians for labeling — Meridian labels need not coincide with the displayed meridian lines. Labels are displayed at intervals if a scalar in the map axes `MLabelLocation` is entered, starting at the prime meridian and repeating at every interval in both directions. If a vector of longitudes is entered, labels are displayed at those meridians. The default locations coincide with the displayed meridian lines, as specified in the `MLineLocation` property.

MLabelParallel — Specify parallel for meridian label placement
`'north'` (default) | `'south'` | `'equator'` | scalar

Specify parallel for meridian label placement — Specifies the latitude location of the displayed meridian labels. If a scalar latitude is specified, all meridian labels are displayed at that latitude. If `'north'` is specified, the maximum of the `MapLatLimit` is used; if `'south'` is specified, the minimum of the `MapLatLimit` is used. If `'equator'` is specified, a latitude of 0º is used.

MLabelRound — Specify significant digits for meridian labels
0 (default) | integer scalar

Specify significant digits for meridian labels — Specifies to which power of ten the displayed labels are rounded. For example, if `MLabelRound` is -1, labels are displayed down to the *tenths*. The default value of `MLabelRound` is 0; that is, displayed labels have no decimal places, being rounded to the *ones* column (10^0).

ParallelLabel — Toggle display of parallel labels
`'off'` (default) | `'on'`

Toggle display of parallel labels — Specifies whether the parallel labels are visible or not.

PLabelLocation — Specify parallels for labeling
scalar | vector

Specify parallels for labeling — Parallel labels need not coincide with the displayed parallel lines. Labels are displayed at intervals if a scalar in the map axes `PLabelLocation` is entered, starting at

the equator and repeating at every interval in both directions. If a vector of latitudes is entered, labels are displayed at those parallels. The default locations coincide with the displayed parallel lines, as specified in the `PLineLocation` property.

PLabelMeridian — Specify meridian for parallel label placement
'west' (default) | 'east' | 'prime' | scalar

Specify meridian for parallel label placement — Specifies the longitude location of the displayed parallel labels. If a longitude is specified, all parallel labels are displayed at that longitude. If 'east' is specified, the maximum of the `MapLonLimit` is used; if 'west' is specified, the minimum of the `MapLonLimit` is used. If 'prime' is specified, a longitude of 0° is used.

PLabelRound — Specify significant digits for parallel labels
0 (default) | integer scalar

Specify significant digits for parallel labels — Specifies to which power of ten the displayed labels are rounded. For example, if `PLabelRound` is -1, labels are displayed down to the tenths. The default value of `PLabelRound` is 0; that is, displayed labels have no decimal places, being rounded to the ones column (10^0).

See Also
axesm | axes | gcm | getm | setm

Topics
"Introduction to Mapping Graphics"
"The Map Axes"
"The Map Frame"
"Map Limit Properties"
"The Map Grid"

Introduced before R2006a

axesscale

Resize axes for equivalent scale

Syntax

```
axesscale
axesscale(hbase)
axesscale(hbase,hother)
```

Description

`axesscale` resizes all axes in the current figure to have the same scale as the current axes (`gca`). In this context, scale means the relationship between axes *x*- and *y*-coordinates and figure and paper coordinates. When `axesscale` is used, a unit of length in *x* and *y* is printed and displayed at the same size in all the affected axes. The `XLimMode` and `YLimMode` of the axes are set to `'manual'` to prevent autoscaling from changing the scale.

`axesscale(hbase)` uses the axes `hbase` as the reference axes, and rescales the other axes in the current figure.

`axesscale(hbase,hother)` uses the axes `hbase` as the base axes, and rescales only the axes in `hother`.

Examples

Display Multiple Regions with a Common Scale

Display the conterminous United States, Alaska, and Hawaii in separate axes in the same figure, with a common scale.

Read state names and coordinates. Extract Alaska and Hawaii.

```
states = shaperead('usastatehi.shp', 'UseGeoCoords', true);
statenames = {states.Name};
alaska = states(strcmp('Alaska', statenames));
hawaii = states(strcmp('Hawaii', statenames));
```

Create a figure for the conterminous states.

```
f = figure;
hconus = usamap('conus');
geoshow(states, 'FaceColor', [0.5 1 0.5]);
load conus gtlakelat gtlakelon
geoshow(gtlakelat, gtlakelon,...
    'DisplayType', 'polygon', 'FaceColor', 'cyan')
framem off; gridm off; mlabel off; plabel off
```

Display Alaska and Hawaii on different axes. For now, the axes overlap.

```
halaska = axes('Parent',f);
usamap('alaska')
geoshow(alaska, 'FaceColor', [0.5 1 0.5]);
framem off; gridm off; mlabel off; plabel off

hhawaii = axes('Parent',f);
usamap('hawaii')
geoshow(hawaii, 'FaceColor', [0.5 1 0.5]);
framem off; gridm off; mlabel off; plabel off
```
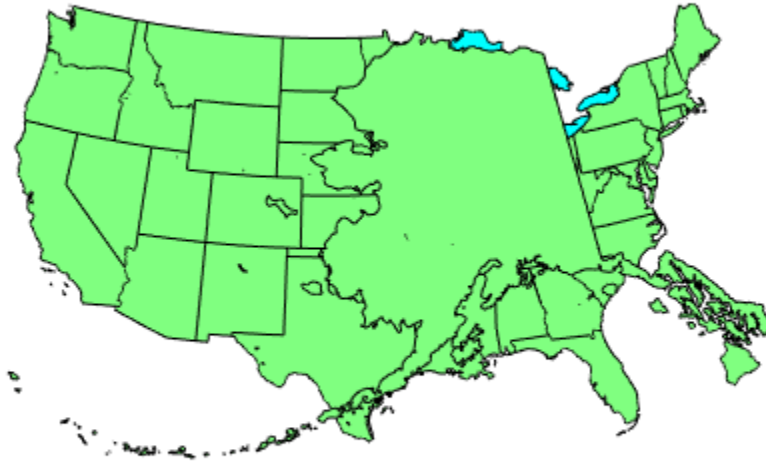
Arrange the axes so they do not overlap. However, this changes the scale of the axes.

```
set(hconus, 'Position',[0.1   0.35 0.85 0.6])
set(halaska,'Position',[0.02  0.08 0.2  0.2])
set(hhawaii,'Position',[0.5   0.1  0.2  0.2])
```
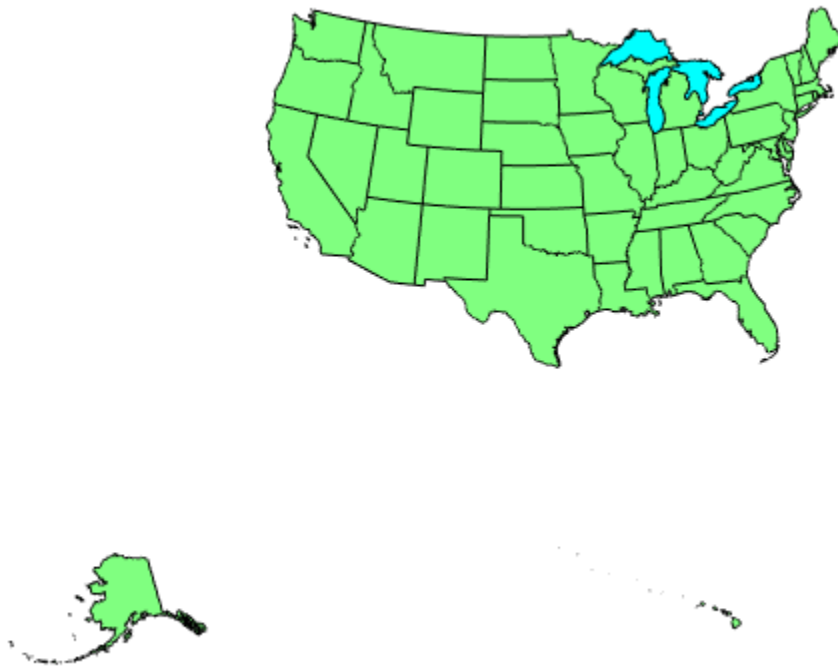
Resize the Alaska and Hawaii axes based on the size of the conterminous United States.

```
axesscale(hconus)
```
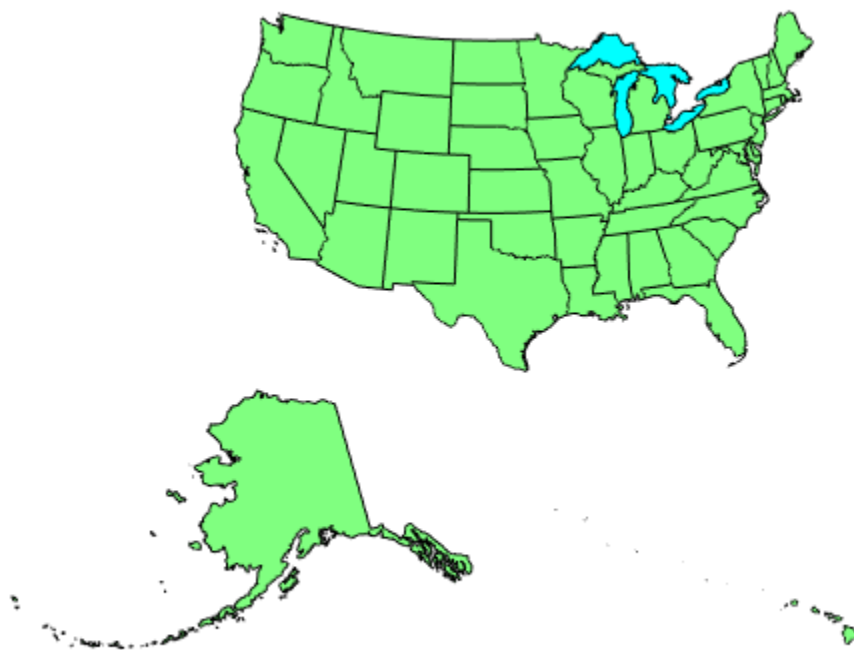
Limitations

The equivalence of scales holds only as long as no commands are issued that can change the scale of one of the axes. For example, changing the units of the ellipsoid or the scale factor in one of the axes would change the scale.

Tips

To ensure the same map scale between axes, use the same ellipsoid and scale factors.

See Also
paperscale

Introduced before R2006a

azimuth

Azimuth between points on sphere or ellipsoid

Syntax

```
az = azimuth(lat1,lon1,lat2,lon2)
az = azimuth(lat1,lon1,lat2,lon2,ellipsoid)
az = azimuth(lat1,lon1,lat2,lon2,units)
az = azimuth(lat1,lon1,lat2,lon2,ellipsoid,units)
az = azimuth(track,...)
```

Description

`az = azimuth(lat1,lon1,lat2,lon2)` calculates the great circle azimuth from point 1 to point 2, for pairs of points on the surface of a sphere. The input latitudes and longitudes can be scalars or arrays of matching size. If you use a combination of scalar and array inputs, the scalar inputs will be automatically expanded to match the size of the arrays. The function measures azimuths clockwise from north and expresses them in degrees or radians.

`az = azimuth(lat1,lon1,lat2,lon2,ellipsoid)` computes the azimuth assuming that the points lie on the ellipsoid defined by the input `ellipsoid`. `ellipsoid` is a `referenceSphere`, `referenceEllipsoid`, or `oblateSpheroid` object, or a vector of the form `[semimajor_axis eccentricity]`. The default ellipsoid is a unit sphere.

`az = azimuth(lat1,lon1,lat2,lon2,units)` uses the input `units` to define the angle units of `az` and the latitude-longitude coordinates. Use `'degrees'` (the default value), in the range from 0 to 360, or `'radians'`, in the range from 0 to 2*pi.

`az = azimuth(lat1,lon1,lat2,lon2,ellipsoid,units)` specifies both the `ellipsoid` vector and the units of `az`.

`az = azimuth(track,...)` uses the input `track` to specify either a great circle or a rhumb line azimuth calculation. Enter `'gc'` for the `track` (the default value), to obtain great circle azimuths for a sphere or geodesic azimuths for an ellipsoid. (Hint to remember name: the letters "g" and "c" are in both great circle and geodesic.) Enter `'rh'` for the `track` to obtain rhumb line azimuths for either a sphere or an ellipsoid.

Examples

Find the azimuth between two points on the same parallel, for example, (10ºN, 10ºE) and (10ºN, 40ºE). The azimuth between two points depends on the *track* value selected.

```
% Try the 'gc' track value.
az = azimuth('gc',10,10,10,40)

% Compare to the result obtained from the 'rh' track value.
az = azimuth('rh',10,10,10,40)
```

Find the azimuth between two points on the same meridian, say (10ºN, 10ºE) and (40ºN, 10ºE):

```
% Try the 'gc' track .
az = azimuth(10,10,40,10)

% Compare to the 'rh' track .
az = azimuth('rh',10,10,40,10)
```

Rhumb lines and great circles coincide along meridians and the Equator. The azimuths are the same because the paths coincide.

More About

Azimuth

An *azimuth* is the angle at which a smooth curve crosses a meridian, taken clockwise from north. The North Pole has an azimuth of 0º from every other point on the globe. You can calculate azimuths for great circles or rhumb lines.

Geodesic

A *geodesic* is the shortest distance between two points on a curved surface, such as an ellipsoid.

Great Circle

A *great circle* is a type of geodesic that lies on a sphere. It is the intersection of the surface of a sphere with a plane passing through the center of the sphere. For great circles, the azimuth is calculated at the starting point of the great circle path, where it crosses the meridian. In general, the azimuth along a great circle is not constant. For more information, see "Great Circles".

Rhumb Line

A *rhumb line* is a curve that crosses each meridian at the same angle. For rhumb lines, the azimuth is the *constant* angle between true north and the entire rhumb line passing through the two points. For more information, see "Rhumb Lines".

Algorithms

Azimuths over Long Geodesics

Azimuth calculations for geodesics degrade slowly with increasing distance and can break down for points that are nearly antipodal or for points close to the Equator. In addition, for calculations on an ellipsoid, there is a small but finite input space. This space consists of pairs of locations in which both points are nearly antipodal *and* both points fall close to (but not precisely on) the Equator. In such cases, you will receive a warning and `az` will be set to NaN for the "problem pairs."

Eccentricity

Geodesic azimuths on an ellipsoid are valid only for small eccentricities typical of the Earth (for example, 0.08 or less).

Alternatives

If you are calculating both the distance and the azimuth, you can call just the `distance` function. The function returns the azimuth as the second output argument. It is unnecessary to call `azimuth` separately.

See Also

distance | elevation | reckon | track | track1 | track2

Introduced before R2006a

boundImageSize

Bound size of raster map

Syntax

```
mapBound = boundImageSize(mapRequest,imageLength)
```

Description

`mapBound = boundImageSize(mapRequest,imageLength)` sets the bounds of the raster map `mapRequest` based on `imageLength`, the length in pixels for the row (`ImageHeight`) or column (`ImageWidth`) dimension.

Examples

Read and Display VMAP0 Basic Layer for Entire Globe

Read the VMAP0 basic layer for the entire globe.

```
vmap0 = wmsfind('vmap0.tiles','SearchField','serverurl');
vmap0 = wmsupdate(vmap0);
layer = refine(vmap0,'basic');
request = WMSMapRequest(layer);
request.Transparent = true;
imageLength = 720;
request = boundImageSize(request,imageLength);
globalImage = getMap(request.Server, request.RequestURL);
```

Display the map. The rendered map has a spatial resolution of 0.5 degrees per cell and an image size of 360-by-720 pixels.

```
figure
worldmap('world')
geoshow(globalImage,request.RasterReference);
title(['VMAP0 ' layer.LayerTitle ' Layer'])
```
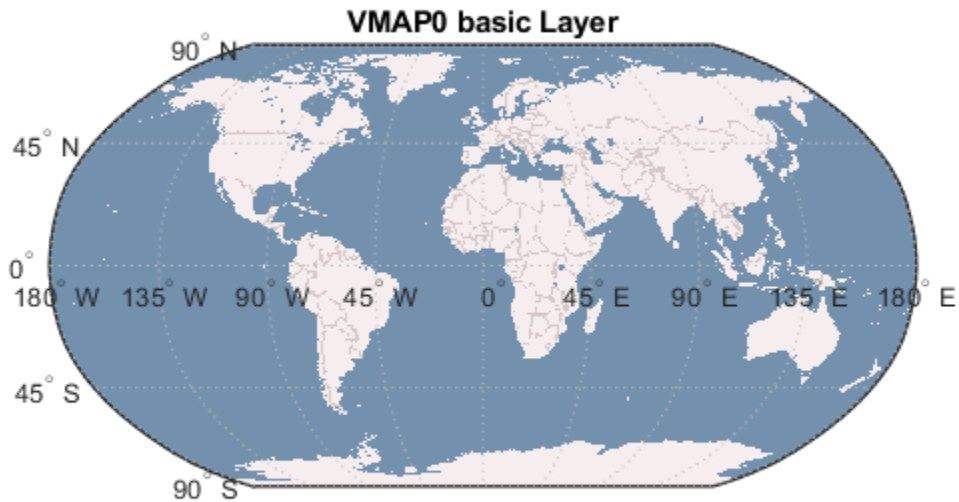
The data used in this example is from Metacarta.

Read and Display Multiple Layers Centered Around London

Read multiple layers centered around London.

```
vmap0 = wmsfind('vmap0.tiles','SearchField','serverurl');
vmap0 = wmsupdate(vmap0);
layers = [ refine(vmap0, 'rail'); refine(vmap0, 'river'); ...
        refine(vmap0, 'priroad'); refine(vmap0, 'secroad'); ...
        refine(vmap0, 'ctylabel'); refine(vmap0, 'basic')];
request = WMSMapRequest(layers);

cities = readgeotable('worldcities.shp');
row = cities.Name == 'London';
london = cities(row,:);

extent = [-.25 .25];
request.Latlim = london.Shape.Latitude + extent;
request.Lonlim = london.Shape.Longitude + extent;

request.Transparent = true;
imageLength = 1024;
request = boundImageSize(request,imageLength);
londonImage = getMap(request.Server,request.RequestURL);
```

Display the map. The rendered map has a spatial extent of 0.5 degrees and an image size of 1024-by-1024 pixels.

```
figure
worldmap(londonImage,request.RasterReference)
geoshow(londonImage,request.RasterReference)
title({'Region Surrounding London, England', ...
   ['with Primary and Secondary Roads, ', ...
    'Rivers, Rails, City Label, and Basic Layers']})
```


Region Surrounding London, England
with Primary and Secondary Roads, Rivers, Rails, City Label, and Basic Layers
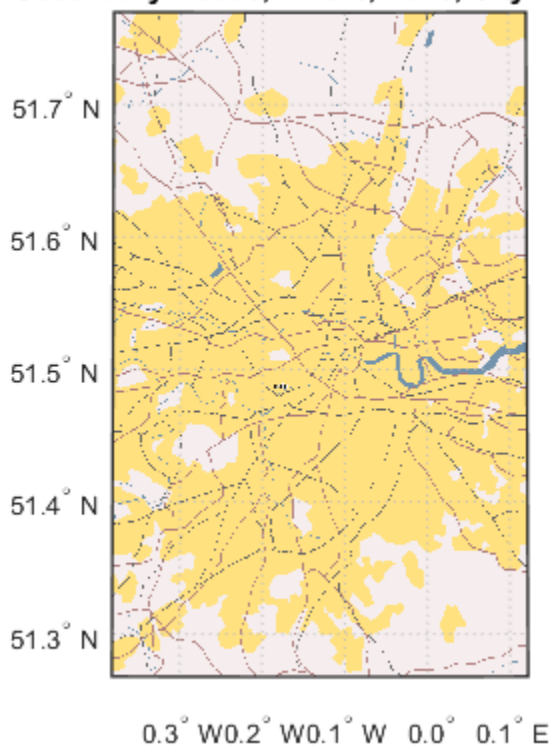
The data used in this example is from Metacarta.

Input Arguments

`mapRequest` — Original Web map service map
WMSMapRequest object

Original Web map service map, specified as a `WMSMapRequest` object.

`imageLength` — Row or column length of input WMS map
positive scalar

Row or column length of input WMS map, specified as a positive scalar. `imageLength` indicates the length in pixels for the row (`ImageHeight`) or column (`ImageWidth`) dimension.

Example: 720

Data Types: `double`

Output Arguments

mapBound — Bound Web map service map
`WMSMapRequest` object

Bound Web map service map, returned as a `WMSMapRequest` object.

Algorithms

The `boundImageSize` function calculates the row or column dimension length by using the aspect ratio of the `Latlim` and `Lonlim` properties or the aspect ratio of the `XLim` and `YLim` properties of , if they are set.

`boundImageSize` measures image dimensions in geographic or map coordinates. The function sets the longest image dimension to `imageLength`, and the shortest to the nearest integer value that preserves the aspect ratio, without changing the coordinate limits. The maximum value of the `MaximumHeight` and `MaximumWidth` properties becomes the maximum value of `imageLength`.

See Also

Introduced in R2009b

bufferm

Buffer zones for latitude-longitude polygons

Syntax

```
[latb,lonb] = bufferm(lat,lon,bufwidth)
[latb,lonb] = bufferm(lat,lon,bufwidth,direction)
[latb,lonb] = bufferm(lat,lon,bufwidth,direction,npts)
```

Description

`[latb,lonb] = bufferm(lat,lon,bufwidth)` computes the buffer zone around a line or polygon. If the vectors `lat` and `lon`, in units of degrees, define a line, then `latb` and `lonb` define a polygon that contains all the points that fall within a certain distance, `bufwidth`, of the line. `bufwidth` is a scalar specified in degrees of arc along the surface. If the vectors `lat` and `lon` define a polygon, then `latb` and `lonb` define a region that contains all the points exterior to the polygon that fall within `bufwidth` of the polygon.

`[latb,lonb] = bufferm(lat,lon,bufwidth,direction)` where `direction` specifies whether the buffer zone is inside (`'in'`) or outside (`'out'`) of the polygon. A third option, `'outPlusInterior'`, returns the union of an exterior buffer (as would be computed using `'out'`) with the interior of the polygon. If you do not supply a `direction` value, `bufferm` uses `'out'` as the default and returns a buffer zone outside the polygon. If you supply `'in'` as the `direction` value, `bufferm` returns a buffer zone inside the polygon. If you are finding the buffer zone around a line, `'out'` is the only valid option.

`[latb,lonb] = bufferm(lat,lon,bufwidth,direction,npts)` controls the number of points used to construct circles about the vertices of the polygon. A larger number of points produces smoother buffers, but requires more time. If `npts` is omitted, 13 points per circle are used.

Examples

Display Buffer Zones Inside and Outside the Great Lakes

Display a simplified version of the five polygons that represent the Great Lakes.

```
load conus
tol = 0.05;
[latr, lonr] = reducem(gtlakelat, gtlakelon, tol);
figure('Color','w')
ax = usamap({'MN','NY'});
setm(ax,'MLabelLocation',5)
geoshow(latr, lonr, 'DisplayType', 'polygon', ...
   'FaceColor', 'blue')
title('Great Lakes')
```
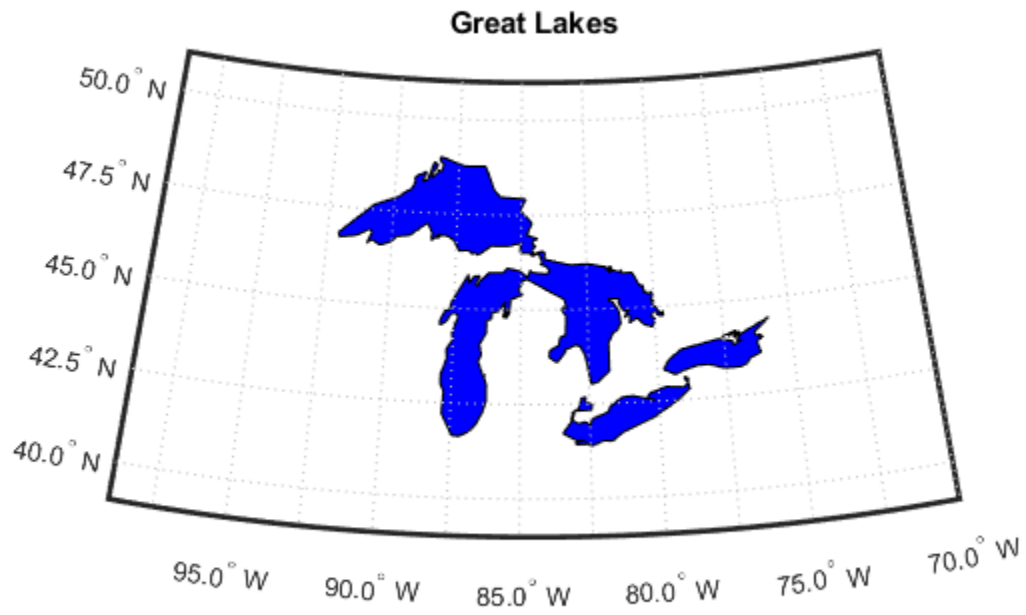
Set the buffer width and display a buffer zone outside the lakes.

```
figure;
bufwidth = 1;
[latb, lonb] = bufferm(latr, lonr, bufwidth);
geoshow(latb, lonb, 'DisplayType', 'polygon', ...
    'FaceColor', 'yellow')
title('Exterior Buffer Zone')
```

Exterior Buffer Zone
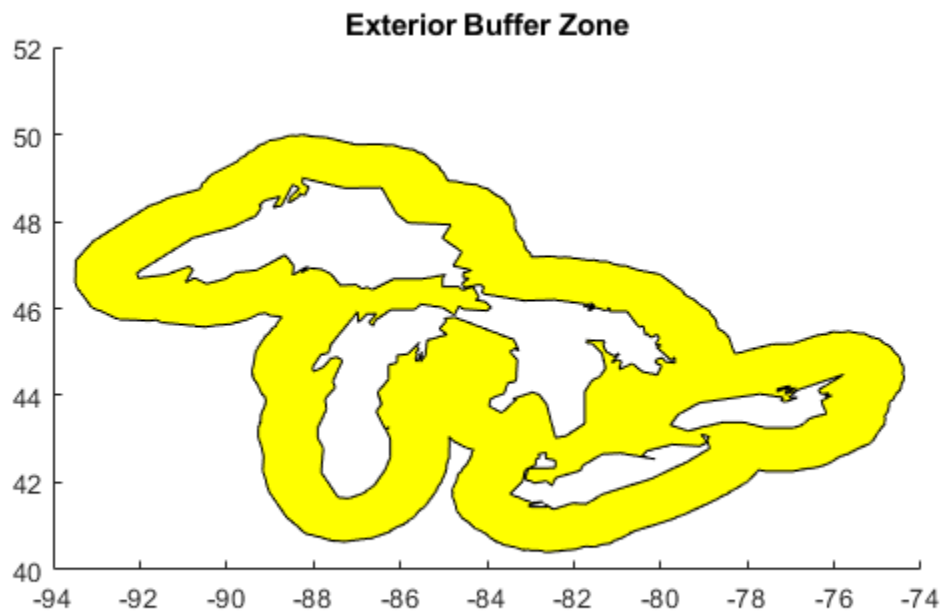
Display a buffer zone inside the polygon.

```
figure;
[lati, loni] = bufferm(latr, lonr, 0.3*bufwidth, 'in');
geoshow(lati, loni, 'DisplayType', 'polygon', ...
   'FaceColor', 'magenta')
title('Interior Buffer Zone')
```
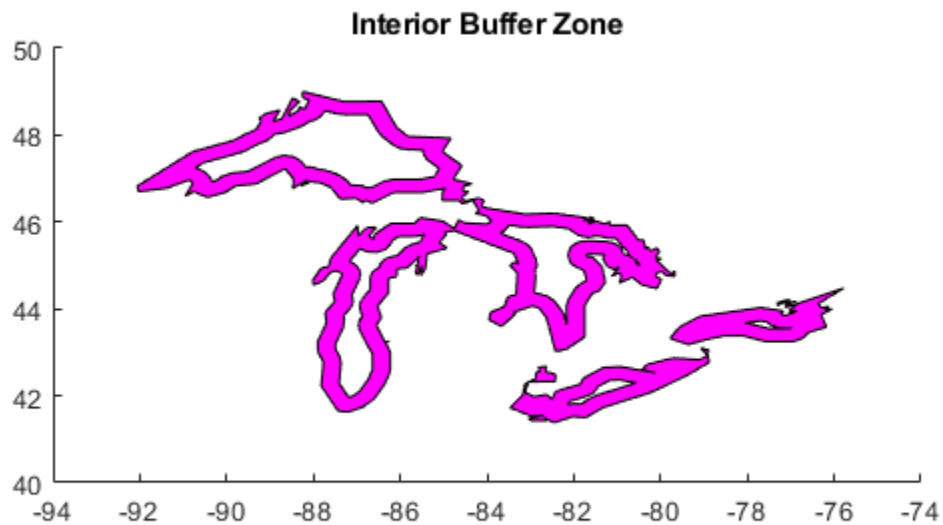
Display the Great Lakes with interior and exterior buffer zones on a backdrop of neighboring states.

```
figure('Color','w')
ax = usamap({'MN','NY'});
setm(ax,'MLabelLocation',5)
geoshow(latb, lonb, 'DisplayType', 'polygon', 'FaceColor', 'yellow')
geoshow(latr, lonr, 'DisplayType', 'polygon', 'FaceColor', 'blue')
geoshow(lati, loni, 'DisplayType', 'polygon', 'FaceColor', 'magenta')
geoshow(uslat, uslon)
geoshow(statelat, statelon)
title('Great Lakes and Buffer Zones with Neighboring States')
```

Great Lakes and Buffer Zones with Neighboring States
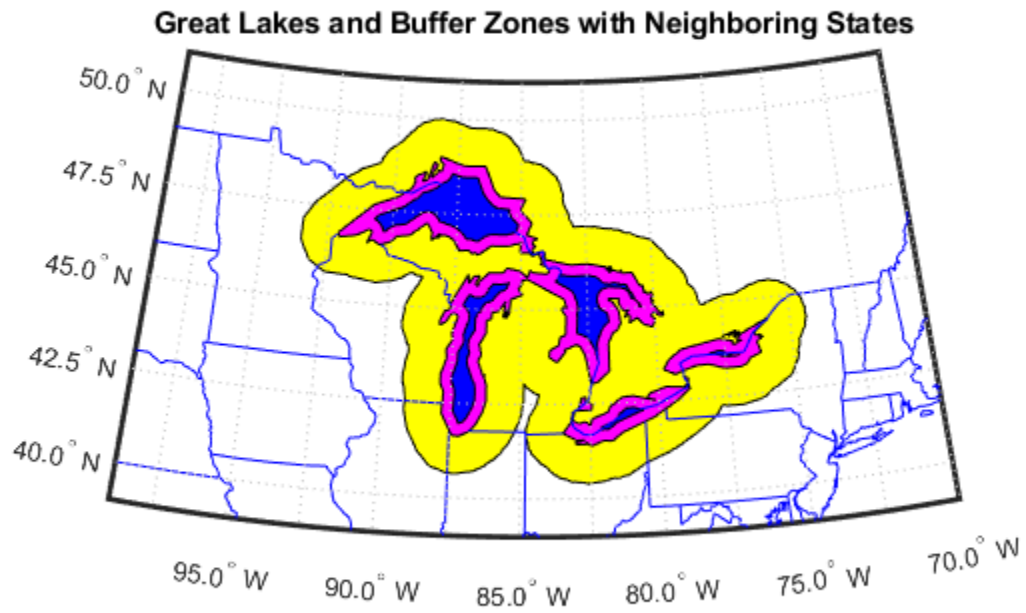
Use the `'outPlusInterior'` option.

```
bufWidth = 0.5;
[latz, lonz] = bufferm(latr, lonr, bufWidth,'outPlusInterior');
figure
geoshow(latz,lonz,'DisplayType','polygon','FaceColor','yellow')
title('Exterior Buffer Zone including Polygon Interior');
```

Exterior Buffer Zone including Polygon Interior
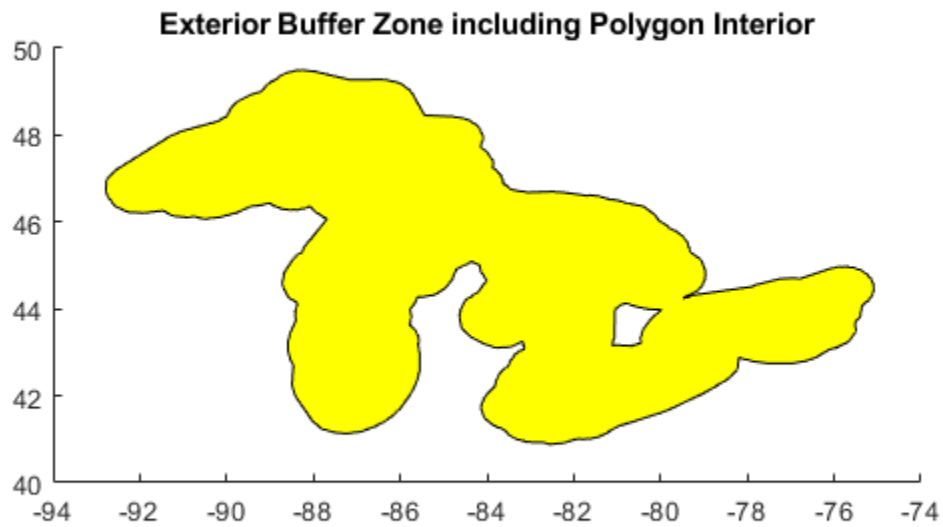
Tips

Close all polygons before processing them with `bufferm`. If a polygon is not closed, `bufferm` assumes it is a line.

See Also

Functions
`polyshape`

Topics
"Create and Display Polygons"

Introduced before R2006a

bufgeoquad

Expand limits of geographic quadrangle

Syntax

```
[latlim,lonlim] = bufgeoquad(latlim,lonlim,buflat,buflon)
```

Description

`[latlim,lonlim] = bufgeoquad(latlim,lonlim,buflat,buflon)` returns an expanded version of the geographic quadrangle defined by `latlim` and `lonlim`.

Examples

Bounding Quadrangle for U.S.

Bounding quadrangle for the Conterminous United States, buffered 2 degrees to the north and south and 3 degrees to the east and west.

Load data and expand the limits of the quadrangle.

```
conus = load('conus.mat');
[latlim, lonlim] = geoquadline(conus.uslat,conus.uslon);
[latlim,lonlim] = bufgeoquad(latlim,lonlim,2,3)

latlim =

   23.1200   51.3800


lonlim =

 -127.7200  -63.9700
```

Input Arguments

`latlim` — Latitude limits
1-by-2 vector

Latitude limits of a geographic quadrangle, specified as a 1-by-2 vector of the form `[southern_limit northern_limit]`, with latitudes in degrees. The two elements must be in ascending order, and lie in the closed interval [-90 90].

Data Types: `single` | `double`

`lonlim` — Longitude limits
1-by-2 vector

Longitude limits of a geographic quadrangle, specified as a 1-by-2 vector of the form `[western_limit eastern_limit]`, with longitudes in degrees. The two limits need not be in numerical ascending order.

Data Types: `single` | `double`

buflat — Latitude buffer size
nonnegative scalar

Latitude buffer size, specified as a nonnegative scalar, in units of degrees.

Data Types: `double`

buflon — Longitude buffer size
nonnegative scalar

Longitude buffer size, specified as a nonnegative scalar, in units of degrees.

Data Types: `double`

Output Arguments

latlim — Latitude limits
1–by-2 vector

Latitude limits of a geographic quadrangle, returned as a 1-by-2 vector of the form `[southern_limit northern_limit]`, in units of degrees. The elements are in ascending order, and both lie in the closed interval [-90 90].

lonlim — Longitude limits
1–by-2 vector

Longitude limits of a geographic quadrangle, returned as a 1-by-2 vector of the form `[western_limit eastern_limit]`, in units of degrees. The limits are wrapped to the interval [-180 180]. They are not necessarily in numerical ascending order.

See Also
geoquadpt | geoquadline | outlinegeoquad

Introduced in R2012b

camheading

Set or query heading angle of camera for geographic globe

Syntax

```
camheading(g,heading)
camheading(g,'auto')
camheading(g,'manual')

outHeading = camheading(g)
outHeading = camheading( ___ )
```

Description

Set Heading and Mode

camheading(g,heading) sets the heading angle of the camera for the specified geographic globe. Setting the heading angle shifts the camera left or right. For more information about how camera rotations affect your view of the globe, see "How Camera Orientation Affects Globe View" on page 1-133.
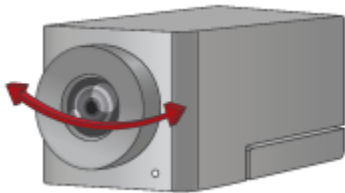
camheading(g,'auto') sets the camera heading to automatic mode, enabling the geographic globe to determine the heading angle based on plotted data. The mode defaults to automatic when you create a geographic globe. If you interact with the globe using your mouse, then the mode switches to automatic.

camheading(g,'manual') sets the camera heading to manual mode, specifying that the geographic globe preserve the heading angle when the plotted data changes. If you change the heading angle using the camheading function, then the mode switches to manual.

Query Heading

outHeading = camheading(g) returns the heading angle of the camera.

outHeading = camheading(___) sets the heading angle or mode and then returns the heading angle of the camera. You can return the heading angle using any combination of input arguments from the previous syntaxes.

Examples

Change Heading Angle of Camera

Create a geographic globe. Position the camera near Mount Kilimanjaro by specifying a latitude, longitude, and ellipsoidal height. Set the pitch angle to 0 degrees, so that the camera points across the summit.

```
uif = uifigure;
g = geoglobe(uif);
campos(g,-3.1519,37.3561,5500)
campitch(g,0)
```
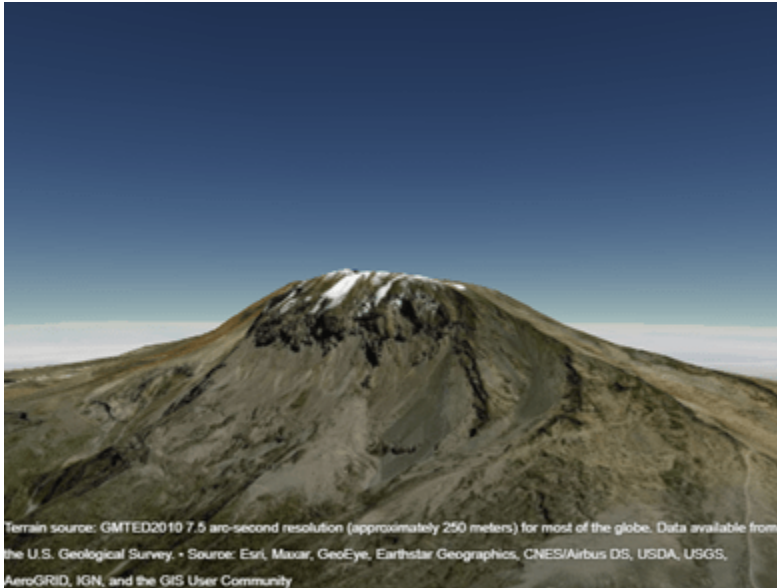

By default, the heading angle is 360 degrees, which is equivalent to a heading angle of 0 degrees. Shift the camera to the right by changing the heading angle to 15 degrees.

```
camheading(g,15)
```
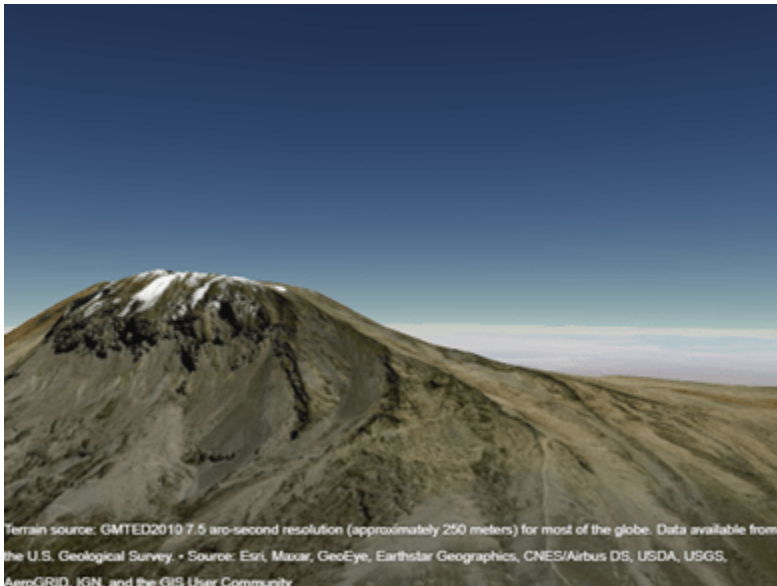
Animate Changes to Heading Angle

Create a geographic globe. Position the camera at the top of Mount Washington by specifying a latitude, longitude, and ellipsoidal height. Change the pitch angle so that the camera faces the horizon instead straight down.

```
uif = uifigure;
g = geoglobe(uif);
campos(g,44.2700,-71.3038,2000)
campitch(g,0)
```
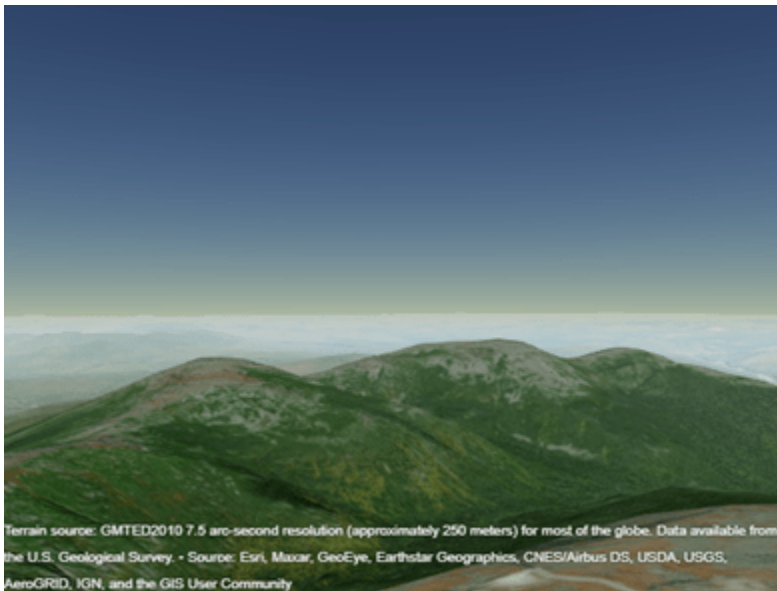

Animate the view by incrementally changing the heading angle. As the heading angle increases, the camera view shifts to the right.

```
for heading = 0:5:180
    camheading(g,heading)
    drawnow
end
```
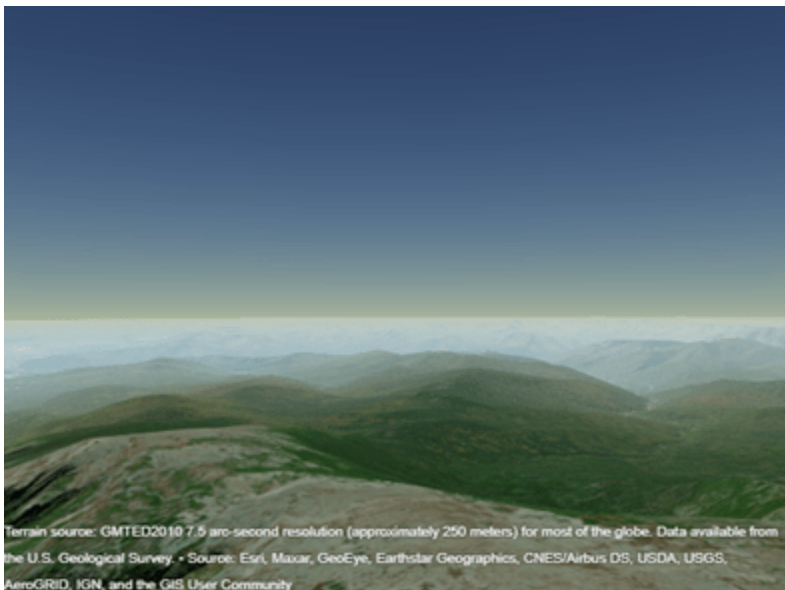
Get Information About Camera View

Get the position and the heading, pitch, and roll angles of the camera. Use this information to control the view of a different geographic globe or to automate navigation.

Create a geographic globe. Navigate to an area of interest using your mouse or gestures. For this example, navigate to an area around Hawaii.

```
uif = uifigure;
g = geoglobe(uif);
```


Query the latitude, longitude, and ellipsoidal height of the camera, and assign each to a variable.

```
[camlat,camlon,camh] = campos(g)

camlat =

    18.1781


camlon =

 -155.9297


camh =

    6.6664e+04
```

Query the heading, pitch, and roll angles of the camera, and assign each to a variable.

```
heading = camheading(g)
pitch = campitch(g)
roll = camroll(g)

heading =

    16.7613


pitch =

  -24.1507


roll =

   359.9977
```

Use these values to control the view of a different geographic globe. For example, create a new geographic globe and programmatically set the view.

```
uif2 = uifigure;
g2 = geoglobe(uif2);
campos(g2,camlat,camlon,camh)
camheading(g2,heading)
campitch(g2,pitch)
camroll(g2,roll)
```

Preserve Camera View

Preserve the position and the heading, pitch, and roll angles of the camera by setting the camera modes to manual. If you do not set the camera modes to manual, then the camera view resets when you plot new data.

Import a sample route along roads in Massachusetts using the `gpxread` function. Create a geographic globe with a road map and no terrain data. Preserve the basemap and terrain settings by using the `hold` function. Then, navigate to an area near Eastern Massachusetts using your mouse.

```
track = gpxread('sample_tracks.gpx','Index',2);
lat = track.Latitude;
lon = track.Longitude;
height = track.Elevation;

uif = uifigure;
g = geoglobe(uif,'Basemap','streets','Terrain','none');
hold(g,'on')
```


Set the camera modes to manual and plot the data. Note that the camera position does not change.

```
campos(g,'manual')
camheight(g,'manual')
camheading(g,'manual')
campitch(g,'manual')
camroll(g,'manual')
geoplot3(g,lat,lon,height,'LineWidth',3)
```

Input Arguments

g — Geographic globe
GeographicGlobe object

Geographic globe, specified as a GeographicGlobe object.[1]

heading — Heading angle of camera
360 (default) | scalar

Heading angle of the camera, specified as a scalar value in the range [–360, 360] degrees.

More About

How Camera Orientation Affects Globe View

The values of the camera heading, pitch, and roll angles affect your view of a geographic globe. Mapping Toolbox references these values to the globe using a north-east-down (NED) coordinate system. As a result, when the heading, pitch, and roll angles of the camera are zero, the camera sits on a plane that is parallel to the tangent plane of the globe at the current latitude and longitude. For more information about NED coordinate systems, see "Choose a 3-D Coordinate System".

---

1.   Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

Change your view of a geographic globe by changing the heading, pitch, and roll angles of the camera:

- Heading — Rotate the camera about its *z*-axis, which shifts the view left or right. Move the view to the right by increasing the heading angle.
- Pitch — Rotate the camera about its *y*-axis, which tilts the view up or down. Tilt the view up by increasing the pitch angle.
- Roll — Rotate the camera about its *x*-axis, which spins the camera around its lens. Spin the view counterclockwise by increasing the roll angle.

Tips

- When the pitch angle is near –90 (the default) or 90 degrees, the camera loses one rotational degree of freedom. As a result, when you change the roll angle, the heading angle may change instead. This phenomenon is called gimbal lock. To avoid the effects of gimbal lock, use the `camheading` function instead of the `camroll` function.

See Also

Functions
geoglobe | campos | camroll | camheight | campitch

Topics
"Visualize Aircraft Line-of-Sight Over Terrain"

"Visualize UAV Flight Path on 2-D and 3-D Maps"

Introduced in R2020b

camheight

Set or query height of camera for geographic globe

Syntax

```
camheight(g,height)
camheight(g,'auto')
camheight(g,'manual')

heightOut = camheight(g)
heightOut = camheight( ___ )
```

Description

Set Height and Mode

`camheight(g,height)` sets the ellipsoidal height of the camera for the specified geographic globe.

`camheight(g,'auto')` sets the camera height to automatic mode, enabling the geographic globe to determine the height of the camera based on the plotted data. The mode defaults to automatic when you create a geographic globe. If you change the camera height using your mouse, then the mode switches to automatic.

`camheight(g,'manual')` sets the camera height to manual mode, specifying that the geographic globe preserve the height of the camera when the plotted data changes. If you change the camera height using the `camheight` function, then the mode switches to manual.

Query Height

`heightOut = camheight(g)` returns the ellipsoidal height of the camera.

`heightOut = camheight( ___ )` sets the height or mode and then returns the ellipsoidal height of the camera. You can return the camera height using any combination of input arguments from the previous syntaxes.

Examples

Change Camera Height

Create a geographic globe. Specify the latitude and longitude of the Eiffel Tower, and specify a height that is 700 meters above the WGS84 reference ellipsoid. Then, move the camera using the `campos` function.

```
uif = uifigure;
g = geoglobe(uif);

lat = 48.8584;
lon = 2.2945;
h = 700;
campos(g,lat,lon,h)
```

Change only the height of the camera by using the `camheight` function. Increase the camera height to 1500 meters above the WGS84 reference ellipsoid.

```
camheight(g,1500)
```


Animate Changes to Camera Height

Create a geographic globe. Position the camera above Mount Washington by specifying a latitude, longitude, and ellipsoidal height.

```
uif = uifigure;
g = geoglobe(uif);
```

```
height0 = 2500;
campos(g,44.2706,-71.3025,height0)
```
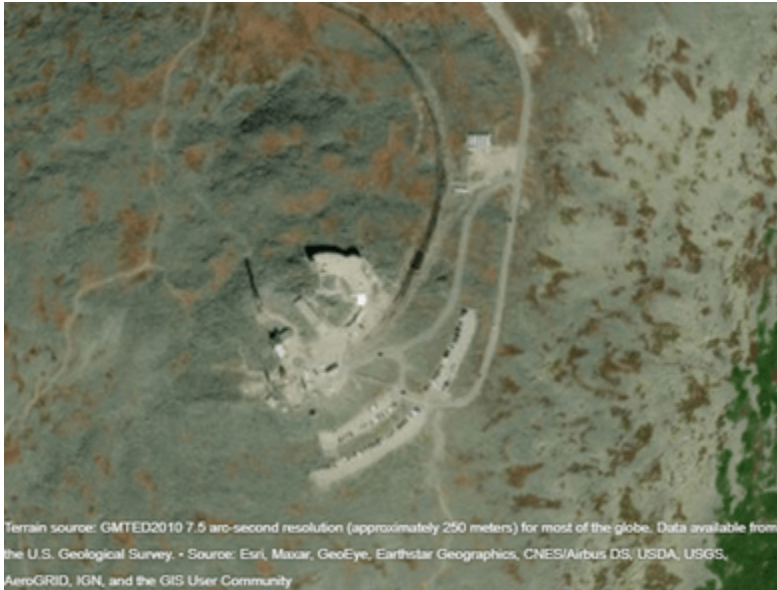

Animate the view by incrementally changing the camera height. As the camera height increases, the view zooms out.

```
for height = 2500:50:5000
    camheight(g,height)
    drawnow
end
```
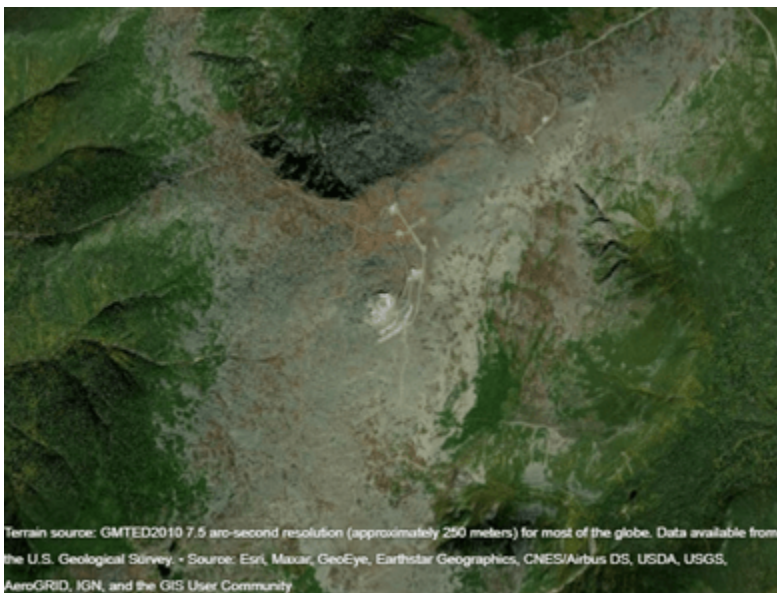
Get Height of Camera

Get the height of the camera. Use this information to control the view of a different geographic globe or to automate navigation.

Create a geographic globe. Zoom in on an area around Spain using your mouse or gestures.

```
uif = uifigure;
g = geoglobe(uif);
```


Query the height of the camera and assign it to a variable.

```
outHeight = camheight(g)
```

```
outHeight =

   1.8803e+06
```

Use this value to control the camera height for a different geographic globe. For example, create a new geographic globe and programmatically set the camera height.

```
uif2 = uifigure;
g2 = geoglobe(uif2);
camheight(g2,outHeight)
```

Preserve Camera View

Preserve the position and the heading, pitch, and roll angles of the camera by setting the camera modes to manual. If you do not set the camera modes to manual, then the camera view resets when you plot new data.

Import a sample route along roads in Massachusetts using the `gpxread` function. Create a geographic globe with a road map and no terrain data. Preserve the basemap and terrain settings by using the `hold` function. Then, navigate to an area near Eastern Massachusetts using your mouse.

```
track = gpxread('sample_tracks.gpx','Index',2);
lat = track.Latitude;
lon = track.Longitude;
height = track.Elevation;

uif = uifigure;
g = geoglobe(uif,'Basemap','streets','Terrain','none');
hold(g,'on')
```


Set the camera modes to manual and plot the data. Note that the camera position does not change.

```
campos(g,'manual')
camheight(g,'manual')
camheading(g,'manual')
campitch(g,'manual')
camroll(g,'manual')
geoplot3(g,lat,lon,height,'LineWidth',3)
```

Convert Between Zoom Level and Camera Height

The strategies you use to programmatically zoom in and out of `GeographicGlobe` and `GeographicAxes` objects are different. For `GeographicGlobe` objects, you specify a camera height using the `camheight` function. For `GeographicAxes` objects, you specify a zoom level using the `ZoomLevel` property or specify latitude and longitude limits using the `geolimits` function. To create `GeographicGlobe` and `GeographicAxes` objects with similar map scales, approximate camera height and zoom level using the `heightToZoomLevel` and `zoomLevelToHeight` local functions (defined here).

You can verify the behavior of the `zoomLevelToHeight` local function by displaying `GeographicAxes` and `GeographicGlobe` objects using comparable magnification levels.

Specify the latitude and longitude of the Sydney Opera House. Create geographic axes with a basemap, map center, and zoom level that allows you to clearly see the building.

```
lat = -33.8572;
lon = 151.2150;
z = 17;
gx = geoaxes('Basemap','satellite','MapCenter',[lat lon],'ZoomLevel',z);
```
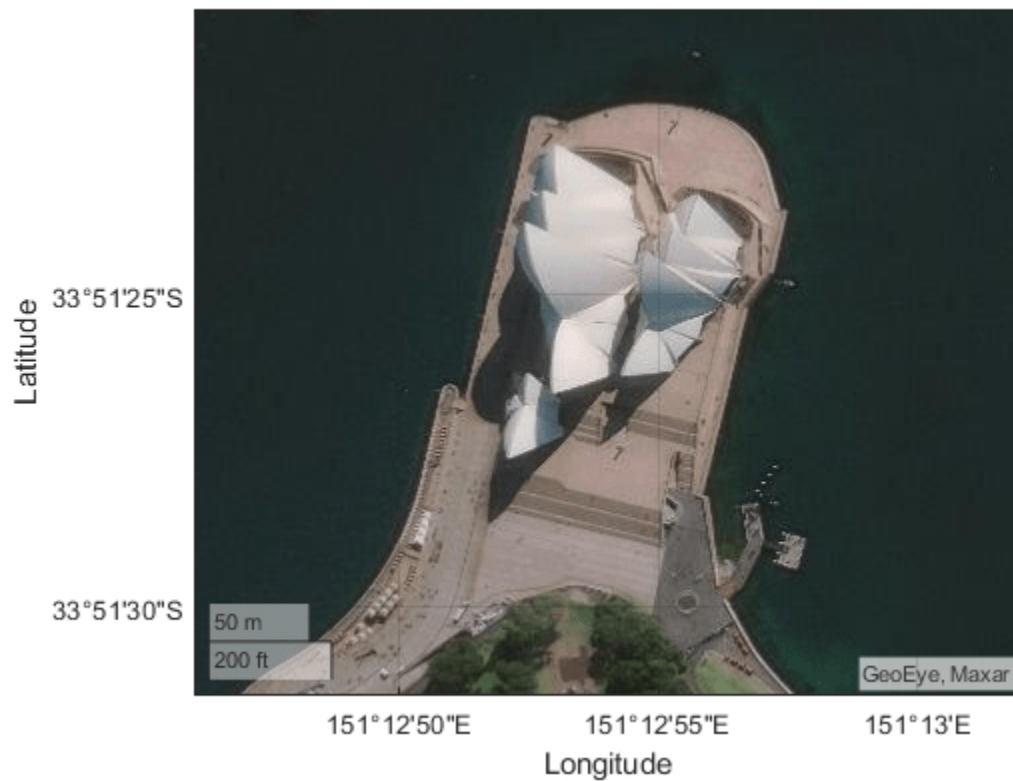
Create a geographic globe. Position the camera above the Sydney Opera House using the `campos` function.

```
uif = uifigure;
g = geoglobe(uif);
campos(g,lat,lon)
```

Calculate an approximate camera height from the zoom level using the `zoomLevelToHeight` local function. Then, set the camera height using the `camheight` function. Note that the geographic axes and geographic globe displays are comparable.

```
h = zoomLevelToHeight(z,lat);
camheight(g,h)
```

Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

To verify the behavior of the `heightToZoomLevel` function, calculate an approximate zoom level from the camera height.

```
z2 = heightToZoomLevel(h,lat)
```

```
z2 = 17
```

Note that `z` and `z2` are equal.

This code defines a local function called `zoomLevelToHeight` that approximates the camera height `h` for a `GeographicGlobe` object using the zoom level `z` and map center latitude `lat` of a `GeographicAxes` object.

```
function h = zoomLevelToHeight(z,lat)
    earthCircumference = 2*pi*6378137;
    h = (earthCircumference*cosd(lat)) / 2^(z-1);
end
```

This code defines a local function called `heightToZoomLevel` that approximates the zoom level `z` for a `GeographicAxes` object using the camera height `h` and latitude `lat` of a `GeographicGlobe` object.

```
function z = heightToZoomLevel(h,lat)
    earthCircumference = 2*pi*6378137;
    z = log2((earthCircumference*cosd(lat)) / h) + 1;
    z = max(0,z);
    z = min(19,z);
end
```

Input Arguments

g — Geographic globe
GeographicGlobe object

Geographic globe, specified as a `GeographicGlobe` object.[2]

height — Ellipsoidal height of camera
15000000 (default) | numeric scalar

Ellipsoidal height of the camera, specified as a numeric scalar in meters. Geographic globe objects use the WGS84 reference ellipsoid. For more information about ellipsoidal height, see "Find Ellipsoidal Height from Orthometric Height".

If you specify the height so that the camera is level with or below the terrain, then the `camheight` function sets the height to a value one meter above the terrain.

See Also

Functions
geoglobe | campitch | campos | camroll | camheading

Topics
"Visualize UAV Flight Path on 2-D and 3-D Maps"
"Visualize Aircraft Line-of-Sight Over Terrain"

Introduced in R2020b

---

2.   Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

campitch

Set or query pitch angle of camera for geographic globe

Syntax

```
campitch(g,pitch)
campitch(g,'auto')
campitch(g,'manual')

outPitch = campitch(g)
outPitch = campitch(___)
```

Description

Set Pitch and Mode

campitch(g,pitch) sets the pitch angle of the camera for the specified geographic globe. Setting the pitch angle tilts the camera up or down. For more information about how camera rotations affect your view of the globe, see "How Camera Orientation Affects Globe View" on page 1-151.
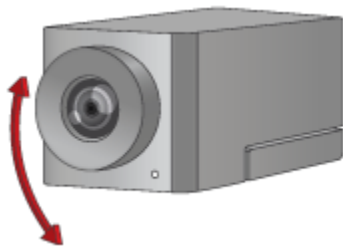
campitch(g,'auto') sets the camera pitch to automatic mode, enabling the geographic globe to determine the pitch angle based on plotted data. The mode defaults to automatic when you create a geographic globe. If you interact with the globe using your mouse, then the mode switches to automatic.

campitch(g,'manual') sets the camera pitch to manual mode, specifying that the geographic globe preserve the pitch angle when the plotted data changes. If you change the pitch angle using the campitch function, then the mode switches to manual.

Query Pitch

outPitch = campitch(g) returns the pitch angle of the camera.

outPitch = campitch(___) sets the pitch angle or mode and then returns the pitch angle of the camera. You can return the pitch angle using any combination of input arguments from the previous syntaxes.

Examples

Change Pitch Angle of Camera

Create a geographic globe. Position the camera near Mount Kilimanjaro by specifying a latitude, longitude, and ellipsoidal height. Set the pitch angle to 0 degrees, so that the camera points across the summit.

```
uif = uifigure;
g = geoglobe(uif);
campos(g,-3.1519,37.3561,5500)
campitch(g,0)
```
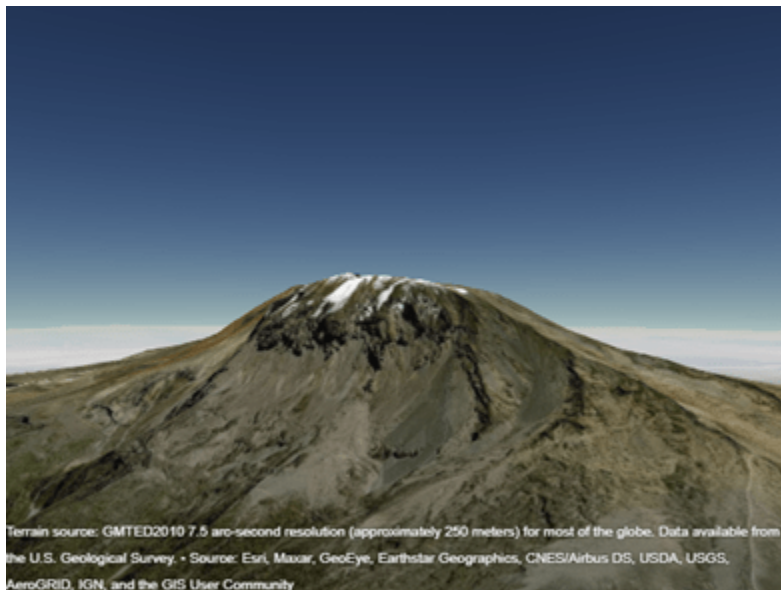

Tilt the camera up by increasing the pitch angle to 15 degrees.

```
campitch(g,15)
```
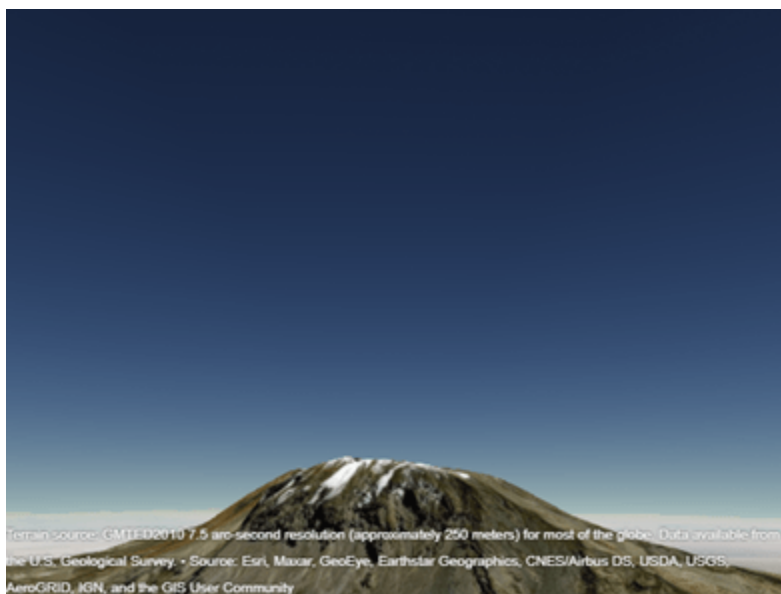
Animate Changes to Pitch Angle

Create a geographic globe. Position the camera near Mount Washington by specifying a latitude, longitude, and ellipsoidal height. Change the heading angle so that the camera faces the mountain. By default, the pitch angle is –90 degrees, so that the camera points at the ground.

```
uif = uifigure;
g = geoglobe(uif);

campos(g,44.2668,-71.3849,1757)
camheading(g,90)
```


Animate the view by incrementally changing the pitch angle. As the pitch angle increases, the camera changes from pointing at the ground to pointing at the sky. After the animation finishes, change the pitch angle to 0 degrees.

```
for pitch = -90:2:90
    campitch(g,pitch)
    drawnow
end
campitch(g,0)
```

Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Get Information About Camera View

Get the position and the heading, pitch, and roll angles of the camera. Use this information to control the view of a different geographic globe or to automate navigation.

Create a geographic globe. Navigate to an area of interest using your mouse or gestures. For this example, navigate to an area around Hawaii.

```
uif = uifigure;
g = geoglobe(uif);
```


Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Query the latitude, longitude, and ellipsoidal height of the camera, and assign each to a variable.

```
[camlat,camlon,camh] = campos(g)

camlat =

   18.1781


camlon =

 -155.9297


camh =

   6.6664e+04
```

Query the heading, pitch, and roll angles of the camera, and assign each to a variable.

```
heading = camheading(g)
pitch = campitch(g)
roll = camroll(g)

heading =

   16.7613


pitch =

  -24.1507


roll =

  359.9977
```

Use these values to control the view of a different geographic globe. For example, create a new geographic globe and programmatically set the view.

```
uif2 = uifigure;
g2 = geoglobe(uif2);
campos(g2,camlat,camlon,camh)
camheading(g2,heading)
campitch(g2,pitch)
camroll(g2,roll)
```

Preserve Camera View

Preserve the position and the heading, pitch, and roll angles of the camera by setting the camera modes to manual. If you do not set the camera modes to manual, then the camera view resets when you plot new data.

Import a sample route along roads in Massachusetts using the `gpxread` function. Create a geographic globe with a road map and no terrain data. Preserve the basemap and terrain settings by using the `hold` function. Then, navigate to an area near Eastern Massachusetts using your mouse.

```
track = gpxread('sample_tracks.gpx','Index',2);
lat = track.Latitude;
lon = track.Longitude;
height = track.Elevation;

uif = uifigure;
g = geoglobe(uif,'Basemap','streets','Terrain','none');
hold(g,'on')
```


Set the camera modes to manual and plot the data. Note that the camera position does not change.

```
campos(g,'manual')
camheight(g,'manual')
camheading(g,'manual')
campitch(g,'manual')
camroll(g,'manual')
geoplot3(g,lat,lon,height,'LineWidth',3)
```

Input Arguments

g — Geographic globe
GeographicGlobe object

Geographic globe, specified as a GeographicGlobe object.[3]

pitch — Pitch angle of camera
–90 (default) | scalar

Pitch angle of the camera, specified as a scalar value in the range [–90, 90] degrees. By default, the pitch angle is –90 degrees, which means that camera points directly toward the surface of the globe.

More About

How Camera Orientation Affects Globe View

The values of the camera heading, pitch, and roll angles affect your view of a geographic globe. Mapping Toolbox references these values to the globe using a north-east-down (NED) coordinate system. As a result, when the heading, pitch, and roll angles of the camera are zero, the camera sits on a plane that is parallel to the tangent plane of the globe at the current latitude and longitude. For more information about NED coordinate systems, see "Choose a 3-D Coordinate System".

---

3.   Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

Change your view of a geographic globe by changing the heading, pitch, and roll angles of the camera:

- Heading — Rotate the camera about its *z*-axis, which shifts the view left or right. Move the view to the right by increasing the heading angle.
- Pitch — Rotate the camera about its *y*-axis, which tilts the view up or down. Tilt the view up by increasing the pitch angle.
- Roll — Rotate the camera about its *x*-axis, which spins the camera around its lens. Spin the view counterclockwise by increasing the roll angle.

Tips

- When the pitch angle is near –90 (the default) or 90 degrees, the camera loses one rotational degree of freedom. As a result, when you change the roll angle, the heading angle may change instead. This phenomenon is called gimbal lock. To avoid the effects of gimbal lock, call the `camheading` function instead of the `camroll` function.

See Also

Functions
geoglobe | campos | camheight | camroll | camheading

Topics
"Visualize Aircraft Line-of-Sight Over Terrain"

"Visualize UAV Flight Path on 2-D and 3-D Maps"

Introduced in R2020b

campos

Set or query position of camera for geographic globe

Syntax

```
campos(g,lat,lon)
campos(g,lat,lon,height)

campos(g,'auto')
campos(g,'manual')

campos(g)
[latOut,lonOut,heightOut] = campos( ___ )
```

Description

Set Position

campos(g,lat,lon) sets the latitude and longitude of the camera for the specified geographic globe.

campos(g,lat,lon,height) sets the latitude, longitude, and ellipsoidal height of the camera. If you want to set only the height of the camera, then use the camheight function instead.

Set Mode

campos(g,'auto') sets the camera position to an automatic mode, enabling the geographic globe to determine the latitude and longitude of the camera based on the plotted data. The mode defaults to automatic when you create a geographic globe. If you change the camera position using your mouse, then the mode switches to automatic. To control the mode for the height of the camera, use the camheight function instead.

campos(g,'manual') sets the camera position to a manual mode, specifying that the geographic globe preserve the latitude and longitude of the camera when the plotted data changes. If you change the camera position using the campos function, then the mode switches to manual.

Query Position

campos(g) displays the latitude, longitude, and ellipsoidal height of the camera as a three-element vector.

[latOut,lonOut,heightOut] = campos(___) sets the position or mode and then returns the latitude, longitude, and height of the camera. You can return the camera position using any of the previous syntaxes.

Examples

Change Camera Position

Create a geographic globe. Specify the latitude and longitude of the Eiffel Tower, and specify a height that is 400 meters above the WGS84 reference ellipsoid. Use these values to move the camera.

```
uif = uifigure;
g = geoglobe(uif);

lat = 48.8584;
lon = 2.2945;
h = 400;
campos(g,lat,lon,h)
```


Get Information About Camera View

Get the position and the heading, pitch, and roll angles of the camera. Use this information to control the view of a different geographic globe or to automate navigation.

Create a geographic globe. Navigate to an area of interest using your mouse or gestures. For this example, navigate to an area around Hawaii.

```
uif = uifigure;
g = geoglobe(uif);
```

Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. · Source: Esri, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Query the latitude, longitude, and ellipsoidal height of the camera, and assign each to a variable.

```
[camlat,camlon,camh] = campos(g)
```

```
camlat =

   18.1781
```

```
camlon =

 -155.9297
```

```
camh =

   6.6664e+04
```

Query the heading, pitch, and roll angles of the camera, and assign each to a variable.

```
heading = camheading(g)
pitch = campitch(g)
roll = camroll(g)
```

```
heading =

   16.7613
```

```
pitch =

  -24.1507
```

```
roll =

   359.9977
```

Use these values to control the view of a different geographic globe. For example, create a new geographic globe and programmatically set the view.

```
uif2 = uifigure;
g2 = geoglobe(uif2);
campos(g2,camlat,camlon,camh)
camheading(g2,heading)
campitch(g2,pitch)
camroll(g2,roll)
```

Preserve Camera View

Preserve the position and the heading, pitch, and roll angles of the camera by setting the camera modes to manual. If you do not set the camera modes to manual, then the camera view resets when you plot new data.

Import a sample route along roads in Massachusetts using the `gpxread` function. Create a geographic globe with a road map and no terrain data. Preserve the basemap and terrain settings by using the `hold` function. Then, navigate to an area near Eastern Massachusetts using your mouse.

```
track = gpxread('sample_tracks.gpx','Index',2);
lat = track.Latitude;
lon = track.Longitude;
height = track.Elevation;

uif = uifigure;
g = geoglobe(uif,'Basemap','streets','Terrain','none');
hold(g,'on')
```


Set the camera modes to manual and plot the data. Note that the camera position does not change.

```
campos(g,'manual')
camheight(g,'manual')
camheading(g,'manual')
```

```
campitch(g,'manual')
camroll(g,'manual')
geoplot3(g,lat,lon,height,'LineWidth',3)
```


Input Arguments

g — Geographic globe
GeographicGlobe object

Geographic globe, specified as a GeographicGlobe object.[4]

lat — Geodetic latitude of camera
0 (default) | numeric scalar

Geodetic latitude of the camera, specified as a numeric scalar in the range [–90, 90] degrees.

lon — Geodetic longitude of camera
0 (default) | numeric scalar

Geodetic longitude of the camera, specified as a numeric scalar in the range [–360, 360].

height — Ellipsoidal height of camera
15000000 (default) | numeric scalar

Ellipsoidal height of the camera, specified as a numeric scalar in meters. Geographic globe objects use the WGS84 reference ellipsoid. For more information about ellipsoidal height, see "Find Ellipsoidal Height from Orthometric Height".

If you specify the height so that the camera is level with or below the terrain, then the campos function sets the height to a value one meter above the terrain.

---

4.   Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

Output Arguments

`latOut` — Geodetic latitude of camera
numeric scalar

Geodetic latitude of the camera, returned as a numeric scalar in degrees.

`lonOut` — Geodetic longitude of camera
numeric scalar

Geodetic longitude of the camera, returned as a numeric scalar in degrees.

`heightOut` — Ellipsoidal height of camera
numeric scalar

Ellipsoidal height of the camera, returned as a numeric scalar in meters. Geographic globe objects use the WGS84 reference ellipsoid. For more information about ellipsoidal height, see "Find Ellipsoidal Height from Orthometric Height".

See Also

Functions
geoglobe | campitch | camheight | camroll | camheading

Topics
"Visualize Aircraft Line-of-Sight Over Terrain"
"Visualize UAV Flight Path on 2-D and 3-D Maps"

Introduced in R2020b

camposm

Set camera position using geographic coordinates

Syntax

```
camposm(lat,long,alt)
[x,y,z] = camposm(lat,long,alt)
```

Description

`camposm(lat,long,alt)` sets the axes `CameraPosition` property of the current map axes to the position specified in geographic coordinates. The inputs `lat` and `long` are assumed to be in the angle units of the current map axes.

`[x,y,z] = camposm(lat,long,alt)` returns the camera position in the projected Cartesian coordinate system.

Examples

Look at northern Australia from a point south and one Earth radius above New Zealand.

```
axesm('globe','galt',0)
gridm('glinestyle','-')
load topo60c
geoshow(topo60c,topo60cR,'DisplayType','texturemap')
demcmap(topo60c)
camlight
material(0.6*[ 1 1 1])
plat = -50;
plon = 160;
tlat = -10;
tlon = 130;
camtargm(tlat,tlon,0)
camposm(plat,plon,1)
camupm(tlat,tlon)
set(gca,'CameraViewAngle',75)
land = shaperead('landareas.shp','UseGeoCoords',true);
linem([land.Lat],[land.Lon])
axis off
```
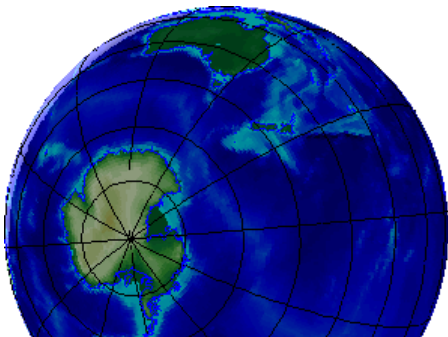
See Also

camtargm | camupm | campos | camva

Introduced before R2006a

camroll

Set or query roll angle of camera for geographic globe

Syntax

```
camroll(g,roll)
camroll(g,'auto')
camroll(g,'manual')

outRoll = camroll(g)
outRoll = camroll( ___ )
```

Description

Set Roll and Mode

`camroll(g,roll)` sets the roll angle of the camera for the specified geographic globe. Setting the roll angle rotates the camera around its lens. For more information about how camera rotations affect your view of the globe, see "How Camera Orientation Affects Globe View" on page 1-168.
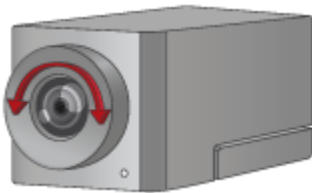
`camroll(g,'auto')` sets the camera roll to automatic mode, enabling the geographic globe to determine the roll angle based on plotted data. The mode defaults to automatic when you create a geographic globe. If you interact with the globe using your mouse, then the mode switches to automatic.

`camroll(g,'manual')` sets a manual mode, specifying that the geographic globe preserve the roll angle when the plotted data changes. If you change the roll angle using the `camroll` function, then the mode switches to manual.

Query Roll

`outRoll = camroll(g)` returns the roll angle of the camera.

`outRoll = camroll( ___ )` sets the roll angle or mode and then returns the roll angle of the camera. You can return the roll angle using any combination of input arguments from the previous syntaxes.
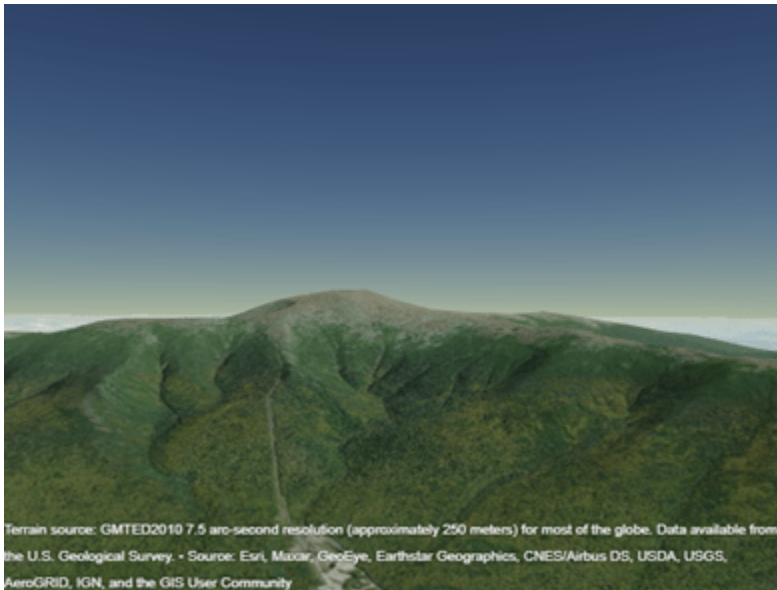
Examples

Change Roll Angle of Camera

Create a geographic globe. Position the camera near Mount Kilimanjaro by specifying a latitude, longitude, and ellipsoidal height. Set the pitch angle to 0 degrees, so that the camera points across the summit.

```
uif = uifigure;
g = geoglobe(uif);
campos(g,-3.1519,37.3561,5500)
campitch(g,0)
```
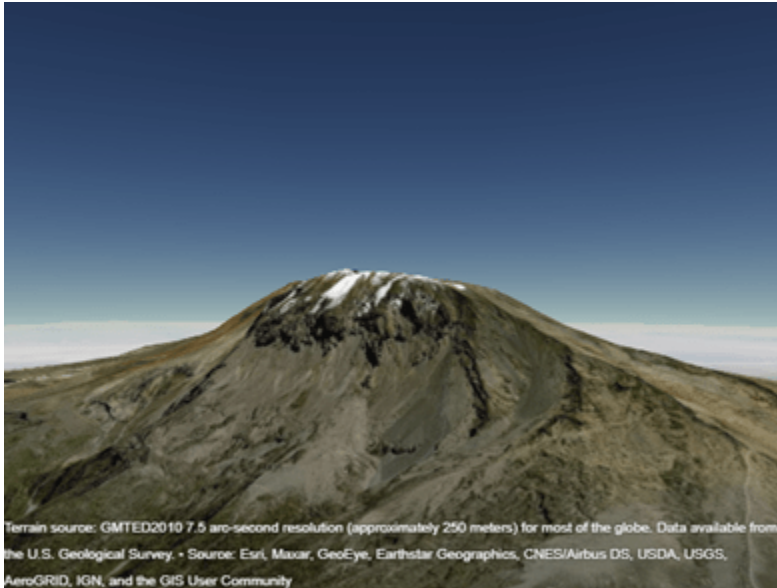

By default, the roll angle is 0 degrees. Rotate the mountain in a counterclockwise direction by increasing the roll angle to 30 degrees.

```
camroll(g,30)
```
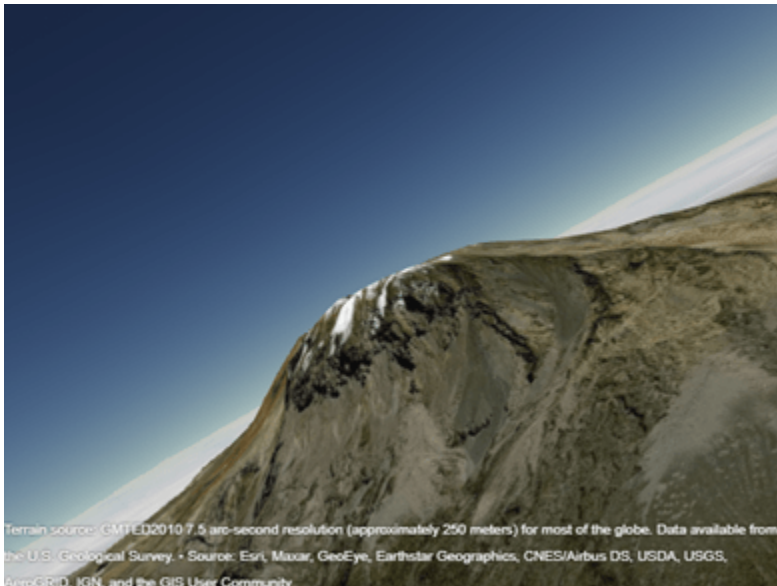
Animate Changes to Roll Angle

Create a geographic globe. Position the camera near Mount Washington by specifying a latitude, longitude, and ellipsoidal height. Change the heading and pitch angles so that the camera faces the mountain.

```
uif = uifigure;
g = geoglobe(uif);

campos(g,44.2668,-71.3849,1757)
camheading(g,90)
campitch(g,0)
```
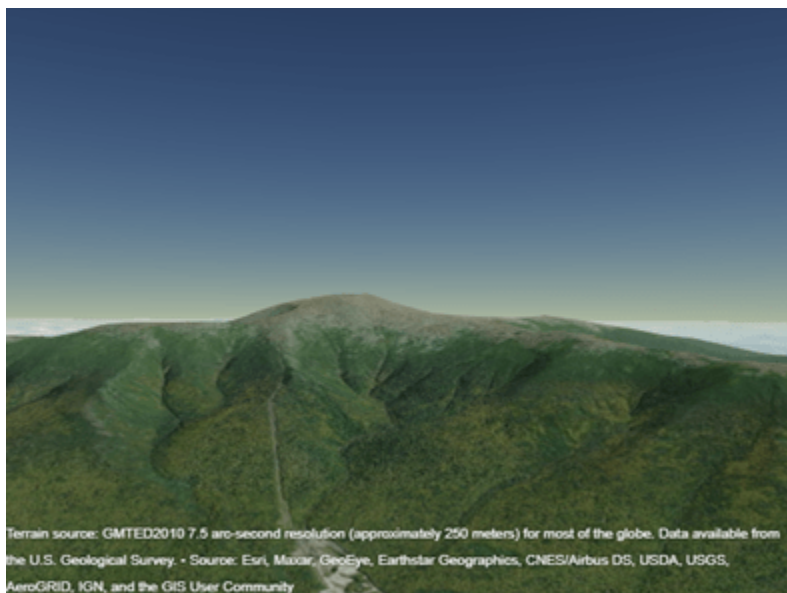

Animate the view by incrementally changing the roll angle. As the roll angle increases, the mountain rotates in a counterclockwise direction.

```
for roll = 0:5:180
    camroll(g,roll)
    drawnow
end
```

Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Get Information About Camera View

Get the position and the heading, pitch, and roll angles of the camera. Use this information to control the view of a different geographic globe or to automate navigation.

Create a geographic globe. Navigate to an area of interest using your mouse or gestures. For this example, navigate to an area around Hawaii.

```
uif = uifigure;
g = geoglobe(uif);
```


Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, DigitalGlobe, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Query the latitude, longitude, and ellipsoidal height of the camera, and assign each to a variable.

```
[camlat,camlon,camh] = campos(g)

camlat =

   18.1781


camlon =

 -155.9297


camh =

   6.6664e+04
```

Query the heading, pitch, and roll angles of the camera, and assign each to a variable.

```
heading = camheading(g)
pitch = campitch(g)
roll = camroll(g)

heading =

   16.7613


pitch =

  -24.1507


roll =

  359.9977
```

Use these values to control the view of a different geographic globe. For example, create a new geographic globe and programmatically set the view.

```
uif2 = uifigure;
g2 = geoglobe(uif2);
campos(g2,camlat,camlon,camh)
camheading(g2,heading)
campitch(g2,pitch)
camroll(g2,roll)
```

Preserve Camera View

Preserve the position and the heading, pitch, and roll angles of the camera by setting the camera modes to manual. If you do not set the camera modes to manual, then the camera view resets when you plot new data.

Import a sample route along roads in Massachusetts using the `gpxread` function. Create a geographic globe with a road map and no terrain data. Preserve the basemap and terrain settings by using the `hold` function. Then, navigate to an area near Eastern Massachusetts using your mouse.

```
track = gpxread('sample_tracks.gpx','Index',2);
lat = track.Latitude;
lon = track.Longitude;
height = track.Elevation;

uif = uifigure;
g = geoglobe(uif,'Basemap','streets','Terrain','none');
hold(g,'on')
```


Set the camera modes to manual and plot the data. Note that the camera position does not change.

```
campos(g,'manual')
camheight(g,'manual')
camheading(g,'manual')
campitch(g,'manual')
camroll(g,'manual')
geoplot3(g,lat,lon,height,'LineWidth',3)
```

Input Arguments

g — Geographic globe
GeographicGlobe object

Geographic globe, specified as a GeographicGlobe object.[5]

roll — Roll angle of camera
0 (default) | scalar

Roll angle of the camera, specified as a scalar value in the range [–360, 360] degrees.

More About

How Camera Orientation Affects Globe View

The values of the camera heading, pitch, and roll angles affect your view of a geographic globe. Mapping Toolbox references these values to the globe using a north-east-down (NED) coordinate system. As a result, when the heading, pitch, and roll angles of the camera are zero, the camera sits on a plane that is parallel to the tangent plane of the globe at the current latitude and longitude. For more information about NED coordinate systems, see "Choose a 3-D Coordinate System".

---

5.   Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

Change your view of a geographic globe by changing the heading, pitch, and roll angles of the camera:

- Heading — Rotate the camera about its *z*-axis, which shifts the view left or right. Move the view to the right by increasing the heading angle.

- Pitch — Rotate the camera about its *y*-axis, which tilts the view up or down. Tilt the view up by increasing the pitch angle.

- Roll — Rotate the camera about its *x*-axis, which spins the camera around its lens. Spin the view counterclockwise by increasing the roll angle.

Tips

- When the pitch angle is near –90 (the default) or 90 degrees, the camera loses one rotational degree of freedom. As a result, when you change the roll angle, the heading angle may change instead. This phenomenon is called gimbal lock. To avoid the effects of gimbal lock, call the `camheading` function instead of the `camroll` function.

See Also

Functions
geoglobe | campos | camheading | camheight | campitch

Topics
"Visualize Aircraft Line-of-Sight Over Terrain"

"Visualize UAV Flight Path on 2-D and 3-D Maps"

Introduced in R2020b

camtargm

Set camera target using geographic coordinates

Syntax

```
camtargm(lat,long,alt)
[x,y,z] = camtargm(lat,long,alt)
```

Description

camtargm(lat,long,alt) sets the axes `CameraTarget` property of the current map axes to the position specified in geographic coordinates. The inputs `lat` and `long` are assumed to be in the angle units of the current map axes.

`[x,y,z] = camtargm(lat,long,alt)` returns the camera target in the projected Cartesian coordinate system.

Examples

Look down the spine of the Andes from a location three Earth radii above the surface.

```
axesm('globe','galt',0)
gridm('glinestyle','-')
load topo60c
geoshow(topo60c,topo60cR,'DisplayType','texturemap');
demcmap(topo60c)
lightm(-80,-180);
material(0.6*[ 1 1 1])
plat = 10;
plon = -65;
tlat = -30;
tlon = -70;
camtargm(tlat,tlon,0);
camposm(plat,plon,3);
camupm(tlat,tlon);
camva(20)
set(gca,'CameraViewAngle',30)
land = shaperead('landareas.shp','UseGeoCoords',true);
linem([land.Lat],[land.Lon])
axis off
```
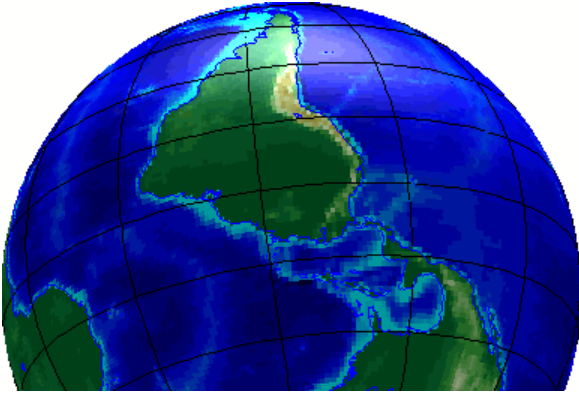
See Also

camposm | camupm | camtarget | camva

Introduced before R2006a

camupm

Set camera up vector using geographic coordinates

Syntax

```
camupm(lat,long)
[x,y,z] = camupm(lat,long)
```

Description

camupm(lat,long) sets the axes CameraUpVector property of the current map axes to the position specified in geographic coordinates. The inputs lat and long are assumed to be in the angle units of the current map axes.

[x,y,z] = camupm(lat,long) returns the camera position in the projected Cartesian coordinate system.

Examples

Look at northern Australia from a point south of and one Earth radius above New Zealand. Set CameraUpVector to the antipode of the camera target for that *down under* view.

```
axesm('globe','galt',0)
gridm('glinestyle','-')
load topo60c
geoshow(topo60c,topo60cR,'DisplayType','texturemap');
demcmap(topo60c)
camlight
material(0.6*[ 1 1 1])
plat = -50;
plon = 160;
tlat = -10;
tlon = 130;
[alat,alon] = antipode(tlat,tlon);
camtargm(tlat,tlon,0)
camposm(plat,plon,1)
camupm(alat,alon)
set(gca,'CameraViewAngle',80)
land = shaperead('landareas.shp','UseGeoCoords',true);
linem([land.Lat],[land.Lon])
axis off
```
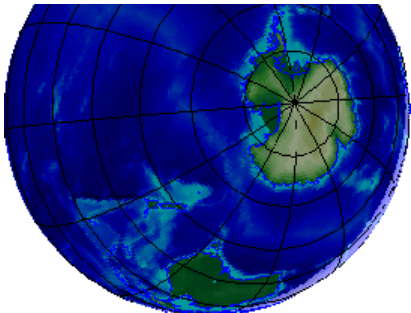
See Also

camtargm | camposm | camup | camva

Introduced before R2006a

cart2grn

Transform projected coordinates to Greenwich system

Syntax

```
[lat,lon,alt] = cart2grn
[lat,lon,alt] = cart2grn(hndl)
[lat,lon,alt] = cart2grn(hndl,mstruct)
```

Description

When objects are projected and displayed on map axes, they are plotted in Cartesian coordinates appropriate for the selected projection. This function transforms those coordinates back into the Greenwich frame, in which longitude is measured positively East from Greenwich (longitude 0), England and negatively West from Greenwich.

`[lat,lon,alt] = cart2grn` returns the latitude, longitude, and altitude data in geographic coordinates of the current map object, removing any clips or trims introduced during the display process from the output data.

`[lat,lon,alt] = cart2grn(hndl)` specifies the displayed map object desired with its handle `hndl`. The default handle is `gco`.

`[lat,lon,alt] = cart2grn(hndl,mstruct)` specifies the map structure associated with the object. The map structure of the current axes is the default.

See Also
`gcm` | `project` | `projinv` | `projfwd`

Introduced before R2006a

cat

Concatenate geographic or planar vector

Syntax

```
v = cat(1,v1,v2,...)
```

Description

`v = cat(1,v1,v2,...)` concatenates the geographic or planar vectors `v1,v2`, and so on, along the first dimension. If the class type of any property is a cell array, then the resultant field in the output `v` is also a cell array.

Examples

Concatenate Two Geopoint Vectors

Create two geopoint vectors. The first vector has one feature, and the second vector has two features. The vectors have different dynamic properties.

```
gp1 = geopoint(52,-101, 'Weather','Cloudy');
gp2 = geopoint([42 42.2], [-110.5 -110.7], 'Temperature', [65.6 63.2]);
```

Concatenate the vectors into a single geopoint vector. Note that the first input argument must be 1.

```
p = cat(1,gp1,gp2)

p =
 3x1 geopoint vector with properties:

 Collection properties:
       Geometry: 'point'
       Metadata: [1x1 struct]
 Feature properties:
       Latitude: [52 42 42.2000]
      Longitude: [-101 -110.5000 -110.7000]
        Weather: {'Cloudy'  ''  ''}
    Temperature: [0 65.6000 63.2000]
```

The concatenated vector has three features, and all features have all dynamic properties. Default property values are used when the value was not specified in the original geopoint vector. For example, the `'Temperature'` value of the first feature is set to 0 since no `'Temperature'` value was specified in `gp1`.

Input Arguments

`v1,v2,...` — Geographic or planar vectors to be concatenated
`geopoint`, `geoshape`, `mappoint`, or `mapshape` objects

Geographic or planar vectors to be concatenated, specified as `geopoint`, `geoshape`, `mappoint`, or `mapshape` objects. All of `v1`, `v2`,... are the same type of object.

Output Arguments

v — Concatenated geographic or planar vector
`geopoint`, `geoshape`, `mappoint`, or `mapshape` object

Concatenated geographic or planar vector, returned as a `geopoint`, `geoshape`, `mappoint`, or `mapshape` object. The object type of `v` matches the object type of `v1,v2,...`.

See Also
`vertcat`

Introduced in R2012a

changem

Substitute values in data array

Syntax

```
mapout = changem(Z,newcode,oldcode)
```

Description

`mapout = changem(Z,newcode,oldcode)` returns a data grid `mapout` identical to the input data grid, except that each element of `Z` with a value contained in the vector `oldcode` is replaced by the corresponding element of the vector `newcode`.

`oldcode` is 0 (scalar) by default, in which case `newcode` must be scalar. Otherwise, `newcode` and `oldcode` must be the same size.

Examples

Invent a map:

```
A = magic(3)

A =
     8     1     6
     3     5     7
     4     9     2
```

Replace instances of 8 or 9 with 0s:

```
B = changem(A,[0 0],[9 8])
B =
     0     1     6
     3     5     7
     4     0     2
```

Introduced before R2006a

circcirc

Intersections of circles in Cartesian plane

Syntax

```
[xout,yout] = circcirc(x1,y1,r1,x2,y2,r2)
```

Description

`[xout,yout] = circcirc(x1,y1,r1,x2,y2,r2)` finds the points of intersection (if any), given two circles, each defined by center and radius in x -y coordinates. In general, two points are returned. When the circles do not intersect or are identical, `NaN`s are returned.

When the two circles are tangent, two identical points are returned. All inputs must be scalars.

See Also
```
linecirc
```

Introduced before R2006a

clabelm

Add contour labels to map contour display

Syntax

```
clabelm(C)
clabelm(C,h)
clabelm(C,v)
clabelm(C,h,v)
clabelm( ___ ,Name,Value)

clabelm(C,'manual')
clabelm(C,h,'manual')

text_handles = clabel( ___ )
```

Description

`clabelm(C)` labels all contours displayed in the current contour plot. Labels are upright and displayed with `'+'` symbols. `clabelm` randomly selects label positions.

`clabelm(C,h)` rotates the labels and inserts them in the contour lines. This syntax inserts only those labels that fit within the contour, depending on the size of the contour.

`clabelm(C,v)` labels only the contour levels specified by the vector, `v`.

`clabelm(C,h,v)` labels only the contour levels specified by vector `v`, rotates the labels, and inserts them in the contour lines.

`clabelm( ___ ,Name,Value)` specifies the text object properties and the `'LabelSpacing'` contour group property, using one or more `Name,Value` pair arguments, in addition to any of the input arguments in previous syntaxes.

`clabelm(C,'manual')` places contour labels at locations you select with a mouse. Click the mouse or press the space bar to label the contour closest to the center of the crosshair. Press the **Return** key while the cursor is within the figure window to terminate labeling.

`clabelm(C,h,'manual')` places contour labels at locations you select with a mouse. Press the **Return** key while the cursor is within the figure window to terminate labeling. The labels are rotated and inserted in the contour lines.

`text_handles = clabel( ___ )` additionally returns an array containing the handles of the text objects created, using any of the input arguments in the previous syntaxes. If you call `clabel` without the `h` argument, `text_handles` also contains the handles of line objects used to create the `'+'` symbols.

Input Arguments

C — Contour matrix
2-by-n matrix

Contour matrix containing the data that defines the contour lines. C is returned by the `contourm`, `contourfm`, or `contour3m` functions.

h — Handle to the contour group object

Handle to the contour group object returned by the `contourm`, `contourfm`, or `contour3m` functions.

v — Contour level values
vector

Contour level values, specified as a row or column vector of individual values.

Example: [0,10,20]

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

LabelSpacing — Spacing between labels
144 (default) | scalar

Spacing between labels on each contour line, specified as the comma-separated pair consisting of `'LabelSpacing'` and a scalar. Specify the label spacing in points, where 1 point = $^1/_{72}$ inch.

Example: 'LabelSpacing',72

Output Arguments

text_handles — Handles of text objects

Handles of the text objects that `clabelm` creates. The `UserData` properties of the text objects contain the contour values displayed.

If you call `clabelm` without the `h` argument, `text_handles` also contains the handles of line objects used to create the '+' symbols.

Examples

Add Labels to Contour Plot

Get geoid heights and a geographic postings reference object from the EGM96 geoid model. Then, create a contour plot using a Miller projection. Add a frame using `framem` and eliminate extra white space using `tightmap`.

```
[N,R] = egm96geoid;
axesm miller
[C,h] = contourm(N,R,-100:50:80);
framem
tightmap
```
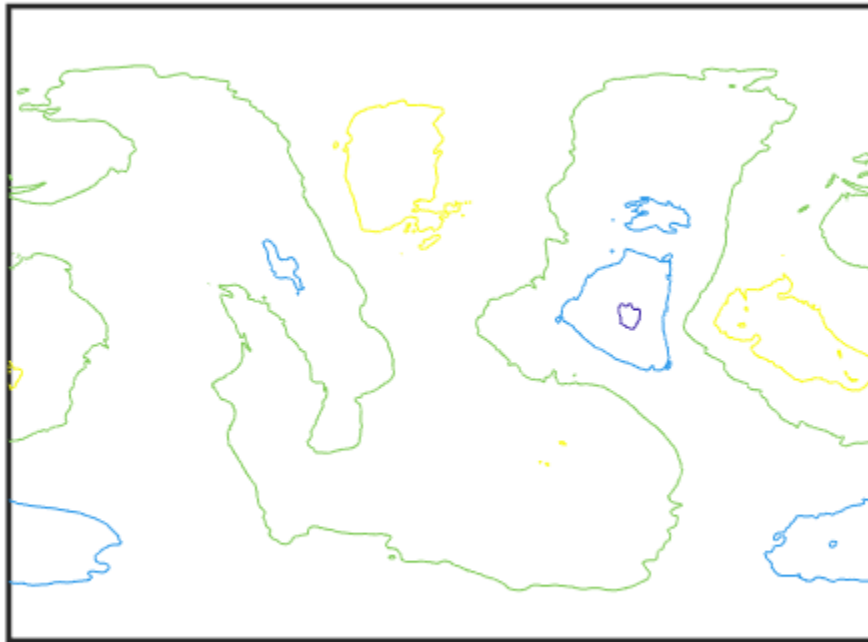
Add labels to the contour plot.

```
clabelm(C,h)
```
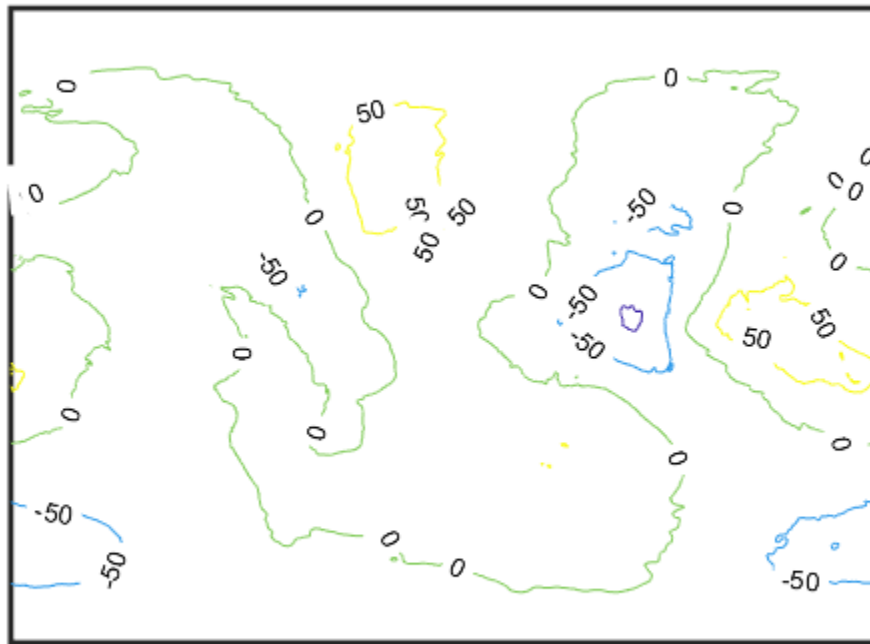
See Also

clegendm | contourm | contour3m | contourfm | clabel

Introduced before R2006a

clegendm

Add legend labels to map contour display

Syntax

```
clegendm(C,h)
clegendm(C,h,loc)
clegendm( ___ ,unitstr)
clegendm( ___ ,labels)
hl = clegendm( ___ )
```

Description

`clegendm(C,h)` adds a legend specifying the contour line heights, `C`, to the current map contour plot, `h`.

`clegendm(C,h,loc)` places the legend in a specified location.

`clegendm( ___ ,unitstr)` appends a string `unitstr` to each entry in the legend.

`clegendm( ___ ,labels)` uses the text specified in `labels` to label the legend.

`hl = clegendm( ___ )` returns the handle to the `legend` object created.

Examples

Create Legend for Contour Display

Load elevation raster data and a geographic cells reference object. Create a map axes object for the world. Display a contour plot using the raster data. Then, create a legend in the lower-right corner of the map. Specify the contour elevations as meters.

```
load topo60c
worldmap('world')
[c,h] = contourm(topo60c,topo60cR,-6000:1500:6000);
clegendm(c,h,4,'m')
```
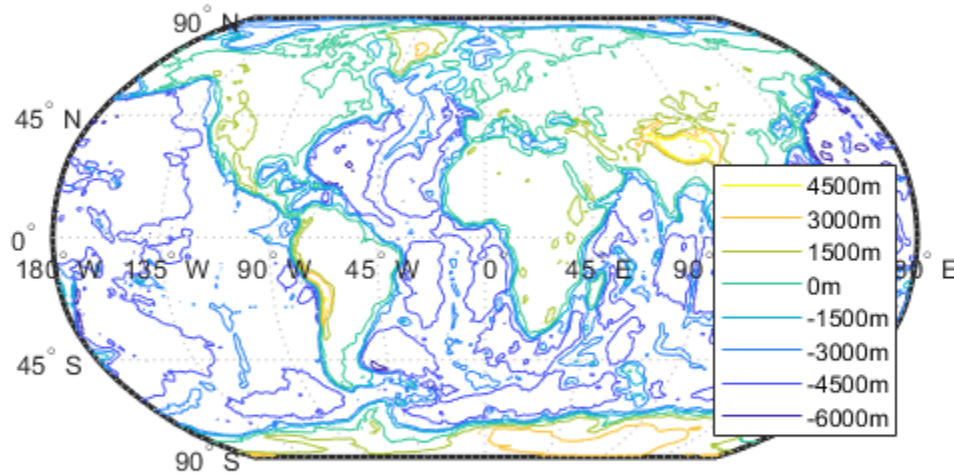
Input Arguments

C — Contour matrix
numeric matrix

Contour matrix, specified as a matrix with two rows. The first row represents longitude data and the second row represents latitude data. You can create a contour matrix by using `contourm`, `contour3m`, or `contourfm`.

h — Handle of contour patches
hggroup

Handle to the contour patches drawn onto the current axes, returned as an `hggroup`. You can get a handle to contour patches by using`contourm`, `contour3m`, or `contourfm`.

loc — Location
0 (default) | 1 | 2 | 3 | 4 | -1

Location to place legend, specified as one of the following integers.

Value	Placement
0	Automatic placement (default)
1	Upper right corner

Value	Placement
2	Upper left corner
3	Lower left corner
4	Lower right corner
-1	To the right of the plot

unitstr — Text to append
character vector | string scalar

Text to append to each entry in the legend, specified as a character vector or string scalar.

labels — Labels
cell array of character vectors | string array

Labels in the legend, specified as a string array or cell array of character vectors. `labels` must have the same number of entries as the line children of `h`.

Output Arguments

hl — Handle to `legend` object
handle

Handle to `legend` object created by the `clegendm` function, returned as a handle.

See Also

`clabelm` | `contourm` | `contour3m` | `contourfm` | `contourc` | `contourcbar`

Introduced before R2006a

clipdata

Clip data at `+/-pi` in longitude, `+/-pi` in latitude

Syntax

```
[lat,long,splitpts] = clipdata(lat,long,object)
```

Description

`[lat,long,splitpts] = clipdata(lat,long,object)` inserts `NaN`s at the appropriate locations in a map object so that a displayed map is clipped at the appropriate edges. It assumes that the clipping occurs at `+/- pi/2` radians in the latitude (*y*) direction and `+/- pi` radians in the longitude (*x*) direction.

The input data must be in radians and properly transformed for the particular aspect and origin so that it fits in the specified clipping range.

The output data is in radians, with `NaN`s placed at the proper locations. The output variable `splitpts` returns the row and column indices of the clipped elements (columns 1 and 2 respectively). These indices are necessary to restore the original data if the map parameters or projection are ever changed.

The `object` parameter can have any of the following values:

Object to clip	Description
`'surface'`	graticules
`'light'`	lights
`'line'`	lines
`'patch'`	patches
`'text'`	text object location points
`'point'`	point data
`'none'`	skip all clipping operations

See Also
`trimdata` | `undoclip` | `undotrim`

Introduced before R2006a

clma

Clear current map axes

Syntax

```
clma
clma all
clma purge
```

Description

`clma` deletes all displayed map objects from the current map axes but leaves the frame if it is displayed.

`clma all` deletes all displayed map objects, including the frame, but it leaves the map structure intact, thereby retaining the map axes.

`clma purge` removes the map definition from the current axes, but leaves all objects projected on the axes intact.

See Also
cla | clmo | handlem | hidem | namem | showm | tagm

Introduced before R2006a

clmo

Clear specified graphics objects from map axes

Syntax

```
clmo
clmo(handle)
clmo(object)
```

Description

`clmo` deletes all displayed graphics objects on the current axes.

`clmo(handle)` deletes those objects specified by their handles.

`clmo(object)` deletes those objects with names identical to the value `object`. This can be any value recognized by the `handlem` function, including entries in the `Tag` property of each object, or the object `Type` if the `Tag` property is empty.

See Also

clma | handlem | hidem | namem | showm | tagm

Introduced before R2006a

closePolygonParts

Close all rings in multipart polygon

Syntax

```
[xdata, ydata] = closePolygonParts(xdata, ydata)
[lat, lon] = closePolygonParts(lat, lon, angleunits)
```

Description

`[xdata, ydata] = closePolygonParts(xdata, ydata)` ensures that each ring in a multipart (`NaN`-separated) polygon is "closed" by repeating the start point at the end of each ring, unless the start and end points are already identical. Coordinate vectors `xdata` and `ydata` must match in size and have identical `NaN` locations.

`[lat, lon] = closePolygonParts(lat, lon, angleunits)` works with latitude-longitude data and accounts for longitude wrapping with a period of `360` if `angleunits` is `'degrees'` and `2*pi` if `angleunits` is `'radians'`. For a ring to be considered closed, the latitudes of its first and last vertices must match exactly, but their longitudes need only match modulo the appropriate period. Such rings are returned unaltered.

Examples

Close Polygon in Plane Coordinates

Create two vectors of planar coordinates.

```
xOpen = [1 0 2 NaN 0.5 0.5 1 1];
yOpen = [0 1 2 NaN 0.8 1 1 0.8];
```

Create a closed polygon from these coordinates.

```
[xClosed, yClosed] = closePolygonParts(xOpen,yOpen)
```

```
xClosed = 1×10

    1.0000         0    2.0000    1.0000       NaN    0.5000    0.5000    1.0000    1.0000    0.5

yClosed = 1×10

         0    1.0000    2.0000         0       NaN    0.8000    1.0000    1.0000    0.8000    0.8
```

Display all variables.

```
whos
```

```
  Name         Size             Bytes  Class     Attributes

  xClosed      1x10                80  double
```

```
xOpen         1x8              64  double
yClosed       1x10             80  double
yOpen         1x8              64  double
```

Close Polygon in Latitude-Longitude Coordinates

Load coastline data from MAT-file.

```
load coastlines
```

Construct a two-part polygon based on the coastlines data. The first ring is Antarctica. The longitude of its first vertex is -180 and the longitude of its last vertex is 180. The second ring is a small island from which the last vertex, a replica of the first vertex, is removed.

```
[latparts, lonparts] = polysplit(coastlat, coastlon);
latparts{2}(end) = [];
lonparts{2}(end) = [];
latparts(3:end) = [];
lonparts(3:end) = [];
[lat, lon] = polyjoin(latparts, lonparts);
```

Examine how `closePolygonParts` treats the two rings. In both cases, the first and last vertices differ. However, Antarctica remains unchanged while the small island is closed back up.

```
[latClosed, lonClosed] = closePolygonParts(lat, lon, 'degrees');
[latpartsClosed, lonpartsClosed] = polysplit(latClosed, lonClosed);
lonpartsClosed{1}(end) - lonpartsClosed{1}(1)   % Result is 360
```

```
ans = 360
```

```
lonpartsClosed{2}(end) - lonpartsClosed{2}(1)   % Result is 0
```

```
ans = 0
```

See Also
isShapeMultipart | removeExtraNanSeparators

Topics
"Create and Display Polygons"

Introduced in R2006a

colorui

Interactively define RGB color

Compatibility

Note `colorui` will be removed in a future release. Use `uisetcolor` instead.

Syntax

```
c = colorui
c = colorui(InitClr)
c = colorui(InitClr,FigTitle)
```

Description

`c = colorui` creates an interface for the definition of an RGB color triplet. On Windows® platforms, `colorui` produces the same interface as `uisetcolor`. On other machines, `colorui` produces a platform-independent dialog for specifying the color values.

`c = colorui(InitClr)` initializes the color value to the RGB triple given in `initclr`.

`c = colorui(InitClr,FigTitle)` where the character vector `FigTitle` specifies the window label.

The output value `c` is the selected RGB triple if the **Accept** or **OK** button is pushed. If the user presses **Cancel**, then the output value is set to `0`.

See Also
uisetcolor

combntns

All possible combinations of set of values

Note `combntns` will be removed in a future release. Use `nchoosek` instead.

Syntax

```
combos = combntns(set,subset)
```

Description

`combos = combntns(set,subset)` returns a matrix whose rows are the various combinations that can be taken of the elements of the vector `set` of length `subset`. Many combinatorial applications can make use of a vector `1:n` for the input set to return generalized, indexed combination subsets.

The `combntns` function provides the combinatorial subsets of a set of numbers. It is similar to the mathematical expression *a choose b*, except that instead of the number of such combinations, the actual combinations are returned. In combinatorial counting, the ordering of the values is not significant.

The numerical value of the mathematical statement *a choose b* is `size(combos,1)`.

Examples

How can the numbers 1 to 5 be taken in sets of three (that is, what is *5 choose 3*)?

```
combos = combntns(1:5,3)

combos =
     1     2     3
     1     2     4
     1     2     5
     1     3     4
     1     3     5
     1     4     5
     2     3     4
     2     3     5
     2     4     5
     3     4     5
size(combos,1)   % "5 choose 3"

ans =
    10
```

Note that if a value is repeated in the input vector, each occurrence is treated as independent:

```
combos = combntns([2 2 5],2)

combos =
     2     2
```

```
2    5
2    5
```

Tips

This is a recursive function.

Introduced before R2006a

comet3m

Project 3-D comet plot on map axes

Note `comet3m` will be removed in a future release. Use `projfwd` and `comet3` instead. For more information, see "Compatibility Considerations".

Syntax

```
comet3m(lat,lon,z)
comet3m(lat,lon,z,p)
```

Description

`comet3m(lat,lon,z)` traces a comet plot through the points specified by the input latitude, longitude, and altitude vectors.

`comet3m(lat,lon,z,p)` specifies a comet body of length `p*length(lat)`. The input `p` is `0.1` by default.

A comet plot is an animated graph in which a circle (the comet *head*) traces the data points on the screen. The comet *body* is a trailing segment that follows the head. The *tail* is a solid line that traces the entire function.

Examples

Create a 3-D comet plot of the coastlines data:

```
load coastlines
z = (1:length(coastlat))'/3000;
axesm miller
framem; gridm;
setm(gca,'galtitude',max(z)+.5)
view(3)
comet3m(coastlat,coastlon,z,0.01)
```

Compatibility Considerations

comet3m will be removed
Not recommended starting in R2013b

`comet3m` will be removed in a future release. Use `projfwd` and `comet3` instead. You can update your code using these replacement patterns.

Will Be Removed	Recommended
comet3m(lat,lon,z)	mstruct = gcm; [x,y] = projfwd(mstruct,lat,lon); comet3(x,y,z)

Will Be Removed	Recommended

`comet3m(lat,lon,z,p)`	`mstruct = gcm;`
	`[x,y] = projfwd(mstruct,lat,lon);`
	`comet3(x,y,z,p)`

See Also
`comet3` | `comet` | `projfwd`

Introduced before R2006a

cometm

Project 2-D comet plot on map axes

Note `cometm` will be removed in a future release. Use `projfwd` and `comet` instead. For more information, see "Compatibility Considerations".

Syntax

```
cometm(lat,lon)
cometm(lat,lon,p)
```

Description

`cometm(lat,lon)` traces a comet plot through the points specified by the input latitude and longitude vectors.

`cometm(lat,lon,p)` specifies a comet body of length `p*length(lat)`. The input `p` is `0.1` by default.

A comet plot is an animated graph in which a circle (the comet *head*) traces the data points on the screen. The comet *body* is a trailing segment that follows the head. The *tail* is a solid line that traces the entire function.

Examples

Create a comet plot of the coastlines data:

```
load coastlines
axesm miller
framem
cometm(coastlat,coastlon,0.01)
```

Compatibility Considerations

cometm will be removed
Not recommended starting in R2013b

`cometm` will be removed in a future release. Use `projfwd` and `comet` instead. You can update your code using these replacement patterns.

Will Be Removed	Recommended

`cometm(lat,lon)`	`mstruct = gcm;`
	`[x,y] = projfwd(mstruct,lat,lon);`
	`comet(x,y)`

`cometm(lat,lon,p)`	`mstruct = gcm;`
	`[x,y] = projfwd(mstruct,lat,lon);`
	`comet(x,y,p)`

See Also

comet | comet3 | projfwd

Introduced before R2006a

map.geodesy.ConformalLatitudeConverter

Convert between geodetic and conformal latitudes

Description

A `ConformalLatitudeConverter` object provides conversion methods between geodetic and conformal latitudes for an ellipsoid with a given eccentricity.

The conformal latitude maps an ellipsoid (oblate spheroid) to a sphere while preserving shapes and angles locally. (Curves that meet at a given angle on the ellipsoid meet at the same angle on the sphere.) Use conformal latitudes when implementing conformal map projections on the ellipsoid.

Creation

Syntax

```
converter = map.geodesy.ConformalLatitudeConverter
converter = map.geodesy.ConformalLatitudeConverter(spheroid)
```

Description

`converter = map.geodesy.ConformalLatitudeConverter` creates a `ConformalLatitudeConverter` object for a sphere and sets the `Eccentricity` property to `0`.

`converter = map.geodesy.ConformalLatitudeConverter(spheroid)` creates a conformal latitude converter object and sets the `Eccentricity` property to match the specified spheroid object.

Input Arguments

spheroid — Reference spheroid
`referenceEllipsoid` object | `oblateSpheroid` object | `referenceSphere` object

Reference spheroid, specified as a `referenceEllipsoid` object, `oblateSpheroid` object, or `referenceSphere` object. The term reference spheroid is used synonymously with reference ellipsoid. To create a reference spheroid, use the creation function for the object. To specify the reference ellipsoid for WGS84, use the `wgs84Ellipsoid` function.

For more information about reference spheroids, see "Reference Spheroids".

Example: `spheroid = referenceEllipsoid('GRS 80');`

Properties

Eccentricity — Ellipsoid eccentricity
`0` | numeric scalar

Ellipsoid eccentricity, specified as a numeric scalar. `Eccentricity` is in the interval [0, 0.5]. Eccentricities larger than 0.5 are possible in theory, but do not occur in practice and are not supported.

Data Types: `double`

Object Functions

forward   Convert geodetic latitude to authalic, conformal, isometric, or rectifying latitude
inverse   Convert authalic, conformal, isometric, or rectifying latitude to geodetic latitude

Examples

Create a Conformal Latitude Converter Object and Set Property

```
grs80 = referenceEllipsoid('GRS 80');

conv1 = map.geodesy.ConformalLatitudeConverter;
conv1.Eccentricity = grs80.Eccentricity

conv1 =

  ConformalLatitudeConverter with properties:

    Eccentricity: 0.0818
```

Create a Conformal Latitude Converter Object Specifying a Spheroid

```
grs80 = referenceEllipsoid('GRS 80');

conv2 = map.geodesy.ConformalLatitudeConverter(grs80)

conv2 =

  ConformalLatitudeConverter with properties:

    Eccentricity: 0.0818
```

See Also

Functions
geocentricLatitude | parametricLatitude

Objects
AuthalicLatitudeConverter | IsometricLatitudeConverter |
RectifyingLatitudeConverter

Introduced in R2013a

contains

Package: `map.rasterref`

Determine if geographic or map raster contains points

Syntax

```
tf = contains(R,lat,lon)
tf = contains(R,xWorld,yWorld)
```

Description

`tf = contains(R,lat,lon)` determines whether the points (`lat`, `lon`) in geographic coordinates fall within the bounds of geographic raster R.

`tf = contains(R,xWorld,yWorld)` determines whether the points (`xWorld`, `yWorld`) in the world coordinate system fall within the bounds of map raster R contains .

Examples

Check If Single Point Exists Within Bounds of Planar Raster

Create a `MapPostingsReference` raster reference object.

```
xWorldLimits = [207000 208000];
yWorldLimits = [912500 913000];
rasterSize = [11 21];
R = maprefpostings(xWorldLimits,yWorldLimits,rasterSize,'ColumnsStartFrom','north')

R =
  MapPostingsReference with properties:

             XWorldLimits: [207000 208000]
             YWorldLimits: [912500 913000]
               RasterSize: [11 21]
     RasterInterpretation: 'postings'
         ColumnsStartFrom: 'north'
            RowsStartFrom: 'west'
    SampleSpacingInWorldX: 50
    SampleSpacingInWorldY: 50
     RasterExtentInWorldX: 1000
     RasterExtentInWorldY: 500
         XIntrinsicLimits: [1 21]
         YIntrinsicLimits: [1 11]
       TransformationType: 'rectilinear'
       CoordinateSystemType: 'planar'
              ProjectedCRS: []
```

Check if the raster contains the point (207549,912753). The expected result is 1 (true) since the *x*-coordinate is within `R.XWorldLimits` and the *y*-coordinate is within `R.YWorldLimits`.

```
tf = contains(R,207549,912753)
```

```
tf = logical
   1
```

Check If Multiples Points Exist Within Bounds of Geographic Raster

Create a `GeographicCellsReference` raster reference object.

```
latlim = [0 89];
lonlim = [-180 179];
rasterSize = [90 360];
R = georefcells(latlim,lonlim,rasterSize,'ColumnsStartFrom','north')
```

```
R =
  GeographicCellsReference with properties:

             LatitudeLimits: [0 89]
            LongitudeLimits: [-180 179]
                 RasterSize: [90 360]
       RasterInterpretation: 'cells'
           ColumnsStartFrom: 'north'
              RowsStartFrom: 'west'
        CellExtentInLatitude: 0.988888888888889
      CellExtentInLongitude: 0.997222222222222
     RasterExtentInLatitude: 89
    RasterExtentInLongitude: 359
            XIntrinsicLimits: [0.5 360.5]
            YIntrinsicLimits: [0.5 90.5]
       CoordinateSystemType: 'geographic'
              GeographicCRS: []
                  AngleUnit: 'degree'
```

Check if points exist within the northern hemisphere.

```
pts_lat = [32 0 -10 32 212];
pts_lon = [-80 0 80 360 -80];
tf = contains(R,pts_lat,pts_lon)
```

```
tf = 1x5 logical array

   1   1   0   1   0
```

The first point is in the northern hemisphere. The second point is the origin, and `tf(2)` indicates the origin exists within the bounds of the northern hemisphere. The third point is in the southern hemisphere. The fourth point is identical to the first point after longitude wrapping. The element `tf(4)` demonstrates that the geographic raster supports wrapping of longitude coordinates. The last element `tf(5)` indicates that the geographic raster does not support wrapping of latitude coordinates.

Input Arguments

R — Geographic or map raster
GeographicCellsReference, GeographicPostingsReference, MapCellsReference, or MapPostingsReference object

Geographic or map raster, specified as a GeographicCellsReference, GeographicPostingsReference, MapCellsReference, or MapPostingsReference object.

lat — Latitude coordinates
numeric scalar or vector

Latitude coordinates, specified as a numeric scalar or vector.

Data Types: single | double

lon — Longitude coordinates
numeric scalar or vector

Longitude coordinates, specified as a numeric scalar or vector. Elements of lon can be wrapped arbitrarily without affecting the result.

Data Types: single | double

xWorld — *x*-coordinates in the world coordinate system
numeric scalar or vector

x-coordinates in the world coordinate system, specified as a numeric scalar or vector.

Data Types: single | double

yWorld — *y*-coordinates in the world coordinate system
numeric scalar or vector

y-coordinates in the world coordinate system, specified as a numeric scalar or vector.

Data Types: single | double

Output Arguments

tf — Flag indicating geographic or map raster vector contains points in the world coordinate system
logical scalar or vector

Flag indicating geographic or map raster vector contains points in the world coordinate system, returned as a logical scalar or vector. The *k*th element of tf is True when R contains the point (xWorld(*k*), yWorld(*k*)) in the world coordinate system.

Data Types: logical

See Also

Introduced in R2013b

contour3m

Project 3-D contour plot of map data

Description

The `contour3m` function is the same as the `contourm` function except that the lines for each contour level are drawn in their own horizontal plane, at the *z*-coordinate equal to the value of that level.

Examples

Contour Geoid Heights as 3-D Surface

Get geoid heights and a geographic postings reference object from the EGM96 geoid model. Then, create a 3-D contour plot with 40 contour levels using a Miller projection. Eliminate extra white space using `tightmap`.

```
[N,R] = egm96geoid;
axesm miller
contour3m(N,R,40,'LineColor','k')
tightmap
```
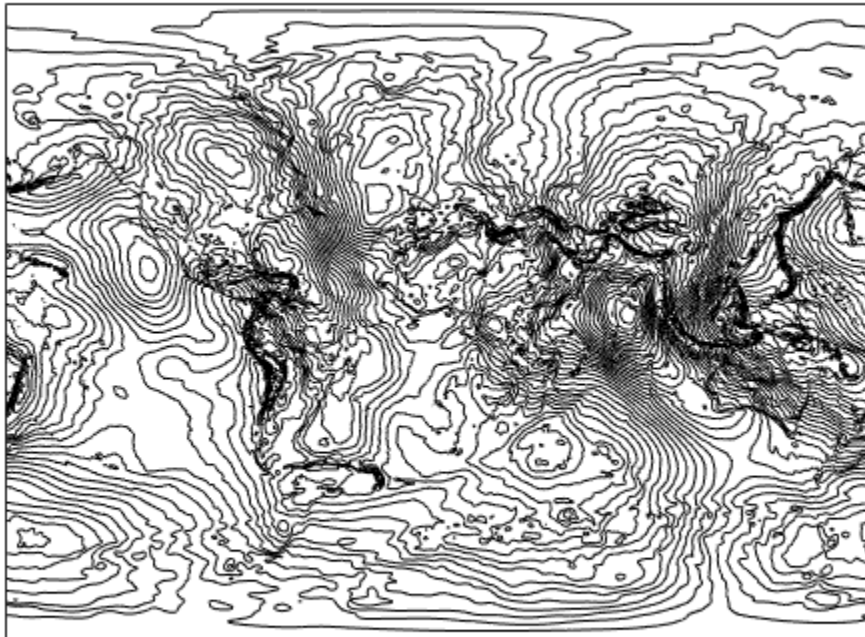
Add the geoid as a surface, exaggerate the aspect ratio of the axes using `daspect`, and view the map in 3-D.

```
hold on
geoshow(N,R,'DisplayType','surface')
daspect([1 1 50])
view(3)
```
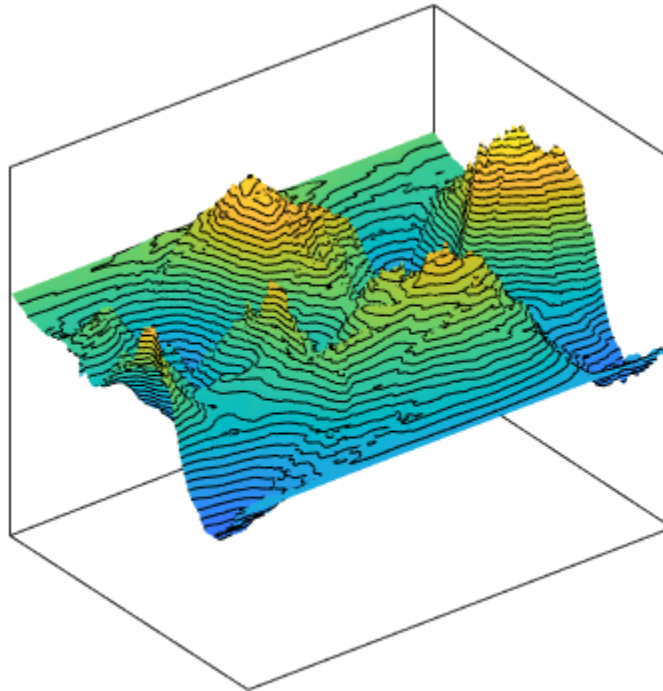

Contour Topography and Bathymetry of South Asia

This example shows how to contour in a map axes the topography and bathymetry of South Asia and the northern Indian Ocean with a contour interval of 500 meters.

Load elevation raster data and a geographic cells reference object. Crop the data to an area around South Asia.

```
load topo60c
latlim = [0 50];
lonlim = [35 115];
[Z,R] = geocrop(topo60c,topo60cR,latlim,lonlim);
```

Display the data as a contour plot using a standard Lambert conformal conic projection.

```
figure
axesm('lambertstd','MapLatLimit',latlim,'MapLonLimit',lonlim)
```

```
tightmap
axis off
contour3m(Z,R,'k','LevelStep',500)
```
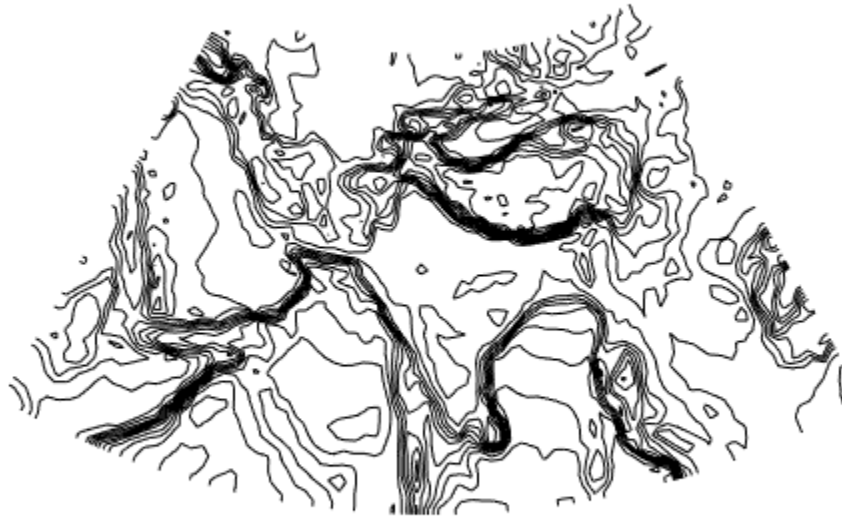

Display the topography as a surface. Apply a colormap appropriate for elevation data using the `demcmap` function. Then, add a title.

```
geoshow(Z,R,'DisplayType','surface')
demcmap(Z)
title({'South Asia Topography and Bathymetry', ...
    'with 500 m Contours'});
```

**South Asia Topography and Bathymetry
with 500 m Contours**
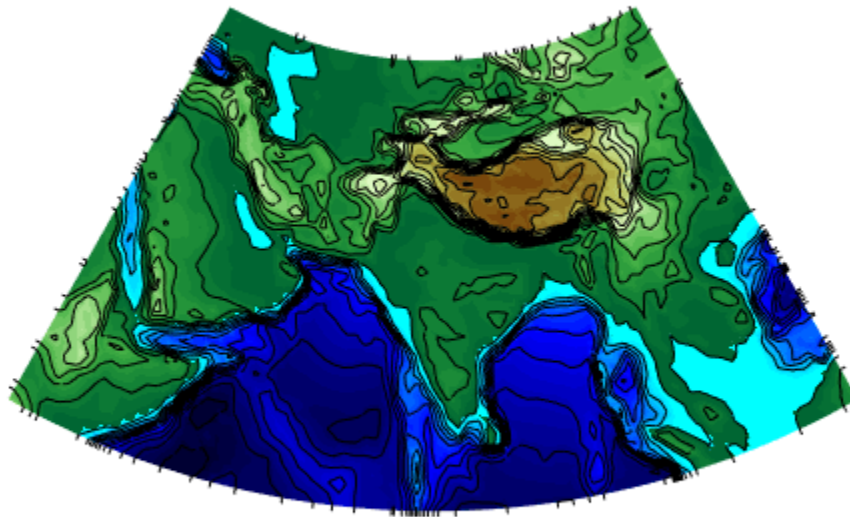
View the display in 3-D.

```
set(gca,'DataAspectRatio',[1 1 40000])
view(3)
```

South Asia Topography and Bathymetry
with 500 m Contours
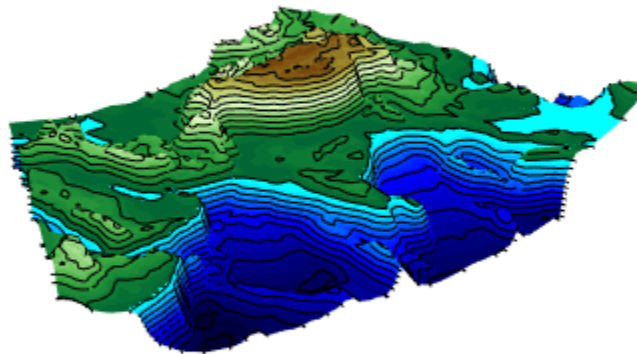
Tips

- If you use `contour3m` with the `globe` map display, the `contour3m` function warns. Be careful to scale the input data relative to the radius of your reference sphere.

See Also

clabel | clabelm | clegendm | contour | contour3 | contourm | contourfm | geoshow | plot

Introduced before R2006a

contourcbar

Color bar for filled contour map display

Syntax

```
H = contourcbar(...)
```

Description

`H = contourcbar(...)` creates a color bar associated with a filled contour display created with `contourfm`, `contourm`, `contour3m`, or `geoshow`. It supports the same syntax and usage options as the function `colorbar`.

Examples

Add Colorbar with Elevation Data

Load elevation raster data and a geographic cells reference object.

```
load topo60c
```

Create a map axes object with limits appropriate for North America. Display the elevation data on the map axes using a filled contour plot. Then, set the colormap limits and add a colorbar.

```
worldmap('north america')
contourfm(topo60c,topo60cR,-7000:1000:3000)
caxis([-8000 4000])
contourcbar
```
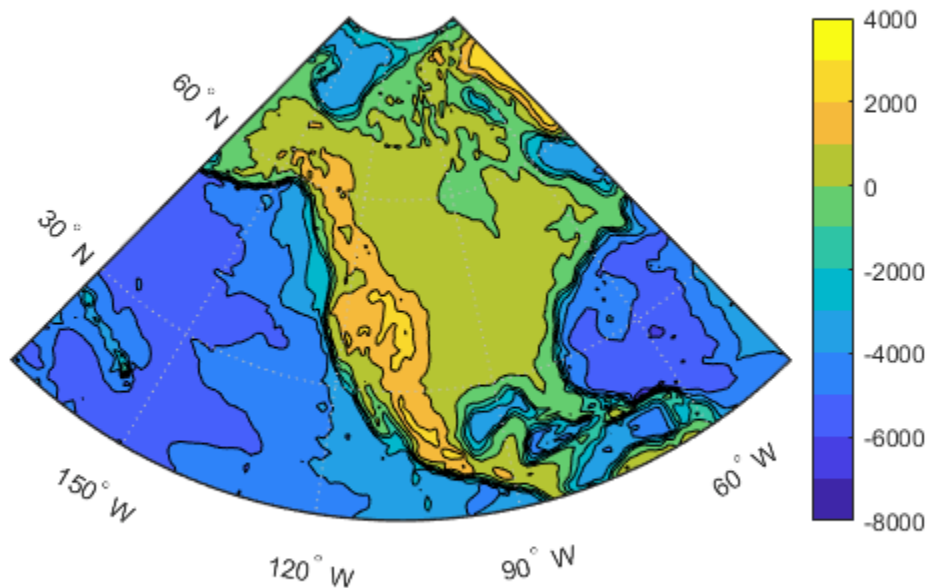
Add Colorbar to Contour Plot with Non-Uniform Levels

Create a world map. Move the meridian labels to the bottom of the map and reduce the number of meridian labels.

```
figure
ax = worldmap('world');
setm(ax,'MLabelParallel',-90)
setm(ax,'MLabelLocation',90)
```

Get geoid heights and a geographic postings reference object from the EGM96 geoid model. Specify non-uniform levels in meters at which to contour the geoid. Then, display the geoid as a filled contour plot.

```
[N,R] = egm96geoid;
levels = [-70 -40 -20 -10 0 10 20 40 70];
geoshow(N,R,'DisplayType','contour',...
    'LevelList',levels,'Fill','on','LineColor','black')
```
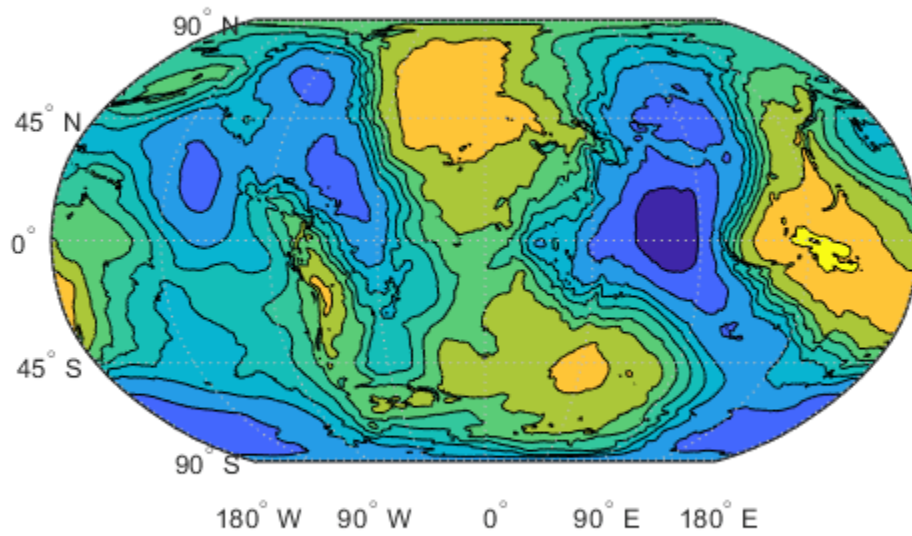
Add a colorbar to the bottom of the figure. Add a label.

```
cb = contourcbar('peer',ax,'Location','southoutside');
cb.XLabel.String = 'Geoid Undulation in Meters';
```
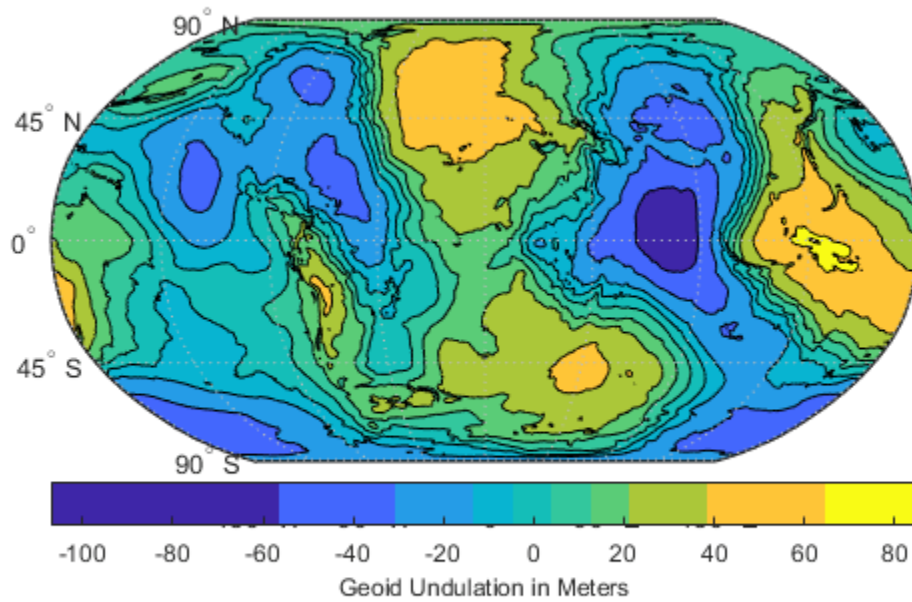
Set the colormap limits using the `caxis` function. Then, apply a different colormap.

```
caxis([-110 90])
colormap(hsv)
```

Geoid Undulation in Meters
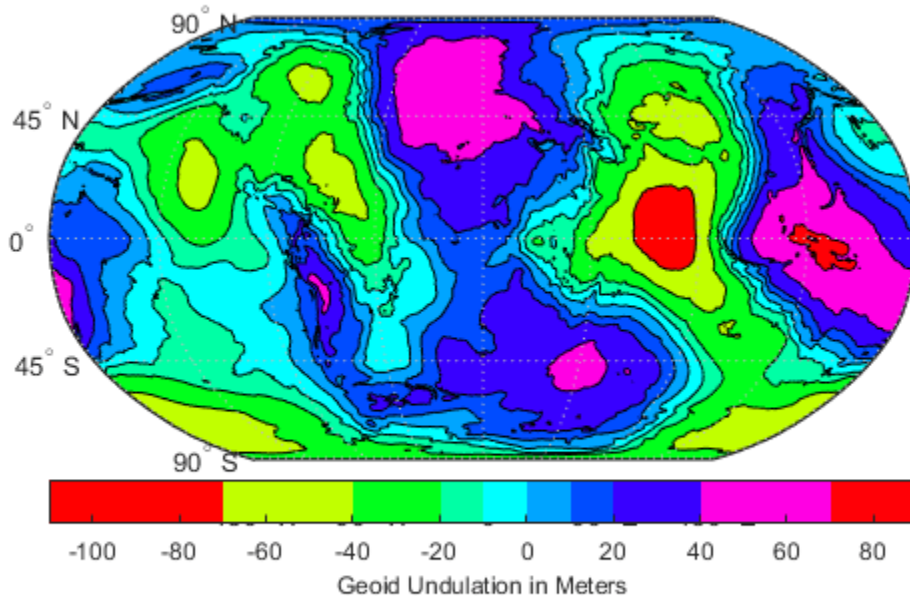
Tips

- If a *peer* axes is specified when calling `contourcbar`, it should be a map axes containing an object created using one of the Mapping Toolbox functions listed previously. Otherwise the current axes should contain such an object.

- If a Mapping Toolbox contour object is present, then the color bar is filled with solid blocks of color which bound each other at the contour levels used in the plot. Thus, the contour levels bounding a fill polygon of a given color can be inferred graphically by inspecting the upper and lower limits of the corresponding block in the color bar. In the absence of a Mapping Toolbox contour object an ordinary color bar is created.

- If multiple Mapping Toolbox contour objects are present in the same axes, then the levels used to divide the color bar into blocks will correspond to the first contour object that is found. This situation could occur when a larger data set is broken up into multiple grid tiles, for example, but as long the tiles all use the same contour level list, the color bar will correctly represent them all.

See Also

`clegendm` | `colorbar` | `contourfm`

Introduced in R2011b

contourcmap

Contour colormap and colorbar for current axes

Syntax

```
contourcmap(cmapstr)
contourcmap(cmapstr,cdelta)
contourcmap(...,Name,Value)
h = contourcmap(...)
```

Description

`contourcmap(cmapstr)` updates the figure's colormap for the current axes with the colormap specified by `cmapstr`. If the axes contains Mapping Toolbox contour objects, the resultant colormap contains the same number of colors as the original colormap. Otherwise, the resultant colormap contains ten colors.

`contourcmap(cmapstr,cdelta)` updates the figure's colormap with colors varying according to `cdelta`. If the axes contains Mapping Toolbox contour objects, the value of `cdelta` is ignored.

`contourcmap(...,Name,Value)` allows you to add a colorbar and control the properties of the colorbar. Parameter names can be abbreviated and are case-insensitive.

`h = contourcmap(...)` returns a handle to the colorbar axes.

Input Arguments

cmapstr

A character vector that specifies a colormap. Valid entries for `cmapstr` include `'pink'`, `'hsv'`, `'jet'`, or the name of any similar MATLAB colormap function.

cdelta

A scalar or vector. If `cdelta` is a scalar, it represents a step size, and colors are generated at multiples of `cdelta`. If `cdelta` is a vector of evenly spaced values, colors are generated at those values; otherwise an error is issued.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Colorbar

Character vector with values `'on'` or `'off'` specifying whether a colorbar is present, `'on'`, or absent from the axes, `'off'`.

Default: `'off'`

Location

Character vector specifying the location of the colorbar. Permissible values are `'vertical'`, `'horizontal'`, or `'none'`.

Default: `'vertical'`

ColorAlignment

Character vector specifying the alignment of the labels in the colorbar. Permissible values are `'center'`, where the labels are centered on the color bands, or `'ends'`, where the labels are centered on the color breaks. If the axes contains Mapping Toolbox contour objects, the `ColorAlignment` will be set automatically to `'center'` for contour lines and `'ends'` for filled contours, and cannot be modified.

SourceObject

Handle of the graphics object which is used to determine the color limits for the colormap. The `SourceObject` value is the handle of a currently displayed object.

Default: `gca`

TitleString

Title of the colorbar axes, specified as a character vector.

XLabelString

X label of the colorbar axes, specified as a character vector.

YLabelString

Y label of the colorbar axes, specified as a character vector.

ZLabelString

Z label of the colorbar axes, specified as a character vector. In addition, properties and values that can be applied to the title and labels of the colorbar axes are valid.

Output Arguments

h

A handle to the colorbar axes.

Examples

Display World Map with Colormap

Display a world map with a colormap representing contour intervals in meters.

First, load elevation raster data and a geographic cells reference object.

```
load topo60c
```

Create a map axes object with limits appropriate for the data. Display the elevation data using a filled contour plot. Then, update the colormap and add a labeled colorbar.

```
worldmap(topo60c,topo60cR)
contourfm(topo60c,topo60cR)
contourcmap('jet','Colorbar','on', ...
    'Location','horizontal', ...
    'TitleString','Contour Intervals in Meters');
```
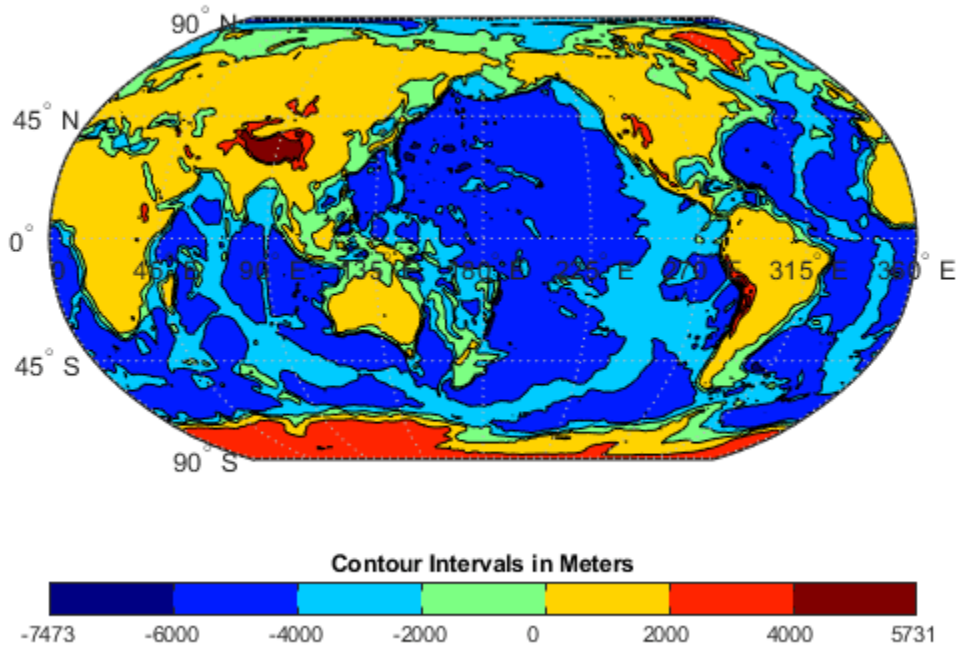

Display Custom Contour Intervals

Display a world map with a colormap in which the colors vary at a step size of 2000.

First, load elevation raster data and a geographic cells reference object. Then, load coastline coordinates.

```
load topo60c
load coastlines
```

Create a map axes object with limits appropriate for the data. Display the elevation and coastline data.

```
worldmap(topo60c,topo60cR)
geoshow(topo60c,topo60cR,'DisplayType','texturemap')
geoshow(coastlat,coastlon,'Color','k')
```
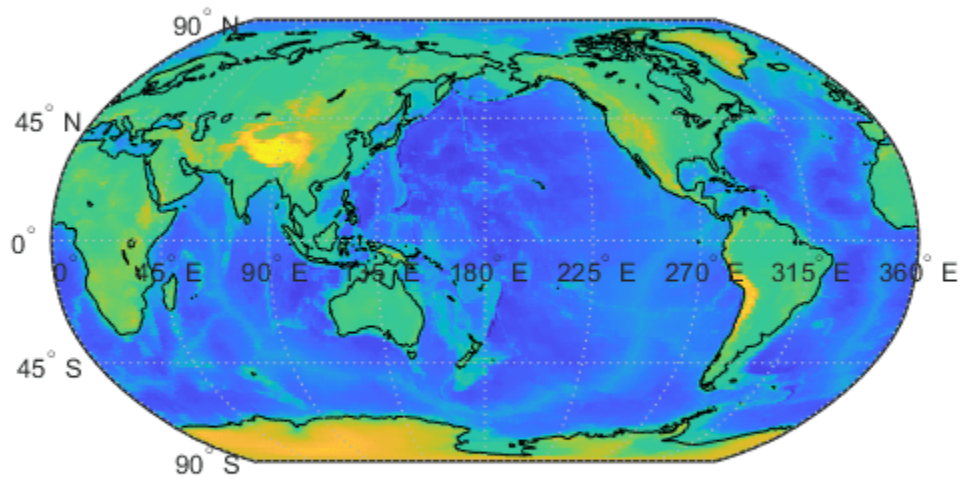
Update the colormap and add a labeled colorbar. Specify the step size as the second argument.

```
contourcmap('summer',2000,'Colorbar','on', ...
    'Location','horizontal', ...
    'TitleString','Contour Intervals in Meters')
```

Contour Intervals in Meters
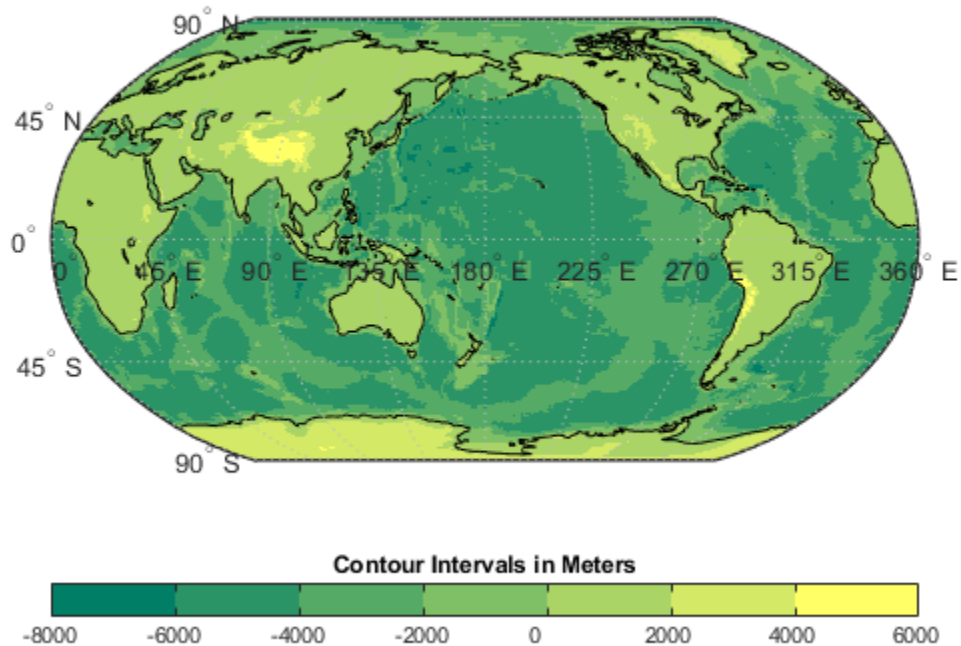
Restrict Range of Color Values

Get geoid heights and a geographic postings reference object from the EGM96 geoid model. Display a colorbar using the `jet` colormap. Specify the contour interval as 10 meters.

```
[N,R] = egm96geoid;
worldmap(N,R)
geoshow(N,R,'DisplayType','surface')
contourcmap('jet',10,'Colorbar','on','Location','horizontal')
```
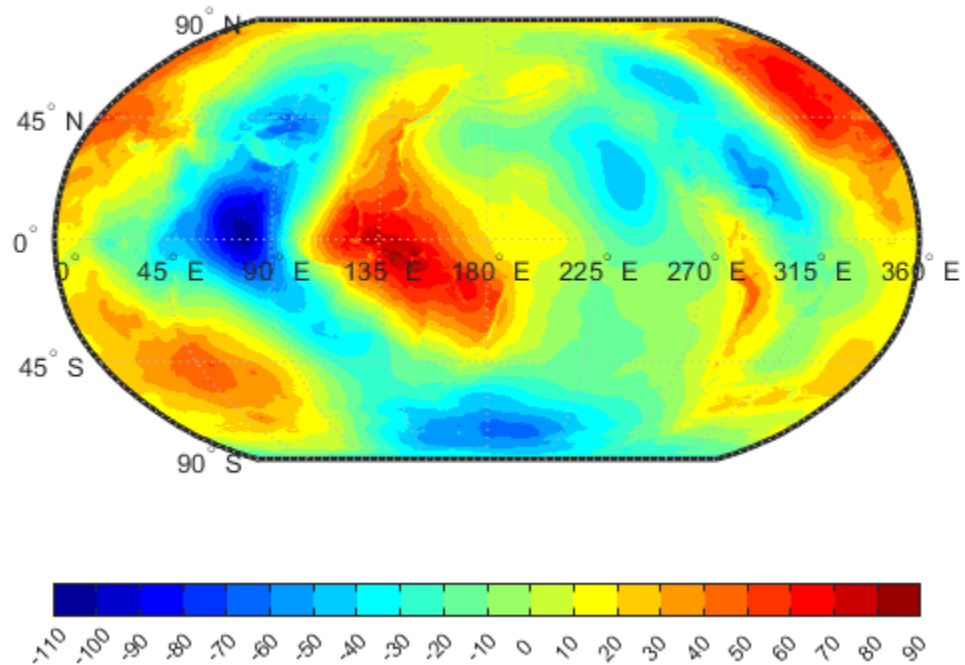
Restrict the range of color values by specifying a vector of evenly spaced values.

```
range = 0:5:50;
contourcmap('jet',range,'Colorbar','on','Location','horizontal')
```
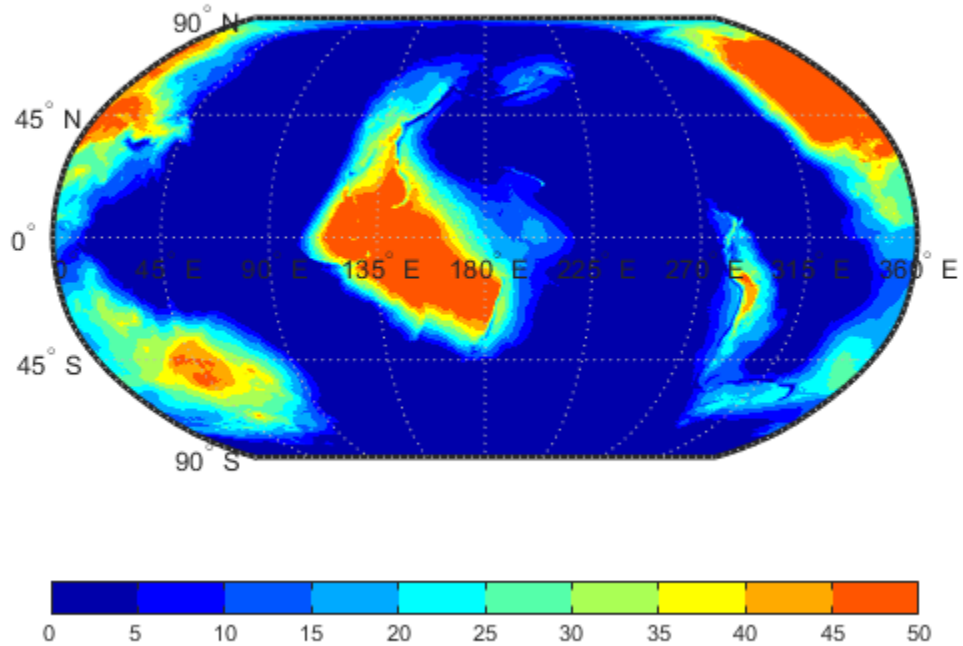
See Also

clabelm | clegendm | colormap | contour3m | contourcbar | contourfm | contourm

Introduced before R2006a

contourfm

Project filled 2-D contour plot of map data

Description

The `contourfm` function is the same as the `contourm` function except that the areas between contours are filled with colors. For each contour interval, `contourfm` selects a distinct color from the figure's colormap. You can obtain the same result by setting `'Fill','on'` and `'LineColor','black'` when calling `contourm`.

Examples

Display Filled Contours for EGM96 Geoid Heights

Get geoid heights and a geographic postings reference object from the EGM96 geoid model. Load coastline latitude and longitude data.

```
[N,R] = egm96geoid;
load coastlines
```

Create a filled contour plot of the geoid data on a world map. Plot the coastline data using `geoshow` and display a colorbar for the contour plot using `contourcbar`.

```
worldmap('world')
levels = -120:20:100;
contourfm(N,R,levels,'LineStyle','none')
geoshow(coastlat,coastlon,'Color','k')
contourcbar
```
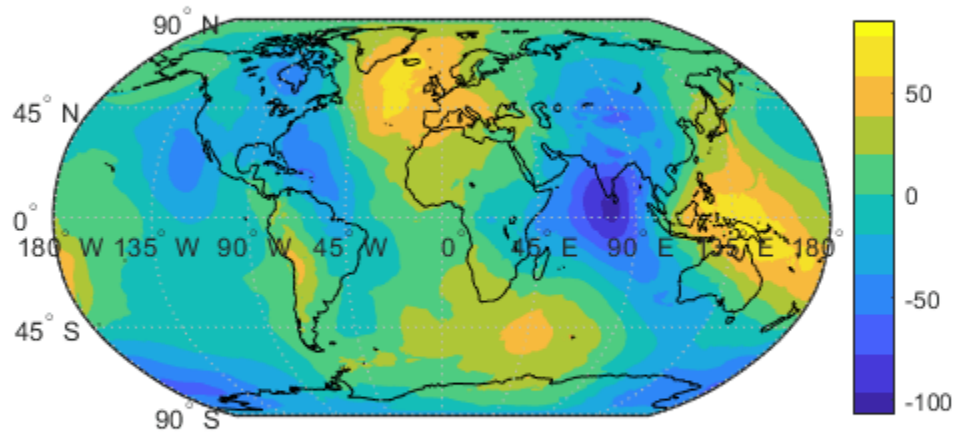
See Also
contourcbar | contourm | contour3m | clabelm | meshm | surfm

Introduced before R2006a

contourm

Project 2-D contour plot of map data

Syntax

```
contourm(Z,R)
contourm(lat,lon,Z)
contourm( ___ ,n)
contourm( ___ ,V)
contourm( ___ ,linetype)
contourm( ___ ,Name,Value)
C = contourm( ___ )
[C,h] = contourm( ___ )
```

Description

`contourm(Z,R)` creates a contour plot of the regular data grid Z with geographic reference R.

`contourm(lat,lon,Z)` displays a contour plot of the geolocated data grid, Z with geolocation defined by `lat` and `lon`.

`contourm( ___ ,n)` draws n contour levels.

`contourm( ___ ,V)` draws contours at the levels specified by V.

`contourm( ___ ,linetype)` specifies the line style and color for the contours.

`contourm( ___ ,Name,Value)` allows you to set optional parameters. Parameter names can be abbreviated, and case does not matter. In addition, any of the following `hggroup` properties can be specified: `'HandleVisibility'`, `'Parent'`, `'Tag'`, `'UserData'`, and `'Visible'`.

`C = contourm( ___ )` returns a standard contour matrix C, with the first row representing longitude data and the second row representing latitude data.

`[C,h] = contourm( ___ )` returns the contour matrix and the handle to the contour patches drawn onto the current axes. The handle is of type `hggroup`.

Examples

Display Contours for EGM96 Geoid Heights

Get geoid heights and a geographic postings reference object from the EGM96 geoid model.

```
[N,R] = egm96geoid;
```

Create a contour plot of the geoid data. Add axis labels and a legend.

```
[c,h] = contourm(N,R,'LevelStep',20,'ShowText','on');
xlabel('Longitude')
```

```
ylabel('Latitude')
clegendm(c,h,-1)
```
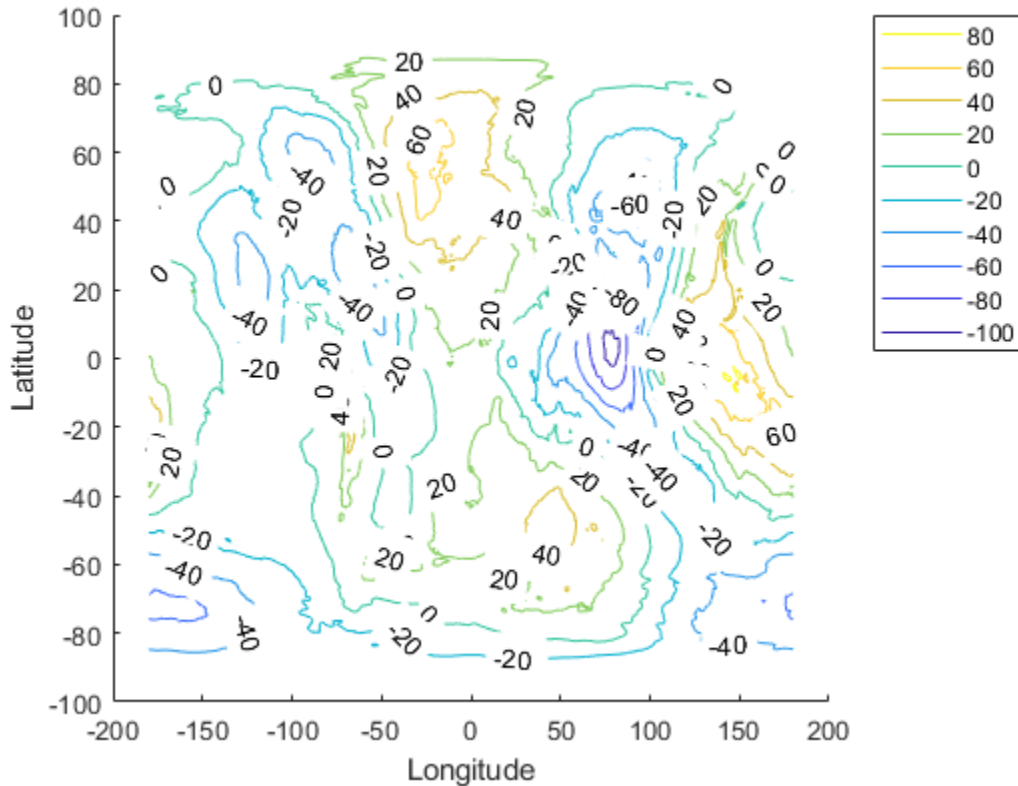

Display Contour Geoid Heights for Area Including Korea

Contour geoid heights for an area including Korea with a backdrop of terrain elevations and bathymetry.

Get elevation and geoid height data for an area around the Korean peninsula.

```
load korea5c
N = egm96geoid(korea5cR);
```

Create a map axes object with appropriate limits. Then, display the elevation data. Apply a colormap.

```
figure
worldmap(korea5cR.LatitudeLimits,korea5cR.LongitudeLimits)
geoshow(korea5c,korea5cR,'DisplayType','texturemap')
demcmap(korea5c)
```
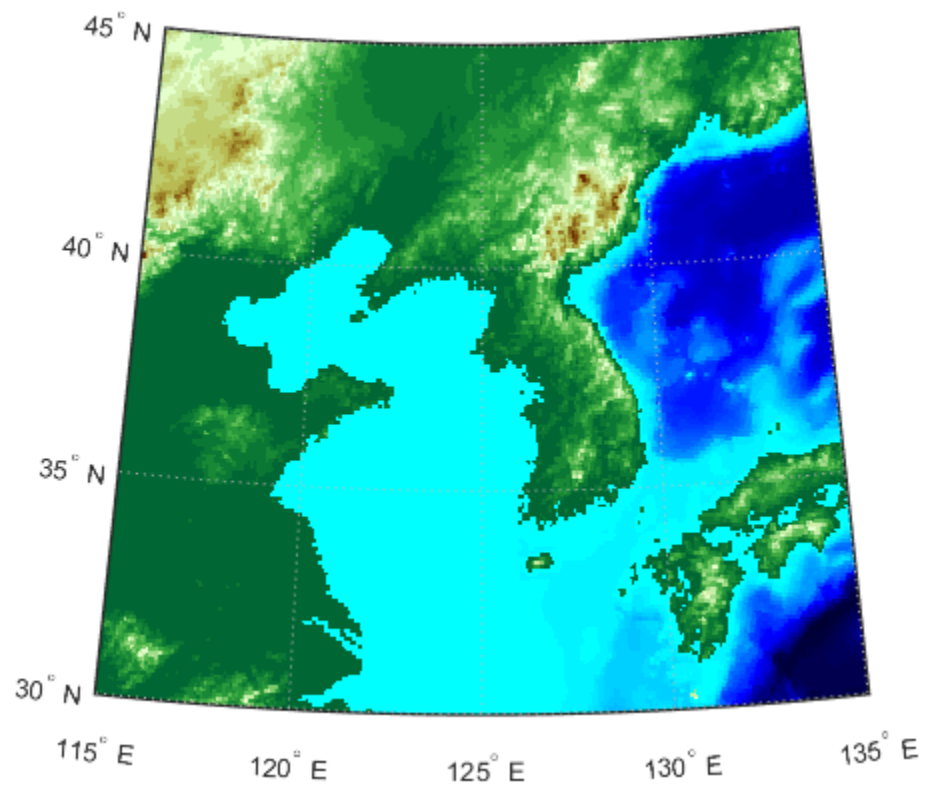
Display contours of the geoid values from -100 to 100 in increments of 5.

```
[c,h] = contourm(N,korea5cR,-100:5:100,'k');
```
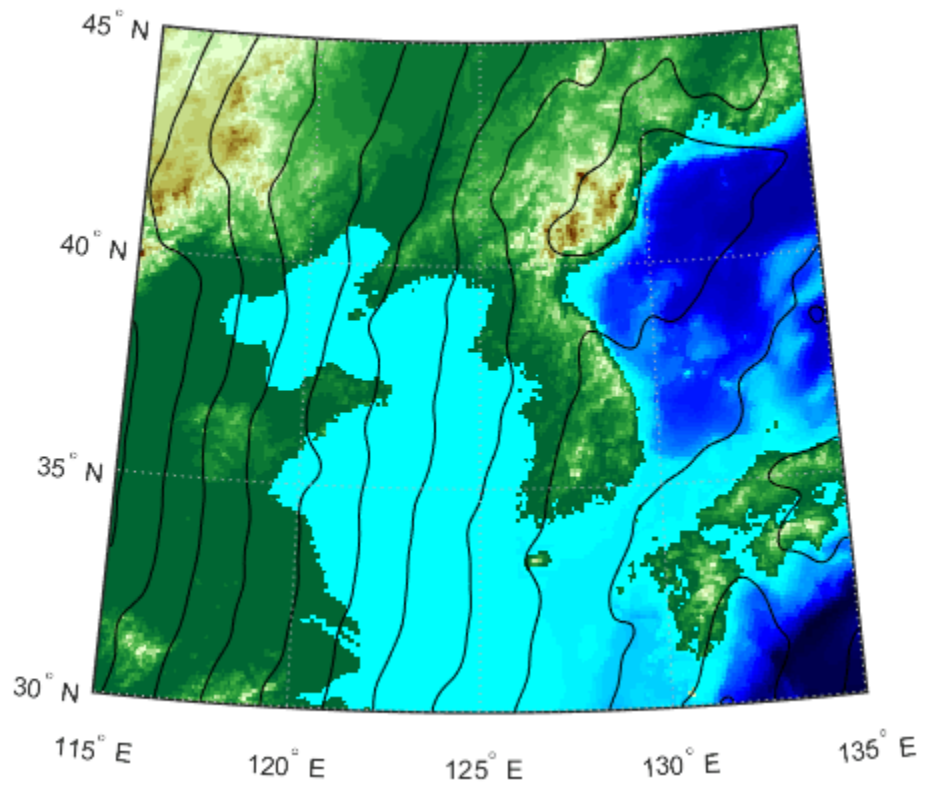
Add red labels with white backgrounds to the contours.

```
t = clabelm(c,h);
set(t,'Color','r')
set(t,'BackgroundColor','white')
set(t,'FontWeight','bold')
```
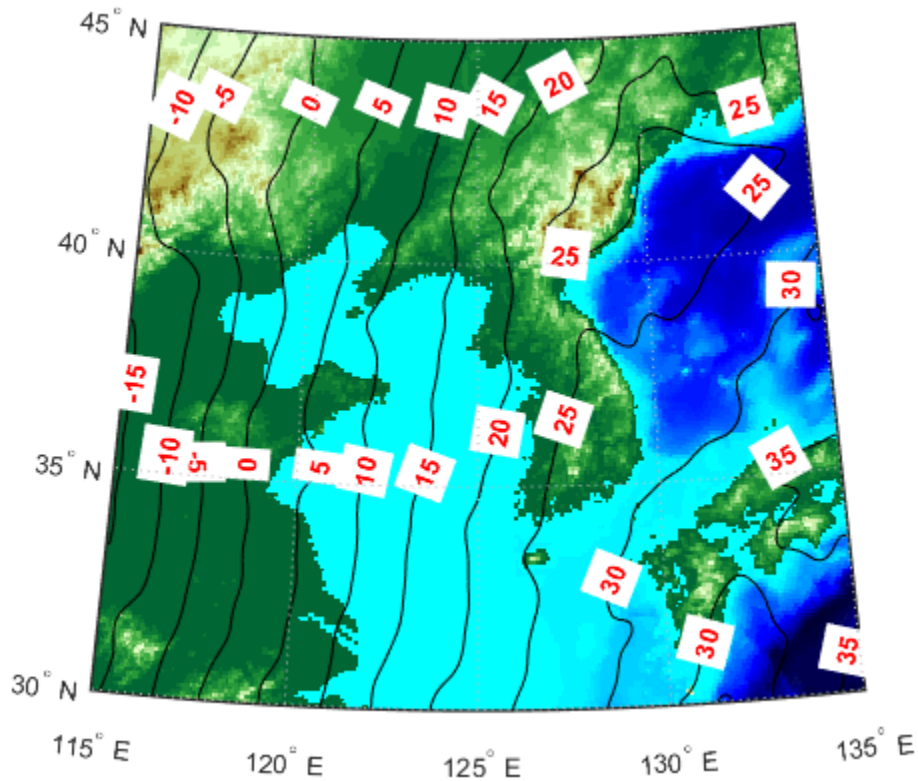
Input Arguments

Z — Regular or geolocated data grid
M-by-*N* matrix

Regular or geolocated data grid, specified as an *M*-by-*N* matrix.

If the grid contains regions with missing data, set the corresponding elements of Z to NaN. Contour lines terminate when entering such areas. Similarly, if you use `'Fill','on'` or call `contourfm`, such null-data areas will not be filled. If you use the syntax `contourm(lat,lon,Z,...)`, however, `lat` and `lon` must have finite, non-NaN values everywhere. In this case, set Z to NaN in null data areas, but make sure the corresponding elements of `lat` and `lon` have finite values that specify actual locations on the Earth.

R — Geographic reference
geographic raster reference object | vector | matrix

Geographic reference, specified as one of the following. For more information about referencing vectors and matrices, see "Georeferenced Raster Data".

Type	Description
Geographic raster reference object	`GeographicCellsReference` or `GeographicPostingsReference` geographic raster reference object. The `RasterSize` property must be consistent with the size of the data grid, `size(Z)`.
Vector	1-by-3 numeric vector with elements: `[cells/degree northern_latitude_limit western_longitude_limit]`
Matrix	3-by-2 numeric matrix that transforms raster row and column indices to or from geographic coordinates according to: `[lon lat] = [row col 1] * R` R defines a (non-rotational, non-skewed) relationship in which each column of the data grid falls along a meridian and each row falls along a parallel. For more information about referencing vectors and matrices, see "Georeferenced Raster Data". If the current axis is a map axis, the coordinates of Z are projected using the projection structure from the axis. The contours are drawn at their corresponding Z level.

lat, lon — Geolocation array
M-by-*N* matrix | *M*-element vector

Geolocation array with latitude or longitude coordinates, specified as a matrix of the same size as Z, or a vector with length matching the number of rows in Z.

n — Number of contour levels
numeric scalar

Number of contour levels, specified as a numeric scalar.

V — Value of contour levels
numeric vector
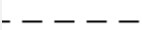
Value of contour levels, specified as a numeric vector with length greater than or equal to two. Use V = [v v] to compute a single contour at level v.
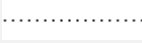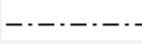
linetype — Line style and color
character vector | string

Line style and color, specified as a character vector or string containing symbols. The symbols can appear in any order. You do not need to specify both characteristics (line style and color).

Example: `'--r'` is a red dashed line

Line Style	Description	Resulting Line
`'-'`	Solid line	—————
`'--'`	Dashed line	– — — — —

Line Style	Description	Resulting Line
':'	Dotted line	....
'-.'	Dash-dotted line	—.—.—.—.-

Color Name	Short Name	Appearance
'red'	'r'	
'green'	'g'	
'blue'	'b'	
'cyan'	'c'	
'magenta'	'm'	
'yellow'	'y'	
'black'	'k'	
'white'	'w'	

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'LabelSpacing',72`

Fill — Color areas between contour lines
`'off'` (default) | `'on'`

Color areas between contour lines, specified as the comma-separated pair consisting of `'Fill'` and `'off'` or `'on'`. By default `contourm` draws a line (which may have multiple parts) for each contour level. If you set `Fill` to `'on'`, then `contourm` colors the polygonal regions between the lines, selecting a distinct color for each contour interval from the colormap of the figure in which the contours are drawn. Setting `Fill` to `'on'` is almost the same as calling `contourfm`; the only difference is that `contourfm` also sets `LineColor` to black by default.

LabelSpacing — Spacing between labels
numeric scalar

Spacing between labels on each contour line, specified as the comma-separated pair consisting of `'LabelSpacing'` and a numeric scalar. When you display contour line labels either by calling `clabelm` or by specifying `'ShowText','on'`, the labels by default are spaced 144 points (2 inches) apart on each line. You can specify the spacing by setting `LabelSpacing` to a value in points. If the length of an individual contour line is less than the specified value, only one contour label is displayed on that line.

LevelList — Values at which contour lines are drawn
numeric vector

Values at which contour lines are drawn, specified as the comma-separated pair consisting of `'LevelList'` and a numeric vector. This property uses a row vector of increasing values to specify the levels at which contour lines are drawn.

LevelStep — Spacing of contour lines
positive real scalar

Spacing of contour lines, specified as the comma-separated pair consisting of `'LevelStep'` and a numeric scalar. The `contourm` function draws contour lines at regular intervals determined by the value of `LevelStep`, unless the optional third argument, `n` (number of contour levels) or `V` (vector specifying contour levels) is provided. If `n` or `V` is used in combination with the `LevelStep` parameter, then the `LevelStep` parameter is ignored. If `n`, `V`, and the `LevelStep` parameter are all omitted, `contourm` selects a uniform step automatically.
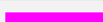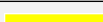
LineColor — Contour line colors
`'flat'` (default) | RGB triplet | `'r'` | `'g'` | `'b'` | `'none'` | ...

Contour line colors, specified as the comma-separated pair consisting of `'LineColor'` and `'flat'`, a color name, a short name, an RGB triplet, or `'none'`. The default value of `'flat'` selects a distinct color for lines at each contour level from the colormap of the figure in which the contours are drawn. The value `'none'` applies no color to the lines.

For a custom color, specify an RGB triplet. An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.

Alternatively, you can specify some common colors by name. This table lists the named color options and equivalent RGB triplets.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

LineStyle — Line style for contour lines
`'-'` (default) | `'--'` | `':'` | `'-.'` | `'none'`

Line style for contour lines, specified as the comma-separated pair consisting of `'LineStyle'` and `'-'` (solid), `'--'` (dashed), `':'` (dotted), `'-.'` (dash-dot), or `'none'`. The specifiers work the same as for line objects in MATLAB graphics.

LineWidth — Width of contour lines in points
`0.5` (default) | numeric scalar

Width of contour lines in points, specified as the comma-separated pair consisting of `'LineWidth'` and a numeric scalar. 1 point = 1/72 inch.

ShowText — Flag to display labels on contour lines
`'off'` (default) | `'on'`

Flag to display labels on contour lines, specified as the comma-separated pair consisting of `'ShowText'` and `'off'` or `'on'`. If you set `ShowText` to `'on'`, `contourm` displays text labels on each contour line indicating the value of the corresponding contour level. Another way to add labels to your contour lines is to call `clabelm` after calling `contourm`.

Output Arguments

C — Contour matrix
numeric matrix

Standard contour matrix, returned as a matrix with two rows. The first row represents longitude data and the second row represents latitude data.

h — Handle of contour patches
hggroup

Handle to the contour patches drawn onto the current axes, returned as an `hggroup`.

Tips

- You have three ways to control the number of contour levels that display in your map:

  1 Set the number of contour levels by specifying the scalar n in the syntax `contourm(Z,R,n)` or `contourm(lat,lon,Z,n)`.
  2 Use the vector V to specify the levels at which contours are drawn with the syntax `contourm(Z,R,V)` or `contourm(lat,lon,Z,V)`.
  3 Choose regular intervals at which the contours are drawn by setting the `LevelStep` parameter.

  If you do not use any of the above methods to set your contour levels, the `contourm` function displays around five contour levels.

See Also
clabelm | clegendm | contour | contourc | contour3 | contour3m | contourfm | geoshow | plot

Introduced before R2006a

convertlat

Convert between geodetic and auxiliary latitudes

Syntax

```
latout = convertlat(ellipsoid,latin,from,to,units)
```

Description

`latout = convertlat(ellipsoid,latin,from,to,units)` converts latitude values in `latin` from type `from` to type `to`. `ellipsoid` is a `referenceSphere`, `referenceEllipsoid`, or `oblateSpheroid` object, or a vector of the form `[semimajor_axis eccentricity]`.

`latin` is an array of input latitude values. `from` and `to` are each one of the latitude types listed below:

Latitude Type	Description
geodetic	The geodetic latitude is the angle that a line perpendicular to the surface of the ellipsoid at the given point makes with the equatorial plane.
authalic	The authalic latitude maps an ellipsoid to a sphere while preserving surface area. Authalic latitudes are used in place of the geodetic latitudes when projecting the ellipsoid using an equal area projection.
conformal	The conformal latitude maps an ellipsoid conformally onto a sphere. Conformal latitudes are used in place of the geodetic latitudes when projecting the ellipsoid with a conformal projection.
geocentric	The geocentric latitude is the angle that a line connecting a point on the surface of the ellipsoid to its center makes with the equatorial plane.
isometric	The isometric latitude is a nonlinear function of the geodetic latitude.
parametric	The parametric latitude of a point on the ellipsoid is the latitude on a sphere of radius a, where a is the semimajor axis of the ellipsoid, for which the parallel has the same radius as the parallel of geodetic latitude.
rectifying	The rectifying latitude is used to map an ellipsoid to a sphere in such a way that distance is preserved along meridians.

`latin` has the angle units specified by `units`: either `'degrees'` or `'radians'`. The output array, `latout`, has the same size and units as `latin`.

To properly project rectified latitudes, the radius must also be scaled to ensure the equal meridional distance property. This is accomplished by `rsphere`.

Examples

```
% Plot the difference between the auxiliary latitudes
% and geocentric latitude, from equator to pole,
% using the GRS 80 ellipsoid. Avoid the polar region with
```

```
% the isometric latitude, and scale down the difference
% by a factor of 200.
grs80 = referenceEllipsoid('grs80');
geodetic = 0:2:90;
authalic = ...
convertlat(grs80,geodetic,'geodetic','authalic', 'deg');
conformal = ...
convertlat(grs80,geodetic,'geodetic','conformal', 'deg');
geocentric = ...
convertlat(grs80,geodetic,'geodetic','geocentric','deg');
parametric = ...
convertlat(grs80,geodetic,'geodetic','parametric','deg');
rectifying = ...
convertlat(grs80,geodetic,'geodetic','rectifying','deg');
isometric = ...
convertlat(grs80,geodetic(1:end-5), ...
'geodetic','isometric','deg');
plot(geodetic, (authalic - geodetic),...
geodetic, (conformal - geodetic),...
geodetic, (geocentric - geodetic),...
geodetic, (parametric - geodetic),...
geodetic, (rectifying - geodetic),...
geodetic(1:end-5), (isometric - geodetic(1:end-5))/200);
title('Auxiliary Latitudes vs. Geodetic')
xlabel('geodetic latitude, degrees')
ylabel('departure from geodetic, degrees');
legend('authalic','conformal','geocentric', ...
'parametric','rectifying', 'isometric/200',...
'Location','NorthWest');
```
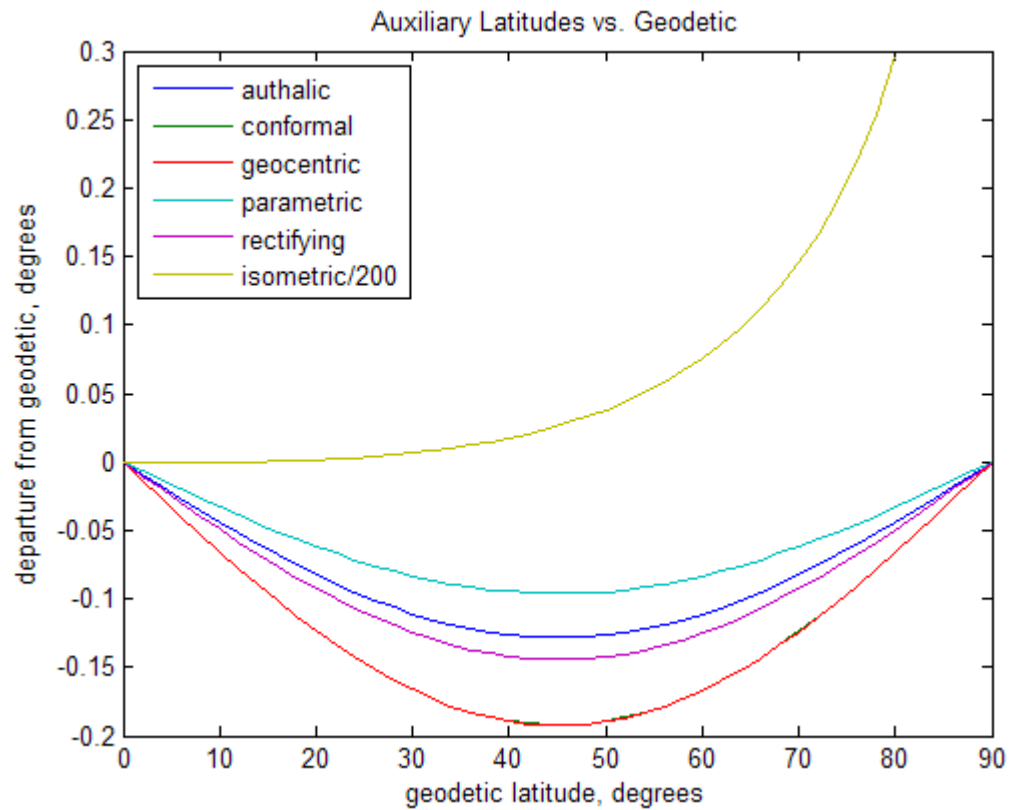
See Also
referenceEllipsoid | referenceSphere | oblateSpheroid | rsphere

Introduced before R2006a

crossfix

Cross-fix positions from bearings and ranges

Syntax

```
[newlat,newlon] = crossfix(lat,lon,az)
[newlat,newlon] = crossfix(lat,lon,az_range,case)
[newlat,newlon] = crossfix(lat,lon,az_range,case,drlat,drlon)
[newlat,newlon] = crossfix(lat,lon,az,units)
[newlat,newlon] = crossfix(lat,lon,az_range,case,units)
[newlat,newlon] = crossfix(lat,lon,az_range,drlat,drlon,units)
[newlat,newlon] = crossfix(lat,lon,az_range,case,drlat,drlon,units)
mat = crossfix(...)
```

Description

`[newlat,newlon] = crossfix(lat,lon,az)` returns the intersection points of all pairs of great circles passing through the points given by the column vectors `lat` and `lon` that have azimuths `az` at those points. The outputs are two-column matrices `newlat` and `newlon` in which each row represents the two intersections of a possible pairing of the input great circles. If there are *n* input objects, there will be *n choose 2* pairings.

`[newlat,newlon] = crossfix(lat,lon,az_range,case)` allows the input `az_range` to specify either azimuths or ranges. Where the vector `case` equals 1, the corresponding element of `az_range` is an azimuth; where `case` is 0, `az_range` is a range. The default value of `case` is a vector of ones (azimuths).

`[newlat,newlon] = crossfix(lat,lon,az_range,case,drlat,drlon)` resolves the ambiguities when there is more than one intersection between two objects. The scalar-valued `drlat` and `drlon` provide the location of an estimated (dead reckoned) position. The outputs `newlat` and `newlon` are column vectors in this case, returning only the intersection closest to the estimated point. When this option is employed, if any pair of objects fails to intersect, no output is returned and the warning `No Fix` is displayed.

`[newlat,newlon] = crossfix(lat,lon,az,units)`, `[newlat,newlon] = crossfix(lat,lon,az_range,case,units)`, `[newlat,newlon] = crossfix(lat,lon,az_range,drlat,drlon,units)`, and `[newlat,newlon] = crossfix(lat,lon,az_range,case,drlat,drlon,units)` allow the specification of the angle units to be used for all angles and ranges, where `units` is any valid angle units value. The default value of `units` is `'degrees'`.

`mat = crossfix(...)` returns the output in a two- or four-column matrix `mat`.

This function calculates the points of intersection between a set of objects taken in pairs. Given great circle azimuths and/or ranges from input points, the locations of the possible intersections are returned. This is different from the navigational function `navfix` in that `crossfix` uses great circle measurement, while `navfix` uses rhumb line azimuths and nautical mile distances.

Examples

Find intersections of points on circles

This example shows how to find the intersection of points on circles.

Create map axes.

```
figure('color','w');
ha = axesm('mapproj','mercator', ...
    'maplatlim',[-10 15],'maplonlim',[-10 20],...
    'MLineLocation',5,'PLineLocation',5);
axis off
gridm on
framem on
mlabel on
plabel on
```
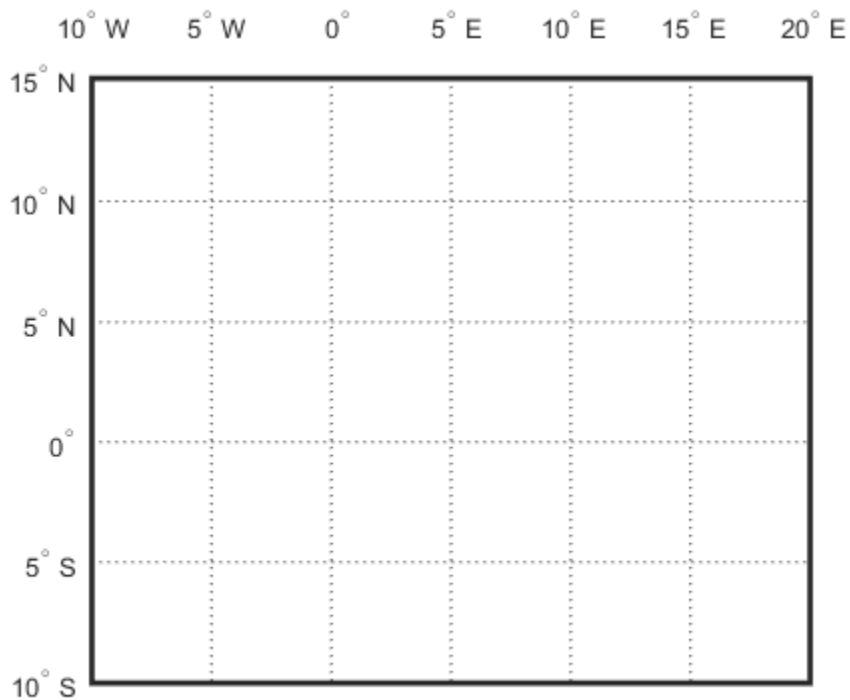

Define latitudes and longitudes of three arbitrary points, and then define three radii, all 8 degrees.

```
latpts = [0;5;0];
lonpts = [0;5;10];
radii = [8;8;8];
```

Obtain the intersections of imagined small circles around these points.

```
[newlat,newlon] = crossfix(latpts,lonpts,radii,[0;0;0])
```

```
newlat = 3×2

    7.5594   -2.5744
    6.2529   -6.2529
    7.5594   -2.5744


newlon = 3×2

   -2.6260    7.5770
    5.0000    5.0000
   12.6260    2.4230
```

Draw red circle markers at the given points.

```
geoshow(latpts,lonpts,'DisplayType','point',...
    'markeredgecolor','r','markerfacecolor','r','marker','o')
```
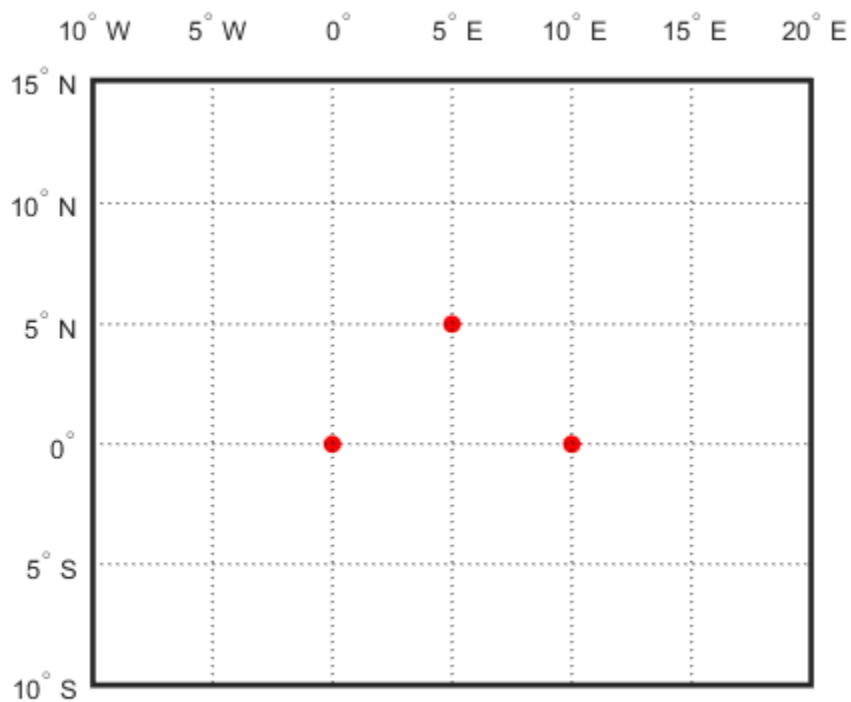

Draw magenta diamond markers at the points of intersection.

```
geoshow(reshape(newlat,6,1),reshape(newlon,6,1),'DisplayType','point',...
    'markeredgecolor','m','markerfacecolor','m','marker','d')
```
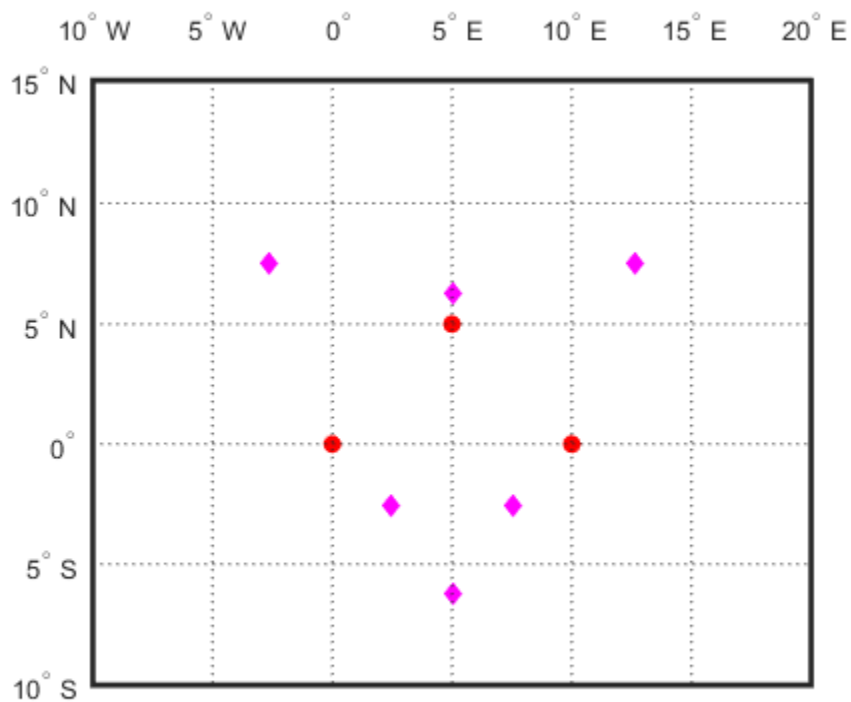
Generate a small circle 8 deg radius for each original point.

```
[latc1,lonc1] = scircle1(latpts(1),lonpts(1),radii(1));
[latc2,lonc2] = scircle1(latpts(2),lonpts(2),radii(2));
[latc3,lonc3] = scircle1(latpts(3),lonpts(3),radii(3));
```

Plot the small circles to show the intersections are as determined.

```
geoshow(latc1,lonc1,'DisplayType','line',...
    'color','b','linestyle','-')
geoshow(latc2,lonc2,'DisplayType','line',...
    'color','b','linestyle','-')
geoshow(latc3,lonc3,'DisplayType','line',...
    'color','b','linestyle','-')
```
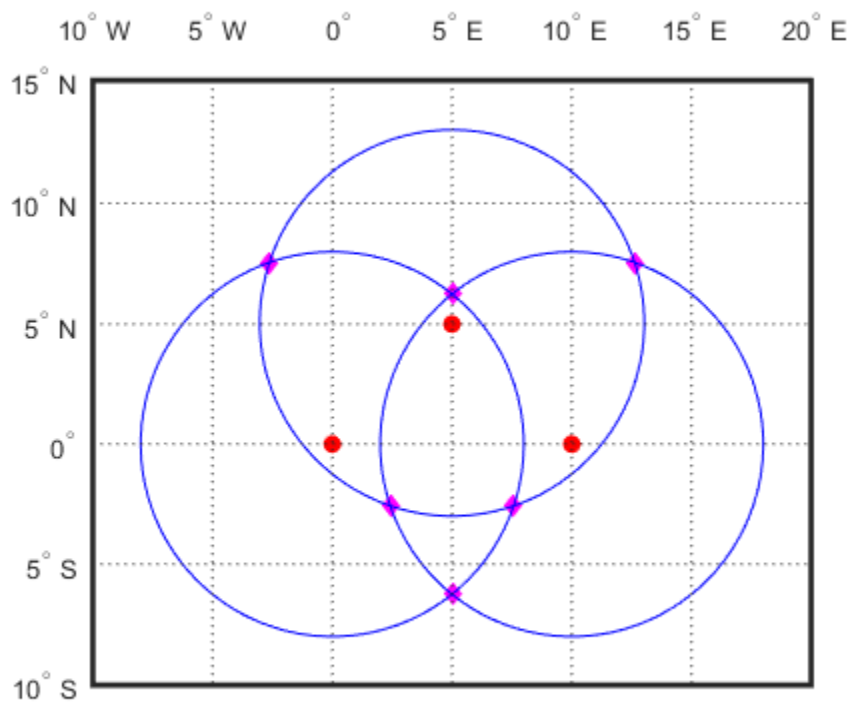
Find Intersection when provided with dead reckoning position

Find intersection when a dead reckoning position is provided, (0º,5ºE). `crossfix` returns one from each pair (the closest one).

```
[newlat,newlon] = crossfix([0 5 0]',[0 5 10]',...
                            [8 8 8]',[0 0 0]',0,5)

newlat =

    -2.5744
     6.2529
    -2.5744

newlon =

     7.5770
     5.0000
     2.4230
```

See Also
gcxgc | gcxsc | scxsc | rhxrh | polyxpoly | navfix

Introduced before R2006a

daspectm

Control vertical exaggeration in map display

Syntax

```
daspectm(zunits)
daspectm(zunits,vfac)
daspectm(zunits,vfac,lat,long)
daspectm(zunits,vfac,lat,long,az)
daspectm(zunits,vfac,lat,long,az,radius)
```

Description

daspectm(zunits) sets the 'DataAspectRatio' property of the map axes so that the *z*-axis is in proportion to the *x*-and *y*-projected coordinates. This permits elevation data to be displayed without vertical distortion. The zunits parameter specifies the units of the elevation data, and can be any length units recognized by unitsratio.

daspectm(zunits,vfac) sets the 'DataAspectRatio' property so that the *z*-axis is vertically exaggerated by the factor vfac. If omitted, the default is no vertical exaggeration.

daspectm(zunits,vfac,lat,long) sets the aspect ratio based on the local map scale at the specified geographic location. If omitted, the default is the center of the map limits.

daspectm(zunits,vfac,lat,long,az) also specifies the direction along which the scale is computed. If omitted, 90 degrees (west) is assumed.

daspectm(zunits,vfac,lat,long,az,radius) specifies the radius of the sphere. radius can be one of the values supported by km2deg, or it can be the (numerical) radius of the desired sphere in zunits. If omitted, the default radius of the Earth is used.

Examples

Display Elevation Map with Vertical Exaggeration

Load elevation data and a geographic cells reference object for the Korean peninsula. Create a world map with appropriate latitude and longitude limits, then display the data using meshm. Apply a colormap appropriate for elevation data using demcmap. Then, view the map in 3-D.

```
load korea5c
latlim = korea5cR.LatitudeLimits;
lonlim = korea5cR.LongitudeLimits;
worldmap(latlim,lonlim)
meshm(korea5c,korea5cR,korea5cR.RasterSize,korea5c)
demcmap(korea5c)
view(3)
```
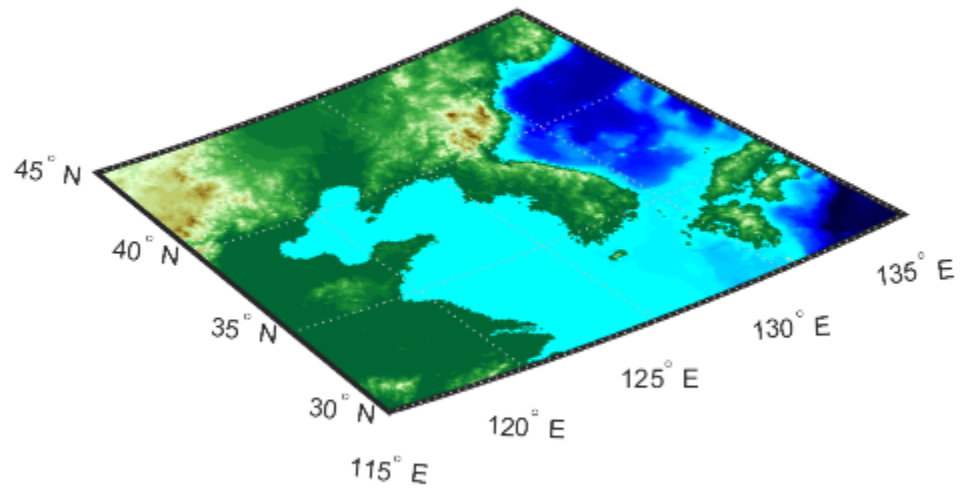
Set the vertical exaggeration factor to 30 using `daspectm`. Add light using `camlight`.

```
daspectm('m',30)
camlight
```
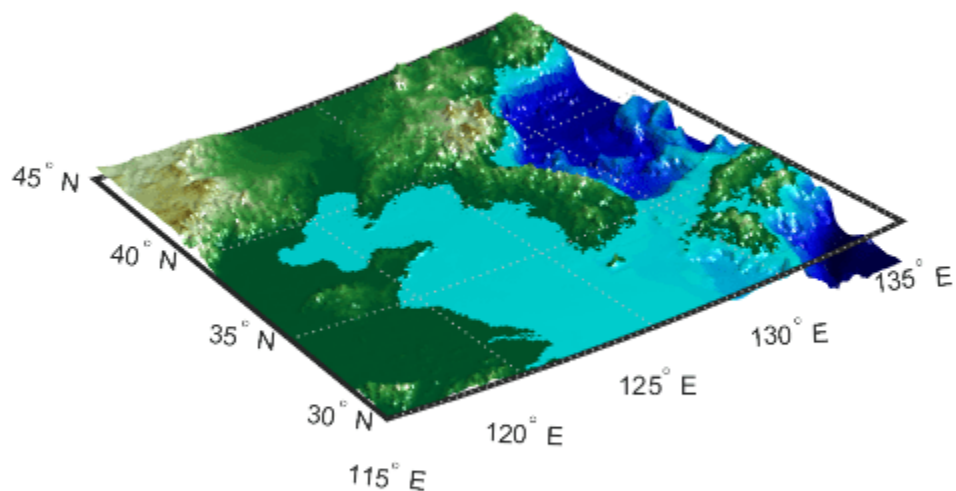
Limitations

The relationship between the vertical and horizontal coordinates holds only as long as the `geoid` or scale factor properties of the map axes remain unchanged. If you change the scaling between geographic coordinates and projected axes coordinates, execute `daspectm` again.

See Also
daspect | paperscale

Introduced before R2006a

dcwdata

Read selected DCW worldwide basemap data

Note dcwdata will be removed in a future release. The VMAP0 dataset has replaced DCW and can be accessed using vmap0data.

Syntax

```
struct = dcwdata(library,latlim,lonlim,theme,topolevel)
struct = dcwdata(devicename,library,...)
[struct1, struct2,...] = dcwdata(...,{topolevel1,topolevel2,...})
```

Description

struct = dcwdata(*library*,latlim,lonlim,theme,*topolevel*) reads data for the specified theme and topology level directly from the DCW CD-ROM. There are four CDs, one for each of the libraries: 'NOAMER' (North America), 'SASAUS' (Southern Asia and Australia), 'EURNASIA' (Europe and Northern Asia), and 'SOAMAFR' (South America and Africa). The desired theme is specified by a two-letter code. A list of valid codes is displayed when an invalid code, such as '?', is entered. The region of interest can be given as a point latitude and longitude or as a region with two-element vectors of latitude and longitude limits. The units of latitude and longitude are degrees. The data covering the requested region is returned, but will include data extending to the edges of the 5-by-5 degree tiles. The result is returned as a Version 1 Mapping Toolbox display structure.

struct = dcwdata(*devicename*,*library*,...) specifies the logical device name of the CD-ROM for computers that do not automatically name the mounted disk.

[struct1, struct2,...] = dcwdata(...,{*topolevel1*,*topolevel2*,...}) reads several topology levels. The levels must be specified as a cell array with the entries 'patch', 'line', 'point', or 'text'. Entering {'all'} for the topology level argument is equivalent to {'patch', 'line', 'point', 'text'}. Upon output, the data structures are returned in the output arguments by topology level in the same order as they were requested.

Background

The Digital Chart of the World (DCW) is a detailed and comprehensive source of publicly available global vector data. It was digitized from the Operational Navigation Charts (scale 1:1,000,000) and Jet Navigation Charts (1:2,000,000), compiled by the U.S. Defense Mapping Agency (DMA) along with mapping agencies in Australia, Canada, and the United Kingdom. The digitized data was published on four CD-ROMS by the DMA and is distributed by the U.S. Geological Survey (USGS).

The DCW is out of print and has been succeeded by the Vector Map Level 0 (VMAP0).

The DCW organizes data into 17 different themes, such as political/oceans (PO), drainage (DN), roads (RD), or populated places (PP). The data is further tiled into 5-by-5 degree tiles and separated by topology level (patches, lines, points, and text).

Examples

On a Macintosh computer,

```
s = dcwdata('NOAMER',41,-69,'?','patch');

??? Error using ==> dcwdata
Theme not present in library NOAMER
Valid two-letter theme identifiers are:
PO: Political/Oceans
PP: Populated Places
LC: Land Cover
VG: Vegetation
RD: Roads
RR: Railroads
UT: Utilities
AE: Aeronautical
DQ: Data Quality
DN: Drainage
DS: Supplemental Drainage
HY: Hypsography
HS: Supplemental Hypsography
CL: Cultural Landmarks
OF: Ocean Features
PH: Physiography
TS: Transportation Structure
POpatch = dcwdata('NOAMER',[41 44],[-72 -69],'PO','patch')
POpatch =
1x234 struct array with fields:
    type
    otherproperty
    tag
    altitude
    lat
    long
    tag2
    tag3
```

On an MS-DOS based operating system with the CD-ROM as the `d:` drive,

```
[RDtext,RDline] = dcwdata('d:','SASAUS',[-48 -34],[164 180],...
    'RD',{'text','line'});
```

On a UNIX® operating system with the CD-ROM mounted as `\cdrom`,

```
[POpatch,POline,POpoint,POtext] = dcwdata('\cdrom',...
    'EURNASIA',-48 ,164,'PO',{'all'});
```

Tips

Latitudes and longitudes use WGS84 as a horizontal datum. Elevations are in feet above mean sea level. The data set does not contain bathymetric data.

Some DCW themes do not contain all topology levels. In those cases, empty matrices are returned.

The data is tagged with object descriptions. Some data is provided with alternate tags in `tag2` and `tag3` fields. These alternate tags contain information that supplements the standard tag, such as the

names of political entities or values of elevation. The `tag2` field generally has the actual values or codes associated with the data. If the information in the `tag2` field expands to more verbose descriptions, these are provided in the `tag3` field.

Point data for which there are descriptions of both the type and the individual names of objects is returned twice within the structure. The first set is a collection of points of the same type with appropriate tag. The second is a set of individual points with the tag `'Individual Points'` and the name of the object in the `tag2` field.

Patches are broken at the tile boundaries. Setting the `EdgeColor` to `'none'` and plotting the lines gives the map a normal appearance.

The DCW was published in 1992 based on data compiled some years earlier. The political boundaries do not reflect recent changes such as the dissolution of the Soviet Union, Czechoslovakia, and Yugoslavia. In some cases, the boundaries of the successor nations are present as lower level political units. A new version, called VMAP0.

For information about the format of display structures, see "Version 1 Display Structures" on page 1-286 in the reference page for `displaym`.

References

[1] U.S. National Geospatial Intelligence Agency. *Military Specification: Digital Chart of the World (DCW)*. MIL-D-89009. April 13, 1992.

See Also

dcwgaz | dcwread | dcwrhead | displaym | extractm | mlayers | updategeostruct | vmap0data

Introduced before R2006a

dcwgaz

Search DCW worldwide basemap gazette file

Note dcwgaz will be removed in a future release. The VMAP0 dataset has replaced DCW and can be explored using vmap0ui.

Syntax

```
dcwgaz(library,object)
dcwgaz(devicename,library,object)
mtextstruc = dcwgaz(...)
[mtextstruc,mpointstruc] = dcwgaz(...)
```

Description

dcwgaz(library,object) searches the DCW library for items beginning with the character vector object . There are four CDs, one for each of the libraries: 'NOAMER' (North America), 'SASAUS' (Southern Asia and Australia), 'EURNASIA' (Europe and Northern Asia), and 'SOAMAFR' (South America and Africa). Items that exactly match or begin with the *object* are displayed on screen.

dcwgaz(*devicename*,*library*,*object*) specifies the logical device name of the CD-ROM for computers that do not automatically name the mounted disk.

mtextstruc = dcwgaz(...) displays the matched items on screen and returns a Mapping Toolbox display structure with the matches as text entries.

[mtextstruc,mpointstruc] = dcwgaz(...) returns the matches in structures formatted both as text and as points.

Background

In addition to the geographic data, the Digital Chart of the World (DCW) also includes an extensive gazette feature. The gazette is a collection of the names of geographic items mentioned in the various themes of a DCW disk. One DCW disk can contain about 10,000 to 15,000 names. This function allows you to search the gazette for names beginning with a particular character vector.

Examples

On a Macintosh computer,

```
s = dcwgaz('EURNASIA','apatin')

APATIN
s =
               type: 'text'
    otherproperty: {1x2 cell}
               tag: 'Built up area'
            string: 'APATIN'
```

```
        altitude: []
             lat: 45.6660
            long: 18.9830
```

On a UNIX operating system with the CD-ROM mounted as `'\cdrom'`,

```
[mtextstruc,mpointstruc] = ...
    dcwgaz('\cdrom','SOAMAFR', 'cape good')

Cape Goodenough
Cape Goodenough
Cape Goodenough
mtextstruc =
1x3 struct array with fields:
    type
    otherproperty
    tag
    string
    altitude
    lat
    long
mpointstruc =
1x3 struct array with fields:
    type
    otherproperty
    tag
    string
    altitude
    lat
    long
```

Tips

The search is not case sensitive. Items that match are those that begin with the *object* character vector. Spaces are significant.

See Also

dcwdata | dcwread | dcwrhead | mlayers | updategeostruct

Introduced before R2006a

dcwread

Read DCW worldwide basemap file

Note dcwread will be removed in a future release. The VMAP0 dataset has replaced DCW and can be read using vmap0read.

Syntax

```
dcwread(filepath,filename)
dcwread(filepath,filename,recordIDs)
dcwread(filepath,filename,recordIDs,field,varlen)
struc = dcwread(...)
[struc,field] = dcwread(...)
[struc,field,varlen] = dcwread(...)
[struc,field,varlen,description] = dcwread(...)
[struc,field,varlen,description,narrativefield] = dcwread(...)
```

Description

dcwread reads a DCW file. The user selects the file interactively.

dcwread(*filepath*,*filename*) reads the specified file. The combination [*filepath filename*] must form a valid complete file name.

dcwread(*filepath*,*filename*,recordIDs) reads selected records or fields from the file. If recordIDs is a scalar or a vector of integers, the function returns the selected records. If recordIDs is a cell array of integers, all records of the associated fields are returned.

dcwread(*filepath*,*filename*,recordIDs,field,varlen) uses previously read field and variable-length record information to skip parsing the file header (see below).

struc = dcwread(...) returns the file contents in a structure.

[struc,field] = dcwread(...) returns the file contents and a structure describing the format of the file.

[struc,field,varlen] = dcwread(...) also returns a vector describing the fields that have variable-length records.

[struc,field,varlen,description] = dcwread(...) also returns, description, a character vector that describes the contents of the file.

[struc,field,varlen,description,narrativefield] = dcwread(...) also returns the name of the narrative file for the current file.

Background

The Digital Chart of the World (DCW) uses binary files in a variety of formats. This function determines the format of the file and returns the contents in a structure. The field names of this structure are the same as the field names in the DCW file.

Examples

The following examples use the Macintosh directory system and file separators for the path name:

```
s = dcwread('NOAMER:DCW:NOAMER:','GRT')
s =
                  ID: 1
           DATA_TYPE: 'GEO'
               UNITS: '014'
           ELLIPSOID: 'WGS 84'
    ELLIPSOID_DETAIL: 'A=6378137,B=6356752 Meters'
      VERT_DATUM_REF: 'MEAN SEA LEVEL'
     VERT_DATUM_CODE: '015'
         SOUND_DATUM: 'MEAN SEA LEVEL'
    SOUND_DATUM_CODE: '015'
      GEO_DATUM_NAME: 'WGS 84'
      GEO_DATUM_CODE: 'WGE'
     PROJECTION_NAME: 'DECIMAL DEGREES'

s = dcwread('NOAMER:DCW:NOAMER:AE:','INT.VDT')
s =
5x1 struct array with fields:
    ID
    TABLE
    ATTRIBUTE
    VALUE
    DESCRIPTION
for i = 1:length(s); disp(s(i)); end
              ID: 1
           TABLE: 'AEPOINT.PFT'
       ATTRIBUTE: 'AEPTTYPE'
           VALUE: 1
     DESCRIPTION: 'Active civil'

              ID: 2
           TABLE: 'AEPOINT.PFT'
       ATTRIBUTE: 'AEPTTYPE'
           VALUE: 2
     DESCRIPTION: 'Active civil and military'
              ID: 3
           TABLE: 'AEPOINT.PFT'
       ATTRIBUTE: 'AEPTTYPE'
           VALUE: 3
     DESCRIPTION: 'Active military'

              ID: 4
           TABLE: 'AEPOINT.PFT'
       ATTRIBUTE: 'AEPTTYPE'
           VALUE: 4
     DESCRIPTION: 'Other'
```

```
            ID: 5
         TABLE: 'AEPOINT.PFT'
     ATTRIBUTE: 'AEPTTYPE'
         VALUE: 5
   DESCRIPTION: 'Added from ONC when not available from DAFIF'
s = dcwread('NOAMER:DCW:NOAMER:AE:','AEPOINT.PFT',1)
s =
           ID: 1
    AEPTTYPE: 4
    AEPTNAME: 'THULE AIR BASE'
     AEPTVAL: 251
    AEPTDATE: '19900502000000000000'
    AEPTICAO: '1261'
    AEPTDKEY: 'BR17652'
     TILE_ID: 94
      END_ID: 1

s = dcwread('NOAMER:DCW:NOAMER:AE:','AEPOINT.PFT',{1,2})
s =
4678x1 struct array with fields:
    ID
    AEPTTYPE
```

Tips

This function reads all DCW files except index files (files with names ending in `'X'`), thematic index files (files with names ending in `'TI'`), and spatial index files (files with names ending in `'SI'`).

File separators are platform dependent. The *filepath* input must use appropriate file separators, which you can determine using the MATLAB `filesep` function.

See Also
dcwdata | dcwgaz | dcwrhead

Introduced before R2006a

dcwrhead

Read DCW worldwide basemap file headers

Note dcwrhead will be removed in a future release. The VMAP0 dataset has replaced DCW and the header data can be read using vmap0rhead.

Syntax

```
dcwrhead
dcwrhead(filepath,filename)
dcwrhead(filepath,filename,fid)
dcwrhead(...)
hdr = dcwrhead(...)
```

Description

dcwrhead allows the user to select the header file interactively.

dcwrhead(*filepath*,*filename*) reads from the specified file. The combination [filepath filename] must form a valid complete file name.

dcwrhead(*filepath*,*filename*,fid) reads from the already open file associated with fid.

dcwrhead(...) with no output arguments displays the formatted header information on the screen.

hdr = dcwrhead(...) returns the DCW header as a character vector.

Background

The Digital Chart of the World (DCW) uses headers in most files to document the contents and format of that file. This function reads the header, displays a formatted version in the command window, or returns it as a character vector.

Examples

The following example uses the Macintosh file separators and path name:

```
dcwrhead('NOAMER:DCW:NOAMER:AE:','AEPOINT.PFT')
Aeronautical Points
AEPOINT.DOC
ID=I,         1,P,Row Identifier,-,-,
AEPTTYPE=I,    1,N,Airport Type,INT.VDT,-,
AEPTNAME=T,   50,N,Airport Name,-,-,
AEPTVAL=I,     1,N,Airport Elevation Value,-,-,
AEPTDATE=D,    1,N,Aeronautical Information Date,-,-,
AEPTICAO=T,    4,N,International Civil Organization Number,-,-,
AEPTDKEY=T,    7,N,DAFIF Reference Number,-,-,
TILE_ID=S,     1,F,Tile Reference Identifier,-,AEPOINT.PTI,
END_ID=I     1,F,Entity Node Primitive Foreign Key,-,-,
```

```
s = dcwrhead('NOAMER:DCW:NOAMER:AE:','AEPOINT.PFT')
s =
;Aeronautical Points;AEPOINT.DOC;ID=I,          1,P,Row
Identifier,-,-,:AEPTTYPE=I,     1,N,Airport
Type,INT.VDT,-,:AEPTNAME=T,    50,N,Airport Name,-,-,:AEPTVAL=I,
1,N,Airport Elevation Value,-,-,:AEPTDATE=D,     1,N,Aeronautical
Information Date,-,-,:AEPTICAO=T,     4,N,International Civil
Organization Number,-,-,:AEPTDKEY=T,     7,N,DAFIF Reference
Number,-,-,:TILE_ID=S,     1,F,Tile Reference
Identifier,-,AEPOINT.PTI,:END_ID=I     1,F,Entity Node Primitive
Foreign Key,-,-,:;
```

Tips

This function reads all DCW files except index files (files with names ending in `'X'`), thematic index files (files with names ending in `'TI'`), and spatial index files (files with names ending in `'SI'`).

File separators are platform dependent. The *filepath* input must use appropriate file separators, which you can determine using the MATLAB `filesep` function.

See Also
dcwdata | dcwgaz | dcwread

Introduced before R2006a

defaultm

Initialize or reset map projection structure

Syntax

```
mstruct = defaultm(projid)
mstruct = defaultm(mstruct)
```

Description

`mstruct = defaultm(projid)` initializes a map projection structure, where `projid` is a string scalar or character vector that matches one of the entries in the last column of the table displayed by the `maps` function. The output `mstruct` is a map projection structure. It is a scalar structure whose fields correspond to Map Axes Properties.

`mstruct = defaultm(mstruct)` checks an existing map projection structure, sets empty properties, and adjusts dependent properties. The `Origin`, `FLatLimit`, `FLonLimit`, `MapLatLimit`, and `MapLonLimit` properties may be adjusted for compatibility with each other and with the `MapProjection` property and (in the case of UTM or UPS) the `Zone` property.

With `defaultm`, you can construct a map projection structure (`mstruct`) that contains all the information needed to project and unproject geographic coordinates using , `projinv`, `projfwd`, `vfwdtran`, or `vinvtran` without creating a map axes or making any use at all of MATLAB graphics. Relevant parameters in the `mstruct` include the projection name, angle units, zone (for UTM or UPS), origin, aspect, false easting, false northing, and (for conic projections) the standard parallel or parallels. In very rare cases you might also need to adjust the frame limit (`FLatLimit` and `FLonLimit`) or map limit (`MapLatLimit` and `MapLonLimit`) properties.

You should make exactly two calls to `defaultm` to set up your `mstruct`, using the following sequence:

1 Construct a provisional version containing default values for the projection you've selected: `mstruct = defaultm(projection);`

2 Assign appropriate values to `mstruct.angleunits`, `mstruct.zone`, `mstruct.origin`, etc.

3 Set empty properties and adjust interdependent properties as needed to finalize your map projection structure: `mstruct = defaultm(mstruct);`

If you've set field `prop1` of `mstruct` to `value1`, field `prop2` to `value2`, and so forth, then the following sequence

```
mstruct = defaultm(projection);
mstruct.prop1 = value1;
mstruct.prop2 = value2;
...
mstruct = defaultm(mstruct);
```

produces exactly the same result as the following:

```
f = figure;
ax = axesm(projection, prop1, value1, prop2, value2, ...);
```

```
mstruct = getm(ax);
close(f)
```

but it avoids the use of graphics and is more efficient.

Note Angle-valued properties are in degrees by default. If you want to work in radians instead, you can make the following assignment in between your two calls to `defaultm`:

```
mstruct.angleunits = 'radians';
```

You must also use values in radians when assigning any angle-valued properties (such as `mstruct.origin`, `mstruct.parallels`, `mstruct.maplatlimit`, `mstruct.maplonlimit`, etc.).

See the Mapping Toolbox User's Guide section on "Work in UTM Without a Displayed Map" for information and an example showing the use of `defaultm` in combination with UTM.

Examples

Create Map Projection Structure

Create an empty map projection structure for a Mercator projection:

```
mstruct = defaultm('mercator')

mstruct =
     mapprojection: 'mercator'
              zone: []
        angleunits: 'degrees'
            aspect: 'normal'
       falseeasting: []
      falsenorthing: []
        fixedorient: []
              geoid: [1 0]
        maplatlimit: []
        maplonlimit: []
        mapparallels: 0
         nparallels: 1
             origin: []
         scalefactor: []
            trimlat: [-86 86]
            trimlon: [-180 180]
              frame: []
              ffill: 100
         fedgecolor: [0 0 0]
         ffacecolor: 'none'
          flatlimit: []
         flinewidth: 2
          flonlimit: []
               grid: []
          galtitude: Inf
             gcolor: [0 0 0]
          glinestyle: ':'
          glinewidth: 0.5000
       mlineexception: []
```

```
                mlinefill: 100
               mlinelimit: []
            mlinelocation: []
             mlinevisible: 'on'
            plineexception: []
                plinefill: 100
               plinelimit: []
            plinelocation: []
             plinevisible: 'on'
                fontangle: 'normal'
                fontcolor: [0 0 0]
                 fontname: 'helvetica'
                 fontsize: 9
                fontunits: 'points'
               fontweight: 'normal'
              labelformat: 'compass'
            labelrotation: 'off'
               labelunits: []
            meridianlabel: []
            mlabellocation: []
            mlabelparallel: []
               mlabelround: 0
             parallellabel: []
            plabellocation: []
            plabelmeridian: []
               plabelround: 0
```

Now change the map origin to `[0 90 0]`, and fill in default projection parameters accordingly:

```
mstruct.origin = [0 90 0];
mstruct = defaultm(mstruct)

mstruct =
        mapprojection: 'mercator'
                 zone: []
            angleunits: 'degrees'
                aspect: 'normal'
          falseeasting: 0
         falsenorthing: 0
            fixedorient: []
                 geoid: [1 0]
            maplatlimit: [-86 86]
            maplonlimit: [-90 270]
            mapparallels: 0
             nparallels: 1
                 origin: [0 90 0]
            scalefactor: 1
                trimlat: [-86 86]
                trimlon: [-180 180]
                  frame: 'off'
                  ffill: 100
             fedgecolor: [0 0 0]
             ffacecolor: 'none'
              flatlimit: [-86 86]
             flinewidth: 2
              flonlimit: [-180 180]
                   grid: 'off'
              galtitude: Inf
```

```
          gcolor: [0 0 0]
      glinestyle: ':'
      glinewidth: 0.5
   mlineexception: []
        mlinefill: 100
       mlinelimit: []
    mlinelocation: 30
     mlinevisible: 'on'
   plineexception: []
        plinefill: 100
       plinelimit: []
    plinelocation: 15
     plinevisible: 'on'
        fontangle: 'normal'
        fontcolor: [0 0 0]
         fontname: 'Helvetica'
         fontsize: 10
        fontunits: 'points'
       fontweight: 'normal'
      labelformat: 'compass'
    labelrotation: 'off'
       labelunits: 'degrees'
    meridianlabel: 'off'
     mlabellocation: 30
     mlabelparallel: 86
        mlabelround: 0
    parallellabel: 'off'
     plabellocation: 15
     plabelmeridian: -90
         plabelround: 0
```

Project Coordinates Using Map Projection Structure

This example shows how to perform the same projection computations that are done within Mapping Toolbox display commands by calling the `defaultm` and `projfwd` functions.

Create an empty map projection structure for a Sinusoidal projection, using the `defaultm` function. The function returns an `mstruct` .

```
mstruct = defaultm('sinusoid');
```

Set the map limits for the `mstruct` . To populate the fields of the map projection structure and ensure the effects of property settings are properly implemented, call `defaultm` a second time.

```
mstruct.maplonlimit = [-150 -30];
mstruct.geoid = referenceEllipsoid('grs80','kilometers');
mstruct = defaultm(mstruct);
```

Load coastline data and trim it to the map limits.

```
load coastlines
[latt,lont] = maptriml(coastlat,coastlon, ...
    mstruct.maplatlimit,mstruct.maplonlimit);
```

Project the latitude and longitude vectors into plane coordinates with the Sinusoidal projection and display the result. The plot shows that the data are projected in the specified aspect.

```
[x,y] = projfwd(mstruct,latt,lont);
figure
plot(x,y)
axis equal
```
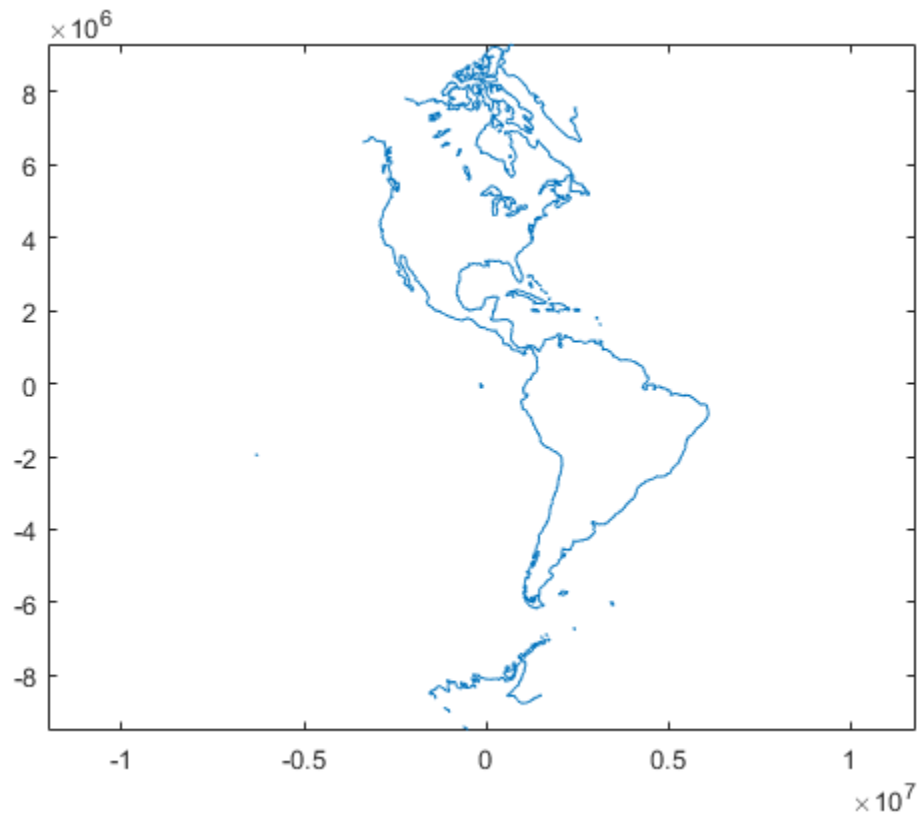

See Also

axesm | gcm | projinv | projfwd | setm

Introduced before R2006a

deg2km

Convert spherical distance from degrees to kilometers

Syntax

```
km = deg2km(deg)
km = deg2km(deg,radius)
km = deg2km(deg,sphere)
```

Description

`km = deg2km(deg)` converts distances from degrees to kilometers, as measured along a great circle on a sphere with a radius of 6371 km, the mean radius of the Earth.

`km = deg2km(deg,radius)` converts distances from degrees to kilometers, as measured along a great circle on a sphere having the specified radius.

`km = deg2km(deg,sphere)` converts distances from degrees to kilometers, as measured along a great circle on a sphere approximating an object in the Solar System.

Examples

Convert Degrees to Kilometers on Earth

Find the circumference of the Earth in kilometers. By default, the `deg2km` function assumes your input is a measurement of the Earth, which is represented as a sphere with a radius of 6371 kilometers.

```
km = deg2km(360)
```

```
km = 4.0030e+04
```

Convert Degrees to Kilometers on Mars

Find the circumference of Mars in kilometers.

```
km = deg2km(360,'mars')
```

```
km = 2.1300e+04
```

Input Arguments

deg — Distance in degrees
numeric array

Distance in degrees, specified as a numeric array.

Data Types: `single` | `double`

radius — Radius
6371 (default) | numeric scalar

Radius of sphere in units of kilometers, specified as a numeric scalar.

sphere — Sphere
`'sun'` | `'moon'` | `'mercury'` | `'venus'` | `'earth'` | ...

Sphere approximating an object in the Solar System, specified as one of the following values: `'sun'`, `'moon'`, `'mercury'`, `'venus'`, `'earth'`, `'mars'`, `'jupiter'`, `'saturn'`, `'uranus'`, `'neptune'`, or `'pluto'`. The value of `sphere` is case-insensitive.

Output Arguments

km — Distance in kilometers
numeric array

Distance in kilometers, returned as a numeric array.

Data Types: `single` | `double`

See Also
deg2nm | deg2sm | deg2rad | rad2deg | rad2km | km2deg

Introduced in R2007a

deg2nm

Convert spherical distance from degrees to nautical miles

Syntax

```
nm = deg2nm(deg)
nm = deg2nm(deg,radius)
nm = deg2nm(deg,sphere)
```

Description

`nm = deg2nm(deg)` converts distances from degrees to nautical miles, as measured along a great circle on a sphere with a radius of 3440.065 nautical miles, the mean radius of the Earth.

`nm = deg2nm(deg,radius)` converts distances from degrees to nautical miles, as measured along a great circle on a sphere having the specified radius.

`nm = deg2nm(deg,sphere)` converts distances from degrees to nautical miles, as measured along a great circle on a sphere approximating an object in the Solar System.

Examples

Convert Arc Length to Nautical Miles

One degree of arc length is about 60 nautical miles, using a spherical model of the Earth.

```
deg2nm(1)
```

```
ans =
   60.0405
```

This is not true on Mercury, of course.

```
deg2nm(1,'mercury')
```

```
ans =
   22.9852
```

Input Arguments

deg — Distance in degrees
numeric array

Distance in degrees, specified as a numeric array.

Data Types: `single` | `double`

radius — Radius
3440.065 (default) | numeric scalar

Radius of sphere in units of nautical miles, specified as a numeric scalar.

sphere — Sphere
`'sun'` | `'moon'` | `'mercury'` | `'venus'` | `'earth'` | …

Sphere approximating an object in the Solar System, specified as one of the following values: `'sun'`, `'moon'`, `'mercury'`, `'venus'`, `'earth'`, `'mars'`, `'jupiter'`, `'saturn'`, `'uranus'`, `'neptune'`, or `'pluto'`. The value of `sphere` is case-insensitive.

Output Arguments

nm — Distance in nautical miles
numeric array

Distance in nautical miles, returned as a numeric array.

Data Types: `single` | `double`

See Also

deg2km | deg2sm | deg2rad | rad2deg | rad2nm | nm2deg

Introduced in R2007a

deg2sm

Convert spherical distance from degrees to statute miles

Syntax

```
sm = deg2sm(deg)
sm = deg2sm(deg,radius)
sm = deg2sm(deg,sphere)
```

Description

`sm = deg2sm(deg)` converts distances from degrees to statute miles as measured along a great circle on a sphere with a radius of 3958.748 sm, the mean radius of the Earth.

`sm = deg2sm(deg,radius)` converts distances from degrees to statute miles as measured along a great circle on a sphere having the specified radius.

`sm = deg2sm(deg,sphere)` converts distances from degrees to statute miles, as measured along a great circle on a sphere approximating an object in the Solar System.

Examples

Convert Degrees to Statute Miles on Earth

Find the circumference of the Earth in statute miles. By default, the `deg2sm` function assumes your input is a measurement of the Earth, which is represented as a sphere with a radius of 3958.748 statute miles.

```
sm = deg2sm(360)
```

```
sm = 2.4874e+04
```

Convert Degrees to Statute Miles on Mars

Find the circumference of Mars in statute miles.

```
sm = deg2sm(360,'mars')
```

```
sm = 1.3235e+04
```

Input Arguments

deg — Distance in degrees
numeric array

Distance in degrees, specified as a numeric array.

Data Types: single | double

radius — Radius
3958.748 (default) | numeric scalar

Radius of sphere in units of statute miles, specified as a numeric scalar.

sphere — Sphere
'sun' | 'moon' | 'mercury' | 'venus' | 'earth' | …

Sphere approximating an object in the Solar System, specified as one of the following values: 'sun', 'moon', 'mercury', 'venus', 'earth', 'mars', 'jupiter', 'saturn', 'uranus', 'neptune', or 'pluto'. The value of sphere is case-insensitive.

Output Arguments

sm — Distance in statute miles
numeric array

Distance in statute miles, returned as a numeric array.

Data Types: single | double

See Also

deg2rad | rad2deg | deg2km | deg2nm | sm2deg | rad2sm

Introduced in R2007a

degrees2dm

Convert degrees to degrees-minutes

Syntax

```
DM = degrees2dm(angleInDegrees)
```

Description

`DM = degrees2dm(angleInDegrees)` converts angles from values in degrees which may include a fractional part (sometimes called "decimal degrees") to degrees-minutes representation.

Examples

Convert Angle in Degrees to Degree-Minutes

```
angleInDegrees = [ 30.8457722555556; ...
                  -82.0444189583333; ...
                   -0.504756513888889;...
                    0.004116666666667];
dm = degrees2dm(angleInDegrees)
```

dm = *4×2*

```
  30.0000    50.7463
 -82.0000     2.6651
        0   -30.2854
        0     0.2470
```

Input Arguments

angleInDegrees — Angle in degrees
n-element real-valued column vector

Angle in degrees, specified as an n-element real-valued column vector.

Output Arguments

DM — Angle in degrees-minutes representation
n-by-2 real-valued matrix

Angle in degrees-minutes representation, returned as an *n*-by-2 real-valued matrix. Each row specifies one angle, with the format `[D M]`:

- `D` contains the "degrees" element and is integer-valued.
- `M` contains the "minutes" element and may have a fractional part.

In any given row of DM, the sign of the first nonzero element indicates the sign of the overall angle. A positive number indicates north latitude or east longitude; a negative number indicates south latitude or west longitude. Any remaining element in that row has nonnegative values.

See Also

dm2degrees | deg2rad | degrees2dms | rad2deg

Topics
"Angle Representations and Angular Units"
"Angles as Binary and Formatted Numbers"

Introduced in R2007a

degrees2dms

Convert degrees to degrees-minutes-seconds

Syntax

```
DMS = degrees2dms(angleInDegrees)
```

Description

`DMS = degrees2dms(angleInDegrees)` converts angles from values in degrees which may include a fractional part (sometimes called "decimal degrees") to degrees-minutes-seconds representation.

Examples

Convert Angle in Degrees to Degree-Minute-Seconds

Convert an angle specified as a real-valued column vector to degrees-minutes-seconds representation. The output value is an *n*-by-3 real-valued matrix. Each row in the output specifies one angle, with the format `[Degrees Minutes Seconds]`.

```
angleInDegrees = [ 30.8457722555556; ...
                  -82.0444189583333; ...
                   -0.504756513888889; ...
                    0.004116666666667];
dms = degrees2dms(angleInDegrees)
```

dms = 4×3

```
  30.0000    50.0000    44.7801
 -82.0000     2.0000    39.9082
       0   -30.0000    17.1235
       0         0    14.8200
```

Customize Degree-Minute-Seconds Display Format

Convert angles in degrees to degree-minute-second representation. Display the result using the default display provided by `angl2str`.

```
angleInDegrees = [ 30.8457722555556; ...
                  -82.0444189583333; ...
                   -0.504756513888889; ...
                    0.004116666666667];
angl2str(angleInDegrees,'ns','degrees2dms')
```

ans = 4x25 char array
```
    ' 30^{\circ} 50' 44.78" N '
    ' 82^{\circ} 02' 39.91" S '
    '  0^{\circ} 30' 17.12" S '
```

```
'  0^{\circ} 00' 14.82" N '
```

Alternatively, specify a custom display format by converting the angles to degree-minute-second representation and using `sprintf`. The result is a single string.

```
dms = degrees2dms(angleInDegrees)

dms = 4×3

   30.0000   50.0000   44.7801
  -82.0000    2.0000   39.9082
        0  -30.0000   17.1235
        0         0   14.8200


nonnegative = all((dms >= 0),2);
hemisphere = repmat('N', size(nonnegative));
hemisphere(~nonnegative) = 'S';
absvalues = num2cell(abs(dms'));
values = [absvalues; num2cell(hemisphere')];
sprintf('%2.0fd:%2.0fm:%7.5fs:%s\n',values{:})

ans =
    '30d:50m:44.78012s:N
     82d: 2m:39.90825s:S
      0d:30m:17.12345s:S
      0d: 0m:14.82000s:N
     '
```

Input Arguments

`angleInDegrees` — Angle in degrees
n-element real-valued column vector

Angle in degrees, specified as an n-element real-valued column vector.

Output Arguments

DMS — Angle in degrees-minutes-seconds representation
n-by-3 real-valued matrix

Angle in degrees-minutes-seconds representation, returned as an *n*-by-3 real-valued matrix. Each row specifies one angle, with the format `[D M S]`:

- `D` contains the "degrees" element and is integer-valued.
- `M` contains the "minutes" element and is integer-valued.
- `S` contains the "seconds" element and may have a fractional part.

In any given row of `DMS`, the sign of the first nonzero element indicates the sign of the overall angle. A positive number indicates north latitude or east longitude; a negative number indicates south latitude or west longitude. Any remaining elements in that row will have nonnegative values.

See Also
dms2degrees | deg2rad | degrees2dm | rad2deg

Topics
"Angle Representations and Angular Units"
"Angles as Binary and Formatted Numbers"

Introduced in R2007a

degtorad

(Not recommended) Convert angles from degrees to radians

Note `degtorad` is not recommended. Use the `deg2rad` function instead. For more information, see "Compatibility Considerations".

Syntax

`angleInRadians = degtorad(angleInDegrees)`

Description

`angleInRadians = degtorad(angleInDegrees)` converts angle units from degrees to radians.

Examples

Convert Degrees to Radians

Convert 90 degrees to radians.

`angleInRadians = degtorad(90)`

```
angleInRadians =

    1.5708
```

Input Arguments

angleInDegrees — Angle in degrees
scalar | vector | matrix | N-D array

Angle in degrees, specified as a scalar, vector, matrix, or N-D array.

Compatibility Considerations

degtorad is not recommended
Not recommended starting in R2015b

The `degtorad` function is not recommended. Use the `deg2rad` function instead.

To update your code, replace instances of the `degtorad` function with the `deg2rad` function.

While the `degtorad` function accepts input arguments of integer types such as `uint8`, the `deg2rad` function accepts only single- and double-precision types. You can convert integer types to single- or double-precision types by using the `single` or `double` function.

See Also
deg2rad

Introduced in R2009b

demcmap

Colormaps appropriate to terrain elevation data

Syntax

```
demcmap(Z)
demcmap(Z,ncolors)
demcmap(Z,ncolors,cmapsea,cmapland)

demcmap('inc',Z,deltaz)
demcmap('inc',Z,deltaz,cmapsea,cmapland)

[cmap,climits] = demcmap( ___ )
```

Description

demcmap(Z) sets the colormap and color axis limits based on the elevation data limits derived from input argument Z.

- The default colormap assigns shades of green and brown for positive elevations, and various shades of blue for negative elevation values below sea level.
- The number of colors assigned to land and to sea are in proportion to the ranges in terrain elevation and bathymetric depth and total 64 by default. The color axis limits are computed such that the interface between land and sea maps to the zero elevation contour.
- The colormap is applied to the current figure and the color axis limits are applied to the current axes.

demcmap(Z,ncolors) creates a colormap of length ncolors.

demcmap(Z,ncolors,cmapsea,cmapland) assigns cmapsea and cmapland to elevations below and above sea level respectively.

demcmap('inc',Z,deltaz) chooses number of colors and color axis limits such that each color approximately represents the increment of elevation deltaz.

- The literal 'inc' signals demcmap that the first argument after Z will be deltaz.

demcmap('inc',Z,deltaz,cmapsea,cmapland) assigns cmapsea and cmapland to elevations below and above sea level respectively.

[cmap,climits] = demcmap(___) returns colormap cmap and color axis limit climits, using any of the above syntaxes, but does not apply them to figure or axes properties.

- Even if only one output argument is specified, no change occurs to figure or axes properties.

Examples

Displaying Elevation Data With Default Colormap

Load elevation raster data and a geographic cells reference object. Then, apply a colormap by specifying the maximum and minimum values of the data.

```
load topo60c
axesm hatano
meshm(topo60c,topo60cR)
zlimits = [min(topo60c(:)) max(topo60c(:))];
demcmap(zlimits)
colorbar
```
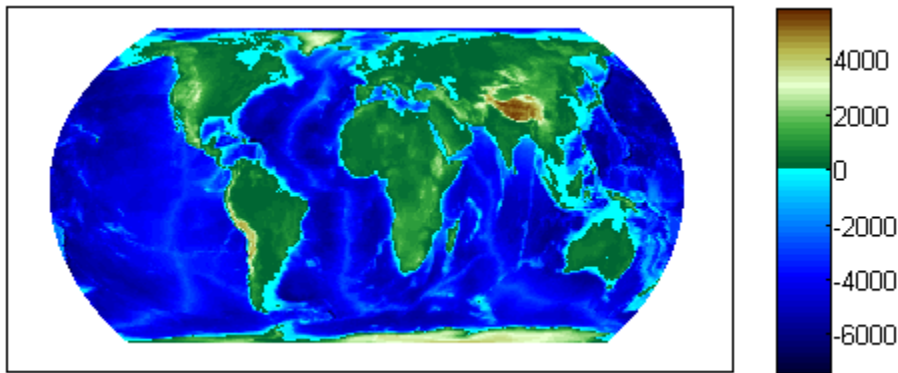

Defining Custom Land And Sea Colormaps

Custom RGB colormaps, for example `cmapssea` and `cmapland`, are used to populate figure colormaps by interpolation. The colors in each colormap map to the land and sea regions of the map. Fewer colors have been specified in total than the default number of 64. `demcmap` determines maximum and minimum elevation data limits internally as shown in the below example when the first argument is the elevation data grid.

```
load topo60c    % grid of elevation data
axesm hatano
meshm(topo60c,topo60cR)
cmapsea  = [.8  0 .8;  0 0 .8];
cmapland = [.7  0  0; .8 .8 0; 1  1 .8 ];
demcmap(topo60c,32,cmapsea,cmapland)
colorbar
```
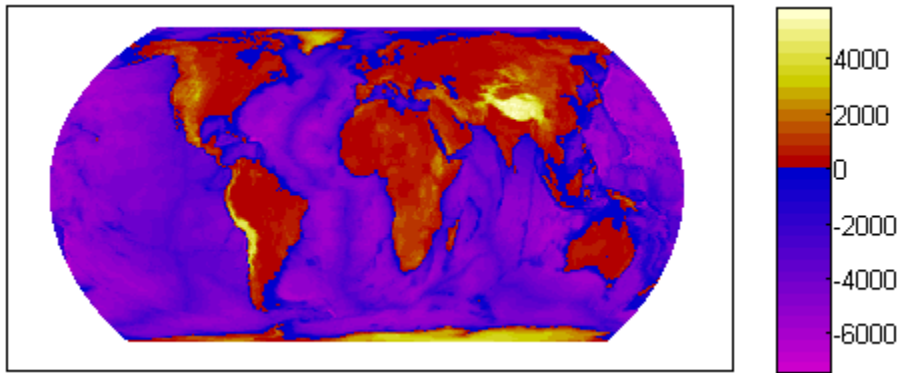
Colormap in Which Each Color Approximates a User Defined Increment

The following `demcmap` example controls the color quantization by choosing an optimal number of colors such that each color represents an elevation increment of approximately 2000 .

```
load topo60c
worldmap('world')
geoshow(topo60c,topo60cR,'DisplayType','texturemap')
demcmap('inc',[max(topo60c(:)) min(topo60c(:))],2000);
colorbar
```
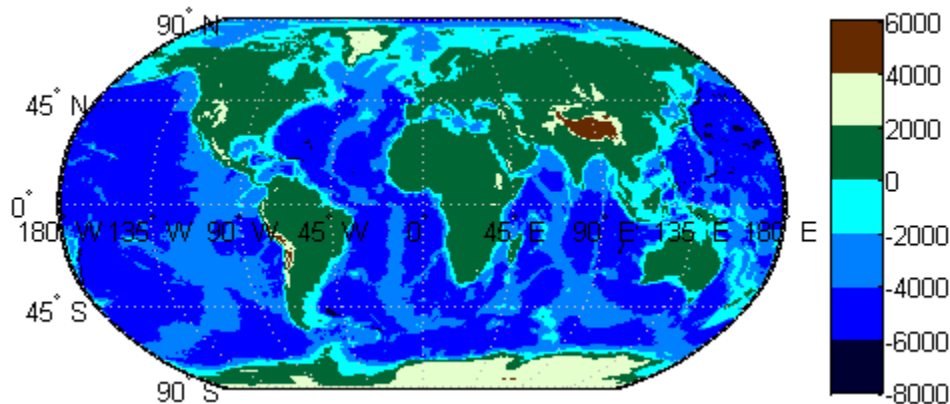

Input Arguments

Z — Terrain elevation limits
vector | matrix

Terrain elevation limits specified as a vector or matrix. If Z is a 2 element vector, then it specifies the minimum and maximum limits of terrain elevation data; ordering is not important. If Z is a matrix, then it specifies an elevation grid in which positive and negative values represent points above and below sea level respectively. The above two syntaxes for `demcmap` are identical in their effect on the figure colormap and axes properties.

Data Types: `single` | `double` | `int8` | `int32` | `uint8` | `uint16` | `uint32`

`ncolors` — Number of colors in colormap
64 (default) | scalar

Number of colors in the colormap specified as a scalar. It defines the number of rows `m` in the mx3 RGB matrix of the figure colormap.

Data Types: `double`

`cmapsea,cmapland` — RGB colormap matrices
matrix

- RGB colormaps specified as mx3 arrays containing any number of rows. The two colormaps need not be equal in length. They serve as the basis set for populating the figure colormap by interpolation.

- `cmapsea` and `cmapland` replace the default colormap. The default colormap for land or sea can be retained by providing an empty matrix in place of either colormap matrix.

That part of the figure colormap assigned to negative elevations is derived from `cmapssea`; `cmapland` plays a similar role for positive elevations.

Data Types: `double`

`deltaz` — Increment of elevation
scalar

The increment of elevation specified as a scalar. The color quantization of the default or user supplied colormap is adjusted such that each discrete color approximately represents a `deltaz` increment in elevation.

Data Types: `double`

Output Arguments

`cmap` — RGB colormap
matrix

RGB colormap returned as a matrix constructed for the figure colormap. Supply output arguments when you want to obtain the colormap and color axis limits without applying them automatically to the figure or axes properties. These properties remain unchanged even if only one output (`cmap`) is specified.

Data Types: `double`

`climits` — Color axis limits
vector

Color axis limits returned as a vector. `climits` may differ somewhat from those derived from input argument `Z` due to the quantization which results from fitting a limited number of colors over the range limit of the elevation data.

Supply output arguments when you want to obtain the colormap and color axis limits without applying them automatically to the figure or axes.

Data Types: `double`

Algorithms

If the elevation grid data contains both positive and negative values, then the computed colormap, `cmap`, has a "sea" partition of length `nsea` and "land" partition of length `nland`. The sum of `nsea` and `nland` equals the total number of entries in the computed colormap. The actual values of `nsea` and `nland` depend upon the number of entries and the relative range of the negative and positive limits of the elevation data. The sea partition consists of rows 1 through `nsea`, and the land partition consists of rows `nsea + 1` through `ncolors`. The sea and land partitions of the figure colormap are populated with colors interpolated from the basis RGB colormaps, `cmapsea` and `cmapland`. In the figure below, the sea and land 3x3 RGB colormaps shown are the default colors used by `demcmap` to populate the figure colormap when no user specified colormaps are provided.

demcmap([100 -100], 24)
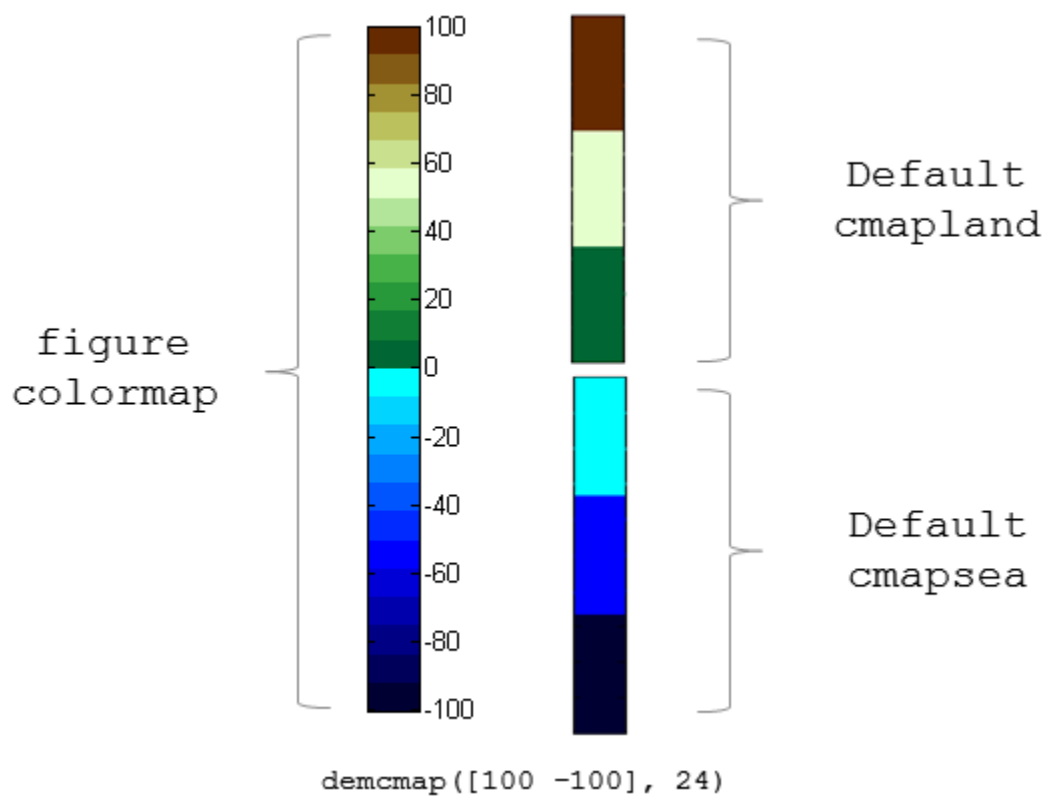
If the elevation grid data contains only positive or negative values, then the figure colormap is derived solely from the corresponding sea or land colormap.

See Also

`caxis` | `colormap` | `meshlsrm` | `meshm` | `surflsrm` | `surfm`

Introduced before R2006a

departure

Departure of longitudes at specified latitudes

Syntax

```
dist = departure(long1,long2,lat)
dist = departure(long1,long2,lat,ellipsoid)
dist = departure(long1,long2,lat,units)
dist = departure(long1,long2,lat,geoid,units)
```

Description

`dist = departure(long1,long2,lat)` computes the departure distance from `long1` to `long2` at the input latitude `lat`. Departure is the distance along a specific parallel between two meridians. The output `dist` is returned in degrees of arc length on a sphere.

`dist = departure(long1,long2,lat,ellipsoid)` computes the departure assuming that the input points lie on the ellipsoid defined by the input `ellipsoid`. `ellipsoid` is a `referenceSphere`, `referenceEllipsoid`, or `oblateSpheroid` object, or a vector of the form [`semimajor_axis eccentricity`].

`dist = departure(long1,long2,lat,units)` where `units` defines the angle units of the input and output data. In this form, the departure is returned as an arc length in the units specified by `units`. If `units` is omitted, `'degrees'` is assumed.

`dist = departure(long1,long2,lat,geoid,units)` is a valid calling form. In this case, the departure is computed in the same units as the semimajor axes of the ellipsoid.

Examples

Calculate Departure Distance on Sphere and Ellipsoid

On a spherical Earth, the departure distance is proportional to the cosine of the latitude. For example, calculate the departure distance for 0°.

```
distance = departure(0, 10, 0)
```

```
distance = 10
```

Now calculate the distance for 60°.

```
distance = departure(0, 10, 60)
```

```
distance = 5.0000
```

When you calculate the same departure distances on an ellipsoid, the result is more complicated. Again, calculate the departure distance for 0°.

```
distance = departure(0, 10, 0, referenceEllipsoid('earth', 'nm'))
```

```
distance = 601.0772
```

Now calculate the distance at 60°. You can see that the value is not exactly half the 0° value.

```
distance = departure(0, 10, 60, referenceEllipsoid('earth', 'nm'))
```

```
distance = 301.2959
```

More About

Departure

Departure is the distance along a parallel between two points. Whereas a degree of latitude is always the same distance, a degree of longitude is different in length at different latitudes. In practice, this distance is usually given in nautical miles.

See Also

distance | stdm

Introduced before R2006a

disp

Display geographic or planar vector

Syntax

```
disp(v)
```

Description

`disp(v)` prints the size of the geographic or planar vector `v`, and its properties and dynamic properties, if they exist.

If the command window is large enough, the values of the properties are also shown, otherwise only their size is shown. You can control the display of the numerical values using the `format` command.

Examples

Display a Geoshape Vector

Create a geoshape vector.

```
gs = geoshape(shaperead('worldcities.shp', 'UseGeoCoords', true));
```

Display the entire geoshape vector.

```
disp(gs)
```

```
 318x1 geoshape vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
 Vertex properties:
  (318 features concatenated with 317 delimiters)
     Latitude: [5.2985 NaN 24.6525 NaN 5.6106 NaN 37.0613 NaN 9.0235 NaN ... ]
    Longitude: [-3.9509 NaN 54.7589 NaN -0.2121 NaN 35.3894 NaN 38.7575 ... ]
 Feature properties:
         Name: {1x318 cell}
```

Display only the first two feature of the geoshape vector. Notice that the property values are shown because they are short enough to fit on the command window.

```
disp(gs(1:2));
```

```
 2x1 geoshape vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
 Vertex properties:
  (2 features concatenated with 1 delimiter)
```

```
        Latitude: [5.2985 NaN 24.6525]
       Longitude: [-3.9509 NaN 54.7589]
 Feature properties:
            Name: {'Abidjan'  'Abu Dhabi'}
```

Input Arguments

v — Geographic or planar vector to be displayed
geopoint, geoshape, mappoint, or mapshape objects

Geographic or planar vector to be displayed, specified as a `geopoint`, `geoshape`, `mappoint`, or `mapshape` object.

See Also

format | fieldnames | length | properties

Introduced in R2012a

disp

Display properties of WMS layers or capabilities

Syntax

```
disp(layers,Name,Value,...)
disp(capabilities)
```

Description

`disp(layers,Name,Value,...)` displays the index number followed by the property names and property values of the Web map service layers, `layers`. You can specify additional options using one or more `Name,Value` pair arguments.

`disp(capabilities)` displays the properties of the Web map service capabilities document, `capabilities`. The function removes hyperlinks and expands character vector and cell array properties.

Examples

Display Specific Properties of WMS Layers

Search the WMS Database for layers that contain `srtm30`. Display the `LayerTitle` and `LayerName` properties without an Index.

```
layers = wmsfind('srtm30');
disp(layers(1:5),'Index', 'off', ...
   'Properties',{'layertitle','layername'})

  5x1 WMSLayer

  Properties:
      LayerTitle: 'Estimated Seafloor Depth Gradients: srtm30plus (US West Coast) - magnitude_gra
       LayerName: 'erdSrtm30plusSeafloorGradient:magnitude_gradient'

      LayerTitle: 'Estimated Seafloor Depth Gradients: srtm30plus (US West Coast) - sea_floor_dep
       LayerName: 'erdSrtm30plusSeafloorGradient:sea_floor_depth'

      LayerTitle: 'Estimated Seafloor Depth Gradients: srtm30plus (US West Coast) - x_gradient'
       LayerName: 'erdSrtm30plusSeafloorGradient:x_gradient'

      LayerTitle: 'Estimated Seafloor Depth Gradients: srtm30plus (US West Coast) - y_gradient'
       LayerName: 'erdSrtm30plusSeafloorGradient:y_gradient'

      LayerTitle: 'SRTM30_PLUS Estimated Topography, 30 seconds, Global, v11 - z'
       LayerName: 'srtm30plus:z'
```

Your results might be different because the WMS Database changes each release.

Sort and Display Properties of WMS Layers

Sort and display the `LayerName` property with an index.

Search the WMS Database for layers that contain information about elevation.

```
layers = wmsfind('elevation');
```

Sort the layers.

```
[layerNames,index] = sort({layers.LayerName});
```

Display the first five results.

```
layers = layers(index);
disp(layers(1:5),'Label','off','Properties','layername')
```

```
  5x1 WMSLayer

  Properties:
          Index: 1
'0'

          Index: 2
'0'

          Index: 3
'133'

          Index: 4
'134'

          Index: 5
'141'
```

Your results might be different because the WMS Database changes each release.

Display WMS Capabilities

Create a `WMSCapabilities` object from the contents of a downloaded capabilities file from the NASA SVS Image Server.

```
nasa = wmsfind('NASA SVS Image', 'SearchField', 'servertitle');
serverURL = nasa(1).ServerURL;
server = WebMapServer(serverURL);
capabilities = server.getCapabilities;
```

Display the properties of the capabilities document.

```
disp(capabilities)
```

```
  WMSCapabilities

  Properties:
        ServerTitle: 'NASA SVS Image Server'
          ServerURL: 'https://svs.gsfc.nasa.gov/cgi-bin/wms?'
```

```
       ServiceName: 'WMS'
           Version: '1.3.0'
          Abstract: 'Web Map Server maintained by the Scientific Visualization Studio at NASA's
     OnlineResource: 'http://svs.gsfc.nasa.gov/'
 ContactInformation: [1x1 struct]
  AccessConstraints: 'none'
              Fees: 'none'
        KeywordList: {}
       ImageFormats: {'image/png'}
         LayerNames: {326x1 cell}
             Layer: [326x1 WMSLayer]
         AccessDate: '01-Apr-2021'

  Methods
```

Your results might be different because the WMS Database changes each release.

Input Arguments

`layers` — Layers to display
array of `WMSLayer` objects

Layers to display, specified as an array of `WMSLayer` objects.

`capabilities` — WMS capabilities document to display
`WMSCapabilities` object

WMS capabilities document to display, specified as a `WMSCapabilities` object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

You can abbreviate parameter names, and case does not matter.

Example: `'Properties',{'layertitle','layername'}`

`Properties` — Properties to display
`'all'` (default) | character vector or cell array of character vectors

Properties to display, specified as a character vector or cell array of character vectors. The properties are displayed in the same order as they are provided to `Properties`. Permissible values are: `'servertitle'`, `'servername'`, `'layertitle'`, `'layername'`, `'latlim'`, `'lonlim'`, `'abstract'`, `'coordrefsyscodes'`, `'details'`, or `'all'`. To list all the properties, set `'Properties'` to `'all'`.

Example: `{'coordrefsyscodes','latlim','lonlim'}`

`Label` — Flag to display property values
`'on'` (default) | `'off'`

Flag to display property values, specified as the character vector `'on'` or `'off'`. The value is case-insensitive. If you set `'Label'` to `'on'`, then the property name appears followed by its value. If you set `'Label'` to `'off'`, then only the property value appears in the output.

Example: `'off'`

Index — Flag to display property indices
`'on'` (default) | `'off'`

Flag to display property indices, specified as the character vector `'on'` or `'off'`. The value is case-insensitive. If you set `'Index'` to `'on'`, then `disp` lists the element's index in the output. If you set `'Index'` to `'off'`, then `disp` does not list the index value in the output.

Example: `'off'`

See Also
getCapabilities | wmsfind

Introduced in R2009b

displaym

Display geographic data from display structure

Syntax

```
displaym(displaystruct)
displaym(displaystruct,str)
displaym(displaystruct,strings)
displaym(displaystruct,strings,searchmethod)
h = displaym(displaystruct)
```

Description

displaym(displaystruct) projects the data contained in the input displaystruct, a Version 1 Mapping Toolbox display structure, in the current axes. The current axes must be a map axes with a valid map definition. See the remarks about "Version 1 Display Structures" on page 1-286 below for details on the contents of display structures.

displaym(displaystruct,str) displays the vector data elements of displaystruct whose 'tag' fields contains character vectors beginning with 'str'. Vector data elements are those whose 'type' field is either 'line' or 'patch'. The match is case-insensitive.

displaym(displaystruct,strings) displays the vector data elements of displaystruct whose 'tag' field matches with one of the elements (or rows) of strings. strings is a cell array of character vectors (or a 2-D character array). In the case of character array, trailing blanks are stripped from each row before matching.

displaym(displaystruct,strings,searchmethod) controls the method used to match the values of the tag field in displaystruct, as follows:

- 'strmatch' — Search for matches at the beginning of the tag
- 'findstr' — Search within the tag
- 'exact' — Search for exact matches

Note that when searchmethod is specified the search is case-sensitive.

h = displaym(displaystruct) returns handles to the graphic objects created by displaym.

Note The type of *display structure* accepted by displaym is not the same as a *geographic data structure* (geostructs and mapstructs). introduced in Mapping Toolbox Version 2. Use geoshow or mapshow instead of displaym to display geostructs or mapstructs—created using shaperead and gshhs, for example. For more information, see "Geographic Data Structures".

Tips

The following section documents the contents of display structures.

Version 1 Display Structures

A display structure is a MATLAB structure array with the following fields:

- A `tag` field names an individual feature or object
- A `type` field specifies a MATLAB graphics object type (`'line'`, `'patch'`, `'surface'`, `'text'`, or `'light'`) or has the value `'regular'`, specifying a regular data grid
- `lat` and `long` fields contain coordinate vectors of latitudes and longitudes, respectively
- An `altitude` field contains a vector of vertical coordinate values
- A `string` property contains text to be displayed if `type` is `'text'`
- MATLAB graphics properties are specified explicitly, on a per-feature basis, in an `otherproperty` field

The choice of options for the `type` field reveals that a display structure can contain

- Vector geodata (`type` is `'line'` or `'patch'`)
- Raster geodata (`type` is `'surface'` or `'regular'`)
- Graphic objects (`type` is `'text'` or `'light'`)

The following table indicates which fields are used in the six types of display structures:

Field Name	Type 'light'	Type 'line'	Type 'patch'	Type 'regular'	Type 'surface'	Type 'text'
type	•	•	•	•	•	•
tag	•	•	•	•	•	•
lat	•	•	•		•	•
long	•	•	•		•	•
map				•	•	
maplegend				•		
meshgrat				•		
string						•
altitude	•	•	•	•	•	•
otherproperty	•	•	•	•	•	•

Some fields can contain empty entries, but each indicated field must exist for the objects in the structure array to be displayed correctly. For instance, the `altitude` field can be an empty matrix and the `otherproperty` field can be an empty cell array.

The `type` field must be one of the specified map object types: `'line'`, `'patch'`, `'regular'`, `'surface'`, `'text'`, or `'light'`.

The `tag` field must be different from the `type` field usually containing the name or kind of map object. Its contents must not be equal to the name of the object type (i.e., line, surface, text, etc.).

The `lat`, `long`, and `altitude` fields can be scalar values, vectors, or matrices, as appropriate for the map object type.

The `map` field is a data grid. If `map` is a regular data grid, `maplegend` is its corresponding referencing vector, and `meshgrat` is a two-element vector specifying the graticule mesh size. If `map` is a geolocated data grid, `lat` and `long` are the matrices of latitude and longitude coordinates.

The `otherproperty` field is a cell array containing any additional display properties appropriate for the map object. Cell array entries can be a line specification, such as `'r+'`, or property name/ property value pairs, such as `'color','red'`. If the `otherproperty` field is left as an empty cell array, default colors are used in the display of lines and patches based on the `tag` field.

Note In some cases you can use the `geoshow` function as a direct alternative to `displaym`. It accepts display structures of type `line` and `patch`.

See Also

extractm | geoshow | mapshow | mlayers | updategeostruct

Introduced before R2006a

dist2str

Convert numeric distance values into text

Syntax

```
str = dist2str(distin)
str = dist2str(dist,format)
str = dist2str(dist,format,units)
str = dist2str(dist,format,digits)
str = dist2str(dist,format,units,n)
```

Description

`str = dist2str(distin)` converts a numerical vector of distances in kilometers, `distin`, to a character array. The output character array is useful for the display of distances. The purpose of this function is to convert distance-valued variables into text suitable for map display.

`str = dist2str(dist,format)` specifies the notation to be used for the character array in `format`. If blank or `'none'`, the result is a simple numerical representation (no indicator for positive distances, minus signs for negative distances). The only other format is `'pm'` (for *plus-minus*) prefixes a + for positive distances.

`str = dist2str(dist,format,units)` defines the units in which the input distances are supplied. Units must be one of the following: `'feet'`, `'kilometers'`, `'meters'`, `'nauticalmiles'`, `'statutemiles'`, `'degrees'`, or `'radians'`. Note that statute miles are encoded as `'mi'` in the character array, whereas in most Mapping Toolbox functions, `'mi'` indicates international miles. If omitted or blank, `'kilometers'` is assumed.

`str = dist2str(dist,format,digits)` or `str = dist2str(dist,format,units,n)` uses the input n to determine the number of decimal digits in the output matrix. If `n = -2`, the default, `dist2str` rounds to the nearest hundredth. If `n = 0`, `dist2str` rounds the output to the nearest integer. Note that this sign convention for n is opposite to the one used by the MATLAB `round` function.

Examples

Convert Vector of Numeric Values to Strings

Create a numeric vector.

```
d = [-3.7 2.95 87];
```

Convert the numeric values to strings.

```
str = dist2str(d,'none','km')
```

```
str = 3x8 char array
    '-3.70 km'
    ' 2.95 km'
```

```
'87.00 km'
```

Now change the units to nautical miles, add plus signs to positive values, and truncate to the tenths position.

```
str = dist2str(d,'pm','nm',-1)

str = 3x8 char array
    ' -3.7 nm'
    ' +3.0 nm'
    '+87.0 nm'
```

See Also
angl2str

Introduced before R2006a

distance

Distance between points on sphere or ellipsoid

Syntax

```
[arclen,az] = distance(lat1,lon1,lat2,lon2)
[arclen,az] = distance(lat1,lon1,lat2,lon2,ellipsoid)
[arclen,az] = distance(pt1,pt2)
[arclen,az] = distance(pt1,pt2,ellipsoid)
[arclen,az] = distance( ___ ,units)
[arclen,az] = distance(track, ___ )
```

Description

`[arclen,az] = distance(lat1,lon1,lat2,lon2)` computes the lengths, `arclen`, of the great circle arcs connecting pairs of points on the surface of a sphere. In each case, the shorter (minor) arc is assumed. The function can also compute the azimuths, `az`, of the second point in each pair with respect to the first (that is, the angle at which the arc crosses the meridian containing the first point).

`[arclen,az] = distance(lat1,lon1,lat2,lon2,ellipsoid)` computes geodesic arc length and azimuth assuming that the points lie on the reference ellipsoid defined by the input `ellipsoid`.

`[arclen,az] = distance(pt1,pt2)` accepts *N*-by-2 coordinate arrays of pairs of points, `pt1` and `pt2`, that store latitude coordinates in the first column and longitude coordinates in the second column.

This syntax is equivalent to `arclen = distance(pt1(:,1),pt1(:,2),pt2(:,1),pt2(:,2))`.

`[arclen,az] = distance(pt1,pt2,ellipsoid)` computes geodesic arc length and azimuth assuming that the points lie on the reference ellipsoid defined by the input `ellipsoid`.

`[arclen,az] = distance( ___ ,units)` also specifies the angle units of the latitude and longitude coordinates for any of the preceding syntaxes.

`[arclen,az] = distance(track, ___ )` also specifies whether the track is a great circle/geodesic or a rhumb line arc, for any of the preceding syntaxes.

Examples

Find Difference in Distance Along Two Tracks

Using `pt1,pt2` notation, find the distance from Norfolk, Virginia (37°N, 76°W), to Cape St. Vincent, Portugal (37°N, 9°W), just outside the Straits of Gibraltar. The distance between these two points depends upon the *track* value selected.

```
arclen = distance('gc',[37,-76],[37,-9])
```

```
arclen =

   52.3094
```

```
arclen = distance('rh',[37,-76],[37,-9])
```

```
arclen =

    53.5086
```

The difference between these two tracks is 1.1992 degrees, or about 72 nautical miles. This represents about 2% of the total trip distance. The tradeoff is that at the cost of those 72 miles, the entire trip can be made on a rhumb line with a fixed course of 90º, due east, while in order to follow the shorter great circle path, the course must be changed continuously.

On a meridian and on the Equator, great circles and rhumb lines coincide, so the distances are the same. For example,

```
% Great circle distance
arclen = distance(37,-76,67,-76)
```

```
arclen =

    30.0000
```

```
% Rhumb line distance
arclen = distance('rh',37,-76,67,-76)
```

```
arclen =

    30.0000
```

Input Arguments

lat1, lon1 — First set of latitude or longitude coordinates
numeric scalar | numeric array

First set of latitude or longitude coordinates, specified as a numeric scalar or numeric array. The coordinates are expressed in degrees unless units is specified as 'radians'.

Data Types: single | double

lat2, lon2 — Second set of latitude or longitude coordinates
numeric scalar | numeric array

Second set of latitude or longitude coordinates, specified as a numeric scalar or numeric array. The coordinates are expressed in degrees unless units is specified as 'radians'.

Data Types: single | double

pt1 — First set of point coordinates
N-by-2 numeric matrix

First set of point coordinates, specified as an *N*-by-2 numeric matrix. pt1 is equivalent to [lat1 lon1] when lat1 and lon1 are column vectors.

Data Types: single | double

pt2 — Second set of point coordinates
N-by-2 numeric matrix

Second set of point coordinates, specified as an *N*-by-2 numeric matrix. `pt2` is equivalent to `[lat2 lon2]` when `lat2` and `lon2` are column vectors.

Data Types: `single` | `double`

ellipsoid — Reference ellipsoid
`referenceSphere` object | `referenceEllipsoid` object | `oblateSpheroid` object | two-element vector

Reference ellipsoid, specified as an `referenceSphere`, `referenceEllipsoid`, or `oblateSpheroid` object, or a two-element vector of the form `[semimajor_axis eccentricity]`.

Example: `referenceEllipsoid('GRS80')`

Example: `[6378.137 0.0818191910428158]`

units — Angle units
`'degrees'` (default) | `'radians'`

Angle units of the latitude and longitude coordinates, specified as `'degrees'` or `'radians'`.

Data Types: `char`

track — Track
`'gc'` (default) | `'rh'`

Track, specified as one of the following character vectors.

- `'gc'` — Great circle distances are computed on a sphere and geodesic distances are computed on an ellipsoid.
- `'rh'` — Rhumb line distances are computed on either a sphere or ellipsoid.

Data Types: `char`

Output Arguments

arclen — Arc length
numeric scalar | numeric vector | numeric array

Arc length, returned as a numeric scalar or array of the same size as the input latitude and longitude arrays, `lat1`, `lon1`, `lat2` and `lon2`. When coordinates are specified using coordinate arrays `pt1` and `pt2`, then `arclen` is a numeric vector of length *N*.

- When `ellipsoid` is not specified, `arclen` is expressed in degrees or radians of arc, consistent with the value of `units`.
- When `ellipsoid` is specified, `arclen`, is expressed in the same length units as the semimajor axis of the ellipsoid.

Data Types: `double`

az — Azimuth
numeric scalar | numeric vector | numeric array

Azimuth of the second point in each pair with respect to the first, returned as a numeric scalar or array of the same size as the input latitude and longitude arrays, `lat1`, `lon1`, `lat2` and `lon2`. When

coordinates are specified using coordinate arrays `pt1` and `pt2`, then `arclen` is a numeric vector of length *N*. `az` is measured clockwise from north.

Data Types: `single` | `double`

Tips

- The size of nonscalar latitude and longitude coordinates, `lat1`, `lon1`, `lat2`, and `lon2`, must be consistent. When given a combination of scalar and array inputs, the `distance` function automatically expands scalar inputs to match the size of the arrays.

- To express the output `arclen` as an arc length in either degrees or radians, omit the `ellipsoid` argument. This is possible only on a sphere. If `ellipsoid` is supplied, `arclen` is a distance expressed in the same units as the semimajor axis of the ellipsoid. Specify `ellipsoid` as [R 0] to compute `arclen` as a distance on a sphere of radius R, with `arclen` having the same units as R.

Algorithms

Distance calculations for geodesics degrade slowly with increasing distance and may break down for points that are nearly antipodal, as well as when both points are very close to the Equator. In addition, for calculations on an ellipsoid, there is a small but finite input space, consisting of pairs of locations in which both the points are nearly antipodal *and* both points fall close to (but not precisely on) the Equator. In this case, a warning is issued and both `arclen` and `az` are set to `NaN` for the "problem pairs."

Alternatives

Distance between two points can be calculated in two ways. For great circles (on the sphere) and geodesics (on the ellipsoid), the distance is the shortest surface distance between two points. For rhumb lines, the distance is measured along the rhumb line passing through the two points, which is not, in general, the shortest surface distance between them.

When you need to compute both distance and azimuth for the same point pair(s), it is more efficient to do so with a single call to `distance`. That is, use

```
[arclen az] = distance(...);
```

rather than the slower

```
arclen = distance(...)
az = azimuth(...)
```

See Also

Functions
azimuth | elevation | reckon | track | track1 | track2 | trackg

Objects
referenceEllipsoid | referenceSphere | oblateSpheroid

Topics
"Great Circles"

"Rhumb Lines"

Introduced before R2006a

distortcalc

Distortion parameters for map projections

Syntax

```
areascale = distortcalc(lat,long)
areascale = distortcalc(mstruct,lat,long)
[areascale,angdef,maxscale,minscale,merscale,parscale] = distortcalc(...)
```

Description

`areascale = distortcalc(lat,long)` computes the area distortion for the current map projection at the specified geographic location. An area scale of 1 indicates no scale distortion. Latitude and longitude can be scalars, vectors, or matrices in the angle units of the defined map projection.

`areascale = distortcalc(mstruct,lat,long)` uses the projection defined in the map structure `mstruct`.

`[areascale,angdef,maxscale,minscale,merscale,parscale] = distortcalc(...)` computes the area scale, maximum angular deformation of right angles (in the angle units of the defined projection), the particular maximum and minimum scale distortions in any direction, and the particular scale along the meridian and parallel. You can also call `distortcalc` with fewer output arguments, in the order shown.

Background

Map projections inevitably introduce distortions in the shapes and sizes of objects as they are transformed from three-dimensional spherical coordinates to two-dimensional Cartesian coordinates. The amount and type of distortion vary between projections, over the projection, and with the selection of projection parameters such as standard parallels. This function allows a quantitative evaluation of distortion parameters.

Examples

At the equator, the Mercator projection is free of both area and angular distortion:

```
axesm mercator
[areascale,angdef] = distortcalc(0,0)

areascale =
      1.0000
angdef =
      8.5377e-007
```

At 60 degrees north, objects are shown at 400% of their true area. The projection is conformal, so angular distortion is still zero.

```
[areascale,angdef] = distortcalc(60,0)
```

```
areascale =
        4.0000
angdef =
      4.9720e-004
```

Tips

This function uses a finite difference technique. The geographic coordinates are perturbed slightly in different directions and projected. A small amount of error is introduced by numerical computation of derivatives and the variation of map distortion parameters.

See Also

mdistort | tissot

Introduced before R2006a

distdim

Convert length units

Syntax

```
distOut = distdim(distIn,from,to)
distOut = distdim(distIn,from,to,radius)
distOut = distdim(distIn,from,to,sphere)
```

Compatibility

Note distdim has been replaced by unitsratio, but will be maintained for backward compatibility. See "Replacing distdim" on page 1-299 for details.

Description

distOut = distdim(distIn,*from*,*to*) converts distIn from the units specified by *from* to the units specified by *to*. *from* and *to* are case-insensitive, and may equal any of the following:

'meters' or 'm'	
'feet' or 'ft'	U.S. survey feet
'kilometers' or 'km'	
'nauticalmiles' or 'nm'	
'miles', 'statutemiles', 'mi', or 'sm'	Statute miles
'degrees' or 'deg'	
'radians' or 'rad'	

If either *from* or *to* indicates angular units ('degrees' or 'radians'), the conversion to or from linear distance is made along a great circle arc on a sphere with a radius of 6371 km, the mean radius of the Earth.

distOut = distdim(distIn,*from*,*to*,radius) specifies the radius when one of the units, either *from* or *to*, indicates angular units. distdim uses a great circle arc on a sphere of the given radius. The specified length units must apply to radius as well as to the input distance (when *from* indicates length) or output distance (when *to* indicates length). If neither *from* nor *to* indicates angular units, or if both do, then the value of radius is ignored.

distOut = distdim(distIn,*from*,*to*,sphere), where either *from* or *to* indicates angular units, uses a great circle arc on a sphere approximating a body in the Solar System. *sphere* may be one of the following: 'sun', 'moon', 'mercury', 'venus', 'earth', 'mars', 'jupiter', 'saturn', 'uranus', 'neptune', or 'pluto', and is case-insensitive. If neither *to* nor *from* is angular, *sphere* is ignored.

Examples

Convert 100 kilometers to nautical miles:

```
distkm = 100

distkm =
    100

distnm = distdim(distkm,'kilometers','nauticalmiles')

distnm =
    53.9957
```

A degree of arc length is about 60 nautical miles:

```
distnm = distdim(1,'deg','nm')

distnm =
    60.0405
```

This is not accidental. It is the original definition of the nautical mile. Naturally, this assumption does not hold on other planets:

```
distnm = distdim(1,'deg','nm','mars')

distnm =
    31.9474
```

Tips

Arc Lengths of Angles Not Constant

Distance is expressed in one of two general forms: as a linear measure in some unit (kilometers, miles, etc.) or as angular arc length (degrees or radians). While the use of linear units is generally understood, angular arc length is not always as clear. The conversion from angular units to linear units for the arc along any circle is the angle in radians multiplied by the radius of the circle. On the sphere, this means that radians of latitude are directly translatable to kilometers, say, by multiplying by the radius of the Earth in kilometers (about 6,371 km). However, the linear distance associated with radians of longitude changes with latitude; the radius in question is then not the radius of the Earth, but the (chord) radius of the small circle defining that parallel. The angle in radians or degrees associated with any distance is the arc length of a great circle passing through the points of interest. Therefore, the radius in question always refers to the radius of the relevant sphere, consistent with the `distance` function.

Exercise Caution with 'feet' and 'miles'

Exercise caution with 'feet' and 'miles'. `distdim` interprets `'feet'` and `'ft'` as U.S. survey feet, and does not support international feet at all. In contrast, `unitsratio` follows the opposite, and more standard approach, interpreting both `'feet'` and `'ft'` as international feet. `unitsratio` provides separate options, including `'surveyfeet'` and `'sf'`, to indicate survey feet. By definition, one international foot is exactly 0.3048 meters and one U.S. survey foot is exactly 1200/3937 meters. For many applications, the difference is significant. Most projected coordinate systems use either the meter or the survey foot as a standard unit. International feet are less likely to be used, but do occur sometimes. Likewise, `distdim` interprets `'miles'` and `'mi'` as statute miles (also known as U.S.

survey miles), and does not support international miles at all. By definition, one international mile is 5,280 international feet and one statute mile is 5,280 survey feet. You can evaluate:

```
unitsratio('millimeter','statute mile') - ...
    unitsratio('millimeter','mile')
```

to see that the difference between a statute mile and an international mile is just over three millimeters. This may seem like a very small amount over the length of a single mile, but mixing up these units could result in a significant error over a sufficiently long baseline. Originally, the behavior of `distdim` with respect to `'miles'` and `'mi'` was documented only indirectly, via the now-obsolete `unitstr` function. As with feet, `unitsratio` takes a more standard approach. `unitsratio` interprets `'miles'` and `'mi'` as international miles, and `'statute miles'` and `'sm'` as statute miles. (`unitsratio` accepts several other character vectors for each of these units; see the `unitsratio` help for further information.)

Replacing `distdim`

If both *from* and *to* are known at the time of coding, then you may be able to replace `distdim` with a direct conversion utility, as in the following examples:

`distdim(dist,'nm',km')`	⇒ `nm2km(dist)`
`distdim(dist,'sm','deg')`	⇒ `sm2deg(dist)`
`distdim(dist, 'rad', 'km', 'moon')`	⇒ `rad2km(dist,'moon')`

If there is no appropriate direct conversion utility, or you won't know the values of *from* and/or *to* until run time, you can generally replace

```
distdim(dist, FROM, TO)
```

with

```
unitsratio(TO, FROM) * dist
```

If you are using units of feet or miles, see the cautionary note above about how they are interpreted. For example, with `distIn` in meters and `distOut` in survey feet, `distOut = distdim(distIn, 'meters', 'feet')` should be replaced with `distOut = unitsratio('survey feet','meters') * distIn`. Saving a multiplicative factor computed with `unitsratio` and using it to convert in a separate step can make code cleaner and more efficient than using `distdim`. For example, replace

```
dist1_meters = distdim(dist1_nm, 'nm', 'meters');
dist2_meters = distdim(dist2_nm, 'nm', 'meters');
```

with

```
metersPerNM = unitsratio('meters','nm');
dist1_meters = metersPerNM * dist1_nm;
dist2_meters = metersPerNM * dist2_nm;
```

`unitsratio` does not perform great-circle conversion between units of length and angle, but it can be easily combined with other functions to do so. For example, to convert degrees to meters along a great-circle arc on a sphere approximating the planet Mars, you could replace

```
 distdim(dist, 'degrees', 'meters', 'mars')
```

with

```
unitsratio('meters','km') * deg2km(dist, 'mars')
```

See Also

deg2km | deg2nm | deg2sm | km2deg | km2nm | km2rad | km2sm | nm2deg | nm2km | nm2rad | nm2sm | rad2km | rad2nm | rad2sm | sm2deg | sm2km | sm2nm | sm2rad | unitsratio

Introduced before R2006a

dm2degrees

Convert degrees-minutes to degrees

Syntax

```
angleInDegrees = dm2degrees(DM)
```

Description

`angleInDegrees = dm2degrees(DM)` converts angles from degrees-minutes representation to values in degrees which may include a fractional part (sometimes called "decimal degrees").

Examples

Convert Angle in Degree-Minutes to Degrees

```
dm = [ ...
     30   44.78012; ...
    -82   39.90825; ...
      0  -17.12345; ...
      0   14.82000];
format long g
angleInDegrees = dm2degrees(dm)

angleInDegrees = 4×1

   30.7463
  -82.6651
   -0.2854
    0.2470
```

Input Arguments

DM — Angle in degrees-minutes representation
n-by-2 real-valued matrix

Angle in degrees-minutes representation, specified as an *n*-by-2 real-valued matrix. Each row specifies one angle, with the format `[D M]`:

- `D` contains the "degrees" element and must be integer-valued.
- `M` contains the "minutes" element and may have a fractional part. The absolute value of `M` must be less than 60.

For an angle that is positive (north latitude or east longitude) or equal to zero, all elements in the row must be nonnegative. For a negative angle (south latitude or west longitude), the first nonzero element in the row must be negative and the remaining value, if any, is nonnegative.

Output Arguments

angleInDegrees — Angle in degrees
n-element column vector

Angle in degrees, returned as an n-element column vector. The kth element corresponds to the kth row of DM.

Algorithms

For an input row with value [D M], with integer-valued D and real M, the output value will be

```
SGN * (abs(D) + abs(M)/60)
```

where SGN is 1 if D and M are both nonnegative and -1 if the first nonzero element of [D M] is negative. An error results if a nonzero D is followed by a negative M.

See Also

degrees2dm | deg2rad | dms2degrees | str2angle

Topics
"Angle Representations and Angular Units"
"Angles as Binary and Formatted Numbers"

Introduced in R2007a

dms2degrees

Convert degrees-minutes-seconds to degrees

Syntax

```
angleInDegrees = dms2degrees(DMS)
```

Description

`angleInDegrees = dms2degrees(DMS)` converts angles from degrees-minutes-seconds representation to values in degrees which may include a fractional part (sometimes called "decimal degrees").

Examples

Convert Angle in Degree-Minute-Seconds to Degrees

```
dms = [ ...
    30   50   44.78012; ...
   -82    2   39.90825; ...
     0  -30   17.12345; ...
     0    0   14.82000];
format long g
angleInDegrees = dms2degrees(dms)

angleInDegrees = 4×1

   30.8458
  -82.0444
   -0.5048
    0.0041
```

Input Arguments

DMS — Angle in degrees-minutes-seconds representation
n-by-3 real-valued matrix

Angle in degrees-minutes-seconds representation, specified as an *n*-by-3 real-valued matrix. Each row specifies one angle, with the format [D M S]:

- `D` contains the "degrees" element and must be integer-valued.
- `M` contains the "minutes" element and must be integer-valued. The absolute value of `M` must be less than 60.
- `S` contains the "seconds" element and may have a fractional part. The absolute value of `S` must be less than 60.

For an angle that is positive (north latitude or east longitude) or equal to zero, all elements in the row must be nonnegative. For a negative angle (south latitude or west longitude), the first nonzero element in the row must be negative and the remaining values are nonnegative.

Output Arguments

angleInDegrees — Angle in degrees
n-element column vector

Angle in degrees, returned as an n-element column vector. The kth element corresponds to the kth row of DMS.

Algorithms

For an input row with value [D M S], the output value will be

```
SGN * (abs(D) + abs(M)/60 + abs(S)/3600)
```

where SGN is 1 if D, M, and S are all nonnegative and -1 if the first nonzero element of [D M S] is negative. An error results if a nonzero element is followed by a negative element.

See Also
degrees2dm | deg2rad | dm2degrees | str2angle

Topics
"Angle Representations and Angular Units"
"Angles as Binary and Formatted Numbers"

Introduced in R2007a

double

Package: `map.geotiff`

Convert TIFF tag property values to row vector of doubles

Syntax

```
tiffTagValue = double(rpctag)
```

Description

`tiffTagValue = double(rpctag)` returns a 92-element row vector of class `double`, representing the values of the TIFF tag. This is the format required to write the property values to a TIFF file.

Examples

Convert RPCCoefficientTag Properties to 92-Element Vector

Create an RPCCoefficientTag object and view the object summary display.

```
rpctag = map.geotiff.RPCCoefficientTag

rpctag =
  RPCCoefficientTag with properties:

                 BiasErrorInMeters: -1
               RandomErrorInMeters: -1
                        LineOffset: 0
                      SampleOffset: 0
            GeodeticLatitudeOffset: 0
           GeodeticLongitudeOffset: 0
              GeodeticHeightOffset: 0
                         LineScale: 1
                       SampleScale: 1
             GeodeticLatitudeScale: 1
            GeodeticLongitudeScale: 1
               GeodeticHeightScale: 1
          LineNumeratorCoefficients: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
        LineDenominatorCoefficients: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
        SampleNumeratorCoefficients: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
      SampleDenominatorCoefficients: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

Call the RPCCoefficientTag class method `double` to convert the object into a vector of `doubles`. View the result.

```
tifftagvalues = double(rpctag)

tifftagvalues = 1×92

   -1   -1    0    0    0    0    0    1    1    1    1    1    0    0    0    0
```

Input Arguments

rpctag — Rational Polynomial Coefficients (RPC) TIFF tag property values
RPCCoefficientTag object

Rational Polynomial Coefficients (RPC) TIFF tag property values, specified as an
RPCCoefficientTag object.

Example: tifftagvalues = double(rpctag);

Output Arguments

tiffTagValue — Rational Polynomial Coefficients (RPC) TIFF tag property values
92-element vector

Rational Polynomial Coefficients (RPC) TIFF tag property values, returned as 92-element vector of
class double.

See Also
RPCCoefficientTag | geotiffinfo | geotiffwrite

Introduced in R2015b

dreckon

Dead reckoning positions for track

Syntax

```
[drlat,drlong,drtime] = dreckon(waypoints,time,speed)
[drlat,drlong,drtime] = dreckon (waypoints,time,speed,spdtimes)
```

Description

`[drlat,drlong,drtime] = dreckon(waypoints,time,speed)` returns the positions and times of required dead reckoning (DR) points for the input track that starts at the input time. The track should be in navigational track format (two columns, latitude then longitude, in order of traversal). These waypoints are the starting and ending points of each leg of the track. There is one fewer track leg than waypoints, as the last point included is the end of the track. In navigation, the first waypoint would be a navigational fix, taken at `time`. The `speed` input can be a scalar, in which case a constant speed is used throughout, or it can be a vector in which one speed is given for each track leg (that is, speed changes coincide with course changes).

`[drlat,drlong,drtime] = dreckon (waypoints,time,speed,spdtimes)` allows speed changes to occur independent of course changes. The elements of the `speed` vector must have a one-to-one correspondence with the elements of the `spdtimes` vector. This latter variable consists of the time interval after `time` at which each speed order *ends*. For example, if `time` is 6.75, and the first element of `spdtimes` is 1.35, then the first `speed` element is in effect from 6.75 to 8.1 hours. When this syntax is used, the last output DR is the *earlier* of the final `spdtimes` time or the final `waypoints` point.

Background

This is a navigational function. It assumes that all latitudes and longitudes are in degrees, all distances are in nautical miles, all times are in hours, and all speeds are in knots, that is, nautical miles per hour.

Dead reckoning is an estimation of position at various times based on courses, speeds, and times elapsed from the last certain position, or fix. In navigational practice, a dead reckoning position, or DR, must be plotted at every course change, every speed change, and at every hour, on the hour. Navigators also DR at other times that are not relevant to this function.

Often in practice, when two events occur that require DRs within a very short time, only one DR is generated. This function mimics that practice by setting a tolerance of 3 minutes (0.05 hours). No two DRs will fall closer than that.

Refer to "Navigation" in the *Mapping Toolbox Guide* for further information.

Examples

Assume that a navigator gets a fix at noon, 1200Z, which is (10.3ºN, 34.67ºW). He's in a hurry to make a 1330Z rendezvous with another ship at (9.9ºN, 34.5ºW), so he plans on a speed of 25 knots. After the rendezvous, both ships head for (0º, 37ºW). The engineer wants to take an engine off line

for maintenance at 1430Z, so at that time, speed must be reduced to 15 knots. At 1530Z, the maintenance will be done. Determine the DR points up to the end of the maintenance.

```
waypoints = [10.1 -34.6; 9.9 -34.5; 0 -37]

waypoints =
   10.1000  -34.6000                      % Fix at noon
    9.9000  -34.5000                      % Rendezvous point
         0  -37.0000                      % Ultimate destination

speed = [25; 15];
spdtimes = [2.5; 3.5];                    % Elapsed times after fix
noon = 12;
[drlat,drlong,drtime] = dreckon(waypoints,noon,speed,spdtimes);
[drlat,drlong,drtime]

ans =
    9.8999  -34.4999   12.5354                     % Course change at waypoint
    9.7121  -34.5478   13.0000                     % On the hour
    9.3080  -34.6508   14.0000                     % On the hour
    9.1060  -34.7022   14.5000                     % Speed change to 15 kts
    8.9847  -34.7330   15.0000                     % On the hour
    8.8635  -34.7639   15.5000                     % Stop at final spdtime, last
                            % waypoint has not been reached
```

See Also
legs | navfix | track

Introduced before R2006a

driftcorr

Heading to correct for wind or current drift

Syntax

```
heading = driftcorr(course,airspeed,windfrom,windspeed)
[heading,groundspeed,windcorrangle] = driftcorr(...)
```

Description

`heading = driftcorr(course,airspeed,windfrom,windspeed)` computes the heading that corrects for drift due to wind (for aircraft) or current (for watercraft). `course` is the desired direction of movement (in degrees), `airspeed` is the speed of the vehicle relative to the moving air or water mass, `windfrom` is the direction facing into the wind or current (in degrees), and `windspeed` is the speed of the wind or current (in the same units as `airspeed`).

`[heading,groundspeed,windcorrangle] = driftcorr(...)` also returns the ground speed and wind correction angle. The wind correction angle is positive to the right, and negative to the left.

Examples

An aircraft cruising at a speed of 160 knots plans to fly to an airport due north of its current position. If the wind is blowing from 310 degrees at 45 knots, what heading should the aircraft fly to remain on course?

```
course=0; airspeed=160;windfrom=310; windspeed = 45;
[heading,groundspeed,windcorrangle] =
driftcorr(course,airspeed,windfrom,windspeed)

heading =

      347.56

groundspeed =

      127.32

windcorrangle =

     -12.442
```

The required heading is 348 degrees, which amounts to a wind correction angle of 12 degrees to the left of course. The headwind component reduces the aircraft's ground speed to 127 knots.

See Also
driftvel

Introduced before R2006a

driftvel

Wind or current from heading, course, and speeds

Syntax

```
[windfrom,windspeed] = driftvel (course,groundspeed,heading,airspeed)
```

Description

`[windfrom,windspeed] = driftvel (course,groundspeed,heading,airspeed)` computes the wind (for aircraft) or current (for watercraft) from course, heading, and speeds. `course` and `groundspeed` are the direction and speed of movement relative to the ground (in degrees), `heading` is the direction in which the vehicle is steered, and `airspeed` is the speed of the vehicle relative to the air mass or water. The output `windfrom` is the direction facing into the wind or current (in degrees), and `windspeed` is the speed of the wind or current (in the same units as `airspeed` and `groundspeed`).

Examples

An aircraft is cruising at a true air speed of 160 knots and a heading of 10 degrees. From the Global Positioning System (GPS) receiver, the pilot determines that the aircraft is progressing over the ground at 155 knots in a northerly direction. What is the wind aloft?

```
course = 0; groundspeed = 155; heading = 10; airspeed = 160;
[windfrom,windspeed] =
driftvel(course,groundspeed,heading,airspeed)

windfrom =
      84.717

windspeed =
      27.902
```

The wind is blowing from the right, 085 degrees at 28 knots.

See Also
`driftcorr`

Introduced before R2006a

dted

(To be removed) Read U.S. Department of Defense Digital Terrain Elevation Data (DTED)

Note dted will be removed in a future release. Use readgeoraster instead. For more information, see "Compatibility Considerations".

Syntax

```
[Z, refvec] = dted
[Z, refvec] = dted(filename)
[Z, refvec] = dted(filename, samplefactor)
[Z, refvec] = dted(filename, samplefactor, latlim, lonlim)
[Z, refvec] = dted(foldername, samplefactor, latlim, lonlim)
[Z, refvec, UHL, DSI, ACC] = dted(...)
```

Description

[Z, refvec] = dted returns all of the elevation data in a DTED file as a regular data grid, Z, with elevations in meters. The file is selected interactively. This function reads the DTED elevation files, which generally have file names ending in .dtN, where N is 0,1,2,3,... refvec is the associated three-element referencing vector that geolocates Z.

[Z, refvec] = dted(filename) returns all of the elevation data in the specified DTED file. The file must be found on the MATLAB path. If not found, the file may be selected interactively.

[Z, refvec] = dted(filename, samplefactor) subsamples data from the specified DTED file. samplefactor is a scalar integer. When samplefactor is 1 (the default), DTED reads the data at its full resolution. When samplefactor is an integer n greater than one, every nth point is read.

[Z, refvec] = dted(filename, samplefactor, latlim, lonlim) reads the data for the part of the DTED file within the latitude and longitude limits. The limits must be two-element vectors in units of degrees.

[Z, refvec] = dted(foldername, samplefactor, latlim, lonlim) reads and concatenates data from multiple files within a DTED CD-ROM or folder structure. The foldername input is a string scalar or character vector with the name of a folder containing the DTED folder. Within the DTED folder are subfolders for each degree of longitude, each of which contain files for each degree of latitude. For DTED CD-ROMs, foldername is the device name of the CD-ROM drive.

[Z, refvec, UHL, DSI, ACC] = dted(...) returns structures containing the DTED User Header Label (UHL), Data Set Identification (DSI) and Accuracy metadata records.

Background

The U. S. Department of Defense, through the National Geospatial Intelligence Agency, produces several kinds of digital cartographic data. One is digital elevation data, in a series called DTED, for Defense Digital Terrain Elevation Data. The data is available as 1-by-1 degree quadrangles at horizontal resolutions ranging from about 1 kilometer to 1 meter. The lowest resolution data is

available to the public. Certain higher resolution data is restricted to the U.S. Department of Defense and its contractors.

DTED Level 0 files have 121-by-121 points. DTED Level 1 files have 1201-by-1201. The edges of adjacent tiles have redundant records. Maps extend a half a cell outside the requested map limits. The 1 kilometer data and some higher-resolution data is available online, as are product specifications and documentation. DTED files are binary. No line ending conversion or byte-swapping is required when downloading a DTED file.

Examples

Read and Display DTED Data

Read and display DTED data for an area around South Boulder Peak in Colorado.

First, import the elevation data. Replace unused output arguments with a tilde character (~). Associate the elevation data with geographic locations by creating a geographic postings reference object. To create a reference object, specify the latitude limits, longitude limits, and size of the elevation data grid.

```
[Z,~] = dted('n39_w106_3arc_v2.dt1');

latlim = [39 40];
lonlim = [-106 -105];
R = georefpostings(latlim,lonlim,size(Z));
```

Create a set of map axes, then plot the data using `geoshow`. Display a colormap appropriate for elevation data using `demcmap`.

```
usamap(latlim,lonlim);
geoshow(Z,R,'DisplayType','surface')
demcmap(Z)
```
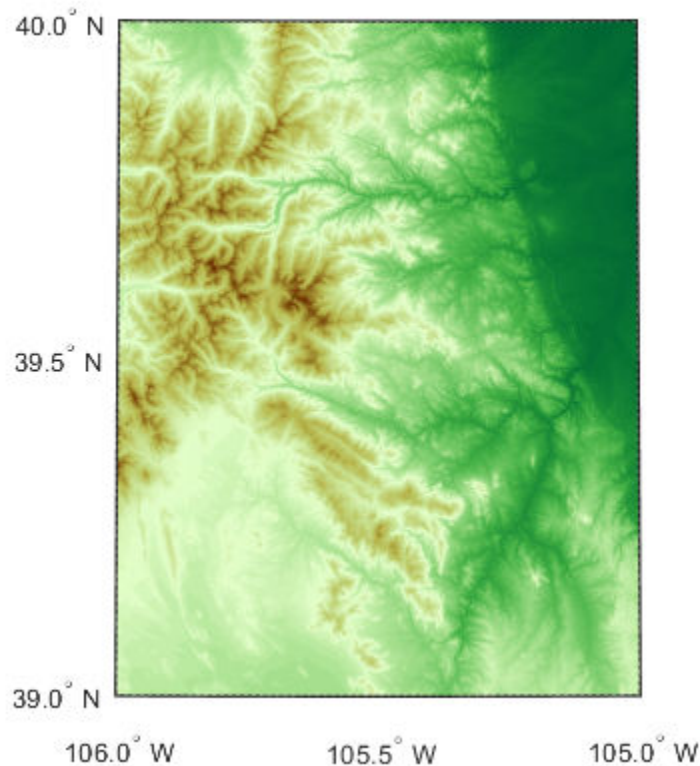
The DTED file used in this example is courtesy of the US Geological Survey.

Tips

Latitude-Dependent Sampling

In DTED files north of 50° North and south of 50° South, where the meridians have converged significantly relative to the equator, the longitude sampling interval is reduced to half of the latitude sampling interval. In order to retain square output cells, this function reduces the latitude sampling to match the longitude sampling. For example, it will return a 121-by-121 elevation grid for a DTED file covering from 49 to 50 degrees north, but a 61-by-61 grid for a file covering from 50 to 51 degrees north. When you supply a folder name instead of a file name, and `latlim` spans either 50° North or 50° South, an error results.

Snapping Latitude and Longitude Limits

If you call `dted` specifying arbitrary latitude-longitude limits for a region of interest, the grid and referencing vector returned will not exactly honor the limits you specified unless they fall precisely on grid cell boundaries. Because grid cells are discrete and cannot be arbitrarily divided, the data grid returned will include all areas between your latitude-longitude limits and the next row or column of cells, potentially in all four directions.

Data Sources and Information

DTED files contain digital elevation maps covering 1-by-1-degree quadrangles at horizontal resolutions ranging from about 1 kilometer to 1 meter. For details on locating DTED for download over the Internet, see "Find Geospatial Data Online".

Null Data Values

Some DTED Level 1 and higher data tiles contain null data cells, coded with value -32767. When encountered, these null data values are converted to NaN.

Nonconforming Data Encoding

DTED files from some sources may depart from the specification by using two's complement encoding for binary elevation files instead of "sign-bit" encoding. This difference affects the decoding of negative values, and incorrect decoding usually leads to nonsensical elevations.

Thus, if the DTED function determines that all the (nonnull) negative values in a file would otherwise be less than -12,000 meters, it issues a warning and assumes two's complement encoding.

Compatibility Considerations

dted will be removed
Not recommended starting in R2020a

Raster reading functions that return referencing vectors will be removed, including dted. Instead, use readgeoraster, which returns a geographic postings reference object. Reference objects have several advantages over referencing vectors.

- Unlike referencing vectors, reference objects have properties that document the size of the associated raster, its geographic limits, and the direction of its rows and columns. For examples of reference object properties, see GeographicPostingsReference.

- You can manipulate the limits of rasters associated with geographic reference objects using the geocrop function.

- You can manipulate the size and resolution of rasters associated with geographic reference objects using the georesize function.

- Most functions that accept referencing vectors as input also accept reference objects.

Get metadata about DTED files by accessing properties of the reference object or by using the georasterinfo function.

This table shows some typical usages of dted and how to update your code to use readgeoraster and georasterinfo. The readgeoraster function requires you to specify a file extension. For example, use [Z,R] = readgeoraster('n39_w106_3arc_v2.dt1').

Will Be Removed	Recommended
[Z,refvec] = dted(filename);	[Z,R] = readgeoraster(filename);
[Z,refvec] = dted(filename,samplefactor);	[Z,R] = readgeoraster(filename); [Z,R] = georesize(Z,R,1/samplefactor);

Will Be Removed	Recommended
[Z,refvec] = dted(filename,samplefactor,latlim,lonlim);	[Z,R] = readgeoraster(filename); [Z,R] = geocrop(Z,R,latlim,lonlim); [Z,R] = georesize(Z,R,1/samplefactor);

[Z,refvec,UHL,DSI,ACC] = dted(filename);	[Z,R] = readgeoraster(filename);
	info = georasterinfo(filename);
	meta = info.Metadata;

The `readgeoraster` function returns data using the native data type embedded in the file. Return a different data type by specifying the `'OutputType'` name-value pair. For example, use `[Z,R] = readgeoraster(filename,'OutputType','double')`.

The `readgeoraster` function does not automatically replace missing data with `NaN` values. If your data set uses large negative numbers to indicate missing data, replace them with `NaN` values using the `standardizeMissing` function.

```
[Z,R] = readgeoraster('MtWashington-ft.grd');
info = georasterinfo('MtWashington-ft.grd');
m = info.MissingDataIndicator;
Z = standardizeMissing(Z,m);
```

Reading and concatenating multiple DTED files from a folder using the `readgeoraster` function is not supported. Find the standard names of the DTED files required to cover a region by using the `dteds` function. Then read the individual files by using the `readgeoraster` function. For information about concatenating raster tiles, see "Mosaic Spatially Referenced Raster Tiles".

See Also
readgeoraster | georasterinfo | dteds

Introduced before R2006a

dteds

DTED file names for latitude-longitude quadrangle

Syntax

```
fname = dteds(latlim,lonlim)
fname = dteds(latlim,lonlim,level)
```

Description

`fname = dteds(latlim,lonlim)` returns Level 0 DTED file names (folder and name) required to cover the geographic region specified by `latlim` and `lonlim`. This function constructs the file names for a given geographic region based on the file-naming convention established by the Defense Digital Terrain Elevation Data (DTED) database.

`fname = dteds(latlim,lonlim,level)` controls the level for which the file names are generated. Valid inputs for the `level` of the DTED files include 0, 1, or 2.

Background

The U. S. Department of Defense produces several kinds of digital cartographic data. One is digital elevation data, in a series called DTED, for Defense Digital Terrain Elevation Data. The data is available as 1-by-1 degree quadrangles at horizontal resolutions ranging from about 1 kilometer to 1 meter. The lowest resolution data is available to the public. Higher resolution data is restricted to the U.S. Department of Defense and its contractors.

Determining the files needed to cover a particular region requires knowledge of the DTED database naming conventions. This function constructs the file names for a given geographic region based on these conventions.

Examples

Which files are needed for Cape Cod?

```
latlim = [ 41.15 42.22]; lonlim = [-70.94 -69.68];
dteds(latlim,lonlim,1)

ans =
    '\DTED\W071\N41.dt1'
    '\DTED\W070\N41.dt1'
    '\DTED\W071\N42.dt1'
    '\DTED\W070\N42.dt1'
```

See Also
readgeoraster

Introduced before R2006a

earthRadius

Mean radius of planet Earth

Syntax

```
R = earthRadius
R = earthRadius(lengthUnit)
```

Description

`R = earthRadius` returns the scalar value 6371000, the mean radius of the Earth in meters.

`R = earthRadius(lengthUnit)` returns the mean radius of the Earth using the specified unit of length. The `lengthUnit` input may be any value accepted by the `validateLengthUnit` function.

Examples

Retrieve the mean radius of the Earth, using several units.

```
earthRadius              % Returns 6371000

earthRadius('meters')    % Returns 6371000

earthRadius('km')        % Returns 6371
```

See Also
`unitsratio` | `validateLengthUnit`

Introduced in R2010b

eastof

Wrap longitudes to values east of specified meridian

Compatibility

Note The `eastof` function is obsolete and will be removed in a future release of Mapping Toolbox software. Replace it with the following calls, which are also more efficient:

`eastof(lon,meridian,'degrees') ==> meridian+mod(lon-meridian,360)`

`eastof(lon,meridian,'radians') ==> meridian+mod(lon-meridian,2*pi)`

Syntax

```
lonWrapped = eastof(lon,meridian)
lonWrapped = eastof(lon,meridian,angleunits)
```

Description

`lonWrapped = eastof(lon,meridian)` wraps angles in `lon` to values in the interval `[meridian meridian+360)`. `lon` is a scalar longitude or vector of longitude values. All inputs and outputs are in degrees.

`lonWrapped = eastof(lon,meridian,angleunits)` where the character vector `angleunits` specifies the input and output units. `angleunits` can be either `'degrees'` or `'radians'`. It may be abbreviated and is case-insensitive. If `angleunits` is `'radians'`, the input is wrapped to the interval `[meridian meridian+2*pi)`.

Introduced before R2006a

ecc2flat

Flattening of ellipse from eccentricity

Note Support for nonscalar input, including the syntax `f = ecc2flat(ellipsoid)`, will be removed in a future release.

Syntax

```
f = ecc2flat(ecc)
f = ecc2flat(ellipsoid)
```

Description

`f = ecc2flat(ecc)` computes the flattening of an ellipse (or ellipsoid of revolution) given its eccentricity `ecc`. Except when the input has 2 columns (or is a row vector), each element is assumed to be an eccentricity and the output `f` has the same size as `ecc`.

`f = ecc2flat(ellipsoid)`, where `ellipsoid` has two columns (or is a row vector), assumes that the eccentricity is in the second column, and a column vector is returned.

See Also
ecc2n | flat2ecc | majaxis | minaxis

Introduced before R2006a

ecc2n

Third flattening of ellipse from eccentricity

Note Support for nonscalar input, including the syntax `n = ecc2n(ellipsoid)`, will be removed in a future release.

Syntax

```
n = ecc2n(ecc)
n = ecc2n(ellipsoid)
```

Description

`n = ecc2n(ecc)` computes the parameter *n* (the "third flattening") of an ellipse (or ellipsoid of revolution) given its eccentricity `ecc`. *n* is defined as (*a-b*)/(*a+b*), where *a* is the semimajor axis and *b* is the semiminor axis. Except when the input has 2 columns (or is a row vector), each element is assumed to be an eccentricity and the output *n* has the same size as `ecc`.

`n = ecc2n(ellipsoid)`, where `ellipsoid` has two columns (or is a row vector), assumes that the eccentricity is in the second column, and a column vector is returned.

See Also

ecc2flat | majaxis | minaxis | n2ecc

Introduced before R2006a

ecef2aer

Transform geocentric Earth-centered Earth-fixed coordinates to local spherical

Syntax

```
[az,elev,slantRange] = ecef2aer(X,Y,Z,lat0,lon0,h0,spheroid)
[ ___ ] = ecef2aer( ___ ,angleUnit)
```

Description

`[az,elev,slantRange] = ecef2aer(X,Y,Z,lat0,lon0,h0,spheroid)` transforms the geocentric Earth-centered Earth-fixed (ECEF) Cartesian coordinates specified by X, Y, and Z to the local azimuth-elevation-range (AER) spherical coordinates specified by `az`, `elev`, and `slantRange`. Specify the origin of the local AER system with the geodetic coordinates `lat0`, `lon0`, and `h0`. Each coordinate input argument must match the others in size or be scalar. Specify `spheroid` as the reference spheroid for the geodetic coordinates.

`[ ___ ] = ecef2aer( ___ ,angleUnit)` specifies the units for latitude, longitude, azimuth, and elevation. Specify `angleUnit` as `'degrees'` (the default) or `'radians'`.

Examples

Calculate AER Coordinates from ECEF Coordinates

Find the AER coordinates of a satellite with respect to a satellite dish, using the ECEF coordinates of the satellite and the geodetic coordinates of the satellite dish.

First, specify the reference spheroid as WGS84 with length units measured in kilometers. For more information about WGS84, see "Reference Spheroids". The units for the ellipsoidal height, slant range, and ECEF coordinates must match the units specified by the `LengthUnit` property of the reference spheroid.

```
wgs84 = wgs84Ellipsoid('kilometers');
```

Specify the geodetic coordinates of the local origin. In this example, the local origin is the satellite dish. Specify `h0` as ellipsoidal height in kilometers.

```
lat0 = 42.3221;
lon0 = -71.3576;
h0 = 0.0847;
```

Specify the ECEF coordinates of the point of interest. In this example, the point of interest is the satellite.

```
x = 10766.0803;
y = 14143.6070;
z = 33992.3880;
```

Then, calculate the AER coordinates of the satellite with respect to the satellite dish. In this example, `slantRange` displays in scientific notation.

```
[az,elev,slantRange] = ecef2aer(x,y,z,lat0,lon0,h0,wgs84)

az = 24.8012

elev = 14.6185

slantRange = 3.6272e+04
```

Reverse the transformation using the `aer2ecef` function. In this example, the results display in scientific notation.

```
[x,y,z] = aer2ecef(az,elev,slantRange,lat0,lon0,h0,wgs84)

x = 1.0766e+04

y = 1.4144e+04

z = 3.3992e+04
```

Input Arguments

X — ECEF *x*-coordinates
scalar | vector | matrix | N-D array

ECEF *x*-coordinates of one or more points in the geocentric ECEF system, specified as a scalar, vector, matrix, or N-D array. Specify values in units that match the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Data Types: `single` | `double`

Y — ECEF *y*-coordinates
scalar | vector | matrix | N-D array

ECEF *y*-coordinates of one or more points in the geocentric ECEF system, specified as a scalar, vector, matrix, or N-D array. Specify values in units that match the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Data Types: `single` | `double`

Z — ECEF *z*-coordinates
scalar | vector | matrix | N-D array

ECEF *z*-coordinates of one or more points in the geocentric ECEF system, specified as a scalar, vector, matrix, or N-D array. Specify values in units that match the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Data Types: `single` | `double`

lat0 — Geodetic latitude of local origin
scalar | vector | matrix | N-D array

Geodetic latitude of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: single | double

lon0 — Geodetic longitude of local origin
scalar | vector | matrix | N-D array

Geodetic longitude of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify the values in degrees. To use values in radians, specify the angleUnit argument as 'radians'.

Data Types: single | double

h0 — Ellipsoidal height of local origin
scalar | vector | matrix | N-D array

Ellipsoidal height of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify values in units that match the LengthUnit property of the spheroid object. For example, the default length unit for the reference ellipsoid created by wgs84Ellipsoid is 'meter'.

For more information about ellipsoidal height, see "Find Ellipsoidal Height from Orthometric Height".

Data Types: single | double

spheroid — Reference spheroid
referenceEllipsoid object | oblateSpheroid object | referenceSphere object

Reference spheroid, specified as a referenceEllipsoid object, oblateSpheroid object, or referenceSphere object. The term reference spheroid is used synonymously with reference ellipsoid. To create a reference spheroid, use the creation function for the object. To specify the reference ellipsoid for WGS84, use the wgs84Ellipsoid function.

For more information about reference spheroids, see "Reference Spheroids".

Example: spheroid = referenceEllipsoid('GRS 80');

angleUnit — Angle units
'degrees' (default) | 'radians'

Angle units, specified as 'degrees' (the default) or 'radians'.

Output Arguments

az — Azimuth angles
scalar | vector | matrix | N-D array

Azimuth angles of one or more points in the local AER system, returned as a scalar, vector, matrix, or N-D array. Azimuths are measured clockwise from north. Values are specified in degrees within the half-open interval [0 360). To use values in radians, specify the angleUnit argument as 'radians'.

elev — Elevation angles
scalar | vector | matrix | N-D array

Elevation angles of one or more points in the local AER system, returned as a scalar, vector, matrix, or N-D array. Elevations are measured with respect to a plane that is perpendicular to the normal of the spheroid surface. If the local origin is on the surface of the spheroid (h0 = 0), then the plane is tangent to the spheroid.

Values are specified in degrees within the closed interval [-90 90]. To use values in radians, specify the `angleUnit` argument as `'radians'`.

slantRange — Distances from local origin
scalar | vector | matrix | N-D array

Distances from the local origin, returned as a scalar, vector, matrix, or N-D array. Each distance is calculated along a straight, 3-D, Cartesian line. Units are specified by the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also
`ecef2enu` | `ecef2ned` | `aer2ecef` | `geodetic2aer`

Topics
"Choose a 3-D Coordinate System"

Introduced in R2012b

ecef2enu

Transform geocentric Earth-centered Earth-fixed coordinates to local east-north-up

Syntax

```
[xEast,yNorth,zUp] = ecef2enu(X,Y,Z,lat0,lon0,h0,spheroid)
[ ___ ] = ecef2enu( ___ ,angleUnit)
```

Description

[xEast,yNorth,zUp] = ecef2enu(X,Y,Z,lat0,lon0,h0,spheroid) transforms the geocentric Earth-centered Earth-fixed (ECEF) Cartesian coordinates specified by X, Y, and Z to the local east-north-up (ENU) Cartesian coordinates specified by xEast, yNorth, and zUp. Specify the origin of the local ENU system with the geodetic coordinates lat0, lon0, and h0. Each coordinate input argument must match the others in size or be scalar. Specify spheroid as the reference spheroid for the geodetic coordinates.

[___] = ecef2enu(___ ,angleUnit) specifies the units for latitude and longitude. Specify angleUnit as 'degrees' (the default) or 'radians'.

Examples

Calculate ENU Coordinates from ECEF Coordinates

Find the ENU coordinates of orbital debris with respect to a satellite, using the ECEF coordinates of the debris and the geodetic coordinates of the satellite.

First, specify the reference spheroid as WGS84 with length units measured in kilometers. For more information about WGS84, see "Reference Spheroids". The units for the ellipsoidal height, ECEF coordinates, and ENU coordinates must match the units specified by the LengthUnit property of the reference spheroid.

```
wgs84 = wgs84Ellipsoid('kilometer');
```

Specify the geodetic coordinates of the local origin. In this example, the local origin is the satellite. Specify h0 as ellipsoidal height in kilometers.

```
lat0 = 45.9132;
lon0 = 36.7484;
h0 = 1877.7532;
```

Specify the ECEF coordinates of the point of interest. In this example, the point of interest is the orbital debris.

```
x = 5507.5289;
y = 4556.2241;
z = 6012.8208;
```

Then, calculate the ENU coordinates of the debris with respect to the satellite. In this example, zUp displays in scientific notation.

```
[xEast,yNorth,zUp] = ecef2enu(x,y,z,lat0,lon0,h0,wgs84)

xEast = 355.6013

yNorth = -923.0832

zUp = 1.0410e+03
```

Reverse the transformation using the `enu2ecef` function. In this example, the results display in scientific notation.

```
[x,y,z] = enu2ecef(xEast,yNorth,zUp,lat0,lon0,h0,wgs84)

x = 5.5075e+03

y = 4.5562e+03

z = 6.0128e+03
```

Input Arguments

X — ECEF *x*-coordinates
scalar | vector | matrix | N-D array

ECEF *x*-coordinates of one or more points in the geocentric ECEF system, specified as a scalar, vector, matrix, or N-D array. Specify values in units that match the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Data Types: `single` | `double`

Y — ECEF *y*-coordinates
scalar | vector | matrix | N-D array

ECEF *y*-coordinates of one or more points in the geocentric ECEF system, specified as a scalar, vector, matrix, or N-D array. Specify values in units that match the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Data Types: `single` | `double`

Z — ECEF *z*-coordinates
scalar | vector | matrix | N-D array

ECEF *z*-coordinates of one or more points in the geocentric ECEF system, specified as a scalar, vector, matrix, or N-D array. Specify values in units that match the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Data Types: `single` | `double`

lat0 — Geodetic latitude of local origin
scalar | vector | matrix | N-D array

Geodetic latitude of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

lon0 — Geodetic longitude of local origin
scalar | vector | matrix | N-D array

Geodetic longitude of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

h0 — Ellipsoidal height of local origin
scalar | vector | matrix | N-D array

Ellipsoidal height of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify values in units that match the `LengthUnit` property of the `spheroid` object. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

For more information about ellipsoidal height, see "Find Ellipsoidal Height from Orthometric Height".

Data Types: `single` | `double`

spheroid — Reference spheroid
`referenceEllipsoid` object | `oblateSpheroid` object | `referenceSphere` object

Reference spheroid, specified as a `referenceEllipsoid` object, `oblateSpheroid` object, or `referenceSphere` object. The term reference spheroid is used synonymously with reference ellipsoid. To create a reference spheroid, use the creation function for the object. To specify the reference ellipsoid for WGS84, use the `wgs84Ellipsoid` function.

For more information about reference spheroids, see "Reference Spheroids".

Example: `spheroid = referenceEllipsoid('GRS 80');`

angleUnit — Angle units
`'degrees'` (default) | `'radians'`

Angle units, specified as `'degrees'` (the default) or `'radians'`.

Output Arguments

xEast — ENU *x*-coordinates
scalar | vector | matrix | N-D array

ENU *x*-coordinates of one or more points in the local ENU system, returned as a scalar, vector, matrix, or N-D array. Units are specified by the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

yNorth — ENU *y*-coordinates
scalar | vector | matrix | N-D array

ENU *y*-coordinates of one or more points in the local ENU system, returned as a scalar, vector, matrix, or N-D array. Units are specified by the `LengthUnit` property of the `spheroid` argument.

For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

zUp — ENU *z*-coordinates
scalar | vector | matrix | N-D array

ENU *z*-coordinates of one or more points in the local ENU system, returned as a scalar, vector, matrix, or N-D array. Units are specified by the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Tips

To transform vectors instead of coordinate locations, use the `ecef2enuv` function.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`ecef2ned` | `ecef2aer` | `enu2ecef` | `geodetic2enu`

Topics
"Choose a 3-D Coordinate System"

Introduced in R2012b

ecef2enuv

Rotate geocentric Earth-centered Earth-fixed vector to local east-north-up

Syntax

```
[uEast,vNorth,wUp] = ecef2enuv(U,V,W,lat0,lon0)
[ ___ ] = ecef2enuv( ___ ,angleUnit)
```

Description

[uEast,vNorth,wUp] = ecef2enuv(U,V,W,lat0,lon0) returns vector components uEast, vNorth, and wUp in a local east-north-up (ENU) system corresponding to vector components U, V, and W in a geocentric Earth-centered Earth-fixed (ECEF) system. Specify the origin of the system with the geodetic coordinates lat0 and lon0. Each coordinate input argument must match the others in size or be scalar.

[___] = ecef2enuv(___ ,angleUnit) specifies the units for latitude and longitude. Specify angleUnit as 'degrees' (the default) or 'radians'.

Examples

Calculate ENU Vector Components from ECEF Components

Find the ENU velocity components of a ground vehicle using its ECEF velocity components.

Specify the geodetic coordinates of the vehicle in degrees and the ECEF velocity components in kilometers per hour.

```
lat0 = 17.4114;
lon0 = 78.2700;

U = 27.9799;
V = -1.0990;
W = -15.7723;
```

Calculate the ENU components of the vehicle. The units for the ENU components match the units for the ECEF components. Thus, the ENU components are returned in kilometers per hour. The rotation performed by ecef2enuv does not affect the speed of the vehicle.

```
[uEast,vNorth,wUp] = ecef2enuv(U,V,W,lat0,lon0)

uEast = -27.6190

vNorth = -16.4298

wUp = -0.3186
```

Reverse the rotation using the enu2ecefv function.

```
[U,V,W] = enu2ecefv(uEast,vNorth,wUp,lat0,lon0)
```

```
U = 27.9799

V = -1.0990

W = -15.7723
```

Input Arguments

U — ECEF *x*-components
scalar value | vector | matrix | N-D array

ECEF *x*-components of one or more vectors, specified as a scalar value, vector, matrix, or N-D array.

Data Types: `single` | `double`

V — ECEF *y*-components
scalar value | vector | matrix | N-D array

ECEF *y*-components of one or more vectors, specified as a scalar value, vector, matrix, or N-D array.

Data Types: `single` | `double`

W — ECEF *z*-components
scalar value | vector | matrix | N-D array

ECEF *z*-components of one or more vectors, specified as a scalar value, vector, matrix, or N-D array.

Data Types: `single` | `double`

`lat0` — Geodetic latitude of local origin
scalar | vector | matrix | N-D array

Geodetic latitude of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

`lon0` — Geodetic longitude of local origin
scalar | vector | matrix | N-D array

Geodetic longitude of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

`angleUnit` — Angle units
`'degrees'` (default) | `'radians'`

Angle units, specified as `'degrees'` (the default) or `'radians'`.

Output Arguments

`uEast` — ENU *x*-components
scalar value | vector | matrix | N-D array

ENU *x*-components of one or more vectors, returned as a scalar value, vector, matrix, or N-D array. Values are returned in the units specified by U, V, and W.

vNorth — ENU *y*-components
scalar value | vector | matrix | N-D array

ENU *y*-components of one or more vectors, returned as a scalar value, vector, matrix, or N-D array. Values are returned in the units specified by U, V, and W.

wUp — ENU *z*-components
scalar value | vector | matrix | N-D array

ENU *z*-components of one or more vectors, returned as a scalar value, vector, matrix, or N-D array. Values are returned in the units specified by U, V, and W.

Tips

To transform coordinate locations instead of vectors, use the `ecef2enu` function.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
enu2ecefv | ecef2nedv | ned2ecefv

Topics
"Vectors in 3-D Coordinate Systems"

Introduced in R2012b

ecef2geodetic

Transform geocentric Earth-centered Earth-fixed coordinates to geodetic

Syntax

```
[lat,lon,h] = ecef2geodetic(spheroid,X,Y,Z)
[lat,lon,h] = ecef2geodetic( ___ ,angleUnit)

[lat,lon,h] = ecef2geodetic(X,Y,Z,spheroid)
```

Description

[lat,lon,h] = ecef2geodetic(spheroid,X,Y,Z) transforms the geocentric Earth-Centered Earth-Fixed (ECEF) Cartesian coordinates specified by X, Y, and Z to the geodetic coordinates specified by lat, lon, and h. Specify spheroid as the reference spheroid for the geodetic coordinates.

[lat,lon,h] = ecef2geodetic(___ ,angleUnit) specifies the units for latitude and longitude. Specify angleUnit as 'degrees' (the default) or 'radians'.

[lat,lon,h] = ecef2geodetic(X,Y,Z,spheroid) is supported but not recommended. Unlike the previous syntaxes, this syntax returns lat and lon in radians. Specify spheroid as either a reference spheroid or an ellipsoid vector of the form [semimajor_axis, eccentricity]. Specify X, Y, and Z in the same units as the length unit of the spheroid argument. Additionally, the output h returns in the same units as the length unit of the spheroid argument.

Examples

Calculate Geodetic Coordinates from ECEF Coordinates

Find the geodetic coordinates of Paris, France, using its ECEF coordinates.

First, specify the reference spheroid as WGS84 with length units measured in kilometers. For more information about WGS84, see "Reference Spheroids". The units for the ECEF coordinates and ellipsoidal height must match the units specified by the LengthUnit property of the reference spheroid.

```
wgs84 = wgs84Ellipsoid('kilometer');
```

Specify the ECEF coordinates of Paris in kilometers.

```
x = 4201;
y = 172.46;
z = 4780.1;
```

Then, calculate the geodetic coordinates of Paris. The result h is ellipsoidal height in kilometers.

```
[lat,lon,h] = ecef2geodetic(wgs84,x,y,z)
```

```
lat = 48.8562
```

```
lon = 2.3508
```

```
h = 0.0674
```

Reverse the transformation using the `geodetic2ecef` function. In this example, x and z display in scientific notation.

```
[x,y,z] = geodetic2ecef(wgs84,lat,lon,h)
```

```
x = 4.2010e+03
```

```
y = 172.4600
```

```
z = 4.7801e+03
```

Input Arguments

spheroid — Reference spheroid
referenceEllipsoid object | oblateSpheroid object | referenceSphere object

Reference spheroid, specified as a `referenceEllipsoid` object, `oblateSpheroid` object, or `referenceSphere` object. The term reference spheroid is used synonymously with reference ellipsoid. To create a reference spheroid, use the creation function for the object. To specify the reference ellipsoid for WGS84, use the `wgs84Ellipsoid` function.

For more information about reference spheroids, see "Reference Spheroids".

Example: `spheroid = referenceEllipsoid('GRS 80');`

X — ECEF *x*-coordinates
scalar | vector | matrix | N-D array

ECEF *x*-coordinates of one or more points in the geocentric ECEF system, specified as a scalar, vector, matrix, or N-D array. Specify values in units that match the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Data Types: single | double

Y — ECEF *y*-coordinates
scalar | vector | matrix | N-D array

ECEF *y*-coordinates of one or more points in the geocentric ECEF system, specified as a scalar, vector, matrix, or N-D array. Specify values in units that match the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Data Types: single | double

Z — ECEF *z*-coordinates
scalar | vector | matrix | N-D array

ECEF *z*-coordinates of one or more points in the geocentric ECEF system, returned as a scalar, vector, matrix, or N-D array. Units are specified by the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

angleUnit — Angle units
'degrees' (default) | 'radians'

Angle units, specified as 'degrees' (the default) or 'radians'.

Output Arguments

lat — Geodetic latitude
scalar | vector | matrix | N-D array

Geodetic latitude of one or more points, returned as a scalar, vector, matrix, or N-D array. Values are specified in degrees within the interval [-90 90]. To use values in radians, specify the angleUnit argument as 'radians'.

lon — Geodetic longitude
scalar | vector | matrix | N-D array

Geodetic longitude of one or more points, returned as a scalar, vector, matrix, or N-D array. Values are specified in degrees within the interval [-180 180]. To use values in radians, specify the angleUnit argument as 'radians'.

h — Ellipsoidal height
scalar | vector | matrix | N-D array

Ellipsoidal height of one or more points, returned as a scalar, vector, matrix, or N-D array. Values are specified in units that match the LengthUnit property of the spheroid object. For example, the default length unit for the reference ellipsoid created by wgs84Ellipsoid is 'meter'.

For more information about ellipsoidal height, see "Find Ellipsoidal Height from Orthometric Height".

Tips

*   The geocentric Cartesian (ECEF) coordinate system is fixed with respect to the Earth, with its origin at the center of the spheroid and its positive X-, Y-, and Z axes intersecting the surface at the following points:

	Latitude	Longitude	Notes
X-axis	0	0	Equator at the Prime Meridian
Y-axis	0	90	Equator at 90-degrees East
Z-axis	90	0	North Pole

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also
geodetic2ecef | ecefOffset | ecef2aer | ecef2ned | ecef2enu

Topics
"Choose a 3-D Coordinate System"

Introduced in R2012b

ecef2lv

Convert geocentric (ECEF) to local vertical coordinates

Note `ecef2lv` will be removed in a future release. Use `ecef2enu` instead. In `ecef2enu`, the latitude and longitude of the local origin are in degrees by default, so the optional `angleUnit` input should be included, with the value `'radians'`.

Syntax

`[xl,yl,zl] = ecef2lv(x,y,z,phi0,lambda0,h0,ellipsoid)`

Description

`[xl,yl,zl] = ecef2lv(x,y,z,phi0,lambda0,h0,ellipsoid)` converts geocentric point locations specified by the coordinate arrays `x`, `y`, and `z` to the local vertical coordinate system, with its origin at geodetic latitude `phi0`, longitude `lambda0`, and ellipsoidal height `h0`. The arrays `x`, `y`, and `z` may be of any shape, as long as they all match in size. `phi0`, `lambda0`, and `h0` must be scalars. `ellipsoid` is a `referenceEllipsoid` (`oblateSpheroid`) object, a `referenceSphere` object, or a vector of the form `[semimajor axis, eccentricity]`. `x`, `y`, `z`, and `h0` must have the same length units as the semimajor axis. `phi0` and `lambda0` must be in radians. The output coordinate arrays, `xl`, `yl`, and `zl` are the local vertical coordinates of the input points. They have the same size as `x`, `y`, and `z` and have the same length units as the semimajor axis.

In the local vertical Cartesian system defined by `phi0`, `lambda0`, `h0`, and `ellipsoid`, the `xl` axis is parallel to the plane tangent to the ellipsoid at (`phi0`,`lambda0`) and points due east. The `yl` axis is parallel to the same plane and points due north. The `zl` axis is normal to the ellipsoid at (`phi0`,`lambda0`) and points outward into space. The local vertical system is sometimes referred to as east-north-up or ENU.

More About

Local Vertical System

In the local vertical Cartesian system defined by `phi0`, `lambda0`, `h0`, and `ellipsoid`, the `xl` axis is parallel to the plane tangent to the ellipsoid at (`phi0`,`lambda0`) and points due east. The `yl` axis is parallel to the same plane and points due north. The `zl` axis is normal to the ellipsoid at (`phi0`,`lambda0`) and points outward into space. The local vertical system is sometimes referred to as East-North-Up or ENU.

Geocentric System

The geocentric Cartesian coordinate system, also known as Earth-Centered, Earth-Fixed (ECEF), is fixed with respect to the Earth, with its origin at the center of the spheroid and its positive X-, Y-, and Z axes intersecting the surface at the following points:

	Latitude	Longitude	Notes
X-axis	0	0	Equator at the Prime Meridian

	Latitude	Longitude	Notes
Y-axis	0	90	Equator at 90-degrees East
Z-axis	90	0	North Pole

See Also

ecef2enu

Introduced before R2006a

ecef2ned

Transform geocentric Earth-centered Earth-fixed coordinates to local north-east-down

Syntax

```
[xNorth,yEast,zDown] = ecef2ned(X,Y,Z,lat0,lon0,h0,spheroid)
[ ___ ] = ecef2ned( ___ ,angleUnit)
```

Description

`[xNorth,yEast,zDown] = ecef2ned(X,Y,Z,lat0,lon0,h0,spheroid)` transforms the geocentric Earth-centered Earth-fixed (ECEF) Cartesian coordinates specified by X, Y, and Z to the local north-east-down (NED) Cartesian coordinates specified by `xNorth`, `yEast`, and `zDown`. Specify the origin of the local NED system with the geodetic coordinates `lat0`, `lon0`, and `h0`. Each coordinate input argument must match the others in size or be scalar. Specify `spheroid` as the reference spheroid for the geodetic coordinates.

`[ ___ ] = ecef2ned( ___ ,angleUnit)` specifies the units for latitude and longitude. Specify `angleUnit` as `'degrees'` (the default) or `'radians'`.

Examples

Calculate NED Coordinates from ECEF Coordinates

Find the NED coordinates of Mount Mansfield with respect to a nearby aircraft, using the ECEF coordinates of Mount Mansfield and the geodetic coordinates of the aircraft.

First, specify the reference spheroid as WGS84 with length units measured in kilometers. For more information about WGS84, see "Reference Spheroids". The units for the ellipsoidal height, ECEF coordinates, and NED coordinates must match the units specified by the `LengthUnit` property of the reference spheroid.

```
wgs84 = wgs84Ellipsoid('kilometer');
```

Specify the geodetic coordinates of the local origin. In this example, the local origin is the aircraft. Specify `h0` as ellipsoidal height in kilometers.

```
lat0 = 44.532;
lon0 = -72.782;
h0 = 1.699;
```

Specify the ECEF coordinates of the point of interest. In this example, the point of interest is Mount Mansfield.

```
x = 1345.660;
y = -4350.891;
z = 4452.314;
```

Then, calculate the NED coordinates of Mount Mansfield with respect to the aircraft. Since the ellipsoidal height of the aircraft is greater than the height of Mount Mansfield, a passenger needs to

look down to see the mountaintop. The *z*-axis of an NED coordinate system points down. Thus, the value of zDown is positive.

```
[xNorth,yEast,zDown] = ecef2ned(x,y,z,lat0,lon0,h0,wgs84)

xNorth = 1.3343

yEast = -2.5444

zDown = 0.3600
```

Reverse the transformation using the ned2ecef function. In this example, the results display in scientific notation.

```
[x,y,z] = ned2ecef(xNorth,yEast,zDown,lat0,lon0,h0,wgs84)

x = 1.3457e+03

y = -4.3509e+03

z = 4.4523e+03
```

Input Arguments

X — ECEF *x*-coordinates
scalar | vector | matrix | N-D array

ECEF *x*-coordinates of one or more points in the geocentric ECEF system, specified as a scalar, vector, matrix, or N-D array. Specify values in units that match the LengthUnit property of the spheroid argument. For example, the default length unit for the reference ellipsoid created by wgs84Ellipsoid is 'meter'.

Data Types: single | double

Y — ECEF *y*-coordinates
scalar | vector | matrix | N-D array

ECEF *y*-coordinates of one or more points in the geocentric ECEF system, specified as a scalar, vector, matrix, or N-D array. Specify values in units that match the LengthUnit property of the spheroid argument. For example, the default length unit for the reference ellipsoid created by wgs84Ellipsoid is 'meter'.

Data Types: single | double

Z — ECEF *z*-coordinates
scalar | vector | matrix | N-D array

ECEF *z*-coordinates of one or more points in the geocentric ECEF system, specified as a scalar, vector, matrix, or N-D array. Specify values in units that match the LengthUnit property of the spheroid argument. For example, the default length unit for the reference ellipsoid created by wgs84Ellipsoid is 'meter'.

Data Types: single | double

lat0 — Geodetic latitude of local origin
scalar | vector | matrix | N-D array

Geodetic latitude of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

lon0 — Geodetic longitude of local origin
scalar | vector | matrix | N-D array

Geodetic longitude of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

h0 — Ellipsoidal height of local origin
scalar | vector | matrix | N-D array

Ellipsoidal height of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify values in units that match the `LengthUnit` property of the `spheroid` object. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

For more information about ellipsoidal height, see "Find Ellipsoidal Height from Orthometric Height".

Data Types: `single` | `double`

spheroid — Reference spheroid
`referenceEllipsoid` object | `oblateSpheroid` object | `referenceSphere` object

Reference spheroid, specified as a `referenceEllipsoid` object, `oblateSpheroid` object, or `referenceSphere` object. The term reference spheroid is used synonymously with reference ellipsoid. To create a reference spheroid, use the creation function for the object. To specify the reference ellipsoid for WGS84, use the `wgs84Ellipsoid` function.

For more information about reference spheroids, see "Reference Spheroids".

Example: `spheroid = referenceEllipsoid('GRS 80');`

angleUnit — Angle units
`'degrees'` (default) | `'radians'`

Angle units, specified as `'degrees'` (the default) or `'radians'`.

Output Arguments

xNorth — NED *x*-coordinates
scalar | vector | matrix | N-D array

NED *x*-coordinates of one or more points in the local NED system, returned as a scalar, vector, matrix, or N-D array. Units are specified by the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

yEast — NED *y*-coordinates
scalar | vector | matrix | N-D array

NED *y*-coordinates of one or more points in the local NED system, returned as a scalar, vector, matrix, or N-D array. Units are specified by the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

zDown — NED *z*-coordinates
scalar | vector | matrix | N-D array

NED *z*-coordinates of one or more points in the local NED system, returned as a scalar, vector, matrix, or N-D array. Units are specified by the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Tips

To transform vectors instead of coordinate locations, use the `ecef2nedv` function.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also
`ecef2enu` | `ecef2aer` | `ned2ecef` | `geodetic2ned`

Topics
"Choose a 3-D Coordinate System"

Introduced in R2012b

ecef2nedv

Rotate geocentric Earth-centered Earth-fixed vector to local north-east-down

Syntax

```
[uNorth,vEast,wDown] = ecef2nedv(U,V,W,lat0,lon0)
[ ___ ] = ecef2nedv( ___ ,angleUnit)
```

Description

`[uNorth,vEast,wDown] = ecef2nedv(U,V,W,lat0,lon0)` returns vector components `uNorth`, `vEast`, and `wDown` in a local north-east-down (NED) system corresponding to vector components U, V, and W in a geocentric Earth-centered Earth-fixed (ECEF) system. Specify the origin of the system with the geodetic coordinates `lat0` and `lon0`. Each coordinate input argument must match the others in size or be scalar.

`[ ___ ] = ecef2nedv( ___ ,angleUnit)` specifies the units for latitude and longitude. Specify `angleUnit` as `'degrees'` (the default) or `'radians'`.

Examples

Calculate NED Vector Components from ECEF Components

Find the NED velocity components of an aircraft using its ECEF components.

Specify the geodetic coordinates of the aircraft in degrees and the ECEF velocity components in kilometers per hour.

```
lat0 = 61.64;
lon0 = 30.70;

U = 530.2445;
V = 492.1283;
W = 396.3459;
```

Calculate the NED components of the aircraft. The units for the NED components match the units for the ECEF components. Thus, the NED components are returned in kilometers per hour. The negative value of `wDown` means the aircraft is ascending.

```
[uNorth,vEast,wDown] = ecef2nedv(U,V,W,lat0,lon0)
```

```
uNorth = -434.0403
```

```
vEast = 152.4451
```

```
wDown = -684.6964
```

Reverse the rotation using the `ned2ecefv` function.

```
[U,V,W] = ned2ecefv(uNorth,vEast,wDown,lat0,lon0)
```

```
U = 530.2445

V = 492.1283

W = 396.3459
```

Input Arguments

U — ECEF *x*-components
scalar value | vector | matrix | N-D array

ECEF *x*-components of one or more vectors, specified as a scalar value, vector, matrix, or N-D array.

Data Types: `single` | `double`

V — ECEF *y*-components
scalar value | vector | matrix | N-D array

ECEF *y*-components of one or more vectors, specified as a scalar value, vector, matrix, or N-D array.

Data Types: `single` | `double`

W — ECEF *z*-components
scalar value | vector | matrix | N-D array

ECEF *z*-components of one or more vectors, specified as a scalar value, vector, matrix, or N-D array.

Data Types: `single` | `double`

lat0 — Geodetic latitude of local origin
scalar | vector | matrix | N-D array

Geodetic latitude of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

lon0 — Geodetic longitude of local origin
scalar | vector | matrix | N-D array

Geodetic longitude of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

angleUnit — Angle units
`'degrees'` (default) | `'radians'`

Angle units, specified as `'degrees'` (the default) or `'radians'`.

Output Arguments

uNorth — NED *x*-components
scalar value | vector | matrix | N-D array

NED *x*-components of one or more vectors in the local NED system, returned as a scalar value, vector, matrix, or N-D array. Values are returned in the units specified by U, V, and W.

vEast — NED *y*-components
scalar value | vector | matrix | N-D array

NED *y*-components of one or more vectors in the local NED system, returned as a scalar value, vector, matrix, or N-D array. Values are returned in the units specified by U, V, and W.

wDown — NED *z*-components
scalar value | vector | matrix | N-D array

NED *z*-components of one or more vectors in the local NED system, returned as a scalar value, vector, matrix, or N-D array. Values are returned in the units specified by U, V, and W.

Tips

To transform coordinate locations instead of vectors, use the `ecef2ned` function.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also
ecef2enuv | ned2ecefv | enu2ecefv

Topics
"Vectors in 3-D Coordinate Systems"

Introduced in R2012b

ecefOffset

Cartesian ECEF offset between geodetic coordinates

Syntax

```
[deltaX,deltaY,deltaZ] = ecefOffset(spheroid,lat1,lon1,h1,lat2,lon2,h2)
[] = ecefOffset( ___ ,angleUnit)
```

Description

`[deltaX,deltaY,deltaZ] = ecefOffset(spheroid,lat1,lon1,h1,lat2,lon2,h2)` returns the Earth-Centered Earth-Fixed (ECEF) Cartesian offset between the geodetic coordinates specified by `lat1`, `lon1`, and `h1` and the coordinates specified by `lat2`, `lon2`, and `h2`. Specify `spheroid` as the reference spheroid for the geodetic coordinates.

`[] = ecefOffset( ___ ,angleUnit)`, where `angleUnit` is `'radians'`, specifies the units for latitude and longitude as radians. If you do not specify an angle unit, then latitude and longitude are in degrees.

Examples

Calculate Cartesian ECEF Offset Between Geodetic Positions

Find the ECEF offset between Paris, France and Miami, Florida.

First, specify the reference spheroid as WGS 84. For more information about WGS84, see "Reference Spheroids". The units for the ellipsoidal height and offset vector must match the units specified by the `LengthUnit` property of the reference spheroid. The default length unit for the reference spheroid created by `wgs84Ellipsoid` is `'meter'`.

```
wgs84 = wgs84Ellipsoid;
```

Specify the geodetic coordinates of Paris and Miami. Specify `hParis` and `hMiami` as ellipsoidal height in meters. The value of `hMiami` is negative because Miami is below the surface of the reference spheroid.

```
latParis = 48.8567;
lonParis = 2.3508;
hParis = 80;

latMiami = 25.7753;
lonMiami = -80.2089;
hMiami = -25;
```

Calculate the ECEF offset between the two geodetic positions. The values `dx`, `dy`, and `dz` are specified in meters. For this example, the results display in scientific notation.

```
[dx,dy,dz] = ecefOffset(wgs84,latParis,lonParis,hParis,latMiami,lonMiami,hMiami)
```

```
dx = -3.2236e+06
```

```
dy = -5.8359e+06

dz = -2.0235e+06
```

Calculate the straight-line, 3-D Cartesian distance from Paris to Miami.

```
d = norm([dx dy dz])
```

```
d = 6.9674e+06
```

Input Arguments

spheroid — Reference spheroid
referenceEllipsoid object | oblateSpheroid object | referenceSphere object

Reference spheroid, specified as a referenceEllipsoid object, oblateSpheroid object, or referenceSphere object. The term reference spheroid is used synonymously with reference ellipsoid. To create a reference spheroid, use the creation function for the object. To specify the reference ellipsoid for WGS84, use the wgs84Ellipsoid function.

For more information about reference spheroids, see "Reference Spheroids".

Example: spheroid = referenceEllipsoid('GRS 80');

lat — Geodetic latitude
scalar | vector

Geodetic latitude of one or more points, specified as a scalar or vector. Specify the values in degrees. To use values in radians, specify the angleUnit argument as 'radians'.

Data Types: single | double

lon — Geodetic longitude
scalar | vector

Geodetic longitude of one or more points, specified as a scalar or vector. Specify the values in degrees. To use values in radians, specify the angleUnit argument as 'radians'.

Data Types: single | double

h — Ellipsoidal height
scalar | vector

Ellipsoidal height of one or more points, specified as a scalar or vector. Specify values in units that match the LengthUnit property of the spheroid object. For example, the default length unit for the reference ellipsoid created by wgs84Ellipsoid is 'meter'.

For more information about ellipsoidal height, see "Find Ellipsoidal Height from Orthometric Height".

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64

angleUnit — Units of the latitude and longitude coordinates
'degrees' | 'radians'

Units of the latitude and longitude coordinates, specified as 'degrees' or 'radians'.

Data Types: char

Output Arguments

deltaX — ECEF offset in *x*-axis direction
scalar | vector

ECEF offset in the *x*-axis direction, returned as a scalar or vector. Units are specified by the LengthUnit property of the spheroid object. For example, the default length unit for the reference ellipsoid created by wgs84Ellipsoid is 'meter'.

deltaY — ECEF offset in *y*-axis direction
scalar | vector

ECEF offset in the *y*-axis direction, returned as a scalar or vector. Units are specified by the LengthUnit property of the spheroid object. For example, the default length unit for the reference ellipsoid created by wgs84Ellipsoid is 'meter'.

deltaZ — ECEF offset in *z*-axis direction
scalar | vector

ECEF offset in the *z*-axis direction, returned as a scalar or vector. Units are specified by the LengthUnit property of the spheroid object. For example, the default length unit for the reference ellipsoid created by wgs84Ellipsoid is 'meter'.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

geodetic2ecef | ecef2geodetic | oblateSpheroid | referenceEllipsoid | referenceSphere

Introduced in R2012b

egm96geoid

Geoid height from Earth Gravitational Model 1996 (EGM96)

Note Syntaxes of the `egm96geoid` function that return referencing vectors will be removed in a future release. Use a syntax that returns a reference object instead *(requires R2020a or later)*. For more information, see "Compatibility Considerations".

Syntax

```
N = egm96geoid(lat,lon)
N = egm96geoid(R)
[N,globalR] = egm96geoid

[N,refvec] = egm96geoid(samplefactor)
[N,refvec] = egm96geoid(samplefactor,latlim,lonlim)
```

Description

`N = egm96geoid(lat,lon)` returns the height in meters of the geoid at the specified latitude and longitude from the "Earth Gravitational Model of 1996 (EGM96)" on page 1-353. Specify latitude and longitude in degrees. *(since R2019b)*

`N = egm96geoid(R)` returns geoid heights at the cell center or posting locations specified by the geographic postings reference or geographic cells reference object R. *(since R2020a)*

`[N,globalR] = egm96geoid` returns geoid heights for the entire globe as a 721-by-1441 matrix spaced at 15-minute intervals with latitude limits `[-90 90]` and longitude limits `[0 360]`. The function also returns a geographic postings reference object that contains spatial referencing information for the geoid heights. *(since R2020a)*

`[N,refvec] = egm96geoid(samplefactor)` returns a grid `N` of geoid heights from EGM96, sampled horizontally and vertically at `samplefactor` intervals. The output argument `refvec` is a referencing vector used to associate each geoid height with a latitude and longitude.

`[N,refvec] = egm96geoid(samplefactor,latlim,lonlim)` returns geoid heights within specified latitude and longitude limits.

Examples

Geoid Height from Latitude and Longitude

Find geoid heights from EGM96 by specifying latitude and longitude values in degrees. The result is returned in meters.

```
lat = 27.988056;
lon = 86.925278;
N = egm96geoid(lat,lon)
```

```
N = -28.7444
```

Geoid Heights within Limits

View geoid heights from EGM96 for an area including Europe.

First, create a `GeographicPostingsReference` object. Specify the latitude and longitude limits of the area in degrees. Specify the raster size as the number of rows and columns for the grid of geoid heights. Then, return the grid of geoid heights for the area by calling `egm96geoid` on the reference object.

```
latlim = [35 72];
lonlim = [-12 51];
rasterSize = [100 100];
R = georefpostings(latlim,lonlim,rasterSize);
N = egm96geoid(R);
```

Note that the size of N matches the raster size of R.

```
size(N)
```

```
ans = 1×2

   100   100
```

Load coastline latitude and longitude data into the workspace. Then, plot the geoid height and coastline data on a set of map axes.

```
load coastlines
```

```
worldmap(latlim,lonlim)
geoshow(N,R,'DisplayType','surface')
geoshow(coastlat,coastlon,'Color','k')
```
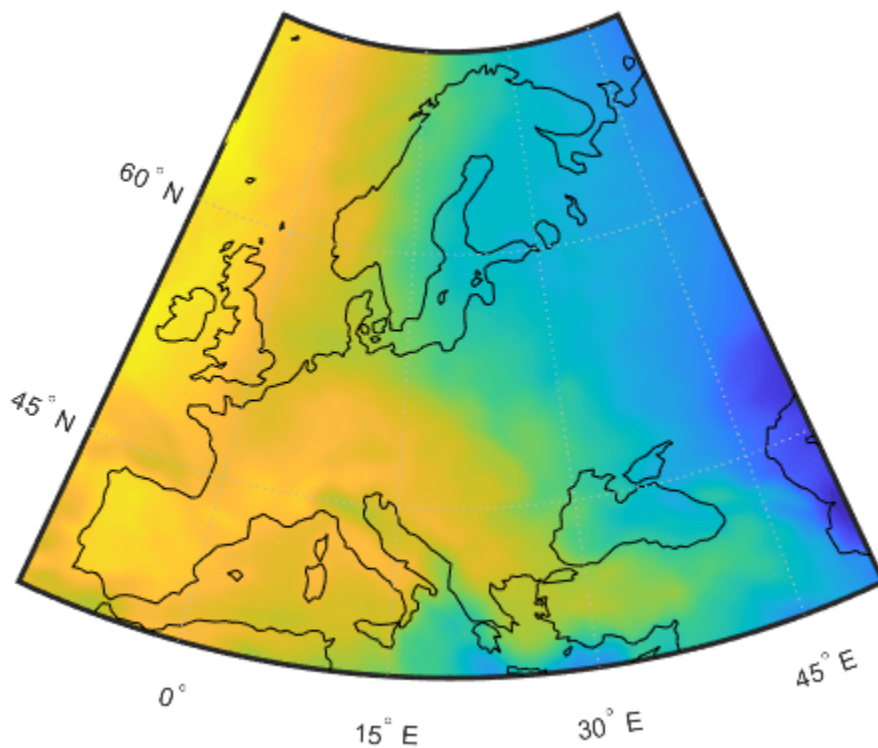
Geoid Heights for Entire Globe

View geoid heights from EGM96 for the entire globe.

First, return the geoid heights and a referencing object for the globe. The geoid heights are spaced at 15-minute intervals. Load coastline latitude and longitude data into the workspace. Then, plot the geoid heights and coastline data on a set of map axes.

```
[N,R] = egm96geoid;
load coastlines

worldmap('World')
geoshow(N,R,'DisplayType','surface')
geoshow(coastlat,coastlon,'Color','k')
```
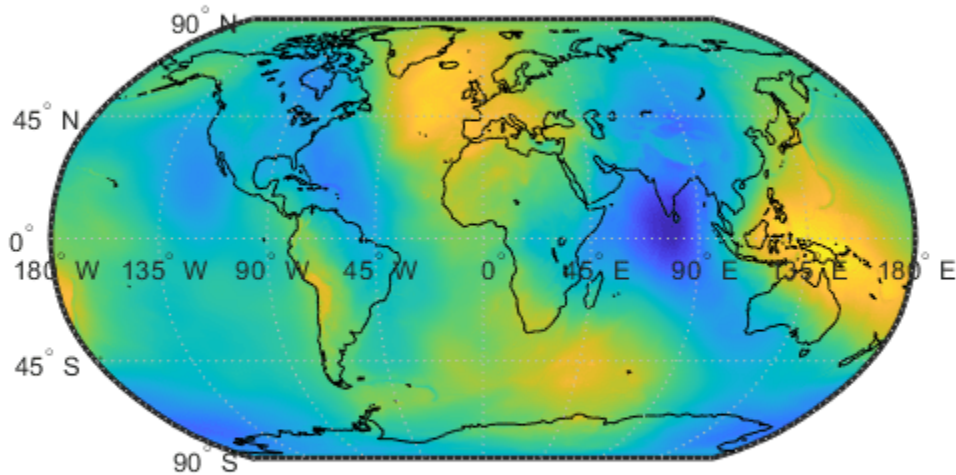
Input Arguments

lat — Latitude
scalar | vector | matrix

Latitude in degrees, specified as a scalar, vector, or matrix. The dimension of lat depends on the dimension of the geoid heights you want to find.

- To find a single geoid height, specify lat and lon as scalars.
- To find several geoid heights, specify lat and lon as vectors of the same length.
- To find a *p*-by-*q* grid of geoid heights, specify lat and lon as *p*-by-*q* matrices.

Data Types: single | double

lon — Longitude
scalar | vector | matrix

Longitude in degrees, specified as a scalar, vector, or matrix. The dimension of lon depends on the dimension of the geoid heights you want to find.

- To find a single geoid height, specify lat and lon as scalars.
- To find several geoid heights, specify lat and lon as vectors of the same length.
- To find a *p*-by-*q* grid of geoid heights, specify lat and lon as *p*-by-*q* matrices.

Data Types: single | double

R — Geographic reference
GeographicPostingsReference object | GeographicCellsReference object

Geographic reference, specified as a GeographicPostingsReference object or a GeographicCellsReference object that contains geospatial referencing information for N. The RasterSize property of the geographic reference object determines the size of the data grid, size(N).

samplefactor — Sample factor
positive integer

Sample factor, specified as a positive integer.

Example: egm96geoid(2) returns a grid of geoid heights spaced at 30 minute intervals.

Data Types: double

latlim — Latitude limits
two-element vector

Latitude limits, specified as a two-element vector of the form [southernLimit northernLimit]. Specify latitude limits in the range [-90 90].

Example: [50 65]

Data Types: double

lonlim — Longitude limits
two-element vector

Longitude limits, specified as a two-element vector of the form [westernLimit easternLimit]. Specify longitude limits in the range [-180 180] or [0 360].

Example: [170 190] returns data centered on the 180-degree meridian.

Example: [-10 10] returns data centered on the Prime Meridian.

Data Types: double

Output Arguments

N — Geoid height
scalar | vector | matrix

Geoid height in meters, returned as a scalar, vector, or matrix.

The size of N depends on the syntax:

Syntax	Size of N
N = egm96geoid(lat,lon)	Size of lat and lon
N = egm96geoid(R)	R.RasterSize
[N,globalR] = egm96geoid	721-by-1441 matrix

Data Types: double

globalR — Geographic reference
GeographicPostingsReference object

Geographic reference, returned as a GeographicPostingsReference object of size 721-by-1441 with latitude limits [-90 90] and longitude limits [0 360].

refvec — Referencing vector
three-element vector

Referencing vector, returned as a three-element vector of the form [s nlat wlon] with these values:

- s – Number of geoid height samples per degree
- nlat – Northernmost latitude in degrees, plus 1/(2*s)
- wlon – Westernmost longitude in degrees, minus 1/(2*s)

MATLAB uses refvec to associate each geoid height with a latitude and longitude. Geoid height samples are located along latitude–longitude grid lines, as opposed to between the grid lines. For more information, see "Geographic Interpretations of Geolocated Grids".

Data Types: double

More About

Earth Gravitational Model of 1996 (EGM96)

The geoid is an equipotential surface of the Earth's gravity field that approximates global mean sea level. You can visualize the geoid as the surface of the ocean without effects such as weather, waves, and land. The Earth Gravitational Model of 1996 (EGM96) is based on the ellipsoid specified by the World Geodetic System of 1984 (WGS84), so the egm96geoid function returns heights above or below the surface of the WGS84 ellipsoid.

Compatibility Considerations

egm96geoid syntaxes that return referencing vectors will be removed
Not recommended starting in R2020b

Syntaxes of the egm96geoid function that return referencing vectors will be removed. Use a syntax that returns a geographic raster reference object instead. Reference objects have several advantages over referencing vectors.

- Unlike referencing vectors, reference objects have properties that document the size of the associated raster, its geographic limits, and the direction of its rows and columns. For more information about reference object properties, see the GeographicCellsReference and GeographicPostingsReference objects.
- You can manipulate the limits of rasters associated with reference objects using the geocrop function.
- You can manipulate the size and resolution of rasters associated with reference objects using the georesize function.
- Most functions that accept referencing vectors as inputs also accept reference objects.

This table shows the syntaxes of the `egm96geoid` function that return referencing vectors and how to update your code to use syntaxes that return reference objects instead.

Will Be Removed	Recommended

`[N,refvec] = egm96geoid(samplefactor);`	`[~,globalR] = egm96geoid;`
	`latlim = globalR.LatitudeLimits;`
	`lonlim = globalR.LongitudeLimits;`
	`globalSize = globalR.RasterSize;`
	`rasterSize = 1 + ceil((globalSize - 1) / samplefactor`
	`R = georefpostings(latlim,lonlim,rasterSize);`
	`N = egm96geoid(R);`

`[N,refvec] = egm96geoid(samplefactor,latlim,lonlim)`	`[~,globalR] = egm96geoid;`
	`spacing = samplefactor ...`
	`    * globalR.SampleSpacingInLatitude;`
	`R = georefpostings(latlim,lonlim,spacing,spacing);`
	`N = egm96geoid(R);`

See Also

Functions
GeographicPostingsReference | GeographicCellsReference | geoshow | ndgrid

Topics
"The Shape of the Earth"
"Find Ellipsoidal Height from Orthometric and Geoid Height"
"Georeferenced Raster Data"

Introduced before R2006a

elevation

Local vertical elevation angle, range, and azimuth

Note `elevation` will be removed in a future release. Use `geodetic2aer` instead.

The reference point comes second in the `geodetic2aer` argument list, and the outputs are ordered differently. The replacement pattern is:

```
[azimuthangle, elevationangle, slantrange] = geodetic2aer(lat2, lon2, alt2,
lat1, lon1, alt1, spheroid, ...)
```

Unlike `elevation`, `geodetic2aer` requires a spheroid input, and it must be an `oblateSpheroid`, `referenceEllipsoid`, or `referenceSphere` object, not a 2-by-1 ellipsoid vector.

You can use the following steps to convert an ellipsoid vector, `ellipsoid`, to an `oblateSpheroid` object, `spheroid`:

- `spheroid = oblateSpheroid;`
- `spheroid.SemimajorAxis = ellipsoid(1);`
- `spheroid.Eccentricity = ellipsoid(2);`

When elevation is called with only 6 inputs, the GRS 80 reference ellipsoid, in meters, is used by default. To replace this usage, use `referenceEllipsoid('GRS80','meters')` as the spheroid input for `geodetic2aer`.

If an `angleunits` input is included, it must follow the `spheroid` input in the call to `geodetic2aer`, rather than preceding it.

You can specify the `lengthunits` parameter when calling `elevation`, but `geodetic2aer` has no such input. Instead, set the `LengthUnit` property of the input spheroid to the desired value. In this case a `referenceEllipsoid` or `referenceSphere` object must be used (not an `oblateSpheroid` object).

Syntax

```
[elevationangle,slantrange,azimuthangle] = ...
  elevation(lat1,lon1,alt1,lat2,lon2,alt2)
[...] = elevation(lat1,lon1,alt1,lat2,lon2,alt2,...
  angleunits)
[...] = elevation(lat1,lon1,alt1,lat2,lon2,alt2,...
  angleunits,distanceunits)
[...] = elevation(lat1,lon1,alt1,lat2,lon2,alt2,...
  angleunits,ellipsoid)
```

Description

`[elevationangle,slantrange,azimuthangle] = ...`
`  elevation(lat1,lon1,alt1,lat2,lon2,alt2)` computes the elevation angle, slant range, and azimuth angle of point 2 (with geodetic coordinates `lat2`, `lon2`, and `alt2`) as viewed from point

1 (with geodetic coordinates `lat1`, `lon1`, and `alt1`). The coordinates `alt1` and `alt2` are ellipsoidal heights. The elevation angle is the angle of the line of sight above the local horizontal at point 1. The slant range is the three-dimensional Cartesian distance between point 1 and point 2. The azimuth is the angle from north to the projection of the line of sight on the local horizontal. Angles are in units of degrees; altitudes and distances are in meters. The figure of the earth is the default ellipsoid (GRS 80).

Inputs can be vectors of points, or arrays of any shape, but must match in size, with the following exception: Elevation, range, and azimuth from a single point to a set of points can be computed very efficiently by providing scalar coordinate inputs for point 1 and vectors or arrays for point 2.

`[...] = elevation(lat1,lon1,alt1,lat2,lon2,alt2,...`
  angleunits) where `angleunits` specifies the units of the input and output angles. If you omit `angleunits`, `'degrees'` is assumed.

`[...] = elevation(lat1,lon1,alt1,lat2,lon2,alt2,...`
  angleunits,distanceunits) where `distanceunits` specifies the altitude and slant-range units. If you omit `distanceunits`, `'meters'` is the default. Any units value recognized by `unitsratio` may be used.

`[...] = elevation(lat1,lon1,alt1,lat2,lon2,alt2,...`
  angleunits,ellipsoid) uses `ellipsoid` to specify the ellipsoid. `ellipsoid` is a `referenceSphere`, `referenceEllipsoid`, or `oblateSpheroid` object, or a vector of the form `[semimajor_axis eccentricity]`. If `ellipsoid` is supplied, the altitudes must be in the same units as the semimajor axis, and the slant range will be returned in these units. If `ellipsoid` is omitted, the default is a unit sphere. Distances are in meters unless otherwise specified.

Note The line-of-sight azimuth angles returned by `elevation` will generally differ slightly from the corresponding outputs of `azimuth` and `distance`, except for great circle azimuths on a spherical earth.

Examples

Find the elevation angle of a point 90 degrees from an observer assuming that the observer and the target are both 1000 km above the Earth.

```
lat1 = 0; lon1 = 0; alt1 = 1000*1000;
lat2 = 0; lon2 = 90; alt2 = 1000*1000;
elevang = elevation(lat1,lon1,alt1,lat2,lon2,alt2)

elevang =
    -45
```

Visually check the result using the `los2` line of sight function. Construct a data grid of zeros to represent the Earth's surface. The `los2` function with no output arguments creates a figure displaying the geometry.

```
Z = zeros(180,360);
refvec = [1 90 -180];
los2(Z,refvec,lat1,lon1,lat2,lon2,alt1,alt1);
```
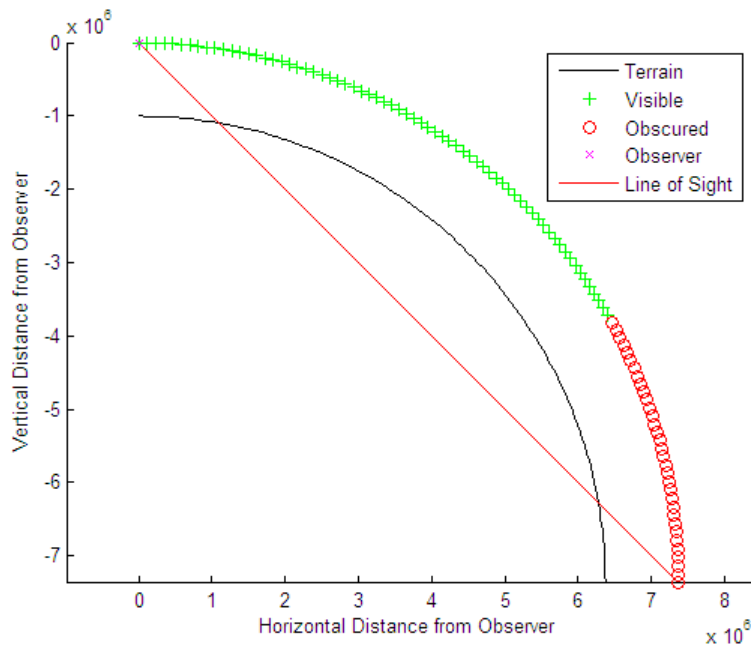
See Also

geodetic2aer

Introduced before R2006a

ellipse1

Geographic ellipse from center, semimajor axes, eccentricity, and azimuth

Syntax

```
[lat,lon] = ellipse1(lat0,lon0,ellipse)
[lat,lon] = ellipse1(lat0,lon0,ellipse,offset)
[lat,lon] = ellipse1(lat0,lon0,ellipse,offset,az)
[lat,lon] = ellipse1(lat0,lon0,ellipse,offset,az,ellipsoid)
[lat,lon] = ellipse1( ___ ,angleUnit)
[lat,lon] = ellipse1(lat0,lon0,ellipse,offset,az,ellipsoid,angleUnit,npts)
[lat,lon] = ellipse1(trackStr,...)
mat = ellipse1(...)
```

Description

`[lat,lon] = ellipse1(lat0,lon0,ellipse)` computes ellipse(s) with center(s) at `lat0,lon0`. The ellipse is defined by the third input, which is of the form `[semimajor axis,eccentricity]`, where the eccentricity input can be a two-element row vector or a two-column matrix. The ellipse input must have the same number of rows as the input scalar or column vectors `lat0` and `lon0`. The input semimajor axis is in degrees of arc length on a sphere. All ellipses are oriented so that their major axes run north-south.

`[lat,lon] = ellipse1(lat0,lon0,ellipse,offset)` computes the ellipse(s) where the major axis is rotated from due north by an azimuth offset. The `offset` angle is measured clockwise from due north. If `offset = []`, then no offset is assumed.

`[lat,lon] = ellipse1(lat0,lon0,ellipse,offset,az)` uses the input `az` to define the ellipse arcs computed. The arc azimuths are measured clockwise from due north. If `az` is a column vector, then the arc length is computed from due north. If `az` is a two-column matrix, then the ellipse arcs are computed starting at the azimuth in the first column and ending at the azimuth in the second column. If `az = []`, then a complete ellipse is computed.

`[lat,lon] = ellipse1(lat0,lon0,ellipse,offset,az,ellipsoid)` computes the ellipse on the ellipsoid defined by the input `ellipsoid`. `ellipsoid` is a `referenceSphere`, `referenceEllipsoid`, or `oblateSpheroid` object, or a vector of the form `[semimajor_axis eccentricity]`. If omitted, the unit sphere, is assumed. When an ellipsoid is supplied, the input semimajor axis must be in the same units as the ellipsoid semimajor axes. In this calling form, the units of the ellipse semimajor axis are not assumed to be in degrees.

`[lat,lon] = ellipse1( ___ ,angleUnit)` where `angleUnit` defines the units of the inputs and outputs. `angleUnit` can be `'degrees'` or `'radians'`. If you omit `angleUnit`, `ellipse1` uses `'degrees'`.

`[lat,lon] = ellipse1(lat0,lon0,ellipse,offset,az,ellipsoid,angleUnit,npts)` uses the scalar `npts` to determine the number of points per ellipse computed. If `npts` is omitted, 100 points are used.

`[lat,lon] = ellipse1(trackStr,...)` where `trackStr` specifies either great circle (`'gc'`) or rhumb line (`'rh'`) distances from the ellipse center.

mat = ellipse1(...) returns a single output argument where mat = [lat lon]. This is useful if only one ellipse is computed.

You can define multiple ellipses with a common center by providing scalar lat0 and lon0 inputs and a two-column ellipse matrix.

Examples

Create and plot the small ellipse centered at (0º,0º), with a semimajor axis of 10º and a semiminor axis of 5º.

```
axesm mercator
ecc = axes2ecc(10,5);
plotm(0,0,'r+')
[elat,elon] = ellipse1(0,0,[10 ecc],45);
plotm(elat,elon)
```

If the desired radius is known in some nonangular distance unit, use the radius returned by the earthRadius function as the ellipsoid input to set the range units. (Use an empty azimuth entry to specify a full ellipse.)

```
[elat,elon] = ellipse1(0,0,[550 ecc],45,[],earthRadius('nm'));
plotm(elat,elon,'m--')
```

For just an arc of the ellipse, enter an azimuth range:

```
[elat,elon] = ellipse1(0,0,[5 ecc],45,[-30 70]);
plotm(elat,elon,'c-')
```
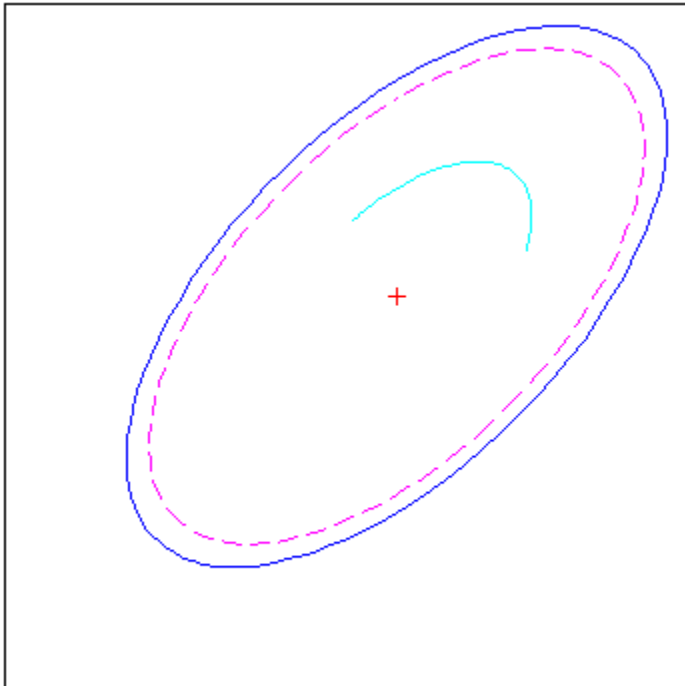
See Also

axes2ecc | scircle1 | track1

Introduced before R2006a

encodem

Fill in regular data grid from seed values and locations

Syntax

```
newgrid = encodem(Z,seedmat)
newgrid = encodem(Z,seedmat,stopvals)
```

Description

`newgrid = encodem(Z,seedmat)` fills in regions of the input data grid, Z, with desired new values. The boundary consists of the edges of the matrix and any entries with the value 1. The *seeds*, or starting points, and the values associated with them, are specified by the three-column matrix `seedmat`, the rows of which have the form `[row column value]`.

`newgrid = encodem(Z,seedmat,stopvals)` allows you to specify a vector, `stopvals`, of stopping values. Any value that is an element of `stopvals` will act as a boundary.

This function *fills in* regions of data grids with desired values. If a *boundary* exists, the new value replaces all entries in all four directions until the boundary is reached. The boundary is made up of selected stopping values and the edges of the matrix. The new value tries to flood the region exhaustively, stopping only when no new spaces can be reached by moving up, down, left, or right without hitting a stopping value.

Examples

For this imaginary map, fill in the upper right region with 7s and the lower left region with 3s:

```
Z = eye(4)

Z =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1

newgrid = encodem(Z,[4,1,3; 1,4,7])

newgrid =
     1     7     7     7
     3     1     7     7
     3     3     1     7
     3     3     3     1
```

See Also
`imbedm`

Introduced before R2006a

enu2aer

Transform local east-north-up coordinates to local spherical

Syntax

```
[az,elev,slantRange] = enu2aer(xEast,yNorth,zUp)
[ ___ ] = enu2aer( ___ ,angleUnit)
```

Description

`[az,elev,slantRange] = enu2aer(xEast,yNorth,zUp)` transforms the local east-north-up (ENU) Cartesian coordinates specified by `xEast`, `yNorth`, and `zUp` to the local azimuth-elevation-range (AER) spherical coordinates specified by `az`, `elev`, and `slantRange`. Both coordinate systems use the same local origin. Each input argument must match the others in size or be scalar.

`[ ___ ] = enu2aer( ___ ,angleUnit)` specifies the units for azimuth and elevation. Specify `angleUnit` as `'degrees'` (the default) or `'radians'`.

Examples

Calculate AER Coordinates from ENU Coordinates

Find the AER coordinates of a ground vehicle with respect to a parking gate, using the ENU coordinates of the vehicle with respect to the same gate.

First, specify the ENU coordinates of the vehicle. For this example, specify the coordinates in meters.

```
xEast = 8.4504;
yNorth = 12.4737;
zUp = 1.1046;
```

Then, calculate the AER coordinates of the vehicle. The azimuth and elevation are specified in degrees. The units for the slant range match the units specified by the ENU coordinates. Thus, the slant range is specified in meters.

```
[az,elev,slantRange] = enu2aer(xEast,yNorth,zUp)
```

```
az = 34.1160
```

```
elev = 4.1931
```

```
slantRange = 15.1070
```

Reverse the transformation using the `aer2enu` function.

```
[xEast,yNorth,zUp] = aer2enu(az,elev,slantRange)
```

```
xEast = 8.4504
```

```
yNorth = 12.4737
```

```
zUp = 1.1046
```

Input Arguments

xEast — ENU *x*-coordinates
scalar | vector | matrix | N-D array

ENU *x*-coordinates of one or more points in the local ENU system, specified as a scalar, vector, matrix, or N-D array.

Data Types: `single` | `double`

yNorth — ENU *y*-coordinates
scalar | vector | matrix | N-D array

ENU *y*-coordinates of one or more points in the local ENU system, specified as a scalar, vector, matrix, or N-D array.

Data Types: `single` | `double`

zUp — ENU *z*-coordinates
scalar value | vector | matrix | N-D array

ENU *z*-coordinates of one or more points in the local ENU system, specified as a scalar, vector, matrix, or N-D array.

Data Types: `single` | `double`

angleUnit — Angle units
`'degrees'` (default) | `'radians'`

Angle units, specified as `'degrees'` (the default) or `'radians'`.

Output Arguments

az — Azimuth angles
scalar | vector | matrix | N-D array

Azimuth angles of one or more points in the local AER system, returned as a scalar, vector, matrix, or N-D array. Azimuths are measured clockwise from north. Values are specified in degrees within the half-open interval [0 360). To use values in radians, specify the `angleUnit` argument as `'radians'`.

elev — Elevation angles
scalar | vector | matrix | N-D array

Elevation angles of one or more points in the local AER system, returned as a scalar, vector, matrix, or N-D array. Elevations are calculated with respect to the `xEast-yNorth` plane that contains the local origin. If the local origin is on the surface of the spheroid, then the `xEast-yNorth` plane is tangent to the spheroid.

Values are specified in degrees within the closed interval [-90 90]. Positive elevations correspond to positive `zUp` values, and negative elevations correspond to negative `zUp` values. An elevation of 0 indicates that the point lies in the `xEast-yNorth` plane. To use values in radians, specify the `angleUnit` argument as `'radians'`.

slantRange — Distances from local origin
scalar | vector | matrix | N-D array

Distances from the local origin, returned as a scalar, vector, matrix, or N-D array. Each distance is calculated along a straight, 3-D, Cartesian line. Values are returned in the units specified by `xEast`, `yNorth`, and `zUp`.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`aer2enu` | `ned2aer` | `enu2ecef` | `enu2geodetic`

Topics
"Choose a 3-D Coordinate System"

Introduced in R2012b

enu2ecef

Transform local east-north-up coordinates to geocentric Earth-centered Earth-fixed

Syntax

```
[X,Y,Z] = enu2ecef(xEast,yNorth,zUp,lat0,lon0,h0,spheroid)
[ ___ ] = enu2ecef( ___ ,angleUnit)
```

Description

[X,Y,Z] = enu2ecef(xEast,yNorth,zUp,lat0,lon0,h0,spheroid) transforms the local east-north-up (ENU) Cartesian coordinates specified by xEast, yNorth, and zUp to the geocentric Earth-centered Earth-fixed (ECEF) Cartesian coordinates specified by X, Y, and Z. Specify the origin of the local ENU system with the geodetic coordinates lat0, lon0, and h0. Each coordinate input argument must match the others in size or be scalar. Specify spheroid as the reference spheroid for the geodetic coordinates.

[___] = enu2ecef(___ ,angleUnit) specifies the units for latitude and longitude. Specify angleUnit as 'degrees' (the default) or 'radians'.

Examples

Calculate ECEF Coordinates from ENU Coordinates

Find the ECEF coordinates of orbital debris, using the ENU coordinates of the debris relative to the geodetic coordinates of a satellite.

First, specify the reference spheroid as WGS84 with length units measured in kilometers. For more information about WGS84, see "Reference Spheroids". The units for the ellipsoidal height, ENU coordinates, and ECEF coordinates must match the units specified by the LengthUnit property of the reference spheroid.

```
wgs84 = wgs84Ellipsoid('kilometer');
```

Specify the geodetic coordinates of the local origin. In this example, the local origin is the satellite. Specify h0 as ellipsoidal height in kilometers.

```
lat0 = 45.9132;
lon0 = 36.7484;
h0 = 1877.7532;
```

Specify the ENU coordinates of the point of interest. In this example, the point of interest is the orbital debris.

```
xEast = 355.6013;
yNorth = -923.0832;
zUp = 1041.0164;
```

Then, calculate the ECEF coordinates of the debris. In this example, the results display in scientific notation.

```
[x,y,z] = enu2ecef(xEast,yNorth,zUp,lat0,lon0,h0,wgs84)

x = 5.5075e+03

y = 4.5562e+03

z = 6.0128e+03
```

Reverse the transformation using the `ecef2enu` function. In this example, `zUp` displays in scientific notation.

```
[xEast,yNorth,zUp] = ecef2enu(x,y,z,lat0,lon0,h0,wgs84)

xEast = 355.6013

yNorth = -923.0832

zUp = 1.0410e+03
```

Input Arguments

xEast — ENU *x*-coordinates
scalar | vector | matrix | N-D array

ENU *x*-coordinates of one or more points in the local ENU system, specified as a scalar, vector, matrix, or N-D array. Specify values in units that match the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Data Types: `single` | `double`

yNorth — ENU *y*-coordinates
scalar | vector | matrix | N-D array

ENU *y*-coordinates of one or more points in the local ENU system, specified as a scalar, vector, matrix, or N-D array. Specify values in units that match the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Data Types: `single` | `double`

zUp — ENU *z*-coordinates
scalar | vector | matrix | N-D array

ENU *z*-coordinates of one or more points in the local ENU system, specified as a scalar, vector, matrix, or N-D array. Specify values in units that match the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Data Types: `single` | `double`

lat0 — Geodetic latitude of local origin
scalar | vector | matrix | N-D array

Geodetic latitude of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

`lon0` — Geodetic longitude of local origin
scalar | vector | matrix | N-D array

Geodetic longitude of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

`h0` — Ellipsoidal height of local origin
scalar | vector | matrix | N-D array

Ellipsoidal height of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify values in units that match the `LengthUnit` property of the `spheroid` object. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

For more information about ellipsoidal height, see "Find Ellipsoidal Height from Orthometric Height".

Data Types: `single` | `double`

`spheroid` — Reference spheroid
`referenceEllipsoid` object | `oblateSpheroid` object | `referenceSphere` object

Reference spheroid, specified as a `referenceEllipsoid` object, `oblateSpheroid` object, or `referenceSphere` object. The term reference spheroid is used synonymously with reference ellipsoid. To create a reference spheroid, use the creation function for the object. To specify the reference ellipsoid for WGS84, use the `wgs84Ellipsoid` function.

For more information about reference spheroids, see "Reference Spheroids".

Example: `spheroid = referenceEllipsoid('GRS 80');`

`angleUnit` — Angle units
`'degrees'` (default) | `'radians'`

Angle units, specified as `'degrees'` (the default) or `'radians'`.

Output Arguments

`X` — ECEF *x*-coordinates
scalar | vector | matrix | N-D array

ECEF *x*-coordinates of one or more points in the geocentric ECEF system, returned as a scalar, vector, matrix, or N-D array. Units are specified by the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

`Y` — ECEF *y*-coordinates
scalar | vector | matrix | N-D array

ECEF *y*-coordinates of one or more points in the geocentric ECEF system, returned as a scalar, vector, matrix, or N-D array. Units are specified by the `LengthUnit` property of the `spheroid`

argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Z — ECEF *z*-coordinates
scalar | vector | matrix | N-D array

ECEF *z*-coordinates of one or more points in the geocentric ECEF system, returned as a scalar, vector, matrix, or N-D array. Units are specified by the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Tips

To transform vectors instead of coordinate locations, use the `enu2ecefv` function.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`ned2ecef` | `aer2ecef` | `ecef2enu` | `enu2geodetic`

Topics
"Choose a 3-D Coordinate System"

Introduced in R2012b

enu2ecefv

Rotate local east-north-up vector to geocentric Earth-centered Earth-fixed

Syntax

```
[U,V,W] = enu2ecefv(uEast,vNorth,wUp,lat0,lon0)
[ ___ ] = enu2ecefv( ___ ,angleUnit)
```

Description

[U,V,W] = enu2ecefv(uEast,vNorth,wUp,lat0,lon0) returns vector components U, V, and W in a geocentric Earth-centered Earth-fixed (ECEF) system corresponding to vector components uEast, vNorth, and wUp in a local east-north-up (ENU) system. Specify the origin of the system with the geodetic coordinates lat0 and lon0. Each coordinate input argument must match the others in size or be scalar.

[___] = enu2ecefv(___ ,angleUnit) specifies the units for latitude and longitude. Specify angleUnit as 'degrees' (the default) or 'radians'.

Examples

Calculate ECEF Vector Components from ENU Components

Find the ECEF velocity components of a ground vehicle using its ENU velocity components.

Specify the geodetic coordinates of the vehicle in degrees and the ENU velocity components in kilometers per hour.

```
lat0 = 17.41;
lon0 = 78.27;

uEast = -27.6190;
vNorth = -16.4298;
wUp = -0.3186;
```

Calculate the ECEF components of the vehicle. The units for the ECEF components match the units for the ENU components. Thus, the ECEF components are returned in kilometers per hour. The rotation performed by enu2ecefv does not affect the speed of the vehicle.

```
[U,V,W] = enu2ecefv(uEast,vNorth,wUp,lat0,lon0)
```

U = 27.9798

V = -1.0993

W = -15.7724

Reverse the rotation using the ecef2enuv function.

```
[uEast,vNorth,wUp] = ecef2enuv(U,V,W,lat0,lon0)
```

```
uEast = -27.6190

vNorth = -16.4298

wUp = -0.3186
```

Input Arguments

uEast — ENU *x*-components
scalar value | vector | matrix | N-D array

ENU *x*-components of one or more vectors, specified as a scalar value, vector, matrix, or N-D array.

Data Types: `single` | `double`

vNorth — ENU *y*-components
scalar value | vector | matrix | N-D array

ENU *y*-components of one or more vectors, specified as a scalar value, vector, matrix, or N-D array.

Data Types: `single` | `double`

wUp — ENU *z*-components
scalar value | vector | matrix | N-D array

ENU *z*-components of one or more vectors, specified as a scalar value, vector, matrix, or N-D array.

Data Types: `single` | `double`

lat0 — Geodetic latitude of local origin
scalar | vector | matrix | N-D array

Geodetic latitude of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

lon0 — Geodetic longitude of local origin
scalar | vector | matrix | N-D array

Geodetic longitude of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

angleUnit — Angle units
`'degrees'` (default) | `'radians'`

Angle units, specified as `'degrees'` (the default) or `'radians'`.

Output Arguments

U — ECEF *x*-components
scalar value | vector | matrix | N-D array

ECEF *x*-components of one or more vectors, returned as a scalar value, vector, matrix, or N-D array. Values are returned in the units specified by `uEast`, `vNorth`, and `wUp`.

V — ECEF *y*-components
scalar value | vector | matrix | N-D array

ECEF *y*-components of one or more vectors, returned as a scalar value, vector, matrix, or N-D array. Values are returned in the units specified by `uEast`, `vNorth`, and `wUp`.

W — ECEF *z*-components
scalar value | vector | matrix | N-D array

ECEF *z*-components of one or more vectors, returned as a scalar value, vector, matrix, or N-D array. Values are returned in the units specified by `uEast`, `vNorth`, and `wUp`.

Tips

To transform coordinate locations instead of vectors, use the `enu2ecef` function.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also
`ned2ecefv` | `ecef2enuv` | `ecef2nedv`

Topics
"Vectors in 3-D Coordinate Systems"

Introduced in R2012b

enu2geodetic

Transform local east-north-up coordinates to geodetic

Syntax

```
[lat,lon,h] = enu2geodetic(xEast,yNorth,zUp,lat0,lon0,h0,spheroid)
[ ___ ] = enu2geodetic( ___ ,angleUnit)
```

Description

`[lat,lon,h] = enu2geodetic(xEast,yNorth,zUp,lat0,lon0,h0,spheroid)` transforms the local east-north-up (ENU) Cartesian coordinates specified by `xEast`, `yNorth`, and `zUp` to the geodetic coordinates specified by `lat`, `lon`, and `h`. Specify the origin of the local ENU system with the geodetic coordinates `lat0`, `lon0`, and `h0`. Each coordinate input argument must match the others in size or be scalar. Specify `spheroid` as the reference spheroid for the geodetic coordinates.

`[ ___ ] = enu2geodetic( ___ ,angleUnit)` specifies the units for latitude and longitude. Specify `angleUnit` as `'degrees'` (the default) or `'radians'`.

Examples

Calculate Geodetic Coordinates from ENU Coordinates

Find the geodetic coordinates of the Matterhorn, using the ENU coordinates of the Matterhorn with respect to the geodetic coordinates of Zermatt, Switzerland.

First, specify the reference spheroid as WGS84. For more information about WGS84, see "Reference Spheroids". The units for ellipsoidal height and ENU coordinates must match the units specified by the `LengthUnit` property of the reference spheroid. The default length unit for the reference spheroid created by `wgs84Ellipsoid` is `'meter'`.

```
wgs84 = wgs84Ellipsoid;
```

Specify the geodetic coordinates of the local origin. In this example, the local origin is Zermatt. Specify `h0` as ellipsoidal height in meters.

```
lat0 = 46.017;
lon0 = 7.750;
h0 = 1673;
```

Specify the ENU coordinates of the point of interest. In this example, the point of interest is the Matterhorn.

```
xEast = -7134.8;
yNorth = -4556.3;
zUp = 2852.4;
```

Then, calculate the geodetic coordinates of the Matterhorn. The result `h` is the ellipsoidal height of the mountain in meters. To view the results in standard notation, specify the display format as `shortG`.

```
format shortG
[lat,lon,h] = enu2geodetic(xEast,yNorth,zUp,lat0,lon0,h0,wgs84)

lat =
      45.976


lon =
       7.658


h =
        4531
```

Reverse the transformation using the `geodetic2enu` function.

```
[xEast,yNorth,zUp] = geodetic2enu(lat,lon,h,lat0,lon0,h0,wgs84)

xEast =
      -7134.8


yNorth =
      -4556.3


zUp =
       2852.4
```

Input Arguments

xEast — ENU *x*-coordinates
scalar | vector | matrix | N-D array

ENU *x*-coordinates of one or more points in the local ENU system, specified as a scalar, vector, matrix, or N-D array. Specify values in units that match the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Data Types: `single` | `double`

yNorth — ENU *y*-coordinates
scalar | vector | matrix | N-D array

ENU *y*-coordinates of one or more points in the local ENU system, specified as a scalar, vector, matrix, or N-D array. Specify values in units that match the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Data Types: `single` | `double`

zUp — ENU *z*-coordinates
scalar | vector | matrix | N-D array

ENU *z*-coordinates of one or more points in the local ENU system, specified as a scalar, vector, matrix, or N-D array. Specify values in units that match the `LengthUnit` property of the `spheroid`

argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Data Types: `single` | `double`

lat0 — Geodetic latitude of local origin
scalar | vector | matrix | N-D array

Geodetic latitude of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

lon0 — Geodetic longitude of local origin
scalar | vector | matrix | N-D array

Geodetic longitude of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

h0 — Ellipsoidal height of local origin
scalar | vector | matrix | N-D array

Ellipsoidal height of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify values in units that match the `LengthUnit` property of the `spheroid` object. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

For more information about ellipsoidal height, see "Find Ellipsoidal Height from Orthometric Height".

Data Types: `single` | `double`

spheroid — Reference spheroid
`referenceEllipsoid` object | `oblateSpheroid` object | `referenceSphere` object

Reference spheroid, specified as a `referenceEllipsoid` object, `oblateSpheroid` object, or `referenceSphere` object. The term reference spheroid is used synonymously with reference ellipsoid. To create a reference spheroid, use the creation function for the object. To specify the reference ellipsoid for WGS84, use the `wgs84Ellipsoid` function.

For more information about reference spheroids, see "Reference Spheroids".

Example: `spheroid = referenceEllipsoid('GRS 80');`

angleUnit — Angle units
`'degrees'` (default) | `'radians'`

Angle units, specified as `'degrees'` (the default) or `'radians'`.

Output Arguments

lat — Geodetic latitude
scalar | vector | matrix | N-D array

Geodetic latitude of one or more points, returned as a scalar, vector, matrix, or N-D array. Values are specified in degrees within the interval `[-90 90]`. To use values in radians, specify the `angleUnit` argument as `'radians'`.

lon — Geodetic longitude
scalar | vector | matrix | N-D array

Geodetic longitude of one or more points, returned as a scalar, vector, matrix, or N-D array. Values are specified in degrees within the interval `[-180 180]`. To use values in radians, specify the `angleUnit` argument as `'radians'`.

h — Ellipsoidal height
scalar | vector | matrix | N-D array

Ellipsoidal height of one or more points, returned as a scalar, vector, matrix, or N-D array. Values are specified in units that match the `LengthUnit` property of the `spheroid` object. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

For more information about ellipsoidal height, see "Find Ellipsoidal Height from Orthometric Height".

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also
aer2geodetic | enu2ecef | geodetic2enu | ned2geodetic

Topics
"Choose a 3-D Coordinate System"

Introduced in R2012b

epsm

Accuracy in angle units for certain map computations

Note epsm will be removed in a future release. If necessary, you can replace the expressions listed in the following table with the constants shown:

epsm()	1.0E-6
epsm('deg')	1.0E-6
epsm('rad')	deg2rad(1.0E-6)

Syntax

```
epsm
epsm(units)
```

Description

epsm is the limit of map angular precision. It is useful in avoiding trigonometric singularities, among other things.

epsm(units) returns the same angle in units corresponding to any valid angle units character vector. The default is 'degrees'.

Examples

The value of epsm is 10^{-6} degrees. To put this in perspective, in terms of an angular arc length, the distance is

```
epsmkm = deg2km(epsm)

epsmkm =
   1.1119e-04      % kilometers
```

This is about 11 centimeters, a very small distance on a global scale.

See Also
roundn

Introduced before R2006a

eqa2grn

Convert from equal area to Greenwich coordinates

Syntax

```
[lat,lon] = eqa2grn(x,y)
[lat,lon] = eqa2grn(x,y,origin)
[lat,lon] = eqa2grn(x,y,origin,ellipsoid)
[lat,lon] = eqa2grn(x,y,origin,units)
mat = eqa2grn(x,y,origin...)
```

Description

`[lat,lon] = eqa2grn(x,y)` converts the equal-area coordinate points `x` and `y` to the Greenwich (standard geographic) coordinates `lat` and `lon`.

`[lat,lon] = eqa2grn(x,y,origin)` specifies the location in the Greenwich system of the *x-y* origin (0,0). The two-element vector `origin` must be of the form `[latitude longitude]`. The default places the origin at the Greenwich coordinates (0º,0º).

`[lat,lon] = eqa2grn(x,y,origin,ellipsoid)` specifies the ellipsoidal model of the figure of the Earth using `ellipsoid`. `ellipsoid` is a `referenceSphere`, `referenceEllipsoid`, or `oblateSpheroid` object, or a vector of the form `[semimajor_axis eccentricity]`. The `ellipsoid` is a unit sphere by default.

`[lat,lon] = eqa2grn(x,y,origin,units)` specifies the units for the outputs, where `units` is any valid angle units value. The default value is `'degrees'`.

`mat = eqa2grn(x,y,origin...)` packs the outputs into a single variable.

This function converts data from equal-area *x-y* coordinates to geographic (latitude-longitude) coordinates. The opposite conversion can be performed with `grn2eqa`.

Examples

```
[lat,lon] = eqa2grn(.5,.5)

lat =
    30.0000
lon =
    28.6479
```

See Also
`grn2eqa` | `hista`

Introduced before R2006a

etopo

(To be removed) Read gridded global relief data (ETOPO products)

Note `etopo` will be removed in a future release. Use `readgeoraster` instead. For more information, see "Compatibility Considerations".

Syntax

```
[Z, refvec] = etopo
[Z, refvec] = etopo(samplefactor)
[Z, refvec] = etopo(samplefactor, latlim, lonlim)
[Z, refvec] = etopo(folder, ...)
[Z, refvec] = etopo(filename, ...)
[Z, refvec] = etopo({'etopo5.northern.bat', 'etopo5.southern.bat'}, ...)
```

Description

`[Z, refvec] = etopo` reads the ETOPO data for the entire world from the ETOPO data in the current folder. The `etopo` function searches the current folder first for ETOPO1c binary data, then ETOPO2V2c binary data, then ETOPO2 (2001) binary data, then ETOPO5 binary data, and finally ETOPO5 ASCII data. Once the function finds a case-insensitive file name match, it reads the data. See the table Supported ETOPO Data File Names for a list of possible file names. The `etopo` function returns the data grid, `Z`, as an array of elevations. Data values, in whole meters, represent the elevation of the center of each cell. `refvec`, the associated three-element referencing vector, geolocates Z.

`[Z, refvec] = etopo(samplefactor)` reads the data for the entire world, downsampling the data by `samplefactor`. The scalar integer `samplefactor` when equal to 1 gives the data at its full resolution (10800 by 21600 values for ETOPO1 data, 5400 by 10800 values for ETOPO2 data, and 2160 by 4320 values for ETOPO5 data). When `samplefactor` is an integer *n* greater than one, the `etopo` function returns every *n*th point. If you omit `samplefactor` or leave it empty, it defaults to 1. (If the `etopo` function reads an older, ASCII ETOPO5 data set, then `samplefactor` must divide evenly into the number of rows and columns of the data file.)

`[Z, refvec] = etopo(samplefactor, latlim, lonlim)` reads the data for the part of the world within the specified latitude and longitude limits. Specify the limits of the desired data as two-element vectors of latitude, `latlim`, and longitude, `lonlim`, in degrees. Specify the elements of `latlim` and `lonlim` in ascending order. Specify `lonlim` in the range [0 360] for ETOPO5 data and [-180 180] for ETOPO2 and ETOPO1 data. If `latlim` is empty, the latitude limits are [-90 90]. If `lonlim` is empty, the file type determines the longitude limits.

`[Z, refvec] = etopo(folder, ...)` allows you to use the variable `folder` to specify the path for the ETOPO data file. Otherwise, the `etopo` function searches the current folder for the data.

`[Z, refvec] = etopo(filename, ...)` reads the ETOPO data from the file specified by the case-insensitive string scalar or character vector `filename`. The name of the ETOPO file is as referenced in the ETOPO data file names table. Include the folder name in `filename` or place the file in the current folder or in a folder on the MATLAB path.

[Z, refvec] = etopo({'etopo5.northern.bat', 'etopo5.southern.bat'}, ...) reads the ETOPO data from the specified case-insensitive ETOPO5 ASCII data files. Place the files in the current folder or in a folder on the MATLAB path.

Examples

Read and display ETOPO2V2c data from the file 'ETOPO2V2c_i2_LSB.bin' downsampled to half-degree cell size and display the boundary of the land areas.

```
samplefactor = 15;
[Z, refvec] = etopo('ETOPO2V2c_i2_LSB.bin', samplefactor);
figure
worldmap world
geoshow(Z, refvec, 'DisplayType', 'texturemap');
demcmap(Z, 256);
geoshow('landareas.shp', 'FaceColor', 'none', ...
    'EdgeColor', 'black');
```
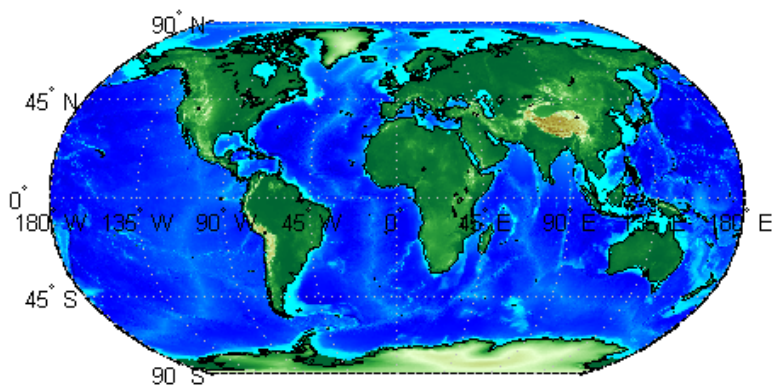

More About

ETOPO Models

According to the National Geophysical Data Center (NGDC) Web site, ETOPO models combine regional and global land topography and ocean bathymetry data from many data sources. ETOPO1, the most recent model, has an Ice Surface version showing the top of the Antarctic and Greenland ice sheets and a Bedrock version showing the bedrock below the ice sheets. For detailed information about the data sources used to create the ETOPO1 model, see the NGDC Web site. NGDC lists the ETOPO2 and ETOPO5 models as deprecated but still available.

Model	Cell Size	Date Available
ETOPO1	1-arc-minute	March 2009
ETOPO2v2	2-minute	2006
ETOPO2	2-minute	2001
ETOPO5	5-minute	1988

Tips

Supported ETOPO Data File Names

Format	File Names
ETOPO1c (cell)	• etopo1_ice_c.flt • etopo1_bed_c.flt • etopo1_ice_c_f4.flt • etopo1_bed_c_f4.flt • etopo1_ice_c_i2.bin • etopo1_bed_c_i2.bin
ETOPO2V2c (cell)	• ETOPO2V2c_i2_MSB.bin • ETOPO2V2c_i2_LSB.bin • ETOPO2V2c_f4_MSB.flt • ETOPO2V2c_f4_LSB.flt • ETOPO2V2c.hdf
ETOPO2 (2001)	• ETOPO2.dos.bin • ETOPO2.raw.bin
ETOPO5 (binary)	• ETOPO5.DOS • ETOPO5.DAT
ETOPO5 (ASCII)	• etopo5.northern.bat • etopo5.southern.bat

Compatibility Considerations

etopo will be removed
Warns starting in R2021b

Raster reading functions that return referencing vectors will be removed, including `etopo`. Instead, use `readgeoraster`, which returns a geographic raster reference object. Reference objects have several advantages over referencing vectors.

- Unlike referencing vectors, reference objects have properties that document the size of the associated raster, its geographic limits, and the direction of its rows and columns. For examples of reference object properties, see `GeographicCellsReference` and `GeographicPostingsReference`.
- You can manipulate the limits of rasters associated with reference objects using the `geocrop` function.
- You can manipulate the size and resolution of rasters associated with reference objects using the `georesize` function.
- Most functions that accept referencing vectors as input also accept reference objects.

This table shows some typical usages of `etopo` and how to update your code to use `readgeoraster` instead.

Will Be Removed	Recommended
`[Z,refvec] = etopo(filename);`	`[Z,R] = readgeoraster(filename);`
`[Z,refvec] = etopo(filename,samplefactor);`	`[Z,R] = readgeoraster(filename);` `[Z,R] = georesize(Z,R,1/samplefactor);`

`[Z,refvec] = etopo(filename,samplefactor,latlim,lonlim)`	`[Z,R] = readgeoraster(filename);`
	`[Z,R] = geocrop(Z,R,latlim,lonlim);`
	`[Z,R] = georesize(Z,R,1/scalefactor);`

Unlike `etopo`, the `readgeoraster` function does not support the ETOPO2 or ETOPO5 models. Instead, read the cell-referenced ETOPO1 model stored in the GeoTIFF format, and specify a geographic coordinate system type. For example, use `[Z,R] = readgeoraster(filename,'CoordinateSystemType','geographic')`. To obtain the resolution of the ETOPO2 or ETOPO5 models, resize the full-resolution ETOPO1 model using the `georesize` function with a scale of `1/2` or `1/5`, respectively.

The `readgeoraster` function returns data using the native data type embedded in the file. Return a different data type by specifying the `'OutputType'` name-value pair. For example, use `[Z,R] = readgeoraster(filename,'OutputType','double')`.

The `readgeoraster` function does not automatically replace missing data with `NaN` values. If your data set uses large negative numbers to indicate missing data, you can replace them with `NaN` values using the `standardizeMissing` function.

```
[Z,R] = readgeoraster('MtWashington-ft.grd');
info = georasterinfo('MtWashington-ft.grd');
m = info.MissingDataIndicator;
Z = standardizeMissing(Z,m);
```

References

[1] "2-minute Gridded Global Relief Data (ETOPO2v2)," U.S. Department of Commerce, National Oceanic and Atmospheric Administration, National Geophysical Data Center, 2006.

[2] Amante, C. and B. W. Eakins, "ETOPO1 1 Arc-Minute Global Relief Model: Procedures, Data Sources and Analysis," *NOAA Technical Memorandum NESDIS NGDC-24*, March 2009.

[3] "Digital Relief of the Surface of the Earth," *Data Announcement 88-MGG-02*, NOAA, National Geophysical Data Center, Boulder, Colorado, 1988.

[4] "ETOPO2v2 Global Gridded 2-minute Database," National Geophysical Data Center, National Oceanic and Atmospheric Administration, U.S. Dept. of Commerce.

See Also

readgeoraster | georasterinfo

Topics
"Find Geospatial Raster Data"

Introduced before R2006a

etopo5

Read global 5-min digital terrain data

Syntax

> **Note** etopo5 will be removed in a future release; use readgeoraster instead.

```
[Z, refvec] = etopo5
[Z, refvec] = etopo5(samplefactor)
[[Z, refvec] = etopo5(samplefactor, latlim, lonlim)
[Z, refvec] = etopo5(folder, ...)
[Z, refvec] = etopo5(file, ...)
```

Description

[Z, refvec] = etopo5 reads the topography data for the entire world for the data in the current folder. The current folder is searched first for ETOPO2 binary data, followed by ETOPO5 binary data, followed by ETOPO5 ASCII data from the file names etopo5.northern.bat and etopo5.southern.bat. Once a match is found the data is read. The data grid, Z, is returned as an array of elevations. Data values are in whole meters, representing the elevation of the center of each cell. refvec is the associated three-element referencing vector that geolocates Z.

[Z, refvec] = etopo5(samplefactor) reads the data for the entire world, downsampling the data by samplefactor. samplefactor is a scalar integer, which when equal to 1 gives the data at its full resolution (1080 by 4320 values). When samplefactor is an integer n greater than one, every nth point is returned. samplefactor must divide evenly into the number of rows and columns of the data file. If samplefactor is omitted or empty, it defaults to 1.

[[Z, refvec] = etopo5(samplefactor, latlim, lonlim) reads the data for the part of the world within the specified latitude and longitude limits. The limits of the desired data are specified as two-element vectors of latitude, latlim, and longitude, lonlim, in degrees. The elements of latlim and lonlim must be in ascending order. If latlim is empty the latitude limits are [-90 90]. lonlim must be specified in the range [0 360]. If lonlim is empty, the longitude limits are [0 360].

[Z, refvec] = etopo5(folder, ...) allows the path for the data file to be specified by folder rather than the current folder.

[Z, refvec] = etopo5(file, ...) reads the data from file, where file is a character vector or a cell array of character vectors containing the name or names of the data files.

> **Note** For details on locating map data for download over the Internet, see the following documentation at the MathWorks Web Site: "Find Geospatial Data Online".

Examples

Example 1

Read every tenth point in the data set:

```
% Read and display the ETOPO5 data from the folder 'etopo5'
% downsampled by a factor of 10.
[Z, refvec] = etopo5('etopo5',10);
axesm merc
geoshow(Z, refvec, 'DisplayType', 'surface');
demcmap(Z);
```

Example 2

Read in data for Korea and Japan at the full resolution:

```
samplefactor = 1; latlim = [30 45]; lonlim = [115 145];
[Z,refvec] = etopo5(samplefactor,latlim,lonlim);
whos Z

  Name      Size          Bytes  Class
  Z       180x360        518400  double array
```

See Also

```
readgeoraster
```

Introduced before R2006a

extractfield

Field values from structure array

Syntax

```
a = extractfield(S,name)
```

Description

`a = extractfield(S,name)` returns the field values specified by the field `name` of structure `S`.

Examples

Extract Fields From Structure

Load a structure that contains information about roads in Concord, MA.

```
roads = shaperead('concord_roads.shp');
r = roads(1:5)
```

```
r=5×1 struct array with fields:
    Geometry
    BoundingBox
    X
    Y
    STREETNAME
    RT_NUMBER
    CLASS
    ADMIN_TYPE
    LENGTH
```

Get the *x*- and *y*-coordinates of the roads. Display the map, and highlight the first few elements using the color magenta.

```
hold on
plot(extractfield(roads,'X'),extractfield(roads,'Y'));
plot(extractfield(r,'X'),extractfield(r,'Y'),'m');
```

Extract the names of the roads, stored in the field STREETNAME. The field values are character vectors, so the result is returned in a cell array.

```
names = extractfield(r,'STREETNAME')
```

```
names = 1x5 cell
  Columns 1 through 4

    {0x0 char}    {'WRIGHT FARM'}    {'WRIGHT FARM'}    {'WRIGHT FARM'}

  Column 5

    {'WRIGHT FARM'}
```

Extract the X field from the structure and examine the format of the returned values. All values have the same numeric data type (double), so the result is returned in a vector.

```
uniformType = extractfield(r,'X')
```

```
uniformType = 1×42
10⁵ ×

    2.0884    2.0884    2.0884    2.0883    2.0883    2.0882    2.0882    2.0882    2.0882    2.(
```

For illustrative purposes, change the X field value in one of the elements to have a different data type. This command converts the second element to data type single.

```
r(2).X = single(r(2).X);
```

Extract the X field values again. This time, the values have different data types, so the result is returned in a cell array.

```
mixedType = extractfield(r,'X')
```

```
mixedType=1×5 cell array
  Columns 1 through 3

    {[2.0884e+05 ... ]}    {[2.0884e+05 ... ]}    {[2.0891e+05 ... ]}

  Columns 4 through 5

    {[2.0898e+05 ... ]}    {[2.0891e+05 ... ]}
```

Input Arguments

S — Structure
structure

Structure, specified as a structure.

name — Field name
string scalar | character vector

Field name, specified as a case-sensitive string scalar or character vector.

Output Arguments

a — Extracted field values
1-by-*n* numeric vector | 1-by-*n* cell array

Extracted field values, returned as a 1-by-*n* numeric vector or cell array. *n* is the total number of elements in the field name of structure S, that is, `n = numel([S(:).(name)])`. a is a cell array if any field values in the field name contain a character vector or if the field values are not uniform in type; otherwise a is the same type as the field values. The shape of the input field is not preserved in a.

See Also
struct | shaperead

Introduced before R2006a

extractm

Coordinate data from line or patch display structure

Note `extractm` will be removed in a future release. The use of display structures is not recommended. Use `geoshape` vectors instead.

Syntax

```
[lat,lon] = extractm(display_struct,object_str)
[lat,lon] = extractm(display_struct,object_strings)
[lat,lon] = extractm(display_struct,object_strings,searchmethod)
[lat,lon] = extractm(display_struct)
[lat,lon,indx] = extractm(...)
mat = extractm(...)
```

Description

`[lat,lon] = extractm(display_struct,object_str)` extracts latitude and longitude coordinates from those elements of `display_struct` having `'tag'` fields that begin with the string scalar or character vector specified by `object_str`. `display_struct` is a Mapping Toolbox display structure in which the `'type'` field has a value of either `'line'` or `'patch'`. The output `lat` and `lon` vectors include `NaN`s to separate the individual map features. The comparison of `'tag'` values is not case-sensitive.

`[lat,lon] = extractm(display_struct,object_strings)` selects features with `'tag'` fields matching any of several different string scalar or character vectors. `object_strings` is a string scalar, character vector, cell array of character vectors, or character array. `extractm` strips trailing spaces from features listed in character arrays before matching.

`[lat,lon] = extractm(display_struct,object_strings,searchmethod)` specifies the method used to match the values of the `'tag'` field in `display_struct`. `searchmethod` can be one of these values:

`'strmatch'`	Search for matches at the beginning of the tag
`'findstr'`	Search within the tag
`'exact'`	Search for exact matches. Note that when *searchmethod* is specified the search is case-sensitive.

`[lat,lon] = extractm(display_struct)` extracts all vector data from the input map structure.

`[lat,lon,indx] = extractm(...)` also returns the vector `indx` identifying which elements of `display_struct` met the selection criteria.

`mat = extractm(...)` returns the vector data in a single matrix, where `mat = [lat lon]`.

Examples

Extract the District of Columbia from the low-resolution U.S. vector data:

```
load greatlakes
[lat, lon] = extractm(greatlakes, 'Erie');
axesm mercator
geoshow(lat,lon, 'DisplayType','polygon', 'FaceColor','blue')
```
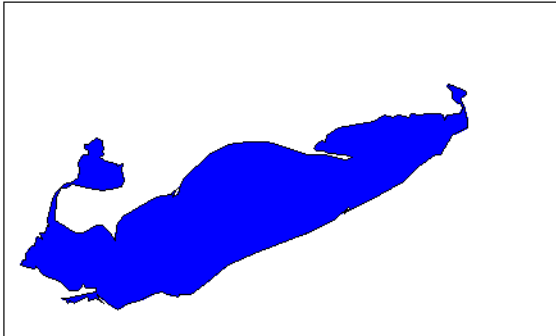

Tips

A Version 1 display structure is a MATLAB structure that can contain line, patch, text, regular data grid, geolocated data grid, and certain other objects and fixed attributes. In Mapping Toolbox Version 2, a new data structure for vector geodata was introduced (called a mapstruct or a geostruct, depending on whether coordinates it contains are projected or unprojected). Geostructs and mapstructs have few required fields and can include any number of user-defined fields, giving them much greater flexibility to represent vector geodata. For information about the contents and format of display structures, see "Version 1 Display Structures" on page 1-286 in the reference page for `displaym`. For information about converting display structures to geographic data structures, see the reference page for `updategeostruct`, which performs such conversions.

See Also

Functions
displaym | extractfield | geoshow | mapshow | updategeostruct | mlayers

Objects
geoshape

Introduced before R2006a

fieldnames

Return dynamic property names of geographic or planar vector

Syntax

```
names = fieldnames(v)
```

Description

`names = fieldnames(v)` returns the names of the dynamic properties of geographic or planar vector `v`.

Examples

View Dynamic Properties of a Mapshape Vector

Create a mapshape vector.

```
ms = mapshape(shaperead('tsunamis.shp'));
```

Display all dynamic properties of the mapshape vector. The displayed properties exclude the Collection properties `Geometry` and `Metadata` and the required mapshape Vertex properties X and Y.

```
fieldnames(ms)
```

```
ans = 18x1 cell
    {'Year'       }
    {'Month'      }
    {'Day'        }
    {'Hour'       }
    {'Minute'     }
    {'Second'     }
    {'Val_Code'   }
    {'Validity'   }
    {'Cause_Code' }
    {'Cause'      }
    {'Eq_Mag'     }
    {'Country'    }
    {'Location'   }
    {'Max_Height' }
    {'Iida_Mag'   }
    {'Intensity'  }
    {'Num_Deaths' }
    {'Desc_Deaths'}
```

Input Arguments

v — Geographic or planar vector
`geopoint`, `geoshape`, `mappoint`, or `mapshape` object

Geographic or planar vector, specified as a `geopoint`, `geoshape`, `mappoint`, or `mapshape` object.

Output Arguments

names — Names of dynamic properties
cell array

Names of dynamic properties of vector `v`, returned as a cell array.

See Also
`disp` | `properties`

Introduced in R2012a

fill3m

Project filled 3-D patch objects on map axes

Syntax

```
h = fill3m(lat,lon,z,cdata)
h = fill3m(lat,lon,z,PropertyName,PropertyValue,...)
```

Description

`h = fill3m(lat,lon,z,cdata)` projects and displays any patch object with vertices defined by vectors `lat` and `lon` to the current map axes. The scalar `z` indicates the altitude plane at which the patch is displayed. The input `cdata` defines the patch face color. The patch handle or handles, h, can be returned.

`h = fill3m(lat,lon,z,PropertyName,PropertyValue,...)` allows any property name/ property value pair supported by `patch` to be assigned to the `fill3m` object.

Examples

```
lat = [30 15 0 0 0 15 30 30]';
lon = [-60 -60 -60 0 60 60 60 0]';
axesm bonne; framem
view(3)
fill3m(lat,lon,2,'b')
```
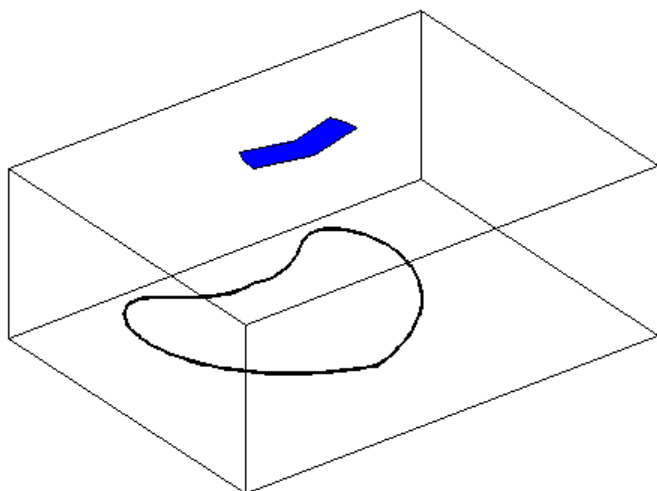

See Also
`fillm` | `patchesm` | `patchm`

Introduced before R2006a

fillm

Project filled 2-D patch objects on map axes

Syntax

```
h = fillm(lat,lon,cdata)
h = fillm(lat,lon,'PropertyName',PropertyValue,...)
```

Description

`h = fillm(lat,lon,cdata)` projects and displays any patch object with vertices defined by the vectors `lat` and `lon` to the current map axes. The input `cdata` defines the patch face color. The patch handle or handles, `h`, can be returned.

`h = fillm(lat,lon,'PropertyName',PropertyValue,...)` allows any property name/ property value pair supported by `patch` to be assigned to the `fillm` object.

Examples

```
lat = [30 15 0 0 0 15 30 30]';
lon = [-60 -60 -60 0 60 60 60 0]';
axesm bonne; framem
fillm(lat,lon,'b')
```
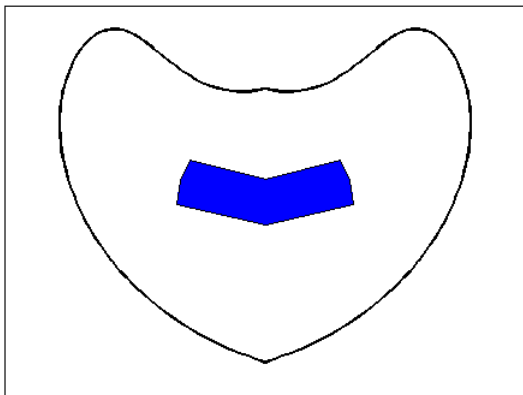

See Also
`fill3m` | `patchesm` | `patchm`

Introduced before R2006a

filterm

Filter latitudes and longitudes based on underlying data grid

Syntax

```
[latout,lonout] = filterm(lat,lon,Z,R,allowed)
[latout,lonout,indx] = filterm( ___ )
```

Description

`[latout,lonout] = filterm(lat,lon,Z,R,allowed)` filters a set of latitudes and longitudes to include only those data points which have a corresponding value in `Z` equal to `allowed`. `R` can be a geographic raster reference object, a referencing vector, or a referencing matrix.

`[latout,lonout,indx] = filterm( ___ )` filters a set of latitudes and longitudes, returning indices of the included points in `indx`.

Examples

Filter Elevation Data

Display points along the equator that are above sea level.

First, load elevation raster data and a geographic cells reference object. The raster contains terrain heights relative to mean sea level. Then, specify the coordinates of evenly spaced points along the equator.

```
load topo60c
lon = (0:5:360)';
lat = zeros(size(lon));
```

Create a logical array representing the terrain above sea level. Then, filter the points along the equator to include only the elements that contain `true`.

```
topoASL = topo60c > 0;
[newlat,newlon] = filterm(lat,lon,topoASL,topo60cR,1);
```

Create a map axes object for the world and display the elevation data. Display the all of the points along the equator using red markers. Then, display the points that are above sea level using yellow circles.

```
worldmap world
geoshow(topo60c,topo60cR,'DisplayType','texturemap')
demcmap(topo60c)
geoshow(lat,lon,'DisplayType','point','MarkerEdgeColor','r')
geoshow(newlat,newlon,'DisplayType','point','Marker','o',...
    'MarkerFaceColor','y')
```
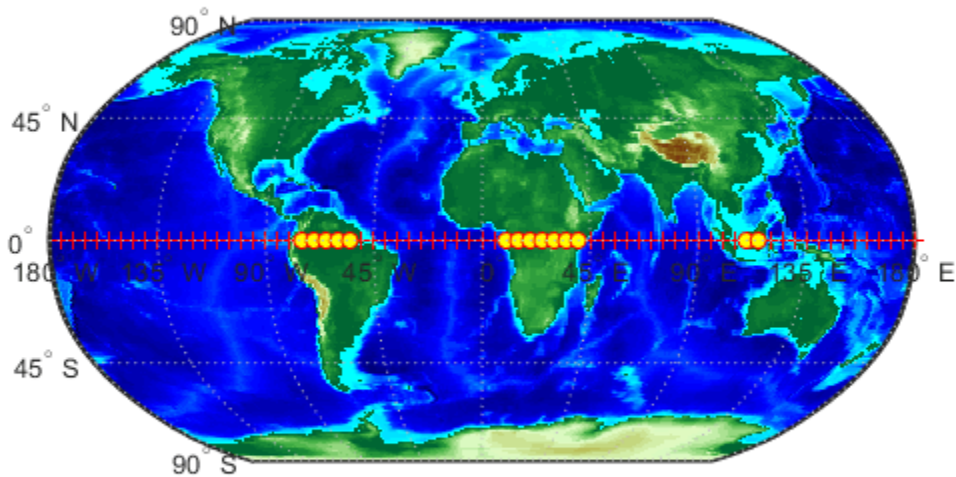
Input Arguments

lat — Latitude values
numeric array

Latitude values, specified as a numeric array.

Data Types: `single` | `double`

lon — Longitude values
numeric array

Longitude values, specified as a numeric array.

Data Types: `single` | `double`

Z — Filter
numeric array

Filter, specified as a numeric array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

R — Spatial referencing information
geographic raster reference object | referencing vector | referencing matrix

Spatial referencing information, specified as a geographic raster reference object, a referencing vector, or a referencing matrix.

If R is a geographic raster reference object, its `RasterSize` property must be consistent with `size(Z)`.

If R is a referencing vector, it must be a 1-by-3 with elements:

```
[cells/degree northern_latitude_limit western_longitude_limit]
```

If R is a referencing matrix, it must be 3-by-2 and transform raster row and column indices to or from geographic coordinates according to:

```
[lon lat] = [row col 1] * R
```

If R is a referencing matrix, it must define a (non-rotational, non-skewed) relationship in which each column of the data grid falls along a meridian and each row falls along a parallel. Nearest-neighbor interpolation is used by default. NaN is returned for points outside the grid limits or for which `lat` or `lon` contain NaN. All angles are in units of degrees.

Data Types: `double`

allowed — Allowed values
numeric array or character vector

Allowed values, specified as a numeric array or character vector.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical` | `char`

Output Arguments

latout — Latitudes of filtered points
numeric array

Latitudes of filtered points, returned as a numeric array.

lonout — Longitudes of filtered points
numeric array

Longitudes of filtered points, returned as a numeric array.

indx — Indices of filtered points
numeric array

Indices of filtered points, returned as a numeric array.

See Also
`imbedm` | `hista` | `histr`

Introduced before R2006a

findm

Latitudes and longitudes of nonzero data grid elements

Syntax

```
[lat,lon] = findm(Z,R)
[lat,lon] = findm(latz,lonz,Z)
[lat,lon,val] = findm(...)
mat = findm(...)
```

Description

`[lat,lon] = findm(Z,R)` computes the latitudes and longitudes of the nonzero elements of a regular data grid, Z. R can be a geographic raster reference object, a referencing vector, or a referencing matrix.

If R is a geographic raster reference object, its `RasterSize` property must be consistent with `size(Z)`.

If R is a referencing vector, it must be a 1-by-3 with elements:

`[cells/degree northern_latitude_limit western_longitude_limit]`

If R is a referencing matrix, it must be 3-by-2 and transform raster row and column indices to or from geographic coordinates according to:

`[lon lat] = [row col 1] * R`

If R is a referencing matrix, it must define a (non-rotational, non-skewed) relationship in which each column of the data grid falls along a meridian and each row falls along a parallel. Nearest-neighbor interpolation is used by default. NaN is returned for points outside the grid limits or for which `lat` or `lon` contain NaN. All angles are in units of degrees.

`[lat,lon] = findm(latz,lonz,Z)` returns the latitudes and longitudes of the nonzero elements of a geolocated data grid Z, which is an M-by-N logical or numeric array. Typically `latz` and `lonz` are M-by-N latitude-longitude arrays, but `latz` may be a latitude vector of length M and `lonz` may be a longitude vector of length N.

`[lat,lon,val] = findm(...)` returns the values of the nonzero elements of Z, in addition to their locations.

`mat = findm(...)` returns a single output, where `mat = [lat lon]`.

This function works in two modes: with a regular data grid and with a geolocated data grid.

Examples

The data grid can be the result of a logical operation. For example, load elevation raster data and a geographic cells reference object. Then, find all locations with elevations greater than 5500 meters.

```
load topo60c
[lat,lon] = findm((topo60c > 5500),topo60cR);
[lat lon]

ans =
   34.5000    79.5000
   34.5000    80.5000
   30.5000    84.5000
   28.5000    86.5000
```

These points are in the Himalayas. Find the grid values at these locations using the geographicToDiscrete and sub2ind functions.

```
[row,col] = geographicToDiscrete(topo60cR,lat,lon);
indx = sub2ind(size(topo60c),row,col);
heights = topo60c(indx)

heights =
         5559
         5515
         5523
         5731
```

See Also
find | geographicToDiscrete

Introduced before R2006a

fipsname

Read Federal Information Processing Standard (FIPS) name file used with TIGER thinned boundary files

> **Note** `fipsname` will be removed in a future release. More recent TIGER/Line data sets are available in shapefile format and can be imported using `shaperead`.

Syntax

```
struc = fipsname
struc = fipsname(filename)
```

Description

`struc = fipsname` opens a file selection window to pick the file, reads the FIPS codes, and returns them in a structure.

`struc = fipsname(filename)` reads the specified file.

Background

The TIGER thinned boundary files provided by the U.S. Census use FIPS codes to identify geographic entities. This function reads the FIPS files as provided with the TIGER files. These files generally have names of the format *_name.dat*.

Examples

```
struc = fipsname('st_name.dat')

struc =
1x57 struct array with fields:
    name
    id

s(1)

ans =
    name: 'Alabama'
      id: 1
```

Tips

The FIPS name files, along with TIGER thinned boundary files, are available over the Internet.

Introduced before R2006a

firstCornerX

Package: `map.rasterref`

Return world *x*-coordinate of map raster index (1,1)

Syntax

```
x = firstCornerX(R)
```

Description

`x = firstCornerX(R)` returns the world *x*-coordinate of the outermost corner of the first element (1,1) of map raster R.

Examples

Find *x*-Coordinate of First Corner

Import a GeoTIFF image [1] on page 1-0   of Boston as an array and a `MapCellsReference` object. Find the world *x*-coordinate of the outermost corner of the first cell.

```
[Z,R] = readgeoraster('boston.tif');
x = firstCornerX(R)
```

```
x = 7.6432e+05
```

[1] The data used in this example includes material copyrighted by GeoEye, all rights reserved.

Input Arguments

R — Map raster
`MapCellsReference` or `MapPostingsReference` object

Map raster, specified as a `MapCellsReference` or `MapPostingsReference` object.

Output Arguments

x — World *x*-coordinate
numeric scalar

World *x*-coordinate, returned as a numeric scalar.

Data Types: `double`

See Also
`firstCornerY`

Introduced in R2013b

firstCornerY

Package: `map.rasterref`

Return world *y*-coordinate of map raster index (1,1)

Syntax

```
y = firstCornerY(R)
```

Description

`y = firstCornerY(R)` returns the world *y*-coordinate of the outermost corner of the first element (1,1) of map raster R.

Examples

Find *y*-Coordinate of First Corner

Import a GeoTIFF image [1] on page 1-0 of Boston as an array and a `MapCellsReference` object. Find the world *y*-coordinate of the outermost corner of the first cell.

```
[Z,R] = readgeoraster('boston.tif');
y = firstCornerY(R)
```

```
y = 2.9606e+06
```

[1] The data used in this example includes material copyrighted by GeoEye, all rights reserved.

Input Arguments

R — Map raster
`MapCellsReference` or `MapPostingsReference` object

Map raster, specified as a `MapCellsReference` or `MapPostingsReference` object.

Output Arguments

y — World *y*-coordinate
numeric scalar

World *y*-coordinate, returned as a numeric scalar.

Data Types: `double`

See Also
`firstCornerX`

Introduced in R2013b

flat2ecc

Eccentricity of ellipse from flattening

Note Support for nonscalar input, including the special two-column syntax described below, will be removed in a future release.

Syntax

```
ecc = flat2ecc(f)
ecc = flat2ecc(f)
```

Description

`ecc = flat2ecc(f)` computes the eccentricity of an ellipse (or ellipsoid of revolution) given its flattening `f`. Except when the input has 2 columns (or is a row vector), each element is assumed to be a flattening and the output `ecc` has the same size as `f`.

`ecc = flat2ecc(f)`, where f has two columns (or is a row vector), assumes that the second column is a flattening, and a column vector is returned.

See Also
axes2ecc | ecc2flat | n2ecc

Introduced before R2006a

flatearthpoly

Clip polygon to world limits

Syntax

```
[latf,lonf] = flatearthpoly(lat,lon)
[latf,lonf] = flatearthpoly(lat,lon,longitudeOrigin)
```

Description

`[latf,lonf] = flatearthpoly(lat,lon)` trims `NaN`-separated polygons specified by the latitude and longitude vectors `lat` and `lon` to the limits `[-180 180]` in longitude and `[-90 90]` in latitude, inserting straight segments along the +/- 180-degree meridians and at the poles. Inputs and outputs are in degrees.

Display functions automatically cut and trim geographic data when required by the map projection. Use the `flatearthpoly` function only when performing set operations on polygons.

`[latf,lonf] = flatearthpoly(lat,lon,longitudeOrigin)` centers the longitude limits on the longitude specified by the scalar longitude `longitudeOrigin`.

Examples

Clip Polygon to World Limits

Vector data for geographic objects that encompass a pole must encounter or cross the date line. While the toolbox properly displays such polygons, they can cause problems for functions like the polygon intersection and Boolean operations that work with Cartesian coordinates. When these polygons are treated as Cartesian coordinates, the date line crossing results in a spurious line segment, and the polygon displayed as a patch does not have the interior filled correctly.

```
antarctica = shaperead('landareas', 'UseGeoCoords', true,...
    'Selector', {@(name) strcmp(name,'Antarctica'), 'Name'});
figure
plot(antarctica.Lon, antarctica.Lat)
ylim([-100 -60])
```
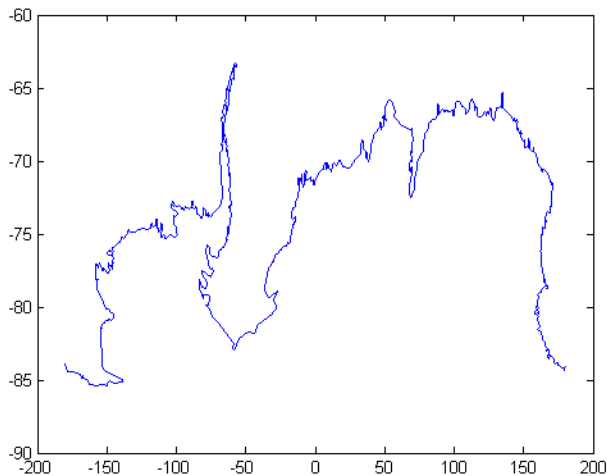
The polygons can be reformatted more appropriately for Cartesian coordinates using the `flatearthpoly` function. The result resembles a map display on a cylindrical projection. The polygon meets the date line, drops down to the pole, sweeps across the longitudes at the pole, and follows the date line up to the other side of the date line crossing.

```
[latf, lonf] = flatearthpoly(antarctica.Lat', antarctica.Lon');
figure; mapshow(lonf, latf, 'DisplayType', 'polygon')
ylim([-100 -60])
xlim([-200 200])
axis square
```
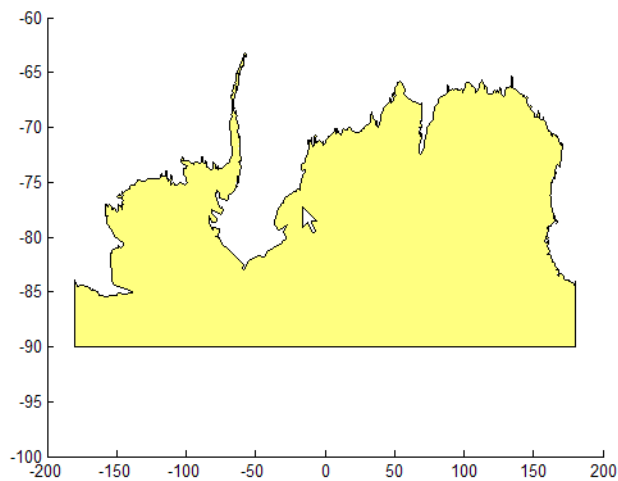

Cut Lines and Polygons at the Antimeridian

Longitude coordinate discontinuities at the Antimeridian can confuse set operations on polygons. To prepare geographic data for use with `polybool` or for patch rendering, cut the polygons at the Antimeridian with the `flatearthpoly` function. The `flatearthpoly` function returns a polygon

with points inserted to follow the Antimeridian up to the pole, traverse the longitudes at the pole, and return to the Antimeridian along the other edge of the Antimeridian.

Create an orthographic view of the Earth and plot the coastlines on it.

```
axesm ortho
setm(gca,'Origin', [60 170]); framem on; gridm on
load coastlines
plotm(coastlat,coastlon)
```


Generate a small circle that encompasses the North Pole and color it yellow.

```
[latc,lonc] = scircle1(75,45,30);
patchm(latc,lonc,'y')
```
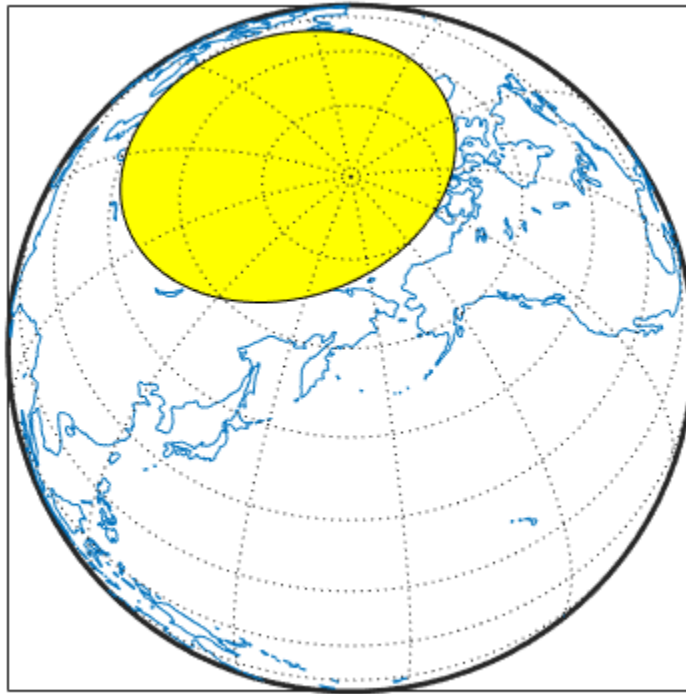
Flatten the small circle using the `flatearthpoly` function.

```
[latf,lonf] = flatearthpoly(latc,lonc);
```

Plot the cut circle that you just generated as a magenta line.

```
plotm(latf,lonf,'m')
```
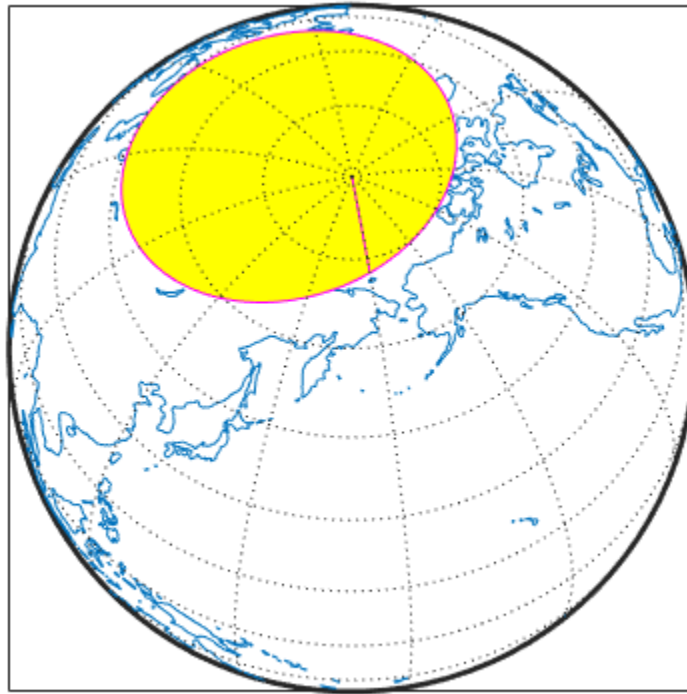
Generate a second small circle that does not include a pole.

```
[latc1, lonc1] = scircle1(20, 170, 30);
```

Flatten the circle and plot it as a red line. Note that the second small circle, which does not cover a pole, is clipped into two pieces along the Antimeridian. The polygon for the first small circle is plotted in plane coordinates to illustrate its flattened shape. The `flatearthpoly` function assumes that the interior of the polygon being flattened is in the hemisphere that contains most of its edge points. Thus a polygon produced by `flatearthpoly` does not cover more than a hemisphere.

```
[latf1,lonf1] = flatearthpoly(latc1,lonc1);
plotm(latf1,lonf1,'r')
```
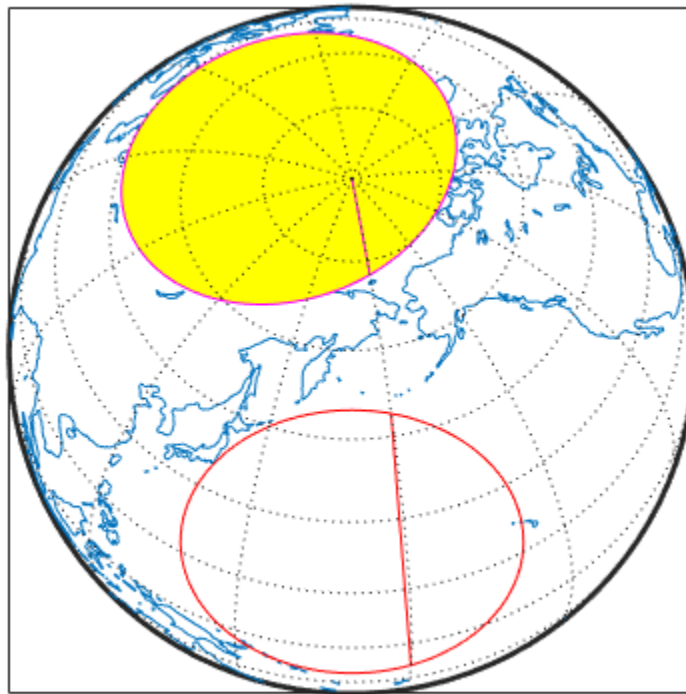
Tips

The polygon defined by `lat` and `lon` must be well-formed:

- The boundaries must not intersect.
- The vertices of outer boundaries must be in a clockwise order and the vertices of inner boundaries must be in a counterclockwise order, such that the interior of the polygon is always to the right of the boundary.

For more information, see "Create and Display Polygons".

See Also
`ispolycw` | `maptrimp` | `poly2cw` | `poly2ccw`

Introduced before R2006a

forward

Package: `map.geodesy`

Convert geodetic latitude to authalic, conformal, isometric, or rectifying latitude

Syntax

```
lat = forward(converter,phi)
lat = forward(converter,phi,angleUnit)
```

Description

`lat = forward(converter,phi)` returns the authalic, conformal, isometric, or rectifying latitude coordinates corresponding to geodetic latitude coordinates `phi`.

`lat = forward(converter,phi,angleUnit)` specifies the units of input `phi`.

Examples

Convert Geodetic Latitude to Authalic Latitude

Specify geodetic latitude coordinates and create an authalic latitude converter. Then, convert the coordinates.

```
phi = [-90 -67.5 -45 -22.5 0 22.5 45 67.5 90];
conv = map.geodesy.AuthalicLatitudeConverter(wgs84Ellipsoid);
beta = forward(conv,phi)
```

beta = *1×9*

```
  -90.0000   -67.4092   -44.8717   -22.4094         0   22.4094   44.8717   67.4092   90.0000
```

Convert Geodetic Latitude to Isometric Latitude

Specify geodetic latitude coordinates and create an isometric latitude converter. Then, convert the coordinates.

```
phi = [-90 -67.5 -45 -22.5 0 22.5 45 67.5 90];
conv = map.geodesy.IsometricLatitudeConverter(wgs84Ellipsoid);
psi = forward(conv,phi)
```

psi = *1×9*

```
     -Inf   -1.6087   -0.8766   -0.4006         0   0.4006   0.8766   1.6087      Inf
```

Convert Geodetic Latitude to Rectifying Latitude Using Radians

Specify geodetic latitude coordinates and convert them to radians. Create a rectifying latitude converter. Then, convert the coordinates by specifying the angle unit as `'radians'`.

```
phi = [-90 -67.5 -45 -22.5 0 22.5 45 67.5 90];
phirad = deg2rad(phi);
conv = map.geodesy.RectifyingLatitudeConverter(wgs84Ellipsoid);
mu = forward(conv,phirad,'radians')
```

mu = *1×9*

```
  -1.5708   -1.1763   -0.7829   -0.3909        0    0.3909    0.7829    1.1763    1.5708
```

Input Arguments

converter — Latitude converter
AuthalicLatitudeConverter, ConformalLatitudeConverter, IsometricLatitudeConverter, or RectifyingLatitudeConverter object

Latitude converter, specified as an AuthalicLatitudeConverter, ConformalLatitudeConverter, IsometricLatitudeConverter, or RectifyingLatitudeConverter object.

phi — Geodetic latitude coordinates
numeric scalar, vector, matrix, or N-D array

Geodetic latitude coordinates, specified as a numeric scalar value, vector, matrix, or N-D array. If angleUnit is not supplied, phi is in degrees. Otherwise, values of phi must be consistent with the units of angleUnit.

Data Types: single | double

angleUnit — Unit of geodetic latitude coordinates
`'degrees'` (default) | `'radians'`

Units of geodetic latitude coordinates, specified as `'degrees'` or `'radians'`.

Output Arguments

lat — Converted latitude coordinates
numeric scalar, vector, matrix, or N-D array

Converted latitude coordinates, returned as a numeric scalar, vector, matrix, or N-D array. lat is the same size as phi.

The interpretation of lat depends on the latitude converter. If the conversion is:

- authalic, lat represents the variable β (beta), and has the same units as phi.
- conformal, lat represents χ (chi), and has the same units as phi.
- isometric, lat represents ψ (psi). lat is a dimensionless number and does not have an angle unit.

- rectifying, `lat` represents μ (mu), and has the same units as `phi`.

See Also

`inverse`

Introduced in R2013a

framem

Toggle and control display of map frame

Syntax

```
framem
framem('on')
framem('off')
framem('reset')
framem(linespec)
framem(PropertyName,PropertyValue,...)
```

Description

`framem` toggles the visibility of the map frame by setting the map axes property `Frame` to `'on'` or `'off'`. The default setting for map axes is `'off'`.

`framem('on')` sets the map axes property `Frame` to `'on'`.

`framem('off')` sets the map axes property `Frame` to `'off'`. When called with the value `'off'`, the map axes property `Frame` is set to `'off'`.

`framem('reset')` resets the entire frame using the current properties. This is essentially a *refresh* option.

`framem(linespec)` sets the map axes `FEdgeColor` property to the color component of any MATLAB `linespec`.

`framem(PropertyName,PropertyValue,...)` sets the appropriate map axes properties to the desired values. These property names and values are described on the `axesm` reference page.

Tips

- You can also create or alter map frame properties using the `axesm` or `setm` functions.

- By default the Clipping property is set to `'off'`. Override this setting with the following code:

```
hgrat = gridm('on');
set(hgrat,'Clipping','on')
```

See Also

axesm | setm

Introduced before R2006a

fromDegrees

Convert angles from degrees

Syntax

```
[A1,...,An] = fromDegrees(toUnits,D1,...,Dn)
```

Description

`[A1,...,An] = fromDegrees(toUnits,D1,...,Dn)` converts the angles specified by `D1,...,Dn` from degrees to the output angle units specified by `toUnits`. The value of `toUnits` can be either `'degrees'` or `'radians'` and may be abbreviated. The inputs `D1,...,Dn` and their corresponding outputs are numeric arrays of various sizes, with `size(An)` matching `size(Dn)`.

See Also

fromRadians | deg2rad | toDegrees | toRadians

Introduced in R2007b

fromRadians

Convert angles from radians

Syntax

```
[A1,...,An] = fromRadians(toUnits,R1,...,Rn)
```

Description

`[A1,...,An] = fromRadians(toUnits,R1,...,Rn)` converts the angles specified by `R1,...,Rn` from radians to the angle units specified by `toUnits`. The value of `toUnits` can be either `'degrees'` or `'radians'` and may be abbreviated. The inputs `R1,...,Rn` and their corresponding outputs are numeric arrays of various sizes, with `size(An)` matching `size(Rn)`.

See Also
`fromDegrees` | `rad2deg` | `toDegrees` | `toRadians`

Introduced in R2007b

gc2sc

Center and radius of great circle

Syntax

```
[lat,lon,radius] = gc2sc(lat0,lon0,az)
[lat,lon,radius] = gc2sc(lat0,lon0,az,angleunits)
mat = gc2sc(...)
```

Description

`[lat,lon,radius] = gc2sc(lat0,lon0,az)` converts a great circle from great circle notation (i.e., lat, lon, azimuth, where (lat, lon) is on the circle) to small circle notation (i.e., lat, lon, radius, where (lat, lon) is the center of the circle and the radius is 90 degrees, which is a definition of a great circle). A great circle has two centers and one is chosen arbitrarily. The other is its antipode. All inputs and outputs are in units of degrees.

`[lat,lon,radius] = gc2sc(lat0,lon0,az,angleunits)` where `angleunits` specifies the units of the inputs and outputs, either `'degrees'` or `'radians'`.

`mat = gc2sc(...)` returns a single output, where `mat = [lat lon radius]`.

Examples

Represent a great circle passing through (25ºS,70ºW) on an azimuth of 45º as a small circle:

```
[lat,lon,radius] = gc2sc(-25,-70,45)

lat =
  -39.8557

lon =
   42.9098

radius =
    90
```

A great circle always bisects the sphere. As a demonstration of this statement, consider the Equator, which passes through any point with a latitude of 0º and proceeds on an azimuth of 90º or 270º. Represent the Equator as a small circle:

```
[lat,lon,radius] = gc2sc(0,-70,270)

lat =

    90

lon =

     0
```

```
radius =

    90
```

Not surprisingly, the small circle is centered on the North Pole. As always at the poles, the longitude is arbitrary because of the convergence of the meridians.

Note that the center coordinates returned by this function always lead to one of two possibilities. Since the great circle bisects the sphere, the antipode of the returned point is also a center with a radius of 90º. In the above example, the South Pole would also be a suitable center for the Equator in a small circle.

More About

Great and Small Circles

A *small circle* is the intersection of a plane with the surface of a sphere. A *great circle* is a small circle with a radius of 90º.

See Also
antipode | crossfix | gcxgc | gcxsc | rhxrh

Introduced before R2006a

gcm

Current map projection structure

Syntax

```
mstruct = gcm
mstruct = gcm(hndl)
```

Description

`mstruct = gcm` returns the map axes *map structure*, which contains the settings for all the current map axes properties.

`mstruct = gcm(hndl)` specifies the map axes by axes handle.

Examples

Establish a map axes with default values, then look at the structure:

```
axesm mercator
mstruct = gcm

mstruct =
    mapprojection: 'mercator'
             zone: []
       angleunits: 'degrees'
           aspect: 'normal'
     falsenorthing: 0
      falseeasting: 0
       fixedorient: []
             geoid: [1 0]
       maplatlimit: [-86 86]
       maplonlimit: [-180 180]
       mapparallels: 0
        nparallels: 1
            origin: [0 0 0]
        scalefactor: 1
            trimlat: [-86 86]
            trimlon: [-180 180]
              frame: 'off'
              ffill: 100
         fedgecolor: [0 0 0]
         ffacecolor: 'none'
          flatlimit: [-86 86]
         flinewidth: 2
          flonlimit: [-180 180]
               grid: 'off'
          galtitude: Inf
             gcolor: [0 0 0]
         glinestyle: ':'
         glinewidth: 0.5000
      mlineexception: []
```

```
       mlinefill: 100
      mlinelimit: []
   mlinelocation: 30
    mlinevisible: 'on'
  plineexception: []
       plinefill: 100
      plinelimit: []
   plinelocation: 15
    plinevisible: 'on'
       fontangle: 'normal'
       fontcolor: [0 0 0]
        fontname: 'Helvetica'
        fontsize: 10
       fontunits: 'points'
      fontweight: 'normal'
     labelformat: 'compass'
   labelrotation: 'off'
      labelunits: 'degrees'
   meridianlabel: 'off'
   mlabellocation: 30
   mlabelparallel: 86
     mlabelround: 0
   parallellabel: 'off'
   plabellocation: 15
   plabelmeridian: -180
     plabelround: 0
```

Tips

You create map structure properties with the `axesm` function. You can query them with the `getm` function and modify them with the `setm` function.

See Also

axesm | getm | setm

Introduced before R2006a

gcpmap

Current mouse point from map axes

Syntax

```
pt = gcpmap
pt = gcpmap(hndl)
```

Description

`pt = gcpmap` returns the current point (the location of last button click) of the current map axes in the form `[latitude longitude z-altitude]`.

`pt = gcpmap(hndl)` specifies the map axes in question by its handle.

Examples

Set up a map axes with a graticule and display a world map:

```
axesm robinson
gridm on
geoshow('landareas.shp')
```

Click somewhere near Boston, Massachusetts to obtain a current point:

```
pt = gcpmap

pt =
       44.171       -69.967              2
       44.171       -69.967              0
whos

  Name       Size                    Bytes  Class          Attributes
  pt         2x3                        48  double array
```
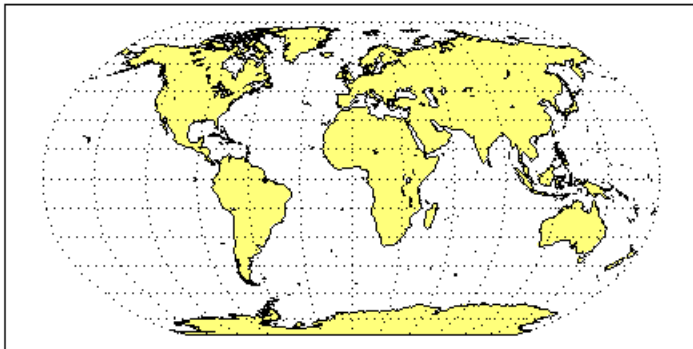
Tips

gcpmap works much like the standard MATLAB function get(gca,'CurrentPoint'), except that the returned matrix is in [lat lon z], not [x y z].

You must use view(2) and an ordinary projection (not the Globe projection) when working with the gcpmap function.

The CurrentPoint property is updated whenever a button-click event occurs in a MATLAB figure window. The pointer does not have to be within the axes, or even the figure window. Coordinates with respect to the requested axes are returned regardless of the pointer location. Likewise, gcpmap will return values that may look reasonable whether the current point is within the graticule bounds or not, and thus must be used with care.

See Also
inputm

Introduced before R2006a

gcwaypts

Equally spaced waypoints along great circle

Syntax

```
[lat,lon] = gcwaypts(lat1,lon1,lat2,lon2)
[lat,lon] = gcwaypts(lat1,lon1,lat2,lon2,nlegs)
pts = gcwaypts(lat1,lon1,lat2,lon2...)
```

Description

`[lat,lon] = gcwaypts(lat1,lon1,lat2,lon2)` returns the coordinates of equally spaced points along a great circle path connecting two endpoints, (`lat1,lon1`) and (`lat2,lon2`).

`[lat,lon] = gcwaypts(lat1,lon1,lat2,lon2,nlegs)` specifies the number of equal-length track legs to calculate. `nlegs+1` output points are returned, since a final endpoint is required. The default number of legs is 10.

`pts = gcwaypts(lat1,lon1,lat2,lon2...)` packs the outputs, which are otherwise two-column vectors, into a two-column matrix of the form `[latitude longitude]`. This format for successive waypoints along a navigational track is called *navigational track format* in this guide. See the `navigational track format` reference page in this section for more information.

Background

This is a navigational function. It assumes that all latitudes and longitudes are in degrees.

In navigational practice, great circle paths are often approximated by rhumb line segments. This is done to come reasonably close to the shortest distance between points without requiring course changes too frequently. The `gcwaypts` function provides an easy means of finding waypoints along a great circle path that can serve as endpoints for rhumb line segments (track legs).

Examples

Determine Equally Spaced Points Along a Great Circle Path

Imagine you own a sailing yacht and are planning a voyage from North Point, Barbados (13.33° N,59.62°W), to Brest, France (48.36°N,4.49°W). Divide the track into three equal-length segments.

```
figure('color','w');
ha = axesm('mapproj','mercator',...
    'maplatlim',[10 55],'maplonlim',[-80 10],...
    'MLineLocation',15,'PLineLocation',15);
axis off, gridm on, framem on;
% Load coastline data and plot it in the figure.
load coastlines;
hg = geoshow(coastlat,coastlon,'displaytype','line','color','b');
% Define point locations for Barbados and Brest
barbados = [13.33 -59.62];
```

```
brest = [48.36 -4.49];
% Calculate the waypoints along the path.
[l,g] = gcwaypts(barbados(1),barbados(2),brest(1),brest(2),3);
geoshow(l,g,'displaytype','line','color','r',...
    'markeredgecolor','r','markerfacecolor','r','marker','o');
geoshow(barbados(1),barbados(2),'DisplayType','point',...
    'markeredgecolor','k','markerfacecolor','k','marker','o')
geoshow(brest(1),brest(2),'DisplayType','point',...
    'markeredgecolor','k','markerfacecolor','k','marker','o')
```
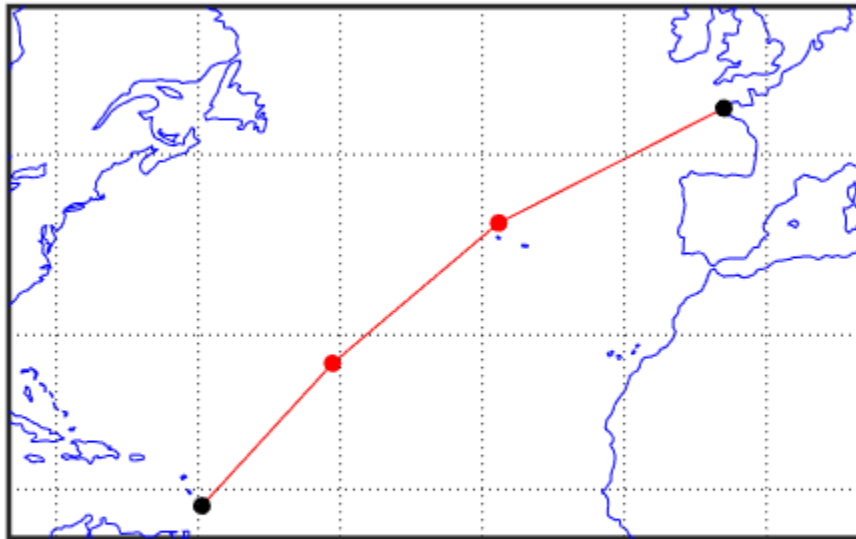

See Also

dreckon | legs | navfix | track

Introduced before R2006a

gcxgc

Intersection points for pairs of great circles

Syntax

```
[lat,lon] = gcxgc(lat1,lon1,az1,lat2,lon2,az2)
[lat,lon] = gcxgc(lat1,lon1,az1,lat2,lon2,az2,units)
latlon = gcxgc( ___ )
```

Description

`[lat,lon] = gcxgc(lat1,lon1,az1,lat2,lon2,az2)` returns in `lat` and `lon` the locations where pairs of great circles intersect. The great circles are defined using *great circle notation*, which consists of a point on the great circle and the azimuth at that point along which the great circle proceeds. For example, the first great circle in a pair would pass through the point (`lat1,lon1`) with an azimuth of `az1` (in angular units).

For any pair of great circles, there are two possible intersection conditions: the circles are identical or they intersect exactly twice on the sphere.

`[lat,lon] = gcxgc(lat1,lon1,az1,lat2,lon2,az2,units)` specifies the angular units used for all inputs, where `units` is any valid angular unit.

`latlon = gcxgc( ___ )` returns a single output consisting of the concatenated latitude and longitude coordinates of the great circle intersection points.

Examples

Find Intersection Points of Two Great Circles

Given a great circle passing through (10ºN,13ºE) and proceeding on an azimuth of 10º, where does it intersect with a great circle passing through (0º, 20ºE), on an azimuth of -23º (that is, 337º)?

```
[newlat,newlon] = gcxgc(10,13,10,0,20,-23)

newlat =
   14.3105  -14.3105

newlon =
   13.7838  -166.2162
```

Note that the two intersection points are always antipodes of each other. As a simple example, consider the intersection points of two meridians, which are just great circles with azimuths of 0º or 180º:

```
[newlat,newlon] = gcxgc(10,13,0,0,20,180)

newlat =

   -90     90
```

```
newlon =

    0   180
```

The two meridians intersect at the North and South Poles, which is exactly correct.

Input Arguments

`lat1, lon1` — Coordinate of point on first great circle
numeric scalar | *n*-element numeric vector

Latitude or longitude coordinate of a point on the first great circle in each pair, specified as one of these values.

- A numeric scalar to find the intersection of a single pair of great circles.
- A *n*-element numeric vector to find the intersection of *n* pairs of great circles.

`lat1` and `lon1` must have the same length.

Example: `10`

Example: `[-10 20 90 -45]`

`az1` — Azimuth of first great circle
positive numeric scalar | *n*-element vector of positive numbers

Azimuth of the first great circle of each pair, in angular units, specified as one of these values.

- A positive numeric scalar to find the intersection of a single pair of great circles.
- A *n*-element vector of positive numbers to find the intersection of *n* pairs of great circles. The length of `az1` matches the length of `lat1` and `lon1`.

Example: `20`

Example: `[20 10 45 45]`

`lat2, lon2` — Coordinate of point on second great circle
numeric scalar | numeric vector

Latitude or longitude coordinate of a point on the second great circle in each pair, specified as one of these values.

- A numeric scalar to find the intersection of a single pair of great circles.
- A *n*-element numeric vector to find the intersection of *n* pairs of great circles.

`lat2` and `lon2` must have the same length as `lat1` and `lon1`.

Example: `3`

Example: `[3 30 85 -45]`

`az2` — Azimuth of second great circle
positive numeric scalar | *n*-element vector of positive numbers

Azimuth of the second great circle of each pair, in angular units, specified as one of these values.

- A positive numeric scalar to find the intersection of a single pair of great circles.
- A *n*-element vector of positive numbers to find the intersection of *n* pairs of great circles. The length of `az2` matches the length of `lat2` and `lon2`.

Example: 15

Example: [15 15 45 50]

units — Angular units
'degrees' (default) | 'radians'

Angular units, specified as 'degrees' or 'radians'.

Output Arguments

lat, lon — Coordinates of great circle intersections
2-element vector | *n*-by-2 matrix

Coordinates of great circle intersections, returned as one of the following.

- 2-element vector when you find the intersection of a single pair of great circles.
- *n*-by-2 matrix when you find the intersection of *n* pairs of great circles.

If a pair of great circles are identical, then `gcxgc` displays a warning and returns NaNs for the latitude and longitude coordinates of the intersection points.

latlon — Concatenated coordinates of great circle intersections
4-element vector | *n*-by-4 matrix

Concatenated coordinates of great circle intersections, returned as one of the following. This output is identical to [`lat lon`].

- 4-element vector when you find the intersection of a single pair of great circles.
- *n*-by-4 matrix when you find the intersection of *n* pairs of great circles.

If a pair of great circles are identical, then `gcxgc` displays a warning and returns NaNs for the latitude and longitude coordinates of the intersection points.

See Also
antipode | gc2sc | scxsc | gcxsc | rhxrh | crossfix | polyxpoly

Introduced before R2006a

gcxsc

Intersection points for great and small circle pairs

Syntax

```
[newlat,newlon] = gcxsc(gclat,gclon,gcaz,sclat,sclon,scrange)
[newlat,newlon] = gcxsc(..., units)
```

Description

`[newlat,newlon] = gcxsc(gclat,gclon,gcaz,sclat,sclon,scrange)` returns the points of intersection of a great circle in *great circle notation* followed by a small circle in *small circle notation*. For multiple pairings, the inputs must be column vectors. The results are two-column matrices with the coordinates of the intersection points. If the circles do not intersect, or are identical, two `NaN`s are returned and a warning is displayed. If the two circles are tangent, the single intersection point is repeated twice.

`[newlat,newlon] = gcxsc(..., units)` where `units` specifies the standard angle unit. The default value is `'degrees'`.

For a pairing of a great circle with a small circle, there are four possible intersection conditions: the circles are identical (possible because great circles are a subset of small circles), they do not intersect, they are tangent to each other (the small circle interior to the great circle) and hence they intersect once, or they intersect twice.

Great circle notation consists of a point on the great circle and the azimuth at that point along which the great circle proceeds.

Small circle notation consists of a center point and a radius in units of angular arc length.

Examples

Given a great circle passing through (43ºN,0º) and proceeding on an azimuth of 10º, where does it intersect with a small circle centered at (47ºN,3ºE) with an arc length radius of 12º?

```
[newlat,newlon] = gcxsc(43,0,10,47,3,12)

newlat =
   35.5068   58.9143

newlon =
   -1.6159    5.4039
```

See Also
gc2sc | gcxgc | scxsc | rhxrh | crossfix | polyxpoly

Introduced before R2006a

geocentric2geodeticLat

Convert geocentric to geodetic latitude

Note geocentric2geodeticLat will be removed in a future release. Use geodeticLatitudeFromGeocentric instead.

Syntax

phiI = geocentric2geodeticlat(ecc, phi_g)

Description

phiI = geocentric2geodeticlat(ecc, phi_g) converts an array of geocentric latitude in radians, phi_g, to geodetic latitude in radians, phiI, on a reference ellipsoid with first eccentricity ecc.

See Also

geodeticLatitudeFromGeocentric

Introduced in R2006b

geocentricLatitude

Convert geodetic to geocentric latitude

Syntax

```
psi = geocentricLatitude(phi,F)
psi = geocentricLatitude(phi,F,angleUnit)
```

Description

`psi = geocentricLatitude(phi,F)` returns the geocentric latitude corresponding to geodetic latitude `phi` on an ellipsoid with flattening `F`.

`psi = geocentricLatitude(phi,F,angleUnit)` specifies the units of input `phi` and output `psi`.

Examples

Convert Geodetic Latitude to Geocentric Latitude

Create a reference ellipsoid and then convert the geodetic latitude to geocentric latitude. The reference ellipsoid contains a flattening factor.

```
s = wgs84Ellipsoid;

geocentricLatitude(45, s.Flattening)

ans =

   44.8076
```

Convert Geodetic Latitude Expressed in Radians to Geocentric Latitude

Create a reference ellipsoid and then convert a geodetic latitude expressed in radians to geocentric latitude. The reference ellipsoid contains a flattening factor.

```
s = wgs84Ellipsoid;

geocentricLatitude(pi/3, s.Flattening, 'radians')

ans =

   1.0443
```

Input Arguments

phi — Geodetic latitude of one or more points
scalar value, vector, matrix, or N-D array

Geodetic latitude of one or more points, specified as a scalar value, vector, matrix, or N-D array. Values must be in units that match the input argument `angleUnit`, if supplied, and in degrees, otherwise.

Data Types: `single` | `double`

F — Flattening of reference spheroid
scalar

Flattening of reference spheroid, specified as a scalar value.

Data Types: `single` | `double`

angleUnit — Unit of measurement for angle
`'degrees'` (default) | `'radians'`

Unit of measurement for angle, specified as either `'degrees'` or `'radians'`.

Data Types: `char`

Output Arguments

psi — Geocentric latitudes of one or more points
scalar value, vector, matrix, or N-D array

Geocentric latitudes of one or more points, returned as a scalar value, vector, matrix, or N-D array. Values are in units that match the input argument `angleUnit`, if supplied, and in degrees, otherwise.

See Also

Functions
parametricLatitude | geodeticLatitudeFromGeocentric

Objects
AuthalicLatitudeConverter | ConformalLatitudeConverter | IsometricLatitudeConverter | RectifyingLatitudeConverter

Introduced in R2013a

geocontourxy

Contour grid in local system with latitude-longitude results

Syntax

```
[contourLines,contourPolygons] = geocontourxy(X,Y,Z,lat0,lon0,h0)
[ ___ ] = geocontourxy( ___ ,Name,Value)
```

Description

`[contourLines,contourPolygons] = geocontourxy(X,Y,Z,lat0,lon0,h0)` returns line and polygon geoshapes containing contour lines and contour fill polygons, respectively. This function is non-graphical. You can plot the return values using `geoshow`, if desired.

`[ ___ ] = geocontourxy( ___ ,Name,Value)` specifies name-value pairs that control aspects of the operation. Parameter names can be abbreviated and are case-insensitive.

Examples

Calculate Contour Lines and Polygons for Area Near Hawaii

Define a set of *X* and *Y* coordinates and create contour lines and contour polygons.

```
X = -150000:10000:150000;
Y =  0:10000:300000;
[xmesh, ymesh] = meshgrid(X/50000, (Y - 150000)/50000);
Z = 8 + peaks(xmesh, ymesh);
lat0 = dm2degrees([  21 18]);
lon0 = dm2degrees([-157 49]);
h0 = 300;
levels = 0:2:18;

[contourLines, contourPolygons] = geocontourxy(X,Y,Z,lat0,lon0,h0, ...
        'LevelList',levels,'XYRotation',120)

contourLines =
 8x1 geoshape vector with properties:

 Collection properties:
       Geometry: 'line'
       Metadata: [1x1 struct]
 Vertex properties:
  (8 features concatenated with 7 delimiters)
       Latitude: [21.1171 21.0296 20.9819 20.9814 20.9880 21.0702 21.1122 ... ]
      Longitude: [-158.5364 -158.5053 -158.4015 -158.3987 -158.3903 ... ]
         Height: [769.9110 771.5545 686.9961 684.5002 672.2181 584.6248 ... ]
 Feature properties:
    ContourLevel: [2 4 6 8 10 12 14 16]


contourPolygons =
 9x1 geoshape vector with properties:
```

```
       Collection properties:
                   Geometry: 'polygon'
                   Metadata: [1x1 struct]
      Vertex properties:
        (9 features concatenated with 8 delimiters)
                   Latitude: [21.1171 21.1630 21.1671 21.1431 21.1122 21.0702 20.9880 ... ]
                  Longitude: [-158.5364 -158.5114 -158.4450 -158.3876 -158.3542 ... ]
                     Height: [769.9110 725.7670 650.4579 599.1402 578.2085 584.6248 ... ]
      Feature properties:
           LowerContourLevel: [0 2 4 6 8 10 12 14 16]
           UpperContourLevel: [2 4 6 8 10 12 14 16 18]
```

Display Hawaii on a map, add a marker, and then display the polygons returned by `geocontourxy` on the map.

```
figure
usamap([18.5 22.5],[-161 -154])
hawaii = shaperead('usastatehi.shp', 'UseGeoCoords', true,...
        'Selector',{@(name) strcmpi(name,'Hawaii'), 'Name'});
geoshow(hawaii)
geoshow(lat0,lon0,'DisplayType','point','Marker','o',...
        'MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',10)

cmap = parula(1 + length(levels));
for k = 1:length(contourPolygons)
      lat = contourPolygons(k).Latitude;
      lon = contourPolygons(k).Longitude;
      geoshow(lat,lon,'Display','polygon', ...
            'FaceColor',cmap(k,:),'FaceAlpha',0.5,'EdgeColor','none')
end
geoshow(contourLines.Latitude,contourLines.Longitude,'Color','black')
```
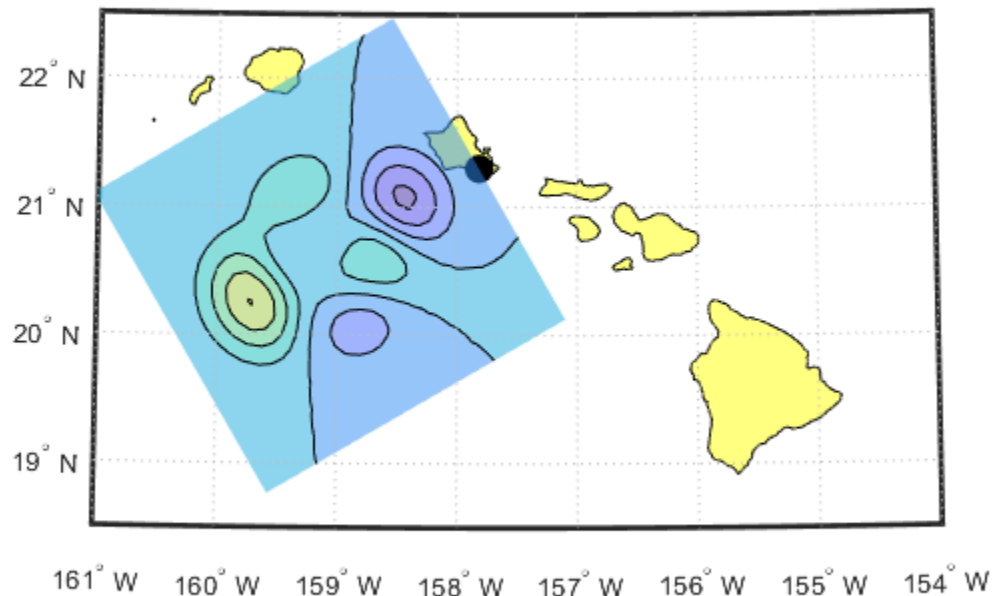
Input Arguments

X — *X*-component of a mesh that locates each element of *Z* in a local *x-y* plane
vector or matrix

X-component of a mesh that locates each element of *Z* in a local *x-y* plane, specified as a vector or matrix. geocontourxy assumes that units are meters unless you provide a Spheroid input, in which case the units of your input must match theLengthUnit property of the Spheroid object.

Data Types: single | double

Y — *Y*-component of a mesh that locates each element of *Z* in a local *x-y* plane
vector or matrix

Y-component of a mesh that locates each element of *Z* in a local *x-y* plane. specified as a vector or matrix. geocontourxy assumes that units are meters unless you provide a Spheroid input, in which case the units of your input must match theLengthUnit property of the Spheroid object.

Data Types: single | double

Z — Data to be contoured
2-D array

Data to be contoured, specified as a 2-D array.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64

`lat0` — Geodetic latitude of local origin (reference) point
scalar value in units of degrees

Geodetic latitude of local origin (reference) point, specified as a scalar value in units of degrees.

Data Types: `single` | `double`

`lon0` — Geodetic longitude of local origin (reference) point
scalar value in units of degrees

Geodetic longitude of local origin (reference) point, specified as a scalar value in units of degrees.

Data Types: `single` | `double`

`h0` — Ellipsoidal height of local origin (reference) point
scalar value

Ellipsoidal height of local origin (reference) point, specified as a scalar value. `geocontourxy` assumes that units are meters unless you provide a `Spheroid` input, in which case the units of your input must match the unit specified in the `LengthUnit` property of the `Spheroid` object.

Data Types: `single` | `double`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `[contourLines, contourPolygons] = geocontourxy(X,Y,Z,lat0,lon0,h0,'LevelList',levels,'XYRotation',120)`

`LevelList` — Contour levels
vector of *Z*-values

Contour levels, specified as a vector of Z-values. By default, the `geocontourxy` function chooses levels that span the range of values.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`XYRotation` — Rotation angle of the local *x-y* system
0 (default) | scalar value in units of degrees

Rotation angle of the local *x-y* system, measured counterclockwise from the *xEast-yNorth* system, specified as a scalar value in units of degrees.

Data Types: `single` | `double`

Spheroid — Reference spheroid
WGS84 reference ellipsoid (default) | `referenceEllipsoid` | `oblateSpheroid` | `referenceSphere`

Reference spheroid, specified as a `referenceEllipsoid`, `oblateSpheroid`, or `referenceSphere` object. Use the constructor for one of these three classes, or the `wgs84Ellipsoid` function, to construct a Mapping Toolbox spheroid object. (You cannot directly pass in to `geocontourxy` the name of your spheroid. Instead, pass that name to `referenceEllipsoid` or `referenceSphere` and use the resulting object.) By default, `geocontourxy` uses the WGS84 reference ellipsoid with units of meters.

Output Arguments

contourLines — Contour lines
line geoshape

Contour lines, returned as a line `geoshape` with one element per contour level. `Latitude` and `Longitude` properties contain contour line vertices in degrees. The contour level value of the *k*-th element is stored in the `ContourLevel` feature property of `contourLines(k)`. A third vertex property, `Height`, contains the ellipsoidal height of each vertex. In combination with `Latitude` and `Longitude`, it completes the definition of the 3-D location of the contour line in the plane that contains the local origin and is parallel to the tangent plane at the origin latitude and longitude.

contourPolygons — Contour polygons
polygon geoshape

Contour polygons, returned as a polygon geoshape with one element (contour fill polygon) per contour interval. `Latitude` and `Longitude` properties contain the vertices of the contour fill polygons, specified in degrees. The `LowerContourLevel` and `UpperContourLevel` properties of `contourPolygons(k)` store the limits of the *k*-th contour interval. As in the case of lines, a third vertex property, `Height`, is included.

See Also
contourm | contourfm | geodetic2enu | enu2geodetic

Topics
"Create and Display Polygons"

Introduced in R2016a

geocrop

Crop geographic raster

Syntax

```
[B,RB] = geocrop(A,RA,latlim,lonlim)
```

Description

`[B,RB] = geocrop(A,RA,latlim,lonlim)` crops the raster specified by A and raster reference RA and returns the cropped raster B and raster reference RB. The returned raster is cropped to geographic limits in degrees close to those specified by `latlim` and `lonlim`.

Examples

Crop Geographic Raster

Crop a geographic raster and display the cropped raster on a map.

First, load elevation raster data and a geographic cells reference object for the Korean peninsula. Then, crop the raster to the limits specified by `latlim` and `lonlim`.

```
load korea5c
latlim = [34.25 38.72];
lonlim = [125.85 129.92];
[B,RB] = geocrop(korea5c,korea5cR,latlim,lonlim);
```

Display the cropped raster as a surface on a map. Apply a colormap appropriate for elevation data using the `demcmap` function.

```
worldmap(latlim,lonlim)
geoshow(B,RB,'DisplayType','surface')
demcmap(B)
```
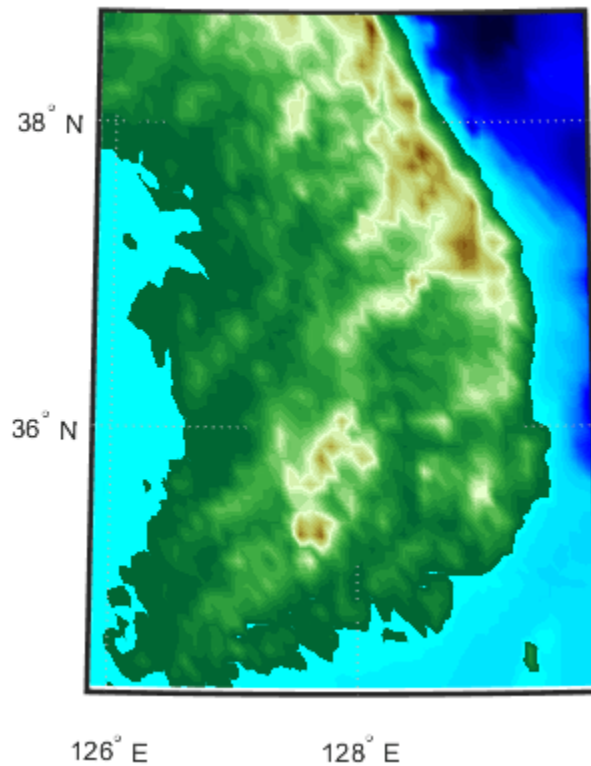
Shift Longitude Limits

Shift the longitude limits of a raster with limits that span 360 degrees using the `geocrop` function. Then, display the shifted data as a surface on a map.

First, load elevation raster data for the world and a geographic cells reference object. Then, shift the longitude limits of the raster from [0, 360] to [-180, 180].

```
load topo60c
latlim = topo60cR.LatitudeLimits;
[B,RB] = geocrop(topo60c,topo60cR,latlim,[-180 180]);
```

Compare the rasters by querying their `LongitudeLimits` properties.

```
topo60cR.LongitudeLimits
```

ans = *1×2*

```
     0   360
```

```
RB.LongitudeLimits
```

ans = *1×2*

```
   -180    180
```

Display the shifted data as a surface on a map. Move the meridian labels to the bottom of the map using the `mlabel` function. Specify meridian labels to display using the `MLabelLocation` property. Then, apply a colormap appropriate for topographic data using the `demcmap` function. Note that the shifted longitude limits appear at the edges of the map.

```
worldmap(RB.LatitudeLimits,RB.LongitudeLimits)
geoshow(B,RB,'DisplayType','surface')
mlabel('south')
setm(gca,'MLabelLocation',-180:90:180)
demcmap(B)
```
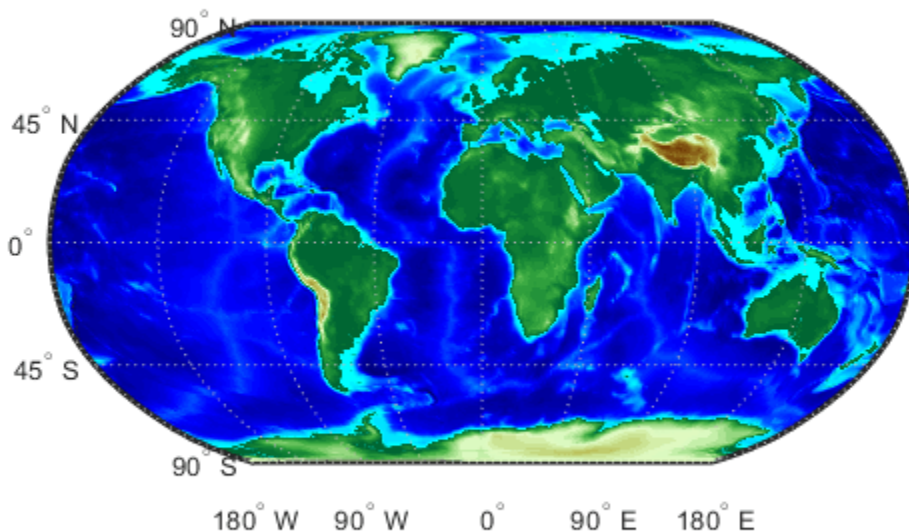

Input Arguments

A — Geographic raster
array

Geographic raster, specified as an *M*-by-*N* or *M*-by-*N*-by-*P* numeric or logical array.

RA — Raster reference
GeographicCellsReference object | GeographicPostingsReference object

Raster reference for A, specified as a `GeographicCellsReference` object or `GeographicPostingsReference` object.

`latlim` — Latitude limits
two-element vector

Latitude limits, specified as a two-element numeric vector of the form `[nlat slat]`, where `nlat` is the northernmost limit in degrees and `slat` is the southernmost limit in degrees.

`lonlim` — Longitude limits
two-element vector

Longitude limits, specified as a two-element numeric vector of the form `[wlon elon]`, where `wlon` is the westernmost limit in degrees and `elon` is the easternmost limit in degrees.

Output Arguments

B — Cropped geographic raster
array

Cropped geographic raster, returned as a numeric or logical array. The data type and size of B matches the data type and size of A.

If the limits specified by `latlim` and `lonlim` do not intersect the raster specified by A and RA, then B is empty.

RB — Raster reference
GeographicCellsReference object | GeographicPostingsReference object

Raster reference for B, returned as a `GeographicCellsReference` object or `GeographicPostingsReference` object. The object type of RB matches the object type of RA.

The exact latitude and longitude limits of RB do not match the limits specified by `latlim` and `lonlim`, unless they coincide with a cell boundary or posting location. Otherwise, the limits of RB are slightly larger than `latlim` and `lonlim`.

If the limits specified by `latlim` and `lonlim` do not intersect the raster specified by A and RA, then RB is empty.

See Also
mapcrop | georesize

Introduced in R2020a

geocrs

Geographic coordinate reference system object

Description

A geographic coordinate reference system (CRS) provides information that assigns latitude, longitude, and height coordinates to physical locations. Geographic CRSs consist of a datum, a prime meridian, and an angular unit of measurement.

Projected CRSs consist of a geographic CRS and several parameters that are used to transform coordinates to and from the geographic CRS. For more information about projected CRSs, see `projcrs`.

Creation

There are several ways to create geographic CRS objects, including:

- Import raster data using functions such as `readgeoraster` or `wmsread`, and then query the `GeographicCRS` property of the returned raster reference object.
- Get information about a shapefile using the `shapeinfo` function, and then query the `CoordinateReferenceSystem` field of the returned structure.
- Access the geographic CRS of a projected CRS by querying the `GeographicCRS` property of a `projcrs` object.
- Use the `geocrs` function (described here).

Syntax

```
g = geocrs(code)
g = geocrs(code,'Authority',authority)
g = geocrs(wkt)
```

Description

`g = geocrs(code)` creates a geographic CRS object using the EPSG code specified by `code`.

`g = geocrs(code,'Authority',authority)` creates a geographic CRS object using the specified `code` and `authority`.

`g = geocrs(wkt)` creates a geographic CRS object using a specified well-known text (WKT) string representation.

Input Arguments

code — Geographic CRS code
positive integer | string scalar | character vector

Geographic CRS code, specified as a positive integer, string scalar, or character vector. By default, the `geocrs` function assumes the `code` argument refers to an EPSG code. To specify other types of codes, use the `'Authority'` name-value pair.

If referring to an EPSG or ESRI code, specify this argument as a positive integer. If referring to an IGNF code, specify this argument as a string scalar or character vector.

For information on valid EPSG codes, see the EPSG home page.

authority — Authority
`'EPSG'` (default) | `'ESRI'` | `'IGNF'`

Authority, specified as `'EPSG'`, `'ESRI'`, or `'IGNF'`. This argument specifies which authority the `geocrs` function uses to determine the properties of the created geographic CRS object. If you do not specify an authority, then the `geocrs` function uses `'EPSG'`.

wkt — Well-known text
string scalar | character vector

Well-known text (WKT), specified as a string scalar or character vector. You can use well-known text in either the WKT 1 or WKT 2 standard.

Properties

Name — CRS name
string scalar

This property is read-only.

CRS name, returned as a string scalar.

Data Types: `string`

Datum — Datum name
string scalar

This property is read-only.

Datum name, returned as a string scalar.

Data Types: `string`

Spheroid — Reference spheroid
`referenceEllipsoid` object | `referenceSphere` object | `oblateSpheroid` object

This property is read-only.

Reference spheroid used by the datum, returned as a `referenceEllipsoid` object, `referenceSphere` object, or `oblateSpheroid` object.

Data Types: `string`

PrimeMeridian — Longitude origin offset from Greenwich
`double`

This property is read-only.

Longitude origin offset from Greenwich, returned as a `double`. The units of the `PrimeMeridian` property match the value of the `AngleUnit` property.

Data Types: `double`

AngleUnit — Angle unit
string scalar

This property is read-only.

Angle unit, returned as a string scalar. The typical values are `"degree"` and `"radian"`.

Data Types: `string`

Object Functions

wktstring   Well-known text string
isequal   Compare two projcrs or geocrs objects for equivalence

Examples

Get Geographic CRS from EPSG Code

Create a geographic CRS object by specifying an EPSG code.

```
g = geocrs(6668)

g =
  geocrs with properties:

            Name: "JGD2011"
           Datum: "Japanese Geodetic Datum 2011"
        Spheroid: [1x1 referenceEllipsoid]
   PrimeMeridian: 0
       AngleUnit: "degree"
```

Get Geographic CRS from ESRI Code

Create a geographic CRS object from an ESRI code by using the `'Authority'` name-value pair.

```
g = geocrs(37220,'Authority','ESRI')

g =
  geocrs with properties:

            Name: "GCS_Guam_1963"
           Datum: "D_Guam_1963"
        Spheroid: [1x1 referenceEllipsoid]
   PrimeMeridian: 0
       AngleUnit: "degree"
```

Get Geographic CRS from IGNF Code

Create a geographic CRS object from an IGNF code by using the `'Authority'` name-value pair. Specify the code using a string or character vector.

```
g = geocrs('RGFG95G','Authority','IGNF')

g =
  geocrs with properties:

              Name: "RGFG95 geographiques (dms)"
             Datum: "Reseau Geodesique Francais Guyane 1995"
          Spheroid: [1x1 referenceEllipsoid]
     PrimeMeridian: 0
         AngleUnit: "degree"
```

Get Geographic CRS from Projection File

Import a WKT projection file as a character vector using the `fileread` function. Then, create a geographic CRS object by specifying the vector.

```
wkt = fileread('landareas.prj');
g = geocrs(wkt)

g =
  geocrs with properties:

              Name: "WGS 84"
             Datum: "World Geodetic System 1984"
          Spheroid: [1x1 referenceEllipsoid]
     PrimeMeridian: 0
         AngleUnit: "degree"
```

Get Geographic CRS from Imported Raster Data

Import raster data as an array and a geographic reference object using the `readgeoraster` function. Then, get the geographic CRS by querying the `GeographicCRS` property of the reference object.

```
[Z,R] = readgeoraster('n39_w106_3arc_v2.dt1');
R.GeographicCRS

ans =
  geocrs with properties:

              Name: "WGS 84"
             Datum: "World Geodetic System 1984"
          Spheroid: [1x1 referenceEllipsoid]
     PrimeMeridian: 0
         AngleUnit: "degree"
```

Alternatively, return information about the same file as a `RasterInfo` object using the `georasterinfo` function. Then, get the geographic CRS by querying the `CoordinateReferenceSystem` property of the object.

```
info = georasterinfo('n39_w106_3arc_v2.dt1');
info.CoordinateReferenceSystem

ans =
  geocrs with properties:

            Name: "WGS 84"
           Datum: "World Geodetic System 1984"
        Spheroid: [1x1 referenceEllipsoid]
    PrimeMeridian: 0
       AngleUnit: "degree"
```

Find Reference Ellipsoid for Geographic CRS

Find the reference ellipsoid for a geographic CRS by creating a `geocrs` object and accessing its `Spheroid` property.

```
g = geocrs(4957);
g.Spheroid

ans =
referenceEllipsoid with defining properties:

               Code: 7019
               Name: 'GRS 1980'
          LengthUnit: 'meter'
       SemimajorAxis: 6378137
       SemiminorAxis: 6356752.31414036
    InverseFlattening: 298.257222101
        Eccentricity: 0.0818191910428158

  and additional properties:

    Flattening
    ThirdFlattening
    MeanRadius
    SurfaceArea
    Volume
```

See Also

Functions
`fileread`

Objects
`projcrs` | `referenceEllipsoid`

Topics
"Reference Spheroids"

External Websites
EPSG home page

Introduced in R2020b

geodetic2aer

Transform geodetic coordinates to local spherical

Syntax

```
[az,elev,slantRange] = geodetic2aer(lat,lon,h,lat0,lon0,h0,spheroid)
[az,elev,slantRange] = geodetic2aer( ___ ,angleUnit)
```

Description

`[az,elev,slantRange] = geodetic2aer(lat,lon,h,lat0,lon0,h0,spheroid)` transforms the geodetic coordinates specified by `lat`, `lon`, and `h` to the local azimuth-elevation-range (AER) spherical coordinates specified by `az`, `elev`, and `slantRange`. Specify the origin of the local AER system with the geodetic coordinates `lat0`, `lon0`, and `h0`. Each coordinate input argument must match the others in size or be scalar. Specify `spheroid` as the reference spheroid for the geodetic coordinates.

`[az,elev,slantRange] = geodetic2aer( ___ ,angleUnit)` specifies the units for latitude, longitude, azimuth, and elevation. Specify `angleUnit` as `'degrees'` (the default) or `'radians'`.

Examples

Calculate AER Coordinates from Geodetic Coordinates

Find the AER coordinates of the Matterhorn with respect to Zermatt, Switzerland, using the geodetic coordinates of Zermatt and the Matterhorn.

First, specify the reference spheroid as WGS 84. For more information about WGS 84, see "Reference Spheroids". The units for ellipsoidal height and slant range must match the units specified by the `LengthUnit` property of the reference spheroid. The default length unit for the reference spheroid created by `wgs84Ellipsoid` is `'meter'`.

```
wgs84 = wgs84Ellipsoid;
```

Specify the geodetic coordinates of the local origin. In this example, the local origin is Zermatt. Specify `h0` as ellipsoidal height in meters.

```
lat0 = 46.017;
lon0 = 7.750;
h0 = 1673;
```

Specify the geodetic coordinates of the point of interest. In this example, the point of interest is the Matterhorn. Specify `h` as ellipsoidal height in meters.

```
lat = 45.977;
lon = 7.658;
h = 4531;
```

Then, calculate the AER coordinates of the Matterhorn with respect to Zermatt. To view the results in standard notation, specify the display format as `shortG`.

```
format shortG
[az,elev,slantRange] = geodetic2aer(lat,lon,h,lat0,lon0,h0,wgs84)

az =
      238.08


elev =
      18.744


slantRange =
      8876.8
```

Reverse the transformation using the `aer2geodetic` function.

```
[lat,lon,h] = aer2geodetic(az,elev,slantRange,lat0,lon0,h0,wgs84)

lat =
      45.977


lon =
       7.658


h =
        4531
```

Input Arguments

`lat` — Geodetic latitude
scalar | vector | matrix | N-D array

Geodetic latitude of one or more points, specified as a scalar, vector, matrix, or N-D array. Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

`lon` — Geodetic longitude
scalar | vector | matrix | N-D array

Geodetic longitude of one or more points, specified as a scalar, vector, matrix, or N-D array. Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

`h` — Ellipsoidal height
scalar | vector | matrix | N-D array

Ellipsoidal height of one or more points, specified as a scalar, vector, matrix, or N-D array. Specify values in units that match the `LengthUnit` property of the `spheroid` object. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

For more information about ellipsoidal height, see "Find Ellipsoidal Height from Orthometric Height".

Data Types: `single` | `double`

lat0 — Geodetic latitude of local origin
scalar | vector | matrix | N-D array

Geodetic latitude of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

lon0 — Geodetic longitude of local origin
scalar | vector | matrix | N-D array

Geodetic longitude of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

h0 — Ellipsoidal height of local origin
scalar | vector | matrix | N-D array

Ellipsoidal height of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify values in units that match the `LengthUnit` property of the `spheroid` object. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

For more information about ellipsoidal height, see "Find Ellipsoidal Height from Orthometric Height".

Data Types: `single` | `double`

spheroid — Reference spheroid
`referenceEllipsoid` object | `oblateSpheroid` object | `referenceSphere` object

Reference spheroid, specified as a `referenceEllipsoid` object, `oblateSpheroid` object, or `referenceSphere` object. The term reference spheroid is used synonymously with reference ellipsoid. To create a reference spheroid, use the creation function for the object. To specify the reference ellipsoid for WGS84, use the `wgs84Ellipsoid` function.

For more information about reference spheroids, see "Reference Spheroids".

Example: `spheroid = referenceEllipsoid('GRS 80');`

angleUnit — Angle units
`'degrees'` (default) | `'radians'`

Angle units, specified as `'degrees'` (the default) or `'radians'`.

Output Arguments

az — Azimuth angles
scalar | vector | matrix | N-D array

Azimuth angles of one or more points in the local AER system, returned as a scalar, vector, matrix, or N-D array. Azimuths are measured clockwise from north. Values are specified in degrees within the half-open interval [0 360). To use values in radians, specify the `angleUnit` argument as `'radians'`.

`elev` — Elevation angles
scalar | vector | matrix | N-D array

Elevation angles of one or more points in the local AER system, returned as a scalar, vector, matrix, or N-D array. Elevations are measured with respect to a plane that is perpendicular to the normal of the spheroid surface. If the local origin is on the surface of the spheroid (`h0 = 0`), then the plane is tangent to the spheroid.

Values are specified in degrees within the closed interval [-90 90]. To use values in radians, specify the `angleUnit` argument as `'radians'`.

`slantRange` — Distances from local origin
scalar | vector | matrix | N-D array

Distances from the local origin, returned as a scalar, vector, matrix, or N-D array. Each distance is calculated along a straight, 3-D, Cartesian line. Units are specified by the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

aer2geodetic | ecef2aer | geodetic2enu | geodetic2ned

Topics
"Choose a 3-D Coordinate System"

Introduced in R2012b

geodetic2ecef

Transform geodetic coordinates to geocentric Earth-centered Earth-fixed

Syntax

```
[X,Y,Z] = geodetic2ecef(spheroid,lat,lon,h)
[X,Y,Z] = geodetic2ecef( ___ ,angleUnit)

[X,Y,Z] = geodetic2ecef(lat,lon,h,spheroid)
```

Description

[X,Y,Z] = geodetic2ecef(spheroid,lat,lon,h) transforms the geodetic coordinates specified by lat, lon, and h to the geocentric Earth-Centered Earth-Fixed (ECEF) Cartesian coordinates specified by X, Y, and Z. Specify spheroid as the reference spheroid for the geodetic coordinates.

[X,Y,Z] = geodetic2ecef(___ ,angleUnit) specifies the units for latitude and longitude. Specify angleUnit as 'degrees' (the default) or 'radians'.

[X,Y,Z] = geodetic2ecef(lat,lon,h,spheroid) is supported but not recommended. Unlike the previous syntaxes, specify lat and lon in radians. Specify spheroid as either a reference spheroid or an ellipsoid vector of the form [semimajor_axis, eccentricity]. Specify h in the same units as the length unit of the spheroid argument. Additionally, the outputs X, Y, and Z return in the same units as the length unit of the spheroid argument.

Examples

Calculate ECEF Coordinates Using Geodetic Coordinates

Find the ECEF coordinates of Paris, France, using its geodetic coordinates.

First, specify the reference spheroid as WGS84 with length units measured in kilometers. For more information about WGS84, see "Reference Spheroids". The units for the ellipsoidal height and ECEF coordinates must match the units specified by the LengthUnit property of the reference spheroid.

```
wgs84 = wgs84Ellipsoid('kilometer');
```

Specify the geodetic coordinates of Paris. Specify h as ellipsoidal height in kilometers.

```
lat = 48.8562;
lon = 2.3508;
h = 0.0674;
```

Then, calculate the ECEF coordinates of Paris. In this example, x and y display in scientific notation.

```
[x,y,z] = geodetic2ecef(wgs84,lat,lon,h)
```

x = 4.2010e+03

y = 172.4603

```
z = 4.7801e+03
```

Reverse the transformation using the `ecef2geodetic` function.

```
[lat,lon,h] = ecef2geodetic(wgs84,x,y,z)
```

```
lat = 48.8562
```

```
lon = 2.3508
```

```
h = 0.0674
```

Input Arguments

spheroid — Reference spheroid
`referenceEllipsoid` object | `oblateSpheroid` object | `referenceSphere` object

Reference spheroid, specified as a `referenceEllipsoid` object, `oblateSpheroid` object, or `referenceSphere` object. The term reference spheroid is used synonymously with reference ellipsoid. To create a reference spheroid, use the creation function for the object. To specify the reference ellipsoid for WGS84, use the `wgs84Ellipsoid` function.

For more information about reference spheroids, see "Reference Spheroids".

Example: `spheroid = referenceEllipsoid('GRS 80');`

lat — Geodetic latitude
scalar | vector | matrix | N-D array

Geodetic latitude of one or more points, specified as a scalar, vector, matrix, or N-D array. Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

lon — Geodetic longitude
scalar | vector | matrix | N-D array

Geodetic longitude of one or more points, specified as a scalar, vector, matrix, or N-D array. Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

h — Ellipsoidal height
scalar | vector | matrix | N-D array

Ellipsoidal height of one or more points, specified as a scalar, vector, matrix, or N-D array. Specify values in units that match the `LengthUnit` property of the `spheroid` object. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

For more information about ellipsoidal height, see "Find Ellipsoidal Height from Orthometric Height".

Data Types: `single` | `double`

angleUnit — Angle units
`'degrees'` (default) | `'radians'`

Angle units, specified as `'degrees'` (the default) or `'radians'`.

Output Arguments

X — ECEF *x*-coordinates
scalar | vector | matrix | N-D array

ECEF *x*-coordinates of one or more points in the geocentric ECEF system, returned as a scalar, vector, matrix, or N-D array. Units are specified by the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Y — ECEF *y*-coordinates
scalar | vector | matrix | N-D array

ECEF *y*-coordinates of one or more points in the geocentric ECEF system, returned as a scalar, vector, matrix, or N-D array. Units are specified by the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Z — ECEF *z*-coordinates
scalar | vector | matrix | N-D array

ECEF *z*-coordinates of one or more points in the geocentric ECEF system, returned as a scalar, vector, matrix, or N-D array. Units are specified by the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Tips

- The geocentric Cartesian (ECEF) coordinate system is fixed with respect to the Earth, with its origin at the center of the spheroid and its positive *x*-, *y*-, and *z*-axes intersecting the surface at the following points:

	Latitude	Longitude	Notes
x-axis	0	0	Equator at the Prime Meridian
y-axis	0	90	Equator at 90-degrees East
z-axis	90	0	North Pole

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also
`ecef2geodetic` | `ecef0ffset` | `geodetic2aer` | `geodetic2ned` | `geodetic2enu` | `ecef2enu`

Topics
"Choose a 3-D Coordinate System"

Introduced in R2012b

geodetic2enu

Transform geodetic coordinates to local east-north-up

Syntax

```
[xEast,yNorth,zUp] = geodetic2enu(lat,lon,h,lat0,lon0,h0,spheroid)
[xEast,yNorth,zUp] = geodetic2enu( ___ ,angleUnit)
```

Description

`[xEast,yNorth,zUp] = geodetic2enu(lat,lon,h,lat0,lon0,h0,spheroid)` transforms the geodetic coordinates specified by `lat`, `lon`, and `h` to the local east-north-up (ENU) Cartesian coordinates specified by `xEast`, `yNorth`, and `zDown`. Specify the origin of the local ENU system with the geodetic coordinates `lat0`, `lon0`, and `h0`. Each coordinate input argument must match the others in size or be scalar. Specify `spheroid` as the reference spheroid for the geodetic coordinates.

`[xEast,yNorth,zUp] = geodetic2enu( ___ ,angleUnit)` specifies the units for latitude and longitude. Specify `angleUnit` as `'degrees'` (the default) or `'radians'`.

Examples

Calculate ENU Coordinates from Geodetic Coordinates

Find the ENU coordinates of the Matterhorn with respect to Zermatt, Switzerland, using their geodetic coordinates.

First, specify the reference spheroid as WGS84. For more information about WGS84, see "Reference Spheroids". The units for ellipsoidal height and ENU coordinates must match the units specified by the `LengthUnit` property of the reference spheroid. The default length unit for the reference spheroid created by `wgs84Ellipsoid` is `'meter'`.

```
wgs84 = wgs84Ellipsoid;
```

Specify the geodetic coordinates of the local origin. In this example, the local origin is Zermatt, Switzerland. Specify `h0` as ellipsoidal height in meters.

```
lat0 = 46.017;
lon0 = 7.750;
h0 = 1673;
```

Specify the geodetic coordinates of the point of interest. In this example, the point of interest is the Matterhorn. Specify `h` as ellipsoidal height in meters.

```
lat = 45.976;
lon = 7.658;
h = 4531;
```

Then, calculate the ENU coordinates of the Matterhorn with respect to Zermatt. View the results in standard notation by specifying the display format as `shortG`.

```
format shortG
[xEast,yNorth,zUp] = geodetic2enu(lat,lon,h,lat0,lon0,h0,wgs84)

xEast =
      -7134.8


yNorth =
      -4556.3


zUp =
       2852.4
```

Reverse the transformation using the `enu2geodetic` function.

```
[lat,lon,h] = enu2geodetic(xEast,yNorth,zUp,lat0,lon0,h0,wgs84)

lat =
       45.976


lon =
        7.658


h =
         4531
```

Input Arguments

`lat` — Geodetic latitude
scalar | vector | matrix | N-D array

Geodetic latitude of one or more points, specified as a scalar, vector, matrix, or N-D array. Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

`lon` — Geodetic longitude
scalar | vector | matrix | N-D array

Geodetic longitude of one or more points, specified as a scalar, vector, matrix, or N-D array. Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

`h` — Ellipsoidal height
scalar | vector | matrix | N-D array

Ellipsoidal height of one or more points, specified as a scalar, vector, matrix, or N-D array. Specify values in units that match the `LengthUnit` property of the `spheroid` object. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

For more information about ellipsoidal height, see "Find Ellipsoidal Height from Orthometric Height".

Data Types: single | double

lat0 — Geodetic latitude of local origin
scalar | vector | matrix | N-D array

Geodetic latitude of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify the values in degrees. To use values in radians, specify the angleUnit argument as 'radians'.

Data Types: single | double

lon0 — Geodetic longitude of local origin
scalar | vector | matrix | N-D array

Geodetic longitude of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify the values in degrees. To use values in radians, specify the angleUnit argument as 'radians'.

Data Types: single | double

h0 — Ellipsoidal height of local origin
scalar | vector | matrix | N-D array

Ellipsoidal height of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify values in units that match the LengthUnit property of the spheroid object. For example, the default length unit for the reference ellipsoid created by wgs84Ellipsoid is 'meter'.

For more information about ellipsoidal height, see "Find Ellipsoidal Height from Orthometric Height".

Data Types: single | double

spheroid — Reference spheroid
referenceEllipsoid object | oblateSpheroid object | referenceSphere object

Reference spheroid, specified as a referenceEllipsoid object, oblateSpheroid object, or referenceSphere object. The term reference spheroid is used synonymously with reference ellipsoid. To create a reference spheroid, use the creation function for the object. To specify the reference ellipsoid for WGS84, use the wgs84Ellipsoid function.

For more information about reference spheroids, see "Reference Spheroids".

Example: spheroid = referenceEllipsoid('GRS 80');

angleUnit — Angle units
'degrees' (default) | 'radians'

Angle units, specified as 'degrees' (the default) or 'radians'.

Output Arguments

xEast — ENU *x*-coordinates
scalar | vector | matrix | N-D array

ENU *x*-coordinates of one or more points in the local ENU system, returned as a scalar, vector, matrix, or N-D array. Units are specified by the LengthUnit property of the spheroid argument.

For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

yNorth — ENU *y*-coordinates
scalar | vector | matrix | N-D array

ENU *y*-coordinates of one or more points in the local ENU system, returned as a scalar, vector, matrix, or N-D array. Units are specified by the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

zUp — ENU *z*-coordinates
scalar | vector | matrix | N-D array

ENU *z*-coordinates of one or more points in the local ENU system, returned as a scalar, vector, matrix, or N-D array. Units are specified by the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

ecef2enu | enu2geodetic | geodetic2aer | geodetic2ned

Topics
"Choose a 3-D Coordinate System"

Introduced in R2012b

geodetic2ned

Transform geodetic coordinates to local north-east-down

Syntax

```
[xNorth,yEast,zDown] = geodetic2ned(lat,lon,h,lat0,lon0,h0,spheroid)
[ ___ ] = geodetic2ned( ___ ,angleUnit)
```

Description

`[xNorth,yEast,zDown] = geodetic2ned(lat,lon,h,lat0,lon0,h0,spheroid)` transforms the geodetic coordinates specified by `lat`, `lon`, and `h` to the local north-east-down (NED) Cartesian coordinates specified by `xNorth`, `yEast`, and `zDown`. Specify the origin of the local NED system with the geodetic coordinates `lat0`, `lon0`, and `h0`. Each coordinate input argument must match the others in size or be scalar. Specify `spheroid` as the reference spheroid for the geodetic coordinates.

`[ ___ ] = geodetic2ned( ___ ,angleUnit)` specifies the units for latitude and longitude. Specify `angleUnit` as `'degrees'` (the default) or `'radians'`.

Examples

Calculate NED Coordinates from Geodetic Coordinates

Find the NED coordinates of Mount Mansfield with respect to a nearby aircraft, using their geodetic coordinates.

First, specify the reference spheroid as WGS 84. For more information about WGS 84, see "Reference Spheroids". The units for the ellipsoidal height and NED coordinates must match the units specified by the `LengthUnit` property of the reference spheroid. The default length unit for the reference spheroid created by `wgs84Ellipsoid` is `'meter'`.

```
wgs84 = wgs84Ellipsoid;
```

Specify the geodetic coordinates of the local origin. In this example, the local origin is the aircraft. Specify `h0` as ellipsoidal height in meters.

```
lat0 = 44.532;
lon0 = -72.782;
h0 = 1699;
```

Specify the geodetic coordinates of the point of interest. In this example, the point of interest is Mount Mansfield. Specify `h` as ellipsoidal height in meters.

```
lat = 44.544;
lon = -72.814;
h = 1340;
```

Then, calculate the NED coordinates of Mount Mansfield with respect to the aircraft. Since the ellipsoidal height of the aircraft is greater than the height of Mount Mansfield, a passenger needs to look down to see the mountaintop. The *z*-axis of an NED coordinate system points down. Thus, the

value of `zDown` is positive. View the results in standard notation by specifying the display format as `shortG`.

```
format shortG
[xNorth,yEast,zDown] = geodetic2ned(lat,lon,h,lat0,lon0,h0,wgs84)

xNorth =
      1334.3


yEast =
      -2543.6


zDown =
      359.65
```

Reverse the transformation using the `ned2geodetic` function.

```
[lat,lon,h] = ned2geodetic(xNorth,yEast,zDown,lat0,lon0,h0,wgs84)

lat =
      44.544


lon =
      -72.814


h =
         1340
```

Input Arguments

lat — Geodetic latitude
scalar | vector | matrix | N-D array

Geodetic latitude of one or more points, specified as a scalar, vector, matrix, or N-D array. Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

lon — Geodetic longitude
scalar | vector | matrix | N-D array

Geodetic longitude of one or more points, specified as a scalar, vector, matrix, or N-D array. Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

h — Ellipsoidal height
scalar | vector | matrix | N-D array

Ellipsoidal height of one or more points, specified as a scalar, vector, matrix, or N-D array. Specify values in units that match the `LengthUnit` property of the `spheroid` object. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

For more information about ellipsoidal height, see "Find Ellipsoidal Height from Orthometric Height".

Data Types: `single` | `double`

lat0 — Geodetic latitude of local origin
scalar | vector | matrix | N-D array

Geodetic latitude of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

lon0 — Geodetic longitude of local origin
scalar | vector | matrix | N-D array

Geodetic longitude of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

h0 — Ellipsoidal height of local origin
scalar | vector | matrix | N-D array

Ellipsoidal height of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify values in units that match the `LengthUnit` property of the `spheroid` object. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

For more information about ellipsoidal height, see "Find Ellipsoidal Height from Orthometric Height".

Data Types: `single` | `double`

spheroid — Reference spheroid
`referenceEllipsoid` object | `oblateSpheroid` object | `referenceSphere` object

Reference spheroid, specified as a `referenceEllipsoid` object, `oblateSpheroid` object, or `referenceSphere` object. The term reference spheroid is used synonymously with reference ellipsoid. To create a reference spheroid, use the creation function for the object. To specify the reference ellipsoid for WGS84, use the `wgs84Ellipsoid` function.

For more information about reference spheroids, see "Reference Spheroids".

Example: `spheroid = referenceEllipsoid('GRS 80');`

angleUnit — Angle units
`'degrees'` (default) | `'radians'`

Angle units, specified as `'degrees'` (the default) or `'radians'`.

Output Arguments

xNorth — NED *x*-coordinates
scalar | vector | matrix | N-D array

NED *x*-coordinates of one or more points in the local NED system, returned as a scalar, vector, matrix, or N-D array. Units are specified by the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

yEast — NED *y*-coordinates
scalar | vector | matrix | N-D array

NED *y*-coordinates of one or more points in the local NED system, returned as a scalar, vector, matrix, or N-D array. Units are specified by the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

zDown — NED *z*-coordinates
scalar | vector | matrix | N-D array

NED *z*-coordinates of one or more points in the local NED system, returned as a scalar, vector, matrix, or N-D array. Units are specified by the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also
`ecef2ned` | `ned2geodetic` | `geodetic2aer` | `geodetic2enu`

Topics
"Choose a 3-D Coordinate System"

Introduced in R2012b

geodetic2geocentricLat

Convert geodetic to geocentric latitude

Note geodetic2geocentricLat will be removed in a future release. Use geocentricLatitude instead.

Syntax

phi_g = geodetic2geocentriclat(ecc, phi)

Description

phi_g = geodetic2geocentriclat(ecc, phi) converts an array of geodetic latitude in radians, phi, to geocentric latitude in radians, phi_g, on a reference ellipsoid with first eccentricity ecc.

See Also
geocentricLatitude

Introduced in R2006b

geodeticLatitudeFromGeocentric

Convert geocentric to geodetic latitude

Syntax

```
phi = geodeticLatitudeFromGeocentric(psi,F)
phi = geodeticLatitudeFromGeocentric(psi,F,angleUnit)
```

Description

`phi = geodeticLatitudeFromGeocentric(psi,F)` returns the geodetic latitude corresponding to geocentric latitude `psi` on an ellipsoid with flattening `F`.

`phi = geodeticLatitudeFromGeocentric(psi,F,angleUnit)` specifies the units of input `psi` and output `phi`.

Examples

Convert Geocentric Latitude to Geodetic Latitude

Create a reference ellipsoid.

```
s = wgs84Ellipsoid;
```

Convert the geocentric latitude to geodetic latitude. The reference ellipsoid contains a flattening factor.

```
geodeticLatitudeFromGeocentric(45, s.Flattening)
```

```
ans = 45.1924
```

Convert Geocentric Latitude Expressed in Radians to Geodetic Latitude

Create a reference ellipsoid.

```
s = wgs84Ellipsoid;
```

Convert a geocentric latitude expressed in radians to geodetic latitude. The reference ellipsoid contains a flattening factor.

```
geodeticLatitudeFromGeocentric(pi/3, s.Flattening, 'radians')
```

```
ans = 1.0501
```

Input Arguments

psi — Geocentric latitude of one or more points
scalar value, vector, matrix, or N-D array

Geocentric latitudes of one or more points, returned as a scalar value, vector, matrix, or N-D array. Values are in units that match the input argument `angleUnit`, if supplied, and in degrees, otherwise.

Data Types: `single` | `double`

F — Flattening of reference spheroid
scalar

Flattening of reference spheroid, specified as a scalar value.

Data Types: `double`

angleUnit — Unit of measurement for angle
`'degrees'` (default) | `'radians'`

Unit of measurement for angle, specified as either `'degrees'` or `'radians'`.

Data Types: `char`

Output Arguments

phi — Geodetic latitude of one or more points
scalar value, vector, matrix, or N-D array

Geodetic latitude of one or more points, returned as a scalar value, vector, matrix, or N-D array. Values must be in units that match the input argument `angleUnit`, if supplied, and in degrees, otherwise.

See Also

Functions
geodeticLatitudeFromParametric | geocentricLatitude

Objects
AuthalicLatitudeConverter | ConformalLatitudeConverter | IsometricLatitudeConverter | RectifyingLatitudeConverter

Introduced in R2013a

geodeticLatitudeFromParametric

Convert parametric to geodetic latitude

Syntax

```
phi = geodeticLatitudeFromParametric(beta,F)
phi = geodeticLatitudeFromParametric(beta,F,angleUnit)
```

Description

`phi = geodeticLatitudeFromParametric(beta,F)` returns the geodetic latitude corresponding to parametric latitude `beta` on an ellipsoid with flattening `F`.

`phi = geodeticLatitudeFromParametric(beta,F,angleUnit)` specifies the units of input `beta` and output `phi`.

Examples

Convert Parametric Latitude to Geodetic Latitude

Create a reference ellipsoid and then convert the parametric latitude to geodetic latitude. The reference ellipsoid contains a flattening factor.

```
s = wgs84Ellipsoid;

geodeticLatitudeFromParametric(45, s.Flattening)

ans =

    45.0962
```

Convert Parametric Latitude Expressed in Radians to Geodetic Latitude

Create a reference ellipsoid and then convert a parametric latitude expressed in radians to geodetic latitude. The reference ellipsoid contains a flattening factor.

```
s = wgs84Ellipsoid;

geodeticLatitudeFromParametric(pi/3, s.Flattening, 'radians')

ans =

    1.0487
```

Input Arguments

beta — Parametric latitude of one or more points
scalar value, vector, matrix, or N-D array

Parametric latitude of one or more points, specified as a scalar value, vector, matrix, or N-D array. Values must be in units that match the input argument `angleUnit`, if supplied, and in degrees, otherwise.

Data Types: `single` | `double`

F — Flattening of reference spheroid
scalar

Flattening of reference spheroid, specified as a scalar value.

Data Types: `double`

angleUnit — Unit of measurement for angle
`'degrees'` (default) | `'radians'`

Unit of measurement for angle, specified as either `'degrees'` or `'radians'`.

Data Types: `char`

Output Arguments

phi — Geodetic latitudes of one or more points
scalar value, vector, matrix, or N-D array

Geodetic latitudes of one or more points, returned as a scalar value, vector, matrix, or N-D array. Values are in units that match the input argument `angleUnit`, if supplied, and in degrees, otherwise.

See Also

Functions
geodeticLatitudeFromGeocentric | parametricLatitude

Objects
AuthalicLatitudeConverter | ConformalLatitudeConverter | IsometricLatitudeConverter | RectifyingLatitudeConverter

Introduced in R2013a

geoglobe

Create geographic globe

Syntax

```
geoglobe(parent)
geoglobe(parent,Name,Value)
g = geoglobe( ___ )
```

Description

`geoglobe(parent)` creates a geographic globe in the specified figure, panel, or tab group. The figure must be created using the `uifigure` function. For information about navigating the globe, see "Geographic Globe Navigation" on page 1-474.

The geographic globe requires hardware graphics support for WebGL™. For more information, see "Tips" on page 1-474.

`geoglobe(parent,Name,Value)` specifies additional options for the globe using one or more name-value pair arguments. Specify the options after all other input arguments. For a list of options, see `GeographicGlobe`.
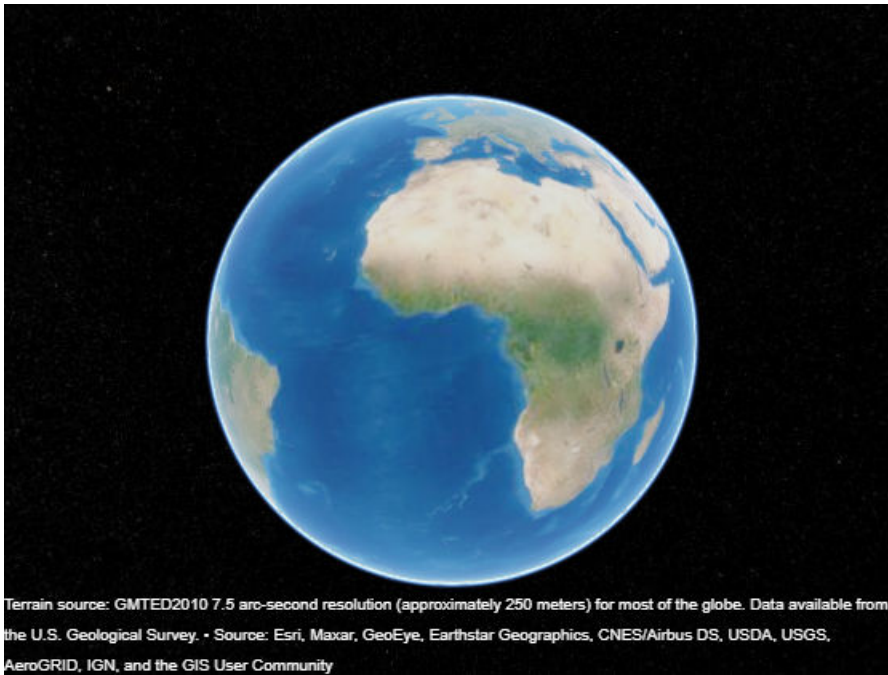
`g = geoglobe( ___ )` returns a `GeographicGlobe` object. This syntax is useful for controlling the properties of the geographic globe.

Examples

Display Geographic Globe

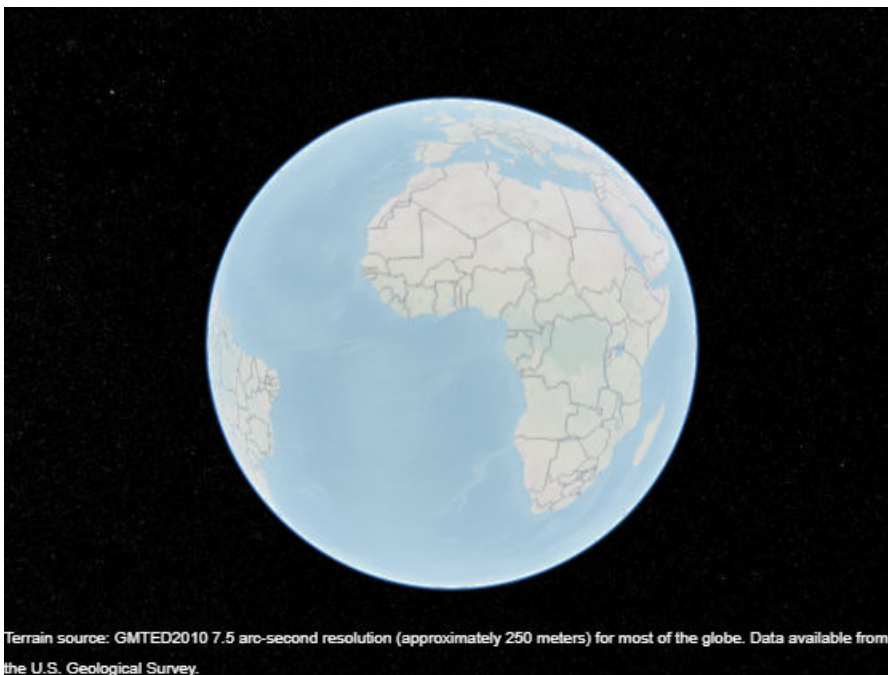Display a geographic globe in a figure created using the `uifigure` function.

```
uif = uifigure;
g = geoglobe(uif);
```

Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community
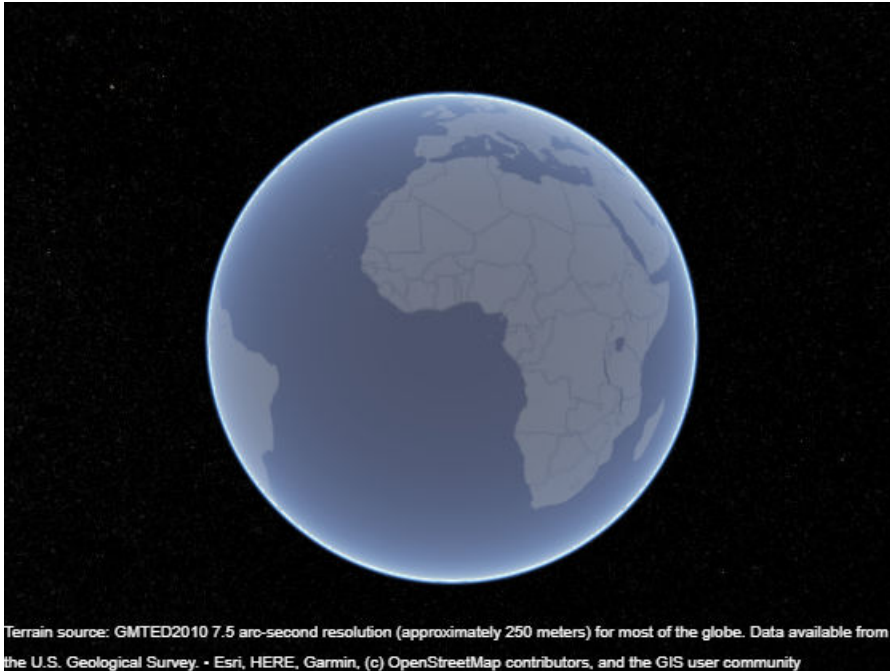
Specify and Change Basemap

Create a geographic globe. Specify the basemap using a name-value pair.

```
uif = uifigure;
g = geoglobe(uif,'Basemap','landcover');
```


Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey.

Change the basemap of an existing geographic globe by using the `geobasemap` function.

```
geobasemap(g,'streets-dark')
```


Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Esri, HERE, Garmin, (c) OpenStreetMap contributors, and the GIS user community

Display Geographic Globes in Tabs

Create a tab group that fills an entire figure. To make the group fill the figure, specify the width and height of the group as the width and height of the figure. Then, display a geographic globe in each tab.

```
uif = uifigure;
pos = [0 0 uif.Position(3) uif.Position(4)];
tgroup = uitabgroup(uif,'Position',pos);

tab1 = uitab(tgroup,'Title','Default Basemap');
g1 = geoglobe(tab1);

tab2 = uitab(tgroup,'Title','Gray Terrain Basemap');
g2 = geoglobe(tab2,'Basemap','grayterrain');
```

Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Input Arguments

parent — Parent container
`Figure` object created using `uifigure` | `Panel` object | `Tab` object

Parent container, specified as a `Figure` object created using the `uifigure` function, a `Panel` object, or a `Tab` object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `geoglobe(uif,'Basemap','streets')` sets the basemap of the geographic globe

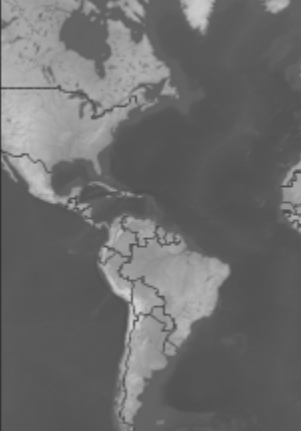
Note The properties listed here are only a subset. For a full list, see `GeographicGlobe`.

Basemap — Map to plot data on
`'satellite'` (default) | `'streets'` | `'streets-light'` | `'streets-dark'` | custom basemap | ...

Map on which to plot data, specified as one of the values listed in the table. Six of the basemaps in the table are tiled data sets created using Natural Earth. Five of the basemaps are high-zoom-level maps hosted by Esri.

	'satellite' (default) Full global basemap composed of high-resolution satellite imagery. Hosted by Esri.	**'streets'** General-purpose road map that emphasizes accurate, legible styling of roads and transit networks. Hosted by Esri.
'streets-light' Map designed to provide geographic context while highlighting user data on a light background. Hosted by Esri.		**'streets-dark'** Map designed to provide geographic context while highlighting user data on a dark background. Hosted by Esri.
'topographic' General-purpose map with styling to depict topographic features. Hosted by Esri.		**'landcover'** Map that combines satellite-derived land cover data, shaded relief, and ocean-bottom relief. The light, natural palette is suitable for thematic and reference maps. Created using Natural Earth.

	`'colorterrain'` Shaded relief map blended with a land cover palette. Humid lowlands are green and arid lowlands are brown. Created using Natural Earth.		`'grayterrain'` Terrain map in shades of gray. Shaded relief emphasizes both high mountains and micro-terrain found in lowlands. Created using Natural Earth.
	`'bluegreen'` Two-tone, land-ocean map with light green land areas and light blue water areas. Created using Natural Earth.		`'grayland'` Two-tone, land-ocean map with gray land areas and white water areas. Created using Natural Earth.
	`'darkwater'` Two-tone, land-ocean map with light gray land areas and dark gray water areas. This basemap is installed with MATLAB. Created using Natural Earth.	Not applicable.	Custom basemap added using the `addCustomBasemap` function.

All basemaps except `'darkwater'` require Internet access. The `'darkwater'` basemap is included with MATLAB and Mapping Toolbox.

If you do not have consistent access to the Internet, you can download the basemaps created using Natural Earth onto your local system by using the Add-On Explorer. The basemaps hosted by Esri are not available for download. For more about downloading basemaps and changing the default basemap on your local system, see "Access Basemaps and Terrain for Geographic Globe".

The basemaps hosted by Esri update periodically. As a result, you might see differences in your visualizations over time.

Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

Example: `g = geoglobe(uifigure,'Basemap','bluegreen')`

Example: `g.Basemap = 'bluegreen'`

Data Types: `char` | `string`

Terrain — Terrain data
`'gmted2010'` (default) | `'none'` | string scalar | character vector

Terrain data, specified as one of these values:

- `'gmted2010'` – Tiled global terrain derived from the GMTED2010 model by the U.S. Geological Survey (USGS) and National Geospatial-Intelligence Agency (NGA) and hosted by MathWorks. Internet access is required to use `'gmted2010'`.
- `'none'` – No terrain.
- String scalar or character vector – Name of custom terrain added using the `addCustomTerrain` function.
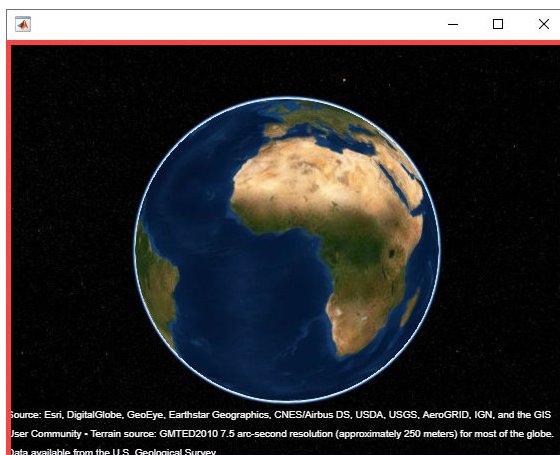
Position — Size and location
`[0 0 1 1]` (default) | four-element vector of form `[left bottom width height]`

Size and location, specified as a four-element vector of the form `[left bottom width height]`. By default, MATLAB measures the values in units normalized to the container. To change the units, set the `Units` property.

- The `left` and `bottom` elements define the distance from the lower-left corner of the container figure, panel, or tab to the lower-left corner of the position boundary.
- The `width` and `height` elements are the position boundary dimensions.

This red line in this figure shows the position boundary of the geographic globe.

Units — Position units
`'normalized'` (default) | `'inches'` | `'centimeters'` | `'points'` | `'pixels'` | `'characters'`

Position units, specified as one of these values.

Units	Description
`'normalized'` (default)	Units normalized with respect to the container, which is typically the figure or a panel. The lower left corner of the container is `(0,0)` and the upper right corner is `(1,1)`.
`'inches'`	Inches.
`'centimeters'`	Centimeters.
`'characters'`	Units based on the default `uicontrol` font of the graphics root object: • The character width is the width of the letter *x*. • The character height is the distance between the baselines of two lines of text.
`'points'`	Typography points. One point equals 1/72 inch.
`'pixels'`	Pixels. Distances in pixels are independent of your system resolution on Windows and Macintosh systems. • On Windows systems, a pixel is 1/96th of an inch. • On Macintosh systems, a pixel is 1/72nd of an inch. • On Linux® systems, the size of a pixel is determined by your system resolution.

When specifying the units as a name-value pair during object creation, specify the `Units` name-value pair before specifying name-value pairs that use those units, for example `Position`.

Limitations

• In the Live Editor, geographic globes appear in a separate window instead of within the live script.
• In MATLAB Online, if you refresh the URL, then the UI figure remains open, but the geographic globe visualization disappears. As a workaround, this code shows how to use the `copyobj` function to copy the property values of a geographic globe, `g`, into a new geographic globe, `g2`, with the same parent, and then delete the original geographic globe. The code can take several minutes to run.

```
g2 = copyobj(g,g.Parent);
delete(g)
```

• Deploying geographic globes using MATLAB Compiler is not supported on Linux.
• If multiple windows requiring WebGL are open at once, then the geographic globe might display this error:

```
Globe Viewer needs to close because the WebGL context has been lost.
```

More About

Geographic Globe Navigation

Interactively navigate the globe using your mouse.

- Pan by left-clicking and dragging.
- Zoom by scrolling or by right-clicking and dragging.
- Tilt and rotate by holding **Ctrl** and dragging or by middle-clicking and dragging.

On a touch screen, navigate the globe using gestures.

- Pan by dragging one finger.
- Zoom by pinching two fingers.
- Tilt by dragging two fingers in the same direction.
- Rotate by dragging two fingers in a circle.

To programmatically navigate the globe, use the `campos`, `camheight`, `camheading`, `campitch`, and `camroll` object functions.

Tips

- The geographic globe requires hardware graphics support for WebGL. To determine if your system has hardware graphics support for WebGL, display axes in a figure created using the `uifigure` function. Get renderer info about the axes using the `rendererinfo` function. Then, query the `GraphicsRenderer` property. Your system has hardware graphics support if the `GraphicsRenderer` property has a value of `'WebGL'`.

  ```
  fig = uifigure;
  ax = axes(fig);
  info = rendererinfo(ax);
  info.GraphicsRenderer

  ans =

      'WebGL'
  ```

- If you create a geographic globe with no output argument, then you can assign the globe to a variable later by using the `findall` function. If there is more than one geographic globe, then `findall` returns a vector of globe objects.

  ```
  uif = uifigure;
  geoglobe(uif)
  g = findall(groot,'Type','globe');
  ```

  If there is more than one geographic globe, then `findall` returns a vector of globe objects.

  ```
  uif = uifigure;
  geoglobe(uif)
  uif2 = uifigure;
  geoglobe(uif2)
  g = findall(groot,'Type','globe')

  g =
  ```

```
2×1 GeographicGlobe array:

GeographicGlobe
GeographicGlobe
```

See Also

Functions
geoplot3

Objects
GeographicGlobe

**Introduced in R2020a**

GeographicCellsReference

Reference raster cells to geographic coordinates

Description

A geographic cells raster reference object contains information that describes the relationship between a geographic coordinate system and an intrinsic coordinate system on page 1-481 that is anchored to the columns and rows of a 2-D spatially referenced raster grid or image of cells.

In order to reference a raster of cells to geographic coordinates by using a `GeographicCellsReference` object, the raster must be sampled regularly in latitude and longitude, and its columns and rows must be aligned with meridians and parallels, respectively. The sample spacing from row to row need not equal the sample spacing from column to column.

This image illustrates a 8-by-10 geographic raster of cells. The raster cells have an extent of 1 degree in latitude and longitude, cover longitude values in the range [40,50] in degrees, and cover latitude values in the range [20,28] in degrees. The boundary of the raster of cells is made up of the outermost boundaries of the outermost cells.
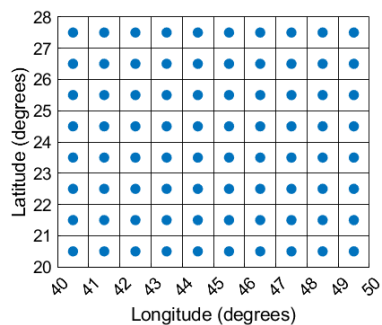
To reference a raster of postings to geographic coordinates, use a `GeographicPostingsReference` object instead.

Creation

You can use these functions to create `GeographicCellsReference` objects:

- `readgeoraster` — Import a raster data file as an array and a reference object.
- `wmsread` — Import a raster map from a Web Map Service (WMS) server as an array and a reference object.
- `georefcells` — Create a geographic raster reference object by specifying the latitude and longitude limits of the raster and either the size of the raster or the extent of the cells.
- `georasterref` — Convert a world file to a geographic raster reference object.
- `refmatToGeoRasterReference` — Convert a referencing matrix to a geographic raster reference object.
- `refvecToGeoRasterReference` — Convert a referencing vector to a geographic raster reference object.

Properties

LatitudeLimits — Latitude limits of the geographic quadrangle bounding the georeferenced raster
[0.5 2.5] (default) | two-element vector

Latitude limits of the geographic quadrangle bounding the georeferenced raster, specified as a two-element vector of the form [southern_limit northern_limit].

Example: [-90 90]

Data Types: double

LongitudeLimits — Longitude limits of the geographic quadrangle bounding the georeferenced raster
[0.5 2.5] (default) | two-element vector

Longitude limits of the geographic quadrangle bounding the georeferenced raster, specified as a two-element vector of the form [western_limit eastern_limit].

Example: [-100 180]

Data Types: double

RasterSize — Number of rows and columns of the raster or image associated with the referencing object
[2 2] (default) | two-element vector of positive integers

Number of rows and columns of the raster or image associated with the referencing object, specified as a two-element vector, [*m n*], where *m* represents the number of rows and *n* the number of columns. For convenience, you can assign a size vector having more than two elements. This enables assignments like R.RasterSize = size(RGB), where RGB is *m*-by-*n*-by-3. In cases like this, the object stores only the first two elements of the size vector and ignores the higher (nonspatial) dimensions.

Example: [200 300]

Data Types: double

RasterInterpretation — Geometric nature of the raster
'cells' (default)

This property is read-only.

Geometric nature of the raster, specified as 'cells'. The value 'cells' indicates that the raster comprises a grid of quadrangular cells, and is bounded on all sides by cell edges. For an *m*-by-*n* raster, points with an intrinsic *x*-coordinate of 1 or *n* or an intrinsic *y*-coordinate of 1 or *m* fall within the raster, not on its edges.

Data Types: char

AngleUnit — Unit of measurement used for angle-valued properties
'degree' (default)

Unit of measurement used for angle-valued properties, specified as 'degree'.

Cannot be set.

Data Types: `char`

ColumnsStartFrom — Edge from which column indexing starts
`'south'` (default) | `'north'`

Edge from which column indexing starts, specified as `'south'` or `'north'`.

Example: `'south'`

Data Types: `char`

RowsStartFrom — Edge from which row indexing starts
`'west'` (default) | `'east'`

Edge from which row indexing starts, specified as `'west'` or `'east'`.

Example: `'east'`

Data Types: `char`

CellExtentInLatitude — Extent in latitude of individual cells
1 (default) | positive numeric scalar

Extent in latitude of individual cells, specified as a positive numeric scalar. Distance, in units of latitude, between the northern and southern limits of a single raster cell. The value is the same for all cells in the raster.

Example: 2.5

Data Types: `double`

CellExtentInLongitude — Extent in longitude of individual cells
1 (default) | positive numeric scalar

Extent in longitude of individual cells, specified as a positive numeric scalar. Distance, in units of longitude, between the western and eastern limits of a single raster cell. The value is always positive, and is the same for all cells in the raster.

Example: 2.5

Data Types: `double`

RasterExtentInLatitude — Latitude extent ("height") of the quadrangle covered by the raster
2 (default) | positive numeric scalar

This property is read-only.

Latitude extent ("height") of the quadrangle covered by the raster, specified as a positive numeric scalar.

Example: 2

Data Types: `double`

RasterExtentInLongitude — Longitude extent ("width") of the quadrangle covered by the raster
2 (default) | positive numeric scalar

This property is read-only.

Longitude extent ("width") of the quadrangle covered by the raster, specified as a positive numeric scalar.

Data Types: `double`

XIntrinsicLimits — Raster limits in intrinsic *x* coordinates
[0.5 2.5] (default) | two-element row vector of positive integers

This property is read-only.

Raster limits in intrinsic *x* coordinates, specified as a two-element row vector of positive integers, `[xMin xMax]`. For an *m*-by-*n* raster, `XIntrinsicLimits` equals `[0.5, m+0.5]`, because the `RasterInterpretation` is `'cells'`.

Example: [0.5 2.5]

Data Types: `double`

YIntrinsicLimits — Raster limits in intrinsic *y* coordinates
[0.5 2.5] (default) | two-element row vector of positive integers

This property is read-only.

Raster limits in intrinsic *y* coordinates, specified as a two-element row vector of positive integers, `[yMin yMax]`. For an *m*-by-*n* raster, `YIntrinsicLimits` equals `[0.5, n+0.5]`, because the `RasterInterpretation` is `'cells'`.

Data Types: `double`

CoordinateSystemType — Type of coordinate system to which the image or raster is referenced
`'geographic'` (default)

This property is read-only.

Type of coordinate system to which the image or raster is referenced, specified as `'geographic'`.

Data Types: `char`

GeographicCRS — Geographic coordinate reference system
[] (default) | `geocrs` object

Geographic coordinate reference system (CRS), specified as a `geocrs` object. A geographic CRS consists of a datum (including its ellipsoid), prime meridian, and angular unit of measurement.

Object Functions

contains	Determine if geographic or map raster contains points
geographicGrid	Geographic coordinates of raster elements
geographicToDiscrete	Transform geographic to discrete coordinates
geographicToIntrinsic	Transform geographic to intrinsic coordinates
intrinsicToGeographic	Transform intrinsic to geographic coordinates
intrinsicXToLongitude	Convert from intrinsic x to longitude coordinates
intrinsicYToLatitude	Convert from intrinsic y to latitude coordinates
latitudeToIntrinsicY	Convert from latitude to intrinsic y coordinates
longitudeToIntrinsicX	Convert from longitude to intrinsic x coordinates

sizesMatch	Determine if geographic or map raster object and image or raster are size-compatible
worldFileMatrix	Return world file parameters for transformation

Examples

Display Geographic Raster with Regularly Spaced Cell Centers

Import a GeoTIFF image as an array and a `GeographicCellsReference` object by using the `readgeoraster` function.

```
[Z,R] = readgeoraster('katrina.tif');
```
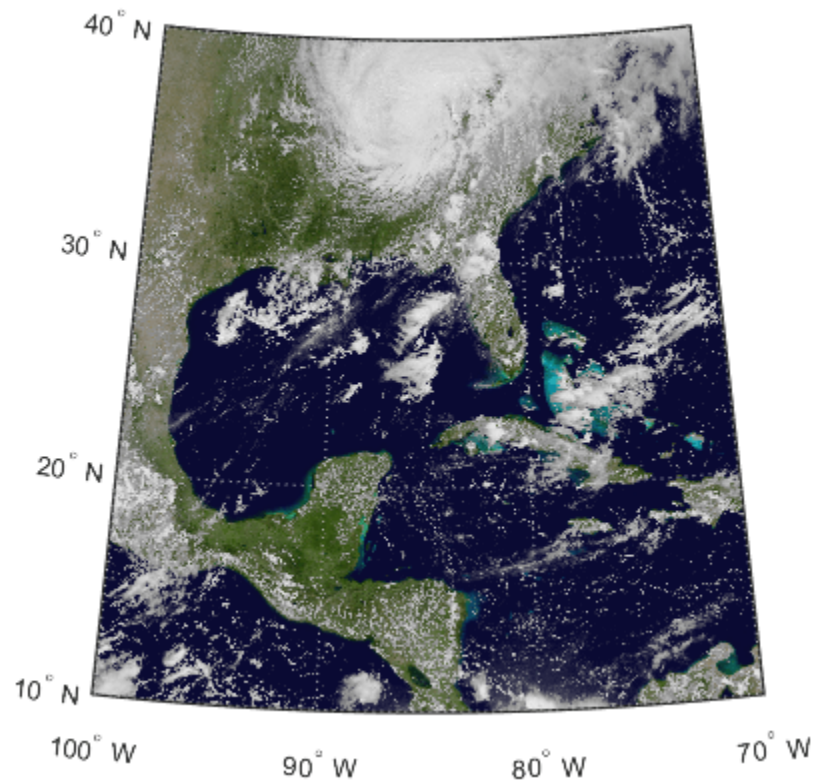
View the properties of the `GeographicCellsReference` object.

```
R
```

```
R =
  GeographicCellsReference with properties:

             LatitudeLimits: [10 40]
            LongitudeLimits: [-100 -70]
                 RasterSize: [1024 1024]
       RasterInterpretation: 'cells'
           ColumnsStartFrom: 'north'
              RowsStartFrom: 'west'
        CellExtentInLatitude: 0.029296875
      CellExtentInLongitude: 0.029296875
      RasterExtentInLatitude: 30
     RasterExtentInLongitude: 30
            XIntrinsicLimits: [0.5 1024.5]
            YIntrinsicLimits: [0.5 1024.5]
       CoordinateSystemType: 'geographic'
               GeographicCRS: [1x1 geocrs]
                   AngleUnit: 'degree'
```

Create a world map with limits that match the limits of the reference object. Then, display the image.

```
worldmap(Z,R)
geoshow(Z,R)
```

More About

Intrinsic Coordinate System

A 2-D Cartesian system with its *x*-axis running parallel to the rows of a raster or image and its *y*-axis running parallel to the columns. *x* increases by 1 from column to column, and *y* increases by 1 from row to row.

Mapping Toolbox and Image Processing Toolbox™ use the convention for the location of the origin relative to the raster cells or sampling points such that, at a sample location or at the center of a cell, *x* has an integer value equal to the column index. Likewise, at a sample location or at the center of a cell, *y* has an integer value equal to the row index. For details, see Image Coordinate Systems (Image Processing Toolbox).

See Also

Functions
georefcells | georasterref | readgeoraster | wmsread

Objects
GeographicPostingsReference | MapCellsReference | MapPostingsReference

Topics
"Spatially Reference Imported Rasters"

Introduced in R2013b

GeographicGlobe

Control geographic globe appearance and behavior

Description

Use a geographic globe to plot 3-D lines and markers over basemaps and terrain.

Creation

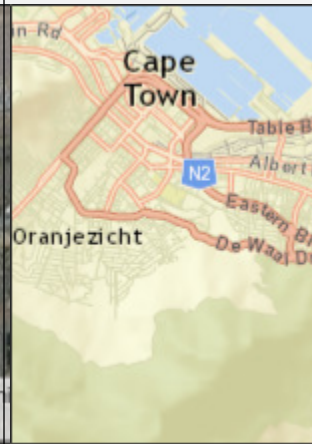Create a geographic globe object using the `geoglobe` function.

Properties

Maps

Basemap — Map to plot data on
`'satellite'` (default) | `'streets'` | `'streets-light'` | `'streets-dark'` | custom basemap | ...

Map on which to plot data, specified as one of the values listed in the table. Six of the basemaps in the table are tiled data sets created using Natural Earth. Five of the basemaps are high-zoom-level maps hosted by Esri.

	`'satellite'` (default) Full global basemap composed of high-resolution satellite imagery. Hosted by Esri.	
	`'streets'` General-purpose road map that emphasizes accurate, legible styling of roads and transit networks. Hosted by Esri.	

'streets-light'

Map designed to provide geographic context while highlighting user data on a light background.

Hosted by Esri.

Esri South Africa, HERE, Garmin, NGA, USGS

'streets-dark'

Map designed to provide geographic context while highlighting user data on a dark background.

Hosted by Esri.

Esri, HERE, Garmin, NGA, USGS

'topographic'

General-purpose map with styling to depict topographic features.

Hosted by Esri.

Esri South Africa, HERE, Garmin, USGS, NGA

'landcover'

Map that combines satellite-derived land cover data, shaded relief, and ocean-bottom relief. The light, natural palette is suitable for thematic and reference maps.

Created using Natural Earth.

'colorterrain'

Shaded relief map blended with a land cover palette. Humid lowlands are green and arid lowlands are brown.

Created using Natural Earth.

'grayterrain'

Terrain map in shades of gray. Shaded relief emphasizes both high mountains and micro-terrain found in lowlands.

Created using Natural Earth.

`'bluegreen'` Two-tone, land-ocean map with light green land areas and light blue water areas. Created using Natural Earth.		`'grayland'` Two-tone, land-ocean map with gray land areas and white water areas. Created using Natural Earth.
`'darkwater'` Two-tone, land-ocean map with light gray land areas and dark gray water areas. This basemap is installed with MATLAB. Created using Natural Earth.	Not applicable.	Custom basemap added using the `addCustomBasemap` function.

All basemaps except `'darkwater'` require Internet access. The `'darkwater'` basemap is included with MATLAB and Mapping Toolbox.

If you do not have consistent access to the Internet, you can download the basemaps created using Natural Earth onto your local system by using the Add-On Explorer. The basemaps hosted by Esri are not available for download. For more about downloading basemaps and changing the default basemap on your local system, see "Access Basemaps and Terrain for Geographic Globe".

The basemaps hosted by Esri update periodically. As a result, you might see differences in your visualizations over time.

Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

Example: `g = geoglobe(uifigure,'Basemap','bluegreen')`

Example: `g.Basemap = 'bluegreen'`

Data Types: `char` | `string`

Terrain — Terrain data
`'gmted2010'` (default) | `'none'` | string scalar | character vector

Terrain data, specified as one of these values:

- `'gmted2010'` – Global terrain derived from the GMTED2010 model created by the U.S. Geological Survey (USGS) and National Geospatial-Intelligence Agency (NGA).
- `'none'` - No terrain.
- string scalar or character vector – Name of custom terrain added using the `addCustomTerrain` function.

Multiple Plots

`ColorOrder` — Color order
seven predefined colors (default) | three-column matrix of RGB triplets

Color order for lines plotted on the globe, specified as a three-column matrix of RGB triplets. This property defines the palette of colors MATLAB uses to create plot objects such as lines. Each row of the array is an RGB triplet. An RGB triplet is a three-element vector whose elements specify the intensities of the red, green, and blue components of a color. The intensities must be in the range [0, 1]. This table lists the default colors.

Colors	ColorOrder Matrix

	<pre>[0 0.4470 0.7410
	  0.8500 0.3250 0.0980
	  0.9290 0.6940 0.1250
	  0.4940 0.1840 0.5560
	  0.4660 0.6740 0.1880
	  0.3010 0.7450 0.9330
	  0.6350 0.0780 0.1840]</pre>

MATLAB assigns colors to objects according to their order of creation. For example, when plotting lines, the first line uses the first color, the second line uses the second color, and so on. If there are more lines than colors, then the cycle repeats.

Change the color order in either of the following ways:

- Call the `colororder` function to change the color order for all globe objects in a UI figure. The colors of existing plots in the UI figure update immediately. If you place additional globe objects into the figure, those globe objects also use the new color order. If you continue to call plotting commands, those commands also use the new colors.
- Set the `ColorOrder` property on the globe, call the `hold` function to set the globe hold state to `'on'`, and then call the desired plotting functions. This is like calling the `colororder` function, but in this case you are setting the color order for the specific globe, not the entire UI figure. Setting the `hold` state to `'on'` is necessary to ensure that subsequent plotting commands do not reset the globe to use the default color order.

`NextSeriesIndex` — SeriesIndex value for next object
whole number

This property is read-only.

`SeriesIndex` value for the next plot object added to the globe, returned as a whole number greater than or equal to `0`. This property is useful when you want to track how the objects cycle through the

colors in the color order. This property maintains a count of the objects in the globe that have a `SeriesIndex` property. MATLAB uses it to assign the value of the `SeriesIndex` property for each new object. The count starts at `1` when you create the globe, and it increases by `1` for each additional object. Thus, the count is typically $n+1$, where n is the number of objects in the globe.

NextPlot — Properties to reset
`'replace'` (default) | `'add'` | `'replacechildren'` | `'replaceall'`

Properties to reset when adding a new plot to the globe, specified as one of these values:

- `'replace'` and `'replaceall'` — Delete existing plots and reset globe properties, except `Position` and `Units`, to their default values before displaying the new plot.
- `'add'` — Add new plots to the existing globe. Do not delete existing plots or reset globe properties before displaying the new plot.
- `'replacechildren'` — Delete existing plots before displaying the new plot. Reset the `ColorOrderIndex` property to `1`, but do not reset other globe properties. The next plot added to the globe uses the first color based on the `ColorOrder` property.

Position

Position — Size and location
`[0 0 1 1]` (default) | four-element vector of form `[left bottom width height]`

Size and location, specified as a four-element vector of the form `[left bottom width height]`. By default, MATLAB measures the values in units normalized to the container. To change the units, set the `Units` property.

- The `left` and `bottom` elements define the distance from the lower left corner of the container UI figure, panel, or tab to the lower left corner of the position boundary.
- The `width` and `height` elements are the position boundary dimensions.

Units — Position units
`'normalized'` (default) | `'inches'` | `'centimeters'` | `'points'` | `'pixels'` | `'characters'`

Position units, specified as one of these values.

Units	Description
`'normalized'` (default)	Units normalized with respect to the container, which is typically the figure or a panel. The lower left corner of the container is `(0,0)` and the upper right corner is `(1,1)`.
`'inches'`	Inches.
`'centimeters'`	Centimeters.
`'characters'`	Units based on the default `uicontrol` font of the graphics root object: • The character width is the width of the letter *x*. • The character height is the distance between the baselines of two lines of text.

Units	Description
`'points'`	Typography points. One point equals 1/72 inch.
`'pixels'`	Pixels. Distances in pixels are independent of your system resolution on Windows and Macintosh systems. • On Windows systems, a pixel is 1/96th of an inch. • On Macintosh systems, a pixel is 1/72nd of an inch. • On Linux systems, the size of a pixel is determined by your system resolution.

When specifying the units as a name-value pair during object creation, specify the `Units` name-value pair before specifying name-value pairs that use those units, for example `Position`.

Interactivity

Visible — State of visibility
`'on'` (default) | `'off'`

State of visibility, specified as one of these values:

- `'on'` — Display the object.
- `'off'` — Hide the object without deleting it. You still can access the properties of an invisible object.

Parent/Child

Parent — Parent container
`Figure` object created using `uifigure` | `Panel` object | `Tab` object

Parent container, specified as a `Figure` object created using the `uifigure` function, a `Panel` object within a UI figure, or a `Tab` object within a UI figure.

Children — Children
empty `GraphicsPlaceholder` array | array of `Line` objects

Children, returned as an array of graphics objects. Use this property to view a list of the children or to reorder the children by setting the property to a permutation of itself.

You cannot add or remove children using the `Children` property. To add a child to this list, set the `Parent` property of the child graphics object to the `GeographicGlobe` object.

HandleVisibility — Visibility of object handle
`'on'` (default) | `'off'` | `'callback'`

Visibility of the object handle in the `Children` property of the parent, specified as one of these values:

- `'on'` — Object handle is always visible.
- `'off'` — Object handle is invisible at all times. This option is useful for preventing unintended changes by another function. Set the `HandleVisibility` to `'off'` to temporarily hide the handle during the execution of that function.
- `'callback'` — Object handle is visible from within callbacks or functions invoked by callbacks, but not from within functions invoked from the command line. This option blocks access to the object at the command line, but permits callback functions to access it.

If the object is not listed in the `Children` property of the parent, then functions that obtain object handles by searching the object hierarchy or querying handle properties cannot return it. Examples of such functions include the `get`, `findobj`, and `close` functions.

Hidden object handles are still valid. Set the root `ShowHiddenHandles` property to `'on'` to list all object handles regardless of their `HandleVisibility` property setting.

Identifiers

Type — Type of graphics object
`'globe'`

This property is read-only.

Type of graphics object returned as `'globe'`.

Tag — Object identifier
`''` (default) | character vector | string scalar

Object identifier, specified as a character vector or string scalar. You can specify a unique `Tag` value to serve as an identifier for an object. When you need access to the object elsewhere in your code, you can use the `findobj` function to search for the object based on the `Tag` value.

UserData — User data
`[]` (default) | array

User data, specified as any MATLAB array. For example, you can specify a scalar, vector, matrix, cell array, character array, table, or structure. Use this property to store arbitrary data on an object.

If you are working in App Designer, create public or private properties in the app to share data instead of using the `UserData` property. For more information, see "Share Data Within App Designer Apps".

Object Functions

Change Hold State and Basemap

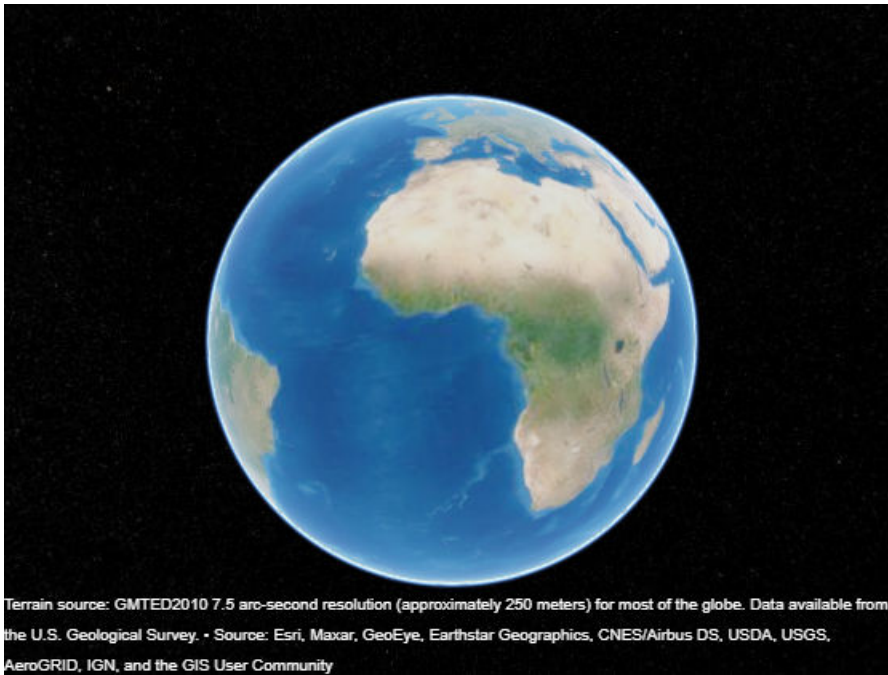| hold	These `hold` syntaxes are supported for geographic globe objects:
	• `hold(g,'on')` retains plots, terrain, and basemaps in the globe specified by g so that new plots added to the globe g do not delete existing plots or reset the terrain and basemaps. New plots use the next color based on the `ColorOrder` property of the globe. MATLAB adjusts the camera line of sight to display the full range of data.
	• `hold(g,'off')` sets the hold state to off so that new plots added to the globe clear existing plots and reset all globe properties, including terrain and basemaps. The next plot added to the globe uses the first color based on the `ColorOrder` property of the globe. This option is the default behavior.
	• `hold(g)` toggles the hold state between on and off.
geobasemap	This `geobasemap` syntax is supported for geographic globe objects:
	• `geobasemap(g,basemap)` sets the basemap for the globe specified by g. For example, `geobasemap(g,'topographic')` sets the basemap to a general-purpose map with styling to depict topographic features. For a list of basemaps, see the `Basemap` property.

Change View

campos	Set or query position of camera for geographic globe
camheight	Set or query height of camera for geographic globe
camheading	Set or query heading angle of camera for geographic globe
campitch	Set or query pitch angle of camera for geographic globe
camroll	Set or query roll angle of camera for geographic globe

Examples

Display Geographic Globe

Display a geographic globe in a figure created using the `uifigure` function.

```
uif = uifigure;
g = geoglobe(uif);
```

Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

See Also
geoglobe | geoplot3

Introduced in R2020a

geographicGrid

Package: `map.rasterref`

Geographic coordinates of raster elements

Syntax

```
[lat,lon] = geographicGrid(R)
[lat,lon] = geographicGrid(R,gridOption)
```

Description

`[lat,lon] = geographicGrid(R)` returns the geographic coordinates of raster elements as the 2-D arrays `lat` and `lon`. The coordinates of raster element `(i,j)` are `(lat(i,j),lon(i,j))`.

`[lat,lon] = geographicGrid(R,gridOption)`, where `gridOption` is `'gridvectors'`, returns `lat` and `lon` as vectors. The coordinates of raster element `(i,j)` are `(lat(i),lon(j))`. The default for `gridOption` is `'fullgrid'`, which returns `lat` and `lon` as 2-D arrays.
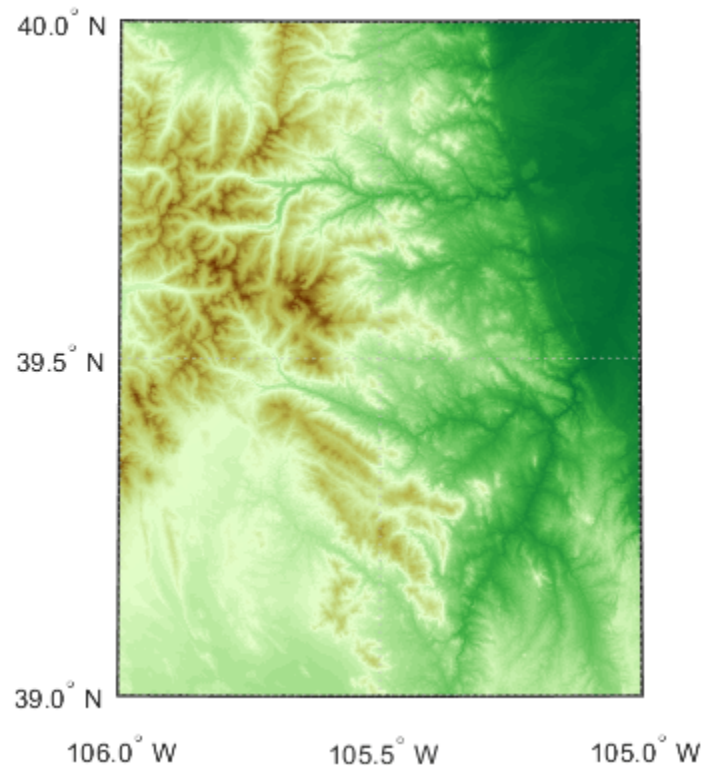
Examples

Find Coordinates of Raster Elements

Import elevation data [1] on page 1-0   for an area around South Boulder Peak in Colorado as an array and a geographic postings reference object. Find the coordinates of each element in the array.

```
[Z,R] = readgeoraster('n39_w106_3arc_v2.dt1');
[lat,lon] = geographicGrid(R);
```

Create a map with latitude and longitude limits that match the limits of the data. Display the data using an appropriate colormap.

```
usamap(R.LatitudeLimits,R.LongitudeLimits)
surfm(lat,lon,Z)
demcmap(Z)
```

[1] The elevation data used in this example is courtesy of the U.S. Geological Survey.

Get Vectors Instead of Arrays

Create a geographic cells reference object for a 3-by-4 raster with latitude values in the range [0, 30] degrees and longitude values in the range [-20, 20] degrees. Get the coordinates of the raster elements and return them as row vectors.

```
R = georefcells([0 30],[-20 20],[3 4]);
[lat,lon] = geographicGrid(R,'gridvectors')
```

lat = *1×3*

     5   15   25

lon = *1×4*

  -15   -5   5   15

If you do not specify the second argument as `'gridvectors'`, then the `geographicGrid` function returns 2-D arrays by default.

```
[latFull,lonFull] = geographicGrid(R)
```

```
latFull = 3×4

     5      5      5      5
    15     15     15     15
    25     25     25     25


lonFull = 3×4

   -15     -5      5     15
   -15     -5      5     15
   -15     -5      5     15
```

Input Arguments

R — Spatial reference
GeographicCellsReference object | GeographicPostingsReference object

Spatial reference, specified as a GeographicCellsReference or GeographicPostingsReference object.

If R is a GeographicCellsReference object, then lat and lon are cell centers. If R is a GeographicPostingsReference object, then lat and lon are posting points.

gridOption — Grid option
'fullgrid' (default) | 'gridvectors'

Grid option, specified as one of these values:

- 'fullgrid' — Return lat and lon as 2-D arrays, where each column of lat is identical and each row of lon is identical. This is the default behavior.
- 'gridvectors' — Return lat and lon as row vectors. Use this option when you want to reduce memory usage and when 2-D arrays are unnecessary, such as when plotting large data sets with the surfm function.

This table shows the difference between 'fullgrid' and 'gridvectors'.

'fullgrid'	'gridvectors'

R = georefcells([0 30],[-20 20],[3 4]);
[lat,lon] = geographicGrid(R)
 	
lat =
 	
    5   5   5   5
   15   15   15   15
   25   25   25   25
 	
lon =
 	
  -15   -5   5   15
  -15   -5   5   15
  -15   -5   5   15
	R = georefcells([0 30],[-20 20],[3 4]);
	[lat,lon] = geographicGrid(R,'gridvectors')
	
	lat =
	
	    5 15 25
	
	lon =
	
	  -15 -5 5 15

Data Types: char | string

Output Arguments

lat — Latitudes
2-D array (default) | row vector

Latitudes, returned as a 2-D array or a row vector. By default, lat is a 2-D array. To return lat as a row vector, specify gridOption as 'gridvectors'.

By default, and when gridOption is 'fullgrid', the sizes of lat and lon each equal the RasterSize property of R. When gridOption is 'gridvectors', the lengths of lat and lon equal the first and second elements of the RasterSize property of R, respectively.

lon — Longitudes
2-D array (default) | row vector

Longitudes, returned as a 2-D array or a row vector. By default, lon is a 2-D array. To return lon as a row vector, specify gridOption as 'gridvectors'.

By default, and when gridOption is 'fullgrid', the sizes of lat and lon each equal the RasterSize property of R. When gridOption is 'gridvectors', the lengths of lat and lon equal the first and second elements of the RasterSize property of R, respectively.

See Also

Functions
ndgrid | meshgrid | intrinsicToGeographic | worldGrid

Objects
GeographicCellsReference | GeographicPostingsReference

**Introduced in R2021a**

GeographicPostingsReference

Reference raster postings to geographic coordinates

Description

A geographic postings raster reference object contains information that describes the relationship between a geographic coordinate system and an intrinsic coordinate system on page 1-501 that is anchored to the columns and rows of a 2-D spatially referenced grid of point samples (or postings).

In order to reference a raster of postings to geographic coordinates by using a `GeographicPostingsReference` object, the raster must be sampled regularly in latitude and longitude, and its columns and rows must be aligned with meridians and parallels, respectively. The sample spacing from row to row need not equal the sample spacing from column to column.

This image illustrates a 9-by-11 geographic raster of postings. The postings are spaced 1 degree apart in latitude and longitude, cover longitude values in the range [40,50] in degrees, and cover latitude values in the range [20,28] in degrees. The boundary of the raster of postings is a geographic quadrangle with edges that pass through the outermost postings.
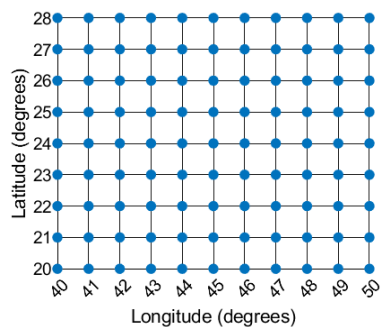
To reference a raster of cells to geographic coordinates, use a `GeographicCellsReference` object instead.

Creation

You can use these functions to create `GeographicPostingsReference` objects:

- `readgeoraster` — Import a raster data file as an array and a reference object.
- `georefpostings` — Create a geographic raster reference object by specifying the latitude and longitude limits of the raster and either the size of the raster or the spacing of the postings.
- `georasterref` — Convert a world file to a geographic raster reference object.
- `refmatToGeoRasterReference` — Convert a referencing matrix to a geographic raster reference object.

Properties

LatitudeLimits — Latitude limits of the geographic quadrangle bounding the georeferenced raster
[0.5 1.5] (default) | two-element vector

Latitude limits of the geographic quadrangle bounding the georeferenced raster, specified as a two-element vector of the form [southern_limit northern_limit].

Example: [-20 70]

Data Types: double

LongitudeLimits — Longitude limits of the geographic quadrangle bounding the georeferenced raster
[0.5 1.5] (default) | two-element vector

Longitude limits of the geographic quadrangle bounding the georeferenced raster, specified as a two-element vector of the form [western_limit eastern_limit].

Example: [-100 180]

Data Types: double

RasterSize — Number of rows and columns of the raster or image associated with the referencing object
[2 2] (default) | two-element vector of positive integers

Number of rows and columns of the raster or image associated with the referencing object, specified as a two-element vector, [m n], where m represents the number of rows and n the number of columns. For convenience, you can assign a size vector having more than two elements. This enables assignments like R.RasterSize = size(RGB), where RGB is m-by-n-by-3. In cases like this, the object stores only the first two elements of the size vector and ignores the higher (nonspatial) dimensions.

Example: [200 300]

Data Types: double

RasterInterpretation — Geometric nature of the raster
'postings' (default)

This property is read-only.

Geometric nature of the raster, specified as 'postings'.

The value 'postings' indicates that the raster comprises a grid of sample points, where rows or columns of samples run along the edge of the grid. For an m-by-n raster, points with an intrinsic x-coordinate of 1 or n and/or an intrinsic y-coordinate of 1 or m fall right on an edge (or corner) of the raster.

Data Types: char

AngleUnit — Unit of measurement used for angle-valued properties
'degree' (default)

This property is read-only.

Unit of measurement used for angle-valued properties, specified as `'degree'`.

Data Types: `char`

ColumnsStartFrom — Edge from which column indexing starts
`'south'` (default) | `'north'`

Edge from which column indexing starts, specified as `'south'` or `'north'`.

Example: `'south'`

Data Types: `char`

RowsStartFrom — Edge from which row indexing starts
`'west'` (default) | `'east'`

Edge from which row indexing starts, specified as `'west'` or `'east'`.

Example: `'east'`

Data Types: `char`

SampleSpacingInLatitude — North-south distance in latitude between adjacent samples in the raster
1 (default) | positive numeric scalar

North-south distance in latitude between adjacent samples (postings) in the raster, specified as a positive numeric scalar. The value is always positive, and is the constant throughout the raster.

Example: 2.5

Data Types: `double`

SampleSpacingInLongitude — East-west distance in longitude between adjacent samples in the raster
1 (default) | positive numeric scalar

East-west distance in longitude between adjacent samples (postings) in the raster, specified as a positive numeric scalar. The value is always positive, and is the constant throughout the raster.

Example: 2.5

Data Types: `double`

RasterExtentInLatitude — Latitude extent ("height") of the quadrangle covered by the raster
1 (default) | positive numeric scalar

This property is read-only.

Latitude extent ("height") of the quadrangle covered by the raster, specified as a positive numeric scalar.

Data Types: `double`

RasterExtentInLongitude — Longitude extent ("width") of the quadrangle covered by the raster
1 (default) | positive numeric scalar

This property is read-only.

Longitude extent ("width") of the quadrangle covered by the raster, specified as a positive numeric scalar.

Data Types: `double`

XIntrinsicLimits — Raster limits in intrinsic *x* coordinates
[1 2] (default) | two-element row vector of positive integers

This property is read-only.

Raster limits in intrinsic *x* coordinates, specified as a two-element row vector of positive integers, [xMin xMax]. For an *m*-by-*n* raster, XIntrinsicLimits equals [1 *m*], because the RasterInterpretation is 'postings'.

Data Types: `double`

YIntrinsicLimits — Raster limits in intrinsic *y* coordinates
[1 2] (default) | two-element row vector of positive integers

This property is read-only.

Raster limits in intrinsic *y* coordinates, specified as a two-element row vector of positive integers, [yMin yMax]. For an *m*-by-*n* raster, YIntrinsicLimits equals [1 *m*], because the RasterInterpretation is 'postings'.

Data Types: `double`

CoordinateSystemType — Type of coordinate system to which the image or raster is referenced
'geographic' (default) | two-element row vector of positive integers

This property is read-only.

Type of coordinate system to which the image or raster is referenced, specified as 'geographic'.

Data Types: `char`

GeographicCRS — Geographic coordinate reference system
[] (default) | `geocrs` object

Geographic coordinate reference system (CRS), specified as a `geocrs` object. A geographic CRS consists of a datum (including its ellipsoid), prime meridian, and angular unit of measurement.

Object Functions

contains	Determine if geographic or map raster contains points
geographicGrid	Geographic coordinates of raster elements
geographicToDiscrete	Transform geographic to discrete coordinates
geographicToIntrinsic	Transform geographic to intrinsic coordinates
intrinsicToGeographic	Transform intrinsic to geographic coordinates
intrinsicXToLongitude	Convert from intrinsic x to longitude coordinates
intrinsicYToLatitude	Convert from intrinsic y to latitude coordinates
latitudeToIntrinsicY	Convert from latitude to intrinsic y coordinates
longitudeToIntrinsicX	Convert from longitude to intrinsic x coordinates
sizesMatch	Determine if geographic or map raster object and image or raster are size-compatible

worldFileMatrix   Return world file parameters for transformation

Examples

Display Geographic Raster of Regularly Spaced Posting Points

Import elevation data as an array and a `GeographicPostingsReference` object by using the `readgeoraster` function. Prepare the data for plotting by specifying the output type as `'double'`.

```
[Z,R] = readgeoraster('n39_w106_3arc_v2.dt1','OutputType','double');
```
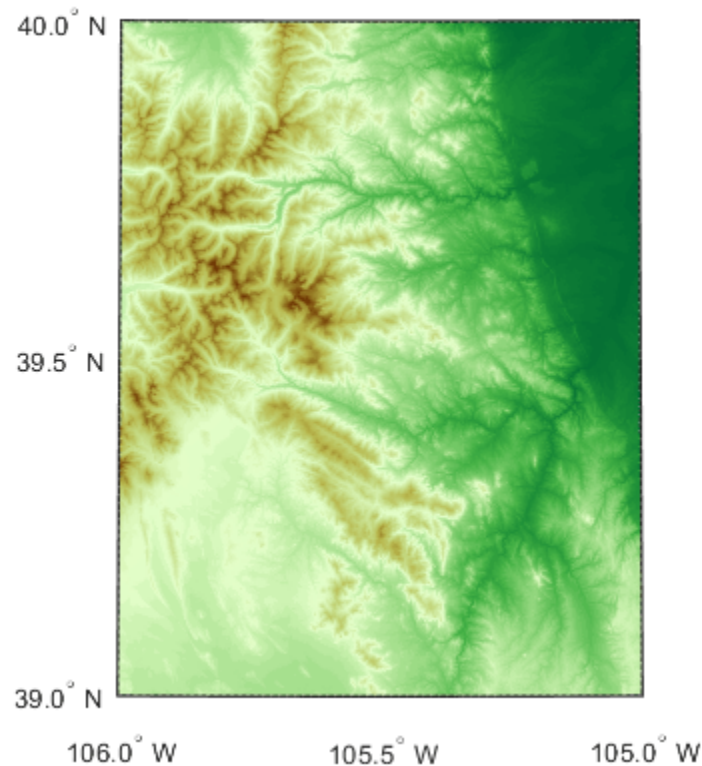
View the properties of the `GeographicPostingsReference` object.

```
R
```

```
R =
  GeographicPostingsReference with properties:

              LatitudeLimits: [39 40]
             LongitudeLimits: [-106 -105]
                  RasterSize: [1201 1201]
        RasterInterpretation: 'postings'
            ColumnsStartFrom: 'north'
               RowsStartFrom: 'west'
     SampleSpacingInLatitude: 1/1200
    SampleSpacingInLongitude: 1/1200
      RasterExtentInLatitude: 1
     RasterExtentInLongitude: 1
            XIntrinsicLimits: [1 1201]
            YIntrinsicLimits: [1 1201]
        CoordinateSystemType: 'geographic'
                GeographicCRS: [1x1 geocrs]
                   AngleUnit: 'degree'
```

Create map axes with limits that match the limits of the reference object. Display the elevation data as a surface. Apply a colormap that is appropriate for elevation data.

```
usamap(R.LatitudeLimits,R.LongitudeLimits)
geoshow(Z,R,'DisplayType','surface')
demcmap(Z)
```

More About

Intrinsic Coordinate System

A 2-D Cartesian system with its *x*-axis running parallel to the rows of a raster or image and its *y*-axis running parallel to the columns. *x* increases by 1 from column to column, and *y* increases by 1 from row to row.

Mapping Toolbox and Image Processing Toolbox use the convention for the location of the origin relative to the raster cells or sampling points such that, at a sample location or at the center of a cell, *x* has an integer value equal to the column index. Likewise, at a sample location or at the center of a cell, *y* has an integer value equal to the row index. For details, see Image Coordinate Systems (Image Processing Toolbox).

See Also

Functions
georefcells | georasterref | readgeoraster

Objects
GeographicCellsReference | MapCellsReference | MapPostingsReference

Topics
"Spatially Reference Imported Rasters"

Introduced in R2013b

geographicToDiscrete

Package: `map.rasterref`

Transform geographic to discrete coordinates

Syntax

`[I,J] = geographicToDiscrete(R,lat,lon)`

Description

`[I,J] = geographicToDiscrete(R,lat,lon)` returns the indices corresponding to geographic coordinates `lat` and `lon` in geographic raster `R`. If `R.RasterInterpretation` is:

- `'cells'`, then `I` and `J` are the row and column subscripts of the raster cells (or image pixels)
- `'postings'`, then `I` and `J` refer to the nearest sample point (posting)

Examples

Find Indices of Nearest Posting Point from Geographic Coordinates

Import elevation data [1] on page 1-0   for a region in Colorado as an array and a `GeographicPostingsReference` object. Find the indices of the posting point nearest to the latitude and longitude of South Boulder Peak, which are `39.9539` and `-105.2992` degrees, respectively.

```
[Z,R] = readgeoraster('n39_w106_3arc_v2.dt1');
[I,J] = geographicToDiscrete(R,39.9539,-105.2992)
```

`I = 56`

`J = 842`

The result means that the posting point closest to the geographic coordinates is in row 56 and column 842 of the array.

[1] The elevation data used in this example is from the U.S. Geological Survey.

Input Arguments

R — Geographic raster
`GeographicCellsReference` or `GeographicPostingsReference` object

Geographic raster, specified as a `GeographicCellsReference` or `GeographicPostingsReference` object.

lat — Latitude coordinates
numeric array

Latitude coordinates, specified as a numeric array.

Data Types: `single` | `double`

lon — Longitude coordinates
numeric array

Longitude coordinates, specified as a numeric array. `lon` is the same size as `lat`.

Data Types: `single` | `double`

Output Arguments

I — Longitude indices
array of integers

Longitude indices, returned as an array of integers. `I` is the same size as `lat`.

For an m-by-n raster, $1 \leq$ `I` $\leq m$, except for points (`lat`(k), `lat`(k)) that fall outside the bounds of the raster as defined by the function `contains`. In this case `I`(k) and `J`(k) are `NaN`.

Data Types: `double`

J — Latitude indices
array of integers

Latitude indices, returned as an array of integers. `J` is the same size as `lat`.

For an m-by-n raster, $1 \leq$ `J` $\leq n$ except for points (`lat`(k), `lat`(k)) that fall outside the bounds of the raster as defined by the function `contains`. In this case `I`(k) and `J`(k) are `NaN`.

Data Types: `double`

See Also
`contains` | `geographicToIntrinsic` | `latitudeToIntrinsicY` | `longitudeToIntrinsicX` | `worldToDiscrete`

Introduced in R2013b

geographicToIntrinsic

Package: `map.rasterref`

Transform geographic to intrinsic coordinates

Syntax

```
[xIntrinsic,yIntrinsic] = geographicToIntrinsic(R,lat,lon)
```

Description

`[xIntrinsic,yIntrinsic] = geographicToIntrinsic(R,lat,lon)` returns the intrinsic coordinates corresponding to geographic coordinates `lat` and `lon` in geographic raster R.

Examples

Find Intrinsic Coordinates from Geographic Coordinates

Find the intrinsic coordinates of a cell within a raster by specifying a raster reference object and geographic coordinates.

First, load a geographic cells reference object for the Korean peninsula. To do this, load the `korea5cR` variable from the `korea5c` MAT-file. Then, specify the geographic coordinates of Seoul.

```
load korea5c korea5cR
lat = 37.57;
lon = 126.98;
```

Find the intrinsic coordinates.

```
[xIntrinsic,yIntrinsic] = geographicToIntrinsic(korea5cR,lat,lon)

xIntrinsic = 144.2600

yIntrinsic = 91.3400
```

The result means that the geographic coordinates are in the cell in column 144 and row 91 of the raster.

You can reverse the operation by using the `intrinsicToGeographic` function.

```
[lat,lon] = intrinsicToGeographic(korea5cR,xIntrinsic,yIntrinsic)

lat = 37.5700

lon = 126.9800
```

Input Arguments

R — Geographic raster
`GeographicCellsReference` or `GeographicPostingsReference` object

Geographic raster, specified as a `GeographicCellsReference` or `GeographicPostingsReference` object.

lat — Latitude coordinates
numeric array

Latitude coordinates, specified as a numeric array. Valid values of `lat` are in the range [-90, 90] degrees or are NaN. `lat` coordinates can be outside the bounds of the raster R.

Data Types: `single` | `double`

lon — Longitude coordinates
numeric array

Longitude coordinates, specified as a numeric array. `lon` is the same size as `lat`. `lon` coordinates can be outside the bounds of the raster R.

Data Types: `single` | `double`

Output Arguments

xIntrinsic — *x*-coordinates in intrinsic coordinate system
numeric array

x-coordinates in intrinsic coordinate system, returned as a numeric array. `xIntrinsic` is the same size as `lat`.

When `lon`(*k*) is outside the bounds of raster R, `xIntrinsic`(*k*) is extrapolated in the intrinsic coordinate system.

Data Types: `double`

yIntrinsic — *y*-coordinates in intrinsic coordinate system
numeric array

y-coordinates in intrinsic coordinate system, returned as a numeric array. `yIntrinsic` is the same size as `lat`.

When `lat`(*k*) is valid and outside the bounds of raster R, `yIntrinsic`(*k*) is extrapolated in the intrinsic coordinate system.

Data Types: `double`

See Also
`intrinsicToGeographic` | `latitudeToIntrinsicY` | `longitudeToIntrinsicX` | `geographicToDiscrete` | `worldToIntrinsic`

Introduced in R2013b

geointerp

Geographic raster interpolation

Syntax

```
Vq = geointerp(V,R,latq,lonq)
Vq = geointerp( ___ ,method)
```

Description

`Vq = geointerp(V,R,latq,lonq)` interpolates the geographically referenced raster V, using bilinear interpolation. The function returns a value in `Vq` for each of the query points in arrays `latq` and `lonq`. `R` is a geographic raster reference object that specifies the location and extent of data in `V`.

`Vq = geointerp( ___ ,method)` specifies alternate interpolation methods.

Examples

Interpolate Values at Specific Latitudes and Longitudes

Load elevation raster data and a geographic cells reference object.

```
load topo60c
```

Specify the latitude and longitude values you want to interpolate. Then, interpolate the values.

```
latq = [-40 -20 20 40];
lonq = [42 54 38 62];
Vq = geointerp(topo60c,topo60cR,latq,lonq)
```

```
Vq = 1×4
10³ ×

   -2.8327   -4.3855   -0.7125    0.1700
```

Input Arguments

V — Georeferenced raster grid
numeric or logical array

Georeferenced raster grid, specified as numeric or logical array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

R — Geographic raster
`GeographicCellsReference` or `GeographicPostingsReference` object

Geographic raster, specified as a `GeographicCellsReference` or `GeographicPostingsReference` object.

To convert a referencing matrix to a geographic raster reference object, use `refmatToGeoRasterReference`.To convert a referencing vector to a geographic raster reference object, use `refvecToGeoRasterReference`.

latq — Latitude of query point coordinates
numeric array

Latitude of query point coordinates, specified as a numeric array.

Data Types: `single` | `double`

lonq — Longitude of query point coordinates,
numeric array

Longitude of query point coordinates, specified as a numeric array.

Data Types: `single` | `double`

method — Interpolation methods
`'linear'` (default) | `'nearest'` | `'cubic'` | `'spline'`

Interpolation methods, specified as one of the following values.

Method	Description
`'nearest'`	Nearest neighbor interpolation
`'linear'`	Bilinear interpolation
`'cubic'`	Bicubic interpolation
`'spline'`	Spline interpolation

Data Types: `char` | `string`

Output Arguments

Vq — Interpolated values
numeric array

Interpolated values, returned as a numeric array.

See Also
`mapinterp` | `interp2` | `griddedInterpolant`

Introduced in R2017a

geoloc2grid

Convert geolocated data array to regular data grid

Syntax

```
[Z,R] = geoloc2grid(lat,lon,A,cellsize)
```

Description

`[Z,R] = geoloc2grid(lat,lon,A,cellsize)` converts the geolocated data array `A`, given geolocation points in `lat` and `lon`, to produce a regular data grid, `Z`, and the corresponding raster reference object `R`. `cellsize` is a scalar that specifies the width and height of data cells in the regular data grid, using the same angular units as `lat` and `lon`. Data cells in `Z` falling outside the area covered by `A` are set to `NaN`.

Examples

Display Geolocated and Regular Data Grids

Load the geolocated data array `map1` and grid it to ½-degree cells.

```
load mapmtx
cellsize = 0.5;
[Z,R] = geoloc2grid(lt1,lg1,map1,cellsize);
```

Create a figure.

```
f = figure;
[cmap,clim] = demcmap(map1);
set(f,'Colormap',cmap,'Color','w')
```

Define map limits.

```
latlim = [-35 70];
lonlim = [0 100];
```

Display `map1` as a geolocated data array and `Z` as a regular data grid using subplots.

```
subplot(1,2,1)
ax = axesm('mercator','MapLatLimit',latlim,...
   'MapLonLimit',lonlim,'Grid','on',...
   'MeridianLabel','on','ParallelLabel','on');
set(ax,'Visible','off')
geoshow(lt1,lg1,map1,'DisplayType','texturemap');

subplot(1,2,2)
ax = axesm('mercator','MapLatLimit',latlim,...
   'MapLonLimit',lonlim,'Grid','on',...
   'MeridianLabel','on','ParallelLabel','on');
set(ax,'Visible','off')
geoshow(Z,R,'DisplayType','texturemap');
```
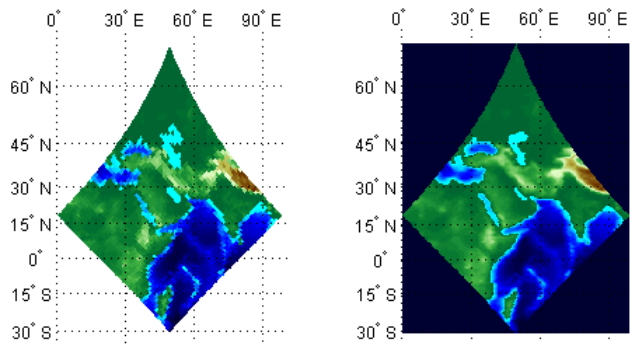
Tips

`geoloc2grid` provides an easy-to-use alternative to gridding geolocated data arrays with `imbedm`. There is no need to preallocate the output map; there are no data gaps in the output (even if `cellsize` is chosen to be very small), and the output map is smoother.

Compatibility Considerations

geoloc2grid returns raster reference objects instead of referencing vectors
Behavior changed in R2021a

Starting in R2021a, the `geoloc2grid` function returns a raster reference object instead of a referencing vector. This change is unlikely to affect your existing code because most Mapping Toolbox functions that accept referencing vectors as input also accept raster reference objects.

See Also

Functions
`imbedm`

Objects
`GeographicCellsReference` | `GeographicPostingsReference` | `MapPostingsReference` | `MapCellsReference`

Introduced before R2006a

geopeaks

Generate synthetic data set on sphere

Syntax

```
Z = geopeaks(lat,lon)
Z = geopeaks(R)
Z = geopeaks( ___ ,spheroid)
```

Description

`Z = geopeaks(lat,lon)` evaluates a "peaks-like" function at specific latitudes and longitudes on the surface of a sphere, returning the synthetic data set `Z`. The function is continuous and smooth at all points, including the poles. Reminiscent of the MATLAB `peaks` function, `geopeaks` undulates gently between values of `-10` and `8`, with about a half dozen local extrema.

`Z = geopeaks(R)` evaluates the `geopeaks` function at cell centers or sample posting points defined by a geographic raster reference object, `R`.

`Z = geopeaks( ___ ,spheroid)` evaluates the function on a specific spheroid. The choice of spheroid makes very little difference. This option exists mainly to support formal testing. If you do not specify `spheroid` and the `GeographicCRS` property of `R` is not empty, then `geopeaks` uses the spheroid contained in the `Spheroid` property of the `geocrs` object in the `GeographicCRS` property of `R`.

Examples

Generate Profile Along Meridian

Define latitude and longitude values along meridian that includes Paris, France.

```
lon = dms2degrees([2 21 3]);
lat = -90:0.5:90;
```

Generate a data set, specifying a spheroid.

```
z = geopeaks(lat, lon, wgs84Ellipsoid);
```

Display the profile.

```
figure
plot(lat,z)
xlabel('Latitude')
```
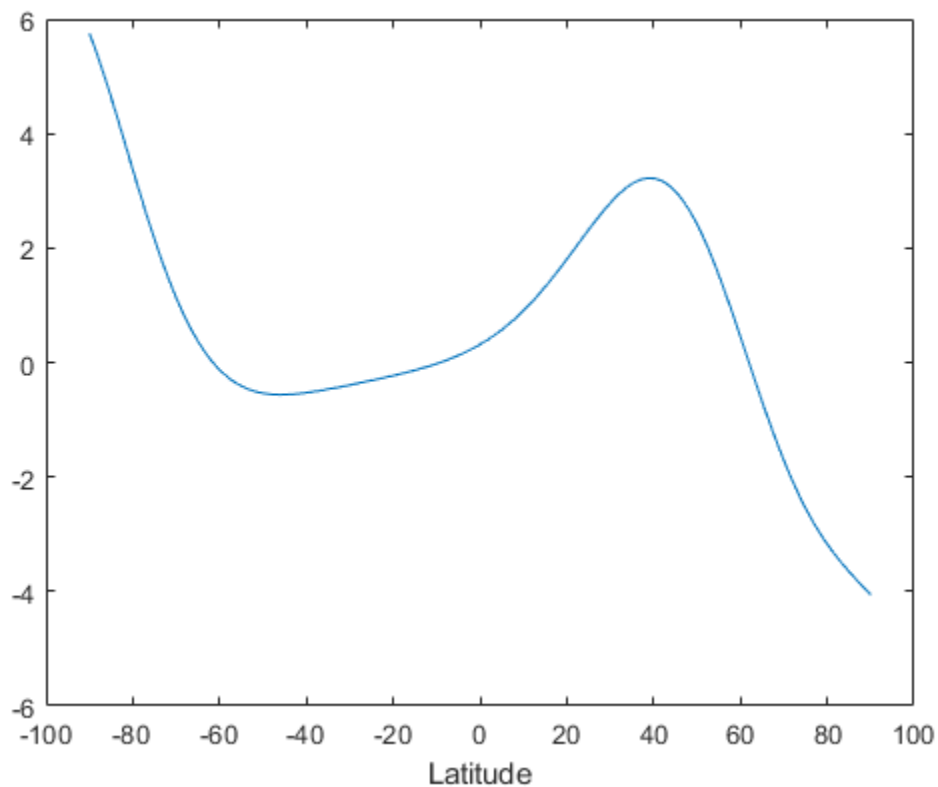
Generate Global Raster and Display Results on World Map

Create a raster reference object for a 181-by-361 grid of postings.

```
latlim = [-90 90];
lonlim = [-180 180];
sampleSpacing = 1;
R = georefpostings(latlim,lonlim,sampleSpacing,sampleSpacing)

R =
  GeographicPostingsReference with properties:

              LatitudeLimits: [-90 90]
             LongitudeLimits: [-180 180]
                  RasterSize: [181 361]
        RasterInterpretation: 'postings'
            ColumnsStartFrom: 'south'
               RowsStartFrom: 'west'
      SampleSpacingInLatitude: 1
     SampleSpacingInLongitude: 1
       RasterExtentInLatitude: 180
      RasterExtentInLongitude: 360
             XIntrinsicLimits: [1 361]
             YIntrinsicLimits: [1 181]
          CoordinateSystemType: 'geographic'
```

```
          GeographicCRS: []
             AngleUnit: 'degree'
```

Generate a data set.

```
Z = geopeaks(R);
```

Display the resultant data set on a map.

```
figure
worldmap world
geoshow(Z,R,'DisplayType','surface','CData',Z,'ZData',zeros(size(Z)))
load coastlines
geoshow(coastlat,coastlon,'Color','k')
colorbar
```
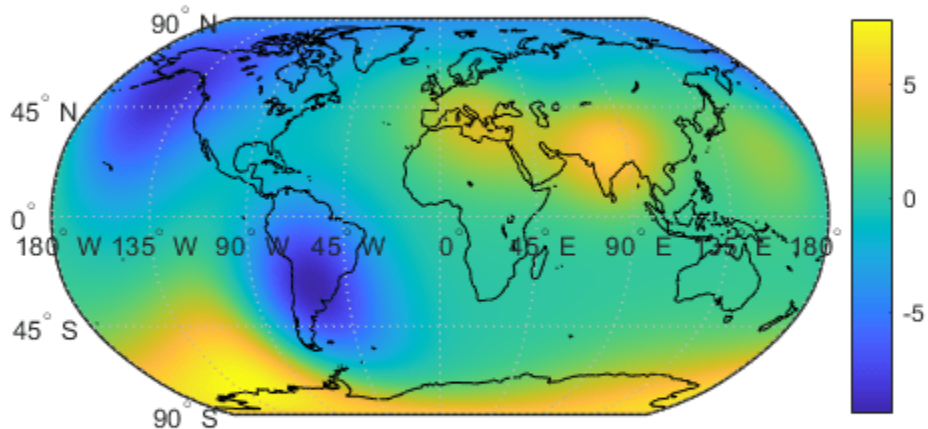

Input Arguments

lat — Geodetic latitude of one or more points
scalar, vector, or matrix

Geodetic latitude of one or more points, specified as a scalar value, vector, or matrix. Values must be in degrees.

The `lat` input argument must match the `lon` input argument in size unless either value is scalar (in which case it will expand in size to match the other), or `lat` is a column vector and `lon` is a row vector (they will expand to form a plaid latitude-longitude mesh).

Example: `lat = -90:0.5:90`

Data Types: `single` | `double`

`lon` — Geodetic longitude of one or more points
scalar, vector, or matrix

Geodetic longitude of one or more points, specified as a scalar value, vector, or matrix. Values must be in degrees.

The `lon` input argument must match the `lat` input argument in size unless either value is scalar (in which case it will expand in size to match the other), or `lon` is a column vector and `lat` is a row vector (they will expand to form a plaid latitude-longitude mesh).

Example: `lon = -180:0.5:180`

Data Types: `single` | `double`

R — Geographic raster
`GeographicCellsReference` or `GeographicPostingsReference` object

Geographic raster, specified as a `GeographicCellsReference` or `GeographicPostingsReference` object. The geographic raster stores the latitude and longitude of points.

Example: `R = georefcells([0 80], [-140 60], 0.25, 0.25)`

spheroid — Reference spheroid
`referenceEllipsoid` | `oblateSpheroid` | `referenceSphere`

Reference spheroid, specified as a `referenceEllipsoid`, `oblateSpheroid`, or `referenceSphere` object.

Example: `spheroid = referenceEllipsoid('GRS80')`

Output Arguments

Z — Synthetic data set
scalar value, vector, or matrix

Synthetic data set, returned as a scalar value, vector, or matrix of class `single` or `double`, depending on the class of the input. The function is evaluated at each element of `lat` and `lon` (following expansion as noted above), or at each cell center or posting point defined by R.

See Also
peaks

Introduced in R2015b

geoplot3

Geographic globe plot

Syntax

```
geoplot3(g,lat,lon,h)
geoplot3( ___ ,LineSpec)
geoplot3( ___ ,Name,Value)
p = geoplot3( ___ )
```

Description

geoplot3(g,lat,lon,h) plots a 3-D line in the geographic globe specified by g at the vertices specified by lat, lon, and h.

geoplot3(___ ,LineSpec) sets the line style, marker, and color.

geoplot3(___ ,Name,Value) specifies additional options for the line using one or more name-value pair arguments. Specify the options after all other input arguments. For a list of options, see Line Properties.

p = geoplot3(___) returns a Line object. This syntax is useful for controlling the properties of the line.

Examples

Plot Line over Local Region

Plot the path of a glider above a local region. First, import sample data representing the path. Get the latitude, longitude, and geoid height values.

```
trk = gpxread('sample_mixed','FeatureType','track');
lat = trk.Latitude;
lon = trk.Longitude;
h = trk.Elevation;
```

Create a geographic globe. Then, plot the path as a line. By default, the view is directly above the data. Tilt the view by holding **Ctrl** and dragging.

```
uif = uifigure;
g = geoglobe(uif);
geoplot3(g,lat,lon,h,'c')
```

Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community
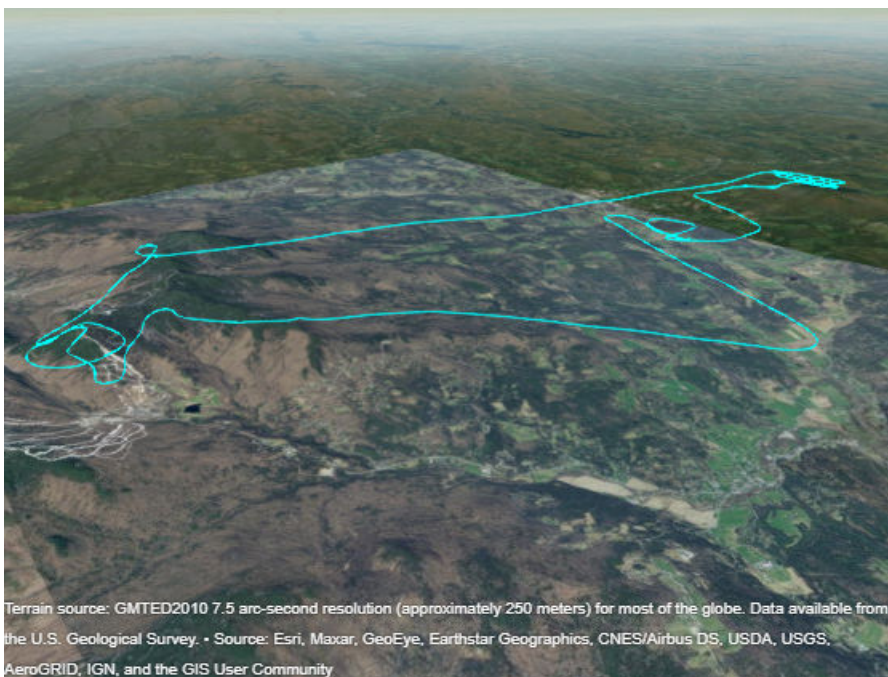
Plot Line Between Distant Points

When you plot a line between points that are far apart, the data may be obscured because the line passes through the Earth. View the entire line by inserting points between the specified data points.

For example, specify the coordinates of New York City and Paris. Then, plot a line between them. Indicate there is no height data by specifying the fourth argument of `geoplot3` as an empty array. Note that you cannot see the line because it passes through the Earth.

```
lat = [40.71 48.86];
lon = [-74.01 2.35];
uif = uifigure;
g = geoglobe(uif);
geoplot3(g,lat,lon,[],'y','LineWidth',2)
```

To see the line, insert points along a great circle using the `interpm` function. Then, plot the line again. Note that the line is visible.

```
[latI,lonI] = interpm(lat,lon,0.1,'gc');
geoplot3(g,latI,lonI,[],'y','LineWidth',2)
```
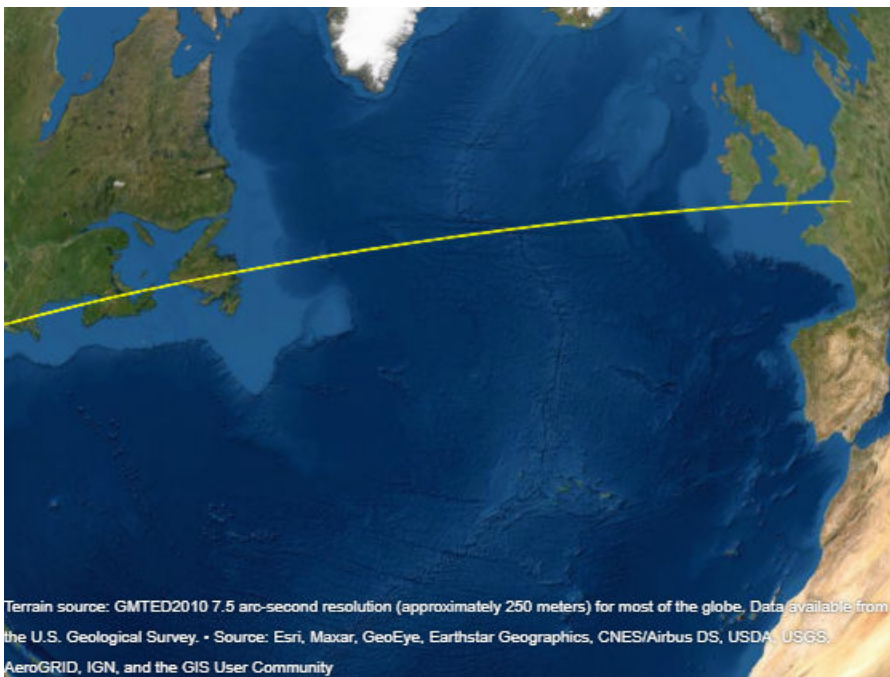
Plot Line over Global Region

When you plot a line over a large region such as a state or country, part of the line may be obscured because it passes through terrain. View the entire line by removing the terrain data from the globe.
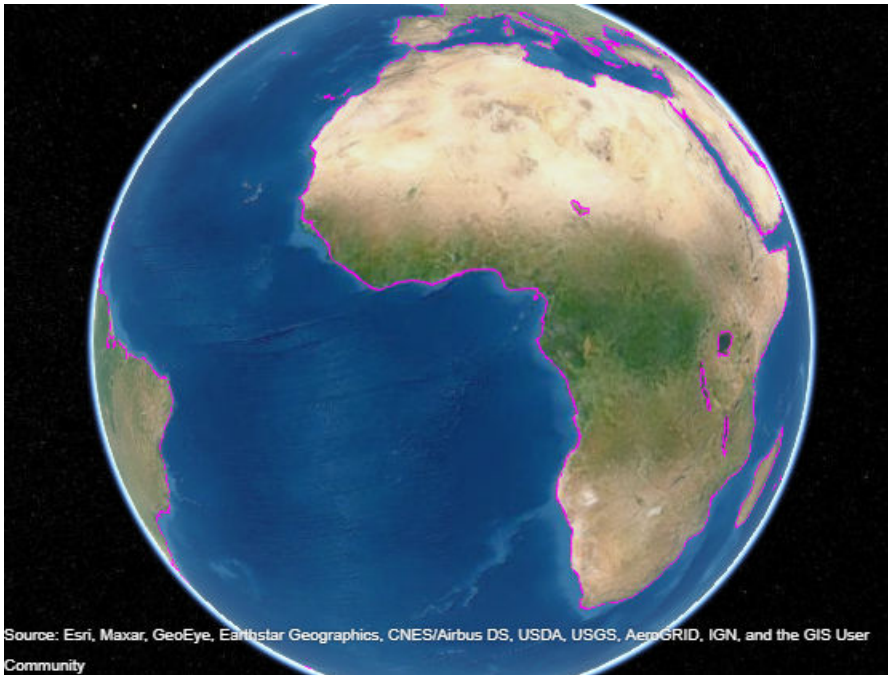
For example, import sample coastline data and plot it on a geographic globe. By default, the globe includes terrain data derived from the GMTED2010 model. Note that the line appears broken.

```
load coastlines
uif = uifigure;
g = geoglobe(uif);
p = geoplot3(g,coastlat,coastlon,[],'m');
```


To see the line, set the `Terrain` property of the globe to `'none'`. Indicate the plotted data sits on the WGS84 reference ellipsoid by setting the `HeightReference` property of the line to `'ellipsoid'`. Note that the line is visible over the basemap.

```
g.Terrain = 'none';
p.HeightReference = 'ellipsoid';
```

Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Plot Circle Markers Instead of Line

Import sample data representing the path of a glider. Get the latitude, longitude, and geoid height values.

```
trk = gpxread('sample_mixed','FeatureType','track');
lat = trk.Latitude;
lon = trk.Longitude;
h = trk.Elevation;
```

Create a geographic globe. Then, plot the data using circle markers. Plot a marker at every 25th data point by setting the `MarkerIndices` property.

```
uif = uifigure;
g = geoglobe(uif);
mskip = 1:25:length(lat);
geoplot3(g,lat,lon,h,'co','MarkerIndices',mskip)
```

Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community
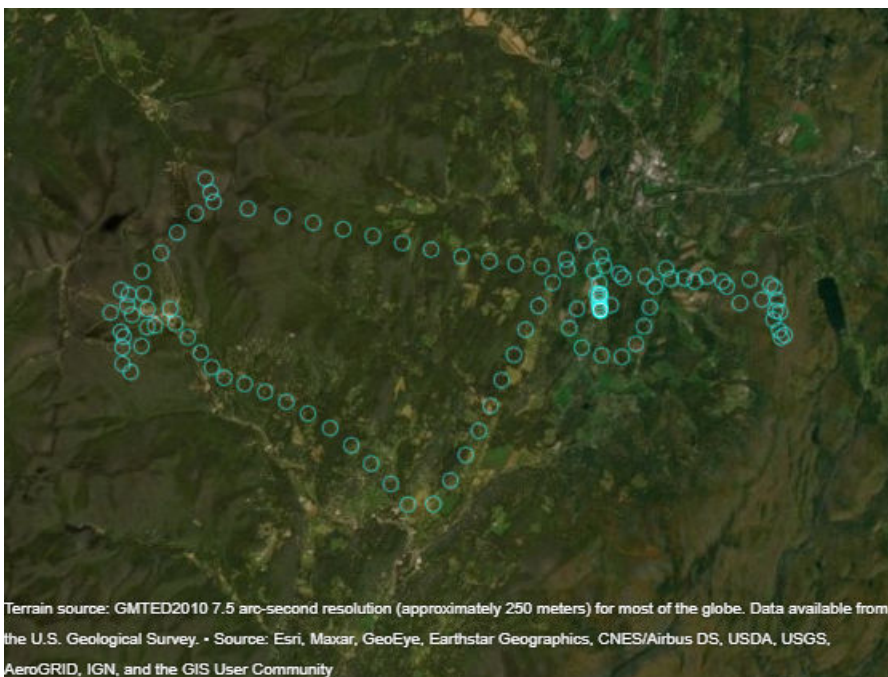
Plot Data with Height Referenced to Terrain

Plot a line from the surface of Gross Reservoir to a point above South Boulder Peak.

Specify the latitude, longitude, and height of the two endpoints. Specify the heights relative to the terrain, so that 0 represents ground level and not sea level.

```
lat = [39.95384 39.95];
lon = [-105.29916 -105.3608];
hTerrain = [10 0];
```

Plot the line on a geographic globe. Indicate that height values are referenced to the terrain using the HeightReference property. By default, the view is directly above the data. Tilt the view by holding **Ctrl** and dragging.

```
uif = uifigure;
g = geoglobe(uif);
geoplot3(g,lat,lon,hTerrain,'y','HeightReference','terrain', ...
    'LineWidth',3)
```

Terrain source: GMTED2010 7.5 arc-second resolution (approximately 250 meters) for most of the globe. Data available from the U.S. Geological Survey. • Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community
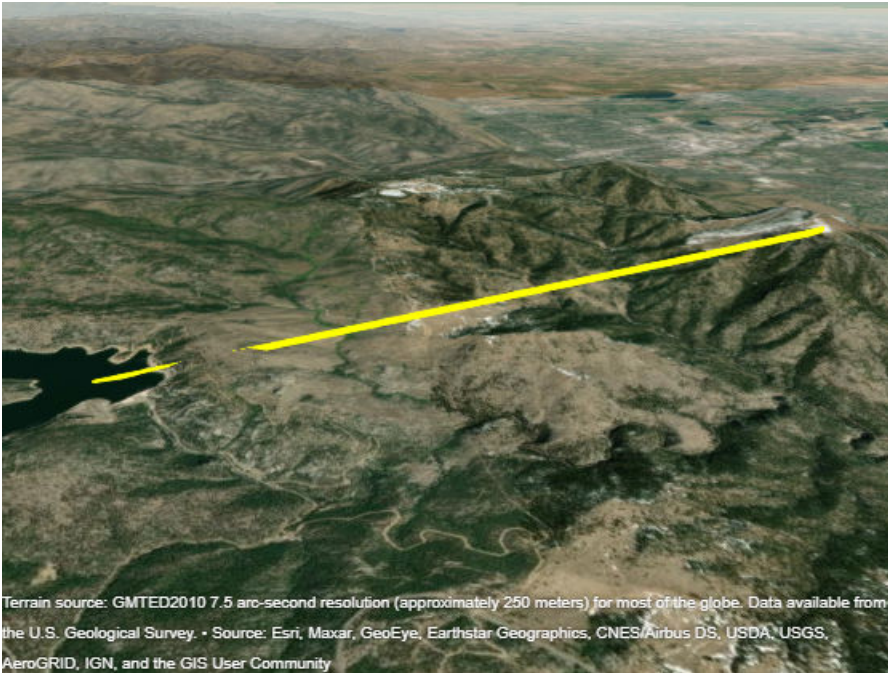
Input Arguments

g — Geographic globe
GeographicGlobe object

Geographic globe, specified as a `GeographicGlobe` object.[6]

lat — Geodetic latitudes
vector

Geodetic latitudes in degrees, specified as a vector.

`lat` and `lon` must be the same size.

Data Types: single | double

lon — Geodetic longitudes
vector

Geodetic longitudes in degrees, specified as a vector.

`lat` and `lon` must be the same size.

Data Types: single | double

h — Heights
vector

Heights in meters, specified as a vector. By default, height values are referenced to the geoid, or mean sea level.

---

6. Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

Reference height values to the WGS84 reference ellipsoid by setting the `HeightReference` property of the line to `'ellipsoid'`. Reference height values to the terrain, or ground, by setting the `HeightReference` property to `'terrain'`.

h must be either a scalar or a vector of the same size as `lat` and `lon`. If h is a scalar, then every point is plotted at the same height.

Data Types: `single` | `double`

LineSpec — Line style, marker, and color
character vector | string scalar

Line style, marker, and color, specified as a character vector or string containing symbols. The symbols can appear in any order. You do not need to specify all three characteristics (line style, marker, and color). For example, if you omit the line style and specify the marker, then the plot shows only the marker and no line.

Example: `'-or'` is a red solid line with circle markers

Line Style and Marker	Description
-	Solid line (default)
o	Circle marker

Color	Description
y	yellow
m	magenta
c	cyan
r	red
g	green
b	blue
w	white
k	black

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `geoplot3(g,1:10,1:10,1:10,'Color','r')` changes the color of the line

> **Note** The properties listed here are only a subset. For a full list, see Line Properties.

HeightReference — Height reference
`'geoid'` (default) | `'terrain'` | `'ellipsoid'`

Height reference, specified as one of these values:

- `'geoid'` – Height values are relative to the geoid (mean sea level).

- `'terrain'` – Height values are relative to the ground.
- `'ellipsoid'` – Height values are relative to the WGS84 reference ellipsoid.

For more information about terrain, geoid, and ellipsoid height, see "Find Ellipsoidal Height from Orthometric and Geoid Height".
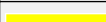
Color — Line color
[0 0.4470 0.7410] (default) | RGB triplet | hexadecimal color code | `'r'` | `'g'` | `'b'` | ...

Line color, specified as an RGB triplet, a hexadecimal color code, a color name, or a short name.
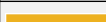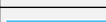
RGB triplets and hexadecimal color codes are useful for specifying custom colors.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1]; for example, [0.4 0.6 0.7].
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (#) followed by three or six hexadecimal digits, which can range from 0 to F. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`'red'`	`'r'`	[1 0 0]	`'#FF0000'`	
`'green'`	`'g'`	[0 1 0]	`'#00FF00'`	
`'blue'`	`'b'`	[0 0 1]	`'#0000FF'`	
`'cyan'`	`'c'`	[0 1 1]	`'#00FFFF'`	
`'magenta'`	`'m'`	[1 0 1]	`'#FF00FF'`	
`'yellow'`	`'y'`	[1 1 0]	`'#FFFF00'`	
`'black'`	`'k'`	[0 0 0]	`'#000000'`	
`'white'`	`'w'`	[1 1 1]	`'#FFFFFF'`	

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

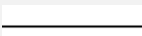| RGB Triplet	Hexadecimal Color Code	Appearance
[0 0.4470 0.7410]	`'#0072BD'`	
[0.8500 0.3250 0.0980]	`'#D95319'`	
[0.9290 0.6940 0.1250]	`'#EDB120'`	
[0.4940 0.1840 0.5560]	`'#7E2F8E'`	
[0.4660 0.6740 0.1880]	`'#77AC30'`	
[0.3010 0.7450 0.9330]	`'#4DBEEE'`	
[0.6350 0.0780 0.1840]	`'#A2142F'`	

Example: `'blue'`

Example: [0 0 1]

Example: '#0000FF'

LineStyle — Line style
'-' (default) | 'none'

Line style, specified as one of these options:

Line Style	Description	Resulting Line
'-'	Solid line (default)	_____
'none'	No line	No line

Marker — Marker symbol
'none' (default) | 'o'

Marker symbol, specified as 'none' or 'o'. By default, the line does not display markers. Specify 'o' to display circle markers at each data point or vertex.

Markers do not tilt or rotate as you navigate the globe.

Limitations

Unlike most Line objects, lines created using geoplot3 cannot have their parent changed to any object except a geographic globe.

See Also

geoglobe | Line Properties

Introduced in R2020a

geopoint

Geographic point vector

Description

A geopoint vector is a container object that holds geographic point coordinates and attributes. The points are coupled, such that the size of the latitude and longitude coordinate arrays are always equal and match the size of any dynamically added attribute arrays. Each entry of a coordinate pair and associated attributes, if any, represents a discrete element in the geopoint vector.

To create a geographic point shape for use with a geospatial table, create a `geopointshape` object instead.

Creation

Syntax

```
p = geopoint()
p = geopoint(latitude,longitude)
p = geopoint(latitude,longitude,Name,Value)
p = geopoint(structArray)
p = geopoint(latitude,longitude,structArray)
```

Description

`p = geopoint()` constructs an empty geopoint vector with these default property settings:

```
p =

 0x1 geopoint vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
 Feature properties:
     Latitude: []
    Longitude: []
```

`p = geopoint(latitude,longitude)` sets the `Latitude` and `Longitude` properties of geopoint vector `p`

`p = geopoint(latitude,longitude,Name,Value)` sets the `Latitude` and `Longitude` properties, then adds dynamic properties to the geopoint vector using `Name,Value` argument pairs. You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

`p = geopoint(structArray)` constructs a new `geopoint` vector from the fields of the structure, `structArray`.

- If `structArray` contains the field `Lat`, and does not contain a field `Latitude`, then the `Lat` values are assigned to the `Latitude` property. Similar behavior occurs when `structArray` contains the field `Lon`, and does not contain the field `Longitude`.

- If `structArray` contains both `Lat` and `Latitude` fields, then both field values are assigned to `p`. Similar behavior occurs for `Lon` and `Longitude` fields when both are present in `structArray`

- Other fields of `structArray` are assigned to `p` and become dynamic properties. Field values in `structArray` that are not numeric data types, string scalars, string arrays, character vectors, or cell arrays of numeric data types or character vectors are ignored.

`p = geopoint(latitude,longitude,structArray)` sets the `Latitude` and `Longitude` properties, and sets dynamic properties from the field values of the structure, `structArray`.

- If `structArray` contains the fields `Lat`, `Latitude`, `Lon`, or `Longitude`, then those field values are ignored.

Properties

Each element in a geopoint vector is considered a feature. For more about the property types in `geopoint`, see "Collection Properties" on page 1-542 and "Feature Properties" on page 1-542.

Dynamic properties are new features that are added to a geopoint vector and that apply to each individual feature in the geopoint vector. You can attach new dynamic Feature properties to the object during construction with a `Name,Value` pair or after construction using dot (`.`) notation. This is similar to adding dynamic fields to a structure. For an example of adding dynamic Feature properties, see "Construct Geopoint Vector Using Name-Value Pairs" on page 1-531.

Geometry — Type of geometry
`'point'`

Type of geometry, specified as `'point'`. For `geopoint`, `Geometry` is always `'point'`.

Data Types: `char` | `string`

Latitude — Latitude coordinates
numeric row or column vector

Latitude coordinates, specified as a numeric row or column vector.

Data Types: `double` | `single`

Longitude — Longitude coordinates
numeric row or column vector

Longitude coordinates, specified as a numeric row or column vector.

Data Types: `double` | `single`

Metadata — Information for the entire set of geopoint vector elements
scalar structure

Information for the entire set of geopoint vector elements, specified as a scalar structure. You can add any data type to the structure.

- If `Metadata` is provided as a dynamic property `Name` in the constructor, and the corresponding `Value` is a scalar structure, then `Value` is copied to the `Metadata` property. Otherwise, an error is issued.

- If a `Metadata` field is provided by `structArray`, and both `Metadata` and `structArray` are scalar structures, then the `Metadata` field value is copied to the `Metadata` property value. If `structArray` is a scalar but the `Metadata` field is not a structure, then an error is issued. If `structArray` is not scalar, then the `Metadata` field is ignored.

Data Types: `struct`

Object Functions

append	Append features to geographic or planar vector
cat	Concatenate geographic or planar vector
disp	Display geographic or planar vector
fieldnames	Return dynamic property names of geographic or planar vector
isempty	Determine if geographic or planar vector is empty
isfield	Determine if dynamic property exists in geographic or planar vector
isprop	Determine if property exists in geographic or planar vector
length	Return number of elements in geographic or planar vector
properties	Return property names of geographic or planar vector
rmfield	Remove dynamic property from geographic or planar vector
rmprop	Remove property from geographic or planar vector
size	Return size of geographic or planar vector
struct	Convert geographic or planar vector to scalar structure
vertcat	Vertically concatenate geographic or planar vectors

Examples

Construct Geopoint Vector and Display It

This example shows how to create a `geopoint` vector, specifying latitude and longitude coordinates, and display it.

Create Geopoint Containing Single Point and Display It

Create a `geopoint` vector using the latitude and longitude of Paris, France, and display it. When using a `geopoint` vector, the geometry of the constructed object is always `'point'`.

```
lat = 48.8566;
lon = 2.3522;
p = geopoint(lat,lon)

p =
 1×1 geopoint vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1×1 struct]
 Feature properties:
     Latitude: 48.8566
    Longitude: 2.3522
```

Display the point in a plot. You can pass a `geopoint` vector directly to the `geoshow` command. `geoshow` can read the latitude and longitude values from the `geopoint` vector and also reads the geometry type.
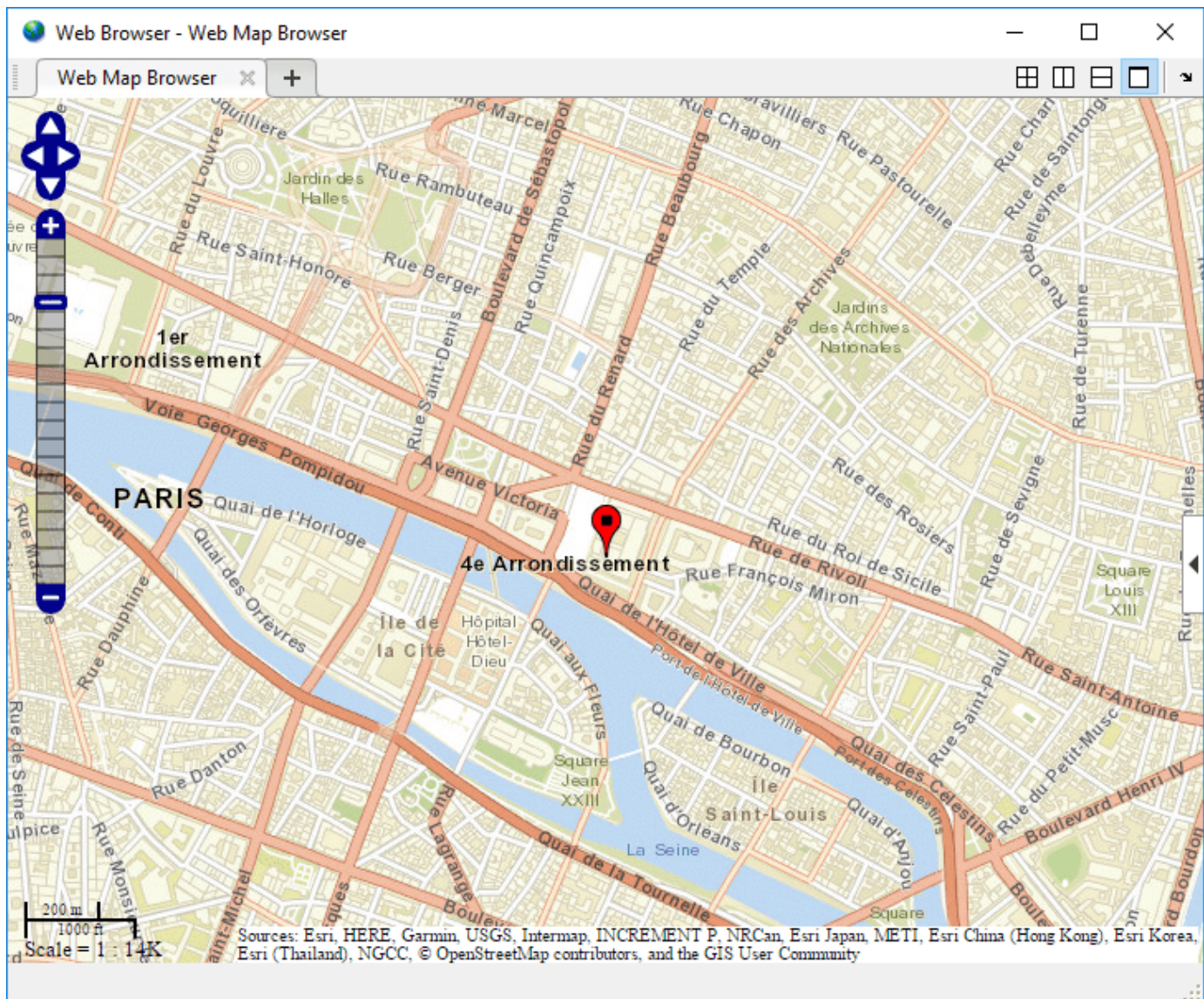
```
geoshow(p)
```
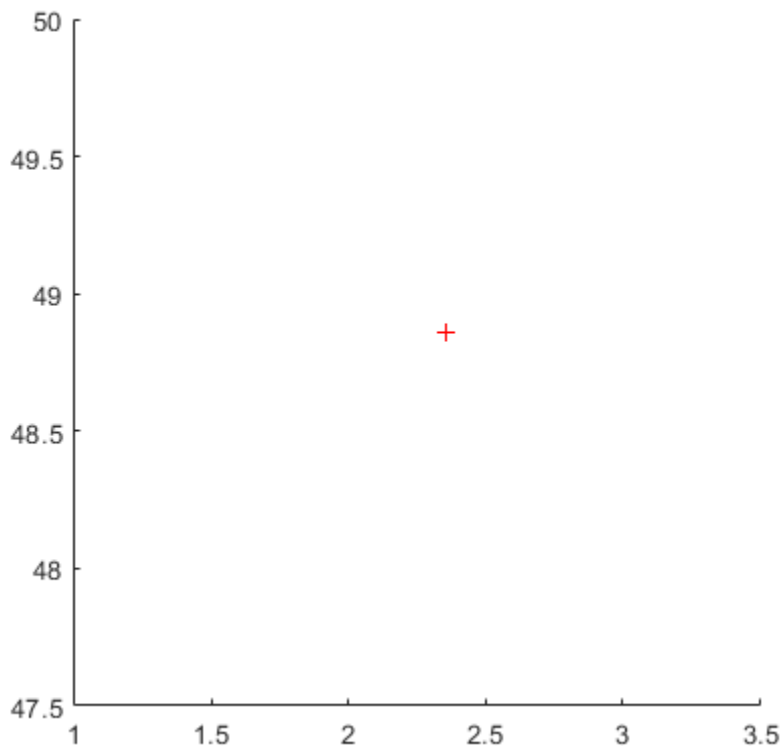

Display Point on a Map

To display your point with more context, plot your point over a map using the web map display function `wmmarker`. You can pass a `geopoint` vector directly to the `wmmarker` command.

```
wmmarker(p)
```

The `wmmarker` function opens a web map and displays the point on the map.

Construct Geopoint Vector from File

Import data from a text file with the latitudes and longitudes of some European capitals. The latitude coordinates are in the first column and the longitude coordinates are in the second column. The coordinates are separated by single space character.

```
data = importdata('european_capitals.txt');
```

Create a `geopoint` vector containing the latitude and longitude data.

```
p = geopoint(data(:,1),data(:,2))
```

```
p =
 13×1 geopoint vector with properties:
```

```
Collection properties:
    Geometry: 'point'
    Metadata: [1×1 struct]
Feature properties:
    Latitude: [48.8566 51.5074 40.4168 41.9028 52.5200 52.3680 52.2297 47.4979 44.4268 50.0755
    Longitude: [2.3522 -0.1278 -3.7038 12.4964 13.4050 4.9036 21.0122 19.0402 26.1025 14.4378 17
```
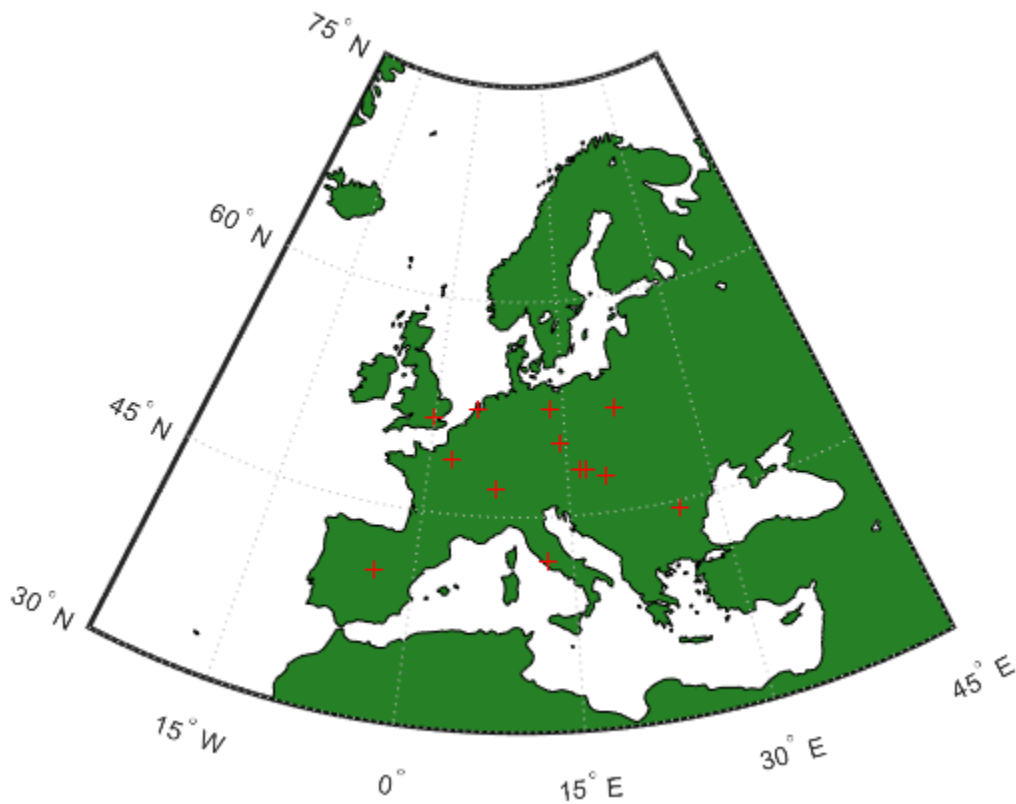
Plot the points on a map showing the landmass of Europe.

```
worldmap europe
geoshow('landareas.shp','FaceColor', [0.15 0.5 0.15])
geoshow(p)
```


Alternatively, you can also plot these points over a map using the `geoplot` function. This example includes a line specification parameter to specify a plus sign marker and the color red. The example also increases the line width for better visibility of the markers.

```
figure
geoplot(p.Latitude,p.Longitude,'+r','LineWidth',2)
```
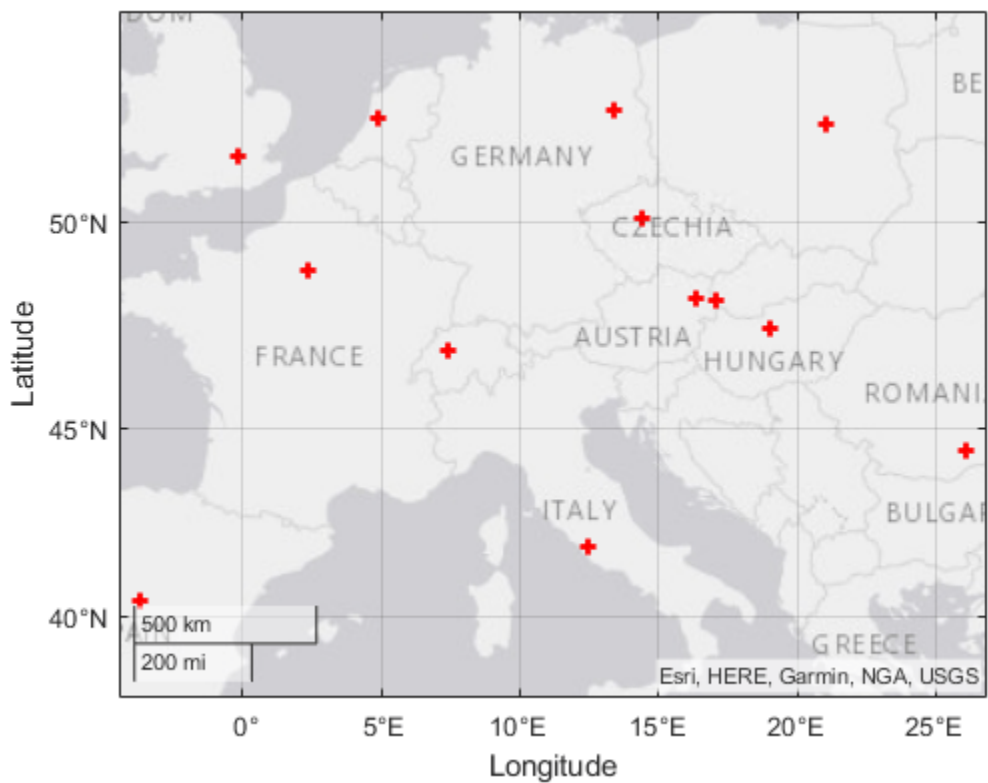
Construct Geopoint Vector Using Name-Value Pairs

Create a geopoint vector, specifying `Latitude`, `Longitude`, and `Temperature`, where `Temperature` is part of a Name-Value pair.

```
point = geopoint(42, -72, 'Temperature', 89)
```

```
point =

 1x1 geopoint vector with properties:

 Collection properties:
       Geometry: 'point'
       Metadata: [1x1 struct]
 Feature properties:
       Latitude: 42
      Longitude: -72
    Temperature: 89
```

Construct a geopoint object specifying names.

```
p = geopoint([51.519 48.871], [-.13 2.4131],...
    'Name', {"London", "Paris"})
```

```
p =
```

```
2x1 geopoint vector with properties:

Collection properties:
    Geometry: 'point'
    Metadata: [1x1 struct]
Feature properties:
    Latitude: [51.5190 48.8710]
    Longitude: [-0.1300 2.4131]
        Name: {'London' 'Paris'}
```

Construct Geopoint Vector From a Structure Array

Read shape data into a geostruct (a structure array containing `Lat` and `Lon` fields).

```
S = shaperead('worldcities.shp', 'UseGeoCoords', true)

S =

318x1 struct array with fields:

    Geometry
    Lon
    Lat
    Name
```

Create a geopoint vector specifying the geostruct.

```
p = geopoint(S)

p =

 318x1 geopoint vector with properties:

 Collection properties:
    Geometry: 'point'
    Metadata: [1x1 struct]
 Feature properties:
    Latitude: [1x318 double]
    Longitude: [1x318 double]
        Name: {1x318 cell}
```

Add a `Filename` field to the `Metadata` structure. The `Metadata` property pertains to all elements of a geopoint vector.

```
p.Metadata.Filename = 'worldcities.shp';
m = p.Metadata

ans =

    Filename: 'worldcities.shp'
```

Construct a Geopoint Vector Using Numeric Arrays and a Structure Array

Create a structure array.

```
[structArray, A] = shaperead('worldcities.shp', 'UseGeoCoords', true)

structArray =

318x1 struct array with fields:
    Geometry
    Lon
    Lat

A =

318x1 struct array with fields:
    Name
```

Use the numeric arrays and the structure containing the list of names to construct a geopoint vector.

```
p = geopoint([structArray.Lat], [structArray.Lon], A)

p =

 318x1 geopoint vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
 Feature properties:
     Latitude: [1x318 double]
    Longitude: [1x318 double]
         Name: {1x318 cell}
```

Add Coordinate and Dynamic Properties

Generate an empty geopoint vector using the default constructor, then populate the geopoint vector using dot notation with properties from data fields in structure structArray.

```
structArray = shaperead('worldcities.shp', 'UseGeoCoords', true);
p = geopoint();
p.Latitude  = [structArray.Lat];
p.Longitude = [structArray.Lon];
p.Name = structArray.Name;
p

p =

 318x1 geopoint vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
 Feature properties:
     Latitude: [1x318 double]
    Longitude: [1x318 double]
         Name: {1x318 cell}
```

Add New Values to Existing geopoint Vector

This example shows how to add new values to an existing geopoint vector. The example appends data about Paderborn Germany to the geopoint vector of data about world cities.

Read world cities data using the `shaperead` command. `shaperead` returns a structure array.

```
structArray = shaperead('worldcities.shp', 'UseGeoCoords', true);
```

Create a geopoint vector from the structure array. Display the last of the 318 elements in the vector.

```
p = geopoint(structArray);
p(end)

ans =
 1x1 geopoint vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
 Feature properties:
     Latitude: 34.8519
    Longitude: 113.8061
         Name: 'Zhengzhou'
```

Add the Paderborn data to the end of the geopoint vector. Display the last of the existing elements and the new element.

```
lat = 51.715254;
lon = 8.75213;
p = append(p, lat, lon, 'Name', 'Paderborn');
p(end-1:end)

ans =
 2x1 geopoint vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
 Feature properties:
     Latitude: [34.8519 51.7153]
    Longitude: [113.8061 8.7521]
         Name: {'Zhengzhou'  'Paderborn'}
```

Another way to add a point at the end of a vector is to use linear indexing. For example, add data about Arlington, Massachusetts to the end of the world cities vector. Notice how, after the initial assignment statement appends a value to the Latitude property vector, using `end+1`, all other property vectors automatically expand by one element. Display the last of the existing elements and the new element.

```
p(end+1).Latitude = 42.417060

p =
 320x1 geopoint vector with properties:

 Collection properties:
```

```
     Geometry: 'point'
     Metadata: [1x1 struct]
 Feature properties:
     Latitude: [5.2985 24.6525 5.6106 37.0613 9.0235 -34.6645 12.8767 ... ]
    Longitude: [-3.9509 54.7589 -0.2121 35.3894 38.7575 138.8528 44.5408 ... ]
         Name: {1x320 cell}


p(end).Longitude = -71.170200;
p(end).Name = 'Arlington';
p(end-1:end)

ans =
 2x1 geopoint vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
 Feature properties:
     Latitude: [51.7153 42.4171]
    Longitude: [8.7521 -71.1702]
         Name: {'Paderborn'  'Arlington'}
```

Manipulate a Geopoint Vector

Construct a geopoint vector containing two features and then add two dynamic properties.

```
lat = [51.519 48.871];
lon = [-.13 2.4131];
p = geopoint(lat, lon);

p.Name = {'London', 'Paris'};   % Add character feature dynamic property
p.ID = [1 2]                    % Add numeric feature dynamic property

p =
 2x1 geopoint vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
 Feature properties:
     Latitude: [51.5190 48.8710]
    Longitude: [-0.1300 2.4131]
         Name: {'London' 'Paris'}
           ID: [1 2]
```

Add the coordinates for a third feature.

```
p(3).Latitude = 45.472;
p(3).Longitude = 9.184

p =
 3x1 geopoint vector with properties:
```

```
Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
 Feature properties:
     Latitude: [51.5190 48.8710 45.4720]
    Longitude: [-0.1300 2.4131 9.1840]
         Name: {'London' 'Paris' ''}
           ID: [1 2 0]
```

Note that lengths of all feature properties are synchronized with default values.

Set the values for the `ID` feature dynamic property with more values than contained in `Latitude` or `Longitude`.

```
p.ID = 1:4

p =

 4x1 geopoint vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
 Feature properties:
     Latitude: [51.5190 48.8710 45.4720 0]
    Longitude: [-0.1300 2.4131 9.1840 0]
         Name: {'London' 'Paris' '' ''}
           ID: [1 2 3 4]
```

Note that all feature properties are expanded to match in size.

Set the values for the ID feature dynamic property with fewer values than contained in the `Latitude` or `Longitude` properties.

```
p.ID = 1:2

p =

 4x1 geopoint vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
 Feature properties:
     Latitude: [51.5190 48.8710 45.4720 0]
    Longitude: [-0.1300 2.4131 9.1840 0]
         Name: {'London' 'Paris' '' ''}
           ID: [1 2 0 0]
```

The `ID` property values expand to match the length of the `Latitude` and `Longitude` property values.

Set the value of either coordinate property (`Latitude` or `Longitude`) with fewer values.

```
 p.Latitude = [51.519 48.871]

p =
```

```
 2x1 geopoint vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
 Feature properties:
     Latitude: [51.5190 48.8710]
    Longitude: [-0.1300 2.4131]
         Name: {'London' 'Paris'}
           ID: [1 2]
```

All properties shrink to match in size.

Remove the ID property by setting its value to [ ].

```
p.ID = []
```

```
p =

 2x1 geopoint vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
 Feature properties:
     Latitude: [51.5190 48.8710]
    Longitude: [-0.1300 2.4131]
         Name: {'London' 'Paris'}
```

Remove all dynamic properties and set the object to empty by setting a coordinate property value to [ ].

```
p.Latitude = []
```

```
p =

 0x1 geopoint vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
 Feature properties:
     Latitude: []
    Longitude: []
```

Sort Dynamic Properties and Extract Subsets

Read data from shapefile. Initially the field names of the class are in random order.

```
structArray = shaperead('tsunamis.shp', 'UseGeoCoords', true);
% Field names in random order
p = geopoint(structArray)
```

```
p =

 162x1 geopoint vector with properties:
```

```
    Collection properties:
          Geometry: 'point'
          Metadata: [1x1 struct]
    Feature properties:
          Latitude: [1x162 double]
         Longitude: [1x162 double]
              Year: [1x162 double]
             Month: [1x162 double]
               Day: [1x162 double]
              Hour: [1x162 double]
            Minute: [1x162 double]
            Second: [1x162 double]
          Val_Code: [1x162 double]
          Validity: {1x162 cell}
        Cause_Code: [1x162 double]
             Cause: {1x162 cell}
            Eq_Mag: [1x162 double]
           Country: {1x162 cell}
          Location: {1x162 cell}
        Max_Height: [1x162 double]
          Iida_Mag: [1x162 double]
         Intensity: [1x162 double]
        Num_Deaths: [1x162 double]
       Desc_Deaths: [1x162 double
```

Using the method `fieldnames` and typical MATLAB vector notation, the field names in the geopoint vector are alphabetically sorted.

```
p = p(:, sort(fieldnames(p)))

p =

 162x1 geopoint vector with properties:

 Collection properties:
        Geometry: 'point'
        Metadata: [1x1 struct]
 Feature properties:
        Latitude: [1x162 double]
       Longitude: [1x162 double]
           Cause: {1x162 cell}
      Cause_Code: [1x162 double]
         Country: {1x162 cell}
             Day: [1x162 double]
     Desc_Deaths: [1x162 double]
          Eq_Mag: [1x162 double]
            Hour: [1x162 double]
        Iida_Mag: [1x162 double]
       Intensity: [1x162 double]
        Location: {1x162 cell}
      Max_Height: [1x162 double]
          Minute: [1x162 double]
           Month: [1x162 double]
      Num_Deaths: [1x162 double]
          Second: [1x162 double]
        Val_Code: [1x162 double]
        Validity: {1x162 cell}
            Year: [1x162 double]
```

Using typical MATLAB vector notation, extract a subset of data from the base geopoint vector into a geopoint vector albeit smaller in size.

```
subp = p(20:40,{'Location','Country','Year'}) % get subset of data

subp =

 21x1 geopoint vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
 Feature properties:
     Latitude: [1x21 double]
    Longitude: [1x21 double]
     Location: {1x21 cell}
      Country: {1x21 cell}
         Year: [1x21 double]
```

Note that the coordinate properties `Latitude` and `Longitude`, and the Collection properties, are retained in this subset of geopoint vectors.

Set, Get, and Remove Dynamic Property Values

To set property values, use the `()` operator, or assign array values to corresponding fields, or use dot `'.'` notation (`object.property`) to assign new property values.

```
pts = geopoint();
pts.Latitude = [42 44 45];
pts.Longitude = [-72 -72.1 -71];
pts

pts =

 3x1 geopoint vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
 Feature properties:
     Latitude: [42 44 45]
    Longitude: [-72 -72.1000 -71]
```

Use `()` to assign values to fields.

```
pts(3).Latitude = 45.5;
pts

pts =

 3x1 geopoint vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
 Feature properties:
```

```
      Latitude: [42 44 45.5000]
     Longitude: [-72 -72.1000 -71]
```

Use dot notation to create new dynamic properties

```
pts.Name = {'point1', 'point2', 'point3'}

pts =

 3x1 geopoint vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
 Feature properties:
     Latitude: [42 44 45.5000]
    Longitude: [-72 -72.1000 -71]
         Name: {'point1' 'point2' 'point3'}
```

Get property values

```
pts.Name

ans =

    'point1'    'point2'    'point3'
```

Remove dynamic properties. To delete or remove dynamic properties, set them to `[]` or set the `Latitude` or `Longitude` property to `[]`.

```
pts.Temperature = 1:3

pts =

 3x1 geopoint vector with properties:

 Collection properties:
      Geometry: 'point'
      Metadata: [1x1 struct]
 Feature properties:
      Latitude: [42 44 45.5000]
     Longitude: [-72 -72.1000 -71]
          Name: {'point1' 'point2' 'point3'}
   Temperature: [1 2 3]
```

By setting the `Temperature` property to `[]`, this dynamic property is deleted.

```
pts.Temperature = []

pts =

 3x1 geopoint vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
 Feature properties:
     Latitude: [42 44 45.5000]
```

```
       Longitude: [-72 -72.1000 -71]
            Name: {'point1' 'point2' 'point3'}
```

To clear all fields in the `geopoint` vector, set the `Latitude` or `Longitude` property to `[]`.

```
pts.Latitude = []

pts =

 0x1 geopoint vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
 Feature properties:
     Latitude: []
    Longitude: []
```

Modify geopoint Object and Its Metadata

If you typically store latitude and longitude coordinates in an N-by-2 or 2-by-M array, you can assign these numeric values to a geopoint vector. If the coordinates are N-by-2, the first column is assigned to the `Latitude` property and the second column to the `Longitude` property. If the coordinates are 2-by-M, then the first row is assigned to the `Latitude` property and the second row to the `Longitude` property.

```
load coastlines;
ltln = [coastlat coastlon];        % 9865x2 array
pts = geopoint;                    % null constructor
pts(1:numel(coastlat)) = ltln;     % assign array
pts.Metadata.Name = 'coastline';
pts

pts =
 9865x1 geopoint vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
 Feature properties:
     Latitude: [-83.8300 -84.3300 -84.5000 -84.6700 -84.9200 -85.4200 ... ]
    Longitude: [-180 -178 -174 -170 -166.0000 -163 -158 -152 -146 -147 ... ]
```

```
pts.Metadata

ans = struct with fields:
    Name: 'coastline'
```

More About

Collection Properties

Collection properties contain only one value per class instance. In contrast, the Feature property type has attribute values associated with each feature. `Geometry` and `Metadata` are the only two Collection properties.

Feature Properties

Feature properties contain one value (a scalar number, a scalar string, or a character vector) for each feature in a geopoint vector. They are suitable for properties such as name, owner, serial number, or age, that describe a given feature (an element of a geopoint vector) as a whole. The `Latitude` and `Longitude` coordinate properties are feature properties as there is one value for each element in the geopoint vector.

Feature properties can be added after construction using dot notation. This is similar to adding dynamic fields to a structure.

Tips

- If `Latitude`, `Longitude`, or a dynamic property is set with more values than features in the geopoint vector, then all other properties expand in size using 0 for numeric values and an empty character vector (`''`) for cell values. See "Manipulate a Geopoint Vector" on page 1-535 for examples of these behaviors.

- If a dynamic property is set with fewer values than the number of features, then this dynamic property expands to match the size of the other properties by inserting a 0, if the value is numeric, or an empty character vector (`''`), if the value is a cell array.

- If the `Latitude` or `Longitude` property of the geopoint vector is set with fewer values than contained in the object, then all other properties shrink in size.

- If either `Latitude` or `Longitude` are set to [ ], then both coordinate properties are set to [ ] and all dynamic properties are removed.

- If a dynamic property is set to [ ], then it is removed from the object.

See Also

Functions
gpxread | shaperead

Objects
geoshape | mappoint | mapshape

Introduced in R2012a

geopointshape

Point shape in geographic coordinates

Description

A `geopointshape` object represents a point or multipoint in geographic coordinates. A multipoint is an individual point shape that contains a set of point locations.

To represent a point or multipoint in planar coordinates, use a `mappointshape` object instead.

Creation

To create `geopointshape` objects, either:

- Import point data in geographic coordinates as a geospatial table using the `readgeotable` function, and then query the `Shape` variable of the table.
- Use the `geopointshape` function (described here).

Syntax

`shape = geopointshape(lat,lon)`

Description

`shape = geopointshape(lat,lon)` creates a `geopointshape` object or array of `geopointshape` objects with the specified latitude and longitude coordinates. The sizes of `lat`, `lon`, and the `geopointshape` object array `shape` match.

Input Arguments

`lat` — Latitude coordinates
numeric array | cell array of numeric arrays

Latitude coordinates, specified as a numeric array or a cell array of numeric arrays.

- Create a point by specifying a scalar, such as `39`.
- Create a multipoint by specifying an array within a cell, such as `{[38 -30 29]}`.
- Create an array of points by specifying an array, such as `[38 -30 29]`.
- Create an array of points and multipoints by specifying a cell array of numeric arrays, such as `{39,[38 -30 29]}`.

Create placeholders for points with missing data by including `NaN` values. The `NaN` values in `lat` must correspond to the `NaN` values in `lon`.

The size of `lat` must match the size of `lon`. For cell arrays, the size of the array in each cell of `lat` must match the size of the array in the corresponding cell of `lon`.

Data Types: `double` | `cell`

lon — Longitude coordinates
numeric array | cell array of numeric arrays

Longitude coordinates, specified as a numeric array or a cell array of numeric arrays.

- Create a point by specifying a scalar, such as -113.
- Create a multipoint by specifying an array within a cell, such as {[-66 -31 42]}.
- Create an array of points by specifying an array, such as [-66 -31 42].
- Create an array of points and multipoints by specifying a cell array of numeric arrays, such as {-113,[-66 -31 42]}.

Create placeholders for points with missing data by including NaN values. The NaN values in lat must correspond to the NaN values in lon.

The size of lat must match the size of lon. For cell arrays, the size of the array in each cell of lat must match the size of the array in the corresponding cell of lon.

Properties

NumPoints — Number of points
array of nonnegative integers

This property is read-only.

Number of points, returned as an array of nonnegative integers.

For a geopointshape scalar, the value of NumPoints is 1 when the geopointshape object represents a single point and more than 1 when the object represents a multipoint.

For a geopointshape array, the size of NumPoints matches the size of the array.

Data Types: double

Latitude — Latitude coordinates
array

Latitude coordinates, specified as an array.

For a geopointshape scalar, the size of Latitude matches the value of NumPoints.

For a geopointshape array, the size of Latitude matches the size of NumPoints. If the array contains geopointshape objects with multipoints, then accessing the Latitude property of the array is not supported. Instead, access the Latitude property of individual objects within the array.

This property is read-only for arrays when any element of NumPoints is greater than 1.

Latitude and Longitude must be the same size.

Data Types: double

Longitude — Longitude coordinates
array

Longitude coordinates, specified as an array.

For a `geopointshape` scalar, the size of `Longitude` matches the value of `NumPoints`.

For a `geopointshape` array, the size of `Longitude` matches the size of `NumPoints`. If the array contains `geopointshape` objects with multipoints, then accessing the `Longitude` property of the array is not supported. Instead, access the `Longitude` property of individual elements within the array.

This property is read-only for arrays when any element of `NumPoints` is greater than `1`.

`Latitude` and `Longitude` must be the same size.

Data Types: `double`

Geometry — Geometric type
"point" (default)

This property is read-only.

Geometric type, returned as `"point"`.

Data Types: `string`

CoordinateSystemType — Coordinate system type
"geographic" (default)

This property is read-only.

Coordinate system type, returned as `"geographic"`.

Data Types: `string`

GeographicCRS — Geographic coordinate reference system
`[]` (default) | `geocrs` object

Geographic coordinate reference system (CRS), specified as a `geocrs` object. A geographic CRS consists of a datum (including its ellipsoid), prime meridian, and angular unit of measurement.

Examples

Get Information About Imported Geographic Points

Import a GPX file containing the coordinates of locations in Boston as a geospatial table. The GPX file represents the locations using points. Get information about the points by querying the `Shape` variable of the table.

```
GT = readgeotable("boston_placenames.gpx");
GT.Shape

ans =
  13×1 geopointshape array with properties:

             NumPoints: [13×1 double]
              Latitude: [13×1 double]
             Longitude: [13×1 double]
              Geometry: "point"
    CoordinateSystemType: "geographic"
```

```
        GeographicCRS: [1×1 geocrs]
```

Extract the latitude and longitude coordinates from the `Shape` variable. Display the locations on a road map.

```
lat = GT.Shape.Latitude;
lon = GT.Shape.Longitude;
geoscatter(lat,lon,"filled")
geobasemap("streets")
```
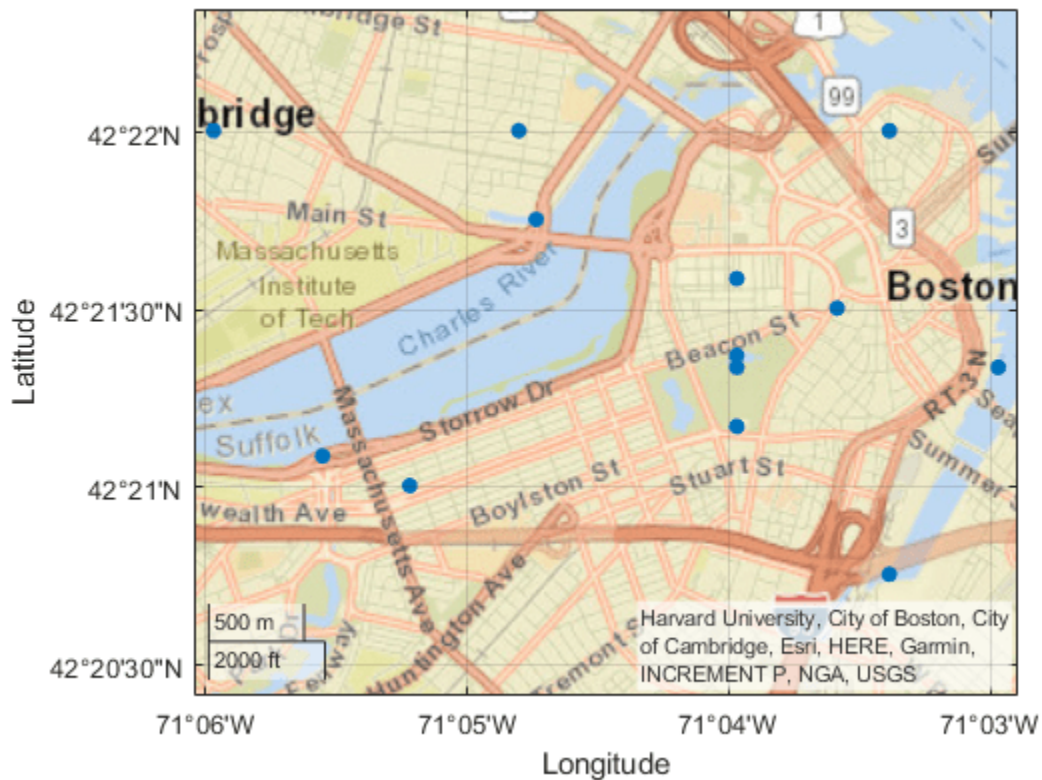

Create Geographic Points and Multipoints

Create an individual point as a `geopointshape` scalar. Specify the geographic CRS as the World Geodetic System of 1984, which has the EPSG code `4326`.

```
point = geopointshape(39,-113);
g = geocrs(4326);
point.GeographicCRS = g
```

```
point =
  geopointshape with properties:

            NumPoints: 1
             Latitude: 39
```

```
                 Longitude: -113
                  Geometry: "point"
       CoordinateSystemType: "geographic"
               GeographicCRS: [1x1 geocrs]
```

Create a multipoint as a `geopointshape` scalar.

```
multipoint = geopointshape({[38 -30 29]},{[-66 -31 42]});
multipoint.GeographicCRS = g

multipoint =
  geopointshape with properties:

                 NumPoints: 3
                  Latitude: [38 -30 29]
                 Longitude: [-66 -31 42]
                  Geometry: "point"
       CoordinateSystemType: "geographic"
               GeographicCRS: [1x1 geocrs]
```

Create three individual points as a 1-by-3 `geopointshape` vector.

```
pointVector = geopointshape([38 -20 29],[-66 -31 42]);
pointVector.GeographicCRS = g

pointVector =
  1x3 geopointshape array with properties:

                 NumPoints: [1 1 1]
                  Latitude: [38 -20 29]
                 Longitude: [-66 -31 42]
                  Geometry: "point"
       CoordinateSystemType: "geographic"
               GeographicCRS: [1x1 geocrs]
```

Create one individual point and one multipoint as a 1-by-2 `geopointshape` vector.

```
pointMultipoint = geopointshape({39,[38 -30 29]},{-113,[-66 -31 42]});
pointMultipoint.GeographicCRS = g

pointMultipoint =
  1x2 geopointshape array with properties:

                 NumPoints: [1 3]
                  Geometry: "point"
       CoordinateSystemType: "geographic"
               GeographicCRS: [1x1 geocrs]
```

See Also

Functions
readgeotable

Objects
geolineshape | geopolyshape | mappointshape

Topics
"Create Geospatial Tables"

Introduced in R2021b

geolineshape

Line shape in geographic coordinates

Description

A `geolineshape` object represents a line or multiline in geographic coordinates. A *multiline* is an individual line shape that contains a set of separate lines.

To represent a line or multiline in planar coordinates, use a `maplineshape` object instead.

Creation

To create `geolineshape` objects, either:

- Import line data in geographic coordinates as a geospatial table using the `readgeotable` function, and then query the `Shape` variable of the table.
- Use the `geolineshape` function (described here).

Syntax

```
shape = geolineshape(lat,lon)
```

Description

`shape = geolineshape(lat,lon)` creates a `geolineshape` object or array of `geolineshape` objects with vertices at the specified latitude and longitude coordinates. The sizes of `lat`, `lon`, and the `geolineshape` object array `shape` match.

Input Arguments

`lat` — Latitude coordinates
numeric vector | cell array of numeric vectors

Latitude coordinates, specified as a numeric vector or a cell array of numeric vectors.

- Create a line by specifying a vector, such as `[65 62 53 66]`.
- Create a multiline by specifying a vector and including line breaks as `NaN` values, such as `[55 34 18 NaN 14 19 42 26]`.
- Create an array of lines and multilines by specifying a cell array of vectors, such as `{[55 34 18],[14 19 NaN 42 26 31]}`.

The `NaN` values in `lat` must correspond to the `NaN` values in `lon`.

The size of `lat` must match the size of `lon`. For cell arrays, the size of the vector in each cell of `lat` must match the size of the vector in the corresponding cell of `lon`.

Data Types: `double` | `cell`

lon — Longitude coordinates
numeric vector | cell array of numeric vectors

Longitude coordinates, specified as a numeric vector or a cell array of numeric vectors.

- Create a line by specifying a vector, such as `[4 59 121 98]`.
- Create a multiline by specifying a vector and including line breaks as `NaN` values, such as `[78 56 63 NaN 83 106 104 126]`.
- Create an array of lines and multilines by specifying a cell array of vectors, such as `{[78 56 63],[83 106 NaN 104 126 131]}`.

The `NaN` values in `lat` must correspond to the `NaN` values in `lon`.

The size of `lat` must match the size of `lon`. For cell arrays, the size of the vector in each cell of `lat` must match the size of the vector in the corresponding cell of `lon`.

Properties

NumParts — Number of line parts
array of nonnegative integers

This property is read-only.

Number of line parts, returned as an array of nonnegative integers.

For a `geolineshape` scalar, the value of `NumParts` is `1` when the `geolineshape` object represents a single line and more than `1` when the object represents a multiline.

For a `geolineshape` array, the size of `NumParts` matches the size of the array.

Data Types: `double`

Geometry — Geometric type
`"line"` (default)

This property is read-only.

Geometric type, returned as `"line"`.

Data Types: `string`

CoordinateSystemType — Coordinate system type
`"geographic"` (default)

This property is read-only.

Coordinate system type, returned as `"geographic"`.

Data Types: `string`

GeographicCRS — Geographic coordinate reference system
`[]` (default) | `geocrs` object

Geographic coordinate reference system (CRS), specified as a `geocrs` object. A geographic CRS consists of a datum (including its ellipsoid), prime meridian, and angular unit of measurement.

Examples

Get Information About Imported Geographic Lines

Import the `tracks` layer of a GPX file containing routes in Massachusetts as a geospatial table. The `tracks` layer represents the routes using lines. Get information about the line in the second track by querying the `Shape` variable of the table.

```
GT = readgeotable("sample_tracks.gpx",Layer="tracks");
GT.Shape(2)
```

```
ans =
  geolineshape with properties:

              NumParts: 5
              Geometry: "line"
   CoordinateSystemType: "geographic"
        GeographicCRS: [1×1 geocrs]
```

Create a subtable that contains the second track. Display the second track on a web map.

```
GT2 = GT(2,:);
wmline(GT2,LineWidth=3)
```
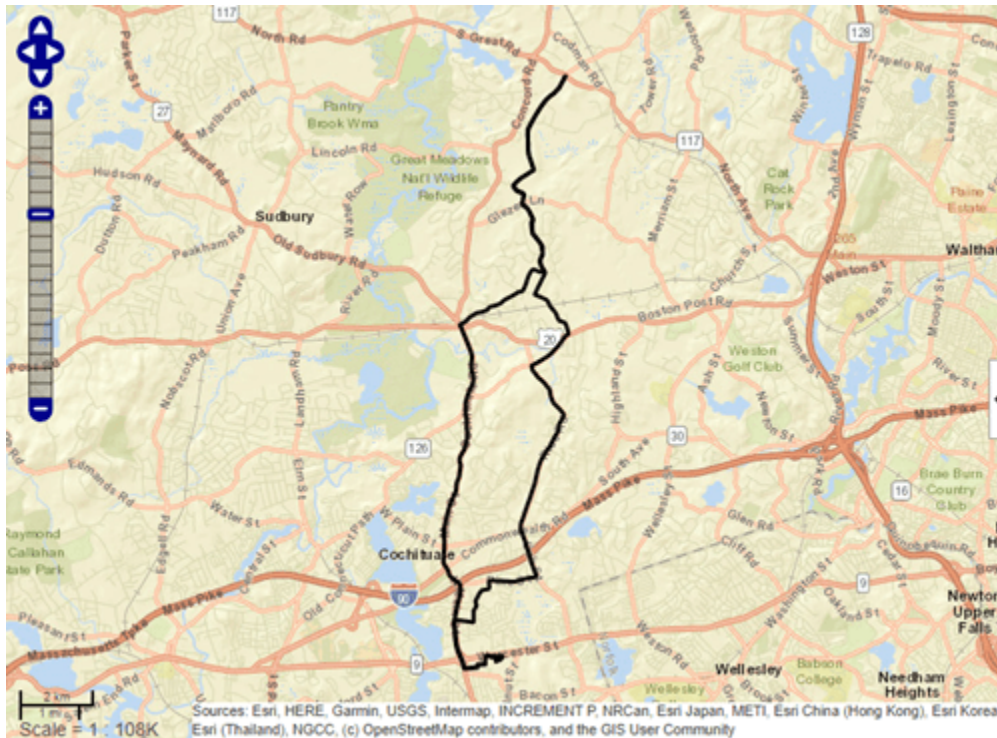
Create Geographic Lines and Multilines

Create an individual line as a `geolineshape` scalar. Specify the geographic CRS as the World Geodetic System of 1984, which has the EPSG code `4326`.

```
lat = [65 62 53 66];
lon = [4 59 121 98];
lineshp = geolineshape(lat,lon);

g = geocrs(4326);
lineshp.GeographicCRS = g
```

```
lineshp =
  geolineshape with properties:

               NumParts: 1
               Geometry: "line"
    CoordinateSystemType: "geographic"
          GeographicCRS: [1x1 geocrs]
```

Create a multiline as a `geolineshape` scalar.

```
lat = [55 34 18 NaN 14 19 42 26];
lon = [78 56 63 NaN 83 106 104 126];
multiline = geolineshape(lat,lon);
multiline.GeographicCRS = g
```

```
multiline =
  geolineshape with properties:

               NumParts: 2
               Geometry: "line"
    CoordinateSystemType: "geographic"
          GeographicCRS: [1x1 geocrs]
```

Create one individual line and one multiline as a 1-by-2 `geolineshape` array.

```
lat = {[55 34 18],[14 19 NaN 42 26 37]};
lon = {[78 56 63],[83 106 NaN 104 126 113]};
lineMultiline = geolineshape(lat,lon);
lineMultiline.GeographicCRS = g
```

```
lineMultiline=1×2 object
  1x2 geolineshape array with properties:

               NumParts: [1 2]
               Geometry: "line"
    CoordinateSystemType: "geographic"
          GeographicCRS: [1x1 geocrs]
```

See Also

Functions
readgeotable

Objects
geopointshape | geopolyshape | maplineshape

Topics
"Create Geospatial Tables"

Introduced in R2021b

geopolyshape

Polygon in geographic coordinates

Description

A `geopolyshape` object represents a polygon or multipolygon in geographic coordinates. A polygon is a region bounded by a closed curve and can include interior holes. A multipolygon is an individual polygon shape that includes multiple nonintersecting regions.

To represent a polygon or multipolygon in planar coordinates, use a `mappolyshape` object instead.

Creation

To create `geopolyshape` objects, either:

- Import polygon data in geographic coordinates as a geospatial table using the `readgeotable` function, and then query the `Shape` variable of the table.
- Use the `geopolyshape` function (described here).

Syntax

shape = geopolyshape(lat,lon)

Description

shape = geopolyshape(lat,lon) creates a `geopolyshape` object or array of `geopolyshape` objects with vertices at the specified latitude and longitude coordinates. The sizes of `lat`, `lon`, and the `geopolyshape` object array `shape` match.

The `geopolyshape` function assumes that `lat` and `lon` define polygons with valid topology. A polygon has valid topology when:

- Region interiors are to the right as you trace boundaries from vertex to vertex.
- The boundaries have no self-intersections.

In general, the outer boundaries of polygons with valid topology have vertices in a clockwise order and the interior holes have vertices in a counterclockwise order.

Input Arguments

`lat` — Latitude coordinates
numeric vector | cell array of numeric vectors

Latitude coordinates, specified as a numeric vector or a cell array of numeric vectors.

- Create a polygon by specifying a vector, such as [39 45 19 39].

- Create a polygon with holes or a multipolygon by specifying a vector and including breaks between the hole and region boundaries as `NaN` values, such as `[37 46 31 20 37 NaN 45 49 35 32 45 NaN 35 40 42 35]`.
- Create an array of polygons and multipolygons by specifying a cell array of vectors, such as `{[37 46 31 20 37],[45 49 35 32 45 NaN 35 40 42 35]}`.

The `NaN` values in `lat` must correspond to the `NaN` values in `lon`.

The size of `lat` must match the size of `lon`. For cell arrays, the size of the vector in each cell of `lat` must match the size of the vector in the corresponding cell of `lon`.

Data Types: `double` | `cell`

`lon` — Longitude coordinates
numeric vector | cell array of numeric vectors

Longitude coordinates, specified as a numeric vector or a cell array of numeric vectors.

- Create a polygon by specifying a vector, such as `[-113 -49 -100 -113]`.
- Create a polygon with holes or a multipolygon by specifying a vector and including breaks between the hole and region boundaries as `NaN` values, such as `[69 90 105 79 69 NaN 6 52 43 14 6 NaN 18 32 22 18]`.
- Create an array of polygons and multipolygons by specifying a cell array of vectors, such as `{[69 90 105 79 69],[6 52 43 14 6 NaN 18 32 22 18]}`.

The `NaN` values in `lat` must correspond to the `NaN` values in `lon`.

The size of `lat` must match the size of `lon`. For cell arrays, the size of the vector in each cell of `lat` must match the size of the vector in the corresponding cell of `lon`.

Properties

`NumRegions` — Number of polygon regions
array of nonnegative integers

This property is read-only.

Number of polygon regions, returned as an array of nonnegative integers. A region is a connected area such that any two points within the area can be connected by a path entirely within the area. Regions may contain holes.

For a `geopolyshape` scalar, the value of `NumRegions` is `1` when the `geopolyshape` object represents a single polygon and more than `1` when the object represents a multipolygon.

For a `geopolyshape` array, the size of `NumRegions` matches the size of the array.

Data Types: `double`

`NumHoles` — Number of holes
array of nonnegative integers

This property is read-only.

Number of holes in the polygon or multipolygon, returned as an array of nonnegative integers.

For a `geopolyshape` array, the size of `NumHoles` matches the size of the array.

Data Types: `double`

Geometry — Geometric type
`"polygon"` (default)

This property is read-only.

Geometric type, returned as `"polygon"`.

Data Types: `string`

CoordinateSystemType — Coordinate system type
`"geographic"` (default)

This property is read-only.

Coordinate system type, returned as `"geographic"`.

Data Types: `string`

GeographicCRS — Geographic coordinate reference system
`[]` (default) | `geocrs` object

Geographic coordinate reference system (CRS), specified as a `geocrs` object. A geographic CRS consists of a datum (including its ellipsoid), prime meridian, and angular unit of measurement.

Examples

Get Information About Imported Geographic Polygons

Import a shapefile containing worldwide land areas as a geospatial table. The shapefile represents the land areas using polygons.

```
GT = readgeotable("landareas.shp");
```

Create a subtable that contains the polygon representing Africa, Europe, and Asia. Get information about the polygon by querying the `Shape` variable of the table.

```
row = GT.Name == "Africa and Eurasia";
rowGT = GT(row,:);
rowGT.Shape
```

```
ans =
  geopolyshape with properties:

              NumRegions: 1
                NumHoles: 22
                Geometry: "polygon"
    CoordinateSystemType: "geographic"
            GeographicCRS: [1x1 geocrs]
```

Display the polygon on a world map.

```
worldmap([-45 80],[-25 195])
geoshow(rowGT)
```
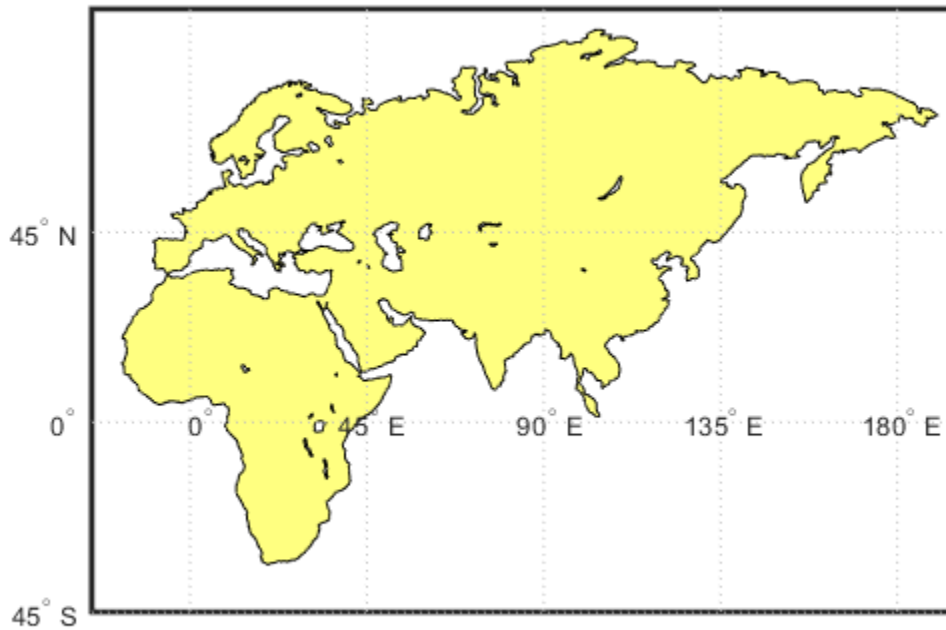

Create Geographic Polygons and Multipolygons

Create an individual polygon with no holes as a `geopolyshape` scalar. Specify the geographic CRS as the World Geodetic System of 1984, which has the EPSG code `4326`.

```
lat = [39 45 19 39];
lon = [-113 -49 -100 -113];
polyshp = geopolyshape(lat,lon);

g = geocrs(4326);
polyshp.GeographicCRS = g

polyshp =
  geopolyshape with properties:

             NumRegions: 1
               NumHoles: 0
               Geometry: "polygon"
    CoordinateSystemType: "geographic"
          GeographicCRS: [1x1 geocrs]
```

Create a multipolygon with two regions and one hole as a `geopolyshape` scalar.

```
lat = [37 46 31 20 37 NaN 45 49 35 32 45 NaN 35 40 42 35];
lon = [69 90 105 79 69 NaN 6 52 43 14 6 NaN 18 32 22 18];
multipolygon = geopolyshape(lat,lon);
multipolygon.GeographicCRS = g

multipolygon =
  geopolyshape with properties:

              NumRegions: 2
                NumHoles: 1
                Geometry: "polygon"
      CoordinateSystemType: "geographic"
            GeographicCRS: [1x1 geocrs]
```

Create two individual polygons as a 1-by-2 `geopolyshape` array. The second polygon contains a hole.

```
lat = {[37 46 31 20 37],[45 49 35 32 45 NaN 35 40 42 35]};
lon = {[69 90 105 79 69],[6 52 43 14 6 NaN 18 32 22 18]};
polyArray = geopolyshape(lat,lon);
polyArray.GeographicCRS = g

polyArray=1×2 object
  1x2 geopolyshape array with properties:

              NumRegions: [1 1]
                NumHoles: [0 1]
                Geometry: "polygon"
      CoordinateSystemType: "geographic"
            GeographicCRS: [1x1 geocrs]
```

See Also

Functions
readgeotable

Objects
geopointshape | geolineshape | mappolyshape

Topics
"Create Geospatial Tables"

Introduced in R2021b

geoquadline

Geographic quadrangle bounding multi-part line

Syntax

```
[latlim,lonlim] = geoquadline(lat,lon)
```

Description

`[latlim,lonlim] = geoquadline(lat,lon)` returns the limits of the tightest possible geographic quadrangle that bounds a line connecting vertices with geographic coordinates specified by `lat` and `lon`.

Examples

Bounding Quadrangle for the Brahmaputra River

Read shape data and then create a bounding box around the line.

```
brahmaputra = shaperead('worldrivers.shp','Selector', ...
    {@(name) strcmp(name,'Brahmaputra'), 'Name'}, 'UseGeoCoords',true);
[latlim, lonlim] = geoquadline(brahmaputra.Lat, brahmaputra.Lon)

latlim =

   23.8285   30.3831


lonlim =

   81.8971   95.4970
```

Input Arguments

lat — Latitudes along a line
vector

Latitudes along a line, specified as a vector representing an ordered sequences of vertices, in units of degrees. The line may be broken into multiple parts, delimited by values of `NaN`.

Data Types: `single` | `double`

lon — Longitudes along a line
vector

Longitudes along a line, specified as a vector representing an ordered sequences of vertices, in units of degrees. The line may be broken into multiple parts, delimited by values of `NaN`.

Data Types: `single` | `double`

Output Arguments

`latlim` — Latitude limits
1–by-2 vector

Latitude limits of a geographic quadrangle, returned as a 1-by-2 vector of the form `[southern_limit northern_limit]`, in units of degrees. The elements are in ascending order, and both lie in the closed interval [-90 90].

`lonlim` — Longitude limits
1–by-2 vector

Longitude limits of a geographic quadrangle, returned as a 1-by-2 vector of the form `[western_limit eastern_limit]`, in units of degrees. The limits are wrapped to the interval [-180 180]. They are not necessarily in numerical ascending order.

See Also

bufgeoquad | geoquadpt | outlinegeoquad | ingeoquad

Introduced in R2012b

geoquadpt

Geographic quadrangle bounding scattered points

Syntax

```
[latlim,lonlim] = geoquadpt(lat,lon)
```

Description

`[latlim,lonlim] = geoquadpt(lat,lon)` returns the limits of the tightest possible geographic quadrangle that bounds a set of points with geographic coordinates `lat` and `lon`.

In most cases, `tf = ingeoquad(lat,lon,latlim,lonlim)` will return true, but `tf` may be false for points on the edges of the quadrangle, due to round off. `tf` will also be false for elements of `lat` that fall outside the interval [-90 90] and elements of `lon` that are not finite.

Examples

Bounding Quadrangle Including Tokyo and Honolulu

In this case the output quadrangle straddles the 180-degree meridian, hence the elements of `lonlim` are in descending numerical order, although they are ordered from west to east.

Read a set of points and then create a bounding box around the points.

```
cities = shaperead('worldcities.shp','Selector', ...
    {@(name) any(strcmp(name,{'Tokyo','Honolulu'})),'Name'}, ...
    'UseGeoCoords',true);
[latlim,lonlim] = geoquadpt([cities.Lat],[cities.Lon])

latlim =

   21.3178   35.7082


lonlim =

  139.6401 -157.8291
```

Input Arguments

`lat` — Point latitudes
vector | matrix | N-D array

Point latitudes, specified as a vector, matrix, or N-D array, in units of degrees.

Data Types: `single` | `double`

`lon` — Point longitudes
vector | matrix | N-D array

Point longitudes, specified as a vector, matrix, or N-D array, in units of degrees.

Data Types: `single` | `double`

Output Arguments

`latlim` — Latitude limits
1–by-2 vector

Latitude limits of a geographic quadrangle, returned as a 1-by-2 vector of the form `[southern_limit northern_limit]`, in units of degrees. The elements are in ascending order, and both lie in the closed interval [-90 90].

`lonlim` — Longitude limits
1–by-2 vector

Longitude limits of a geographic quadrangle, returned as a 1-by-2 vector of the form `[western_limit eastern_limit]`, in units of degrees. The limits are wrapped to the interval [-180 180]. They are not necessarily in numerical ascending order.

See Also
bufgeoquad | geoquadline | outlinegeoquad | ingeoquad

Introduced in R2012b

georasterinfo

Information about geospatial raster data file

Syntax

```
info = georasterinfo(filename)
```

Description

`info = georasterinfo(filename)` returns a `RasterInfo` object for the geographic or projected raster data file specified by `filename`. Supported file formats include Esri Binary Grid, Esri GridFloat, GeoTIFF, and DTED. For a full list of supported formats, see "Supported Formats and Extensions" on page 1-565.

Examples

Get Information About Geospatial Raster Data

Get information about a geospatial raster data file by creating a `RasterInfo` object.

```
info = georasterinfo('boston.tif')

info =
  RasterInfo with properties:

                     Filename: "C:\Users\jbenham\OneDrive - MathWorks\Documents\MATLAB\ExampleMan
             FileModifiedDate: 13-May-2011 22:28:45
                     FileSize: 38729900
                   FileFormat: "GeoTIFF"
                   RasterSize: [2881 4481]
                     NumBands: 3
                 NativeFormat: "uint8"
         MissingDataIndicator: []
                   Categories: []
                    ColorType: "truecolor"
                     Colormap: []
              RasterReference: [1×1 map.rasterref.MapCellsReference]
    CoordinateReferenceSystem: [1×1 projcrs]
                     Metadata: [1×1 struct]
```

Access individual properties of the `RasterInfo` object using dot notation.

```
info.NativeFormat

ans =
"uint8"
```

Get DTED Metadata

Get information about a DTED file by creating a `RasterInfo` object. Get metadata specific to DTED files by accessing the `Metadata` property of the `RasterInfo` object.

```
info = georasterinfo('n39_w106_3arc_v2.dt1');
md = info.Metadata
```

```
md = struct with fields:
                  AREA_OR_POINT: "Point"
          DTED_CompilationDate: "0002"
               DTED_DataEdition: "02"
          DTED_DigitizingSystem: "SRTM      "
       DTED_HorizontalAccuracy: "0013"
          DTED_HorizontalDatum: "WGS84"
          DTED_MaintenanceDate: "0000"
   DTED_MaintenanceDescription: "0000"
           DTED_MatchMergeDate: "0000"
        DTED_MatchMergeVersion: "A"
            DTED_NimaDesignator: "DTED1"
            DTED_OriginLatitude: "0390000N"
           DTED_OriginLongitude: "1060000W"
      DTED_PartialCellIndicator: "00"
                  DTED_Producer: "USCNIMA "
      DTED_RelHorizontalAccuracy: "NA   "
        DTED_RelVerticalAccuracy: "0006"
           DTED_SecurityCode_DSI: "U"
           DTED_SecurityCode_UHL: "U   "
              DTED_UniqueRef_DSI: "G19 107         "
              DTED_UniqueRef_UHL: "G19 107       "
       DTED_VerticalAccuracy_ACC: "0006"
       DTED_VerticalAccuracy_UHL: "0006"
              DTED_VerticalDatum: "E96"
```

Find the coordinates of the lower-left corner of the data by accessing the `DTED_OriginLatitude` and `DTED_OriginLongitude` fields of the metadata structure. The coordinates are stored as strings. Convert the strings to angles.

```
latS = md.DTED_OriginLatitude;
lonS = md.DTED_OriginLongitude;
latA = str2angle(latS)
```

```
latA = 39
```

```
lonA = str2angle(lonS)
```

```
lonA = -106
```

The DTED file used in this example is courtesy of the US Geological Survey.

Input Arguments

`filename` — Name of raster data file
character vector | string scalar

Name of the raster data file, specified as a character vector or string scalar. The form of `filename` depends on the location of your file.

- If the file is in your current folder or in a folder on the MATLAB path, then specify the name of the file, such as `'myFile.dem'`.
- If the file is not in the current folder or in a folder on the MATLAB path, then specify the full or relative path name, such as `'C:\myfolder\myFile.tif'` or `'dataDir\myFile.dat'`.

For a list of supported file formats, see "Supported Formats and Extensions" on page 1-565.

Data Types: `char` | `string`

More About

Supported Formats and Extensions

The `readgeoraster` and `georasterinfo` functions support these file formats and extensions. In some cases, you can read supported file formats using extensions other than the ones listed.

File Format	Extension
GeoTIFF	`.tif` or `.tiff`
Esri Binary Grid	`.adf`
Esri ASCII Grid	`.asc` or `.grd`
Esri GridFloat	`.flt`
DTED	`.dt0`, `.dt1`, or `.dt2`
SDTS	`.DDF`
USGS DEM	`.dem`
ER Mapper ERS	`.ers`
ENVI	`.dat`
ERDAS IMAGINE	`.img`
Vertical Mapper Numeric Grid	`.grd`
Vertical Mapper Classified Grid	`.grc`
SRTM Height	`.hgt`

Some file formats consist of a data file and multiple supporting files. For example, Esri GridFloat files may have supporting header files (`.hdr`). When you read a data file with supporting files using `readgeoraster` or `georasterinfo`, specify the extension of the data file.

File formats may be referred to using different names. For example, the Esri GridFloat format may also be referred to as Esri `.hdr` Labelled or ITT ESRI `.hdr` RAW Raster. The Esri Binary Grid format may also be referred to as ArcGrid Binary, Esri ArcGIS Binary Grid, or Esri ArcInfo Grid.

See Also

Functions
readgeoraster

Objects
RasterInfo

Topics
"Find Geospatial Raster Data"

Introduced in R2020a

georasterref

Construct geographic raster reference object

Note Use the `georefcells` function or the `georefpostings` function instead, except when constructing a raster reference object from a world file matrix.

Syntax

```
R = georasterref(W,rasterSize)
R = georasterref(W,rasterSize,rasterInterpretation)
R = georasterref(Name,Value)
```

Description

`R = georasterref(W,rasterSize)` creates a reference object for a regular raster of cells in geographic coordinates using the specified world file matrix `W` and raster size `rasterSize`.

`R = georasterref(W,rasterSize,rasterInterpretation)`, where `rasterInterpretation` is `'postings'`, specifies that the raster contains regularly posted samples in geographic coordinates. The default for `rasterInterpretation` is `'cells'`, which specifies a regular raster of cells.

`R = georasterref(Name,Value)` accepts a list of name-value pairs that are used to assign selected properties when initializing a geographic raster reference object.

Input Arguments

`W` — World file matrix
2-by-3 numeric array

World file matrix, specified as a 2-by-3 numeric array. Each of the six elements in `W` matches one of the lines in a world file that defines the transformation in raster referencing object R.

Data Types: `double`

`rasterSize` — Number of rows and columns of the raster
two-element vector

Number of rows (*m*) and columns (*n*) of the raster or image associated with the referencing object, specified as a two-element vector [*m n*]. For convenience, you may assign a size vector having more than two elements to `RasterSize`. This flexibility enables assignments like `R.RasterSize = size(RGB)`, for example, where RGB is *m*-by-*n*-by-3. However, in such cases, only the first two elements of the size vector are actually stored. The higher (non-spatial) dimensions are ignored.

`rasterInterpretation` — Control to handle raster edges
`'cells'` (default) | `'postings'`

Controls handling of raster edges. The `rasterInterpretation` input is optional, and can equal either `'cells'` or `'postings'`.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

You can include any of the following properties, overriding their default values as needed. Alternatively, you may omit any or all properties when constructing your geographic raster reference object. Then, you can customize the result by resetting properties from this list one at a time. The exception is the `RasterInterpretation` property. To have a raster interpretation of `'postings'` (rather than the default, `'cells'`), the name-value pair `'RasterInterpretation','postings'` must be specified in your call to `georasterref`.

LatitudeLimits

Limits in latitude of the geographic quadrangle bounding the georeferenced raster. A two-element vector of the form:

`[southern_limit northern_limit]`

Default: `[0.5 2.5]`

LongitudeLimits

Limits in longitude of the geographic quadrangle bounding the georeferenced raster. A two-element vector of the form:

`[western_limit eastern_limit]`

Default: `[0.5 2.5]`

RasterSize

Two-element vector [*m* *n*] specifying the number of rows (*m*) and columns (*n*) of the raster or image associated with the referencing object. For convenience, you may assign a size vector having more than two elements to `RasterSize`. This flexibility enables assignments like `R.RasterSize = size(RGB)`, for example, where RGB is *m*-by-*n*-by-3. However, in such cases, only the first two elements of the size vector are actually stored. The higher (non-spatial) dimensions are ignored.

Default: `[2 2]`

RasterInterpretation

Controls handling of raster edges, specified as either `'cells'` or `'postings'`. If you want this property to have other than the default value, you must set it when you create the object. Once created, you cannot change the value of this property in a geographic raster reference object.

Default: `'cells'`

ColumnsStartFrom

Edge from which column indexing starts, specified as either `'south'` or `'north'`.

Default: `'south'`

RowsStartFrom

Edge from which row indexing starts, specified as either `'west'` or `'east'`.

Default: `'west'`

Output Arguments

R — Geographic raster
GeographicCellsReference or GeographicPostingsReference object

Geographic raster, specified as a GeographicCellsReference or
GeographicPostingsReference object.

Examples

Construct a referencing object for a global raster comprising a grid of 180-by-360 one-degree cells, with rows that start at longitude -180, and with the first cell located in the northwest corner.

```
R = georasterref('RasterSize', [180 360], ...
     'RasterInterpretation', 'cells', 'ColumnsStartFrom', 'north', ...
     'LatitudeLimits', [-90 90], 'LongitudeLimits', [-180 180])
```

Construct a referencing object for the DTED Level 0 file that includes Sagarmatha (Mount Everest). The DTED columns run from south to north and the first column runs along the western edge of the (one-degree-by-one-degree) quadrangle, consistent with the default values for `'ColumnsStartFrom'` and `'RowsStartFrom'`.

```
R = georasterref('LatitudeLimits',[27 28],'LongitudeLimits',[86 87], ...
        'RasterSize', [121 121], 'RasterInterpretation', 'postings')
```

Repeat the second example with a different strategy: Create an object by specifying only the `RasterInterpretation` value, then modify the object by resetting additional properties. (As noted above, the `RasterInterpretation` of an existing raster reference object cannot be changed.)

```
R = georasterref('RasterInterpretation','postings');
R.RasterSize = [121 121];
R.LatitudeLimits  = [27 28];
R.LongitudeLimits = [86 87];
```

Repeat the first example using a world file matrix as input.

```
W = [1     0   -179.5; ...
     0    -1     89.5];
rasterSize = [180 360];
rasterInterpretation = 'cells';
R = georasterref(W, rasterSize, rasterInterpretation);
```

See Also

Functions
georefcells | georefpostings | maprasterref | worldFileMatrix

Objects
GeographicCellsReference | GeographicPostingsReference

Introduced in R2011a

georefpostings

Reference raster postings to geographic coordinates

Syntax

```
R = georefpostings()
R = georefpostings(latlim,lonlim,rasterSize)
R = georefpostings(latlim,lonlim,latspacing,lonspacing)
R = georefpostings(latlim,lonlim,___,Name,Value)
```

Description

`R = georefpostings()` returns a default referencing object for a raster of regularly posted samples in geographic coordinates.

`R = georefpostings(latlim,lonlim,rasterSize)` constructs a referencing object for a raster spanning the specified limits in latitude and longitude, with the numbers of rows and columns specified by `rasterSize`.

`R = georefpostings(latlim,lonlim,latspacing,lonspacing)` allows the geographic sample spacings to be set precisely. The geographic limits will be adjusted slightly, if necessary, to ensure an integer number of samples in each dimension.

`R = georefpostings(latlim,lonlim,___,Name,Value)` allows the directions of the columns and rows to be specified via name-value pairs.

Examples

Construct Geographic Referencing Object for Global Raster

Define latitude and longitude limits and the dimensions of the raster.

```
latlim = [-90 90];
lonlim = [-180 180];
rasterSize = [181 361];
```

Create the referencing object specifying the raster size.

```
R = georefpostings(latlim,lonlim,rasterSize,'ColumnsStartFrom','north')

R =
  GeographicPostingsReference with properties:

            LatitudeLimits: [-90 90]
           LongitudeLimits: [-180 180]
                RasterSize: [181 361]
     RasterInterpretation: 'postings'
          ColumnsStartFrom: 'north'
             RowsStartFrom: 'west'
    SampleSpacingInLatitude: 1
```

```
    SampleSpacingInLongitude: 1
      RasterExtentInLatitude: 180
     RasterExtentInLongitude: 360
              XIntrinsicLimits: [1 361]
              YIntrinsicLimits: [1 181]
        CoordinateSystemType: 'geographic'
                 GeographicCRS: []
                      AngleUnit: 'degree'
```

Obtain the same result by specifying the sample spacing.

```
spacing = 1;
```

```
R = georefpostings(latlim,lonlim,spacing,spacing,'ColumnsStartFrom','north')
```

```
R =
  GeographicPostingsReference with properties:

                  LatitudeLimits: [-90 90]
                 LongitudeLimits: [-180 180]
                       RasterSize: [181 361]
          RasterInterpretation: 'postings'
               ColumnsStartFrom: 'north'
                   RowsStartFrom: 'west'
       SampleSpacingInLatitude: 1
      SampleSpacingInLongitude: 1
        RasterExtentInLatitude: 180
       RasterExtentInLongitude: 360
              XIntrinsicLimits: [1 361]
              YIntrinsicLimits: [1 181]
          CoordinateSystemType: 'geographic'
                   GeographicCRS: []
                        AngleUnit: 'degree'
```

Input Arguments

`latlim` — Latitude limits in degrees
[0.5 2.5] (default) | 1-by-2 numeric vector

Latitude limits in degrees, specified as a 1-by-2 numeric vector. The number of rows in the resulting raster is specified by `rasterSize`.

Example: `latlim = [-90 90];`

Data Types: `double`

`lonlim` — Longitude limits in degrees
[0.5 2.5] (default) | 1-by-2 numeric vector

Longitude limits in degrees, specified as a 1-by-2 numeric vector. The number of columns in the resulting raster is specified by `rasterSize`.

Example: `lonlim = [-180 180];`

Data Types: double

rasterSize — Size of the raster
[2 2] (default) | 1-by-2 numeric vector

Size of the raster, specified as a 1-by-2 numeric vector.

Example: rasterSize = [180 360];

Data Types: double

latspacing — Vertical spacing of posting
1 (default) | numeric scalar

Vertical spacing of posting, specified as a numeric scalar. The value of latspacing determines the SampleSpacingInLatitude property of R.

Example: latspacing = 1.5

Data Types: double

lonspacing — Horizontal spacing of postings
1 (default) | numeric scalar

Horizontal spacing of postings, specified as a numeric scalar. The value of lonspacing determines the SampleSpacingInLongitude property of R.

Example: lonspacing = 1.5

Data Types: double

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: R = georefpostings(latlim,lonlim,rasterSize,'ColumnsStartFrom','north')

ColumnsStartFrom — Edge from which column indexing starts
'south' (default) | 'north'

Edge from which column indexing starts, specified as either 'north' or 'south'.

Example: R = georefpostings(latlim,lonlim,rasterSize,'ColumnsStartFrom','north')

Data Types: char | string

RowsStartFrom — Edge from which row indexing starts
'west' (default) | 'east'

Edge from which column indexing starts, specified as either 'east' or 'west'.

Example: R = georefpostings(latlim,lonlim,rasterSize,'RowsStartFrom','east')

Data Types: char | string

Output Arguments

R — Object that references raster postings to geographic coordinates
GeographicPostingsReference raster reference object

Object that references raster postings to geographic coordinates, returned as a
GeographicPostingsReference raster reference object.

Tips

- To construct a geographic raster reference object from a world file matrix, use the georasterref function.

See Also

georefcells | GeographicPostingsReference | maprefpostings

Introduced in R2015b

georefcells

Reference raster cells to geographic coordinates

Syntax

```
R = georefcells()
R = georefcells(latlim,lonlim,rasterSize)
R = georefcells(latlim,lonlim,latcellextent,loncellextent)
R = georefcells(latlim,lonlim,___,Name,Value)
```

Description

`R = georefcells()` returns a default referencing object for a regular raster of cells in geographic coordinates.

`R = georefcells(latlim,lonlim,rasterSize)` constructs a referencing object for a raster of cells spanning the specified limits in latitude and longitude, with the numbers of rows and columns specified by rasterSize.

`R = georefcells(latlim,lonlim,latcellextent,loncellextent)` allows the geographic cell extents to be set precisely. If necessary, `georefcells` adjusts the geographic limits slightly to ensure an integer number of cells in each dimension.

`R = georefcells(latlim,lonlim,___,Name,Value)` allows the directions of the columns and rows to be specified via name-value pairs.

Examples

Construct Referencing Object for Global Raster

Construct a referencing object for a global raster comprising a grid of 180-by-360 one-degree cells, with rows that start at longitude -180, and with the first cell located in the northwest corner.

```
latlim = [-90 90];
lonlim = [-180 180];
rasterSize = [180 360];
```

Create a raster referencing object by specifying the raster size.

```
R = georefcells(latlim,lonlim,rasterSize,'ColumnsStartFrom','north')

R =
  GeographicCellsReference with properties:

            LatitudeLimits: [-90 90]
           LongitudeLimits: [-180 180]
                RasterSize: [180 360]
      RasterInterpretation: 'cells'
         ColumnsStartFrom: 'north'
             RowsStartFrom: 'west'
```

```
            CellExtentInLatitude: 1
           CellExtentInLongitude: 1
         RasterExtentInLatitude: 180
        RasterExtentInLongitude: 360
                XIntrinsicLimits: [0.5 360.5]
                YIntrinsicLimits: [0.5 180.5]
            CoordinateSystemType: 'geographic'
                  GeographicCRS: []
                      AngleUnit: 'degree'
```

Obtain the same result by specifying cell extents.

```
extent = 1;
```

```
R = georefcells(latlim,lonlim,extent,extent,'ColumnsStartFrom','north')
```

```
R =
  GeographicCellsReference with properties:

                LatitudeLimits: [-90 90]
               LongitudeLimits: [-180 180]
                    RasterSize: [180 360]
         RasterInterpretation: 'cells'
              ColumnsStartFrom: 'north'
                 RowsStartFrom: 'west'
         CellExtentInLatitude: 1
        CellExtentInLongitude: 1
       RasterExtentInLatitude: 180
      RasterExtentInLongitude: 360
              XIntrinsicLimits: [0.5 360.5]
              YIntrinsicLimits: [0.5 180.5]
          CoordinateSystemType: 'geographic'
                GeographicCRS: []
                    AngleUnit: 'degree'
```

Input Arguments

`latlim` — Latitude limits in degrees
[0.5 2.5] (default) | 1-by-2 numeric vector

Latitude limits in degrees, specified as a 1-by-2 numeric vector. The number of rows in the resulting raster is specified by `rasterSize`.

Example: `latlim = [-90 90];`

Data Types: `double`

`lonlim` — Longitude limits in degrees
[0.5 2.5] (default) | 1-by-2 numeric vector

Longitude limits in degrees, specified as a 1-by-2 numeric vector. The number of columns in the resulting raster is specified by `rasterSize`.

Example: `lonlim = [-180 180];`

Data Types: `double`

rasterSize — Size of the raster
[2 2] (default) | 1-by-2 numeric vector

Size of the raster, specified as a 1-by-2 numeric vector.

Example: `rasterSize = [180 360];`

Data Types: `double`

latcellextent — Height of cells
1 (default) | numeric scalar

Height of cells, specified as a numeric scalar. The value of `latcellextent` determines the `CellExtentInLatitude` property of R.

Example: `latcellextent = 1.5`

Data Types: `double`

loncellextent — Width of cells
1 (default) | numeric scalar

Width of cells, specified as a numeric scalar. The value of `loncellextent` determines the `CellExtentInLongitude` property of R.

Example: `latcellextent = 1.5`

Data Types: `double`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `R = georefcells(latlim,lonlim,rasterSize,'ColumnsStartFrom','north')`

ColumnsStartFrom — Edge from which column indexing starts
`'south'` (default) | `'north'`

Edge from which column indexing starts, specified as either `'north'` or `'south'`.

Example: `R = georefcells(latlim,lonlim,rasterSize,'ColumnsStartFrom','north')`

Data Types: `char` | `string`

RowsStartFrom — Edge from which row indexing starts
`'west'` (default) | `'east'`

Edge from which column indexing starts, specified as either `'east'` or `'west'`.

Example: `R = georefcells(latlim,lonlim,rasterSize,'RowsStartFrom','east')`

Data Types: `char` | `string`

Output Arguments

R — Object that references raster cells to geographic coordinates
GeographicCellsReference raster reference object

Object that references raster cells to geographic coordinates, returned as a
GeographicCellsReference raster reference object.

Tips

- To construct a geographic raster reference object from a world file matrix, use the georasterref function.

See Also

georefpostings | GeographicCellsReference | maprefcells

Introduced in R2015b

georesize

Resize geographic raster

Syntax

```
[B,RB] = georesize(A,RA,scale)
[B,RB] = georesize(A,RA,latscale,lonscale)
[B,RB] = georesize( ___ ,method)
[B,RB] = georesize( ___ ,'Antialiasing',TF)
```

Description

`[B,RB] = georesize(A,RA,scale)` returns the raster B that is `scale` times the size of the raster A. RA is a geographic raster reference object that specifies the location and extent of data in A. `georesize` returns the raster reference object RB that is associated with the returned raster B. By default, `georesize` uses cubic interpolation.

`georesize` preserves the limits of the raster. If the scale specified divides evenly into the numbers of cells in each dimension, or the number of samples in each dimension minus 1, the limits of the output are the same as the input. Otherwise, `georesize` adjusts the output limits by a fraction of the cell extents or sample spacing values.

`[B,RB] = georesize(A,RA,latscale,lonscale)` returns the raster B that is `latscale` times the size of A in column (north-south) direction and `lonscale` times the size of A in the row (east-west) direction.

`[B,RB] = georesize( ___ ,method)` returns a resized raster where `method` specifies the interpolation method.

`[B,RB] = georesize( ___ ,'Antialiasing',TF)` specifies whether to perform antialiasing when shrinking a raster. The default depends on the type of interpolation. For nearest-neighbor interpolation, the default value is `false`. For all other interpolation methods, the default is `true`.

Examples

Resize Geographic Raster

Import a sample geographic raster and geographic cells reference object.

```
[Z,R] = readgeoraster('raster_sample2.tif');
```

Resize the raster using `georesize`. Double the length and width of the raster by specifying the scale as 2. Use nearest neighbor interpolation by specifying the interpolation method as `'nearest'`.

```
[Z2,R2] = georesize(Z,R,2,'nearest');
```

You can also resize the input raster by specifying different scales for the north-south and east-west directions.

```
[Z3,R3] = georesize(Z,R,3,2,'nearest');
R3.RasterSize
```

ans = *1×2*

     6   4

Verify the raster has been resized by comparing the size of the original raster with the size of the updated rasters.

```
R.RasterSize
```

ans = *1×2*

     2   2

```
R2.RasterSize
```

ans = *1×2*

     4   4

```
R3.RasterSize
```

ans = *1×2*

     6   4

If the rasters are small, you can compare them directly.

```
Z
```

Z = *2×2*

    1   2
    3   4

```
Z2
```

Z2 = *4×4*

    1   1   2   2
    1   1   2   2
    3   3   4   4
    3   3   4   4

```
Z3
```

Z3 = *6×4*

    1   1   2   2
    1   1   2   2
    1   1   2   2
    3   3   4   4

```
3     3     4     4
3     3     4     4
```

Resize Geographic Raster Data Set

Load a raster data set showing land elevations and bathymetry for the region around the Korean peninsula, at a resolution of 12 cells per degree. The data includes a raster image, `korea5c`, and an associated geographic raster reference object, `korea5cR`.

```
load korea5c
```

View the raster data set, using `geoshow`, specifying the associated raster reference object.

```
geoshow(korea5c,korea5cR,'DisplayType','texturemap')
```
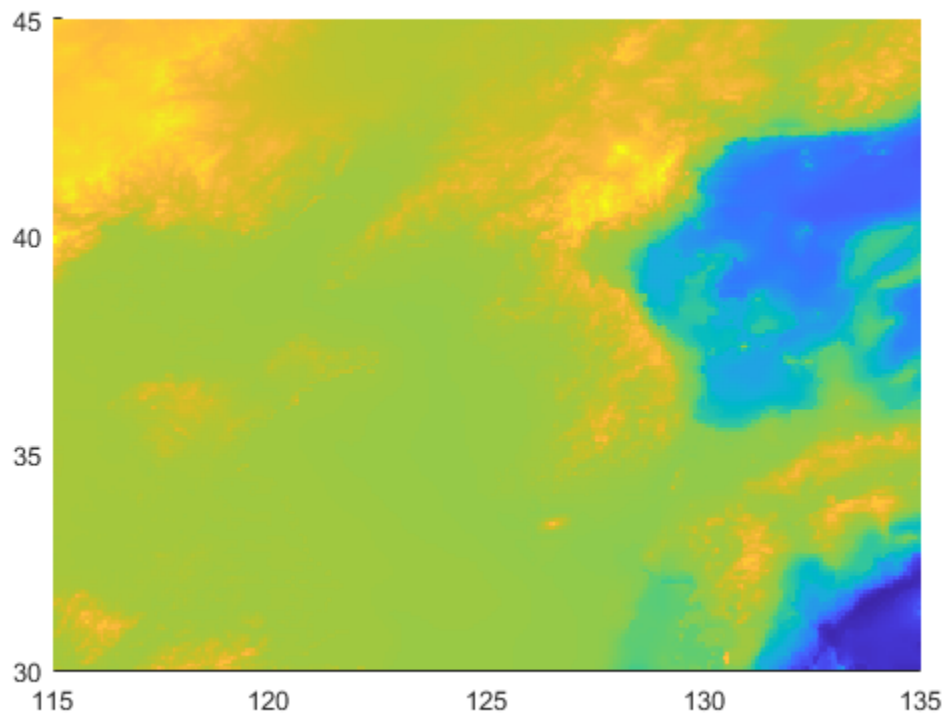

Resize the raster to be a quarter of its original size.

```
[resizedKorea,resizedKoreaR] = georesize(korea5c,korea5cR,0.25);
```

View the resized raster. Note that `geoshow` preserves the original limits of the map in the display so that, at first glance, the resized raster appears to be the same size as the original. A closer look reveals that the size of pixels in the resized raster are larger than the pixels in the original.

```
figure
geoshow(resizedKorea,resizedKoreaR,'DisplayType','texturemap')
```
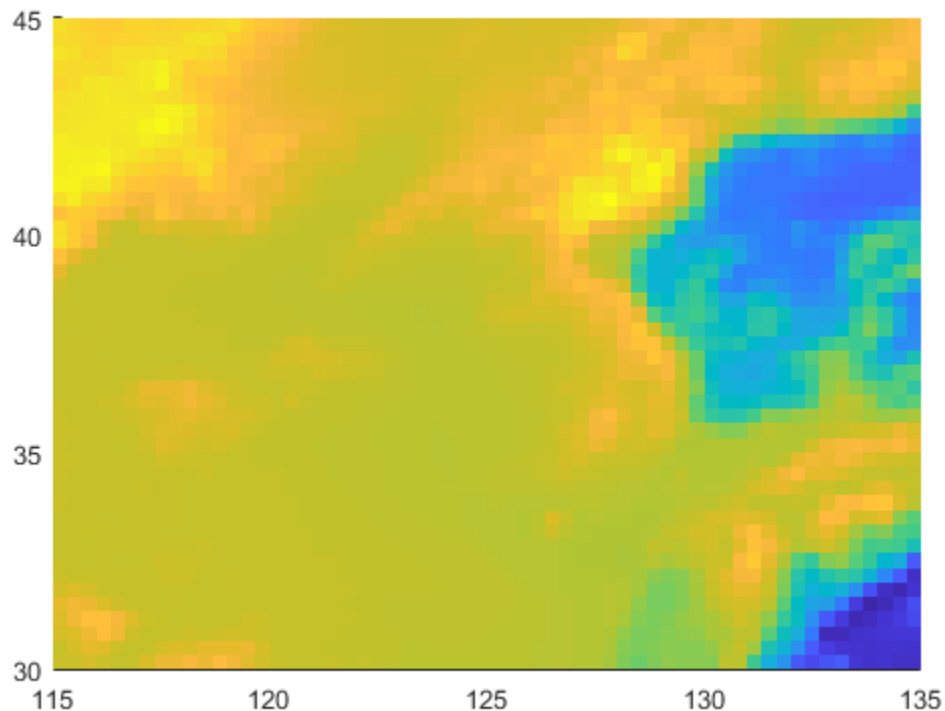

Input Arguments

A — Raster to be resized
numeric or logical array

Raster to be resized, specified as a numeric or logical array. If A has more than two dimensions, such as with a color raster in RGB format, `georesize` only resizes the first two dimensions.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

RA — Information about location and extent of raster
raster reference object

Information about location and extent of raster, specified as a raster reference object. To convert a referencing vector or referencing matrix into a raster reference object, use the `refvecToGeoRasterReference` or `refmatToGeoRasterReference`.

scale — Amount of resizing
numeric scalar

Amount of resizing, specified as numeric scalar. If `scale` is in the range `[0 1]`, B is smaller than A. If `scale` is greater than 1, B is larger than A.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`latscale` — Amount of resizing in north-south direction
numeric scalar

Amount of resizing in north-south direction, specified as numeric scalar. If `latscale` is in the range `[0 1]`, B is smaller than A. If `latscale` is greater than 1, B is larger than A.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`lonscale` — Amount of resizing in east-west direction
numeric scalar

Amount of resizing in east-west direction, specified as numeric scalar. If `lonscale` is in the range `[0 1]`, B is smaller than A. If `lonscale` is greater than 1, B is larger than A.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`method` — Interpolation method
`'cubic'` (default) | `'nearest'` | `'bilinear'`

Interpolation method, specified as one of the following values:

Value	Description
`'nearest'`	Nearest-neighbor interpolation
`'bilinear'`	Bilinear interpolation
`'cubic'`	Cubic interpolation

Data Types: `char` | `string`

Output Arguments

B — Resized raster
numeric or logical array

Resized raster, returned as a numeric or logical array.

RB — Information about location and extent of raster
geographic raster reference object

Information about location and extent of the raster, returned as a geographic raster reference object.

Tips

*   Use `georesize` with raster data in latitude and longitude coordinates. To work with projected raster data, in *x*- and *y*-coordinates, use `mapresize`.

See Also

`geointerp` | `georefcells` | `georefpostings` | `mapresize`

Introduced in R2019a

geoshape

Geographic shape vector

Description

A geoshape vector is an object that represents geographic vector features with either point, line, or polygon topology. The features consist of latitude and longitude coordinates and associated attributes.

Attributes that vary spatially are termed Vertex properties. These elements of the geoshape vector are coupled such that the length of the latitude and longitude coordinate property values are always equal in length to any additional dynamic Vertex properties.

Attributes that only pertain to the overall feature (point, line, polygon) are termed Feature properties. Feature properties are not linked to the autosizing mechanism of the Vertex properties. Both property types can be added to a geoshape vector during construction or by using standard dot (.) notation after construction.

To create a geographic point, line, or polygon shape for use with a geospatial table, create a `geopointshape`, `geolineshape`, or `geopolyshape` object instead.

Creation

Syntax

```
s = geoshape()
s = geoshape(latitude,longitude)
s = geoshape(latitude,longitude,Name,Value)
s = geoshape(structArray)
s = geoshape(latitude,longitude,structArray)
```

Description

`s = geoshape()` constructs an empty geoshape vector, `s`, with these default property settings.

```
s =

 0x1 geoshape vector with properties:

 Collection properties:
     Geometry: 'line'
     Metadata: [1x1 struct]
 Vertex properties:
     Latitude: []
    Longitude: []
```

`s` is always a column vector.

`s = geoshape(latitude,longitude)` sets the `Latitude` and `Longitude` properties of geoshape vector `s`.

`s = geoshape(latitude,longitude,Name,Value)` sets the `Latitude` and `Longitude` properties, then adds dynamic properties to the geoshape vector using `Name,Value` argument pairs. You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

`s = geoshape(structArray)` constructs a `geoshape` vector from the fields of the structure array, `structArray`.

- If `structArray` contains the field `Lat`, and does not contain the field `Latitude`, then the `Latitude` property values are set equal to the `Lat` field values. Similar behavior occurs when `structArray` contains the field `Lon` and does not contain the field `Longitude`.

- If `structArray` contains both `Lat` and `Latitude` fields, then the `Latitude` property values are set equal to the `Latitude` field values. Also, a `Lat` dynamic property is created and its values are set equal to the `Lat` field values. Similar behavior occurs for `Lon` and `Longitude` fields when both are present in `structArray`.

- Other `structArray` fields are assigned to `s` and become dynamic properties. Field values in `structArray` that are not numeric values, string scalars, string arrays, character vectors, logical, or cell arrays of numeric values, logical, or character vectors are ignored. You can specify vectors within cell arrays as either row or column vectors.

`s = geoshape(latitude,longitude,structArray)` sets the `Latitude` and `Longitude` properties, and sets dynamic properties from the field values of `structArray`.

- If `structArray` contains the fields `Lat`, `Latitude`, `Lon` or `Longitude`, then those field values are ignored since the `latitude` and `longitude` input vectors set the `Latitude` and `Longitude` property values.

Properties

The `geoshape` class is a general class that represents various geographic features. This class permits features to have more than one vertex and can thus represent lines and polygons in addition to multipoints. For more about the property types in `geoshape`, see "Collection Properties" on page 1-597, "Vertex Properties" on page 1-597, and "Feature Properties" on page 1-597.

Dynamic properties are new features and vertices that are added to a geoshape vector. You can attach dynamic properties to a geoshape vector during construction using a `Name,Value` argument, or after construction using dot (`.`) notation. This is similar to adding new fields to a structure. For an example of adding dynamic Feature properties, see "Construct a Geoshape Vector with Dynamic Properties" on page 1-587.

Geometry — Shape of every feature in the geoshape vector
`'line'` (default) | `'point'` | `'polygon'`

Shape of every feature in the geoshape vector, specified as `'line'`, `'point'`, or `'polygon'`. `Geometry` is a Collection property so there can be only one value per object instance and its purpose is purely informational. The three allowable values for `Geometry` do not change class behavior. The class does not validate line or polygon topologies.

Data Types: `char` | `string`

Latitude — Latitude coordinates
numeric row or column vector

Latitude coordinates, specified as a numeric row or column vector. `Latitude` is stored as a row vector. `Latitude` is a Vertex property.

Data Types: `double` | `single`

Longitude — Longitude coordinates
numeric row or column vector

Longitude coordinates, specified as a row or column vector. `Longitude` is stored as a row vector. `Longitude` is a Vertex property.

Data Types: `double` | `single`

Metadata — Information for every feature
scalar structure

Information for every feature, specified as a scalar structure. You can add any data type to the structure. `Metadata` is a Collection property, so only one instance per object is allowed.

- If `'Metadata'` is provided as a dynamic property name in the constructor, and the corresponding value is a scalar structure, then the `Value` is copied to the `Metadata` property. Otherwise, an error is issued.
- If a `Metadata` field is provided by `structArray`, and both `Metadata` and `structArray` are scalar structures, then the `Metadata` field value is copied to the `Metadata` property value. If `structArray` is a scalar but the `Metadata` field is not a structure, then an error is issued. If `structArray` is not scalar, then the `Metadata` field is ignored.

Data Types: `struct`

Object Functions

append	Append features to geographic or planar vector
cat	Concatenate geographic or planar vector
disp	Display geographic or planar vector
fieldnames	Return dynamic property names of geographic or planar vector
isempty	Determine if geographic or planar vector is empty
isfield	Determine if dynamic property exists in geographic or planar vector
isprop	Determine if property exists in geographic or planar vector
length	Return number of elements in geographic or planar vector
properties	Return property names of geographic or planar vector
rmfield	Remove dynamic property from geographic or planar vector
rmprop	Remove property from geographic or planar vector
size	Return size of geographic or planar vector
struct	Convert geographic or planar vector to scalar structure
vertcat	Vertically concatenate geographic or planar vectors

Examples

Construct a Default Geoshape Vector, Then Add Properties

Construct an empty geoshape vector.

```
s = geoshape()

s =

 0x1 geoshape vector with properties:

 Collection properties:
     Geometry: 'line'
     Metadata: [1x1 struct]
 Vertex properties:
     Latitude: []
    Longitude: []
```

Set the `Latitude` and `Longitude` property values using dot notation.

```
s.Latitude = 0:45:90;
s.Longitude = [10 10 10];
```

Display the updated geoshape vector.

```
s

s =
 1x1 geoshape vector with properties:

 Collection properties:
     Geometry: 'line'
     Metadata: [1x1 struct]
 Vertex properties:
     Latitude: [0 45 90]
    Longitude: [10 10 10]
```

Construct Geoshape Vector Specifying Latitude and Longitude Values

Create a geoshape vector specifying latitude and longitude values as input arguments.

```
s = geoshape([42 43 45],[10 11 15])

s =
 1x1 geoshape vector with properties:

 Collection properties:
     Geometry: 'line'
     Metadata: [1x1 struct]
 Vertex properties:
     Latitude: [42 43 45]
    Longitude: [10 11 15]
```

Construct a Geoshape Vector with Dynamic Properties

Create a geoshape vector using a Name-Value pair to define a new Feature property. This example defines a property called `'Temperature'` and assigns it the value 89.

```
point = geoshape(42, -72, 'Temperature', 89)

point =
 1x1 geoshape vector with properties:

 Collection properties:
       Geometry: 'line'
       Metadata: [1x1 struct]
 Vertex properties:
       Latitude: 42
      Longitude: -72
 Feature properties:
    Temperature: 89
```

To add dynamic properties to a geoshape vector after it has been constructed, use standard dot notation. Add a dynamic property called `'TemperatureUnits'` with the value `'Fahrenheit'`.

```
point.TemperatureUnits = 'Fahrenheit'

point =
 1x1 geoshape vector with properties:

 Collection properties:
            Geometry: 'line'
            Metadata: [1x1 struct]
 Vertex properties:
            Latitude: 42
           Longitude: -72
 Feature properties:
         Temperature: 89
    TemperatureUnits: 'Fahrenheit'
```

To modify properties, use standard dot notation. Update the temperature, and change `'Geometry'` to `'point'`.

```
point.Temperature = 86;
point.Geometry = 'point'

point =
 1x1 geoshape vector with properties:

 Collection properties:
            Geometry: 'point'
            Metadata: [1x1 struct]
 Vertex properties:
            Latitude: 42
           Longitude: -72
 Feature properties:
         Temperature: 86
    TemperatureUnits: 'Fahrenheit'
```

Construct a Geoshape Vector from a Structure Array

This example highlights the two ways by which a geoshape vector with the same features can be created. The first way uses a structure array in the constructor for a geoshape vector. The second way adds fields of the structure array to a geoshape vector after construction.

First, read data into a structure array. The array in this example contains 128 elements. Each element defines a river as a line using multiple location vertices.

```
structArray = shaperead('worldrivers.shp', 'UseGeoCoords', true);
```

Display the first element in `structArray`. Note that the `Lat` and `Lon` vectors are terminated with a `NaN` delimiter, which separates the Vertex feature data in the geoshape class.

```
structArray(1)
```

```
ans = struct with fields:
       Geometry: 'Line'
    BoundingBox: [2x2 double]
            Lon: [126.7796 126.5321 126.3121 126.2383 126.0362 NaN]
            Lat: [73.4571 73.0669 72.8343 72.6010 72.2894 NaN]
           Name: 'Lena'
```

Method 1: Provide the structure as an argument to the constructor that builds the geoshape vector.

Create a geoshape vector, providing the structure array as an argument to the constructor.

```
shape1 = geoshape(structArray)
```

```
shape1 =
 128x1 geoshape vector with properties:

 Collection properties:
     Geometry: 'line'
     Metadata: [1x1 struct]
 Vertex properties:
  (128 features concatenated with 127 delimiters)
     Latitude: [73.4571 73.0669 72.8343 72.6010 72.2894 NaN NaN 72.2894 ... ]
    Longitude: [126.7796 126.5321 126.3121 126.2383 126.0362 NaN NaN ... ]
 Feature properties:
         Name: {1x128 cell}
```

Note that the `BoundingBox` field in structArray does not get assigned to a property in `shape1` because the field value is not a supported type.

Method 2: Add features to a geoshape vector after construction.

Create an empty geoshape vector.

```
shape2 = geoshape;
```

Add the Vertex properties `Latitude` and `Longitude` from each entry in the structure array using dot notation. Add a dynamic Feature property, `RiverName`, the name of the river from each entry in `structArray`. Since the default value of the `Geometry` Collection property is `'line'` there is no need to set it explicitly in this example.

```
shape2.Latitude  = {structArray.Lat};
shape2.Longitude = {structArray.Lon};
shape2.RiverName = {structArray.Name}
```

```
shape2 =
 128x1 geoshape vector with properties:

 Collection properties:
     Geometry: 'line'
     Metadata: [1x1 struct]
 Vertex properties:
  (128 features concatenated with 127 delimiters)
     Latitude: [73.4571 73.0669 72.8343 72.6010 72.2894 NaN NaN 72.2894 ... ]
    Longitude: [126.7796 126.5321 126.3121 126.2383 126.0362 NaN NaN ... ]
 Feature properties:
    RiverName: {1x128 cell}
```

Construct a Geoshape Vector Using Cell Arrays

First, read data into a structure array. The array in this example contains 128 elements. Each element defines a river as a line using multiple location vertices.

```
structArray = shaperead('worldrivers.shp', 'UseGeoCoords', true)
```

```
structArray=128×1 struct array with fields:
    Geometry
    BoundingBox
    Lon
    Lat
    Name
```

Create latitude and longitude vectors. For illustrative purposes, the vectors do not correspond to the elements of `structArray`.

```
lat = {[0:10:40], [1:5]};
lon = {[-60:30:60], [0:2:8]};
```

Construct a geoshape vector using the latitude and longitude vectors and the structure array.

```
s = geoshape(lat,lon,structArray);
```

Display the first three elements of `s`. Features are separated with a `NaN` delimiter.

```
s(1:3)
```

```
ans =
 3x1 geoshape vector with properties:

 Collection properties:
     Geometry: 'line'
     Metadata: [1x1 struct]
 Vertex properties:
  (3 features concatenated with 2 delimiters)
     Latitude: [0 10 20 30 40 NaN 1 2 3 4 5 NaN 0]
    Longitude: [-60 -30 0 30 60 NaN 0 2 4 6 8 NaN 0]
```

```
Feature properties:
         Name: {'Lena'  'Lena'  'Mackenzie'}
```

Observe that `geoshape` uses the arguments `lat` and `lon` to populate the `Latitude` and `Longitude` properties, even though `structArray` provides `Lat` and `Lon` field values. Also, since `lat` and `lon` have fewer elements than features in `structArray`, the `Latitude` and `Longitude` properties expand in size using a value of 0.

Use Indexing to Append a Single Point and a Shape to a Geoshape Vector

Create a geoshape vector containing a single feature of the locations of world cities.

```
S = shaperead('worldcities.shp', 'UseGeoCoords', true);
cities = geoshape([S.Lat], [S.Lon], 'Name', {{S.Name}});
cities.Geometry = 'point';
```

Append Paderborn Germany to the geoshape vector.

```
lat = 51.715254;
lon = 8.75213;
cities(1).Latitude(end+1) = lat;
cities(1).Longitude(end) = lon;
cities(1).Name{end} = 'Paderborn'

cities =

 1x1 geoshape vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
 Vertex properties:
     Latitude: [1x319 double]
    Longitude: [1x319 double]
         Name: {1x319 cell}
```

The length of each vertex property grows by one when `Latitude(end+1)` is set. The remaining properties are indexed with `end`.

You can display the last point by constructing a geopoint vector.

```
paderborn = geopoint(cities.Latitude(end), cities.Longitude(end), ...
   'Name', cities.Name{end})

paderborn =

 1x1 geopoint vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
 Feature properties:
     Latitude: 51.7153
    Longitude: 8.7521
         Name: 'Paderborn'
```

Create a geoshape vector with two new features containing the cities in the northern and southern hemispheres. Add a `Location` dynamic Feature property to distinguish the different classifications.

```
northern = cities(1).Latitude >= 0;
southern = cities(1).Latitude < 0;
index = {northern; southern};
location = {'Northern Hemisphere', 'Southern Hemisphere'};
hemispheres = geoshape();
for k = 1:length(index)
   hemispheres = append(hemispheres, ...
      cities.Latitude(index{k}), cities.Longitude(index{k}), ...
      'Name', {cities.Name(index{k})}, 'Location', location{k});
end
hemispheres.Geometry = 'point'

hemispheres =

 2x1 geoshape vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
 Vertex properties:
  (2 features concatenated with 1 delimiter)
     Latitude: [1x320 double]
    Longitude: [1x320 double]
         Name: {1x320 cell}
 Feature properties:
     Location: {'Northern Hemisphere' 'Southern Hemisphere'}
```

Plot the northern cities in red and the southern cities in blue.

```
hemispheres.Color = {'red', 'blue'};
figure;worldmap('world')
geoshow('landareas.shp')
for k=1:2
   geoshow(hemispheres(k).Latitude, hemispheres(k).Longitude, ...
      'DisplayType', hemispheres.Geometry, ...
      'MarkerEdgeColor', hemispheres(k).Color)
end
```
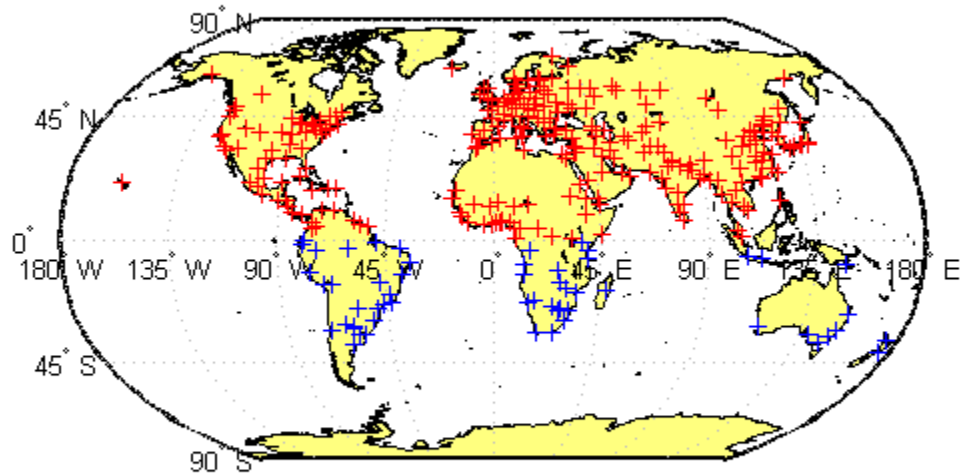
Use Indexing to Sort and Modify Dynamic Features

Construct a geoshape vector and sort its dynamic properties.

```
shape = geoshape(shaperead('tsunamis.shp', 'UseGeoCoords', true));
shape.Geometry = 'point';
shape = shape(:, sort(fieldnames(shape)))
```

```
shape =

 162x1 geoshape vector with properties:

 Collection properties:
       Geometry: 'point'
       Metadata: [1x1 struct]
 Vertex properties:
  (162 features concatenated with 161 delimiters)
       Latitude: [1x323 double]
      Longitude: [1x323 double]
 Feature properties:
          Cause: {1x162 cell}
     Cause_Code: [1x162 double]
        Country: {1x162 cell}
            Day: [1x162 double]
    Desc_Deaths: [1x162 double]
```

```
      Eq_Mag: [1x162 double]
        Hour: [1x162 double]
     Iida_Mag: [1x162 double]
    Intensity: [1x162 double]
     Location: {1x162 cell}
    Max_Height: [1x162 double]
       Minute: [1x162 double]
        Month: [1x162 double]
   Num_Deaths: [1x162 double]
       Second: [1x162 double]
     Val_Code: [1x162 double]
     Validity: {1x162 cell}
         Year: [1x162 double
```

Modify the geoshape vector to contain only the dynamic properties, `Year`, `Month`, `Day`, `Hour`, `Minute`.

```
shape = shape(:, {'Year', 'Month', 'Day', 'Hour', 'Minute'})

shape =

 162x1 geoshape vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
 Vertex properties:
  (162 features concatenated with 161 delimiters)
     Latitude: [1x323 double]
    Longitude: [1x323 double]
 Feature properties:
         Year: [1x162 double]
        Month: [1x162 double]
          Day: [1x162 double]
         Hour: [1x162 double]
       Minute: [1x162 double]
```

Display the first five elements.

```
shape(1:5)

ans =

 5x1 geoshape vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
 Vertex properties:
  (5 features concatenated with 4 delimiters)
     Latitude: [-3.8000 NaN 19.5000 NaN -9.0200 NaN 42.1500 NaN 19.1000]
    Longitude: [128.3000 NaN -156 NaN 157.9500 NaN 143.8500 NaN -155]
 Feature properties:
         Year: [1950 1951 1951 1952 1952]
        Month: [10 8 12 3 3]
          Day: [8 21 22 4 17]
```

```
        Hour: [3 10 NaN 1 3]
      Minute: [23 57 NaN 22 58]
```

Construct a Geoshape Vector from Multiple Objects

Read multiple GPS track log data from a file. `trk1` and `trk2` are geopoint objects.

```
trk1 = gpxread('sample_tracks.gpx')

trk1 =
 1851x1 geopoint vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
 Feature properties:
     Latitude: [42.2995 42.2995 42.2994 42.2994 42.2994 42.2994 42.2994 ... ]
    Longitude: [-71.3502 -71.3504 -71.3504 -71.3504 -71.3504 -71.3504 ... ]
    Elevation: [90.0206 90.0206 90.0206 90.5013 90.0206 90.0206 90.0206 ... ]
         Time: {1x1851 cell}
```

```
trk2 = gpxread('sample_tracks.gpx', 'Index', 2);
```

To construct a geoshape vector with multiple features, place the coordinates into cell arrays.

```
lat = {trk1.Latitude, trk2.Latitude};
lon = {trk1.Longitude, trk2.Longitude};
```

Place the elevation and time values into cell arrays.

```
elevation = {trk1.Elevation, trk2.Elevation};
time = {trk1.Time, trk2.Time};
```

Construct a geoshape vector containing two track log features that include `Elevation` and `Time` as dynamic Vertex properties.

```
tracks = geoshape(lat, lon, 'Elevation', elevation, 'Time', time)

tracks =
 2x1 geoshape vector with properties:

 Collection properties:
     Geometry: 'line'
     Metadata: [1x1 struct]
 Vertex properties:
  (2 features concatenated with 1 delimiter)
     Latitude: [42.2995 42.2995 42.2994 42.2994 42.2994 42.2994 42.2994 ... ]
    Longitude: [-71.3502 -71.3504 -71.3504 -71.3504 -71.3504 -71.3504 ... ]
    Elevation: [90.0206 90.0206 90.0206 90.5013 90.0206 90.0206 90.0206 ... ]
         Time: {1x2591 cell}
```

Each `Latitude` and `Longitude` coordinate pair has associated `Elevation` and `Time` values.

To construct a geoshape vector containing a dynamic Feature property, use an array that is the same length as the coordinate cell array. For example, add a `MaximumElevation` dynamic Feature property.

```
tracks.MaximumElevation = [max(trk1.Elevation) max(trk2.Elevation)]

tracks =
 2x1 geoshape vector with properties:

 Collection properties:
           Geometry: 'line'
           Metadata: [1x1 struct]
 Vertex properties:
  (2 features concatenated with 1 delimiter)
           Latitude: [42.2995 42.2995 42.2994 42.2994 42.2994 42.2994 42.2994 ... ]
          Longitude: [-71.3502 -71.3504 -71.3504 -71.3504 -71.3504 -71.3504 ... ]
          Elevation: [90.0206 90.0206 90.0206 90.5013 90.0206 90.0206 90.0206 ... ]
               Time: {1x2591 cell}
 Feature properties:
   MaximumElevation: [92.4240 76.1000]
```

The Feature property value has only two numeric values, one for each feature.

Store Latitude and Longitude Values in Geoshape Vector

Load coastline data from a MAT-file.

```
load coastlines
```

Create an *n*-by-2 array of coastline latitude and longitude values.

```
pts = [coastlat coastlon];
```

Create a geoshape object and store the latitude and longitude data. If you store latitude and longitude coordinate values in an *n*-by-2 array, `geoshape` assigns the `Latitude` property values to the first **column** and the `Longitude` property values to the second **column**.

```
shape = geoshape();
shape(1) =  pts

shape =
 1x1 geoshape vector with properties:

 Collection properties:
     Geometry: 'line'
     Metadata: [1x1 struct]
 Vertex properties:
     Latitude: [-83.8300 -84.3300 -84.5000 -84.6700 -84.9200 -85.4200 ... ]
    Longitude: [-180 -178 -174 -170 -166.0000 -163 -158 -152 -146 -147 ... ]
```

Note that `Latitude` and `Longitude` are stored are row vectors in the geoshape vector.

Now, create a 2-by-*m* array of coastline latitude and longitude values. Note the semicolon inside the brackets.

```
pts2 = [coastlat; coastlon];
```

Create a geoshape object and store the latitude and longitude data. If you store latitude and longitude coordinate values in a 2-by-*m* array, `geoshape` assigns the `Latitude` property values to the first **row** and the `Longitude` property values to the second **row**.

```
shape2 = geoshape();
shape2(1) = pts2

shape2 =
 1x1 geoshape vector with properties:

 Collection properties:
     Geometry: 'line'
     Metadata: [1x1 struct]
 Vertex properties:
     Latitude: [-83.8300 -84.3300 -84.5000 -84.6700 -84.9200 -85.4200 ... ]
    Longitude: [-83.8300 -84.3300 -84.5000 -84.6700 -84.9200 -85.4200 ... ]
```

More About

Collection Properties

Collection properties contain only one value per class instance. In contrast, the Feature and Vertex property types have attribute values associated with each feature or with each vertex in a set that defines a feature. `Geometry` and `Metadata` are the only two Collection properties.

Vertex Properties

Vertex properties provide a scalar number or a character vector for each vertex in a geoshape object. Vertex properties are suitable for attributes that vary spatially from point to point (vertex to vertex) along a line. Examples of such spatially varying attributes could be elevation, speed, temperature, or time. `Latitude` and `Longitude` are vertex properties since they contain a scalar number for each vertex in a geoshape vector.

Attribute values are associated with each vertex during construction or by using dot notation after construction. This process is similar to adding dynamic fields to a structure. Dynamic Vertex property values of an individual feature match its `Latitude` and `Longitude` values in length.

Feature Properties

Feature properties provide one value (a scalar number, scalar string, or character vector) for each feature in a geoshape vector. They are suitable for properties, such as name, owner, serial number, or age, that describe a given feature (an element of a geoshape vector) as a whole. Like Vertex properties, Feature properties can be added during construction or by using dot notation after construction.

Tips

- The `geoshape` function separates features using `NaN` values. If you display a feature by using a scalar to index into the geoshape vector, such as `s(1)`, then `NaN` values that separate the features do not display.

- If `Latitude`, `Longitude`, or a dynamic property is set with more values than features in the geoshape vector, then all other properties expand in size using 0 for numeric values and an empty character vector (`' '`) for cell values.
- If a dynamic property is set with fewer values than the number of features, then this dynamic property expands to match the size of the other properties, by inserting a 0 if the value is numeric or an empty character vector (`' '`), if the value is a cell array.
- If the `Latitude` or `Longitude` property of the geoshape vector is set with fewer values than contained in the object, then all other properties shrink in size.
- If either `Latitude` or `Longitude` are set to [ ], then both coordinate properties are set to [ ] and all dynamic properties are removed.
- If a dynamic property is set to [ ], then it is removed from the object.
- The geoshape vector can be indexed like any MATLAB vector. You can access any element of the vector to obtain a specific feature. The following examples demonstrate this behavior:

  "Use Indexing to Append a Single Point and a Shape to a Geoshape Vector" on page 1-591

  "Use Indexing to Sort and Modify Dynamic Features" on page 1-593

  "Construct a Geoshape Vector from Multiple Objects" on page 1-595

See Also

Functions
`gpxread` | `shaperead`

Objects
`geopoint` | `mappoint` | `mapshape`

Topics
"Create and Display Polygons"

Introduced in R2012a

geoshow

Display map latitude and longitude data

Syntax

```
geoshow(lat,lon)
geoshow(S)

geoshow(lat,lon,Z)
geoshow(Z,R)

geoshow(lat,lon,I)
geoshow(lat,lon,X,cmap)
geoshow(I,R)
geoshow(X,cmap,R)

geoshow(filename)

geoshow( ___ ,Name,Value)
geoshow(ax, ___ )
h = geoshow( ___ )
```

Description

geoshow(lat,lon) projects and displays the latitude and longitude vectors lat and lon using the projection stored in the current set of map axes. If there are no current map axes, then lat and lon are projected using a default Plate Carrée projection on a set of regular axes.

Note To display data on a set of map axes, create a map using the axesm, worldmap, or usamap functions before calling geoshow.

By default, geoshow displays lat and lon as lines. You can optionally display the vector data as points, multipoints, or polygons by using the DisplayType name-value pair argument.

geoshow(S) displays the vector geographic features stored in S as points, multipoints, lines, multilines, polygons, or multipolygons according to the geometry of S.

You can optionally specify symbolization rules using the SymbolSpec name-value pair argument.

geoshow(lat,lon,Z) projects and displays the geolocated data grid, Z. In this syntax, lat and lon are M-by-N latitude-longitude arrays. Z is an M-by-N array of class double. You can optionally display the data as a surface, mesh, texture map, or contour by using the DisplayType name-value pair argument.

geoshow(Z,R) projects and displays a regular data grid, Z, with referencing object R. You can optionally display the data as a surface, mesh, texture map, or contour by using the DisplayType name-value pair argument. If DisplayType is 'texturemap', then geoshow constructs a surface with ZData values set to 0.

geoshow(lat,lon,I) and

geoshow(lat,lon,X,cmap) projects and displays a geolocated image as a texture map on a zero-elevation surface. The geolocated image I can be a truecolor, grayscale, or binary image. X is an indexed image with colormap cmap. Examples of geolocated images include a color composite from a satellite swath or an image originally referenced to a different coordinate system.

geoshow(I,R) and

geoshow(X,cmap,R) project and display an image georeferenced to latitude-longitude through the referencing object R. The image is shown as a texture map on a zero-elevation surface.

geoshow(filename) projects and displays data from the file specified according to the type of file format.

geoshow(___ ,Name,Value) specifies parameters and corresponding values that modify the type of display or set MATLAB graphics properties. You can use name,value pairs to set:

- Name,Value arguments
- Any MATLAB Graphics line, patch, and surface properties
- Any Mapping Toolbox contour properties

Parameter names can be abbreviated, and case does not matter.

geoshow(ax, ___) sets the parent axes to ax.

h = geoshow(___) returns a handle to a MATLAB graphics object.

Examples

Display Regular Data Grid as Texture Map

Load elevation data and a geographic cells reference object for the Korean peninsula. Create a set of map axes for the Korean peninsula using worldmap.

```
load korea5c
worldmap(korea5c,korea5cR)
```

Display the elevation data as a texture map. Apply a colormap appropriate for elevation data using demcmap.

```
geoshow(korea5c,korea5cR,'DisplayType','texturemap')
demcmap(korea5c)
```
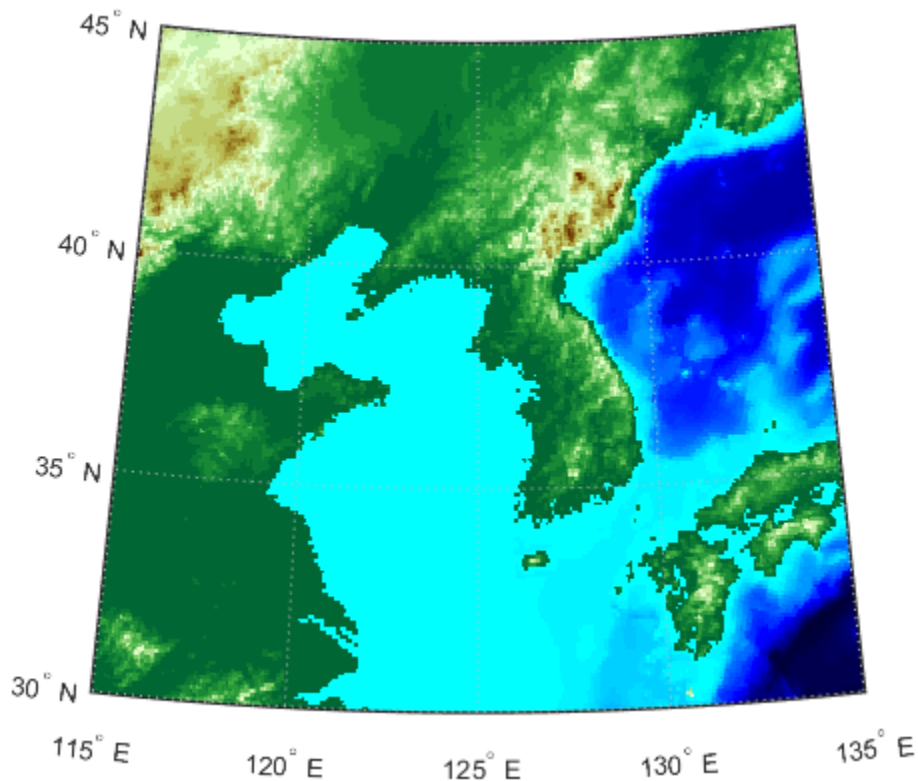
Display Polygons on Map Projection

Import a shapefile containing worldwide land areas as a geospatial table. The shapefile represents the land areas using polygons.

```
landAreas = readgeotable("landareas.shp");
```

Create a subtable that contains the polygon representing Africa, Europe, and Asia. Get information about the polygon by querying the Shape variable of the table.

```
row = landAreas.Name == "Africa and Eurasia";
landAreasSubset = landAreas(row,:);
landAreasSubset.Shape
```

```
ans =
  geopolyshape with properties:

             NumRegions: 1
               NumHoles: 22
               Geometry: "polygon"
    CoordinateSystemType: "geographic"
           GeographicCRS: [1x1 geocrs]
```

Display the polygon on a world map.

```
worldmap([-45 80],[-25 195]);
geoshow(landAreasSubset)
```
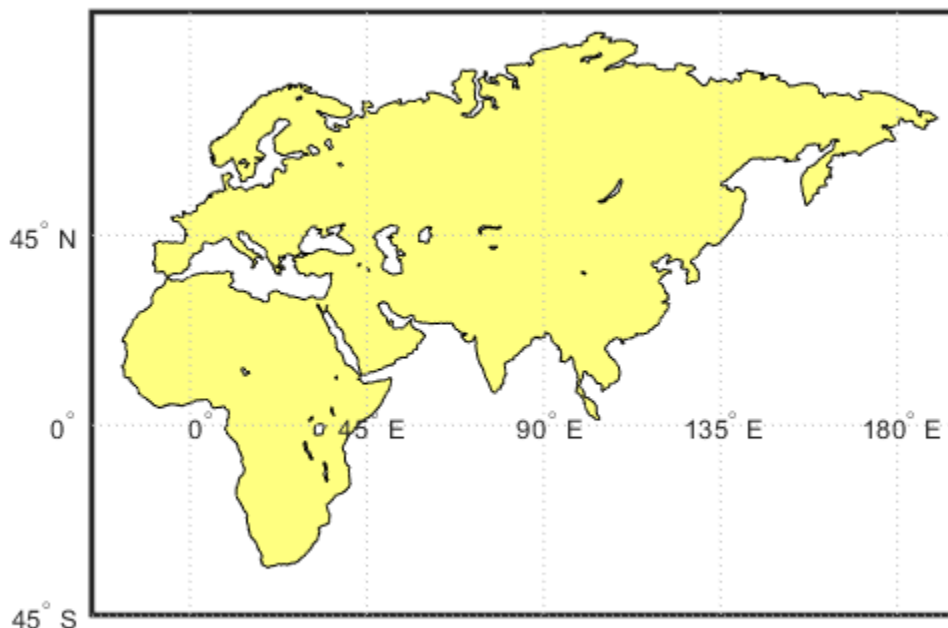

Define Face Colors and Set Default Face Colors

Import a shapefile containing USA state polygons as a geospatial table. Set up an empty map axes with projection and limits suitable for displaying all 50 states.

```
states = readgeotable("usastatehi.shp");
figure
worldmap("na")
```

Create a symbol specification that sets the color of the Alaska and Hawaii polygons to red.

```
symspec = makesymbolspec("Polygon", ...
    {'Name','Alaska','FaceColor','red'}, ...
    {'Name','Hawaii','FaceColor','red'});
```

Display all the state polygons. Set the default face color to blue and the default edge color to black.

```
geoshow(states,"SymbolSpec",symspec, ...
    "DefaultFaceColor","blue", ...
    "DefaultEdgeColor","black");
```
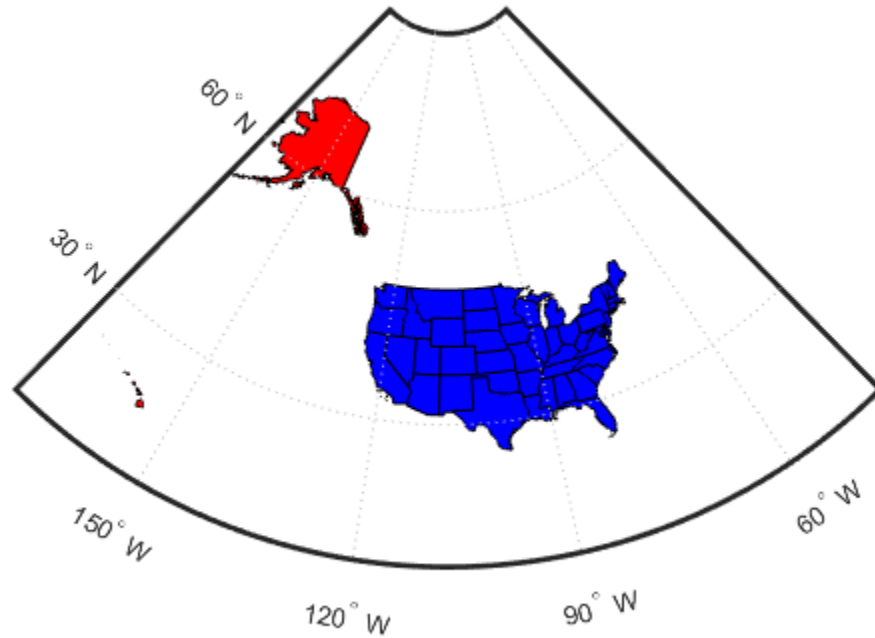
Create Map and Display NaNs as Transparent

Load elevation data and a geographic cells reference object for the Korean peninsula. Insert a band of null values into the elevation data.

```
load korea5c
korea5c(80:100,:) = NaN;
```

Create a set of map axes for the Korean peninsula using `worldmap`. Then, display the elevation data as a surface with transparent null values.

```
worldmap(korea5c,korea5cR)
geoshow(korea5c,korea5cR,'DisplayType','surface')
demcmap(korea5c)
```
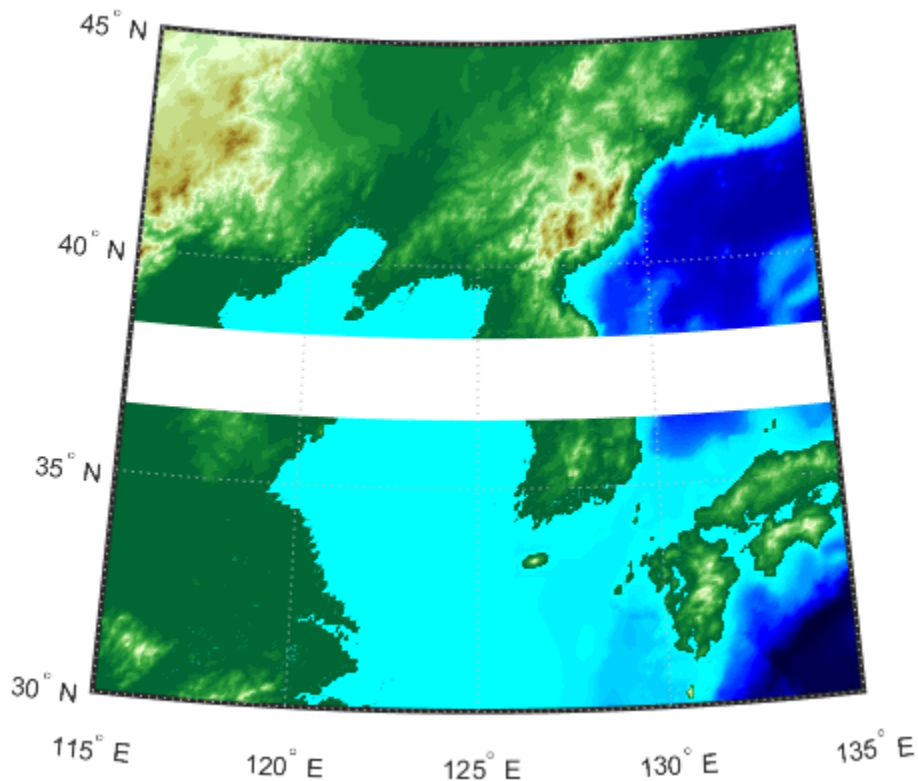
Display EGM96 Geoid Heights Masking Out Land Areas

Get geoid heights and a geographic postings reference object from the EGM96 geoid model. Then, display the geoid heights as a surface using an Eckert projection. Ensure the surface appears under the land mask by setting the `'CData'` name-value pair to the geoid height data and the `'ZData'` name-value pair to a matrix of zeros. Display the frame and grid of the map using `framem` and `gridm`.

```
[N,R] = egm96geoid;
axesm eckert4
Z = zeros(R.RasterSize);
geoshow(N,R,'DisplayType','surface','CData',N,'ZData',Z)
framem
gridm
```

Create a colorbar and add a text description. Then, mask out all the land.

```
cb = colorbar('southoutside');
cb.Label.String = 'EGM96 Geoid Height in Meters';
geoshow('landareas.shp','FaceColor','black')
```

EGM96 Geoid Height in Meters
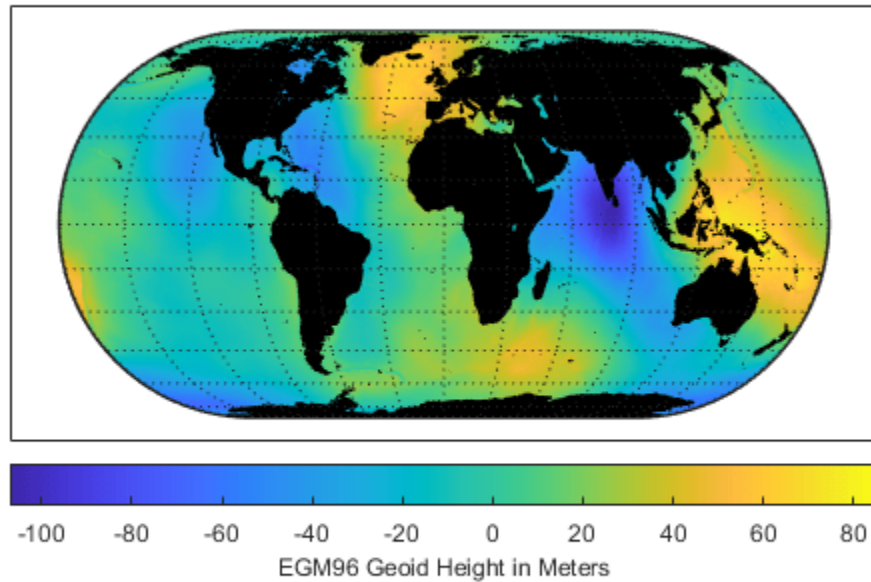
Display EGM96 Geoid Heights as 3-D Surface

Get geoid heights and a geographic postings reference object from the EGM96 geoid model. Then, display the geoid heights as a surface using an Eckert projection.

```
[N,R] = egm96geoid;
axesm eckert4
geoshow(N,R,'DisplayType','surface')
```
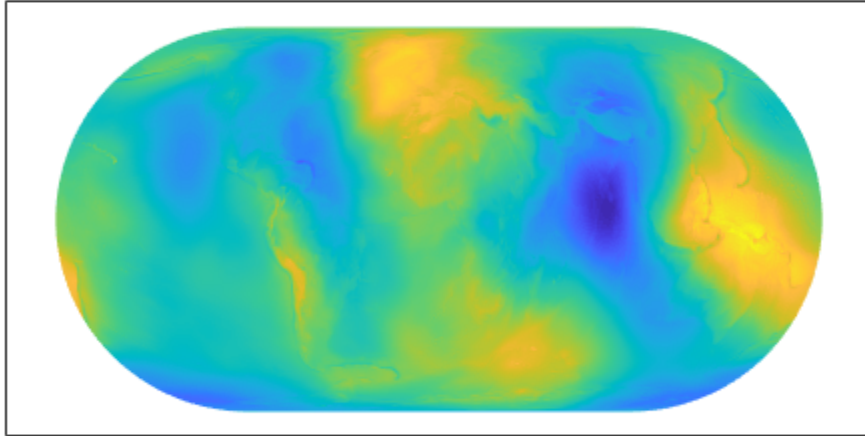
Add light and material. Then, view the map as a 3-D surface.

```
light
material(0.6*[1 1 1])
axis normal
view(3)
```
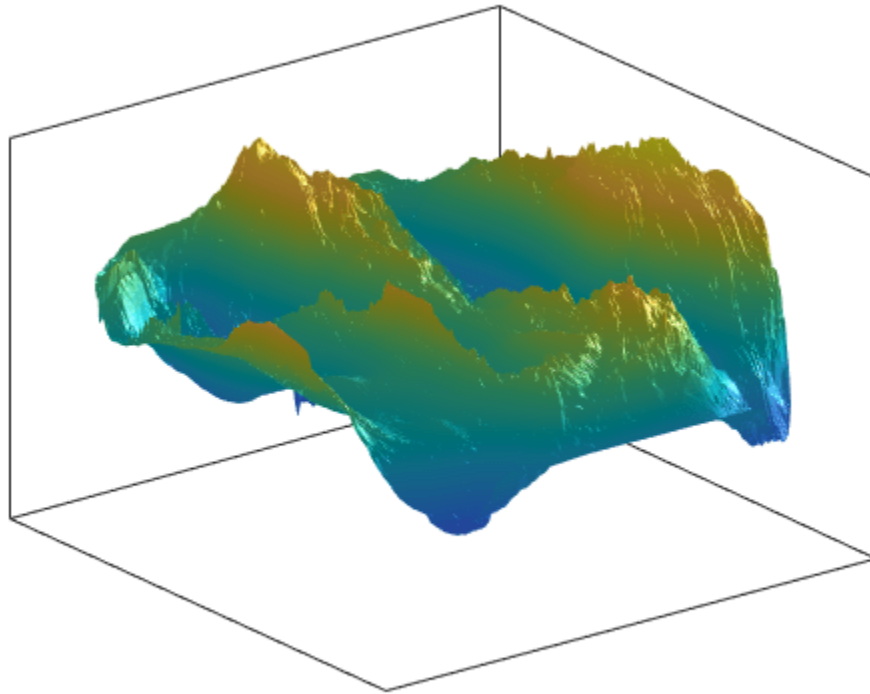
Display Moon Albedo Using Orthographic Projection

Load moon albedo data and a geographic cells reference object.

```
load moonalb20c
```

Then, display the data. To do this, create a map axes object and specify its projection as orthographic. Display the data in the map axes as a texture map using the geoshow function. Then, change the colormap to grayscale and remove the axis lines.

```
axesm ortho
geoshow(moonalb20c,moonalb20cR,'DisplayType','texturemap')
colormap gray
axis off
```

Input Arguments

`lat, lon` — Latitude or longitude data
numeric vector | *M*-by-*N* numeric array

Latitude or longitude data, specified as a numeric vector or an *M*-by-*N* numeric matrix.

- `lat` and `lon` are vectors when used with the syntax `geoshow(lat,lon)`.
- `lat` and `lon` are 2-D arrays when used with the `geoshow(lat,lon,Z)` syntax, the `geoshow(lat,lon,I)` syntax, or the `geoshow(lat,lon,X,cmap)` syntax. If `lat` and `lon` are matrices, they represent coordinate arrays or a geolocation array in geographic coordinates, and must be the same size as `Z`, `I`, or `X`. If `I` is an RGB image, `lat` and `lon` must be matrices that match the first two dimensions of the image.

`lat` and `lon` may contain embedded `NaN`s to delimit individual lines or polygon parts.

`S` — Geographic features
geospatial table | `geopoint` vector | `geoshape` vector | geographic structure array

Geographic features, specified as one of the following:

- A geospatial table. The `Shape` variable of the table must contain `geopointshape`, `geolineshape`, or `geopolyshape` objects. The `geoshow` function does not support geospatial tables containing more than one type of shape object. For more information about geospatial tables, see "Create Geospatial Tables".

- A `geopoint` vector.
- A `geoshape` vector.
- A geographic structure array (`geostruct`) with `Lat` and `Lon` coordinate fields.

The `geoshow` function assumes the coordinates are geographic coordinates and projects them to map coordinates.

If you specify a `mappoint` vector, a `mapshape` vector, or a map structure array (`mapstruct`) with `X` and `Y` coordinate fields, then the `geoshow` function display the vertices as pre-projected map coordinates and issues a warning. If you specify a geospatial table containing `mappointshape`, `maplineshape`, or `mapolyshape` objects, then the `geoshow` function throws an error. In these cases, use the `mapshow` function instead.

Z — Data grid
M-by-*N* numeric array

Data grid, specified as an *M*-by-*N* numeric array that may contain `NaN` values. `Z` is either a georeferenced data grid, or a regular data grid associated with a geographic reference `R`.

R — Geographic reference
geographic raster reference object | vector | matrix

Geographic reference, specified as one of the following. For more information about referencing vectors and matrices, see "Georeferenced Raster Data".

Type	Description
Geographic raster reference object	`GeographicCellsReference` or `GeographicPostingsReference` geographic raster reference object that relates the subscripts of `Z` to geographic coordinates. The `RasterSize` property must be consistent with the size of the data grid, `size(Z)`. If `R` is a `GeographicPostingsReference` object, then the `'image'` and `'texturemap'` values of `DisplayType` are not accepted.
Vector	1-by-3 numeric vector with elements: `[cells/degree northern_latitude_limit western_longitude_limit]`
Matrix	3-by-2 numeric matrix that transforms raster row and column indices to or from geographic coordinates according to: `[lon lat] = [row col 1] * R` `R` defines a (non-rotational, non-skewed) relationship in which each column of the data grid falls along a meridian and each row falls along a parallel. Nearest-neighbor interpolation is used by default. `NaN` is returned for points outside the grid limits or for which `lat` or `lon` contain `NaN`. All angles are in units of degrees.

I — Truecolor, grayscale, or binary image
M-by-*N*-by-3 array | *M*-by-*N* array

Truecolor, grayscale, or binary image, specified as an *M*-by-*N*-by-3 array for truecolor images, or an *M*-by-*N* array for grayscale or binary images. `lat` and `lon` must be *M*-by-*N* arrays. If specified, `'DisplayType'` must be set to `'image'`.

X — Indexed image
M-by-*N* array

Indexed image with color map defined by `cmap`, specified as an *M*-by-*N* array. `lat` and `lon` must be *M*-by-*N* arrays. If specified, `'DisplayType'` must be set to `'image'`.

cmap — Color map
c-by-3 matrix

Color map of indexed image X, specified as an *c*-by-3 numeric matrix. There are *c* colors in the color map, each represented by a red, green, and blue pixel value.

filename — File name
character vector | string scalar

File name, specified as a string scalar or character vector. `geoshow` automatically sets the `DisplayType` parameter according to the format of the data.

Format	DisplayType
Shape file	`'point'`, `'multipoint'`, `'line'`, or `'polygon'`
GeoTIFF	`'image'`
TIFF/JPEG/PNG with a world file	`'image'`
ARC ASCII GRID	`'surface'` (can be overridden)
SDTS raster	`'surface'` (can be overridden)

Data Types: `char` | `string`

ax — Parent axes
axes object

Parent axes, specified as an axes object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'DisplayType','point'`

SymbolSpec — Symbolization rules
structure

Symbolization rules to be used for displaying vector data, specified as the comma-separated pair consisting of `'SymbolSpec'` and a structure returned by `makesymbolspec`. It is used only for vector data stored in geographic data structures. In cases where both `SymbolSpec` and one or more graphics properties are specified, the graphics properties override any settings in the symbolspec structure.

To change the default symbolization rule for a `Name,Value` pair in the `SymbolSpec` structure, prefix the word `'Default'` to the graphics property name.

DisplayType — Display type
`'point'` | `'multipoint'` | `'line'` | `'polygon'` | `'image'` | `'surface'` | `'mesh'` | `'texturemap'` | `'contour'`

Type of graphic display for the data, specified as the comma-separated pair consisting of `'DisplayType'` and one of the following values.

Data Type	Display Type	Type of Property
Vector	`'point'`	*line marker*
	`'multipoint'`	*line marker*
	`'line'`	*line*
	`'polygon'`	*patch*
Image	`'image'`	*surface*
Grid	`'surface'`	*surface*
	`'mesh'`	*surface*
	`'texturemap'`	*surface*
	`'contour'`	*contour*

Valid values of `DisplayType` depend on the format of the map data. For example, if the map data is a geolocated image or georeferenced image, then the only valid value of `DisplayType` is `'image'`. Different display types support different geographic data class types:

Display Type	Supported Class Types
Image	`logical`, `uint8`, `uint16`, and `double`
Surface	`single` and `double`
Texture map	All numeric types and `logical`

Output Arguments

h — Parent axes
handle object | modified patch object

Parent axes, returned as a handle to a MATLAB graphics object or, in the case of polygons, a modified patch object. If a geostruct or shapefile name is input, `geoshow` returns the handle to an `hggroup` object with one child per feature in the geostruct or shapefile, excluding any features that are completely trimmed away. In the case of a polygon geostruct or shapefile, each child is a modified patch object; otherwise it is a line object.

Tips

• When calling `shaperead` to read files that contain coordinates in latitude and longitude, be sure to specify the `shaperead` argument pair `'UseGeoCoords',true`. If you do not include this argument, `shaperead` will create a mapstruct, with coordinate fields labelled X and Y instead of `Lon` and `Lat`. In such cases, `geoshow` assumes that the geostruct is in fact a mapstruct containing

projected coordinates, warns, and calls `mapshow` to display the geostruct data without projecting it.

- If you do not want `geoshow` to draw on top of an existing map, create a new figure or subplot before calling it.

- When you display vector data in a map axes using `geoshow`, you should not subsequently change the map projection using `setm`. You can, however, change the projection with `setm` for raster data. For more information, see "Change Map Projections Using geoshow".

- If you display a polygon, do not set `'EdgeColor'` to either `'flat'` or `'interp'`. This combination may result in a warning.

- When projecting data onto a map axes, `geoshow` uses the projection stored with the map axes. When displaying on a regular axes, it constructs a default Plate Carrée projection with a scale factor of `180/pi`, enabling direct readout of coordinates in degrees.

- When you pass two points to the `geoshow` function, the function projects the locations of each point on the map and displays a straight line that connects them, regardless of the map projection or the distance between the points. To display a projected geodesic or rhumbline curve between two points, first insert intermediate points using the `track2` function.

- `geoshow` can generally be substituted for `displaym`. However, there are limitations where display of specific objects is concerned. See the remarks under `updategeostruct` for further information.

- When you display raster data in a map using `geoshow`, columns near the eastern or western edge may fail to display. This is seldom noticeable, except when the raster is very coarse relative to the displayed area. To include additional columns in the display, it might help to:

  - Resize the grid to a finer mesh.
  - Make sure the cell boundaries and map limits align.
  - Expand the map limits.

See Also

Functions
axesm | makesymbolspec | mapview | mapshow | updategeostruct

Objects
GeographicCellsReference | GeographicPostingsReference

Topics
"Create and Display Polygons"

Introduced before R2006a

geotable2table

Convert geospatial table to table

Syntax

```
T = geotable2table(GT)
T = geotable2table(GT,varnames)
```

Description

`T = geotable2table(GT)` converts the geospatial table `GT` to a table. The function replaces the objects in the `Shape` variable of GT with well-known text (WKT) string representations of geometry. The geospatial table and the table have the same dimensions.

`T = geotable2table(GT,varnames)` replaces the `Shape` variable with variables that have names specified by `varnames`.

Examples

Convert Geospatial Table to Table

Convert a geospatial table to a table, then write the data to an Excel® spreadsheet file.

Import a shapefile containing the coordinates of world cities as a geospatial table.

```
GT = readgeotable("worldcities.shp");
```

Find the coordinate reference system type. The coordinate reference system type helps you determine appropriate variable names for the table.

```
GT.Shape.CoordinateSystemType
```

```
ans =
"geographic"
```

Convert the geospatial table to a table. Replace the `Shape` variable with two variables containing coordinates by specifying the second argument as a two-element vector. The coordinate system is geographic, so use the variable names `"Latitude"` and `"Longitude"`.

```
T = geotable2table(GT,["Latitude" "Longitude"]);
```

Write the table to an Excel spreadsheet file.

```
writetable(T,"worldcities.xlsx")
```

View WKT String Representations of Geometry

Import a shapefile containing a road network in Boston, MA, as a geospatial table. The `Shape` variable of the geospatial table contains `maplineshape` objects.

```
GT = readgeotable("boston_roads.shp");
class(GT.Shape)

ans =
'maplineshape'
```

Convert the geospatial table to a table. The `geotable2table` function replaces the objects in the `Shape` variable with WKT string representations of geometry.

```
T = geotable2table(GT);
```

View the WKT string for the road called `C STREET`.

```
wkt = T.Shape(T.STREETNAME == "C STREET")

wkt =
"LINESTRING (237527.171875 899661.1875,237343.875 899443.0625)"
```

Input Arguments

GT — Geospatial table
geospatial table

Geospatial table. A geospatial table is a `table` or `timetable` object with a `Shape` variable that contains `geopointshape`, `geolineshape`, `geopolyshape`, `mappointshape`, `maplineshape`, or `mappolyshape` objects. The `Shape` variable must be the first variable.

For more information about geospatial tables, see "Create Geospatial Tables".

Data Types: `table`

varnames — Table variable names
string scalar | two-element string vector

Table variable names used to replace the `Shape` variable of the geospatial table, specified as a string scalar or two-element string vector.

- To replace the shape objects in the `Shape` variable with WKT string representations of geometry, specify `varnames` as a string scalar, such as `"WKTStrings"`. Unlike shape objects, WKT string representations of geometry do not contain information about the coordinate reference system.

- To replace the `Shape` variable with two table variables containing latitude and longitude coordinates or *x*- and *y*-coordinates, specify `varnames` as a two-element string vector, such as `["Latitude" "Longitude"]`.

The `geotable2table` function does not project or unproject coordinates. For more information about projecting or unprojecting coordinates, see `projfwd` or `projinv`.

Data Types: `string`

Output Arguments

T — Output table
table object | timetable object

Output table, returned as a `table` or `timetable` object.

- When you specify `varnames` as a string scalar, the output table contains WKT string representations of geometry.

- When you specify `varnames` as a two-element string vector, the output table contains latitude-longitude or *x*- *y* coordinates. If a table row represents an individual point, then the coordinates are returned as numeric scalars. Otherwise, the coordinates are returned as cell arrays of numeric vectors.

Data Types: `table`

See Also

Functions
isgeotable | readgeotable | table2geotable | table2timetable | polyjoin

Introduced in R2021b

map.geotiff.RPCCoefficientTag

Create a Rational Polynomial Coefficients Tag object

Syntax

```
rpctag = map.geotiff.RPCCoefficientTag
rpctag = map.geotiff.RPCCoefficientTag(tiffTagValue)
```

Description

`rpctag = map.geotiff.RPCCoefficientTag` creates a default `RPCCoefficientTag` object.

`rpctag = map.geotiff.RPCCoefficientTag(tiffTagValue)` creates an `RPCCoefficientTag` object and sets the property values to the corresponding values in the 92-element vector specified in `tiffTagValue`.

Examples

Create RPCCoefficientTag Object with Default Properties

Call the RPCCoefficientTag class constructor with no arguments.

```
rpctag = map.geotiff.RPCCoefficientTag

rpctag =
  RPCCoefficientTag with properties:

              BiasErrorInMeters: -1
            RandomErrorInMeters: -1
                     LineOffset: 0
                   SampleOffset: 0
          GeodeticLatitudeOffset: 0
         GeodeticLongitudeOffset: 0
            GeodeticHeightOffset: 0
                      LineScale: 1
                    SampleScale: 1
           GeodeticLatitudeScale: 1
          GeodeticLongitudeScale: 1
             GeodeticHeightScale: 1
        LineNumeratorCoefficients: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
      LineDenominatorCoefficients: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
      SampleNumeratorCoefficients: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
    SampleDenominatorCoefficients: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

Write Raw RPC Coefficient Metadata to GeoTIFF File

This example shows how to write RPC coefficient metadata to a TIFF file. In a real workflow, you would create the RPC coefficient metadata according to the TIFF extension specification. This

example does not show the specifics of how to create valid RPC metadata. To simulate raw RPC metadata, the example creates a sample TIFF file with RPC metadata and then uses `imfinfo` to read this RPC metadata in raw, unprocessed form from the file. The example then writes this raw RPC metadata to a file using the `geotiffwrite` function.

Create Raw RPC Coefficient Metadata

To simulate raw RPC metadata, create a simple test file and write some RPC metadata to the file. For this test file, create a toy image and a referencing object associated with the image.

```
myimage = zeros(180,360);
latlim = [-90 90];
lonlim = [-180 180];
R = georefcells(latlim,lonlim,size(myimage));
```

Create an `RPCCoefficientTag` metadata object and set some of the fields. The toolbox uses the `RPCCoefficientTag` object to represent RPC metadata in human readable form.

```
rpctag = map.geotiff.RPCCoefficientTag;
rpctag.LineOffset = 1;
rpctag.SampleOffset = 1;
rpctag.LineScale = 2;
rpctag.SampleScale = 2;
rpctag.GeodeticHeightScale = 500;
```

Write the image, the associated referencing object, and the `RPCCoefficientTag` object to a file.

```
geotiffwrite('myfile',myimage,R,'RPCCoefficientTag',rpctag)
```

Read Raw RPC Coefficient Metadata

Read the RPC coefficient metadata from the test file using the `imfinfo` function. When it encounters unfamiliar metadata, `imfinfo` returns the data, unprocessed, in the `UnknownTags` field. Note that the `UnknownTags` field contains an array of 92 doubles. This is the raw RPC coefficient metadata, read from the file in unprocessed form.

```
info = imfinfo('myfile.tif');
info.UnknownTags

ans = struct with fields:
        ID: 50844
    Offset: 10676
     Value: [-1 -1 1 1 0 0 0 2 2 1 1 500 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ... ]
```

Write Raw RPC Metadata to a File

Write the raw RPC metadata to a file. First, extract the RPC coefficient metadata from the info structure.

```
value = info.UnknownTags.Value;
```

Then, construct an `RPCCoefficientTag` object, passing the raw RPC metadata (array of 92 doubles) as an argument.

```
rpcdata = map.geotiff.RPCCoefficientTag(value)

rpcdata =
  RPCCoefficientTag with properties:
```

```
                BiasErrorInMeters: -1
              RandomErrorInMeters: -1
                       LineOffset: 1
                     SampleOffset: 1
           GeodeticLatitudeOffset: 0
          GeodeticLongitudeOffset: 0
             GeodeticHeightOffset: 0
                        LineScale: 2
                      SampleScale: 2
            GeodeticLatitudeScale: 1
           GeodeticLongitudeScale: 1
              GeodeticHeightScale: 500
        LineNumeratorCoefficients: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
      LineDenominatorCoefficients: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
      SampleNumeratorCoefficients: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
    SampleDenominatorCoefficients: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

Pass the `RPCCoefficientTag` object to the `geotiffwrite` function and write the RPC metadata to a file.

```
geotiffwrite('myfile2',myimage,R,'RPCCoefficientTag',rpcdata)
```

To verify that the data was written to the file, read the RPC metadata from the TIFF file using `geotiffinfo`. Compare the returned RPC metadata with the metadata written to the test file.

```
ginfo = geotiffinfo('myfile2');
ginfo.GeoTIFFTags.RPCCoefficientTag
```

```
ans =
  RPCCoefficientTag with properties:

                BiasErrorInMeters: -1
              RandomErrorInMeters: -1
                       LineOffset: 1
                     SampleOffset: 1
           GeodeticLatitudeOffset: 0
          GeodeticLongitudeOffset: 0
             GeodeticHeightOffset: 0
                        LineScale: 2
                      SampleScale: 2
            GeodeticLatitudeScale: 1
           GeodeticLongitudeScale: 1
              GeodeticHeightScale: 500
        LineNumeratorCoefficients: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
      LineDenominatorCoefficients: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
      SampleNumeratorCoefficients: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
    SampleDenominatorCoefficients: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

Input Arguments

`tiffTagValue` — Value of `RPCCoefficientTag` properties
92-element vector

Value of `RPCCoefficientTag` properties, specified as a 92-element vector.

Data Types: `double`

See Also

RPCCoefficientTag | geotiffinfo | geotiffwrite | Tiff

Introduced in R2015b

geotiff2mstruct

Convert GeoTIFF information to map projection structure

Syntax

```
mstruct = geotiff2mstruct(proj)
```

Description

`mstruct = geotiff2mstruct(proj)` converts the GeoTIFF projection structure, `proj`, to the map projection structure, `mstruct`. The unit of length of the `mstruct` projection is meter.

The GeoTIFF projection structure, `proj`, must reference a projected coordinate system, as indicated by a value of `'ModelTypeProjected'` in the `ModelType` field. If `ModelType` has the value `'ModelTypeGeographic'` then it doesn't make sense to convert to a map projection structure and an error is issued.

Examples

Verify that unprojecting coordinates using a GeoTIFF projection structure gives the same result as unprojecting them using a map projection structure.

To do this, first get the GeoTIFF projection structure of an image. Convert the corner map coordinates to latitude and longitude by calling `projinv` and specifying the GeoTIFF projection structure.

```
proj = geotiffinfo('boston.tif');
x = proj.CornerCoords.X;
y = proj.CornerCoords.Y;
[latProj,lonProj] = projinv(proj,x,y);
```

Get a map projection structure from the GeoTIFF projection structure using the `geotiff2mstruct` function. The length unit for map projection structures is meter, but the map coordinates are in survey feet. Therefore, convert the corner map coordinates from survey feet to meters. Then, unproject the corner coordinates by calling `projinv` and specifying the map projection structure.

```
mstruct = geotiff2mstruct(proj);
xsf = unitsratio('meter','sf') * x;
ysf = unitsratio('meter','sf') * y;
[latMstruct,lonMstruct] = projinv(mstruct,xsf,ysf);
```

Verify the values are within a tolerance of each other.

```
abs(latProj - latMstruct) <= 1e-7
abs(lonProj - lonMstruct) <= 1e-7


ans =
     1     1     1     1
```

```
ans =
     1     1     1     1
```

See Also
axesm | defaultm | geotiffinfo | projfwd | projinv | projlist

Introduced before R2006a

geotiffinfo

Information about GeoTIFF file

Syntax

```
info = geotiffinfo(filename)
info = geotiffinfo(url)
```

Description

`info = geotiffinfo(filename)` returns a structure whose fields contain image properties and cartographic information about a GeoTIFF file.

`info = geotiffinfo(url)` reads the GeoTIFF image from a URL.

Examples

Return Information about GeoTIFF File

Return information about a GeoTIFF file as a structure by using the `geotiffinfo` function.

```
info = geotiffinfo('boston.tif')
```

```
info = struct with fields:
               Filename: 'C:\TEMP\Bdoc21b_1757077_3096\ib2EDA31\11\tp8127df5f\map-ex98657947\bostor
            FileModDate: '13-May-2011 22:28:45'
               FileSize: 38729900
                 Format: 'tif'
          FormatVersion: []
                 Height: 2881
                  Width: 4481
               BitDepth: 8
              ColorType: 'truecolor'
              ModelType: 'ModelTypeProjected'
                    PCS: 'NAD83 / Massachusetts Mainland'
             Projection: 'SPCS83 Massachusetts Mainland zone (meters)'
                  MapSys: 'STATE_PLANE_83'
                   Zone: 2001
            CTProjection: 'CT_LambertConfConic_2SP'
               ProjParm: [7x1 double]
             ProjParmId: {7x1 cell}
                    GCS: 'NAD83'
                  Datum: 'North American Datum 1983'
              Ellipsoid: 'GRS 1980'
              SemiMajor: 6378137
              SemiMinor: 6.3568e+06
                     PM: 'Greenwich'
       PMLongToGreenwich: 0
              UOMLength: 'US survey foot'
        UOMLengthInMeters: 0.3048
               UOMAngle: 'degree'
```

```
  UOMAngleInDegrees: 1
           TiePoints: [1x1 struct]
          PixelScale: [3x1 double]
          SpatialRef: [1x1 map.rasterref.MapCellsReference]
           RefMatrix: [3x2 double]
         BoundingBox: [2x2 double]
        CornerCoords: [1x1 struct]
        GeoTIFFCodes: [1x1 struct]
         GeoTIFFTags: [1x1 struct]
    ImageDescription: '"GeoEye"'
```

Input Arguments

`filename` — Name of GeoTIFF file
character vector

Name of the GeoTIFF file, specified as a character vector. Include the folder name in `filename` or place the file in the current folder or in a folder on the MATLAB path. If the named file includes the extension `.TIF` or `.TIFF` (either upper- or lowercase), you can omit the extension from `filename`.

If the named file contains multiple GeoTIFF images, `info` is a structure array with one element for each image in the file. For example, `info(3)` would contain information about the third image in the file. If multiple images exist in the file, it is assumed that each image has the same cartographic information and image width and height.

`url` — Internet URL
character vector

Internet URL, specified as a character vector. The URL must include the protocol type (e.g., "http://").

Output Arguments

`info` — Image properties and cartographic information about a GeoTIFF file
structure

Image properties and cartographic information about a GeoTIFF file, returned as a structure containing the following fields.

Field	Description
Filename	Name of the file or URL
FileModDate	Modification date of the file
FileSize	Integer indicating the size of the file in bytes.
Format	File format (always `'tiff'`)
FormatVersion	File format version
Height	Integer indicating the height of the image in pixels
Width	Integer indicating the width of the image in pixels
BitDepth	Integer indicating the number of bits per pixel

Field	Description
ColorType	Type of image: `'truecolor'` for a true-color (RGB) image, `'grayscale'` for a grayscale image, or `'indexed'` for an indexed image.
ModelType	Type of coordinate system used to georeference the image: `'ModelTypeProjected'`, `'ModelTypeGeographic'`, `'ModelTypeGeocentric'`, or `''`.
PCS	Projected coordinate system
Projection	EPSG identifier for the underlying projection method
MapSys	Map system, if applicable: `'STATE_PLANE_27'`, `'STATE_PLANE_83'`, `'UTM_NORTH'`, `'UTM_SOUTH'`, or `''`
Zone	`double` indicating the UTM or State Plane Zone number, empty (`[]`) if not applicable or unknown
CTProjection	GeoTIFF identifier for the underlying projection method
ProjParm	N-by-1 `double` vector containing projection parameter values. The identity of each element is specified by the corresponding element of `ProjParmId`. Lengths are in meters, angles in decimal degrees.
ProjParmId	N-by-1 cell array listing the projection parameter identifier for each corresponding numerical element of `ProjParm`: • `'ProjNatOriginLatGeoKey'` • `'ProjNatOriginLongGeoKey'` • `'ProjFalseEastingGeoKey'` • `'ProjFalseNorthingGeoKey'` • `'ProjFalseOriginLatGeoKey'` • `'ProjFalseOriginLongGeoKey'` • `'ProjCenterLatGeoKey'` • `'ProjCenterLongGeoKey'` • `'ProjAzimuthAngleGeoKey'` • `'ProjRectifiedGridAngleGeoKey'` • `'ProjScaleAtNatOriginGeoKey'` • `'ProjStdParallel1GeoKey'` • `'ProjStdParallel2GeoKey'`
GCS	Geographic coordinate system
Datum	Projection datum type, such as `'North American Datum 1927'` or `'North American Datum 1983'`
Ellipsoid	Ellipsoid name as defined by the `ellipsoid.csv` EPSG file
SemiMajor	`double` indicating the length of the semimajor axis of the ellipsoid, in meters
SemiMinor	`double` indicating the length of the semiminor axis of the ellipsoid, in meters
PM	Prime meridian location, for example, `'Greenwich'` or `'Paris'`

Field	Description
PmLongToGreenwich	double indicating the decimal degrees of longitude between this prime meridian and Greenwich. Prime meridians to the west of Greenwich are negative.
UOMLength	Units of length used in the projected coordinate system
UOMLengthInMeters	double defining the UOMLength unit in meters.
UOMAngle	Angular units used for geographic coordinates
UOMAngleInDegrees	double defining the UOMAngle unit in degrees.
TiePoints	Structure containing the image tiepoints. The structure contains these fields:

Field Name	Description
ImagePoints	Structure containing row and column coordinates of each image tiepoint. The ImagePoints structure contains these fields: • Row — double array of size 1-by-N • Col — double array of size 1-by-N
WorldPoints	Structure containing the *x* and *y* world coordinates of the tiepoints. The WorldPoints structure contains these fields: • X — double array of size 1-by-N • Y — double array of size 1-by-N

Field	Description
PixelScale	3-by-1 double array that specifies the X, Y, Z pixel scale values.
SpatialRef	Value depends on the value of the ModelType field: • 'ModelTypeProjected' — SpatialRef is a map raster reference object. • 'ModelTypeGeographic' — SpatialRef is a geographic raster reference object, unless the geometric transformation is affine, in which case it is '' (empty). • 'ModelTypeGeocentric' — SpatialRef is '' (empty). • '' (empty) — geotiffinfo issues a warning and SpatialRef is a map raster reference object. If the spatial referencing is ambiguously defined by the GeoTIFF file, then SpatialRef is '' (empty).
RefMatrix	3-by-2 double referencing matrix that must be unambiguously defined by the GeoTIFF file. Otherwise it is empty ([]).
BoundingBox	2-by-2 double array that specifies the minimum (row 1) and maximum (row 2) values for each dimension of the image data in the GeoTIFF file.

Field	Description
CornerCoords	Structure with six fields that contains coordinates of the outer corners of the GeoTIFF image. Each field is a 1-by-4 `double` array, or empty (`[]`) if unknown. The arrays contain the coordinates of the outer corners of the corner pixels, starting from the `(1,1)` corner and proceeding clockwise:

Field	Description
X	Easting coordinates in the projected coordinate system. X equals Lon (below) if *ModelType* is `'ModelTypeGeographic'`
Y	Northing coordinates in the projected coordinate system. Y equals Lat (below) if *ModelType* is `'ModelTypeGeographic'`
Row	Row coordinates of the corner.
Col	Column coordinates of the corner.
Lat	Latitudes of the corner.
Lon	Longitudes of the corner.

Field	Description
GeoTIFFCodes	Structure containing raw numeric values for those GeoTIFF fields that are encoded numerically in the file. These raw values, converted to a character vectors elsewhere in the `info` structure, are provided here for reference. • Model • PCS • GCS • UOMLength • UOMAngle • Datum • PM • Ellipsoid • ProjCode • Projection • CTProjection • ProjParmId • MapSys Each is scalar, except for `ProjParmId`, which is a column vector.

Field	Description
GeoTIFFTags	Structure containing field names that match the GeoTIFF tags in the file. At least one GeoTIFF tag must be present in the file or an error is issued. The following fields may be included:

Field	Value
ModelPixelScaleTag	1-by-3 double
ModelTiepointTag	1-by-6 double
ModelTransformationTag	1-by-16 double
GeoKeyDirectoryTag	scalar structure
GeoAsciiParamsTag	character vector
GeoDoubleParamsTag	1-by-N double
RPCCoefficientTag	scalar RPCCoefficientTag

The GeoKeyDirectoryTag contains field names that match the names of the "GeoKeys". For more information about the "GeoKeys" refer to the GeoTIFF specification.

The RPCCoefficientTag contains properties with names corresponding to the tag elements listed in the RPCs in GeoTIFF technical note at: http://geotiff.maptools.org/rpc_prop.html

Field	Description
ImageDescription	Description of the image. If no description is included in the file, the field is omitted.

Tips

Get additional information about the coordinate reference system for a GeoTIFF file by using the georasterinfo function.

See Also

readgeoraster | geotiffwrite | RPCCoefficientTag | projfwd | projinv

Introduced before R2006a

geotiffread

(Not recommended) Read GeoTIFF file

Note `geotiffread` is not recommended, except when reading a GeoTIFF file from a URL or when reading multiple images from the same file. In other situations, use `readgeoraster` instead. For more information, see "Compatibility Considerations".

Syntax

```
[A,R] = geotiffread(filename)
[X,cmap,R] = geotiffread(filename)
[A,refmat,bbox] = geotiffread(filename)
[X,cmap,refmat,bbox] = geotiffread(filename)
[ ___ ] = geotiffread(url)
[ ___ ] = geotiffread( ___ ,idx)
```

Description

`[A,R] = geotiffread(filename)` reads a georeferenced grayscale, RGB, or multispectral image or data grid from the GeoTIFF file specified by `filename` into A and creates a spatial referencing object, R.

`[X,cmap,R] = geotiffread(filename)` reads an indexed image into X and the associated colormap into `cmap`, and creates a spatial referencing object, R.

`[A,refmat,bbox] = geotiffread(filename)` reads a georeferenced grayscale, RGB, or multispectral image or data grid into A, the corresponding referencing matrix into `refmat`, and the bounding box into `bbox`.

`[X,cmap,refmat,bbox] = geotiffread(filename)` reads an indexed image into X, the associated colormap into `cmap`, the referencing matrix into `refmat`, and the bounding box into `bbox`. The referencing matrix must be unambiguously defined by the GeoTIFF file, otherwise it and the bounding box are returned empty.

`[ ___ ] = geotiffread(url)` reads the GeoTIFF image from a URL.

`[ ___ ] = geotiffread( ___ ,idx)` reads one image from a multi-image GeoTIFF file or URL.

Examples

Read and Display the Boston GeoTIFF Image

```
[boston,R] = geotiffread('boston.tif');
figure
mapshow(boston,R);
axis image off
```
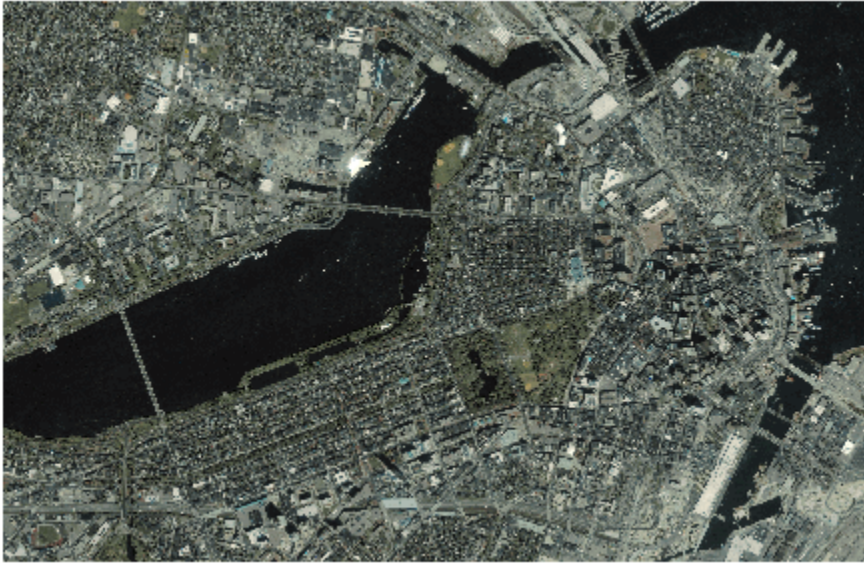
Input Arguments

`filename` — Name of GeoTIFF file
character vector | string scalar

Name of the GeoTIFF file, specified as a string scalar or character vector. Include the folder name in `filename` or place the file in the current folder or in a folder on the MATLAB path. If the named file includes the extension `.TIF` or `.TIFF` (either upper- or lowercase), you can omit the extension from `filename`.

`idx` — Index of image in GeoTIFF file
1 (default) | positive integer

Index of image in GeoTIFF file, specified as a positive integer. For example, if `idx` is 3, `geotiffread` reads the third image in the file. By default, `idx` indexes to the first image in the file.

`url` — Internet URL
character vector | scalar string

Internet URL, specified as a string scalar or character vector. The URL must include the protocol type (e.g., "https://").

Output Arguments

A — Georeferenced image or data grid
M-by-*N* numeric matrix | *M*-by-*N*-by-*P* numeric array

Georeferenced image or data grid, returned as one of the following:

- An *M*-by-*N* numeric matrix when the file contains a grayscale image or data grid

- An *M*-by-*N*-by-*P* numeric array when the file contains a color image, multispectral image, hyperspectral image, or data grid

The class of A depends on the storage class of the pixel data in the file, which is related to the `BitsPerSample` property as returned by the `imfinfo` function.

R — Spatial referencing object
geographic raster reference object | map raster reference object

Spatial referencing object, returned as one of the following.

- A geographic raster reference object of type `GeographicCellsReference` or `GeographicPostingsReference`. This object is returned when the image or data grid is referenced to a geographic coordinate system.

- A map raster reference object of type `MapCellsReference` or `MapPostingsReference`. This object is returned when the image or data grid is referenced to a projected coordinate system.

X — Indexed image
M-by-*N* numeric matrix

Indexed image, returned as an *M*-by-*N* numeric matrix.

cmap — Color map
c-by-3 numeric matrix

Color map associated with indexed image X, specified as an *c*-by-3 numeric matrix. There are *c* colors in the color map, each represented by a red, green, and blue pixel value. Colormap values are rescaled into the range [0,1].

refmat — Referencing matrix
3-by-2 numeric matrix | [ ]

Referencing matrix, returned as 3-by-2 numeric matrix that transforms raster row and column indices to or from geographic coordinates according to:

```
[lon lat] = [row col 1] * refmat
```

`refmat` defines a (non-rotational, non-skewed) relationship in which each column of the data grid falls along a meridian and each row falls along a parallel. `refmat` must be unambiguously defined by the GeoTIFF file, otherwise it and the bounding box, `bbox`, are returned empty.

Data Types: `double`

bbox — Bounding box
2-by-2 numeric matrix | [ ]

Bounding box, returned as a 2-by-2 numeric matrix that specifies the minimum (row 1) and maximum (row 2) values for each dimension of the image data in the GeoTIFF file. `bbox` is returned empty if `refmat` is ambiguously defined by the GeoTIFF file.

Data Types: `double`

Tips

- `geotiffread` imports pixel data using the TIFF-reading capabilities of the MATLAB function `imread` and likewise shares any limitations of `imread`. Consult the `imread` documentation for information on TIFF image support.

Compatibility Considerations

geotiffread is not recommended
Not recommended starting in R2020a

`geotiffread` is not recommended, except when reading a GeoTIFF file from a URL or when reading multiple images from the same file. In other situations, use `readgeoraster` instead. There are no plans to remove `geotiffread`.

Unlike `geotiffread`, which returns a referencing matrix in some cases, the `readgeoraster` function returns a raster reference object. Reference objects have several advantages over referencing matrices.

- Unlike referencing matrices, reference objects have properties that document the size of the associated raster, its limits, and the direction of its rows and columns. For examples of reference object properties, see `GeographicCellsReference` and `MapPostingsReference`.
- You can manipulate the limits of rasters associated with reference objects using the `geocrop` or `mapcrop` functions.
- You can manipulate the size and resolution of rasters associated with reference objects using the `georesize` or `mapresize` functions.

This table shows some typical usages of `geotiffread` and how to update your code to use `readgeoraster` instead. Unlike `geotiffread`, the `readgeoraster` function requires you to specify a file extension. For example, use `[Z,R] = readgeoraster('boston.tif')`.

Not Recommended	Recommended
`[A,R] = geotiffread(filename);`	`[A,R] = readgeoraster(filename);`
`[X,cmap,R] = geotiffread(filename);`	`[X,R,cmap] = readgeoraster(filename);`

`[A,refmat,bbox] = geotiffread(filename);`	`[A,R] = readgeoraster(filename);`
	`xlimits = R.XWorldLimits;`
	`ylimits = R.YWorldLimits;`
	`bbox = [xlimits' ylimits'];`

See Also
geoshow | geotiffinfo | geotiffwrite | imread | mapshow | readgeoraster

Introduced before R2006a

geotiffwrite

Write GeoTIFF file

Syntax

```
geotiffwrite(filename,A,R)
geotiffwrite(filename,X,cmap,R)
geotiffwrite( ___ ,Name,Value)
```

Description

`geotiffwrite(filename,A,R)` writes a georeferenced image or data grid, A, spatially referenced by R, into an output file, `filename`.

`geotiffwrite(filename,X,cmap,R)` writes the indexed image in X and its associated colormap, `cmap`, to `filename`. X is spatially referenced by R.

`geotiffwrite( ___ ,Name,Value)` writes an image or data grid with one or more `Name,Value` pair arguments that control various characteristics of the output file.

Examples

Write Image from JPEG File to GeoTIFF File

Read JPEG image from file.

```
basename = 'boston_ovr';
imagefile = [basename '.jpg'];
RGB = imread(imagefile);
```

Derive world file name from image file name, read the world file, and construct a spatial referencing object.

```
worldfile = getworldfilename(imagefile);
R = worldfileread(worldfile, 'geographic', size(RGB));
```

Write image data and referencing data to GeoTIFF file.

```
filename = [basename '.tif'];
geotiffwrite(filename, RGB, R)
```

Construct an empty map axes and display the map.

```
figure
usamap(RGB, R)
geoshow(filename)
```
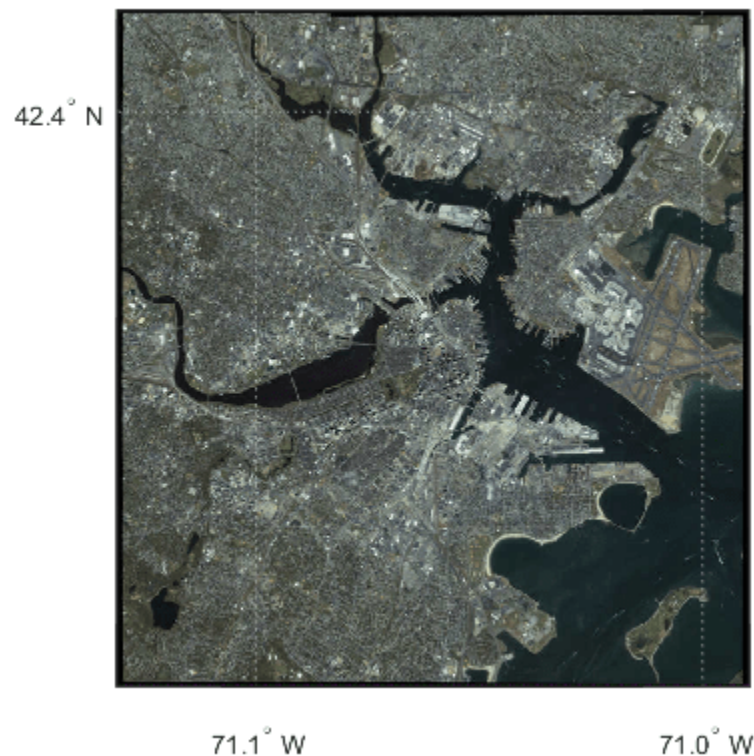
Convert Classic TIFF to Tiled BigTIFF

Convert a georeferenced classic TIFF file to a tiled BigTIFF file by extracting information from the classic TIFF file. First, import a classic TIFF image of Boston and a map cells reference object. Get metadata from the file using `geotiffinfo`.

```
infilename = 'boston.tif';
[A,R] = readgeoraster(infilename);
info = geotiffinfo(infilename);
```

Specify tags to include in the tiled BigTIFF file. To do this, extract the GeoKey directory tag from the metadata. Then, create tags specifying the length and width of the tiles.

```
geoTags = info.GeoTIFFTags.GeoKeyDirectoryTag;
tiffTags = struct('TileLength',1024,'TileWidth',1024);
```

Write the data to a new GeoTIFF file. Specify the file format as BigTIFF using the `'TiffType'` name-value pair. Include tags by specifying the `'GeoKeyDirectoryTag'` and `'TiffTags'` name-value pairs.

```
outfilename = 'boston_bigtiff.tif';
geotiffwrite(outfilename,A,R,'TiffType','bigtiff', ...
                            'GeoKeyDirectoryTag',geoTags, ...
                            'TiffTags',tiffTags)
```

Verify you have written the BigTIFF file by reading the file and querying the tags.

```
biginfo = geotiffinfo(outfilename);
biginfo.GeoTIFFTags.GeoKeyDirectoryTag
```

```
ans = struct with fields:
          GTModelTypeGeoKey: 1
         GTRasterTypeGeoKey: 1
      ProjectedCSTypeGeoKey: 26986
          PCSCitationGeoKey: 'State Plane Zone 2001 NAD = 83'
      ProjLinearUnitsGeoKey: 9003
```

```
t = Tiff(outfilename);
getTag(t,'TileLength')
```

```
ans = 1024
```

```
getTag(t,'TileWidth')
```

```
ans = 1024
```

```
close(t)
```

Write WMS Image to GeoTIFF File

Read data from WMS server.

```
nasaLayers = wmsfind('nasa', 'SearchField', 'serverurl');
layerName = 'bluemarbleng';
layer = refine(nasaLayers, layerName,  'SearchField', 'layername', ...
  'MatchType', 'exact');
[A, R] = wmsread(layer(1));
```

Write data to GeoTIFF file.

```
filename = [layerName '.tif'];
geotiffwrite(filename, A, R)
```

View data in file.

```
figure
worldmap world
geoshow(filename)
```
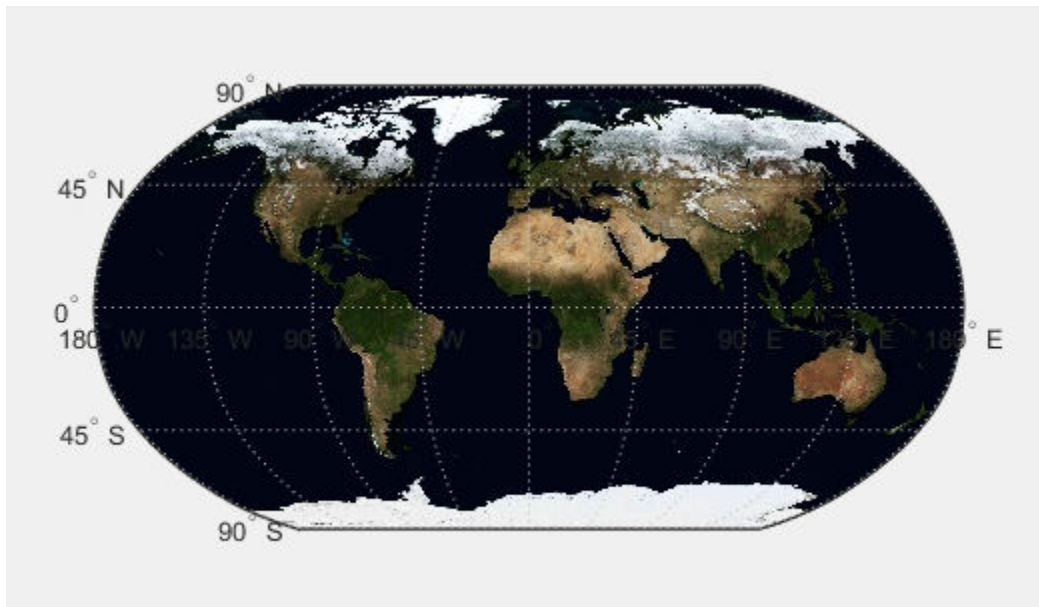
Write Concord Orthophotos to Single GeoTIFF File

Read the two adjacent orthophotos and combine them.

```
X_west = imread('concord_ortho_w.tif');
X_east = imread('concord_ortho_e.tif');
X = [X_west X_east];
```

Construct referencing objects for the orthophotos and for their combination.

```
R_west = worldfileread('concord_ortho_w.tfw', 'planar', size(X_west));
R_east = worldfileread('concord_ortho_e.tfw', 'planar', size(X_east));
R = R_west;
R.XLimWorld = [R_west.XLimWorld(1) R_east.XLimWorld(2)];
R.RasterSize = size(X);
```

Write the combined image to a GeoTIFF file. Use the code number, 26986, indicating the PCS_NAD83_Massachusetts Projected Coordinate System.

```
coordRefSysCode = 26986;
filename = 'concord_ortho.tif';
geotiffwrite(filename, X, R, 'CoordRefSysCode', coordRefSysCode);
```

Display the map.

```
figure
mapshow(filename)
```
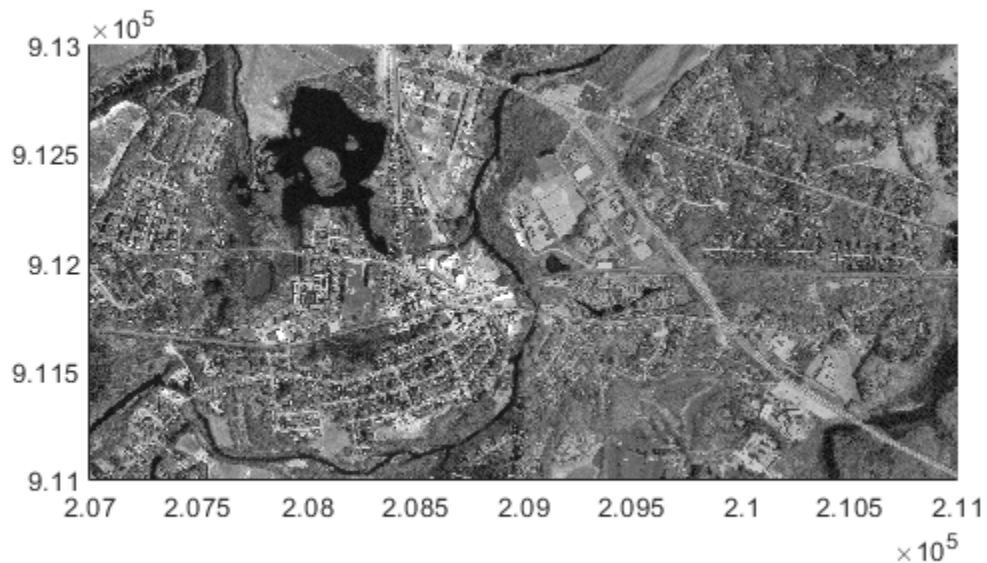
Write Subset of GeoTIFF File to New GeoTIFF File

Import a GeoTIFF image and map cells reference object for an area around Boston using `readgeoraster`.

```
[A,RA] = readgeoraster('boston.tif');
```

Crop the data to the limits specified by `xlimits` and `ylimits` using `mapcrop`.

```
xlimits = [764318 767678];
ylimits = [2951122 2954482];
[B,RB] = mapcrop(A,RA,xlimits,ylimits);
```

Get information about the GeoTIFF image using `geotiffinfo`. Extract the GeoKey directory tag from the information.

```
info = geotiffinfo('boston.tif');
key = info.GeoTIFFTags.GeoKeyDirectoryTag;
```

Write the cropped data and GeoKey directory tag to a file. Verify the cropped data has been written to a file by displaying it.

```
filename = 'boston_subimage.tif';
geotiffwrite(filename,B,RB,'GeoKeyDirectoryTag',key)
```

```
figure
mapshow(filename)
```
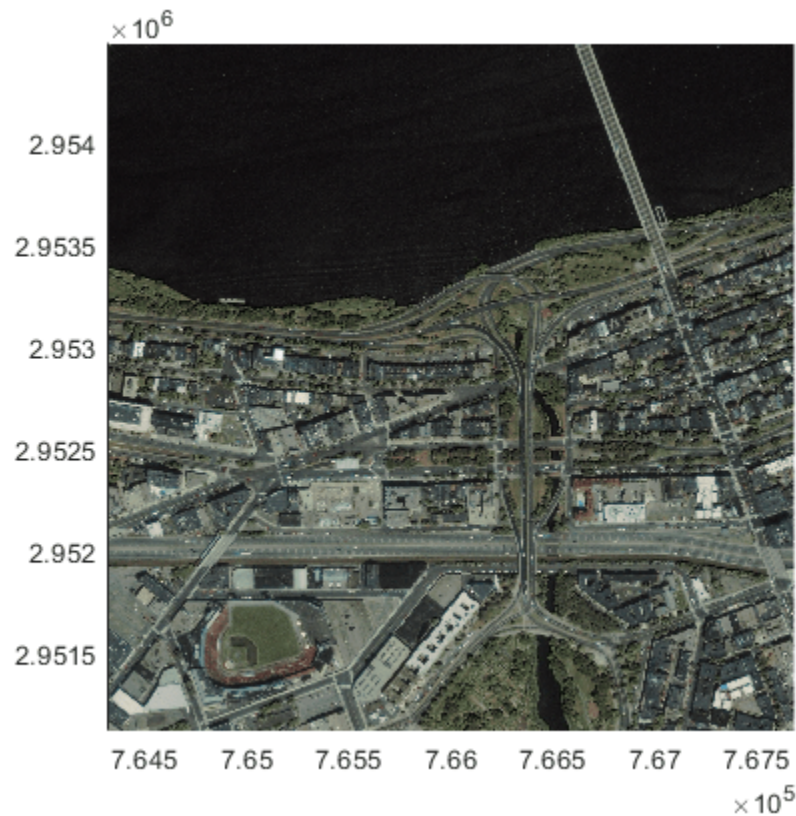

Write Elevation Data to GeoTIFF File

Write elevation data for an area around South Boulder Peak in Colorado to a GeoTIFF file. First, import the elevation data and a geographic postings reference object.

```
[Z,R] = readgeoraster('n39_w106_3arc_v2.dt1','OutputType','double');
```

Specify GeoKey directory tag information for the GeoTIFF file as a structure. Indicate the data is in a geographic coordinate system by specifying the `GTModelTypeGeoKey` field as 2. Indicate that the reference object uses postings (rather than cells) by specifying the `GTRasterTypeGeoKey` field as 2. Indicate the data is referenced to a geographic coordinate reference system by specifying the `GeographicTypeGeoKey` field as 4326.

```
key.GTModelTypeGeoKey  = 2;
key.GTRasterTypeGeoKey = 2;
key.GeographicTypeGeoKey = 4326;
```

Write the data and GeoKey directory tag to a file.

```
filename = 'southboulder.tif';
geotiffwrite(filename,Z,R,'GeoKeyDirectoryTag',key)
```

Verify the data has been written to a file by displaying it on a map.

```
usamap([39 40],[-106 -105])
g = geoshow(filename,'DisplayType','mesh');
demcmap(g.CData)
```
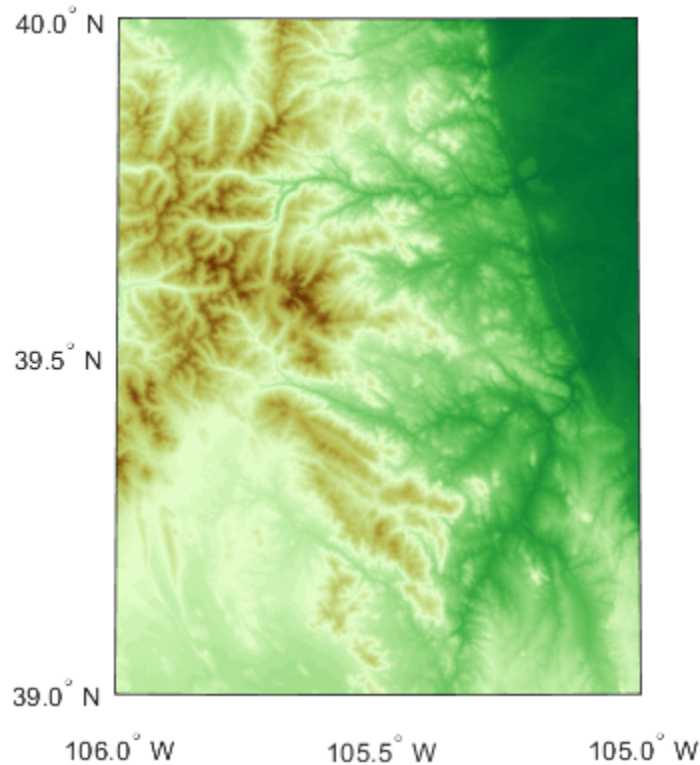

The elevation data used in this example is courtesy of the US Geological Survey.

Write TIFF File Containing RPC Metadata

Create a sample TIFF file with RPC metadata. To do this, create an array of zeros and an associated reference object.

```
A = zeros(180,360);
latlim = [-90 90];
lonlim = [-180 180];
RA = georefcells(latlim,lonlim,size(A));
```

Then, create an `RPCCoefficientTag` metadata object and set some fields with typical values. The `RPCCoefficientTag` object represents RPC metadata in a readable form.

```
rpctag = map.geotiff.RPCCoefficientTag;
rpctag.LineOffset = 1;
rpctag.SampleOffset = 1;
rpctag.LineScale = 2;
```

```
rpctag.SampleScale = 2;
rpctag.GeodeticHeightScale = 500;
```

Write the image, the associated referencing object, and the `RPCCoefficientTag` object to a file.

```
geotiffwrite('myfile',A,RA,'RPCCoefficientTag',rpctag)
```

Write Raw RPC Coefficient Metadata to GeoTIFF File

This example shows how to write RPC coefficient metadata to a TIFF file. In a real workflow, you would create the RPC coefficient metadata according to the TIFF extension specification. This example does not show the specifics of how to create valid RPC metadata. To simulate raw RPC metadata, the example creates a sample TIFF file with RPC metadata and then uses `imfinfo` to read this RPC metadata in raw, unprocessed form from the file. The example then writes this raw RPC metadata to a file using the `geotiffwrite` function.

Create Raw RPC Coefficient Metadata

To simulate raw RPC metadata, create a simple test file and write some RPC metadata to the file. For this test file, create a toy image and a referencing object associated with the image.

```
myimage = zeros(180,360);
latlim = [-90 90];
lonlim = [-180 180];
R = georefcells(latlim,lonlim,size(myimage));
```

Create an `RPCCoefficientTag` metadata object and set some of the fields. The toolbox uses the `RPCCoefficientTag` object to represent RPC metadata in human readable form.

```
rpctag = map.geotiff.RPCCoefficientTag;
rpctag.LineOffset = 1;
rpctag.SampleOffset = 1;
rpctag.LineScale = 2;
rpctag.SampleScale = 2;
rpctag.GeodeticHeightScale = 500;
```

Write the image, the associated referencing object, and the `RPCCoefficientTag` object to a file.

```
geotiffwrite('myfile',myimage,R,'RPCCoefficientTag',rpctag)
```

Read Raw RPC Coefficient Metadata

Read the RPC coefficient metadata from the test file using the `imfinfo` function. When it encounters unfamiliar metadata, `imfinfo` returns the data, unprocessed, in the `UnknownTags` field. Note that the `UnknownTags` field contains an array of 92 doubles. This is the raw RPC coefficient metadata, read from the file in unprocessed form.

```
info = imfinfo('myfile.tif');
info.UnknownTags

ans = struct with fields:
        ID: 50844
    Offset: 10676
     Value: [-1 -1 1 1 0 0 0 2 2 1 1 500 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ... ]
```

Write Raw RPC Metadata to a File

Write the raw RPC metadata to a file. First, extract the RPC coefficient metadata from the info structure.

```
value = info.UnknownTags.Value;
```

Then, construct an `RPCCoefficientTag` object, passing the raw RPC metadata (array of 92 doubles) as an argument.

```
rpcdata = map.geotiff.RPCCoefficientTag(value)

rpcdata =
  RPCCoefficientTag with properties:

              BiasErrorInMeters: -1
            RandomErrorInMeters: -1
                     LineOffset: 1
                   SampleOffset: 1
          GeodeticLatitudeOffset: 0
         GeodeticLongitudeOffset: 0
            GeodeticHeightOffset: 0
                      LineScale: 2
                    SampleScale: 2
           GeodeticLatitudeScale: 1
          GeodeticLongitudeScale: 1
             GeodeticHeightScale: 500
        LineNumeratorCoefficients: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
      LineDenominatorCoefficients: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
      SampleNumeratorCoefficients: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
    SampleDenominatorCoefficients: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

Pass the `RPCCoefficientTag` object to the `geotiffwrite` function and write the RPC metadata to a file.

```
geotiffwrite('myfile2',myimage,R,'RPCCoefficientTag',rpcdata)
```

To verify that the data was written to the file, read the RPC metadata from the TIFF file using `geotiffinfo`. Compare the returned RPC metadata with the metadata written to the test file.

```
ginfo = geotiffinfo('myfile2');
ginfo.GeoTIFFTags.RPCCoefficientTag

ans =
  RPCCoefficientTag with properties:

              BiasErrorInMeters: -1
            RandomErrorInMeters: -1
                     LineOffset: 1
                   SampleOffset: 1
          GeodeticLatitudeOffset: 0
         GeodeticLongitudeOffset: 0
            GeodeticHeightOffset: 0
                      LineScale: 2
                    SampleScale: 2
           GeodeticLatitudeScale: 1
          GeodeticLongitudeScale: 1
```

```
         GeodeticHeightScale: 500
     LineNumeratorCoefficients: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
   LineDenominatorCoefficients: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
   SampleNumeratorCoefficients: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 SampleDenominatorCoefficients: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

Input Arguments

`filename` — Name and location of output file
character vector | string scalar

Name and location of output file, specified as a string scalar or character vector. If your `filename` includes an extension, it must be `'.tif'` or `'.TIF'`. If the input, `A`, is at least 160-by-160 in size, the output file is a tiled GeoTIFF file. Otherwise, `geotiffwrite` organizes the output file as rows-per-strip.

Data Types: `char` | `string`

A — Georeferenced image or data grid
M-by-*N* numeric matrix | *M*-by-*N*-by-*P* numeric array

Georeferenced image or data grid, specified as one of the following:

- An *M*-by-*N* numeric matrix representing a grayscale image or data grid
- An *M*-by-*N*-by-*P* numeric array representing a color image, multispectral image, hyperspectral image, or data grid

The coordinates of `A` are geographic and in the `'WGS 84'` coordinate system, unless you specify `'GeoKeyDirectoryTag'` or `'CoordRefSysCode'` and indicate a different coordinate system.

Data Types: `double` | `single` | `uint8` | `uint16` | `uint32` | `uint64` | `int8` | `int16` | `int32` | `int64` | `logical`

R — Spatial referencing information
geographic raster reference object | map raster reference object | referencing matrix | referencing vector

Spatial referencing information, specified as a geographic raster reference object of type `GeographicCellsReference` or `GeographicPostingsReference`, a map raster reference object of type `MapCellsReference` or `MapPostingsReference`, a referencing matrix, or a referencing vector.

If you are working with image coordinates in a projected coordinate system and `R` is a map raster reference object or a referencing matrix, specify `'GeoKeyDirectoryTag'` or `'CoordRefSysCode'` accordingly.

The `geotiffwrite` function does not use information contained in the `GeographicCRS` property of geographic raster reference objects or the `ProjectedCRS` property of map raster reference objects.

X — Indexed image
M-by-*N* numeric matrix

Indexed image data, specified as an *M*-by-*N* numeric matrix.

Data Types: `uint8` | `uint16`

cmap — Color map
c-by-3 numeric matrix

Color map associated with indexed image X, specified as an *c*-by-3 numeric matrix. There are *c* colors in the color map, each represented by a red, green, and blue pixel value.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'CoordRefSysCode',26986`

CoordRefSysCode — Coordinate reference system code
4326 (default) | positive integer | string scalar | character vector

Coordinate reference system code for the coordinates of the data, specified as the comma-separated pair consisting of `'CoordRefSysCode'` and a positive integer, string scalar, or character vector. You can specify coordinates in either a geographic or a projected coordinate system. If you specify the coordinate system with a string scalar or character vector, include the `'EPSG:'` prefix. To find code numbers, see the EPSG registry or the GeoTIFF specification in the "Tips" on page 1-648 section.

If you specify both the `GeoKeyDirectoryTag` and the `CoordRefSysCode`, the coordinate system code in `CoordRefSysCode` takes precedence over the coordinate system key found in the `GeoKeyDirectoryTag`. If one value specifies a geographic coordinate system and the other value specifies a projected coordinate system, you receive an error.

If you do not specify a value for this argument, the default value is 4326, indicating that the coordinates are geographic and in the `'WGS 84'` geographic coordinate system.

Example: 26986

Example: `'EPSG:26986'`

GeoKeyDirectoryTag — GeoKey directory tag
structure

GeoKey directory tag, specified as the comma-separated pair consisting of `'GeoKeyDirectoryTag'` and a structure that specifies the GeoTIFF coordinate reference system and meta-information. The structure contains field names that match the GeoKey names in the GeoTIFF specification. The field names are case insensitive. The structure can be obtained from the GeoTIFF information structure, returned by `geotiffinfo`, in the field, `GeoTIFFTags.GeoKeyDirectoryTag`.

if you specify the `GTRasterTypeGeoKey` field, `geotiffwrite` ignores it. The value for this GeoKey is derived from R. If you set certain fields of the `GeoKeyDirectoryTag` to inconsistent settings, you receive an error message. For instance, if R is a geographic raster reference object or a refvec, and you specify a `ProjectedCSTypeGeoKey` field or you set the `GTModelTypeGeoKey` field to 1 (projected coordinate system), you receive an error. Likewise, if R is a map raster reference object and you do not specify a `ProjectedCSTypeGeoKey` field or a `CoordRefSysCode`, or the `GTModelTypeGeoKey` field is set to 2 (geographic coordinate system), you receive an error message.

RPCCoefficientTag — Rational Polynomial Coefficients (RPC) tag
`RPCCoefficientTag` object

Values for the optional RPC TIFF tag, specified as the comma-separated pair consisting of `'RPCCoefficientTag'` and an `RPCCoefficientTag` object.

TiffTags — TIFF tags
structure

Values for the TIFF tags in the output file, specified as the comma-separated pair consisting of `'TiffTags'` and a structure. The field names of the structure match the TIFF tag names supported by the Tiff class. The field names are case insensitive.

You cannot set most TIFF tags using the structure input.

TiffTags Exceptions

S G
a ä
m b
p K
e y
K s
e D
y i
s r
D e
i c
r t
e o
c r
t y
X T
b a
e g
r
l S M
y o
T d
a e
g l
P
S i
o x
M e
d l
e S
l c
O a
P l
y e
f T
s a
e g
t
C
o s
u c
n a
t l
s e
e
T
a
g

ModelTiepointTag

ModelTransformationTag

The function sets several TIFF tags. The field names corresponding to the TIFF tag, their corresponding field values set by the function, their permissible values (if different from the Tiff class), and their data type are noted in the following table.

Automatic TIFF Tags

Field Name	Description
Compression	Type of image compression. The default is `'PackBits'`. Other permissible values are `'LZW'`, `'Deflate'`, and `'none'`. Numeric values, `Tiff.Compression.LZW`, `Tiff.Compression.PackBits`, `Tiff.Compression.Deflate`, or `Tiff.Compression.None` can also be used.
PhotometricInterpretation	Type of photometric interpretation. The field name can be shortened to `Photometric`. The value is set based on the input image characteristic, using the following algorithm: If A is [M-by-N-by-3] and is class type `uint8` or `uint16`, then the value is `'RGB'`. For all other sizes and data types, the value is `'MinIsBlack'`. If the X, CMAP syntax is supplied, the value is `'Palette'`. If the value is set to `'RGB'` and A is not [M-by-N-by-3], an error is issued. Permissible values are `'MinIsBlack'`, `'RGB'`, `'Palette'`, `'Separated'`. The numeric values, `Tiff.Photometric.MinIsBlack`, `Tiff.Photometric.RGB`, `Tiff.Photometric.Palette`, `Tiff.Photometric.Separated` can also be used.
Software	Software maker of the file. The value is set to the value `'MATLAB, Mapping Toolbox, The MathWorks, Inc.'`. To remove the value, set the tag to the empty string or character vector (`''`).
RowsPerStrip	A scalar positive integer-valued number specifying the desired rows per strip in the output file. If the size of A is less than [160-by-160], `geotiffwrite` sets `RowsPerStrip` to 1. If you specify `RowsPerStrip` and `TileWidth`, with or without `TileLength`, `geotiffwrite` issues an error.
TileWidth	A scalar positive integer-valued number and a multiple of 16 specifying the width of the tiles. `TileWidth` is set if the size of A is greater than [160-by-160]. If so, the value is such that a maximum of [10-by-10] tiles are created. If you specify both `RowsPerStrip` and `TileWidth`, `geotiffwrite` issues an error.
TileLength	A scalar positive integer-valued number and a multiple of 16 specifying the length of the tiles. `TileLength` is set if the size of A is greater than [160-by-160]. If so, the value is such that a maximum of [10-by-10] tiles are created. If you specify both `RowsPerStrip` and `TileLength`, `geotiffwrite` issues an error.

TiffType — Type of TIFF file
`'classictiff'` (default) | `'bigtiff'`

Type of TIFF file, specified as the comma-separated pair consisting of `'TiffType'` and either `'classictiff'` or `'bigtiff'`. The `'classictiff'` value creates a Classic TIFF file. The `'bigtiff'` value creates a BigTIFF file. In BigTIFF format, files can be larger than 4 GB.

While using the `'bigtiff'` format enables you to create files larger than 4 GB, the data you want to write must fit in memory.

Tips

- If you are working with image coordinates in a projected coordinate system and R is a map raster reference object or a referencing matrix, set the GeoKeyDirectoryTag or CoordRefSysCode argument, accordingly.

- Check the GeoTIFF specification for values of the following parameters:

  - `'CoordRefSysCode'` value for geographic coordinate systems
  - `'CoordRefSysCode'` value for projected coordinate systems
  - GeoKey field names for the `'GeoKeyDirectoryTag'`

See Also
geotiffinfo | readgeoraster | imread | imwrite | RPCCoefficientTag | Tiff

Introduced before R2006a

getCapabilities

Get capabilities document from server

Syntax

```
capabilities = getCapabilities(server)
```

Description

capabilities = getCapabilities(server) retrieves the capabilities document from the Web map service server, server, and updates the RequestURL property of the server.

Examples

Get Capabilities Document From Server

Search the WMS Database for layers from the NASA SVS Image Server. Create a WebMapServer object from one of the layers. Get the capabilities document from the server.

```
nasa = wmsfind('NASA SVS Image','SearchField','servertitle');
serverURL = nasa(1).ServerURL;
server = WebMapServer(serverURL);
capabilities = getCapabilities(server);
```

Input Arguments

server — Web map server
WebMapServer object

Web map server, specified as a WebMapServer object.

Output Arguments

capabilities — Capabilities document
WMSCapabilities object

Capabilities document, returned as a WMSCapabilities object.

Tips

The getCapabilities method accesses the Internet to retrieve the document. Periodically, the WMS server is unavailable. Retrieving the document can take several minutes.

See Also

disp | WebMapServer | WMSCapabilities

Introduced before R2006a

getm

Map object properties

Syntax

```
mat = getm(h)
mat = getm(h,MapPropertyName)
getm('MapProjection')
getm('axes')
getm('units')
```

Description

`mat = getm(h)` returns the map structure of the map axes specified by its handle. If the handle of a child of the map axes is specified, only its properties are returned.

`mat = getm(h,MapPropertyName)` returns the specified property value.

`getm('MapProjection')` lists all available projections.

`getm('axes')` lists the map axes properties by property name.

`getm('units')` lists the available units.

Examples

Create a default map axes and query a property value:

```
axesm('mercator','AngleUnits','degrees')
getm(gca,'MapParallels')

ans =
     0
```

See Also

axesm | setm | Map Axes Properties

Introduced before R2006a

getMap

Get raster map from server

Syntax

```
A = getMap(server,mapRequestURL)
```

Description

`A = getMap(server,mapRequestURL)` dynamically renders and retrieves a color or grayscale, geographically referenced, raster map from the Web map services server, `server`, and stores it in A. Parameters in the URL, `mapRequestURL`, define the map. The `getMap` function also updates the `RequestURL` property of the server with `mapRequestURL`.

Most of the time, you can read a WMS map using `wmsread` function instead of the `getMap` function. Use the `getMap` function when:

- Working with non-EPSG:4326 reference systems
- Creating an animation of a specific geographic area over time
- Retrieving multiple layers from a WMS server

Examples

Read Map From WMS Server

Read a map from the NASA Earth Observations WMS server.

Search the WMS Database for the Blue Marble global mosaic layer from the NASA Earth Observations WMS server.

```
neowms = wmsfind('neowms','SearchField','serverurl');
layer = refine(neowms,'bluemarbleng','MatchType','exact');
```

Read the map from the server.

```
server = WebMapServer(layer.ServerURL);
mapRequest = WMSMapRequest(layer,server);
A = getMap(server,mapRequest.RequestURL);
R = mapRequest.RasterReference;
```

Display the map.

```
figure
worldmap world
geoshow(A,R)
setm(gca,'MLabelParallel',-90,'MLabelLocation',90)
title(layer.LayerTitle)
```
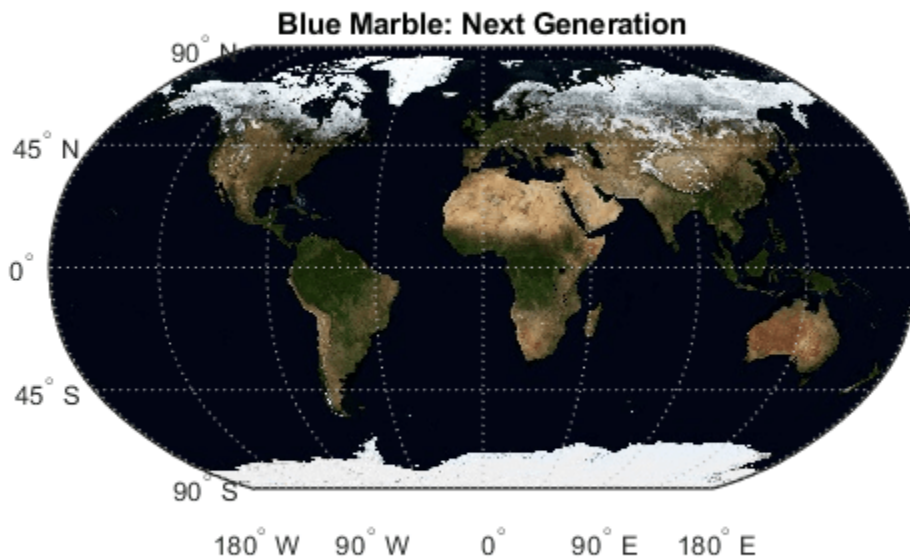
Blue Marble: Next Generation

Input Arguments

server — Web map server
WebMapServer object

Web map server, specified as a WebMapServer object.

mapRequestURL — URL
character vector

URL, specified as a character vector. Parameters in the URL define the map.

You can get a URL from:

- The output of wmsread
- The RequestURL property of a WMSMapRequest object
- An Internet search

Output Arguments

A — Rendered map
color or grayscale image

Rendered map, returned as a color or grayscale image.

Tips

`getMap` accesses the Internet to retrieve the map. Periodically, the WMS server is unavailable. Retrieving the map can take several minutes.

See Also

WMSMapRequest | WebMapServer

Introduced before R2006a

getseeds

Interactively assign seeds for data grid encoding

Note `getseeds` will be removed in a future release.

Syntax

```
[row,col,val] = getseeds(map,R,nseeds)
[row,col,val] = getseeds(map,R,nseeds,seedval)
mat = getseeds(...)
```

Description

`[row,col,val] = getseeds(map,R,nseeds)` allows user to identify geographical objects while customizing a raster map. It prompts the user for mouse click positions of objects and assigns them a code value. The user is prompted for the value to seed at each location. The outputs are the row and column of the seed location and the value assigned at that location. `R` is either a 1-by-3 vector containing elements:

```
[cells/degree northern_latitude_limit western_longitude_limit]
```

or a 3-by-2 referencing matrix that transforms raster row and column indices to/from geographic coordinates according to:

```
[lon lat] = [row col 1] * R
```

If `R` is a referencing matrix, it must define a (non-rotational, non-skewed) relationship in which each column of the data grid falls along a meridian and each row falls along a parallel.

`[row,col,val] = getseeds(map,R,nseeds,seedval)` assigns the value `seedval` to each location supplied. If `seedval` is a scalar then the same value is assigned at each location. Otherwise, if `seedval` is a vector it must be `length(nseeds)` and each entry is assigned to the corresponding location. `getseeds` operates on the current axes (`gca`).

`mat = getseeds(...)` returns a single output matrix where `mat = [row col val]`.

Examples

Load elevation raster data and a geographic cells reference object. Create a map axes object. Then, call `getseeds` and interactively select three points.

```
load topo60c
axesm('gortho','grid','on')
seedmat = getseeds(topo60c,topo60cR,3);
```

When you have selected three points, you are prompted for their values. The regular data grid need not be displayed to execute `getseeds` on it.

See Also

encodem

Introduced before R2006a

getworldfilename

Derive world file name from image file name

Syntax

```
worldfilename = getworldfilename(imagefilename)
```

Description

`worldfilename = getworldfilename(imagefilename)` returns the name of the corresponding world file derived from the name of an image file.

The world file and the image file have the same base name. If `imagefilename` follows the ".3" convention, then you create the world file extension by removing the middle letter and appending the letter `'w'`.

If `imagefilename` has an extension that does not follow the ".3" convention, then a `'w'` is appended to the full image name to construct the world file name.

If `imagefilename` has no extension, then `'.wld'` is appended to construct a world file name.

Examples

Given the following image file names, `worldfilename` would return these world file names:

Image File Name	World File Name
myimage.tif	myimage.tfw
myimage.jpeg	myimage.jpegw
myimage	myimage.wld

See Also
mapshow | mapview | worldfileread | worldfilewrite

Introduced before R2006a

globedem

(To be removed) Read Global Land One-km Base Elevation (GLOBE) data

---

Note `globedem` will be removed in a future release. Use `readgeoraster` instead. For more information, see "Compatibility Considerations".

Syntax

```
[Z,refvec] = globedem(filename,scalefactor)
[Z,refvec] = globedem(filename,scalefactor,latlim,lonlim)
[Z,refvec] = globedem(foldername,scalefactor,latlim,lonlim)
```

Description

`[Z,refvec] = globedem(filename,scalefactor)` reads the GLOBE DEM files and returns the result as a regular data grid. The `filename` is given as a string scalar or character vector that does not include an extension. `globedem` first reads the Esri header file found in the subfolder `'/esri/hdr/'` and then the binary data file name. If the files are not found on the MATLAB path, they can be selected interactively. `scalefactor` is an integer that when equal to 1 gives the data at its full resolution. When `scalefactor` is an integer n larger than 1, every nth point is returned. The map data is returned as an array of elevations and associated three-element referencing vector. Elevations are given in meters above mean sea level, using WGS 84 as a horizontal datum.

`[Z,refvec] = globedem(filename,scalefactor,latlim,lonlim)` allows a subset of the map data to be read. The limits of the desired data are specified as vectors of latitude and longitude in degrees. The elements of `latlim` and `lonlim` must be in ascending order.

`[Z,refvec] = globedem(foldername,scalefactor,latlim,lonlim)` reads and concatenates data from multiple files within a GLOBE folder tree. The `foldername` input is a string scalar or character vector with the name of the folder that contains both the uncompressed data files and the Esri header files.

Background

GLOBE, the Global Land One-km Base Elevation data, was compiled by the National Geophysical Data Center from more than 10 different sources of gridded elevation data. GLOBE can be considered a higher resolution successor to TerrainBase. The data set consists of 16 tiles, each covering 50 by 90 degrees. Tiles require as much as 60 MB of storage. Uncompressed tiles take between 100 and 130 MB.

Examples

Determine the file that contains the area around Cape Cod. (This example assumes you have already downloaded some GLOBE data tiles.)

```
latlim = [41 42.5]; lonlim = [-73 -69.9];
globedems(latlim,lonlim)
```

```
ans =
    'f10g'
```

Extract every 20th point from the tile covering the northeastern United States and eastern Canada. If you specify an empty file name (''), globedem presents a file browser that you use to first select the header file and then select the data file interactively.

```
[Z,refvec] = globedem('',20);
size(Z)

ans =
   300    540
```

Extract a subset of the data for Massachusetts at the full resolution.

```
latlim = [41 42.5]; lonlim = [-73 -69.9];
[Z,refvec] = globedem('f10g',1,latlim,lonlim);
size(Z)

ans =
   181 373
```

Replace the NaNs in the ocean with -1 to color them blue.

```
Z(isnan(Z)) = -1;
```

Extract some data for southern Louisiana in an area that straddles two tiles. Provide the name of the folder containing the data files, and let globedem determine which files are required, read from the files, and concatenate the data into a single regular data grid.

```
latlim =[28.61 31.31]; lonlim = [-91.24 -88.62];
globedems(latlim,lonlim)

ans =
    'e10g'
    'f10g'

[Z,refvec] =
globedem('d:\externalData\globe\elev',1,latlim,lonlim);
size(Z)

ans =
      325.00         315.00
```

Tips

The globedem function reads data from GLOBE Version 1.0. The data is for elevations only. Elevations are given in meters above mean sea level using WGS 84 as a horizontal datum. Areas with no data, such as the oceans, are coded with NaNs.

The data set and documentation are available over the Internet.

Note For details on locating map data for download over the Internet, see the following documentation at the MathWorks Web site: "Find Geospatial Data Online".

Compatibility Considerations

globedem will be removed
Warns starting in R2021b

Raster reading functions that return referencing vectors will be removed, including `globedem`. Instead, use `readgeoraster`, which returns a geographic raster reference object. Reference objects have several advantages over referencing vectors.

- Unlike referencing vectors, reference objects have properties that document the size of the associated raster, its geographic limits, and the direction of its rows and columns. For examples of reference object properties, see `GeographicCellsReference` and `GeographicPostingsReference`.
- You can manipulate the limits of rasters associated with reference objects using the `geocrop` function.
- You can manipulate the size and resolution of rasters associated with reference objects using the `georesize` function.
- Most functions that accept referencing vectors as input also accept reference objects.

This table shows some typical usages of `globedem` and how to update your code to use `readgeoraster` instead. Unlike `globedem`, which requires the `.hdr` header file to be in the `/esri/hdr/` subfolder, the `readgeoraster` function requires the data file and header file to be in the same folder. The `readgeoraster` function also requires you to specify a file extension.

Will Be Removed	Recommended
`[Z,refvec] = globedem(filename,samplefactor);`	`[Z,R] = readgeoraster(filename,'CoordinateSystemType',...` `[Z,R] = georesize(Z,R,1/samplefactor);`

`[Z,refvec] = globedem(filename,samplefactor,latlim,lonlim);`	`[Z,R] = readgeoraster(filename,'CoordinateSystemType',...`
	`[Z,R] = geocrop(Z,R,latlim,lonlim);`
	`[Z,R] = georesize(Z,R,1/scalefactor);`

The `readgeoraster` function returns data using the native data type embedded in the file. Return a different data type by specifying the `'OutputType'` name-value pair. For example, use `[Z,R] = readgeoraster(filename,'OutputType','double')`.

The `readgeoraster` function does not automatically replace missing data with `NaN` values. If your data set uses large negative numbers to indicate missing data, replace them with `NaN` values using the `standardizeMissing` function.

```
[Z,R] = readgeoraster('MtWashington-ft.grd');
info = georasterinfo('MtWashington-ft.grd');
m = info.MissingDataIndicator;
Z = standardizeMissing(Z,m);
```

References

See Web site for the National Oceanic and Atmospheric Administration, National Geophysical Data Center

See Also

demdataui | readgeoraster | georasterinfo

Introduced before R2006a

globedems

GLOBE data file names for latitude-longitude quadrangle

Syntax

```
tileNames = globedems(latlim,lonlim)
tileNames = globedems(lat,lon)
```

Description

`tileNames = globedems(latlim,lonlim)` returns a cell array of the tile names covering the geographic region for GLOBEDEM digital elevation maps. The region is specified by two-element vectors of latitude and longitude limits in units of degrees.

`tileNames = globedems(lat,lon)` returns a cell array of the tile names covering the geographic region for GLOBEDEM digital elevation maps. The region is specified by scalar latitude and longitude points, in units of degrees.

Background

GLOBE, the Global Land One-km Base Elevation data, was compiled by the National Geophysical Data Center from more than 10 different sources of gridded elevation data. The data set consists of 16 tiles, each covering 50 by 90 degrees. Determining which tiles are needed to cover a particular region generally requires consulting an index map. This function takes the place of such a reference by returning the file names for a given geographic region.

Examples

Which tiles are needed for southern Louisiana?

```
latlim =[28.61 31.31];
lonlim = [-91.24 -88.62];
globedems(latlim,lonlim)

ans =
    'e10g'
    'f10g'
```

Tips

The `globedems` function reads data from the format GLOBE Version 1.0.

See Also
`readgeoraster`

Introduced before R2006a

gpxread

Read GPX file

Syntax

```
P = gpxread(filename)
P = gpxread(URL)

S = gpxread( ___ ,'Index',V)

___ = gpxread( ___ ,Name,Value)
```

Description

`P = gpxread(filename)` reads point data from the GPS Exchange Format (GPX) file, `filename`, and returns an *n*-by-1 geopoint vector, P, where *n* is the number of waypoints, or points that define a route or track.

`gpxread` searches the file first for waypoints, then routes, and then tracks, and it returns the first type of data it finds. The Metadata field of P identifies the feature type (`'waypoint'`, `'track'`, or `'route'`) and any additional metadata associated with waypoint, route, or track. If the file contains multiple tracks or routes, P contains the points that define the first track or route in the file. If `gpxread` cannot find any features in the file, it returns an empty geopoint vector.

`P = gpxread(URL)` reads the GPX data from a URL. The URL must include the protocol type (for example, `http://`).

`S = gpxread( ___ ,'Index',V)` returns data from the GPX file in a geoshape vector, rather than a geopoint vector, only if the file contains track or route data and you specify the value of `'Index'` as a vector, V. Use this syntax when you want to work with the data as a line, rather than as a collection of points.

`___ = gpxread( ___ ,Name,Value)` reads data from a GPX file with additional options, specified by one or more `Name,Value` pair arguments, that control various characteristics of the import. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`''`) and is case insensitive. You can specify several name-value pair arguments in any order.

Examples

Read Waypoints and Display Them Over Image

Read and display waypoints from the `boston_placenames.gpx` file and overlay the points onto the `boston.tif` image.

First, import the waypoints and GeoTIFF file.

```
p = gpxread('boston_placenames.gpx');
[A,R] = readgeoraster('boston.tif');
```

Get the projection structure of the GeoTIFF image. Convert the length unit of the *X* and *Y* limits to meters for use with the projection structure.

```
proj = geotiffinfo('boston.tif');
mstruct = geotiff2mstruct(proj);
R.XWorldLimits = R.XWorldLimits * proj.UOMLengthInMeters;
R.YWorldLimits = R.YWorldLimits * proj.UOMLengthInMeters;
```
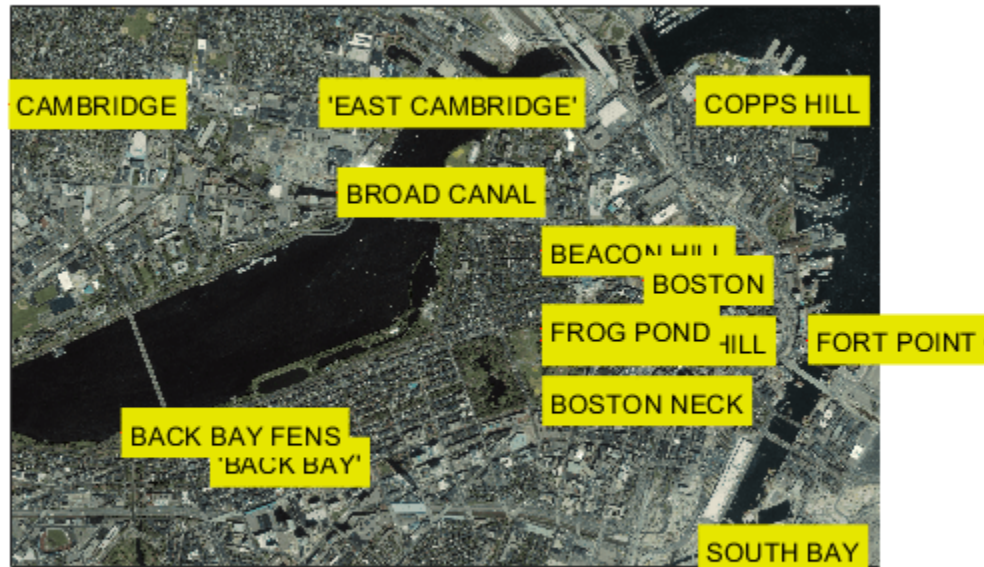
Display the map.

```
axesm(mstruct)
mapshow(A,R)
```


Display the names and positions of each point.

```
for k=1:length(p)
    textm(p(k).Latitude, p(k).Longitude, p(k).Name, ...
        'Color','k','BackgroundColor',[0.9 0.9 0],...
        'Interpreter','none');
end
geoshow(p)
xlim(R.XWorldLimits)
ylim(R.YWorldLimits)
```

Read and Display a Route

Read and display a route from Boston Logan International Airport to MathWorks in Natick, MA.

Read the route information from the GPX file.

```
route = gpxread('sample_route.gpx');
```

Compute `latlim` and `lonlim` with a 0.05 buffer.

```
[latlim, lonlim] = geoquadline(route.Latitude, route.Longitude);
[latlim, lonlim] = bufgeoquad(latlim, lonlim, .05, .05);
```

Display the route.

```
fig = figure;
pos = fig.Position;
fig.Position = [300 300 1.25*pos(3) 1.25*pos(4)];
ax = usamap(latlim, lonlim);
setm(ax, 'MLabelParallel', 43.5)
geoshow(route.Latitude, route.Longitude)
```
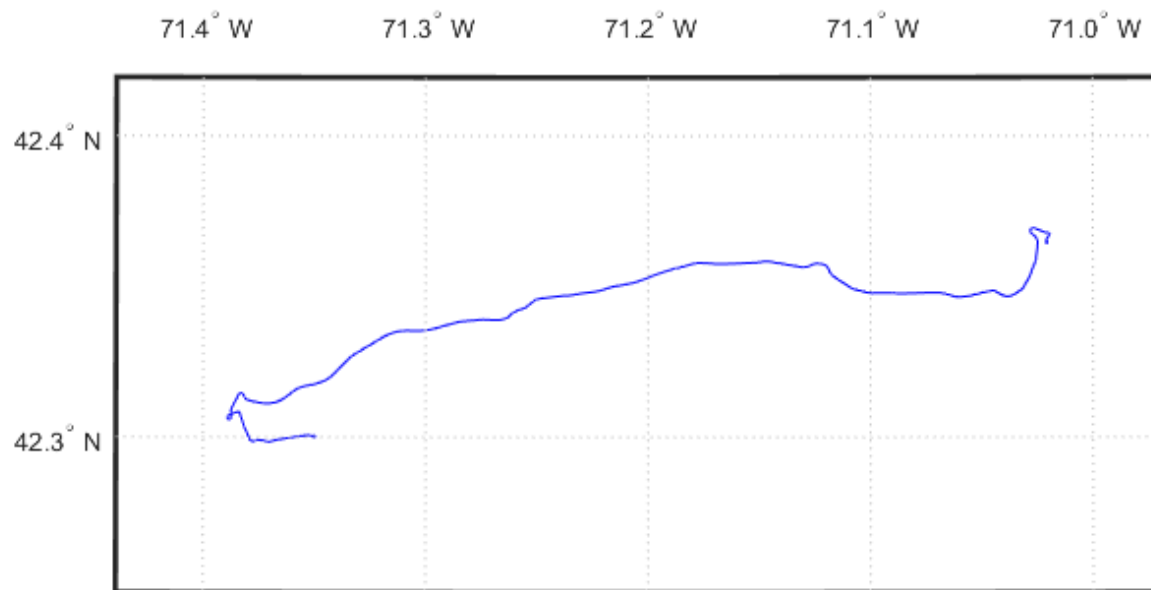
Extract the elements of route that include descriptions of turns, mark and color code each turn on the map, and construct a legend that displays the descriptions. Reverse the order, so that the legend displays the first turn at the top and the last at the bottom.

```
turns = route(~cellfun(@isempty, route.Description));
turns = turns(end:-1:1);
n = length(turns);
colors = cool(n);
for k=1:n
    geoshow(turns(k).Latitude, turns(k).Longitude, ...
        'DisplayType','point','MarkerEdgeColor',colors(k,:),...
        'Tag','turn','DisplayName',turns(k).Description)
end
legend(findobj(ax,'Tag','turn'),'Location','SouthOutside')
```
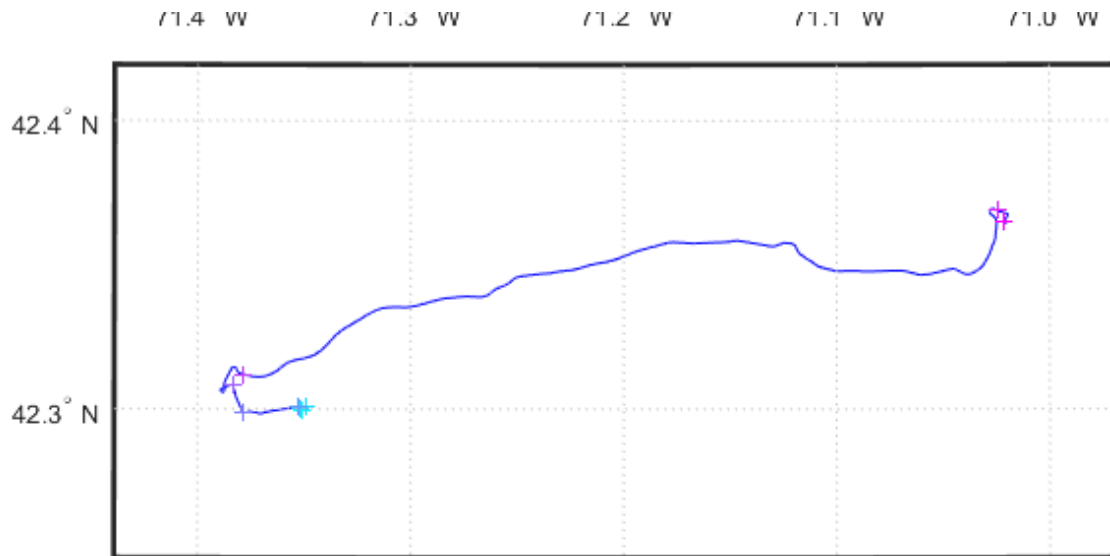
+	Head southeast
+	Keep left at the fork, follow signs for I-90 W/I-93 S/Williams Tunnel/Mass Pike and merge onto I-90 W
	Partial toll road
+	Take exit 13 to merge onto MA-30 E/Cochituate Rd toward Natick
	Partial toll road
+	Turn right onto Speen St
+	Merge onto MA-9 E/Worcester St via the ramp on the left to Boston
+	Slight right onto Apple Hill Dr
+	Turn left toward Apple Hill Dr
+	Take the 1st right onto Apple Hill Dr
	Destination will be on the right
+	The MathWorks, Inc., Natick, MA.

Read and Display Multiple Track Logs on a Web Map

Read track log from a GPX file and display overlaid on a web map.

Read the track logs from a GPX file. `gpxread` returns the data in a geoshape object.

```
tracks = gpxread('sample_tracks', 'Index', 1:2);
```

Display the track logs on a web map with a different color for each track log.

```
webmap('openstreetmap')
colors = {'cyan', 'red'};
wmline(tracks, 'Color', colors)
```
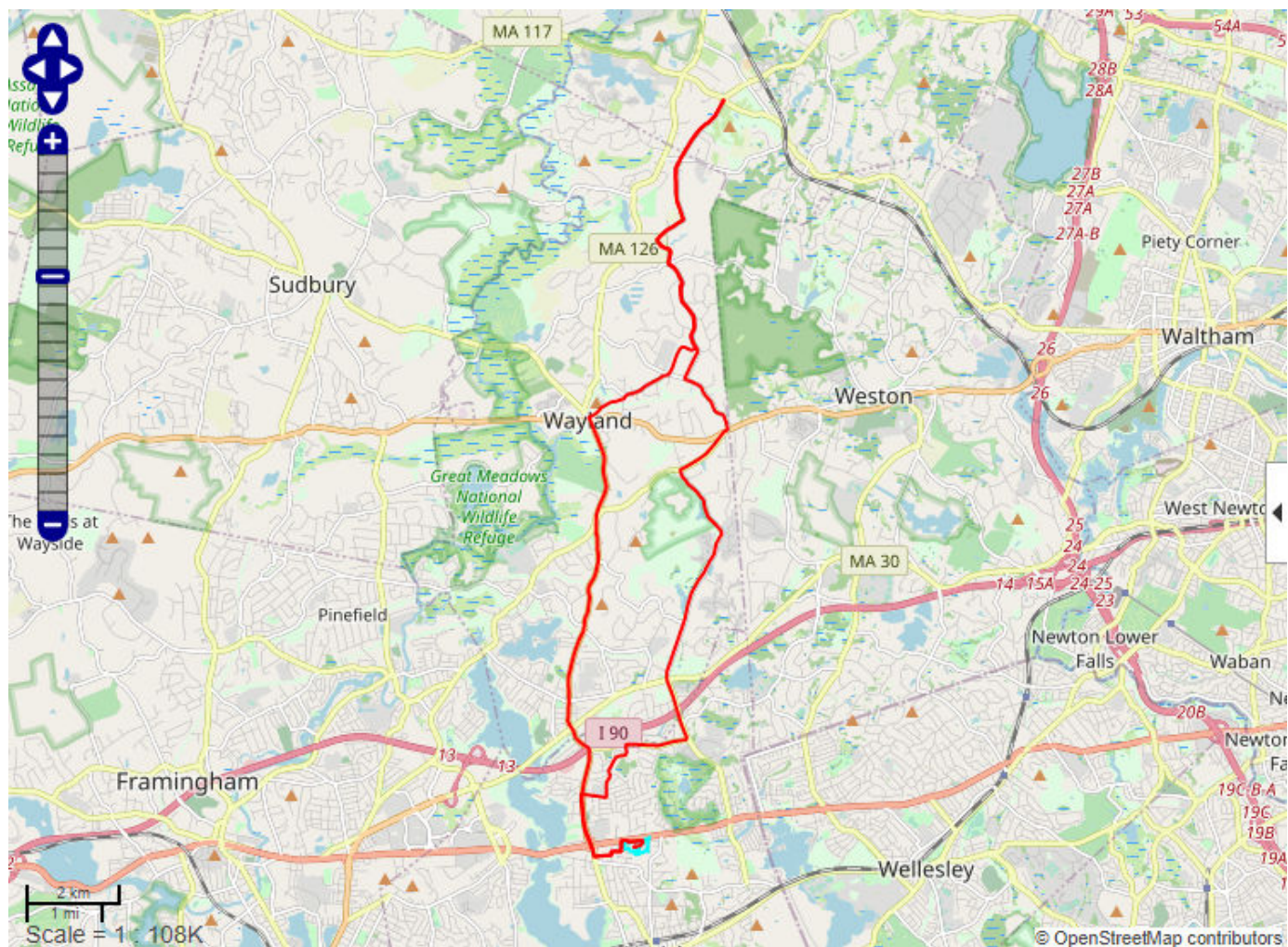
Zoom the web map to view the first track near the MathWorks campus in Natick.

```
[latlim, lonlim] = geoquadline(tracks(1).Latitude, tracks(1).Longitude);
wmlimits(latlim, lonlim)
```
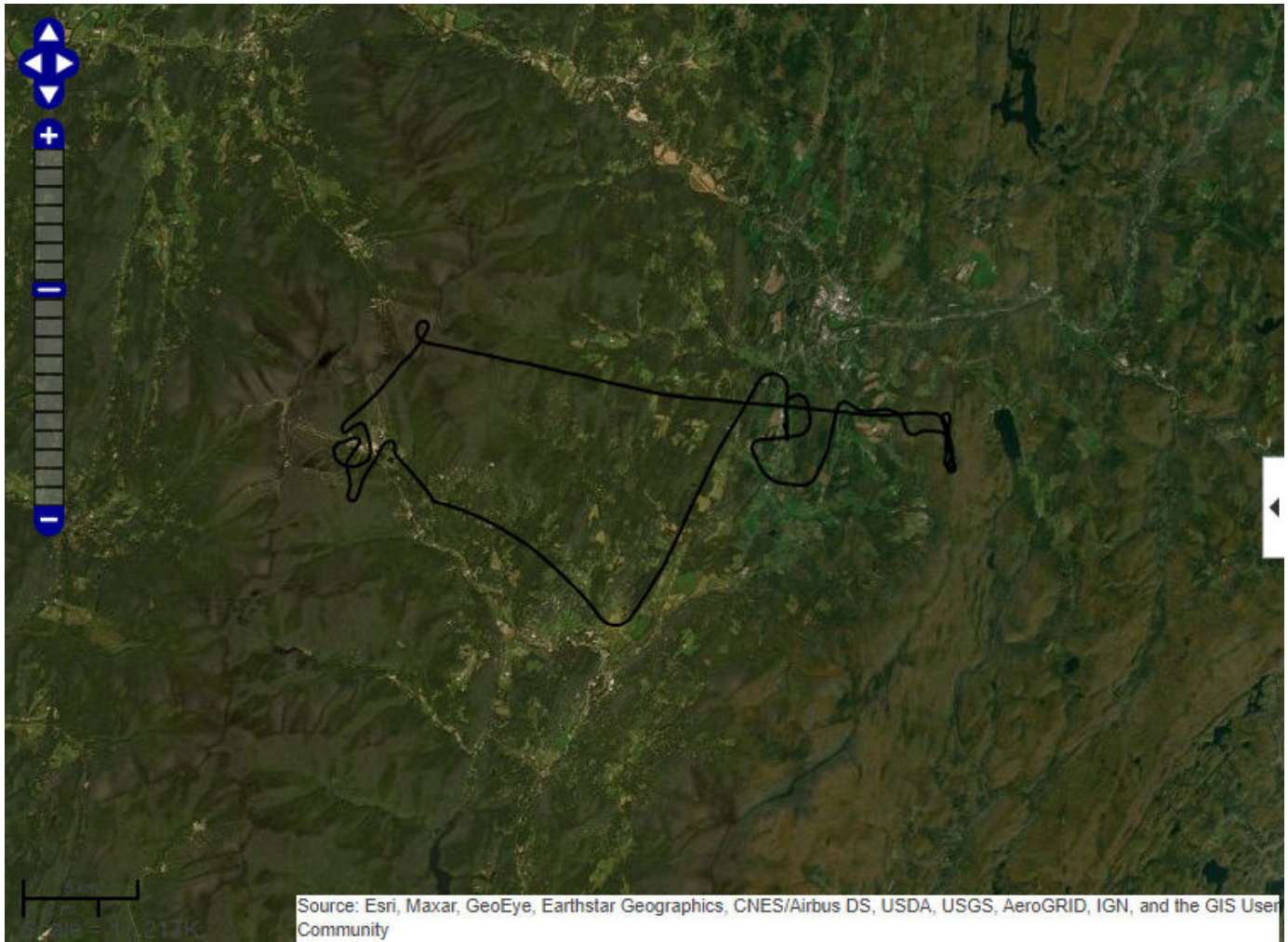
Read and Display Waypoints and Track Log on a Web Map

Read waypoints and track log from the sample_mixed.gpx file.

```
wpt = gpxread('sample_mixed');
trk = gpxread('sample_mixed', 'FeatureType', 'track');
```
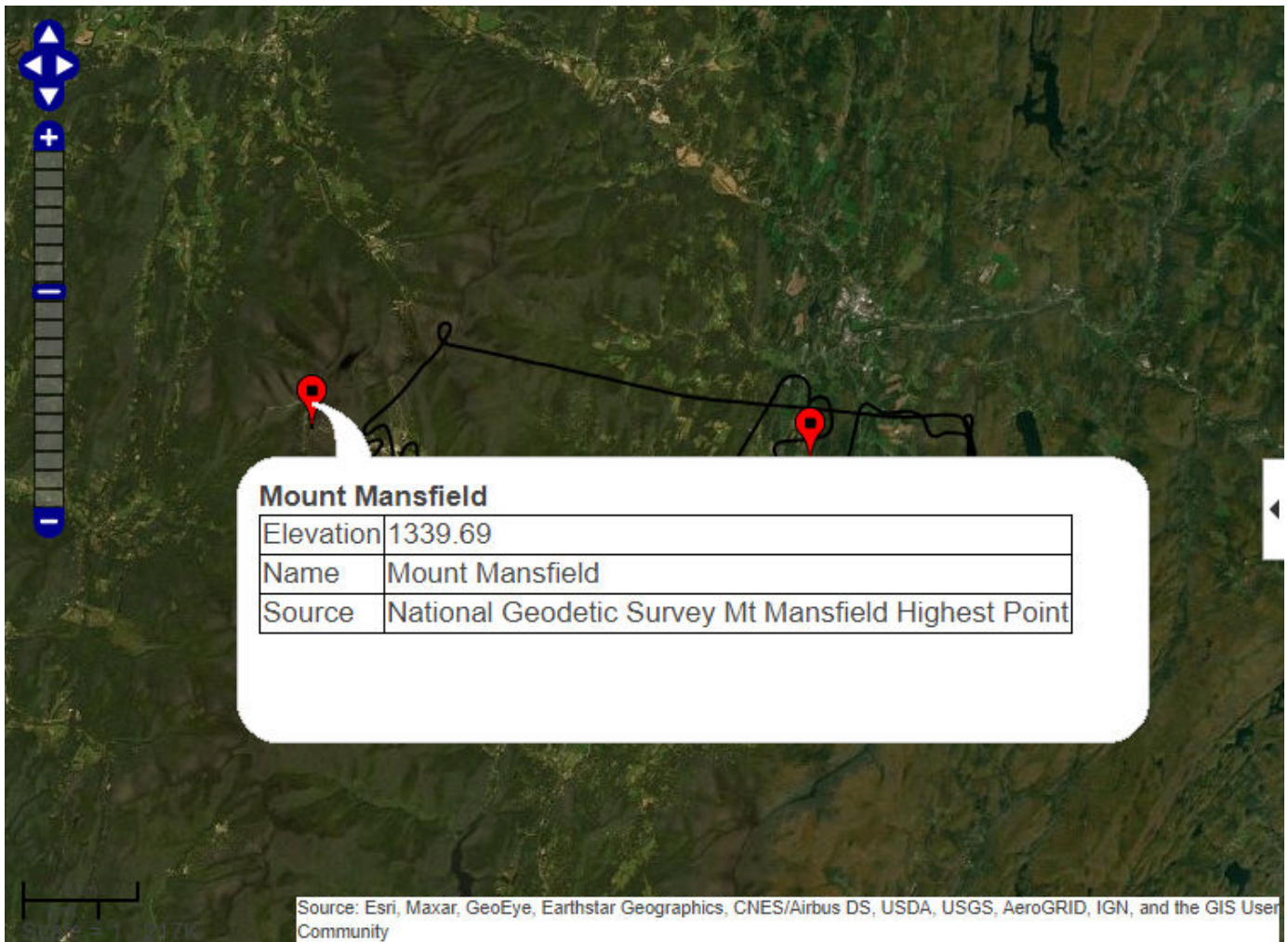
Display the waypoints and the track log on a web map.

```
webmap('worldimagery')
wmline(trk, 'OverlayName', 'Track Logs');
```

Add web markers to mark the positions of each way point.

```
wmmarker(wpt, 'FeatureName', wpt.Name, 'OverlayName', 'Waypoints')
```

Display Elevation and Time-Area Maps

This example shows how to display elevation and time area maps and calculate distance using track logs.

Read the track log from the `sample_mixed.gpx` file.

```
trk = gpxread('sample_mixed.gpx','FeatureType','track');
```

Simplify the format of the time value strings by removing the fractional seconds and time zone offset. Convert the strings to a `datetime` array.

```
timeStr = strrep(trk.Time,'.000Z','');
t = datetime(timeStr);
```

Display an area plot of the time and elevation values.

```
figure
area(t,trk.Elevation)
xtickformat('HH:mm:ss')
```

```
xlabel('time (hours:minutes:seconds)')
ylabel('elevation (meters)')
title('Elevation Area Plot');
```
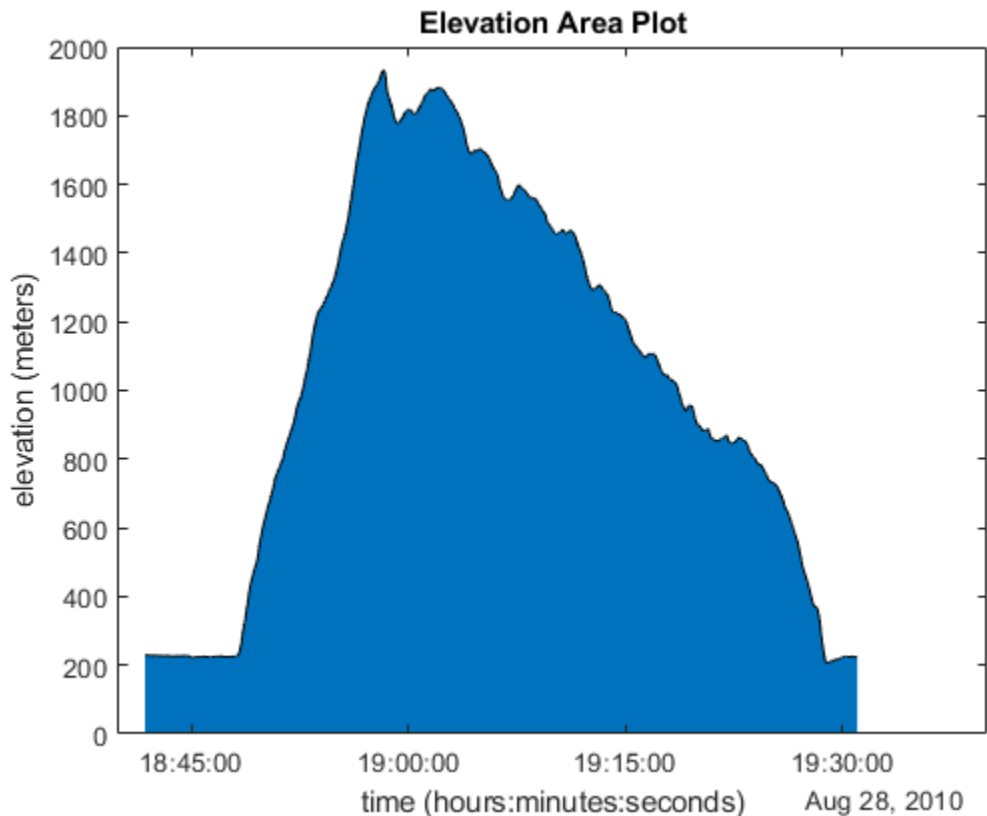

Calculate and display ground track distance. Convert distance in meters to distance in U.S. survey miles.

```
e = wgs84Ellipsoid;
lat = trk.Latitude;
lon = trk.Longitude;
d = distance(lat(1:end-1), lon(1:end-1), lat(2:end), lon(2:end), e);
d = d * unitsratio('sm', 'meter');
```

Display the cumulative ground track distance and elapsed time.

```
elapsedTime = t - t(1);
figure
line(elapsedTime(2:end),cumsum(d))
ylabel('cumulative ground track distance (statute mile)')
xlabel('elapsed time  (hours:minutes:seconds)')
title({'Cumulative Ground Track Distance in Miles',  ...
    ['Total Distance in Miles: ' num2str(sum(d))]});
```
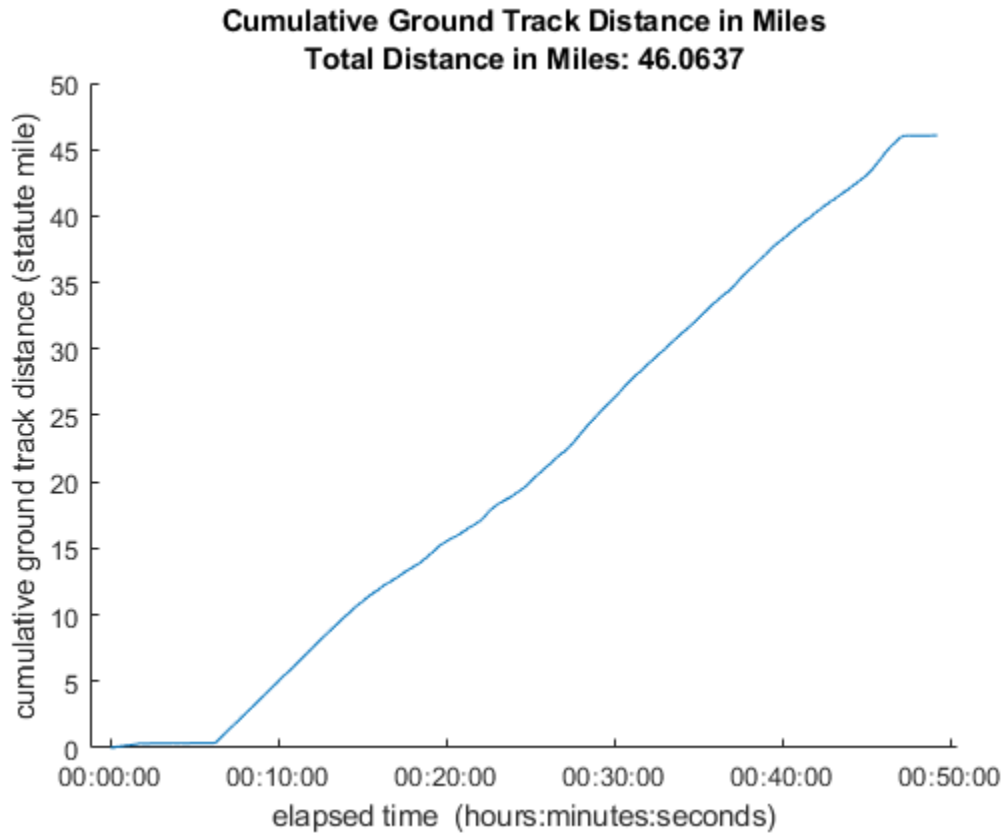
Input Arguments

`filename` — Name of GPX file to open
character vector | string scalar

Name of GPX file to open, specified as a string scalar or character vector. If the file is not in the current folder or in a folder on the MATLAB path, you must specify the folder path. If the file name includes the extension `'.gpx'` (either uppercase or lowercase), you can omit the extension from `filename`.

Example: `'boston_placenames'`

Data Types: `char` | `string`

URL — Internet location containing GPX data
URL

Internet location containing GPX data, specified as a URL. The URL must include protocol type (for example, `https://`).

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'FeatureType','track'`

FeatureType — Type of feature to read from file
`'auto'` (default) | `'track'` | `'route'` | `'waypoint'`

Type of feature to read from file, specified as one of the following: `'track'`, `'route'`, `'waypoint'`, or `'auto'`. If `gpxread` cannot find the specified feature in the file, it returns an empty geopoint vector.

Example: `'FeatureType','waypoint'`

Data Types: `char` | `string`

Index — Index of waypoint, track, or route data in file
scalar or vector of positive integers

Index of waypoint, track, or route data in file, specified as a scalar or vector of positive integers.

- If the value is a scalar, `gpxread` returns the specified waypoint, route, or track as a geopoint vector. If the scalar value is greater than the total number of elements found in the file, `gpxread` returns an empty geopoint vector.

- If the value is a vector, and the file contains waypoints, `gpxread` returns those waypoints specified by the vector. If the file contains routes or tracks (and does not contain waypoints), `gpxread` returns the specified routes or track logs in a geoshape vector. `gpxread` sets the Geometry field of the geoshape vector to `'line'`.

Example: `'Index', [1:2]` would read up to two routes or tracks, if the file contained routes or tracks, in a geoshape vector.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

P — Waypoint, track, or route data
n-by-1 geopoint vector

Waypoint, track, or route data, returned as an *n*-by-1 geopoint vector, where *n* is the number of points.

For a track log or route with multiple segments, `gpxread` concatenates the coordinates of the segments with NaN separators. NaN denotes numeric elements not found in the file. The empty character vector (`''`) is used to denote text elements not found in the file.

S — Track or route data
n-by-1 geoshape vector

Track or route data, returned as an *n*-by-1 geopoint vector

More About

waypoint

A point of interest, or named feature on a map.

track

An ordered list of waypoints that describe a path.

route

An ordered list of waypoints representing a series of turn points leading to a destination.

Tips

- Excluding extensions, GPX version 1.1 is fully supported. If any other version is detected, a warning is issued. However, in most cases, version 1.0 GPX files can be read successfully unless they contain certain metadata tags. For more information, see the GPX 1.1 Schema Documentation.

See Also

geopoint | geoshape | readgeotable | shaperead

Introduced in R2012a

gradientm

Gradient, slope, and aspect of data grid

Syntax

```
[aspect,slope,gradN,gradE] = gradientm(F,R)
[aspect,slope,gradN,gradE] = gradientm(F,R,spheroid)

[aspect,slope,gradN,gradE] = gradientm(lat,lon,F)
[aspect,slope,gradN,gradE] = gradientm(lat,lon,F,spheroid)
[aspect,slope,gradN,gradE] = gradientm(lat,lon,F,spheroid,angleUnit)
```

Description

Regular Data Grids

`[aspect,slope,gradN,gradE] = gradientm(F,R)` returns the aspect angle, slope angle, and north and east components of the gradient for a regular data grid `F` with respect to a geographic reference R. By default, `gradientm` locates the latitude and longitude coordinates referenced by R using the spheroid contained in the `Spheroid` property of the `geocrs` object in the `GeographicCRS` property of R. If the `GeographicCRS` property of R is empty, then `geopeaks` uses GRS80.

`[aspect,slope,gradN,gradE] = gradientm(F,R,spheroid)` uses the specified reference spheroid instead of the spheroid contained in the `Spheroid` property of the `geocrs` object in the `GeographicCRS` property of R or the spheroid GRS80.

Geolocated Data Grids

`[aspect,slope,gradN,gradE] = gradientm(lat,lon,F)` returns the same values for a geolocated data grid `F` with respect to the latitude-longitude mesh defined by `lat` and `lon`. By default, latitude and longitude are in degrees. The default reference spheroid is GRS80.

`[aspect,slope,gradN,gradE] = gradientm(lat,lon,F,spheroid)` uses the specified reference spheroid instead of GRS80.

`[aspect,slope,gradN,gradE] = gradientm(lat,lon,F,spheroid,angleUnit)` specifies the units for latitude and longitude as `'degrees'` (the default) or `'radians'`.

Examples

Calculate and Display Gradient, Slope, and Aspect of Elevation Data

Generate sample elevation data for the region around a mountain summit using a geographic postings reference object and the `geopeaks` function. To do this, first create a reference object for the region by specifying the latitude and longitude limits and the size of the elevation data grid. Next, generate elevation data for the region using `geopeaks`.

```
latlim = [10 45];
lonlim = [60 100];
size = [100 100];
```

```
R = georefpostings(latlim,lonlim,size);
F = geopeaks(R);
```

Compute the aspect angles, slope angles, and gradient components of the data.

```
[aspect,slope,gradN,gradE] = gradientm(F,R);
```

Visualize the results by plotting the data. First, plot the elevation data using an equidistant cylindrical projection. To do this, create a set of map axes and specify the projection using `axesm`. Plot the data as a surface using `geoshow`, and adjust the aspect ratio of the axes using `daspect`. Display a 3-D view of the axes using `view`.

```
figure
axesm('eqdcylin');
geoshow(F,R,'DisplayType','surface')
daspect([1 1 5])
title('Elevation Data')
colorbar
view(3)
```
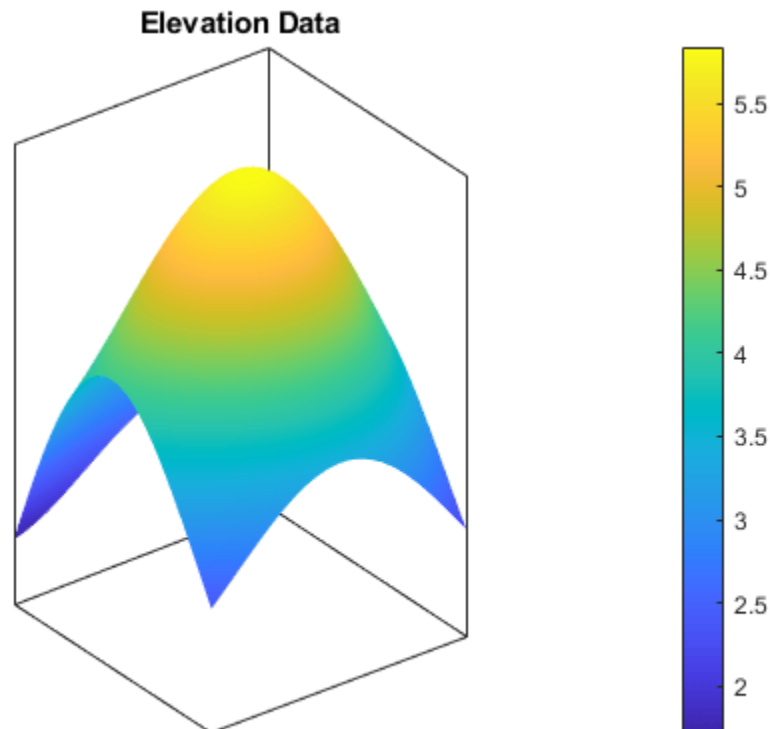

Then, plot the gradient components using the same projection. Note that both the north and east component values are zero at the summit.

```
figure
axesm('eqdcylin')
geoshow(gradN,R,'DisplayType','surface')
title('North Components of Gradient')
colorbar
```
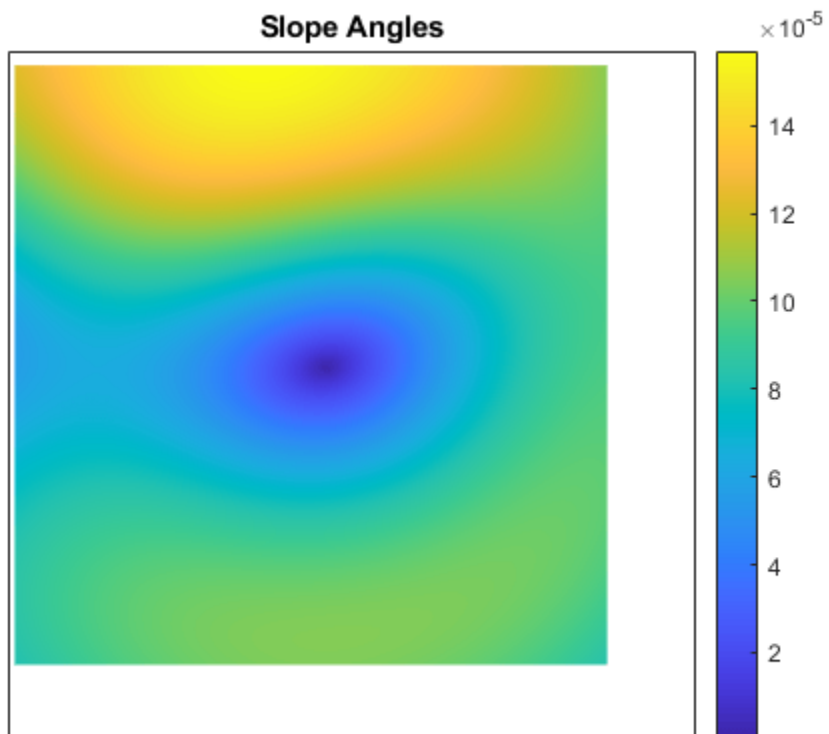
North Components of Gradient
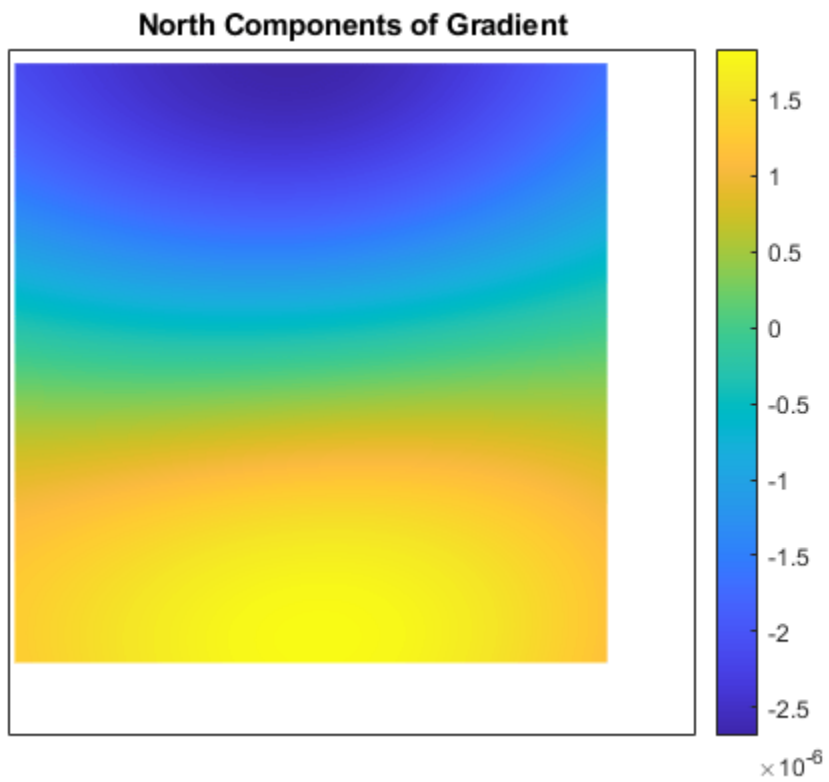
```
figure
axesm('eqdcylin')
geoshow(gradE,R,'DisplayType','surface')
title('East Components of Gradient')
colorbar
```

East Components of Gradient
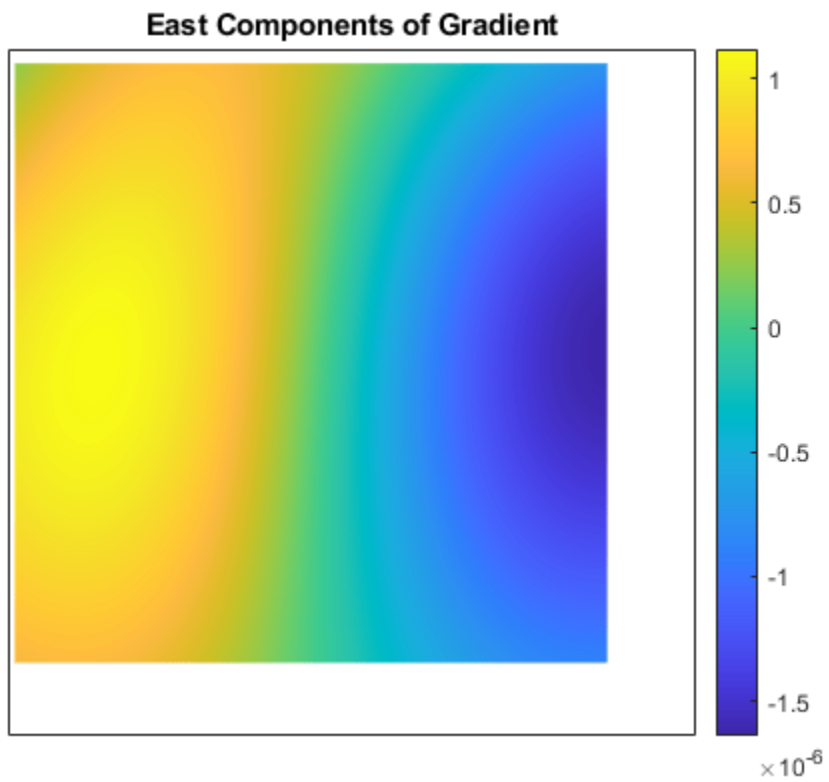
Plot the slope angles. Note that the value of the slope angle is zero at the summit.

```
figure
axesm('eqdcylin')
geoshow(slope,R,'DisplayType','surface')
title('Slope Angles')
colorbar
```

Plot the aspect angles. An aspect angle describes the direction the mountain slope faces as an azimuth measured clockwise from north.

```
figure
axesm('eqdcylin')
geoshow(aspect,R,'DisplayType','surface')
title('Aspect Angles')
colorbar
```

Aspect Angles
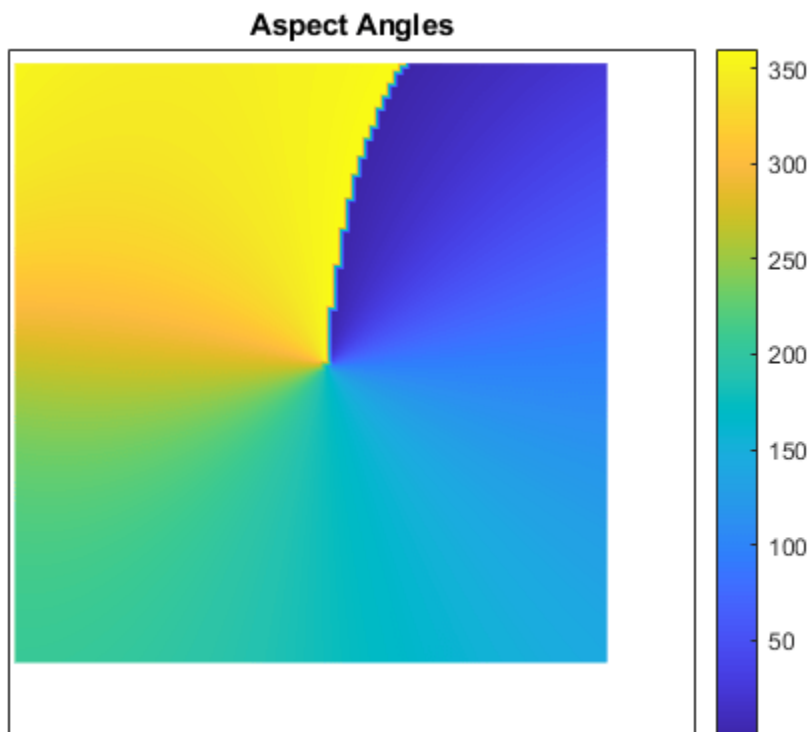
Input Arguments

F — Data grid
numeric matrix

Data grid, specified as a numeric matrix with at least two rows and two columns. The data grid may contain NaN values. F is either a regular data grid associated with a geographic raster reference object, or a georeferenced data grid with respect to a latitude-longitude mesh.

If F is a regular data grid and R is a reference object, then size(F) must be the same as R.RasterSize. If F is a geolocated data grid, then size(F) must be the same as size(lat) and size(lon).

Data Types: single | double

R — Geographic reference
GeographicCellsReference object | GeographicPostingsReference object | 3-by-2 matrix | 1-by-3 vector

Geographic reference that contains geospatial referencing information for F, specified as one of these values:

- GeographicCellsReference or GeographicPostingsReference object, where R.RasterSize is the same as size(F).

- 3-by-2 numeric matrix that associates the row and column indices of a data grid with geographic coordinates, such that `[lon lat] = [row col 1] * R`. The matrix must define a nonrotational and nonskewed relationship in which each column of the data grid falls along a meridian and each row falls along a parallel.
- 1-by-3 numeric vector with elements `[s nlat wlon]`, where `s` is the number of data grid samples per degree, `nlat` is the northernmost latitude of the data grid in degrees, and `wlon` is the westernmost longitude in degrees.

For more information about referencing vectors and matrices, see "Georeferenced Raster Data".

spheroid — Reference spheroid
`'GRS80'` (default) | `referenceEllipsoid` object | `oblateSpheroid` object | `referenceSphere` object | vector

Reference spheroid, specified as a `referenceEllipsoid` object, `oblateSpheroid` object, `referenceSphere` object, or vector of the form `[semimajorAxis eccentricity]`.

For more information about reference spheroids, see "Reference Spheroids".

Example: `spheroid = referenceEllipsoid(wgs84Ellipsoid);`

lat — Latitudes
numeric matrix

Latitudes, specified as a numeric matrix with at least two rows and two columns. By default, specify latitudes in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

`lat` must be the same size as `lon` and `F`.

Data Types: `single` | `double`

lon — Longitudes
numeric matrix

Longitudes, specified as a numeric matrix with at least two rows and two columns. By default, specify longitudes in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

`lon` must be the same size as `lat` and `F`.

Data Types: `single` | `double`

angleUnit — Angle units
`'degrees'` (default) | `'radians'`

Angle units, specified as `'degrees'` (the default) or `'radians'`.

Output Arguments

aspect — Aspect angles
matrix

Aspect angles, returned as a matrix of the same size as `F`. An aspect angle is the direction in which `F` decreases most rapidly, expressed as an azimuth measured clockwise from north.

By default, aspect angles are in degrees. To return values in radians, specify `lat` and `lon` in radians and `angleUnit` as `'radians'`.

If both components of the gradient are zero, then the aspect angle is returned as NaN.

Data Types: double

slope — Slope angles
matrix

Slope angles, returned as a matrix of the same size as F. For the slope angles to have physical meaning, the data grid must specify elevation, and its distance unit must match the length unit of the reference spheroid. Otherwise, a slope angle is the arctangent of the magnitude of the gradient.

By default, slope angles are in degrees. To return values in radians, specify lat and lon in radians and angleUnit as 'radians'.

Data Types: double

gradN — North components of gradient
matrix

North components of the gradient, returned as a matrix of the same size as F. The north component of a gradient is the change in R per unit of distance in the north direction, where the distance unit matches the length unit of the reference spheroid.

Data Types: double

gradE — East components of gradient
matrix

East components of the gradient, returned as a matrix of the same size as F. The east component of a gradient is the change in R per unit of distance in the east direction, where the distance unit matches the length unit of the reference spheroid.

Data Types: double

See Also

Functions
gradient | geoshow | axesm

Objects
GeographicPostingsReference | GeographicCellsReference

Topics
"Geolocated Data Grids"

Introduced before R2006a

grepfields

Identify matching fields in fixed record length files

Note `grepfields` will be removed in a future release. Use `textscan` instead.

Syntax

```
grepfields(filename,searchstring)
grepfields(filename,searchstring,casesens)
grepfields(filename,searchstring,casesens,startcol)
grepfields(filename,searchstring,casesens,startfield,fields)
grepfields(filename,searchstring,casesens,startfield,fields, machineformat)
indx = grepfields(...)
```

Description

`grepfields(filename,searchstring)` displays lines in the file that begin with the search character vector. The file must have fixed-length records with line endings.

`grepfields(filename,searchstring,casesens)`, with `casesens 'matchcase'`, specifies a case-sensitive search. If omitted or `'none'`, the search character vector matches regardless of the case.

`grepfields(filename,searchstring,casesens,startcol)` searches starting with the specified column. `startcol` is an integer between 1 and the bytes per record in the file. In this calling form, the file is regarded as a text file with line endings.

`grepfields(filename,searchstring,casesens,startfield,fields)` searches within the specified field. `startfield` is an integer between 1 and the number of fields per record. The format of the file is described by the `fields` structure. See `readfields` for recognized fields structure entries. In this calling form, the file can be binary and lack line endings. The search is within `startfield`, which must be a character field.

`grepfields(filename,searchstring,casesens,startfield,fields, machineformat)` opens the file with the specified machine format. `machineformat` must be recognized by `fopen`.

`indx = grepfields(...)` returns the record numbers of matched records instead of displaying them on screen.

Examples

Write a binary file and read it:

```
fid = fopen('testbin','wb');
for i = 1:3
    fwrite(fid,['character' num2str(i) ],'char');
    fwrite(fid,i,'int8');
    fwrite(fid,[i i],'int16');
    fwrite(fid,i,'integer*4');
```

```
    fwrite(fid,i,'real*8');
end
fclose(fid);

fs(1).length = 10;fs(1).type = 'char';fs(1).name = 'field 1';
fs(2).length = 1;fs(2).type = 'int8';fs(2).name = 'field 2';
fs(3).length = 2;fs(3).type = 'int16';fs(3).name = 'field 3';
fs(4).length = 1;fs(4).type = 'integer*4';fs(4).name = 'field 4';
fs(5).length = 1;fs(5).type = 'float64';fs(5).name = 'field 5';
```

Find the record matching the character vector `'character2'`. The record contains binary data, which cannot be properly displayed.

```
grepfields('testbin','character2','none',1,fs)
character2? ? ?   ?@

indx = grepfields('testbin','character2','none',1,fs)
indx =
     2
```

Read the formatted file containing the following:

```
----------------------------------------------------------
character data 1  1  2  3 1e6 10D6

character data 2 11 22 33 2e6 20D6

character data 3111222333 3e6 30D6
----------------------------------------------------------

fs(1).length = 16;fs(1).type = 'char';fs(1).name = 'field 1';
fs(2).length = 3;fs(2).type = '%3d';fs(2).name = 'field 2';
fs(3).length = 1;fs(3).type = '%4g';fs(3).name = 'field 3';
fs(4).length = 1;fs(4).type = '%5D';    fs(4).name = 'field 4';
fs(5).length = 1;fs(5).type = 'char';fs(5).name = '';
```

Find the records that match at the beginning of the line.

```
grepfields('testfile1','character')
character data 1  1  2  3 1e6 10D6
character data 2 11 22 33 2e6 20D6
character data 3111222333 3e6 30D6

grepfields('testfile1','character data 2')
character data 2 11 22 33 2e6 20D6
```

Find the records that match, starting the search in column 11.

```
grepfields('testfile1','data 2','none',11)
character data 2 11 22 33 2e6 20D6
```

Search record number 1.

```
grepfields('testfile1','character data 2','none',1,fs)
character data 2 11 22 33 2e6 20D6
```

Limitations

Searches are limited to fields containing character data.

Tips

See `readfields` for a complete discussion of the format and contents of the `fields` argument.

See Also
`readfields` | `fopen`

Introduced before R2006a

gridm

Toggle and control display of graticule lines

Syntax

```
gridm
gridm('on')
gridm('off')
gridm('reset')
gridm(linespec)
gridm(MapAxesPropertyName, PropertyValue,...)
h = gridm(...)
```

Description

`gridm` toggles the display of a latitude-longitude graticule. The choice of meridians and parallels, as well as their graphics properties, depends on the property settings of the map axes.

`gridm('on')` creates the graticule, if it does not yet exist, and makes it visible.

`gridm('off')` makes the graticule invisible.

`gridm('reset')` redraws the graticule using the current map axes properties.

`gridm(linespec)` uses any valid `linespec` to control the graphics properties of the lines in the graticule.

`gridm(MapAxesPropertyName, PropertyValue,...)` sets the appropriate graticule properties to the desired values. For a description of these property names and values, see "Properties That Control the Grid" on page 1-0   section of the `axesm` property reference page.

`h = gridm(...)` returns the handles of the graticule lines. If both parallels and meridians exist, then `h` is a two-element vector: `h(1)` is the handle to the line comprising the parallels, and `h(2)` is the handle to the line comprising the meridians.

Tips

- You can also create or alter map grid properties using the `axesm` or `setm` functions.
- By default the Clipping property is set to `'off'`. Override this setting with the following code:

```
hgrat = gridm('on');
set(hgrat,'Clipping','on')
```

See Also
`axesm` | `setm`

Introduced before R2006a

grid2image

Display regular data grid as image

Syntax

```
grid2image(Z,R)
grid2image(Z,R,PropertyName,PropertyValue,...)
h = grid2image(...)
```

Description

`grid2image(Z,R)` displays a regular data grid Z as an image. The image is displayed in unprojected form, with longitude as *x* and latitude as *y*, producing considerable distortion away from the Equator. Z can be *M*-by-*N* or *M*-by-*N*-by-3, and can contain `double`, `uint8`, or `uint16` data. The grid is georeferenced to latitude-longitude by R, which can be a geographic raster reference object, a referencing vector, or a referencing matrix.

If R is a geographic raster reference object, its `RasterSize` property must be consistent with `size(Z)` and its `RasterInterpretation` must be `'cells'`.

If R is a referencing vector, it must be a 1-by-3 with elements:

```
[cells/degree northern_latitude_limit western_longitude_limit]
```

If R is a referencing matrix, it must be 3-by-2 and transform raster row and column indices to/from geographic coordinates according to:

```
[lon lat] = [row col 1] * R
```

If R is a referencing matrix, it must define a (non-rotational, non-skewed) relationship in which each column of the data grid falls along a meridian and each row falls along a parallel.

`grid2image(Z,R,PropertyName,PropertyValue,...)` uses the specified image properties to display the map. See the `image` function reference page for a list of properties that can be changed.

`h = grid2image(...)` returns the handle of the image object displayed.

Examples

Load elevation raster data and a geographic cells reference object. Then, display the data as an image.

```
load topo60c
figure
grid2image(topo60c,topo60cR)
```
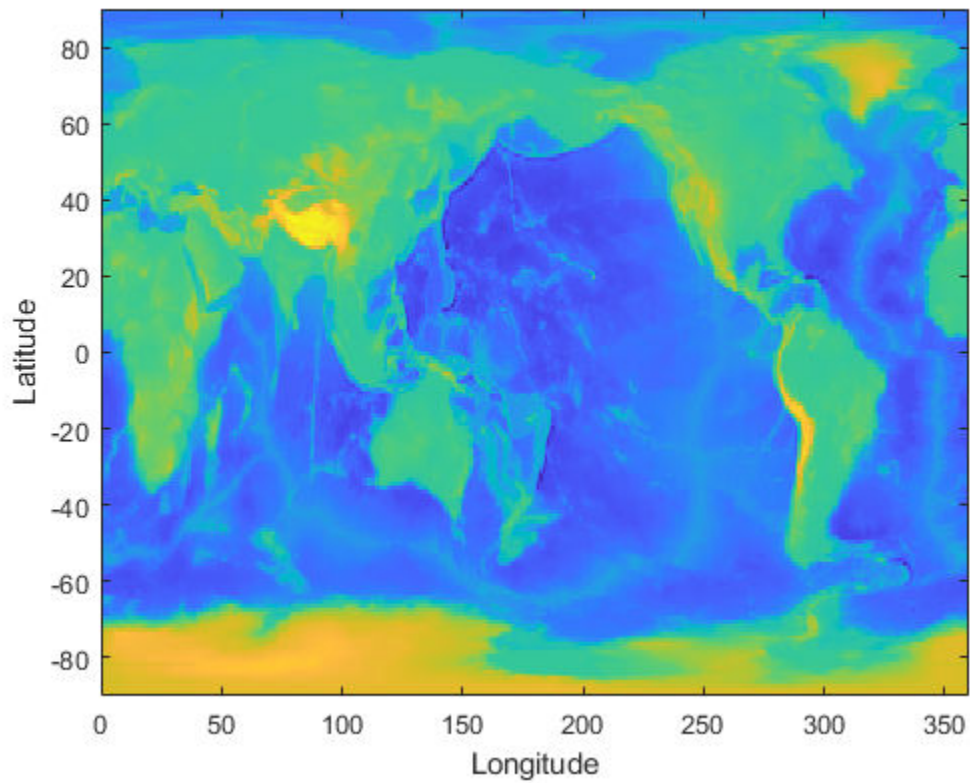
See Also

image | mapshow | mapview | meshm | surfacem | surfm

Introduced before R2006a

grn2eqa

Convert from Greenwich to equal area coordinates

Syntax

```
[x,y] = grn2eqa(lat,lon)
[x,y] = grn2eqa(lat,lon,origin)
[x,y] = grn2eqa(lat,lon,origin,ellipsoid)
[x,y] = grn2eqa(lat,lon,origin,units)
mat = grn2eqa(lat,lon,origin...)
```

Description

`[x,y] = grn2eqa(lat,lon)` converts the Greenwich coordinates `lat` and `lon` to the equal-area coordinate points x and y.

`[x,y] = grn2eqa(lat,lon,origin)` specifies the location in the Greenwich system of the *x-y* origin (0,0). The two-element vector `origin` must be of the form `[latitude, longitude]`. The default places the origin at the Greenwich coordinates (0º,0º).

`[x,y] = grn2eqa(lat,lon,origin,ellipsoid)` specifies the ellipsoidal model of the figure of the Earth using `ellipsoid`. `ellipsoid` is a `referenceSphere`, `referenceEllipsoid`, or `oblateSpheroid` object, or a vector of the form `[semimajor_axis eccentricity]`. The `ellipsoid` is spherical by default.

`[x,y] = grn2eqa(lat,lon,origin,units)` specifies the units for the inputs, where `units` is any valid angle units value. The default value is `'degrees'`.

`mat = grn2eqa(lat,lon,origin...)` packs the outputs into a single variable.

The `grn2eqa` function converts data from Greenwich-based latitude-longitude coordinates to equal-area *x-y* coordinates. The opposite conversion can be performed with `eqa2grn`.

Examples

```
lats = [56 34]; longs = [-140 23];
[x,y] = grn2eqa(lats,longs)

x =
   -2.4435    0.4014
y =
    0.8290    0.5592
```

See Also
eqa2grn | hista

Introduced before R2006a

gshhs

Read Global Self-Consistent Hierarchical High-Resolution Geography

Syntax

```
S = gshhs(filename)
S = gshhs(filename, latlim, lonlim)
indexfilename = gshhs(filename, 'createindex')
```

Description

S = gshhs(filename) reads GSHHG (formerly GSHHS) vector data for the entire world from filename. GSHHG files must have names of the form gshhs_*x*.b, wdb_borders_*x*.b, or wdb_rivers_*x*.b, where *x* is one of the letters c, l, i, h or f, corresponding to increasing resolution (and file size). The result returned in S is a polygon or line geographic data structure array (a *geostruct*, with 'Lat' and 'Lon' coordinate fields).

S = gshhs(filename, latlim, lonlim) reads a subset of the vector data from filename. The limits of the desired data are specified as two-element vectors of latitude, latlim, and longitude, lonlim, in degrees. The elements of latlim and lonlim must be in ascending order. Longitude limits range from [-180 195]. If latlim is empty, the latitude limits are [-90 90]. If lonlim is empty, the longitude limits are [-180 195].

indexfilename = gshhs(filename, 'createindex') creates an index file for faster data access when requesting a subset of a larger dataset. The index file has the same name as the GSHHG data file, but with the extension 'i', instead of 'b' and is written in the same folder as filename. The name of the index file is returned, but no coastline data are read. A call using this option should be followed by an additional call to gshhs to import actual data. On that and subsequent calls, gshhs detects the presence of the index file and uses it to access records by location much faster than it would without an index.

Output Structure

The output structure S contains the following fields. All latitude and longitude values are in degrees.

Field Name	Field Contents
'Geometry'	'Line' or 'Polygon'
'BoundingBox'	[minLon minLat; maxLon maxLat]
'Lon'	Coordinate vector
'Lat'	Coordinate vector
'South'	Southern latitude boundary
'North'	Northern latitude boundary
'West'	Western longitude boundary
'East'	Eastern longitude boundary
'Area'	Area of polygon in square kilometers

Field Name	Field Contents
`'Level'`	Scalar value ranging from 1 to 4, indicates level in topological hierarchy
`'LevelString'`	`'land'`, `'lake'`, `'island_in_lake'`, `'pond_in_island_in_lake'`, or `''`
`'NumPoints'`	Number of points in the polygon
`'FormatVersion'`	Format version of data file. Positive integer for versions 3 and later; empty for versions 1 and 2.
`'Source'`	Source of data: `'WDBII'` or `'WVS'`
`'CrossesGreenwich'`	Scalar flag: `true` if the polygon crosses the prime meridian; `false` otherwise
`'GSHHS_ID'`	Unique polygon scalar id number, starting at 0

For releases 2.0 and higher (`FormatVersion` 7 and higher), the following additional fields are included in the output structure:

Field Name	Field Contents
`'RiverLake'`	Scalar flag: `true` if the polygon is the fat part of a major river and the `Level` value is set to 2; `false` otherwise.
`'AreaFull'`	Area of original full-resolution polygon in units $\frac{1}{10}km^2$.
`'Container'`	ID of container polygon that encloses this polygon. Set to -1 to indicate none.
`'Ancestor'`	ID of ancestor polygon in the full resolution set that was the source of this polygon. Set to -1 to indicate none.

For Release 2.2 and higher (`FormatVersion` 9 and higher) the following additional field is included in the output structure:

Field Name	Field Contents
`'CrossesDateline'`	Scalar flag: `true` if the polygon crosses the dateline; `false` otherwise.

Background

The Global Self-Consistent Hierarchical High-Resolution Geography (formerly the Global Self-Consistent Hierarchical High-Resolution Shoreline) was created by Paul Wessel of the University of Hawaii and Walter H.F. Smith of the NOAA Geosciences Lab. At the full resolution, the data requires 85 MB uncompressed, but lower resolution versions are also provided. This database includes coastlines, major rivers, and lakes. The GSHHG data in various resolutions is available over the Internet from the National Oceanic and Atmospheric Administration, National Geophysical Data Center website.

Version 3 (Release 1.3) of the `gshhs_c.b` (coarse) data set ships with the toolbox in the *matlabroot*/`examples/map/data` folder. For details, type

```
type gshhs_c.txt
```

at the MATLAB command prompt. The `gshhs` function has been qualified on GSHHG releases 1.1 through 2.3.6 (version 15). It should also be able to read newer versions, if they adhere to the same header format as releases 2.0 and 2.1.
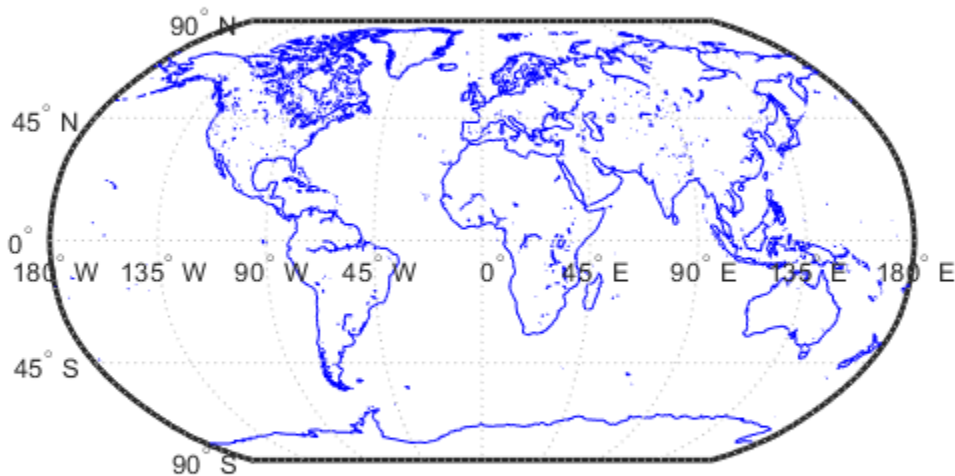
Examples

Read Entire Coarse Data Set and Display Levels

Read the entire coarse data set. You can use similar code to read intermediate, full resolution, or high resolution GSHHG data sets.

```
filename = gunzip('gshhs_c.b.gz', tempdir);
shorelines = gshhs(filename{1});
delete(filename{1})
```

Display data as a coastline.

```
figure
worldmap world
geoshow([shorelines.Lat],[shorelines.Lon])
```


Display each level using a different color.

```
levels = [shorelines.Level];
land = (levels == 1);
lake = (levels == 2);
```

```
island = (levels == 3);
figure
worldmap world
geoshow(shorelines(land),  'FaceColor',[0 1 0])
geoshow(shorelines(lake),  'FaceColor',[0 0 1])
geoshow(shorelines(island),'FaceColor',[1 1 0])
```
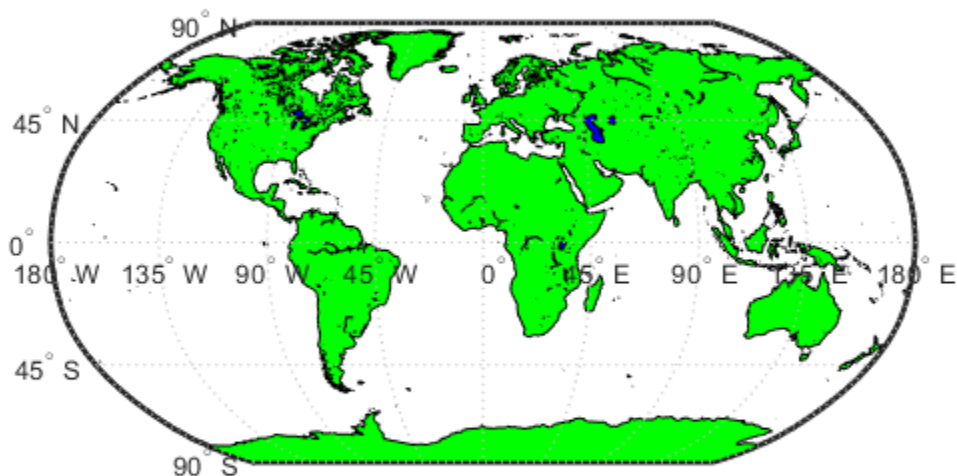

Read GSHHG Dataset Using an Index

Read the entire coarse data set, creating an index. You can use similar code to read intermediate, full-resolution, or high resolution GSHHG data sets.

```
filename = gunzip('gshhs_c.b.gz', tempdir);
indexname = gshhs(filename{1}, 'createindex');
```

Display Africa as a green polygon. Note that `gshhs` detects and uses the index file automatically.
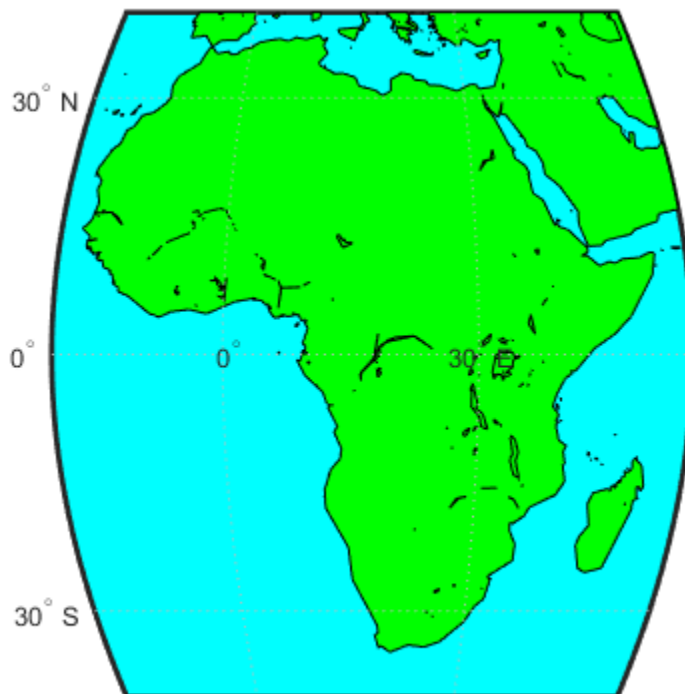
```
figure
worldmap Africa
```
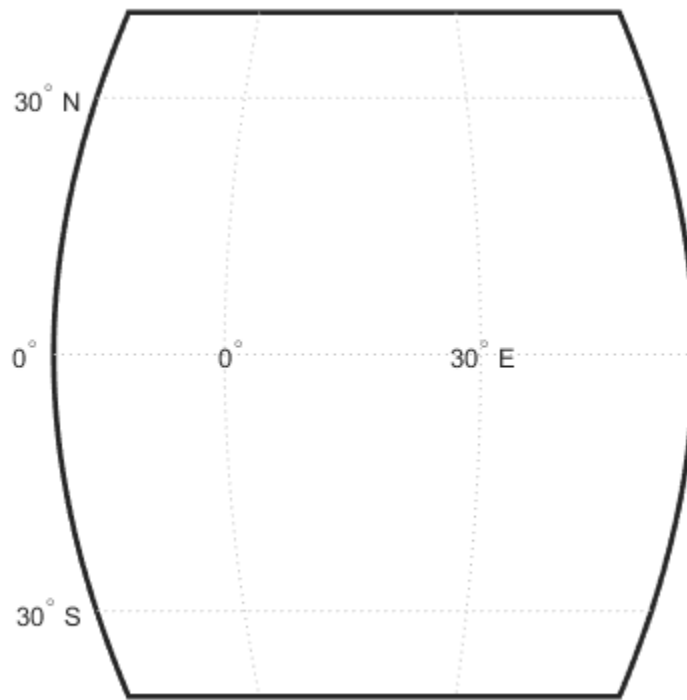
```
projection = gcm;
latlim = projection.maplatlimit;
lonlim = projection.maplonlimit;
africa = gshhs(filename{1}, latlim, lonlim);
delete(filename{1})
delete(indexname)
```

Sort by descending level to keep smaller level 2 and level 3 features on top.

```
[~,ix] = sort([africa.Level],'descend');
africa = africa(ix);
geoshow(africa, 'FaceColor', 'green')
setm(gca, 'FFaceColor', 'cyan')
```

Tips

- If you are extracting data within specified geographic limits and using data other than coarse resolution, consider creating an index file first. Also, to speed rendering when mapping very large amounts of data, you might want to plot the data as NaN-clipped lines rather than as patches.

- When you specify latitude-longitude limits, polygons that completely fall outside those limits are excluded, but no trimming of features that partially traverse the region is performed. If you want to eliminate data outside of a rectangular region of interest, you can use maptrimp with the Lat and Lon fields of the geostruct returned by gshhs to clip the data to your region and still maintain polygon topology.

- You can read the WDB rivers and borders datasets but the LevelString field will be empty. The Level values vary from feature to feature but the interpretations of these values are not documented as part of the GSHHG distribution and are therefore not converted to character vectors.

See Also

dcwdata | geoshow | maptrimp | shaperead | vmap0data | worldmap

Introduced before R2006a

gtextm

Place text on map using mouse

Syntax

```
h = gtextm(text)
h = gtextm(text,PropertyName,PropertyValue,...)
```

Description

`h = gtextm(text)` places `text`, a string scalar or character vector, at the position selected by mouse input. When you call this function, `gtextm` brings up the current map axes and activates the cursor for mouse-click position entry. `gtextm` returns a text object.

`h = gtextm(text,PropertyName,PropertyValue,...)` allows the specification of any properties supported by the MATLAB `text` function.

Examples

Create map axes:

```
axesm('sinusoid','FEdgeColor','red')
gtextm('hello world','FontWeight','bold')
```

Click inside the frame and the text appears.

See Also

`axesm` | `textm`

Introduced before R2006a

gtopo30

(To be removed) Read 30-arc-second global digital elevation data (GTOPO30)

Note `gtopo30` will be removed in a future release. Use `readgeoraster` instead. For more information, see "Compatibility Considerations".

Syntax

```
[Z,refvec] = gtopo30(tilename)
[Z,refvec] = gtopo30(tilename,samplefactor)
[Z,refvec] = gtopo30(tilename,samplefactor,latlim,lonlim)
[Z,refvec] = gtopo30(foldername, ...)
```

Description

`[Z,refvec] = gtopo30(tilename)` reads the GTOPO30 tile specified by `tilename` and returns the result as a regular data grid. `tilename` is a string scalar or character vector which does not include an extension and indicates a GTOPO30 tile in the current folder or on the MATLAB path. If `tilename` is empty or omitted, a file browser will open for interactive selection of the GTOPO30 header file. The data is returned at full resolution with the latitude and longitude limits determined from the GTOPO30 tile. The data grid, `Z`, is returned as an array of elevations. Elevations are given in meters above mean sea level using WGS84 as a horizontal datum. `refvec` is the associated referencing vector.

`[Z,refvec] = gtopo30(tilename,samplefactor)` reads a subset of the elevation data from `tilename`. `samplefactor` is a scalar integer, which when equal to 1 reads the data at its full resolution. When `samplefactor` is an integer n greater than one, every nth point is read. If `samplefactor` is omitted or empty, it defaults to 1.

`[Z,refvec] = gtopo30(tilename,samplefactor,latlim,lonlim)` reads a subset of the elevation data from `tilename` using the latitude and longitude limits `latlim` and `lonlim` specified in degrees. `latlim` is a two-element vector of the form:

`[southern_limit northern_limit]`

Likewise, `lonlim` has the form:

`[western_limit eastern_limit]`

If `latlim` and `lonlim` are omitted, the coordinate limits are determined from the file. The latitude and longitude limits are snapped outward to define the smallest possible rectangular grid of GTOPO30 cells that fully encloses the area defined by the input limits. Any cells in this grid that fall outside the extent of the tile are filled with NaN.

`[Z,refvec] = gtopo30(foldername, ...)` is similar to the syntaxes above except that GTOPO30 data are read and concatenated from multiple tiles within a GTOPO30 CD-ROM or folder structure. The `foldername` input is a string scalar or character vector with the name of the folder which contains the GTOPO30 tile folders or GTOPO30 tiles. Within the tile folders are the uncompressed data files. The `foldername` for CD-ROMs distributed by the USGS is the device name

of the CD-ROM drive. As with the case with a single tile, any cells in the grid specified by `latlim` and `lonlim` are NaN filled if they are not covered by a tile within `foldername`. `samplefactor` if omitted or empty defaults to 1. `latlim` if omitted or empty defaults to `[-90 90]`. `lonlim` if omitted or empty defaults to `[-180 180]`.

For details on locating GTOPO30 data for download over the Internet, see "Find Geospatial Data Online".

Examples

Display Data Grid and Overlay State Line Boundary

To run this example, you must download the GTOPO30 data set. For details on locating this data set for download over the Internet, see "Find Geospatial Data Online".

Extract and display full resolution data for the state of Massachusetts. Read the state line polygon boundary and calculate boundary limits.

```
Massachusetts = shaperead('usastatehi','UseGeoCoords',true, ...
  'Selector',{@(name) strcmpi(name,'Massachusetts'),'Name'});
latlim = [min(Massachusetts.Lat(:)) max(Massachusetts.Lat(:))];
lonlim = [min(Massachusetts.Lon(:)) max(Massachusetts.Lon(:))];
```

Read the GTOPO30 data at full resolution.

```
[Z,refvec] = gtopo30('W100N90',1,latlim,lonlim);
```

Display the data grid and overlay the state line boundary.

```
figure
ax = usamap(Z,refvec);
ax.SortMethod = 'ChildOrder';
geoshow(Z,refvec,'DisplayType','surface')
demcmap(Z)
geoshow(Massachusetts,'DisplayType','polygon',...
  'facecolor','none','edgecolor','y')
```
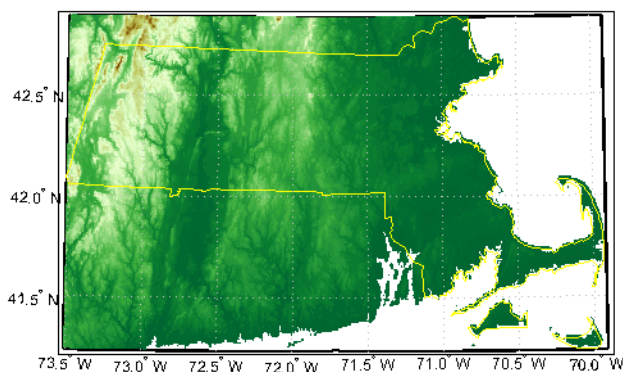

Compatibility Considerations

gtopo30 will be removed
Warns starting in R2021b

Raster reading functions that return referencing vectors will be removed, including `gtopo30`. Instead, use `readgeoraster`, which returns a raster reference object. Reference objects have several advantages over referencing vectors.

- Unlike referencing vectors, reference objects have properties that document the size of the associated raster, its geographic limits, and the direction of its rows and columns. For examples of reference object properties, see `GeographicCellsReference` and `GeographicPostingsReference`.
- You can manipulate the limits of rasters associated with reference objects using the `geocrop` function.
- You can manipulate the size and resolution of rasters associated with reference objects using the `georesize` function.
- Most functions that accept referencing vectors as input also accept reference objects.

This table shows some typical usages of `gtopo30` and how to update your code to use `readgeoraster` instead. Unlike `gtopo30`, the `readgeoraster` function requires you to specify a file extension. For example, use `[Z,R] = readgeoraster('W100N90.dem')`.

Will Be Removed	Recommended
`[Z,refvec] = gtopo30(filename);`	`[Z,R] = readgeoraster(filename);`
`[Z,refvec] = gtopo30(filename,samplefactor)`	`[Z,R] = readgeoraster(filename);` `[Z,R] = georesize(Z,R,1/samplefactor);`

`[Z,refvec] = gtopo30(filename,samplefactor,latlim,lonlim)`	`[Z,R] = readgeoraster(filename);`
	`[Z,R] = geocrop(Z,R,latlim,lonlim);`
	`[Z,R] = georesize(Z,R,1/scalefactor);`

The `readgeoraster` function returns data using the native data type embedded in the file. Return a data using a different data type by specifying the `'OutputType'` name-value pair. For example, use `[Z,R] = readgeoraster(filename,'OutputType','double')`.

The `readgeoraster` function does not automatically replace missing data with `NaN` values. If your data set uses large negative numbers to indicate missing data, you can replace them with `NaN` values using the `standardizeMissing` function.

```
[Z,R] = readgeoraster('MtWashington-ft.grd');
info = georasterinfo('MtWashington-ft.grd');
m = info.MissingDataIndicator;
Z = standardizeMissing(Z,m);
```

See Also
gtopo30s | readgeoraster | georasterinfo

Introduced before R2006a

gtopo30s

GTOPO30 data file names for latitude-longitude quadrangle

Syntax

```
tileNames = gtopo30s(latlim,lonlim)
tileNames = gtopo30s(lat,lon)
```

Description

`tileNames = gtopo30s(latlim,lonlim)` returns a cell array of the tile names covering the geographic region for GTOPO30 digital elevation maps (also referred to as "30-arc second" DEMs). `latlim` and `lonlim` specify the region as two-element vectors of latitude and longitude limits in units of degrees.

`tileNames = gtopo30s(lat,lon)` returns a cell array of the tile names covering the geographic region for GTOPO30 digital elevation maps. `lat` and `lon` specify the region as scalar latitude and longitude points.

See Also
`readgeoraster`

Introduced before R2006a

handlem

Handles of displayed map objects

Syntax

```
handlem
handlem('taglist')
handlem('prompt')
h = handlem(object)
h = handlem(tagstr)
h = handlem( ___ ,axesh)
h = handlem( ___ ,axesh,'searchmethod')
h = handlem(handles)
```

Description

`handlem` displays a dialog box for selecting objects that have their `Tag` property set.

`handlem('taglist')` displays a dialog box for selecting objects that have their `Tag` property set.

`handlem('prompt')` displays a dialog box for selecting objects based on the objects listed below.

`h = handlem(object)` returns the graphics objects in the current axes specified by the input, object. The options for the object are defined by the following list:

`'all'`	All children
`'clabel'`	Contour labels
`'contour'`	`hggroup` containing contours
`'fillcontour'`	`hggroup` containing filled contours
`'frame'`	Map frame
`'grid'`	Map grid lines
`'hggroup'`	All `hggroup` objects
`'hidden'`	Hidden objects
`'image'`	Untagged image objects
`'light'`	Untagged light objects
`'line'`	Untagged line objects
`'map'`	All objects on the map, excluding the frame and grid
`'meridian'`	Longitude grid lines
`'mlabel'`	Longitude labels
`'parallel'`	Latitude grid lines
`'plabel'`	Latitude labels
`'patch'`	Untagged patch objects
`'scaleruler'`	`scaleruler` objects

`'surface'`	Untagged surface objects
`''text''`	Untagged text objects
`'tissot'`	Tissot indicatrices
`'visible'`	Visible objects

`h = handlem(tagstr)` returns any graphics objects whose tags match the value of `tagstr`.

`h = handlem( ___ ,axesh)` searches within the specified axes.

`h = handlem( ___ ,axesh,'searchmethod')` controls the method used to match the `object` input. If omitted, `'exact'` is assumed. Search method `'strmatch'` searches for matches that start at the beginning of the tag. Search method `'findstr'` searches anywhere within the tag for the object .

`h = handlem(handles)` returns those graphics objects in the input vector of graphics objects that are still valid.

You can apply the prefix `'all'` when defining an object type (text, line, patch, light, surface, or image) to find all objects that meet the type criteria (for example, `'allimage'`). Without the `'all'` prefix, `handlem` returns only objects with an empty tag.

See Also
`findobj`

Introduced before R2006a

hidem

Hide specified graphic objects on map axes

Syntax

```
hidem
hidem(handle)
hidem(object)
```

Description

hidem brings up a dialog box for selecting the objects to hide (set their Visible property to 'off').

hidem(handle) hides the objects specified by a vector of handles.

hidem(object) hides those objects specified by the object, which can be any string scalar or character vector recognized by the handlem function.

See Also
clma | clmo | handlem | namem | showm | tagm

Introduced before R2006a

hista

Bin counts for geographic points using equal-area bins

Syntax

```
[latbin,lonbin,count] = hista(lat,lon)
[latbin,lonbin,count] = hista(lat,lon,binarea)
[latbin,lonbin,count] = hista(lat,lon,binarea,spheroid)
[latbin,lonbin,count] = hista( ___ ,angleunits)
```

Description

`[latbin,lonbin,count] = hista(lat,lon)` bins the geographic locations indicated by vectors `lat` and `lon`, using equal area binning on a sphere. The default bin area is 100 square kilometers. The `latbin` and `lonbin` outputs are column vectors indicating the centers of non-empty bins. `count` matches `latbin` and `lonbin` in size, with each element containing a positive integer equal to the number of occurrences in the corresponding bin.

Binning is performed on a mesh within a quadrangle whose latitude and longitude limits match the extrema of the input locations. The input and output latitudes and longitudes are in units of degrees.

`[latbin,lonbin,count] = hista(lat,lon,binarea)` uses the bin size specified by the input `binarea`, which must be in square kilometers

`[latbin,lonbin,count] = hista(lat,lon,binarea,spheroid)` bins the data on the reference spheroid defined by `spheroid`. `spheroid` is a `referenceEllipsoid` (`oblateSpheroid`) object, a `referenceSphere` object, or a vector of the form [semimajor_axis eccentricity]. The eccentricity/flattening of the spheroid is used in determining the latitude extent of the bins. The semimajor axis of the spheroid is used to determine the longitude extent of the bins, but if the length unit of the spheroid is unspecified, the mean radius of the earth in kilometers is used as the equatorial radius.

`[latbin,lonbin,count] = hista( ___ ,angleunits)` where `angleunits` defines the angle units of the inputs and outputs, specified as `'degrees'` or `'radians'`.

Examples

Bin Latitudes and Longitudes

Create some random latitudes.

```
rng(0,'twister')
lats = rand(4)

lats = 4×4

    0.8147    0.6324    0.9575    0.9572
    0.9058    0.0975    0.9649    0.4854
    0.1270    0.2785    0.1576    0.8003
```

```
    0.9134    0.5469    0.9706    0.1419
```

Create some random longitudes.

```
lons = rand(4)
```

```
lons = 4×4

    0.4218    0.6557    0.6787    0.6555
    0.9157    0.0357    0.7577    0.1712
    0.7922    0.8491    0.7431    0.7060
    0.9595    0.9340    0.3922    0.0318
```

Bin the data in 50-by-50 km cells (2500 sq km).

```
[lat,lon,num] = hista(lats,lons,2500);
```

```
[lat,lon,num]
```

```
ans = 9×3

    0.0932   -0.0208    2.0000
    0.5341    0.3235    1.0000
    0.9750    0.3235    2.0000
    0.0932    0.6678    2.0000
    0.5341    0.6678    1.0000
    0.9750    0.6678    4.0000
    0.0932    1.0122    1.0000
    0.5341    1.0122    1.0000
    0.9750    1.0122    2.0000
```

See Also
eqa2grn | grn2eqa | histr

Introduced before R2006a

histr

Histogram for geographic points with equirectangular bins

Syntax

```
[lat,lon,num,wnum] = histr(lats,lons)
[lat,lon,num,wnum] = histr(lats,lons,units)
[lat,lon,num,wnum] = histr(lats,lons,bindensty)
```

Description

`[lat,lon,num,wnum] = histr(lats,lons)` returns the center coordinates of equal-rectangular bins and the number of observations, `num`, falling in each based on the geographically distributed input data. Additionally, an area-weighted observation value, `wnum`, is returned. `wnum` is the bin's `num` divided by its normalized area. The largest bin has the same `num` and `wnum`; a smaller bin has a larger `wnum` than `num`.

`[lat,lon,num,wnum] = histr(lats,lons,units)` where `units` specifies the angle unit. The default value is `'degrees'`.

`[lat,lon,num,wnum] = histr(lats,lons,bindensty)` sets the number of bins per angular unit. For example, if `units` is `'degrees'`, a `bindensty` of 10 would be 10 bins per degree of latitude or longitude, resulting in 100 bins per *square* degree. The default is one cell per angular unit.

The `histr` function sorts geographic data into equirectangular bins for histogram purposes. Equirectangular in this context means that each bin has the same angular measurement on each side (e.g., 1º-by-1º). Consequently, the result is not an equal-area histogram. The `hista` function provides that capability. However, the results of `histr` can be weighted by their area bias to correct for this, in some sense.

Examples

Bin Latitudes and Longitudes with Equirectangular Bins

Create some random latitudes.

```
rng(0,'twister')
lats = rand(4)

lats = 4×4

    0.8147    0.6324    0.9575    0.9572
    0.9058    0.0975    0.9649    0.4854
    0.1270    0.2785    0.1576    0.8003
    0.9134    0.5469    0.9706    0.1419
```

Create some random longitudes.

```
lons = rand(4)
```

```
lons = 4×4

    0.4218    0.6557    0.6787    0.6555
    0.9157    0.0357    0.7577    0.1712
    0.7922    0.8491    0.7431    0.7060
    0.9595    0.9340    0.3922    0.0318
```

Bin the data in 0.5-by-0.5 degree cells (two bins per degree). The bins centered at 0.75°N are slightly smaller in area than the others. wnum reflects the relative count per normalized unit area.

```
[lat,lon,num,wnum] = histr(lats,lons,2);
```

```
[lat,lon,num,wnum]
```

```
ans = 4×4

    0.2500    0.2500    3.0000    3.0000
    0.7500    0.2500    2.0000    2.0002
    0.2500    0.7500    3.0000    3.0000
    0.7500    0.7500    8.0000    8.0006
```

See Also
filterm | hista

Introduced before R2006a

imbedm

Encode data points into regular data grid

Syntax

```
Z = imbedm(lat, lon, value, Z, R)
Z = imbedm(lat, lon, value, Z, R, units)
[Z, indxPointOutsideGrid] = imbedm(...)
```

Description

`Z = imbedm(lat, lon, value, Z, R)` resets certain entries of a regular data grid, Z. R can be a geographic raster reference object, a referencing vector, or a referencing matrix.

If R is a geographic raster reference object, its `RasterSize` property must be consistent with `size(Z)`.

If R is a referencing vector, it must be a 1-by-3 with elements:

`[cells/degree northern_latitude_limit western_longitude_limit]`

If R is a referencing matrix, it must be 3-by-2 and transform raster row and column indices to or from geographic coordinates according to:

`[lon lat] = [row col 1] * R`

If R is a referencing matrix, it must define a (non-rotational, non-skewed) relationship in which each column of the data grid falls along a meridian and each row falls along a parallel. Nearest-neighbor interpolation is used by default. NaN is returned for points outside the grid limits or for which `lat` or `lon` contain NaN. All angles are in units of degrees.

`Z = imbedm(lat, lon, value, Z, R, units)` specifies the units of the vectors `lat` and `lon`, where `units` is any valid angle units character vector (`'degrees'` by default).

`[Z, indxPointOutsideGrid] = imbedm(...)` returns the indices of `lat` and `lon` corresponding to points outside the grid in the variable `indxPointOutsideGrid`.

Examples

Create a simple grid map and embed new values in it:

```
Z = ones(3,6)

Z =
    1     1     1     1     1     1
    1     1     1     1     1     1
    1     1     1     1     1     1
refvec = [1/60 90 -180]

refvec =
    0.0167   90.0000 -180.0000
```

```
newgrid = imbedm([23 -23], [45 -45],[5 5],Z,refvec)

newgrid =
     1     1     1     1     1     1
     1     1     5     5     1     1
     1     1     1     1     1     1
```

See Also
geointerp | geographicToDiscrete

Introduced before R2006a

ind2rgb8

Convert indexed image to uint8 RGB image

Syntax

```
RGB = ind2rgb8(X,cmap)
```

Description

`RGB = ind2rgb8(X,cmap)` creates an RGB image of class `uint8`. X must be `uint8`, `uint16`, or `double`, and `cmap` must be a valid MATLAB colormap.

Examples

```
% Convert the 'concord_ortho_e.tif' image to RGB.
[X, cmap] = imread('concord_ortho_e.tif');
RGB = ind2rgb8(X, cmap);
R = worldfileread('concord_ortho_e.tfw','planar',size(X));
mapshow(RGB, R);
```

See Also
`ind2rgb`

Introduced before R2006a

ingeoquad

True for points inside or on lat-lon quadrangle

Syntax

```
tf = ingeoquad(lat, lon, latlim, lonlim)
```

Description

`tf = ingeoquad(lat, lon, latlim, lonlim)` returns an array `tf` that has the same size as `lat` and `lon`. `tf(k)` is true if and only if the point `lat(k)`, `lon(k)` falls within or on the edge of the geographic quadrangle defined by `latlim` and `lonlim`. `latlim` is a vector of the form [`southern-limit northern-limit`], and `lonlim` is a vector of the form [`western-limit eastern-limit`]. All angles are in units of degrees.

Examples

1 Load elevation data and a geographic cells reference object for the Korean peninsula. Display the data on a world map. Apply a colormap appropriate for elevation data using `demcmap`.

```
load korea5c
figure('Color','white')
worldmap([20 50],[90 150])
geoshow(korea5c,korea5cR,'DisplayType','texturemap');
demcmap(korea5c)
```

2 Outline the quadrangle containing the elevation data:

```
[outlineLat, outlineLon] = outlinegeoquad(korea5cR.LatitudeLimits, ...
   korea5cR.LongitudeLimits,90,5);
geoshow(outlineLat,outlineLon,'DisplayType','line', ...
   'Color','k')
```

3 Generate a track that crosses the elevation data:

```
[lat,lon] = track2(23,110,48,149,[1 0],'degrees',20);
geoshow(lat,lon,'DisplayType','line')
```

4 Identify and mark points on the track that fall within the quadrangle outlining the elevation data:

```
tf = ingeoquad(lat,lon,korea5cR.LatitudeLimits, ...
                       korea5cR.LongitudeLimits);
geoshow(lat(tf),lon(tf),'DisplayType','point')
```
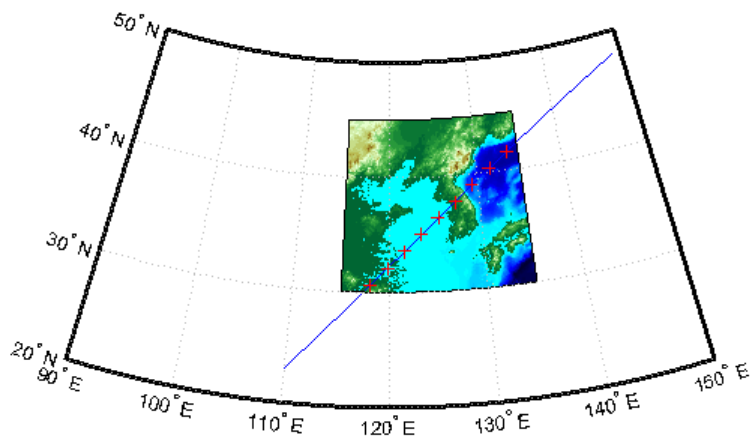
See Also

inpolygon | intersectgeoquad

Introduced in R2008a

intersectgeoquad

Intersection of two latitude-longitude quadrangles

Syntax

```
[latlim, lonlim] = intersectgeoquad(latlim1, lonlim1, latlim2, lonlim2)
```

Description

`[latlim, lonlim] = intersectgeoquad(latlim1, lonlim1, latlim2, lonlim2)` computes the intersection of the quadrangle defined by the latitude and longitude limits `latlim1` and `lonlim1`, with the quadrangle defined by the latitude and longitude limits `latlim2` and `lonlim2`. `latlim1` and `latlim2` are two-element vectors of the form `[southern-limit northern-limit]`. Likewise, `lonlim1` and `lonlim2` are two-element vectors of the form `[western-limit eastern-limit]`. All input and output angles are in units of degrees. The intersection results are given in the output arrays `latlim` and `lonlim`. Given an arbitrary pair of input quadrangles, there are three possible results:

1 *The quadrangles fail to intersect.* In this case, both `latlim` and `lonlim` are empty arrays.

2 *The intersection consists of a single quadrangle.* In this case, `latlim` (like `latlim1` and `latlim2`) is a two-element vector that has the form `[southern-limit northern-limit]`, where `southern-limit` and `northern-limit` represent scalar values. `lonlim` (like `lonlim1` and `lonlim2`), is a two-element vector that has the form `[western-limit eastern-limit]`, with a pair of scalar limits.

3 *The intersection consists of a pair of quadrangles.* This can happen when longitudes wrap around such that the eastern end of one quadrangle overlaps the western end of the other and vice versa. For example, if `lonlim1 = [-90 90]` and `lonlim2 = [45 -45]`, there are two intervals of overlap: `[-90 -45]` and `[45 90]`. These limits are returned in `lonlim` in separate rows, forming a 2-by-2 array. In our example (assuming that the latitude limits overlap), `lonlim` would equal `[-90 -45; 45 90]`. It still has the form `[western-limit eastern-limit]`, but `western-limit` and `eastern-limit` are 2-by-1 rather than scalar. The two output quadrangles have the same latitude limits, but these are replicated so that `latlim` is also 2-by-2.

To continue the example, if `latlim1 = [0 30]` and `latlim2 = [20 50]`, `latlim` equals `[20 30; 20 30]`. The form is still `[southern-limit northern-limit]`, but in this case `southern-limit` and `northern-limit` are 2-by-1.

Examples

Example 1

Nonintersecting quadrangles:

```
[latlim, lonlim] = intersectgeoquad( ...
                  [-40 -60], [-180 180], [40 60], [-180 180])
latlim =
     []

lonlim =
     []
```

Example 2

Intersection is a single quadrangle:

```
[latlim, lonlim] = intersectgeoquad( ...
                   [-40 60], [-120 45], [-60 40], [160 -75])

latlim =
   -40    40

lonlim =
  -120   -75
```

Example 3

Intersection is a pair of quadrangles:

```
[latlim, lonlim] = intersectgeoquad( ...
                   [-30 90],[-10 -170],[-90 30],[170 10])

latlim =
   -30    30
   -30    30

lonlim =
   -10    10
   170  -170
```

Example 4

Inputs and output fully encircle the planet:

```
[latlim, lonlim] = intersectgeoquad( ...
                   [-30 90],[-180 180],[-90 30],[0 360])

latlim =
   -30    30

lonlim =
  -180   180
```

Example 5

Find and map the intersection of the bounding boxes of adjoining U.S. states:

```
usamap({'Minnesota','Wisconsin'})
S = shaperead('usastatehi','UseGeoCoords',true,'Selector',...
    {@(name) any(strcmp(name,{'Minnesota','Wisconsin'})), 'Name'});
geoshow(S, 'FaceColor', 'y')
textm([S.LabelLat], [S.LabelLon], {S.Name},...
    'HorizontalAlignment', 'center')
latlimMN = S(1).BoundingBox(:,2)'

latlimMN =
   43.4995   49.3844

lonlimMN = S(1).BoundingBox(:,1)'
```

```
lonlimMN =
  -97.2385  -89.5612

latlimWI = S(2).BoundingBox(:,2)'

latlimWI =
   42.4918   47.0773

lonlimWI = S(2).BoundingBox(:,1)'

lonlimWI =
  -92.8892  -86.8059

[latlim lonlim] = ...
    intersectgeoquad(latlimMN, lonlimMN, latlimWI, lonlimWI)

latlim =
   43.4995   47.0773

lonlim =
  -92.8892  -89.5612

geoshow(latlim([1 2 2 1 1]), lonlim([1 1 2 2 1]), ...
    'DisplayType','polygon','FaceColor','m')
```
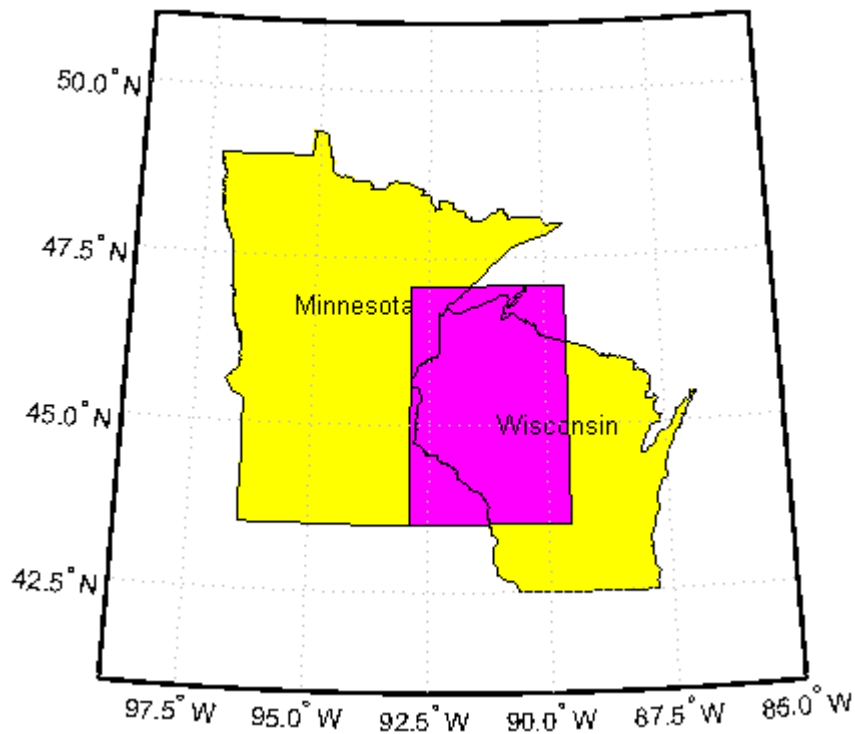

Tips

latlim1 and latlim2 should normally be given in order of increasing numerical value. No error will result if, for example, latlim1(2) < latlim1(1), but the outputs will both be empty arrays.

No such restriction applies to `lonlim1` and `lonlim2`. The first element is always interpreted as the western limit, even if it exceeds the second element (the eastern limit). Furthermore, `intersectgeoquad` correctly handles whatever longitude-wrapping convention may have been applied to `lonlim1` and `lonlim2`.

In terms of output, `intersectgeoquad` wraps `lonlim` such that all elements fall in the closed interval `[-180 180]`. This means that if (one of) the output quadrangle(s) crosses the 180° meridian, its western limit exceeds its eastern limit. The result would be such that

`lonlim(2) < lonlim(1)`

if the intersection comprises a single quadrangle or

`lonlim(k,2) < lonlim(k,1)`

where k equals 1 or 2 if the intersection comprises a pair of quadrangles.

If `abs(diff(lonlim1))` or `abs(diff(lonlim2))` equals `360`, its quadrangle is interpreted as a latitudinal zone that fully encircles the planet, bounded only by one parallel on the south and another parallel on the north. If two such quadrangles intersect, `lonlim` is set to `[-180 180]`.

If you want to display geographic quadrangles generated by this function or any other which are more than one or two degrees in extent, they may not follow curved meridians and parallels very well. The degree of departure depends on the extent of the quadrangle, the map projection, and the map scale. In such cases, you can interpolate intermediate vertices along quadrangle edges with the `outlinegeoquad` function.

See Also
`ingeoquad` | `outlinegeoquad`

Introduced in R2008a

inputm

Latitudes and longitudes of mouse-click locations

Syntax

```
[lat, lon] = inputm
[lat, lon] = inputm(n)
[lat, lon] = inputm(n,h)
[lat, lon, button] = inputm(n)
MAT = imputm(...)
```

Description

`[lat, lon] = inputm` returns the latitudes and longitudes in geographic coordinates of points selected by mouse clicks on a displayed grid. The point selection continues until the return key is pressed.

`[lat, lon] = inputm(n)` returns n points specified by mouse clicks.

`[lat, lon] = inputm(n,h)` prompts for points from the map axes specified by the handle h. If omitted, the current axes (`gca`) is assumed.

`[lat, lon, button] = inputm(n)` returns a third result, `button`, that contains a vector of integers specifying which mouse button was used (1,2,3 from left) or ASCII numbers if a key on the keyboard was used.

`MAT = imputm(...)` returns a single matrix, where `MAT = [lat lon]`.

Tips

`inputm` works much like the standard MATLAB `ginput`, except that the returned values are latitudes and longitudes extracted from the projection, rather than axes *x-y* coordinates. If you click outside of the projection bounds (beyond the map frame in the corners of a Robinson projection, for example), no coordinates are returned for that location.

`inputm` cannot be used with a 3-D display, including those created using `globe`.

See Also
`gcpmap` | `ginput`

Introduced before R2006a

interpm

Densify latitude-longitude sampling in lines or polygons

Syntax

```
[latout,lonout] = interpm(lat,lon,maxdiff)
[latout,lonout] = interpm(lat,lon,maxdiff,method)
[latout,lonout] = interpm(lat,lon,maxdiff,method,units)
```

Description

`[latout,lonout] = interpm(lat,lon,maxdiff)` fills in any gaps in latitude (`lat`) or longitude (`lon`) data vectors that are greater than a defined tolerance `maxdiff` apart in either dimension. The angle units of the three inputs need not be specified, but they must be identical. `latout` and `lonout` are the new latitude and longitude data vectors, in which any gaps larger than `maxdiff` in the original vectors have been filled with additional points. The default method of interpolation used by `interpm` is linear.

`[latout,lonout] = interpm(lat,lon,maxdiff,method)` interpolates between vector data coordinate points using a specified interpolation `method`. Valid interpolation methods are `'gc'` for great circle, `'rh'` for rhumb line, and `'lin'` for linear interpolation.

`[latout,lonout] = interpm(lat,lon,maxdiff,method,units)` specifies the units used, where `units` is any valid angle unit. The default is `'degrees'`.

Examples

Interpolate Vertices Between Known Data Points

Define two vectors containing the latitude and longitude values for a set of vertices. In `lat`, note that a gap of 2 degrees exists between the values 2 and 4. Similarly, in `lon`, a gap of 2 degrees exists between the values 1 and 3.

```
lat = [1 2 4 5];
lon = [1 3 4 5];
```

Call `interpm` to fill in any gaps greater than 1 degree in either vector. For example, `interpm` interpolates and inserts the value 2 into the `lon` vector to fill the gap between the values 1 and 3, and inserts the value 1.5 in the `lat` vector for this new vertex. Similarly, `interpm` inserts the value 3 into the `lat` vector to fill the gap between the values 2 and 4, and inserts the value 3.5 in the `lon` vector for this new vertex. Now, the separation of adjacent vertices is no greater than `maxdiff` in either `newlat` or `newlon`.

```
maxdiff = 1;
[newlat,newlon] = interpm(lat,lon,maxdiff)

newlat = 6×1

    1.0000
```

```
          1.5000
          2.0000
          3.0000
          4.0000
          5.0000


newlon = 6×1

          1.0000
          2.0000
          3.0000
          3.5000
          4.0000
          5.0000
```

See Also
intrplat | intrplon

Introduced before R2006a

intrinsicToGeographic

Package: `map.rasterref`

Transform intrinsic to geographic coordinates

Syntax

```
[lat,lon] = intrinsicToGeographic(R,xIntrinsic,yIntrinsic)
```

Description

`[lat,lon] = intrinsicToGeographic(R,xIntrinsic,yIntrinsic)` returns the geographic coordinates corresponding to intrinsic coordinates (`xIntrinsic`, `yIntrinsic`) in geographic raster R.

Examples

Find Geographic Coordinates from Intrinsic Coordinates

Find the geographic coordinates of cells within a raster by specifying a raster reference object and intrinsic coordinates.

First, load a geographic cells reference object for the Korean peninsula. To do this, load the `korea5cR` variable from the `korea5c` MAT-file. Then, specify the intrinsic coordinates of the cell in the lower-left corner. For this example, the lower-left corner is also the southwest corner because the `ColumnsStartFrom` property of the reference object has a value of `'south'` and the `RowsStartFrom` property has a value of `'west'`. Integer coordinates such as (1,1) indicate the center of a cell.

```
load korea5c korea5cR
xIntrinsic = 1;
yIntrinsic = 1;
```

Find the geographic coordinates.

```
[lat,lon] = intrinsicToGeographic(korea5cR,xIntrinsic,yIntrinsic)
```

```
lat = 30.0417

lon = 115.0417
```

You can reverse the operation by using the `geographicToIntrinsic` function.

```
[xIntrinsic,yIntrinsic] = geographicToIntrinsic(korea5cR,lat,lon)
```

```
xIntrinsic = 1.0000

yIntrinsic = 1.0000
```

Input Arguments

R — Geographic raster
`GeographicCellsReference` or `GeographicPostingsReference` object

Geographic raster, specified as a `GeographicCellsReference` or `GeographicPostingsReference` object.

xIntrinsic — *x*-coordinates in intrinsic coordinate system
numeric array

x-coordinates in intrinsic coordinate system, specified as a numeric array. `xIntrinsic` coordinates can be outside the bounds of the raster R.

Data Types: `single` | `double`

yIntrinsic — *y*-coordinates in intrinsic coordinate system
numeric array

y-coordinates in intrinsic coordinate system, specified as a numeric array. `yIntrinsic` is the same size as `xIntrinsic`. `yIntrinsic` coordinates can be outside the bounds of the raster R.

Data Types: `single` | `double`

Output Arguments

lat — Latitude coordinates
numeric array

Latitude coordinates, returned as a numeric array. `lat` is the same size as `xIntrinsic`.

When a point ($xIntrinsic(k)$, $yIntrinsic(k)$) is outside the bounds of raster R, $lat(k)$ and $lon(k)$ are extrapolated in the geographic coordinate system. However, for any point that extrapolates to a latitude beyond the poles (latitude outside the range [-90, 90] degrees), $lat(k)$ and $lon(k)$ are set to NaN.

Data Types: `double`

lon — Longitude coordinates
numeric array

Longitude coordinates, returned as a numeric array. `lon` is the same size as `xIntrinsic`.

When a point ($xIntrinsic(k)$, $yIntrinsic(k)$) is outside the bounds of raster R, $lat(k)$ and $lon(k)$ are extrapolated in the geographic coordinate system. However, for any point that extrapolates to a latitude beyond the poles (latitude outside the range [-90, 90] degrees), $lat(k)$ and $lon(k)$ are set to NaN.

Data Types: `double`

See Also
`geographicToIntrinsic` | `intrinsicXToLongitude` | `intrinsicYToLatitude` | `intrinsicToWorld`

Introduced in R2013b

intrinsicToWorld

Package: `map.rasterref`

Transform intrinsic to planar world coordinates

Syntax

`[xWorld,yWorld] = intrinsicToWorld(R,xIntrinsic,yIntrinsic)`

Description

`[xWorld,yWorld] = intrinsicToWorld(R,xIntrinsic,yIntrinsic)` returns the planar world coordinates corresponding to intrinsic coordinates (`xIntrinsic`, `yIntrinsic`) in map raster R. If a point is outside the bounds of R, then `intrinsicToWorld` extrapolates the `xWorld` and `yWorld` coordinates.

Examples

Find World Coordinates from Intrinsic Coordinates

Import a GeoTIFF image [1] on page 1-0   of Boston as an array and a `MapCellsReference` object. Find the world coordinates of the center of the cell in column `3500` and row `1200` of the array.

```
[Z,R] = readgeoraster('boston.tif');
[xWorld,yWorld] = intrinsicToWorld(R,3500,1200)
```

```
xWorld = 7.7580e+05
```

```
yWorld = 2.9566e+06
```

You can reverse the operation by using the `worldToIntrinsic` function.

```
[xIntrinsic,yIntrinsic] = worldToIntrinsic(R,xWorld,yWorld)
```

```
xIntrinsic = 3.5000e+03
```

```
yIntrinsic = 1.2000e+03
```

[1] The data used in this example includes material copyrighted by GeoEye, all rights reserved.

Input Arguments

R — Map raster
`MapCellsReference` object | `MapPostingsReference` object

Map raster, specified as a `MapCellsReference` or `MapPostingsReference` object.

xIntrinsic — *x*-coordinates in intrinsic coordinate system
numeric array

x-coordinates in intrinsic coordinate system, specified as a numeric array. `xIntrinsic` coordinates can be outside the bounds of the raster R.

Data Types: `single` | `double`

yIntrinsic — *y*-coordinates in intrinsic coordinate system
numeric array

y-coordinates in intrinsic coordinate system, specified as a numeric array. `yIntrinsic` is the same size as `xIntrinsic`. `yIntrinsic` coordinates can be outside the bounds of the raster R.

Data Types: `single` | `double`

Output Arguments

xWorld — *x*-coordinates in the world coordinate system
numeric array

x-coordinates in the world coordinate system, specified as a numeric array. `xWorld` is the same size as `xIntrinsic`. When `xIntrinsic(k)` is outside the bounds of R, `intrinsicToWorld` extrapolates the `xWorld` coordinate.

Data Types: `double`

yWorld — *y*-coordinates in the world coordinate system
numeric array

y-coordinates in the world coordinate system, specified as a numeric array. `yWorld` is the same size as `xIntrinsic`. When `yIntrinsic(k)` is outside the bounds of R, `intrinsicToWorld` extrapolates the `yWorld` coordinate.

Data Types: `double`

See Also
`worldToIntrinsic` | `intrinsicToGeographic` | `intrinsicXToLongitude` | `intrinsicYToLatitude`

Introduced in R2013b

intrinsicXToLongitude

Package: `map.rasterref`

Convert from intrinsic *x* to longitude coordinates

Syntax

```
lon = intrinsicXToLongitude(R,xIntrinsic)
```

Description

`lon = intrinsicXToLongitude(R,xIntrinsic)` returns the longitude of the meridian corresponding to the *x*-coordinate `xIntrinsic` in the intrinsic coordinate system, based on the relationship defined by geographic raster R.

Examples

Find Longitude from Intrinsic *x*-Coordinate

Import a GeoTIFF image [1] on page 1-0   of the Gulf of Mexico as an array and a `GeographicCellsReference` object. Find the longitude corresponding to the centers of cells in column 144 of the array.

```
[Z,R] = readgeoraster('katrina.tif');
lon = intrinsicXToLongitude(R,144)
```

```
lon = -95.7959
```

[1] The image used in this example is derived from imagery obtained from the NASA Goddard Scientific Visualization Studio (SVS) Image Server.

Input Arguments

R — Geographic raster
`GeographicCellsReference` or `GeographicPostingsReference` object

Geographic raster, specified as a `GeographicCellsReference` or `GeographicPostingsReference` object.

xIntrinsic — *x*-coordinates in intrinsic coordinate system
numeric array

x-coordinates in intrinsic coordinate system, specified as a numeric array. `xIntrinsic` coordinates can be outside the bounds of the raster R.

Data Types: `single` | `double`

Output Arguments

`lon` — Longitude coordinates
numeric array

Longitude coordinates, returned as a numeric array. `lon` is the same size as `xIntrinsic`.

When a point has intrinsic *x*-coordinate outside the bounds of raster R, `lon`(*k*) is extrapolated outside the longitude limits. Elements of `xIntrinsic` with value `NaN` map to `NaN` in `lon`. Longitude values are not wrapped.

Data Types: `double`

See Also

`intrinsicToGeographic` | `intrinsicYToLatitude` | `longitudeToIntrinsicX`

Introduced in R2013b

intrinsicYToLatitude

Package: `map.rasterref`

Convert from intrinsic *y* to latitude coordinates

Syntax

```
lat = intrinsicYToLatitude(R,yIntrinsic)
```

Description

`lat = intrinsicYToLatitude(R,yIntrinsic)` returns the latitude of the small circle corresponding to the *y*-coordinate `yIntrinsic` in the intrinsic coordinate system, based on the relationship defined by geographic raster R.

Examples

Find Latitude from Intrinsic *y*-Coordinate

Import elevation data [1] on page 1-0   for a region in Colorado as an array and a `GeographicPostingsReference` object. Find the latitude corresponding to the posting points in row 91 of the array.

```
[Z,R] = readgeoraster('n39_w106_3arc_v2.dt1');
lat = intrinsicYToLatitude(R,91)
```

```
lat = 39.9250
```

[1] The elevation data used in this example is from the U.S. Geological Survey.

Input Arguments

R — Geographic raster
`GeographicCellsReference` or `GeographicPostingsReference` object

Geographic raster, specified as a `GeographicCellsReference` or `GeographicPostingsReference` object.

yIntrinsic — *y*-coordinates in intrinsic coordinate system
numeric array

y-coordinates in intrinsic coordinate system, specified as a numeric array. `yIntrinsic` coordinates can be outside the bounds of the raster R.

Data Types: `single` | `double`

Output Arguments

`lat` — Latitude coordinates
numeric array

Latitude coordinates, returned as a numeric array. `lat` is the same size as `yIntrinsic`.

When a point has intrinsic *y*-coordinate outside the bounds of raster R, `lat`(*k*) is extrapolated outside the latitude limits. However, when a point extrapolates to a latitude beyond the poles (latitude outside the range [-90, 90] degrees), `lat`(*k*) is set to `NaN`. Elements of `yIntrinsic` with value `NaN` map to `NaN` in `lat`.

Data Types: `double`

See Also

`intrinsicXToLongitude` | `intrinsicToGeographic` | `latitudeToIntrinsicY`

Introduced in R2013b

intrplat

Interpolate latitude at given longitude

Syntax

```
newlat = intrplat(long,lat,newlong)
newlat = intrplat(long,lat,newlong,method)
newlat = intrplat(long,lat,newlong,method,units)
```

Description

`newlat = intrplat(long,lat,newlong)` returns an interpolated latitude, `newlat`, corresponding to a longitude `newlong`. `long` must be a monotonic vector of longitude values. The actual entries must be monotonic; that is, the longitude vector `[350 357 3 10]` is not allowed even though the geographic *direction* is unchanged (use `[350 357 363 370]` instead). `lat` is a vector of the latitude values paired with each entry in `long`.

`newlat = intrplat(long,lat,newlong,method)` specifies the `method` of interpolation employed, listed in the table below.

Method	Description
`'linear'`	Linear, or Cartesian, interpolation (default)
`'pchip'`	Piecewise cubic Hermite interpolation
`'rh'`	Returns interpolated points that lie on rhumb lines between input data
`'gc'`	Returns interpolated points that lie on great circles between input data

`newlat = intrplat(long,lat,newlong,method,units)` specifies the units used, where `units` is any valid angle units string scalar or character vector. The default is `'degrees'`.

The function `intrplat` is a geographic data analogy of the standard MATLAB function `interp1`.

Examples

Interpolate Coordinates at Specific Locations

Define the latitudes and longitudes.

```
lat = [57 68 60 65 56];
lon = [1 3 4 9 13];
```

Specify the longitude for which you want to compute a latitude.

```
newlon = 7.3;
```

Linear Interpolation

Generate a new latitude using the default method of linear interpolation.

```
newlatLinear = intrplat(lon,lat,newlon)
```

```
newlatLinear = 63.3000
```

Great Circle Interpolation

Generate a new latitude using great circle interpolation.

```
newlatGC = intrplat(lon,lat,newlon,'gc')
```

```
newlatGC = 63.5029
```

Rhumb Line Interpolation

Generate a new latitude using interpolation along a rhumb line.

```
newlatRhumb = intrplat(lon,lat,newlon,'rh')
```

```
newlatRhumb = 63.3937
```

Tips

There are separate functions for interpolating latitudes and longitudes, for although the cases are identical when using those methods supported by `interp1`, when latitudes and longitudes are treated like the spherical angles they are (using `'rh'` or `'gc'`), the results are different. Compare the example above to the example under `intrplon`, which reverses the values of latitude and longitude.

See Also

Functions
interpm | intrplon

Topics
"Geographic Interpolation of Vectors"

Introduced before R2006a

intrplon

Interpolate longitude at given latitude

Syntax

```
newlon = intrplon(lat,lon,newlat)
newlon = intrplon(lat,lon,newlat,method)
newlon = intrplon(lat,lon,newlat,method,units)
```

Description

`newlon = intrplon(lat,lon,newlat)` returns an interpolated longitude, `newlon`, corresponding to a latitude `newlat`. `lat` must be a monotonic vector of longitude values. `lon` is a vector of the longitude values paired with each entry in `lat`.

`newlon = intrplon(lat,lon,newlat,method)` specifies the `method` of interpolation employed, listed in the table below.

Method	Description
`'linear'`	Linear, or Cartesian, interpolation (default)
`'pchip'`	Piecewise cubic Hermite interpolation
`'rh'`	Returns interpolated points that lie on rhumb lines between input data
`'gc'`	Returns interpolated points that lie on great circles between input data

`newlon = intrplon(lat,lon,newlat,method,units)` specifies the units used, where `units` is any valid angle units string scalar or character vector. The default is `'degrees'`.

The function `intrplon` is a geographic data analogy of the MATLAB function `interp1`.

Examples

Compare the results of the various methods:

```
long = [25 45]; lat = [30 60];
newlon = intrplon(lat,long,45,'linear')

newlon =
    35

newlon = intrplon(lat,long,45,'rh')

newlon =
   33.6515

newlon = intrplon(lat,long,45,'gc')

newlon =
   32.0526
```

Tips

There are separate functions for interpolating latitudes and longitudes, for although the cases are identical when using those methods supported by `interp1`, when latitudes and longitudes are treated like the spherical angles they are (using `'rh'` or `'gc'`), the results are different. Compare the previous example to the example under `intrplat`, which reverses the values of latitude and longitude.

See Also

Functions
`interpm` | `intrplat`

Topics
"Geographic Interpolation of Vectors"

Introduced before R2006a

inverse

Package: map.geodesy

Convert authalic, conformal, isometric, or rectifying latitude to geodetic latitude

Syntax

```
phi = inverse(converter,lat)
phi = inverse(converter,lat,angleUnit)
```

Description

`phi = inverse(converter,lat)` returns the geodetic latitude coordinates corresponding to authalic, conformal, isometric, or rectifying latitude coordinates `lat`.

`phi = inverse(converter,lat,angleUnit)` specifies the units of output `phi`.

Examples

Convert Conformal Latitude to Geodetic Latitude

Specify conformal latitude coordinates and create a conformal latitude converter. Then, convert the coordinates.

```
chi = [-90 -67.3637 -44.8077 -22.3643 0 22.3643 44.8077 67.3637 90];
conv = map.geodesy.ConformalLatitudeConverter(wgs84Ellipsoid);
phi = inverse(conv,chi)

phi = 1×9

  -90.0000  -67.5000  -45.0000  -22.5000         0   22.5000   45.0000   67.5000   90.0000
```

Convert Isometric Latitude to Geodetic Latitude Using Radians

Specify isometric latitude coordinates and convert them to radians. Create an isometric latitude converter. Then, convert the coordinates by specifying the angle unit as `'radians'`.

```
psi = [-Inf -1.6087 -0.87663 -0.40064 0 0.40064 0.87663 1.6087 Inf];
conv = map.geodesy.IsometricLatitudeConverter(wgs84Ellipsoid);
phi = inverse(conv,psi,'radians')

phi = 1×9

   -1.5708   -1.1781   -0.7854   -0.3927         0    0.3927    0.7854    1.1781    1.5708
```

Input Arguments

converter — Latitude converter
AuthalicLatitudeConverter, ConformalLatitudeConverter, IsometricLatitudeConverter, or RectifyingLatitudeConverter object

Latitude converter, specified as an AuthalicLatitudeConverter, ConformalLatitudeConverter, IsometricLatitudeConverter, or RectifyingLatitudeConverter object.

lat — Latitude coordinates to convert
numeric scalar, vector, matrix, or N-D array

Latitude coordinates to convert, specified as a numeric scalar, vector, matrix, or N-D array.

The interpretation of lat depends on the latitude converter. If the conversion is:

- authalic, lat represents the variable β (beta).
- conformal, lat represents χ (chi).
- isometric, lat represents ψ (psi). lat is a dimensionless number and does not have an angle unit.
- rectifying, lat represents μ (mu).

For authalic, conformal, and rectifying conversions, the values of lat must be consistent with angleUnit.

angleUnit — Unit of latitude coordinates
'degrees' (default) | 'radians'

Units of latitude coordinates, specified as 'degrees' or 'radians'.

Output Arguments

phi — Geodetic latitude coordinates
numeric scalar, vector, matrix, or N-D array

Geodetic latitude coordinates, specified as a numeric scalar value, vector, matrix, or N-D array. phi is the same size as lat. If angleUnit is not supplied, phi is in degrees. Otherwise, values of phi are consistent with the units of angleUnit.

Data Types: single | double

See Also
forward

Introduced in R2013a

map.geodesy.isDegree

True if input matches `'degree'` and false if `'radian'`

Syntax

```
tf = map.geodesy.isDegree(angleUnit)
```

Description

`tf = map.geodesy.isDegree(angleUnit)` returns `true` if `angleUnit` is a partial match for `'degree'` (or `'degrees'`) and `false` if `angleUnit` is a partial match for `'radian'` (or `'radians'`). If `angleUnit` matches neither `'degrees'` or `'radians'`, `map.geodesy.isDegree` returns an error.

Examples

Test Inputs to a Function for Validity Before Processing

Create a function to calculate a cosine. In the function, use `map.geodesy.isDegree` to check the validity of the inputs.

```
function y = cosine(x, angleUnit)
% X can be in either degrees or radians

if map.geodesy.isDegree(angleUnit)
    y = cosd(x);
else
    y = cos(x);
end
```

Input Arguments

angleUnit — Angle unit value
`'degree'` | `'radian'`

Angle unit value, specified as `'degree'` or `'radian'`.

Data Types: `char` | `string`

Output Arguments

tf — True/false flag indicating if a match was found
logical scalar

True/false flag indicating if a match was found, returned as a logical scalar.

See Also

Introduced in R2013a

isempty

Determine if geographic or planar vector is empty

Syntax

```
tf = isempty(v)
```

Description

`tf = isempty(v)` determines whether the geographic or planar vector `v` is empty.

Examples

Check If a Geopoint Vector Is Empty

Create a default geopoint vector.

```
gp = geopoint()

gp =

 0x1 geopoint vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
 Feature properties:
     Latitude: []
    Longitude: []
```

Check if the geopoint vector is empty. A returned value of 1 (true) indicates the vector is empty.

```
isempty(gp)

ans = logical
    1
```

Create a second geopoint vector, specifying a geographic point. Confirm that this vector is not empty. A returned value of 0 (false) indicates the vector is not empty.

```
gp2 = geopoint(42.356,-71.101)

gp2 =
 1x1 geopoint vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
 Feature properties:
     Latitude: 42.3560
```

```
    Longitude: -71.1010
```

```
isempty(gp2)
```

```
ans = logical
   0
```

Input Arguments

v — Geographic or planar vector
geopoint, geoshape, mappoint, or mapshape object

Geographic or planar vector, specified as a geopoint, geoshape, mappoint, or mapshape object.

Output Arguments

tf — Flag indicating geographic or planar vector is empty
logical scalar

Flag indicating geographic or planar vector is empty, returned as a logical scalar. tf is True when v is empty.

Data Types: logical

Introduced in R2012a

isequal

Compare two `projcrs` or `geocrs` objects for equivalence

Syntax

```
tf = isequal(crs1,crs2)
```

Description

`tf = isequal(crs1,crs2)` returns logical `1` (`true`) if the coordinate reference systems (CRSs) `crs1` and `crs2` are equivalent. Otherwise, it returns logical `0` (`false`).

- Two `projcrs` objects are equivalent if they have the same geographic CRS, projection method, projection parameters, and length unit.
- Two `geocrs` objects are equivalent if they have the same datum name, reference spheroid, prime meridian, and angle unit.

The `isequal` function does not consider CRS names.

Examples

Compare Two Coordinate Reference Systems

Get information about two raster data sets using the `georasterinfo` function. Assign their coordinate reference systems to variables. In this case, the coordinate reference systems are `projcrs` objects.

```
info1 = georasterinfo('MtWashington-ft.grd');
info2 = georasterinfo('MountDartmouth-ft.grd');

crs1 = info1.CoordinateReferenceSystem;
crs2 = info2.CoordinateReferenceSystem;
```

Compare the coordinate reference systems.

```
tf = isequal(crs1,crs2)
```

```
tf = logical
   1
```

Compare Coordinate Reference Systems with Different WKT Strings

Two coordinate reference systems can be equivalent, even if the strings returned by the `wktstring` function are not identical.

Return information about a shapefile as a structure. Find the coordinate reference system by querying the `CoordinateReferenceSystem` field.

```
info = shapeinfo('landareas.shp');
g1 = info.CoordinateReferenceSystem

g1 =
  geocrs with properties:

              Name: "WGS 84"
             Datum: "World Geodetic System 1984"
          Spheroid: [1x1 referenceEllipsoid]
     PrimeMeridian: 0
         AngleUnit: "degree"
```

Create a second coordinate reference system by using the `geocrs` function.

```
g2 = geocrs(4326)

g2 =
  geocrs with properties:

              Name: "WGS 84"
             Datum: "World Geodetic System 1984"
          Spheroid: [1x1 referenceEllipsoid]
     PrimeMeridian: 0
         AngleUnit: "degree"
```

The coordinate reference systems have the same datum name, reference spheroid, prime meridian, and angle unit. Therefore, the coordinate reference systems are equivalent.

```
isequal(g1,g2)

ans = logical
   1
```

Note that the WKT strings are not identical. The string for `g2` contains additional information about the area of use ("USAGE") and authority code ("ID").

```
wktstring(g1,'Format','Formatted')

ans =
    "GEOGCRS["WGS 84",
        DATUM["World Geodetic System 1984",
            ELLIPSOID["WGS 84",6378137,298.257223563,
                LENGTHUNIT["metre",1]],
            ID["EPSG",6326]],
        PRIMEM["Greenwich",0,
            ANGLEUNIT["Degree",0.0174532925199433]],
        CS[ellipsoidal,2],
            AXIS["longitude",east,
                ORDER[1],
                ANGLEUNIT["Degree",0.0174532925199433]],
            AXIS["latitude",north,
                ORDER[2],
                ANGLEUNIT["Degree",0.0174532925199433],
                ID["EPSG",7030]]]"
```

```
wktstring(g2,'Format','Formatted')

ans =
    "GEOGCRS["WGS 84",
        DATUM["World Geodetic System 1984",
            ELLIPSOID["WGS 84",6378137,298.257223563,
                LENGTHUNIT["metre",1]]],
        PRIMEM["Greenwich",0,
            ANGLEUNIT["degree",0.0174532925199433]],
        CS[ellipsoidal,2],
            AXIS["geodetic latitude (Lat)",north,
                ORDER[1],
                ANGLEUNIT["degree",0.0174532925199433]],
            AXIS["geodetic longitude (Lon)",east,
                ORDER[2],
                ANGLEUNIT["degree",0.0174532925199433]],
        USAGE[
            SCOPE["Horizontal component of 3D system."],
            AREA["World."],
            BBOX[-90,-180,90,180]],
        ID["EPSG",4326]]"
```

Input Arguments

crs1 — First coordinate reference system
projcrs object | geocrs object

First coordinate reference system, specified as a `projcrs` or `geocrs` object.

crs2 — Second coordinate reference system
projcrs object | geocrs object

Second coordinate reference system, specified as a `projcrs` or `geocrs` object.

Tips

- If you expect two coordinate reference systems to be equivalent, but the `isequal` function returns `0` (`false`), you can inspect the WKT strings for differences using the `wktstring` function.

See Also

Objects
projcrs | geocrs

Introduced in R2021a

isfield

Determine if dynamic property exists in geographic or planar vector

Syntax

```
tf = isfield(v,name)
tf = isfield(v,names)
```

Description

`tf = isfield(v,name)` determines whether the value specified by `name` is a dynamic property in geographic or planar vector `v`.

`tf = isfield(v,names)` determines whether each value specified by `names` is a dynamic property in `v`.

Examples

Check If a Dynamic Property Exists in a Mappoint Vector

Create a mappoint vector.

```
mp = mappoint(-33.961, 18.484, 'Name', 'Cape Town')

mp =
 1x1 mappoint vector with properties:

 Collection properties:
    Geometry: 'point'
    Metadata: [1x1 struct]
 Feature properties:
           X: -33.9610
           Y: 18.4840
        Name: 'Cape Town'
```

Check if individual properties are dynamic properties in the mappoint vector.

```
isfield(mp, 'X')

ans = logical
   0
```

This result is 0 (false) because property `X` in the mappoint vector is not a dynamic property.

```
isfield(mp, 'Name')

ans = logical
   1
```

This result is 1 (true) because the property `Name` is a dynamic property that exists in the mappoint vector.

```
isfield(mp,'Latitude')

ans = logical
   0
```

This result is 0 (false) because the dynamic property `Latitude` does not exist in the mappoint vector.

Check If Multiple Dynamic Properties Exist in a Geoshape Vector

Create a geoshape vector.

```
gs = geoshape(-33.961, 18.484, 'Name', 'Cape Town')

gs =
 1x1 geoshape vector with properties:

 Collection properties:
     Geometry: 'line'
     Metadata: [1x1 struct]
 Vertex properties:
     Latitude: -33.9610
    Longitude: 18.4840
 Feature properties:
         Name: 'Cape Town'
```

Check if a group of properties are dynamic properties in the geoshape vector.

```
tf = isfield(gs, {'Latitude','longitude','Name'})

tf = 1x3 logical array

   0   0   1
```

The first element of `tf` is 0 (false) because the property `Latitude` exists in the geoshape vector but is not a dynamic property. The second element of `tf` is 0 (false) because the property `longitude` does not exist in the geoshape vector (property names are case-sensitive.) The last element of `tf` is 1 (true), indicating that `Name` is a dynamic property in the geoshape vector.

Input Arguments

v — Geographic or planar vector
geopoint, geoshape, mappoint, or mapshape object

Geographic or planar vector, specified as a `geopoint`, `geoshape`, `mappoint`, or `mapshape` object.

name — Name of single property
character vector

Name of a single property, specified as a character vector.

names — Name of multiple properties
cell array of character vectors

Name of multiple properties, specified as a cell array of character vectors.

Output Arguments

tf — Flag indicating the dynamic property exists in the geographic or planar vector
logical scalar or vector

Flag indicating the dynamic property exists in the geographic or planar vector, returned as a logical scalar or vector. Each element of `tf` is `True` when the corresponding value in `name` or `names` is a dynamic property that exists in `v`.

Data Types: `logical`

See Also
`fieldnames` | `isprop`

Introduced in R2012a

isgeotable

Determine if input is geospatial table

Syntax

```
tf = isgeotable(A)
```

Description

`tf = isgeotable(A)` determines if the input `A` is a geospatial table. The function returns a logical `1` (`true`) if `A` is a geospatial table and a logical `0` (`false`) otherwise.

A geospatial table is a `table` or `timetable` object in which:

- The first variable of the table is called `Shape`.
- The `Shape` variable contains `geopointshape`, `geolineshape`, `geopolyshape`, `mappointshape`, `maplineshape`, or `mappolyshape` objects.

For more information about geospatial tables, see "Create Geospatial Tables".

Examples

Determine if Input is Geospatial Table

Import a shapefile as a geospatial table. Verify that it is a geospatial table.

```
GT = readgeotable("concord_roads.shp");
tf = isgeotable(GT)

tf = logical
   1
```

Convert the geospatial table `GT` to a table `T`. The `isgeotable` function returns `false` because `T` is not a geospatial table.

```
T = geotable2table(GT);
tf = isgeotable(T)

tf = logical
   0
```

Input Arguments

A — Input array
array

Input array. The array can have any data type.

See Also

Functions
readgeotable | istable

Introduced in R2021b

ismap

True for axes with map projection

Syntax

```
mflag = ismap
mflag = ismap(hndl)
[mflag,msg] = ismap(hndl)
```

Description

`mflag = ismap` returns a 1 if the current axes is a map axes, and 0 otherwise.

`mflag = ismap(hndl)` specifies the handle of the axes to be tested.

`[mflag,msg] = ismap(hndl)` returns the character vector `msg` if the axes is not a map axes, specifying why not.

The `ismap` function tests an axes object to determine whether it is a map axes.

See Also

gcm | ismapped

Introduced before R2006a

ismapped

True, if object is projected on map axes

Syntax

```
mflag = ismapped
mflag = ismapped(hndl)
[mflag,msg] = ismapped(hndl)
```

Description

`mflag = ismapped` returns a 1 if the current object is projected on a map axes, and 0 otherwise.

`mflag = ismapped(hndl)` specifies the handle of the object to be tested.

`[mflag,msg] = ismapped(hndl)` returns the character vector `msg` if the axes is not projected on a map axes, specifying why not.

The `ismapped` function tests an object to determine whether it is projected on map axes.

See Also

gcm | ismap

Introduced before R2006a

map.geodesy.IsometricLatitudeConverter

Convert between geodetic and isometric latitudes

Description

An `IsometricLatitudeConverter` object provides conversion methods between geodetic and isometric latitudes for an ellipsoid with a given eccentricity.

The isometric latitude is a nonlinear function of the geodetic latitude that is directly proportional to the spacing of parallels, relative to the Equator, in an ellipsoidal Mercator projection. It is a dimensionless quantity and, unlike other types of auxiliary latitude, the isometric latitude is not angle-valued. It equals `Inf` at the north pole and `-Inf` at the south pole.

Creation

Syntax

```
converter = map.geodesy.IsometricLatitudeConverter
converter = map.geodesy.IsometricLatitudeConverter(spheroid)
```

Description

`converter = map.geodesy.IsometricLatitudeConverter` returns an `IsometricLatitudeConverter` object for a sphere and sets the `Eccentricity` property to `0`.

`converter = map.geodesy.IsometricLatitudeConverter(spheroid)` returns an isometric latitude converter object and sets the `Eccentricity` property to match the specified spheroid object.

Input Arguments

spheroid — Reference spheroid
referenceEllipsoid object | oblateSpheroid object | referenceSphere object

Reference spheroid, specified as a `referenceEllipsoid` object, `oblateSpheroid` object, or `referenceSphere` object. The term reference spheroid is used synonymously with reference ellipsoid. To create a reference spheroid, use the creation function for the object. To specify the reference ellipsoid for WGS84, use the `wgs84Ellipsoid` function.

For more information about reference spheroids, see "Reference Spheroids".

Example: `spheroid = referenceEllipsoid('GRS 80');`

Properties

Eccentricity — Ellipsoid eccentricity
0 | numeric scalar

Ellipsoid eccentricity, specified as a numeric scalar. `Eccentricity` is in the interval [0, 0.5]. Eccentricities larger than 0.5 are possible in theory, but do not occur in practice and are not supported.

Data Types: `double`

Object Functions

forward Convert geodetic latitude to authalic, conformal, isometric, or rectifying latitude
inverse Convert authalic, conformal, isometric, or rectifying latitude to geodetic latitude

Examples

Create an Isometric Latitude Converter Object and Set Property

```
grs80 = referenceEllipsoid('GRS 80');

conv1 = map.geodesy.IsometricLatitudeConverter;
conv1.Eccentricity = grs80.Eccentricity

conv1 =

  IsometricLatitudeConverter with properties:

    Eccentricity: 0.0818
```

Create an Isometric Latitude Converter Object Specifying Spheroid

```
grs80 = referenceEllipsoid('GRS 80');

conv2 = map.geodesy.IsometricLatitudeConverter(grs80)

conv2 =

  IsometricLatitudeConverter with properties:

    Eccentricity: 0.0818
```

See Also

Functions
geocentricLatitude | parametricLatitude

Objects
AuthalicLatitudeConverter | ConformalLatitudeConverter | RectifyingLatitudeConverter

Introduced in R2013a

ispolycw

True if polygon vertices are in clockwise order

Syntax

```
tf = ispolycw(x,y)
```

Description

`tf = ispolycw(x,y)` returns true if the polygonal contour vertices represented by `x` and `y` are ordered in the clockwise direction. `x` and `y` are Cartesian vectors with the same number of elements.

Alternatively, `x` and `y` can contain multiple contours, either in `NaN`-separated vector form or in cell array form. In that case, `ispolycw` returns a logical array containing one true or false value per contour.

`ispolycw` always returns true for polygonal contours containing two or fewer vertices.

Vertex ordering is not well defined for self-intersecting polygonal contours. For such contours, `ispolycw` returns a result based on the order or vertices immediately before and after the left-most of the lowest vertices. In other words, of the vertices with the lowest `y` value, find the vertex with the lowest `x` value. For a few special cases of self-intersecting contours, the vertex ordering cannot be determined using only the left-most of the lowest vertices; for these cases, `ispolycw` uses a signed area test to determine the ordering.

Class Support

`x` and `y` may be any numeric class.

Examples

Orientation of a square:

```
x = [0 1 1 0 0];
y = [0 0 1 1 0];
ispolycw(x, y)                    % Returns 0
ispolycw(fliplr(x), fliplr(y))    % Returns 1
```

Tips

You can use `ispolycw` for geographic coordinates if the polygon does not cross the Antimeridian or contain a pole. A polygon contains a pole if the longitude data spans 360 degrees. To use `ispolycw` with geographic coordinates, specify the longitude vector as `x` and the latitude vector as `y`.

See Also
`poly2cw` | `poly2ccw` | `polyshape`

Topics
"Create and Display Polygons"

Introduced before R2006a

isprop

Determine if property exists in geographic or planar vector

Syntax

```
tf = isprop(v,name)
tf = isprop(v,names)
```

Description

`tf = isprop(v,name)` determines whether the value specified by `name` is a property in geographic or planar vector `v`.

`tf = isprop(v,names)` determines whether each value specified by `names` is a property in `v`.

Examples

Check If a Single Property Exists in Geoshape Vector

Create a geoshape vector.

```
s = geoshape(-33.961, 18.484, 'Name', 'Cape Town');
```

Check if `Latitude` and `Name` are properties in the geoshape vector.

```
isprop(s, 'Latitude')
```

```
ans = logical
   1
```

```
isprop(s, 'Name')
```

```
ans = logical
   1
```

Both `Latitude` and `Name` are properties in the geoshape vector.

Check If Multiple Properties Exist in a Mappoint Vector

Create a mappoint vector.

```
mp = mappoint(-33.961, 18.484, 'Name', 'Cape Town')
```

```
mp =
 1x1 mappoint vector with properties:

 Collection properties:
```

```
    Geometry: 'point'
    Metadata: [1x1 struct]
 Feature properties:
           X: -33.9610
           Y: 18.4840
        Name: 'Cape Town'
```

Check if a group of properties exist in the mappoint vector.

```
tf = isprop(mp, {'X','x','Latitude','Name'})
```

tf = *1x4 logical array*

```
   1   0   0   1
```

The first and last elements of `tf` are 1 (true) because the properties X and `Name` exist in the mappoint vector. The second and third elements of `tf` are 0 (false) because the properties x and `Latitude` do not exist in the mappoint vector. Property names are case-sensitive.

Input Arguments

v — Geographic or planar vector
geopoint, geoshape, mappoint, or mapshape object

Geographic or planar vector, specified as a `geopoint`, `geoshape`, `mappoint`, or `mapshape` object.

name — Name of single property
character vector

Name of a single property, specified as a character vector.

names — Name of multiple properties
cell array of character vectors

Name of multiple properties, specified as a cell array of character vectors.

Output Arguments

tf — Flag indicating the property exists in the geographic or planar vector
logical scalar or vector

Flag indicating the property exists in the geographic or planar vector, returned as a logical scalar or vector. Each element of `tf` is `True` when the corresponding value in `name` or `names` is a property that exists in `v`.

Data Types: `logical`

See Also
isfield | properties

Introduced in R2012a

isShapeMultipart

True if polygon or line has multiple parts

Syntax

```
tf = isShapeMultipart(xdata, ydata)
```

Description

`tf = isShapeMultipart(xdata, ydata)` returns 1 (`true`) if the polygon or line shape specified by `xdata` and `ydata` consists of multiple `NaN`-separated parts (i.e. has inner or multiple polygon rings or multiple line segments). The coordinate arrays `xdata` and `ydata` must match in size and have identical `NaN` locations.

Examples

Check If Datasets Are Multipart

Create a simple data set and check if it's multipart. If a data set contains NaN separators, `isShapeMultipart` returns 1 , otherwise 0.

```
sample_xdata = [0 0 1];
sample_ydata = [0 1 0];
isShapeMultipart(sample_xdata,sample_ydata)
```

```
ans = logical
   0
```

Create simple multipart data sets.

```
multi_xdata = [0 0 1 NaN 2 2 3 3];
multi_ydata = [0 1 0 NaN 2 3 3 2];
isShapeMultipart(multi_xdata,multi_ydata)
```

```
ans = logical
   1
```

Check a real data set.

```
load coastlines
isShapeMultipart(coastlat,coastlon)
```

```
ans = logical
   1
```

Check the data in a shapefile.

```
S = shaperead('concord_hydro_area.shp');
isShapeMultipart(S(1).X, S(1).Y)
```

```
ans = logical
   0
```

Check another dataset in the shapefile.

```
isShapeMultipart(S(14).X, S(14).Y)
```

```
ans = logical
   1
```

See Also
`polysplit`

Topics
"Create and Display Polygons"

Introduced in R2006a

km2deg

Convert spherical distance from kilometers to degrees

Syntax

```
deg = km2deg(km)
deg = km2deg(km,radius)
deg = km2deg(km,sphere)
```

Description

`deg = km2deg(km)` converts distances from kilometers to degrees, as measured along a great circle on a sphere with a radius of 6371 km, the mean radius of the Earth.

`deg = km2deg(km,radius)` converts distances from kilometers to degrees, as measured along a great circle on a sphere having the specified `radius`.

`deg = km2deg(km,sphere)` converts distances from kilometers to degrees, as measured along a great circle on a sphere approximating an object in the Solar System.

Examples

Convert Kilometers to Degrees

Two cities are 340 km apart. How many degrees of arc is that?

```
deg = km2deg(340)
```

```
deg =
    3.0577
```

How many degrees would it be if the cities were on Mars?

```
deg = km2deg(340,'mars')
```

```
deg =
    5.7465
```

Input Arguments

km — Distance in kilometers
numeric array

Distance in kilometers, specified as a numeric array.

Data Types: `single` | `double`

radius — Radius
6371 (default) | numeric scalar

Radius of sphere in units of kilometers, specified as a numeric scalar.

sphere — Sphere
'sun' | 'moon' | 'mercury' | 'venus' | 'earth' | …

Sphere approximating an object in the Solar System, specified as one of the following values: 'sun', 'moon', 'mercury', 'venus', 'earth', 'mars', 'jupiter', 'saturn', 'uranus', 'neptune', or 'pluto'. The value of sphere is case-insensitive.

Output Arguments

deg — Distance in degrees
numeric array

Distance in degrees, returned as a numeric array.

Data Types: single | double

See Also
deg2rad | rad2deg | deg2km | km2rad | nm2deg | sm2deg

Introduced in R2007a

km2nm

Convert kilometers to nautical miles

Syntax

```
nm = km2nm(km)
```

Description

`nm = km2nm(km)` converts distances from kilometers to nautical miles.

See Also

deg2km | km2deg | km2rad | rad2km | deg2nm | nm2deg | nm2rad | rad2nm | deg2sm | sm2deg | deg2sm | sm2rad | rad2sm

Introduced in R2007a

km2rad

Convert spherical distance from kilometers to radians

Syntax

```
rad = km2rad(km)
rad = km2rad(km,radius)
rad = km2rad(km,sphere)
```

Description

`rad = km2rad(km)` converts distances from kilometers to radians, as measured along a great circle on a sphere with a radius of 6371 km, the mean radius of the Earth.

`rad = km2rad(km,radius)` converts distances from kilometers to radians, as measured along a great circle on a sphere having the specified `radius`.

`rad = km2rad(km,sphere)` converts distances from kilometers to radians, as measured along a great circle on a sphere approximating an object in the Solar System.

Examples

Convert Kilometers to Radians

How many radians does 1,000 km span on the Earth and on the Moon?

```
rad = km2rad(1000)

rad =
    0.1570

rad = km2rad(1000,'moon')

rad =
    0.5754
```

Input Arguments

km — Distance in kilometers
numeric array

Distance in kilometers, specified as a numeric array.

Data Types: `single` | `double`

radius — Radius
6371 (default) | numeric scalar

Radius of sphere in units of kilometers, specified as a numeric scalar.

sphere — Sphere
`'sun'` | `'moon'` | `'mercury'` | `'venus'` | `'earth'` | …

Sphere approximating an object in the Solar System, specified as one of the following values: `'sun'`, `'moon'`, `'mercury'`, `'venus'`, `'earth'`, `'mars'`, `'jupiter'`, `'saturn'`, `'uranus'`, `'neptune'`, or `'pluto'`. The value of `sphere` is case-insensitive.

Output Arguments

rad — Distance in radians
numeric array

Distance in radians, returned as a numeric array.

Data Types: `single` | `double`

See Also
deg2rad | rad2deg | km2deg | nm2rad | sm2rad | rad2km

Introduced in R2007a

km2sm

Convert kilometers to statute miles

Syntax

```
sm = km2sm(km)
```

Description

`sm = km2sm(km)` converts distances from kilometers to statute miles.

Examples

How many statute miles is a 10k run?

```
sm = km2sm(10)

sm =
    6.2137
```

See Also

deg2km | km2deg | km2rad | rad2km | deg2nm | nm2deg | nm2rad | rad2nm | deg2sm | sm2deg | deg2sm | sm2rad | rad2sm

Introduced in R2007a

kmlwrite

Write geographic data to KML file

Note The following syntaxes, while still supported, are not recommended. Use `kmlwritepoint` instead.

- `kmlwrite(filename,lat,lon)`
- `kmlwrite(filename,lat,lon,alt)`

Syntax

```
kmlwrite(filename,S)
kmlwrite(filename,address)
kmlwrite( ___ ,Name,Value)
```

Description

`kmlwrite(filename,S)` writes the geographic point, line, or polygon data stored in `S` to the file specified by `filename` in Keyhole Markup Language (KML) format. The `kmlwrite` function creates a KML Placemark in the file and populates the tags in the placemark with the data in `S`.

`kmlwrite(filename,address)` writes `address` to the file specified by `filename` in KML format. `address` is a string scalar or character vector containing freeform address data, that can include street, city, state, country, and/or postal code. To specify multiple addresses, use a cell array of character vectors or string scalars. `kmlwrite` creates a KML Placemark in the file, setting the value of the address tag. An address is an alternative way to specify a point, instead of using latitude and longitude.

`kmlwrite( ___ ,Name,Value)` specifies name-value pairs that set additional KML feature properties. Parameter names can be abbreviated and are case-insensitive.

Examples

Write Point Data to KML File Using Geographic Point Vector

Import point data as a geographic point (`geopoint`) vector.

```
placenames = gpxread("boston_placenames.gpx");
```

Specify the colors to use with the point data.

```
colors = jet(length(placenames));
```

Write the point data to a file called `boston_placenames.kml`. Specify the point names and icon colors using name-value arguments.

```
kmlwrite("boston_placenames.kml",placenames,"Name",placenames.Name,"Color",colors)
```

Write Line Data to KML File Using Geographic Shape Vector

Read line features into a geographic shape (`geoshape`) vector.

```
tracks = gpxread("sample_tracks.gpx","Index",1:2);
```

Write the line data to a KML file. Specify the colors, widths, names, and descriptions of the lines using name-value arguments.

```
colors = ["red" "green"];
description = tracks.Metadata.Name;
name = ["track1" "track2"];

kmlwrite("sample_tracks.kml",tracks,"Color",colors,"LineWidth",2, ...
         "Description",description,"Name",name)
```

Write Geographic Data to KML File Using Geographic Structure Array

Read the names and locations of major European cities from a shapefile. The `shaperead` function returns a geographic structure array (`geostruct`).

```
latlim = [30;75];
lonlim = [-25;45];
cities = shaperead("worldcities.shp","UseGeoCoords",true, ...
                   "BoundingBox",[lonlim, latlim]);
```

Convert the structure to a geographic point (`geopoint`) vector.

```
cities = geopoint(cities);
```

Write the data to a KML file. Include city names in the placemarks by using the `Name` name-value argument. Remove the default `Description` table.

```
kmlwrite("european_cities.kml",cities,"Name",cities.Name,"Description",{});
```

Write Unstructured Address to KML File

Create a cell array of unstructured addresses. For this example, use the names of several Australian cities.

```
address = {'Perth, Australia', ...
           'Melbourne, Australia', ...
           'Sydney, Australia'};
```

Write the unstructured address data to a KML file called `australian_cities.kml`. Include the names of the cities in the placemarks using the `Name` name-value argument.

```
kmlwrite("australian_cities.kml",address,"Name",address);
```

Write Polygon Data to KML File

Import a shapefile containing polygon data into a geospatial table. The polygons represent state boundaries.

```
GT = readgeotable("usastatelo.shp");
```

Create a colormap for the data using random, muted colors.

```
colors = polcmap(height(GT));
```

Write the polygon data to a KML file called `usastatelo.kml`. Specify the labels and the colors of the polygon faces and edges using name-value arguments.

```
kmlwrite("usastatelo.kml",GT,"Name",GT.Name, ...
    "FaceColor",colors,"EdgeColor","k")
```

Input Arguments

`filename` — Name of output file
character vector | string scalar

Name of output file, specified as a string scalar or character vector. `kmlwrite` writes the file in the current folder, unless you specify a full or relative path name. If the file name includes an extension, it must be `.kml`.

Data Types: `char` | `string`

S — Geographic features to write to file
geospatial table | `geopoint` vector | `geoshape` vector | geographic structure array

Geographic features to write to file, specified as one of the following:

- A geospatial table containing `geopointshape`, `geolineshape`, or `geopolyshape` objects. The `kmlwrite` function does not support geospatial tables containing more than one type of shape object. For more information about geospatial tables, see "Create Geospatial Tables".

- A `geopoint` vector.

- A `geoshape` vector. The `Geometry` field identifies the type of the data as `'point'`, `'line'`, or `'polygon'`.

- A geographic structure array (`geostruct`) with `Lat` and `Lon` fields. If you specify a geographic structure array with `X` and `Y` fields, `kmlwrite` throws an error.

The attribute fields of S appear as a table in the Description tag of the Placemark for each element of S. The attribute fields appear in the table in the same order as they occur in S.

If S contains valid altitude data, `kmlwrite` writes the field values to the file as KML altitudes and sets the altitude interpretation to `'relativeToSeaLevel'`. If S does not contain altitude data, `kmlwrite` sets the altitude field in the file to 0 and sets the altitude interpretation to `'clampToGround'`. The altitude data can be in a field named either `Elevation`, `Altitude`, or `Height`. If S contains fields with more than one of these names, `kmlwrite` issues a warning and ignores the altitude fields.

address — Location of KML placemark
character vector | string scalar | cell array of character vectors

Location of KML placemark, specified as a string scalar, character vector or cell array of character vectors containing freeform address data, such as street, city, state, and postal code. If `address` is a cell array, each cell represents a unique location.

Data Types: `char` | `string` | `cell`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Name','Point Reyes'`

Name — Label of object displayed in viewer
`'Address N'`, `'Point N'`, `'Line N'`, or `'Polygon N'`, where N is the index of the feature. (default) | string scalar | character vector | cell array of character vectors

Label for an object displayed in viewer, specified as a string scalar, character vector, or cell array of character vectors. If you specify a string scalar or character vector, `kmlwrite` applies the name to all objects. If a string vector or cell array of character vectors, you must specify a name for each feature. That is, the cell array must be the same length as `S` or `address`. The following describes the default behavior for various features.

Feature	Default Name
Address	`'Address N'` where N is the index of the feature.
Point	`'Point N'` where N is the index of the feature.
Multipoint	`'Multipoint N'` where N is the index of the feature. `kmlwrite` places the points in the named folder and each point is named `'Point M'` where M is the index of the point.
Line	`'Line N'` where N is the index of the feature. If the line data contains `NaN` values, `kmlwrite` places the line segments in a folder named `'Segment M'`, where M is the line segment number.
Polygon	`'Polygon N'` where N is the index of the feature. If the polygon vertex list contains multiple outer rings, `kmlwrite` places each ring in a folder labeled `'Part M'`, where M is the number for that feature.

Data Types: `char` | `string` | `cell`

Description — Content to be displayed in the placemark description balloon
string scalar | character vector | cell array of character vectors | attribute specification

Content to be displayed in the placemark description balloon, specified as a string scalar or character vector, cell array of character vectors, or an attribute specification. `kmlwrite` uses this data to set the values of the feature description tags. The description appears in the description balloon associated with the feature in Google Earth™.

- If you specify a string scalar or character vector, `kmlwrite` applies the description to all objects.
- If you specify a string vector or cell array of character vectors, there must be one label for each feature; that is, it must be the same length as `S` or `address`.

Description elements can be either plain text or tagged with HTML mark up. When in plain text, Google Earth applies basic HTML formatting automatically. For example, Google Earth replaces newlines with line break tags and encloses valid URLs in anchor tags to make them hyperlinks. To see examples of HTML tags that are recognized by Google Earth, view https://earth.google.com.

If you provide an attribute specification, the attribute fields of `S` display as a table in the description tag of the placemark for each element of `S`, in the order in which the fields appear in the specification. To construct an attribute spec, call `makeattribspec` and then modify the output to remove attributes or change the `Format` field for one or more attributes. The `latitude` and `longitude` coordinates of `S` are not considered to be attributes. If included in an attribute spec, `kmlwrite` ignores them.

Data Types: `char` | `string` | `cell`

`Icon` — File name of custom icon
defined by viewer (default) | string scalar | character vector | cell array of character vectors

File name of custom icon, specified as a string scalar, character vector or cell array of character vectors.

- If a string scalar or character vector, `kmlwrite` applies the value to all icons.
- If a string vector or cell array of character vectors, specify an icon for each feature; that is, the cell array must be the same length as `S` or `address`.
- If the string scalar or character vector is an Internet URL, the URL must include the protocol type.
- If the icon file name is not in the current folder, or in a folder on the MATLAB path, specify a full or relative path name.

Data Types: `char` | `string` | `cell`

`IconScale` — Scaling factor for icon
positive numeric scalar or vector

Scaling factor for the icon, specified as a positive numeric scalar or vector.

- If a scalar, `kmlwrite` applies the value to all objects
- If a vector, specify a scale factor for each feature. That is, the cell array must be the same length as `S` or `address`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32`

`Color` — Color of icons, lines, or the faces and edges of polygons
defined by viewer (default) | color name | RGB triplet | cell array of color names | string vector of color names | matrix of RGB triplets | `'none'`
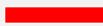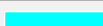
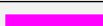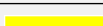Color of icons, lines, or the faces and edges of polygons, specified as one of these values.

- A color name such as `'red'` or a short name such as `'r'`.
- An RGB triplet, which is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A cell array of color names such as `{'red','green','blue'}` or `{'r','g','b'}`.
- A string vector of color names such as `["red" "green" "blue"]` or `["r" "g" "b"]`.

- A matrix of RGB triplets, which is a three-column matrix in which each row is an RGB triplet.

The way you specify the color depends on the desired color scheme.

- To apply the same color to all features, specify a single color name or RGB triplet.
- To apply a different color to each feature, specify a cell array of color names, a string vector of color names, or a matrix of RGB triplets. The number of colors and RGB triplets must match the length of `S` or `address`.
- To create a polygon `geoshape` with no fill or edge color, specify `'none'`.
- To exclude a color specification for icons or lines, specify `'none'`. In this case, the viewer defines the icon or line color.

Note that `Color` specifies the color of polygon faces and edges if you do not specify `FaceColor` and `EdgeColor`.

This table contains the color names and equivalent RGB triplets for some common colors.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Data Types: `char` | `string` | `cell` | `double`

Alpha — Transparency of the icons, lines, or the faces and edges of polygons
1 (fully opaque) (default) | numeric scalar or vector in the range `[0 1]`

Transparency of the icons, lines, or the faces and edges of polygons, specified as a numeric scalar or vector in the range `[0 1]`. If a scalar, kmlwrite applies the value to all features. If a vector, specify a value for each feature; that is, the vector must be the same length as `S` or `address`. If `S` is a polygon geoshape, kmlwrite applies the value to all the polygon faces if `FaceAlpha` is not specified and the polygon edges if `EdgeAlpha` is not specified.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32`

LineWidth — Width of lines and polygon edges in pixels
defined by viewer (default) | positive numeric scalar or vector

Width of lines and polygon edges in pixels, specified as a positive numeric scalar or vector.

- If a scalar, `kmlwrite` applies the value to all polygon edges.
- If a vector, specify a value for each feature. That is, the vector must have the same length as `S`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32`

FaceColor — Color of polygon faces
defined by viewer (default) | color name | RGB triplet | cell array of color names | string vector of color names | matrix of RGB triplets | `'none'`
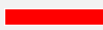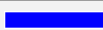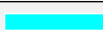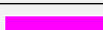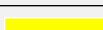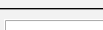
Color of polygon faces, specified as one of these values.

- A color name such as `'red'` or a short name such as `'r'`.
- An RGB triplet, which is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A cell array of color names such as `{'red','green','blue'}` or `{'r','g','b'}`.
- A string vector of color names such as `["red" "green" "blue"]` or `["r" "g" "b"]`.
- A matrix of RGB triplets, which is a three-column matrix in which each row is an RGB triplet.

The way you specify the color depends on the desired color scheme.

- To apply the same color to all polygon faces, specify a single color name or RGB triplet.
- To apply a different color to each face, specify a cell array of color names, a string vector of color names, or a matrix of RGB triplets. The number of colors and RGB triplets must match the length of `S` or `address`.
- To create a polygon `geoshape` with no face color, specify `'none'`.

This table contains the color names and equivalent RGB triplets for some common colors.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Data Types: `char` | `string` | `cell` | `double`

FaceAlpha — Transparency of polygon faces
1 (fully opaque) (default) | numeric scalar or vector in the range `[0 1]`

Transparency of polygon faces, specified as a numeric scalar or vector in the range `[0 1]`.

- If a scalar, `kmlwrite` applies the value to all polygon faces.
- If a vector, specify a value for each polygon face; that is, the vector must be the same length as `S` or `address`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32`

EdgeColor — Color of polygon edges
defined by viewer (default) | color name | RGB triplet | cell array of color names | string vector of color names | matrix of RGB triplets | `'none'`

Color of polygon edges, specified as one of these values.

- A color name such as `'red'` or a short name such as `'r'`.
- An RGB triplet, which is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A cell array of color names such as `{'red','green','blue'}` or `{'r','g','b'}`.
- A string vector of color names such as `["red" "green" "blue"]` or `["r" "g" "b"]`.
- A matrix of RGB triplets, which is a three-column matrix in which each row is an RGB triplet.

The way you specify the color depends on the desired color scheme.

- To apply the same color to the edges of all polygons, specify a single color name or RGB triplet.
- To apply a different edge color for each polygon, specify a cell array of color names, a string vector of color names, or a matrix of RGB triplets. The number of colors and RGB triplets must match the length of `S` or `address`.
- To create a polygon `geoshape` with no edge color, specify `'none'`.

This table contains the color names and equivalent RGB triplets for some common colors.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Data Types: `char` | `string` | `cell` | `double`

EdgeAlpha — Transparency of the polygon edges
1 (fully opaque) (default) | numeric scalar or vector in the range [0 1]

Transparency of the polygon edges, specified as a numeric scalar or vector.

- If a scalar, `kmlwrite` applies the value to all polygon edges.
- If a vector, y specify a value for each polygon edge; that is, the vector must be the same length as `S`. If you do not specify `EdgeAlpha`, `kmlwrite` uses the value of `Alpha`, if specified. If you do not specify either value, `kmlwrite` uses the default value `1` (fully opaque).

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32`

Extrude — Connect polygons to the ground
`false` (default) | `true` | logical or numeric scalar | logical or numeric vector

Connect polygon to the ground, specified as a logical or numeric scalar, `true` (1) or `false` (0), or vector. If a scalar, the values applies to all polygons. If a vector, specify a value for each polygon; that is, the vector must be the same length as `S` or `address`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

CutPolygons — Cut polygon parts
`true` (default) | `false`

Cut polygon parts, specified as a logical or numeric scalar `true` (1) or `false` (0). If `true`, `kmlwrite` cuts polygon parts at the `PolygonCutMeridian` value. If `true`, and the polygon parts require cutting, `kmlwrite` returns an error if the altitude values are nonuniform.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `logical`

PolygonCutMeridian — Meridian where polygon parts are cut
`180` (default) | scalar numeric

Meridian where polygon parts are cut, specified as a scalar numeric.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32`

AltitudeMode — Interpretation of altitude values
`'clampToGround'` | `'relativeToGround'` | `'relativeToSeaLevel'`

Interpretation of altitude values, specified as one of the following values:

Value	Description
`'clampToGround'`	Ignore altitude values and set the feature on the ground. This is the default interpretation when you do not specify altitude values.
`'relativeToGround'`	Set altitude values relative to the actual ground elevation of a particular feature
`'relativeToSeaLevel'`	Set altitude values relative to sea level, regardless of the actual elevation values of the terrain beneath the feature. (Named `'absolute'` in the KML specification.) This is the default interpretation when you specify altitude values.

Data Types: `char` | `string`

LookAt — Position of virtual camera (eye) relative to object being viewed
geopoint vector

Position of the virtual camera (eye) relative to the object being viewed, specified as a geopoint vector. The fields of the geopoint vector, listed below, define the view. `LookAt` is limited to looking down at a feature. To tilt the virtual camera to look above the horizon into the sky, use the `Camera` parameter.

Property Name	Description	Data Type
Latitude	Latitude of the object the camera is looking at, in degrees	Scalar double, from -90 to 90
Longitude	Longitude of the object the camera is looking at, in degrees	Scalar double, from −180 to 180
Altitude	Altitude of the object the camera is looking at from the Earth's surface, in meters	Scalar numeric
Heading	Camera direction (azimuth), in degrees (optional)	Scalar numeric [0 360], default 0 (true North)
Tilt	Angle between the direction of the `LookAt` position and the normal to the surface of the Earth, in degrees (optional)	Scalar numeric [0 90], default 0 (directly above)
Range	Distance in meters from the object specified by `latitude`, `longitude`, and `altitude` to the location of the camera.	Scalar numeric, default 0
AltitudeMode	Interpretation of the camera altitude value (optional)	`'relativeToSeaLevel'`, `'clampToGround'`, (default) `'relativeToGround'`

Camera — Position of virtual camera relative to surface of Earth
geopoint vector

Position of virtual camera (eye) relative to Earth's surface, specified as a geopoint vector. The fields of the geopoint vector, listed below, define the view. `Camera` provides full six degrees of freedom control over the view, so you can position the camera in space and then rotate it around the *x-*, *y-*, and *z*-axes. You can tilt the camera view so that you're looking above the horizon into the sky.

Property Name	Description	Data Type
Latitude	Latitude of the virtual camera (eye), in degrees	Scalar double, in the range [-90 90]
Longitude	Longitude of the virtual camera, in degrees	Scalar double, in the range [-180 180].
Altitude	Distance of the virtual camera from the Earth's surface, in meters	Scalar numeric
Heading	Direction (azimuth) in degrees (optional)	Scalar numeric [0 360], default 0 (true North)
Tilt	Camera rotation around the X-axis, in degrees (optional)	Scalar numeric [0 180], default 0 (directly above)
Roll	Camera rotation in degrees around the Z-axis (optional)	Scalar numeric, in the range [-180 180], default 0
AltitudeMode	Specifies how camera altitude is interpreted (optional)	`'relativeToSeaLevel'`, `'clampToGround'`, `'relativeToGround'` (default)

Tips

- You can view KML files with the Google Earth browser, which must be installed on your computer.

  For Windows, use the `winopen` function:

  ```
  winopen(filename)
  ```

  For Linux, if the file name is a partial path, use the following commands:

  ```
  cmd = 'googleearth ';
  fullfilename = fullfile(pwd, filename);
  system([cmd fullfilename])
  ```

  For Mac, if the file name is a partial path, use the following commands:

  ```
  cmd = 'open -a Google\ Earth ';
  fullfilename = fullfile(pwd, filename);
  system([cmd fullfilename])
  ```

- You can also view KML files with a Google Maps™ browser. The file must be located on a web server that is accessible from the Internet. A private intranet server will not suffice because Google's server must be able to access the URL that you provide. The following is a template for using Google Maps. Replace `your-web-server-path` with a real value.

  ```
  GMAPS_URL = 'http://maps.google.com/maps?q=';
  KML_URL = 'http://your-web-server-path';
  web([GMAPS_URL KML_URL])
  ```

See Also

kmlwriteline | kmlwritepoint | kmlwritepolygon | makeattribspec | shapewrite

Introduced in R2007b

kmlwriteline

Write geographic line data to KML file

Syntax

```
kmlwriteline(filename,latitude,longitude)
kmlwriteline(filename,latitude,longitude,altitude)
kmlwriteline( ___ ,Name,Value)
```

Description

kmlwriteline(filename,latitude,longitude) writes the geographic line data specified by latitude and longitude to the file specified by filename in Keyhole Markup Language (KML) format. kmlwriteline creates a KML Placemark element for each line, using the latitude and longitude values as the coordinates of the points that define the line. kmlwriteline sets the altitude values associated with the line to 0 and sets the altitude interpretation to 'clampToGround'.

kmlwriteline(filename,latitude,longitude,altitude) uses the values of latitude, longitude, and altitude to set the coordinates of the points that define the line. When you specify an altitude value, kmlwriteline sets the AltitudeMode attribute to 'relativeToSeaLevel'.

kmlwriteline(___ ,Name,Value) specifies name-value pairs that set additional KML feature properties. Parameter names can be abbreviated and are case-insensitive.

Examples

Write Line Data to KML File

Load geographic data describing coast lines.

```
load coastlines
```

Define the name of the KML file you want to create.

```
 filename = 'coastlines.kml';
```

Write the line data to the file, specifying the color and width of the line.

```
kmlwriteline(filename, coastlat, coastlon, 'Color','black', ...
        'LineWidth', 3);
```

Retrieve GPS Track Log from GPX File and Write Data to KML File

Read the track log from a GPX file. gpxread returns the data as a geopoint vector.

```
points = gpxread('sample_tracks');
```

Get the latitude, longitude, and altitude values from the data.

```
lat = points.Latitude;
lon = points.Longitude;
alt = points.Elevation;
```

Define the name of the KML file you want to create.

```
filename = 'track.kml';
```

Write the geographic line data to the file, specifying a description and a name.

```
kmlwriteline(filename,lat, lon, alt, ...
        'Description', points.Metadata.Name, 'Name', 'Track Log');
```

Display Equally Spaced Waypoints Along Two Great Circle Tracks

Read the track data into a geopoint vector.

```
cities = geopoint(shaperead('worldcities','UseGeoCoords',true));
```

Get the latitude, longitude, and altitude values from the data. The example uses London and New York.

```
city1 = 'London';
city2 = 'New York';
pt1 = cities(strcmp(city1,cities.Name));
pt2 = cities(strcmp(city2,cities.Name));
lat1 = pt1.Latitude;
lon1 = pt1.Longitude;
lat2 = pt2.Latitude;
lon2 = pt2.Longitude;
nlegs = 20;
[lat,lon] = gcwaypts(lat1,lon1,lat2,lon2,nlegs);
midpoint = nlegs/2;
altscale = 5000;
alt = [0:midpoint midpoint-1:-1:0] * altscale;
```

Specify the view using `LookAt` parameter values.

```
lookLat = 49.155804;
lookLon = -56.698494;
lookAt = geopoint(lookLat, lookLon);
lookAt.Range = 2060400;
lookAt.Heading = 10;
lookAt.Tilt = 70;
```

Write the geographic line data to two KML files, specifying color, width, and view. One track displays altitude values and the other has the track clamped to the ground.

```
width = 4;
filename1 = 'altitudetrack.kml';
kmlwriteline(filename1,lat,lon,alt,'Color','k','LineWidth',width)

filename2 = 'groundtrack.kml';
```

```
kmlwriteline(filename2,lat,lon,alt,'Color','w','LineWidth',width, ...
        'LookAt',lookAt,'AltitudeMode','clampToGround')
```

Input Arguments

`filename` — Name of output file
character vector | string scalar

Name of output file, specified as a string scalar or character vector. `kmlwriteline` creates the file in the current folder, unless you specify a full or relative path name. If the file name includes an extension, it must be `.kml`.

Data Types: `char` | `string`

`latitude` — Latitudes of points that define the line
vector in the range `[-90 90]`

Latitudes of points that define the line, specified as a vector in the range `[-90 90]`.

Data Types: `single` | `double`

`longitude` — Longitudes of points that define the line
vector

Longitudes of points that define the line, specified as a vector. Longitude values automatically wrap to the range `[-180, 180]`.

Data Types: `single` | `double`

`altitude` — Altitude of points that define the line
0 (default) | scalar or vector

Altitude of points that define the line, specified as a scalar or vector. Unit of measure is meters.

- If a scalar, `kmlwriteline` applies the value to each point.
- If a vector, you must specify an altitude value for each point. That is, the vector must have the same length as `latitude` and `longitude`.

Data Types: `single` | `double`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Name','Point Reyes'`

`Name` — Label of line displayed in viewer
`'Line N'` where `N` is the index of the line (default) | string scalar | character vector

Label of line displayed in viewer, specified as a string scalar or character vector.

If the line contains `NaN` values, `kmlwriteline` places the line segments in a folder labeled `'Line 1'`. `kmlwriteline` labels the line segments `'Segment N'`, where `N` is the index value of the line segment.

Data Types: `char` | `string`

Description — Content to be displayed in the line description balloon
string scalar | character vector

Content to be displayed in the line description balloon, specified as a string scalar or character vector. The description appears in the description balloon when the user clicks either the feature name in the Google Earth Places panel or the line in the viewer window.

You can include basic HTML mark up, however, Google Earth applies some HTML formatting automatically. For example, Google Earth replaces newlines with line break tags and encloses valid URLs in anchor tags to make them hyperlinks. To see examples of HTML tags recognized by Google Earth, view https://earth.google.com.

Data Types: `char` | `string`
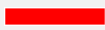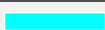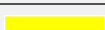
Color — Line color
defined by viewer (default) | color name | RGB triplet | `'none'`

Line color, specified as one of these options.

- A color name such as `'red'` or a short name such as `'r'`.
- An RGB triplet, which is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.

If you do not specify a line color, or specify `'none'`, then the `kmlwriteline` function does not include a color specification in the file and the viewer defines the color.

This table contains the color names and equivalent RGB triplets for some common colors.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Data Types: `char` | `string` | `double`

Alpha — Transparency of line
1 (default) | numeric scalar in the range `[0 1]`

Transparency of line, specified as a numeric scalar in the range `[0 1]`. The default value, `1`, indicates that the line is fully opaque.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

LineWidth — Width of line in pixels
defined by viewer (default) | positive numeric scalar

Width of line in pixels, specified as a positive numeric scalar. If you do not specify a width, `kmlwriteline` does not include width information in the file and the viewer defines the line width.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32`

AltitudeMode — Interpretation of altitude values
`'clampToGround'` (default) | `'relativeToGround'` | `'relativeToSeaLevel'`

Interpretation of altitude values, specified as one of the following:

Value	Description
`'clampToGround'`	Ignore altitude values and set the feature on the ground. The default interpretation when you do not specify `altitude` values.
`'relativeToGround'`	Set altitude values relative to the actual ground elevation of a particular feature.
`'relativeToSeaLevel'`	Set altitude values relative to sea level, regardless of the actual elevation values of the terrain beneath the feature. The default interpretation when you specify altitude values. Called `'absolute'` in KML terminology.

Data Types: `char` | `string`

LookAt — Position of virtual camera (eye) relative to object being viewed
geopoint vector

Position of the virtual camera (eye) relative to the object being viewed, specified as a geopoint vector. The fields of the geopoint vector define the view. `LookAt` is limited to looking down at a feature. To tilt the virtual camera to look above the horizon into the sky, use the `Camera` parameter.

Property Name	Description	Data Type
`Latitude`	Latitude of the line the camera is looking at, in degrees	Scalar double, from -90 to 90
`Longitude`	Longitude of the line the camera is looking at, in degrees	Scalar double, from -180 to 180
`Altitude`	Altitude of the line the camera is looking at from the Earth's surface, in meters	Scalar numeric
`Heading`	Camera direction (azimuth), in degrees (optional)	Scalar numeric `[0 360]`, default `0` (true North)
`Tilt`	Angle between the direction of the `LookAt` position and the normal to the surface of the Earth, in degrees (optional)	Scalar numeric `[0 90]`, default `0` (directly above)

Property Name	Description	Data Type
Range	Distance in meters from the point specified by `latitude`, `longitude`, and `altitude` to the position of the camera.	Scalar numeric, default `0`
AltitudeMode	Interpretation of the camera altitude value (optional)	`'relativeToSeaLevel'`, `'clampToGround'`, (default) `'relativeToGround'`

`Camera` — Position of virtual camera relative to surface of Earth
geopoint vector

Position of virtual camera (eye) relative to Earth's surface, specified as a geopoint vector. The fields of the geopoint vector, listed below, define the view. `Camera` provides full 6 degrees of freedom control over the view, so you can position the camera in space and then rotate it around the *x*-, *y*-, and *z*-axes. You can tilt the camera view so that you're looking above the horizon into the sky.

Property Name	Description	Data Type
Latitude	Latitude of the virtual camera (eye), in degrees	Scalar double, in the range `[-90 90]`
Longitude	Longitude of the virtual camera, in degrees,	Scalar double, in the range `[-180 180]`.
Altitude	Distance of the virtual camera from the Earth's surface, in meters	Scalar numeric
Heading	Direction (azimuth) in degrees (optional)	Scalar numeric `[0 360]`, default `0` (true North)
Tilt	Camera rotation around the X-axis, in degrees (optional)	Scalar numeric `[0 180]`, default `0` (directly above)
Roll	Camera rotation in degrees around the Z-axis (optional)	Scalar numeric, in the range `[-180 180]`, default `0`
AltitudeMode	Specifies how camera altitude is interpreted (optional)	`'relativeToSeaLevel'`, `'clampToGround'`, `'relativeToGround'` (default)

Tips

- If you do not see your line, set `AltitudeMode` to `'clampToGround'`. If the line appears, then you may have a problem with your altitude value.
- You can view KML files with the Google Earth browser, which must be installed on your computer.

  For Windows, use the `winopen` function:

  ```
  winopen(filename)
  ```

  For Linux, if the file name is a partial path, use the following commands:

  ```
  cmd = 'googleearth ';
  fullfilename = fullfile(pwd, filename);
  system([cmd fullfilename])
  ```

For Mac, if the file name is a partial path, use the following commands:

```
cmd = 'open -a Google\ Earth '
fullfilename = fullfile(pwd, filename);
system([cmd fullfilename])
```

- You can also view KML files with a Google Maps browser. The file must be located on a web server that is accessible from the Internet. A private intranet server will not suffice because Google's server must be able to access the URL that you provide. The following is a template for using Google Maps. Replace your-web-server-path with a real value.

```
GMAPS_URL = 'http://maps.google.com/maps?q=';
KML_URL = 'http://your-web-server-path';
web([GMAPS_URL KML_URL])
```

See Also
kmlwrite | kmlwritepoint | kmlwritepolygon | shapewrite

Introduced in R2013a

kmlwritepoint

Write geographic point data to KML file

Syntax

```
kmlwritepoint(filename,latitude,longitude)
kmlwritepoint(filename,latitude,longitude,altitude)
kmlwritepoint( ___ ,Name,Value)
```

Description

kmlwritepoint(filename,latitude,longitude) writes the geographic point data specified by latitude and longitude to the file specified by filename in Keyhole Markup Language (KML) format. kmlwritepoint creates a KML Placemark element for each point, using the latitude and longitude values as coordinates of the points. kmlwritepoint sets the altitude values associated with the points to 0 and sets the altitude interpretation to 'clampToGround'.

kmlwritepoint(filename,latitude,longitude,altitude) writes latitude,longitude, and altitude data as point coordinates. When you specify an altitude value, kmlwritepoint sets the AltitudeMode attribute to 'relativeToSeaLevel'.

kmlwritepoint(___ ,Name,Value) specifies name-value pairs that set additional KML feature properties. Parameter names can be abbreviated and are case-insensitive.

Examples

Write Data to KML File that Includes HTML Format Tags

Define a point by latitude and longitude.

```
lat =  42.299827;
lon = -71.350273;
```

Specify the description text used with the placemark, including HTML tags for formatting.

```
description = sprintf('%s<br>%s</br><br>%s</br>', ...
        '3 Apple Hill Drive', 'Natick, MA. 01760', ...
        'https://www.mathworks.com');
name = 'The MathWorks, Inc.';
iconDir = fullfile(matlabroot,'toolbox','matlab','icons');
iconFilename = fullfile(iconDir, 'matlabicon.gif');
```

Define the name of the KML file you want to create.

```
filename = 'MathWorks.kml';
```

Write the data to a KML file, using the Description parameter to include the names of the cities in the placemarks.

```
kmlwritepoint(filename, lat, lon, ...
        'Description', description, 'Name', name, 'Icon', iconFilename);
```

Retrieve Point Data from Shape File and Write Data to KML File

Read the locations of major cities from a shape file into a geostruct.

```
latlim = [ 30; 75];
lonlim = [-25; 45];
cities = shaperead('worldcities.shp','UseGeoCoords', true, ...
        'BoundingBox', [lonlim, latlim]);
```

Get the latitudes, longitudes, and names of the cities from the geostruct.

```
lat = [cities.Lat];
lon = [cities.Lon];
name = {cities.Name};
```

Define the name of the KML file you want to create.

```
filename = 'European_Cities.kml';
```

Write the geographic data to the file, specifying the names of the cities and the size of the icon.

```
kmlwritepoint(filename, lat, lon, 'Name', name, 'IconScale', 2);
```

Write Point Data to KML File Using Camera to Specify View

Create a geopoint object to specify the viewing options available through the `Camera` parameter. The example sets up a view of the Washington Monument in Washington D.C.

```
camlat = 38.889301;
camlon = -77.039731;
camera = geopoint(camlat,camlon);
camera.Altitude = 500;
camera.Heading = 90;
camera.Tilt = 45;
camera.Roll = 0;
name = 'Camera ground location';
```

Define the name of the KML file you want to create.

```
filename = 'WashingtonMonument.kml';
```

Write the point data to the file with the view specification. Place a marker at the ground location of the camera.

```
lat = camera.Latitude;
lon = camera.Longitude;
kmlwritepoint(filename,lat,lon,'Camera',camera,'Name',name);
```

Write Point Data to KML File Using LookAt to Specify View

Specify the latitude, longitude, and altitude values that define a point. In this example, the location is the Machu Picchu ruins in Peru.

```
lat = -13.163111;
lon = -72.544945;
alt = 2430;
```

Create a geopoint object to specify the viewing options available through the `LookAt` parameter.

```
lookAt = geopoint(lat,lon);
lookAt.Range = 1500;
lookAt.Heading = 260;
lookAt.Tilt = 67;
name = 'LookAt location parameters';
```

Define the name of the KML file you want to create.

```
filename = 'Machu_Picchu.kml';
```

Write the point data to the file, using the `LookAt` parameter to specify the view.

```
kmlwritepoint(filename,lat,lon,alt,'Name',name,'LookAt',lookAt)
```

Use Camera Parameter to Specify View

Specify the latitude and longitude values that define the point that you want to view. In this example, the location is Mount Rainier.

```
lat_rainier = 46.8533;
lon_rainier = -121.7599;
```

Create a geopoint vector to specify the position of the virtual camera (eye) you will use to view the location using the `Camera` parameter.

```
myview = geopoint(46.7, -121.7,'Altitude',2500,'Tilt',85,'Heading',345);
```

Define the name of the KML file you want to create.

```
filename = 'Mt_Rainier.kml';
```

Write the point data to the file, specifying a name and a custom color for the icon.

```
kmlwritepoint(filename,lat_rainier,lon_rainier,'Name','Mt Rainier',...
        'Color','red','IconScale',2,'Camera',myview)
```

Input Arguments

`filename` — Name of output file
string scalar | character vector

Name of output file, specified as a string scalar or character vector. `kmlwritepoint` creates the file in the current folder, unless you specify a full or relative path name. If the file name includes an extension, it must be `.kml`.

Data Types: `char` | `string`

`latitude` — Latitudes of points
vector in the range `[-90 90]`

Latitudes of points, specified as a vector in the range `[-90 90]`.

Data Types: `single` | `double`

`longitude` — Longitudes of points
vector

Longitudes of points, specified as a vector. Longitude values automatically wrap to the range `[-180 180]`, in compliance with the KML specification.

Data Types: `single` | `double`

`altitude` — Altitude of points in meters
0 (default) | scalar or vector

Altitude of points in meters, specified as a scalar or vector.

- If a scalar, `kmlwritepoint` applies the value to each point.
- If a vector, you must specify an altitude value for each point. That is, the vector must have the same length as `latitude` and `longitude`.

Data Types: `single` | `double`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `kmlwritepoint(filename,lat,lon,'Name','Point Reyes','IconScale',2);`

Name — Label of point displayed in viewer
`'Point N'` where N is the index of the point (default) | string scalar | string array | character vector | cell array of character vectors

Label of point displayed in viewer, specified as a string scalar or character vector or cell array of character vectors.

- If a string scalar or character vector, `kmlwritepoint` applies the name to all points.
- If a string array or cell array of character vectors, you must include a label for each point; that is, the cell array must have the same length as `latitude` and `longitude`.

Data Types: `char` | `string` | `cell`

Description — Content to be displayed in the point description balloon
string scalar | character vector | cell array of character vectors

Content to be displayed in the point description balloon, specified as a string scalar or character vector or a cell array of character vectors. The content appears in the description balloon when you click either the feature name in the Google Earth Places panel or the point in the viewer window.

- If a string scalar or character vector, `kmlwritepoint` applies the description to all points.
- If a string array or cell array of character vectors, you must include description information for each point; that is, the cell array must be the same length as `latitude` and `longitude`.

Description elements can be either plain text or marked up with HTML. When it is plain text, Google Earth applies basic formatting, replacing newlines with line break tags and enclosing valid URLs with anchor tags to make them hyperlinks. To see examples of HTML tags that Google Earth recognizes, view https://earth.google.com.

Data Types: `char` | `string` | `cell`

Icon — File name of a custom icon
defined by viewer, for example, Google Earth uses an image of a push pin. (default) | string scalar | string array | character vector | cell array of character vectors

File name of a custom icon, specified as a string scalar or character vector or cell array of character vectors.

- If a string scalar or character vector, `kmlwritepoint` uses the icon for all points.
- If a string array or cell array of character vectors, you must specify an icon for each point. That is, the cell array must be the same length as `latitude` and `longitude`.

If the icon file name is not in the current folder, or in a folder on the MATLAB path, you must specify a full or relative path name. If the file name is an Internet URL, the URL must include the protocol type.

Data Types: `char` | `string` | `cell`

IconScale — Scaling factor for icon
positive numeric scalar or vector

Scaling factor for the icon, specified as a positive numeric scalar or vector.

- If a scalar, `kmlwritepoint` applies the scaling factor to the icon for all points.
- If a vector, you must specify a scaling factor for each icon. That is, the vector must be the same length as `latitude` and `longitude`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32`

Color — Icon color
defined by viewer (default) | color name | RGB triplet | cell array of color names | string vector of color names | matrix of RGB triplets | `'none'`
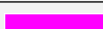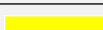
Icon color, specified as one of these options.

- A color name such as `'red'` or a short name such as `'r'`.
- An RGB triplet, which is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A cell array of color names such as `{'red','green','blue'}` or `{'r','g','b'}`.

- A string vector of color names such as `["red" "green" "blue"]` or `["r" "g" "b"]`.
- A matrix of RGB triplets, which is a three-column matrix in which each row is an RGB triplet.

The way you specify the color depends on the desired color scheme.

- To apply the same color to all icons, specify a single color name or RGB triplet.
- To apply a different color to each icon, specify a cell array of color names, a string vector of color names, or a matrix of RGB triplets. The number of colors and RGB triplets must match the length of `latitude` and `longitude`.
- To exclude a color specification for icons, specify `'none'`. In this case, the viewer defines the icon color.

This table contains the color names and equivalent RGB triplets for some common colors.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Data Types: `char` | `string` | `cell` | `double`

Alpha — Transparency of the icons
1 (default) | numeric scalar or vector in the range `[0 1]`

Transparency of the icons, specified as a numeric scalar or vector in the range `[0 1]`. The default value, `1`, indicates fully opaque.

- If a scalar, `kmlwritepoint` applies the value to all icons.
- If a vector, you must specify a value for each icon. That is, the vector must be the same length as `latitude` and `longitude`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32`

AltitudeMode — Interpretation of altitude values
`'clampToGround'` | `'relativeToGround'` | `'relativeToSeaLevel'`

Interpretation of altitude values, specified as one of the following values:

Value	Description
`'clampToGround'`	Ignore altitude values and set the feature on the ground. This is the default interpretation when you do not specify `altitude` values.

Value	Description
`'relativeToGround'`	Set altitude values relative to the actual ground elevation of a particular feature.
`'relativeToSeaLevel'`	Set altitude values relative to sea level, regardless of the actual elevation values of the terrain beneath the feature. This is the default interpretation when you specify altitude values. Called `'absolute'` in KML terminology.

Data Types: `char` | `string`

LookAt — Position of the virtual camera (eye) relative to the object being viewed
geopoint vector

Position of the virtual camera (eye) relative to the object being viewed, specified as a geopoint vector. The view is defined by the fields of the geopoint vector, listed in the table below. `LookAt` is limited to looking down at a feature, you cannot tilt the virtual camera to look above the horizon into the sky. To tilt the virtual camera to look above the horizon into the sky, use the `Camera` parameter.

Property Name	Description	Data Type
`Latitude`	Latitude of the point the camera is looking at, in degrees north or south of the Equator (0 degrees)	Scalar double, from -90 to 90
`Longitude`	Longitude of the point the camera is looking at, in degrees, specifying angular distance relative to the Prime Meridian	Scalar double, in the range `[-180 180]`. Values west of the Meridian range from -180 to 0 degrees. Values east of the Meridian range from 0 to 180 degrees
`Altitude`	Altitude of the point the camera is looking at from the Earth's surface, in meters	Scalar numeric, default 0
`Heading`	Camera direction (azimuth), in degrees (optional)	Scalar numeric `[0 360]`, default 0 (true North)
`Tilt`	Angle between the direction of the LookAt position and the normal to the surface of the earth, in degrees (optional)	Scalar numeric `[0 90]`, default: `0`, directly above.
`Range`	Distance in meters from the point specified by `latitude`, `longitude`, and `altitude` to the point where the camera is positioned—the `LookAt` position.	Scalar numeric, default: `0`
`AltitudeMode`	Interpretation of camera altitude value (optional)	`'relativeToSeaLevel'`, `'clampToGround'`, (default) `'relativeToGround'`

Camera — Position of the virtual camera relative to the surface of the Earth
geopoint vector

Position of the camera relative to the Earth's surface, specified as a geopoint vector. The fields of the geopoint vector, listed below, define the view. `Camera` provides full six-degrees-of-freedom control

over the view, so you can position the camera in space and then rotate it around the *X*, *Y*, and *Z* axes. You can tilt the camera view so that you're looking above the horizon into the sky.

Property Name	Description	Data Type
Latitude	Latitude of the virtual camera (eye), in degrees north or south of the Equator (0 degrees)	Scalar double in the range [-90 90]
Longitude	Longitude of the virtual camera, in degrees, specifying angular distance relative to the Prime Meridian	Scalar double, in the range [-180 180]. Values west of the Meridian range from -180 to 0 degrees. Values east of the Meridian range from 0 to 180 degrees
Altitude	Distance of the virtual camera from the surface of the Earth, in meters	Scalar numeric
Heading	Direction (azimuth) in degrees (optional)	Scalar numeric [0 360], default 0 (true North)
Tilt	Camera rotation around the *X*-axis, in degrees (optional)	Scalar numeric [0 180], default: 0, directly above
Roll	Camera rotation in degrees around the *Z*-axis (optional)	Scalar numeric, in the range [-180 180] default: 0
AltitudeMode	Specifies the interpretation of camera altitude. (optional)	'relativeToSeaLevel', 'clampToGround' , (default) 'relativeToGround'

- If a scalar, kmlwritepoint applies the value to all the points.
- If a vector, you must include an item for each point; that is, the length must be the same length as latitude and longitude.

Tips

- You can view KML files with the Google Earth browser, which must be installed on your computer.

  For Windows, use the winopen function:

  winopen(filename)

  For Linux, if the file name is a partial path, use the following commands:

  ```
  cmd = 'googleearth ';
  fullfilename = fullfile(pwd, filename);
  system([cmd fullfilename])
  ```

  For Mac, if the file name is a partial path, use the following commands:

  ```
  cmd = 'open -a Google\ Earth '
  fullfilename = fullfile(pwd, filename);
  system([cmd fullfilename])
  ```

- You can also view KML files with a Google Maps browser. The file must be located on a web server that is accessible from the Internet. A private intranet server will not suffice because Google's server must be able to access the URL that you provide. The following is a template for using Google Maps. Replace your-web-server-path with a real value.

```
GMAPS_URL = 'http://maps.google.com/maps?q=';
KML_URL = 'http://your-web-server-path';
web([GMAPS_URL KML_URL])
```

See Also

kmlwrite | kmlwriteline | kmlwritepolygon | shapewrite

Introduced in R2013a

kmlwritepolygon

Write geographic polygon to KML file

Syntax

```
kmlwritepolygon(filename,latitude,longitude)
kmlwritepolygon(filename,latitude,longitude,altitude)
kmlwritepolygon( ___ ,Name,Value)
```

Description

`kmlwritepolygon(filename,latitude,longitude)` writes the geographic latitude and longitude data that define polygon vertices to the file specified by `filename` in Keyhole Markup Language (KML) format. `kmlwritepolygon` creates a KML Placemark element for each polygon. By default, `kmlwritepolygon` sets the altitude value associated with the vertices to `0` and sets the altitude interpretation to `'clampToGround'`.

`kmlwritepolygon(filename,latitude,longitude,altitude)` writes the polygon data to a KML file, including `altitude` values for each vertex. `altitude` can be a scalar value, in which case `kmlwritepolygon` uses it as the value for every vertex. If `altitude` is a vector, you must specify a value for every vertex; that is, `altitude` must be the same length as `latitude` and `longitude`. By default, when you specify altitude values, `kmlwritepolygon` sets the altitude interpretation to `'relativeToSeaLevel'`.

`kmlwritepolygon( ___ ,Name,Value)` specifies name-value pairs that set additional KML feature properties. Parameter names can be abbreviated and are not case sensitive.

Examples

Write Coastlines to KML File as Polygon

Load latitude and longitude data that defines the coastlines of the continents.

```
load coastlines
```

Specify the name of output KML file that you want to create.

```
filename = 'coastlines.kml';
```

Write the coastline data to the file as a polygon.

```
kmlwritepolygon(filename,coastlat,coastlon)
```

Create Polygon with Inner Ring

Define the latitude and longitude coordinates of the center of the rings. For this example, the coordinates specify the Eiffel Tower.

```
lat0 = 48.858288;
lon0 = 2.294548;
```

Define the inner radius and the outer radius of two small circles. The examples calls `poly2ccw` to change the direction of the vertex order of the second circle to counter-clockwise. This change of direction is needed to define the space between the two circles as a ring-shaped polygon.

```
outerRadius = .02;
innerRadius = .01;
[lat1,lon1] = scircle1(lat0,lon0,outerRadius);
[lat2,lon2] = scircle1(lat0,lon0,innerRadius);
[lon2,lat2] = poly2ccw(lon2,lat2);
lat = [lat1; NaN; lat2];
lon = [lon1; NaN; lon2];
alt = 500;
```

Specify name of output KML file and write the data to the file.

```
filename = 'EiffelTower.kml';
kmlwritepolygon(filename,lat,lon,alt, ...
      'EdgeColor','g','FaceColor','c','FaceAlpha',.5)
```

Create Polygon That Spans the 180 Degree Meridian

Specify latitude and longitude coordinates that define the vertices of the polygon. For this example, specify longitude values that span the 180 degree meridian.

```
lat = [0 1 1 0 0];
lon = [179.5 179.5 -179.5 -179.5 179.5];
h = 5000;
alt = ones(1,length(lat)) * h;
filename = 'cross180.kml';
kmlwritepolygon(filename,lat,lon,alt,'EdgeColor','r','FaceColor','w')
```

By default, the polygon contains a seam at the 180 degree mark. To remove this seam, set `PolygonCutMeridian` to `0`.

```
filename = 'noseam.kml';
kmlwritepolygon(filename,lat,lon,alt,'EdgeColor','r', ...
      'FaceColor','w','PolygonCutMeridian',0);
```

To display a ramp without a seam, wrap the longitude values to the range `[0 360]`, and set `CutPolygon` to `false`. Use the `Extrude` parameter to connect the polygon to the ground for better visibility.

```
filename = 'ramp.kml';
lon360 = wrapTo360(lon);
altramp = [0 0 h h 0];
kmlwritepolygon(filename,lat,lon360,altramp,'EdgeColor','r', ...
      'FaceColor','w','CutPolygons',false,'Extrude',true);
```

Input Arguments

`filename` — Name of output file
string scalar | character vector

Name of output file, specified as a string scalar or character vector. `kmlwritepolygon` creates the file in the current folder, unless you specify a full or relative path name. If the file name includes an extension, it must be `.kml`.

Data Types: `char` | `string`

`latitude` — Latitudes of polygon vertices
vector in the range `[-90 90]`

Latitudes of polygon vertices, specified as a vector in the range `[-90 90]`.

Data Types: `single` | `double`

`longitude` — Longitude of polygon vertices
vector in the range `[-180, 180]`

Longitude of polygon vertices, specified as a vector in the range `[-180, 180]`.

Data Types: `single` | `double`

`altitude` — Altitude of polygon vertices
0 (default) | scalar or vector

Altitude of polygon vertices, specified as a scalar or vector. Unit of measure is meters.

- If a scalar, `kmlwritepolygon` applies the value to each point.
- If a vector, you must specify an altitude value for each vertex. That is, the vector `altitude` must be the same length as `latitude` and `longitude`.

Data Types: `single` | `double`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example:
`kmlwritepolygon(filename,lat,lon,alt,'EdgeColor','g','FaceColor','c','FaceAlpha',.5)`

`Name` — Label displayed in the viewer for the polygon
`'Polygon 1'` (default) | string scalar | character vector

Label displayed in the viewer for the polygon, specified as the comma-separated pair consisting of `'Name'` and a string scalar or character vector. If the vertex list contains multiple outer rings, `kmlwritepolygon` creates a folder with the value of `Name` and each outer ring labeled `'Part N'` where `N` varies from `1` to the number of outer rings.

Data Types: `char` | `string`

Description — Content to be displayed in the polygon description balloon
character vector | string scalar

Content to be displayed in the polygon description balloon, specified as the comma-separated pair consisting of `'Description'` and a string scalar or character vector. The content appears in the description balloon when you click either the feature name in the Google Earth Places panel or the polygon in the viewer window.

Description elements can be either plain text or HTML markup. When it is plain text, Google Earth applies basic formatting, replacing newlines with line break tags and enclosing valid URLs with anchor tags to make them hyperlinks. To see examples of HTML tags that Google Earth recognizes, see https://earth.google.com.

Data Types: `char` | `string`
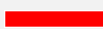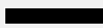
FaceColor — Color of polygon face
defined by viewer (default) | color name | RGB triplet | `'none'`

Color of the polygon face, specified as the comma-separated pair of `'FaceColor'` and one of these options.

- A color name — A color name such as `'red'` or a short name such as `'r'`.
- An RGB triplet — A three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- `'none'` — No fill color.

If you do not specify a face color, then the `kmlwritepolygon` function does not include a color specification in the file and the viewer defines the color.

This table contains the color names and equivalent RGB triplets for some common colors.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Data Types: `char` | `string` | `double`

FaceAlpha — Transparency of the polygon face
1 (default) | numeric scalar in the range `[0 1]`

Transparency of the polygon face, specified as the comma-separated pair consisting of `'FaceAlpha'` and a numeric scalar in the range `[0 1]`. The default value, `1`, indicates that the face is fully opaque.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`
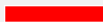
EdgeColor — Color of polygon edges
defined by viewer (default) | color name | RGB triplet | `'none'`

Color of polygon edges, specified as the comma-separated pair of `'EdgeColor'` and one of these values.

- A color name such as `'red'` or a short name such as `'r'`.
- An RGB triplet, which is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- `'none'` — No edge color.

If you do not specify an edge color, then the `kmlwritepolygon` function does not include a color specification in the file and the viewer defines the color.

This table contains the color names and equivalent RGB triplets for some common colors.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Data Types: `char` | `string` | `double`

EdgeAlpha — Transparency of polygon edges
1 (default) | numeric scalar in the range `[0 1]`

Transparency of polygon edge, specified as the comma-separated pair consisting of `'EdgeAlpha'` and a numeric scalar in the range `[0 1]`. The default value, 1, indicates that the edge is fully opaque.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

LineWidth — Width of the polygon edge in pixels
determined by viewer (default)

Width of the polygon edge in pixels, specified as the comma-separated pair consisting of `'LineWidth'` and a positive numeric scalar.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Extrude — Connect polygon to the ground
`false` (default) | `true`

Connect polygon to the ground, specified as the comma-separated pair consisting of `'Extrude'` and a logical scalar or numeric value `true` (1) or `false` (0).

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

CutPolygons — Cut polygon parts
`true` (default) | `false`

Cut polygon parts, specified as the comma-separated pair consisting of `'CutPolygons'` and a logical scalar or numeric value `true` (1) or `false` (0). If `true`, `kmlwritepolygon` cuts polygons at the meridian specified by `PolygonCutMeridian`. `kmlwritepolygon` returns an error if you set this to `true`, the polygon parts require cutting, and the altitude values are nonuniform.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

PolygonCutMeridian — Meridian where polygon parts are cut
`180` (default) | scalar numeric

Meridian where polygon parts are cut, specified as the comma-separated pair consisting of `'PolygonCutMeridian'` and a scalar numeric value.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

AltitudeMode — Interpretation of altitude values
`'clampToGround'` | `'relativeToGround'` | `'relativeToSeaLevel'`

Interpretation of altitude values, specified as the comma-separated pair consisting of `'AltitudeMode'` and any of the following values:

Value	Meaning
`'clampToGround'`	Ignore the altitude values and set the feature on the ground. This value is the default interpretation when you do not specify `altitude` values.
`'relativeToGround'`	Set altitude values relative to the actual ground elevation of a particular feature.
`'relativeToSeaLevel'`	Set altitude values relative to sea level, regardless of the actual elevation values of the terrain beneath the feature. This value is the default interpretation when you specify altitude values. In KML terminology, this interpretation is called `'absolute'`.

Data Types: `char`

LookAt — Position of virtual camera (eye) relative to object being viewed
geopoint vector

Position of virtual camera (eye) relative to object being viewed, specified as the comma-separated pair consisting of `'PLookAt'` and a geopoint vector. The `LookAt` parameter defines the virtual camera that views the polygon. The fields of the geopoint vector define the view, outlined in the table below. `LookAt` can only look down at a feature. To tilt the virtual camera to look above the horizon into the sky, use the `Camera` parameter.

Field	Meaning	Value
`'Latitude'`	Latitude of the point the camera is looking at, in degrees	Scalar `double`
`'Longitude'`	Longitude of the point the camera is looking at, in degrees	Scalar `double`

Field	Meaning	Value
'Altitude'	Altitude of the point the camera is looking at, in meters (optional)	Scalar numeric default: 0
'Heading'	Camera direction (azimuth), in degrees (optional)	Scalar numeric [0 360] default 0
'Tilt'	Angle between the direction of the LookAt position and the normal to the surface of the Earth (optional)	Scalar numeric [0 90] default: 0
'Range'	Distance in meters from the point to the LookAt position	Scalar numeric
'AlititudeMode'	Specifies how the altitude is interpreted for the LookAt point (optional)	'relativeToSeaLevel', 'clampToGround', (default), 'relativeToGround'

Camera — Position and viewing direction of the virtual camera relative to the Earth's surface
geopoint vector

Position and viewing direction of the camera relative to the Earth's surface, specified as the comma-separated pair consisting of 'Camera' and a geopoint vector. The vector contains the following fields. The camera value provides full six-degrees-of-freedom control over the view, so you can position the camera in space and then rotate it around the *X*, *Y*, and *Z* axes. Most importantly, you can tilt the camera view to look above the horizon into the sky.

Field	Meaning	Value
'Latitude'	Latitude of the eye point (virtual camera), specified in degrees	Scalar double
'Longitude'	Longitude of the eye point (virtual camera), specified in degrees	Scalar double
'Altitude'	Distance of the camera from the Earth's surface, specified in meters	Scalar numeric default: 0
'Heading'	Camera direction (azimuth) in degrees (Optional)	Scalar numeric [0 360] default 0
'Tilt'	Camera rotation around the *X* axis, specified in degrees (Optional)	Scalar numeric [0 180] default: 0
'Roll'	Camera rotation around the *Z* axis, specified in degrees (Optional)	Scalar numeric, default: 0
'AlititudeMode'	Specifies how kmlwritepolygon interprets camera altitude values. (Optional)	'relativeToSeaLevel', 'clampToGround', (default), 'relativeToGround'

See Also

kmlwrite | kmlwritepoint | kmlwriteline | shapewrite

Introduced in R2016a

latitudeToIntrinsicY

Package: `map.rasterref`

Convert from latitude to intrinsic *y* coordinates

Syntax

```
yIntrinsic = latitudeToIntrinsicY(R,lat)
```

Description

`yIntrinsic = latitudeToIntrinsicY(R,lat)` returns the *y*-coordinate in the intrinsic coordinate system corresponding to latitude `lat` in the geographic coordinate system, based on the relationship defined by geographic raster R.

Examples

Find Intrinsic *y*-Coordinate from Latitude

Import elevation data [1] on page 1-0   for a region in Colorado as an array and a `GeographicPostingsReference` object. Find the intrinsic *y*-coordinate corresponding to a latitude of `39.9539` degrees.

```
[Z,R] = readgeoraster('n39_w106_3arc_v2.dt1');
yIntrinsic = latitudeToIntrinsicY(R,39.9539)
```

```
yIntrinsic = 56.3200
```

The result means that coordinates with a latitude of `39.9539` degrees fall between rows 56 and 57 of the posting points.

[1] The elevation data used in this example is from the U.S. Geological Survey.

Input Arguments

R — Geographic raster
`GeographicCellsReference` or `GeographicPostingsReference` object

Geographic raster, specified as a `GeographicCellsReference` or `GeographicPostingsReference` object.

`lat` — Latitude coordinates
numeric array

Latitude coordinates, specified as a numeric array. Valid values of `lat` are in the range [-90, 90] degrees. `lat` coordinates can be outside the bounds of the raster R.

Data Types: `single` | `double`

Output Arguments

yIntrinsic — *y*-coordinates in intrinsic coordinate system
numeric array

y-coordinates in intrinsic coordinate system, returned as a numeric array. `yIntrinsic` is the same size as `lat`.

When a point has valid latitude outside the bounds of raster `R`, `yIntrinsic`(*k*) is extrapolated in the intrinsic coordinate system. When `lat`(*k*) is outside the range [-90, 90] degrees, or has a value of `NaN`, the corresponding value `yIntrinsic`(*k*) is set to `NaN`.

Data Types: `double`

See Also

`geographicToIntrinsic` | `intrinsicYToLatitude` | `longitudeToIntrinsicX`

Introduced in R2013b

latlon2pix

(To be removed) Convert latitude-longitude coordinates to pixel coordinates

---

Note The `latlon2pix` function will be removed in a future release. Use the `geographicToIntrinsic` function instead. For more information, see "Compatibility Considerations".

Syntax

```
[row, col ] = latlon2pix(R,lat,lon)
```

Description

`[row, col ] = latlon2pix(R,lat,lon)` calculates pixel coordinates `row`, `col` from latitude-longitude coordinates `lat`, `lon`. `R` is either a 3-by-2 referencing matrix that transforms intrinsic pixel coordinates to geographic coordinates, or a geographic raster reference object. `lat` and `lon` are vectors or arrays of matching size. The outputs `row` and `col` have the same size as `lat` and `lon`. `lat` and `lon` must be in degrees.

Longitude wrapping is handled in the following way: Results are invariant under the substitution `lon = lon +/- n * 360` where `n` is an integer. Any point on the Earth that is included in the image or gridded data set corresponding to `r` will yield row/column values between 0.5 and 0.5 + the image height/width, regardless of what longitude convention is used.

Examples

Find the pixel coordinates of the upper left and lower right outer corners of a 2-by-2 degree gridded data set.

```
R = georefcells([-90 90],[0 360],2,2,'ColumnsStartFrom','north')
[UL_row, UL_col] = latlon2pix(R,  90, 0)     % Upper left
[LR_row, LR_col] = latlon2pix(R, -90, 360)   % Lower right
```

Compatibility Considerations

latlon2pix will be removed
Not recommended starting in R2021b

Some functions that accept referencing matrices as input will be removed, including the `latlon2pix` function. Use a geographic raster reference object and the `geographicToIntrinsic` function instead. Reference objects have several advantages over referencing matrices.

- Unlike referencing matrices, reference objects have properties that document the size of the associated raster, its geographic limits, and the direction of its rows and columns. For more information about reference object properties, see the `GeographicCellsReference` and `GeographicPostingsReference` objects.

- You can manipulate the limits of rasters associated with reference objects using the `geocrop` function.

- You can manipulate the size and resolution of rasters associated with reference objects using the `georesize` function.
- Most functions that accept referencing matrices as inputs also accept reference objects.

To update your code, first create a reference object for either a raster of cells using the `georefcells` function or a raster of regularly posted samples using the `georefpostings` function. Alternatively, convert from a referencing matrix to a reference object using the `refmatToGeoRasterReference` function.

Then, replace uses of the `latlon2pix` function with the `geographicToIntrinsic` function according to this pattern.

Will Be Removed	Recommended
`[row,col] = latlon2pix(R,lat,lon);`	`[col,row] = geographicToIntrinsic(R,lat,lon)`

See Also

`geographicToIntrinsic` | `worldToIntrinsic` | `intrinsicToGeographic`

Introduced before R2006a

lcolorbar

Colorbar with text labels

Syntax

```
lcolorbar(labels)
lcolorbar(labels,'property',value,...)
hcb = lcolorbar(...)
```

Description

`lcolorbar(labels)` appends text labels to a colorbar at the center of each color band. The `labels` input argument is specified as a string array or cell array of character vectors. The number of elements in `labels` must match the length of the colormap.

`lcolorbar(labels,'property',value,...)` controls the properties of the colorbar.

Property	Description
`Location`	Controls the location of the colorbar. Valid values are `'vertical'` (the default) or `'horizontal'`.
`TitleString`	Text of title.
`XLabelString`	Text of *x*-label.
`YLabelString`	Text of *y*-label.
`ZLabelString`	Text of *z*-label.
`ColorAlignment`	Controls whether the colorbar labels are centered on the color bands or the color breaks. Valid values are `'center'` and `'ends'`.

Other valid property-value pairs are any properties and values that can be applied to the title and labels of the colorbar axes.

`hcb = lcolorbar(...)` returns a handle to the colorbar axes.

Examples

```
figure; colormap(jet(5))
labels = {'apples','oranges','grapes','peaches','melons'};
lcolorbar(labels,'fontweight','bold');
```

See Also

contourcmap | **Colormap Editor**

legs

Courses and distances between navigational waypoints

Syntax

```
[course,dist] = legs(lat,lon)
[course,dist] = legs(lat,lon,method)
[course,dist] = legs(pts, ___ )
mat = legs( ___ )
```

Description

`[course,dist] = legs(lat,lon)` returns the azimuths (`course`) and distances (`dist`) between navigational waypoints, which are specified by the column vectors `lat` and `lon`.

`[course,dist] = legs(lat,lon,method)` specifies the logic for the leg characteristics. If the `method` is `'rh'` (the default), `course` and `dist` are calculated in a rhumb line sense. If `method` is `'gc'`, great circle calculations are used.

`[course,dist] = legs(pts, ___ )` specifies waypoints in a single two-column matrix, `pts`.

`mat = legs( ___ )` packs up the outputs into a single two-column matrix, `mat`.

This is a navigation function. All angles are in degrees, and all distances are in nautical miles. Track legs are the courses and distances traveled between navigational waypoints.

Examples

Imagine an airplane taking off from Logan International Airport in Boston (42.3ºN,71ºW) and traveling to LAX in Los Angeles (34ºN,118ºW). The pilot wants to file a flight plan that takes the plane over O'Hare Airport in Chicago (42ºN,88ºW) for a navigational update, while maintaining a constant heading on each of the two legs of the trip.

What are those headings and how long are the legs?

```
lat = [42.3; 42; 34];
long = [-71; -88; -118];
[course,dist] = legs(lat,long,'rh')

course =
  268.6365
  251.2724
dist =
  1.0e+003 *
    0.7569
    1.4960
```

Upon takeoff, the plane should proceed on a heading of about 269º for 756 nautical miles, then alter course to 251º for another 1495 miles.

How much farther is it traveling by not following a great circle path between waypoints? Using rhumb lines, it is traveling

```
totalrh = sum(dist)

totalrh =
    2.2530e+003
```

For a great circle route,

```
[coursegc,distgc] = legs(lat,long,'gc'); totalgc = sum(distgc)

totalgc =
    2.2451e+003
```

The great circle path is less than one-half of one percent shorter.

See Also
dreckon | gcwaypts | navfix | track

length

Return number of elements in geographic or planar vector

Syntax

```
n = length(v)
```

Description

`n = length(v)` returns the number of elements contained in the geographic or planar vector `v`.

Examples

Find Length of Geopoint Vector

Create a geopoint vector and find its length.

```
load coastlines
p = geopoint(coastlat, coastlon);
length(p)
```

```
ans = 9865
```

```
length(coastlat)
```

```
ans = 9865
```

Input Arguments

v — Geographic or planar vector
geopoint, geoshape, mappoint, or mapshape object

Geographic or planar vector, specified as a `geopoint`, `geoshape`, `mappoint`, or `mapshape` object.

Output Arguments

n — Number of elements
nonnegative integer scalar

Number of elements in vector `v`, returned as a nonnegative integer scalar. The result is equivalent to `size(v,1)`.

See Also
size

Introduced in R2012a

lightm

Project light objects on map axes

Syntax

```
h = lightm(lat,lon)
h = lightm(lat,lon,PropertyName,PropertyValue,...)
h = lightm(lat,lon,alt)
```

Description

`h = lightm(lat,lon)` projects a light object at the coordinates `lat` and `lon`. The handle, `h`, of the object can be returned.

`h = lightm(lat,lon,PropertyName,PropertyValue,...)` allows the specification of any property name/property value pair supported by the standard MATLAB `light` function.

`h = lightm(lat,lon,alt)` allows the specification of an altitude, `alt`, for the light object. When omitted, the default is an infinite light source altitude.

Examples

Add Light to Map

Load elevation raster data and a geographic cells reference object. Create a globe frame using map axes.

```
load topo60c
axesm globe
view(120,30)
axis off
```

Display the data and apply a colormap appropriate for elevation data.

```
meshm(topo60c,topo60cR)
demcmap(topo60c)
```

Add light to the globe by specifying a latitude, longitude, and color. Set reflectance properties by using the `material` function. Change the lighting method using the `lighting` function.

```
lightm(0,90,'Color','y')
material([.5 .5 1])
lighting gouraud
```
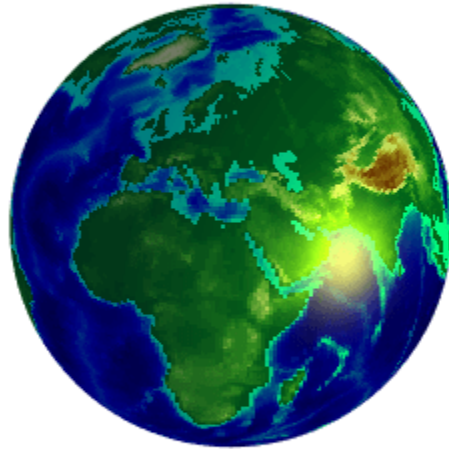
Add a second light source that uses a different color.

```
lightm(90,0,'Color','m')
```

See Also
`light`

Introduced before R2006a

limitm

(To be removed) Determine latitude and longitude limits of regular data grid

---

Note `limitm` will be removed in a future release. Instead, create a geographic raster reference object, and query its `LatitudeLimits` and `LongitudeLimits` properties. For more information, see "Compatibility Considerations".

Syntax

```
[latlim,lonlim] = limitm(Z,R)
latlonlim = limitm(Z,R)
```

Description

`[latlim,lonlim] = limitm(Z,R)` computes the latitude and longitude limits of the geographic quadrangle bounding the regular data grid Z spatially referenced by R. R can be a geographic raster reference object, a referencing vector, or a referencing matrix.

If R is a geographic raster reference object, its `RasterSize` property must be consistent with `size(Z)`.

If R is a referencing vector, it must be 1-by-3 with elements:

```
[cells/degree northern_latitude_limit western_longitude_limit]
```

If R is a referencing matrix, it must be 3-by-2 and transform raster row and column indices to/from geographic coordinates according to:

```
[lon lat] = [row col 1] * R
```

If R is a referencing matrix, it must also define a (non-rotational, non-skewed) relationship in which each column of the data grid falls along a meridian and each row falls along a parallel. The output `latlim` is a vector of the form `[southern_limit northern_limit]` and `lonlim` is a vector of the form `[western_limit eastern_limit]`. All angles are in units of degrees.

`latlonlim = limitm(Z,R)` concatenates `latlim` and `lonlim` into a 1-by-4 row vector of the form:

```
[southern_limit northern_limit western_limit eastern_limit]
```

Examples

Load elevation raster data and a geographic cells reference object. Then, find the latitude and longitude limits.

```
load topo60c
[latlimits,lonlimits] = limitm(topo60c,topo60cR)

latlimits =

   -90    90
```

```
lonlimits =

     0   360
```

The data set covers the whole globe, so the result is expected.

Compatibility Considerations

limitm will be removed
Not recommended starting in R2020b

Some functions that accept referencing vectors or referencing matrices as input will be removed, including the `limitm` function. Instead, create a geographic raster reference object, and query its `LatitudeLimits` and `LongitudeLimits` properties. Reference objects have several advantages over referencing vectors and matrices.

- Unlike referencing vectors and matrices, reference objects have properties that document the size of the associated raster, its geographic limits, and the direction of its rows and columns. For more information about reference object properties, see the `GeographicCellsReference` and `GeographicPostingsReference` objects.
- You can manipulate the limits of rasters associated with reference objects using the `geocrop` function.
- You can manipulate the size and resolution of rasters associated with reference objects using the `georesize` function.

To update your code, first create a reference object for either a raster of cells using the `georefcells` function or a raster of regularly posted samples using the `georefpostings` function. Alternatively, convert from a referencing vector or a referencing matrix to a reference object using the `refvecToGeoRasterReference` or `refmatToGeoRasterReference` function, respectively.

Then, find the limits of the raster by querying the `LatitudeLimits` and `LongitudeLimits` properties of the reference object, `R`.

```
latlim = R.LatitudeLimits;
lonlim = R.LongitudeLimits;
```

See Also
`GeographicPostingsReference` | `GeographicCellsReference`

Introduced before R2006a

Line Properties

Geographic line appearance and behavior

Description

`Line` properties control the appearance and behavior of a `Line` object. By changing property values, you can modify certain aspects of the line plot.

```
uif = uifigure;
g = geoglobe(uif);
p = geoplot3(g,51.5074,0.1900,200)
p.Marker = 'o';
p.LineWidth = 2;
```

Properties

Line

`Color` — Line color
[0 0 0] (default) | RGB triplet | hexadecimal color code | 'r' | 'g' | 'b' | ...

Line color, specified as an RGB triplet, a hexadecimal color code, a color name, or a short name.

RGB triplets and hexadecimal color codes are useful for specifying custom colors.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1]; for example, [0.4 0.6 0.7].
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (#) followed by three or six hexadecimal digits, which can range from 0 to F. The values are not case sensitive. Thus, the color codes '#FF8800', '#ff8800', '#F80', and '#f80' are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
'red'	'r'	[1 0 0]	'#FF0000'	
'green'	'g'	[0 1 0]	'#00FF00'	
'blue'	'b'	[0 0 1]	'#0000FF'	
'cyan'	'c'	[0 1 1]	'#00FFFF'	
'magenta'	'm'	[1 0 1]	'#FF00FF'	
'yellow'	'y'	[1 1 0]	'#FFFF00'	
'black'	'k'	[0 0 0]	'#000000'	
'white'	'w'	[1 1 1]	'#FFFFFF'	

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
[0 0.4470 0.7410]	'#0072BD'	
[0.8500 0.3250 0.0980]	'#D95319'	
[0.9290 0.6940 0.1250]	'#EDB120'	
[0.4940 0.1840 0.5560]	'#7E2F8E'	
[0.4660 0.6740 0.1880]	'#77AC30'	
[0.3010 0.7450 0.9330]	'#4DBEEE'	
[0.6350 0.0780 0.1840]	'#A2142F'	

Example: 'blue'

Example: [0 0 1]

Example: '#0000FF'

LineStyle — Line style
'-' (default) | 'none'

Line style, specified as one of these options:

Line Style	Description	Resulting Line
'-'	Solid line (default)	───────────
'none'	No line	No line

LineWidth — Line width
0.5 (default) | positive value

Line width, specified as a positive value in points, where 1 point = 1/72 of an inch. If the line has markers, then the line width also affects the marker edges.

The line width cannot be thinner than the width of a pixel. If you set the line width to a value that is less than the width of a pixel on your system, the line displays as one pixel wide.

SeriesIndex — Series index
positive integer

Series index, specified as a positive integer. This property is useful for reassigning the colors of several `Line` objects so that they match each other. By default, the `SeriesIndex` property of a line object is a number that corresponds to the object's order of creation, starting at 1.

MATLAB uses the number to calculate indices for assigning color when you call plotting functions. The indices refer to the rows of the arrays stored in the `ColorOrder` property of the parent object.

MATLAB automatically updates the color of the `Line` object when you change its `SeriesIndex`, or when you change the `ColorOrder` property on the parent object. However, the following conditions must be true for the changes to have any effect:

- The `SeriesIndex` property on the `Line` object is greater than `0`.
- The `NextSeriesIndex` property on the parent object is greater than `0`.

Markers

Marker — Marker symbol
'none' (default) | 'o'

Marker symbol, specified as `'none'` or `'o'`. By default, the line does not display markers. Specify `'o'` to display circle markers at each data point or vertex.

Markers do not tilt or rotate as you navigate the globe.

MarkerIndices — Indices of data points at which to display markers
1:length(LatitudeData) (default) | vector of positive integers | scalar positive integer

Indices of data points at which to display markers, specified as a vector of positive integers. If you do not specify the indices, then MATLAB displays a marker at every data point.

Note To see the markers, you must also specify a marker symbol.

Example: `geoplot3(g,lat,lon,h,'-o','MarkerIndices',[1 5 10])` displays a circle marker at the first, fifth, and tenth data points.

Example: `geoplot3(g,lat,lon,h,'-o','MarkerIndices',1:3:length(lat))` displays a circle marker every three data points.

Example: `geoplot3(g,lat,lon,h,'Marker','o','MarkerIndices',5)` displays one circle marker at the fifth data point.

MarkerSize — Marker size
6 (default) | positive value

Marker size, specified as a positive value in points, where 1 point = 1/72 of an inch.

Coordinate Data

LatitudeData — Latitude values
vector

Latitude values, specified as a vector. `LatitudeData` and `LongitudeData` must have the same size.

LongitudeData — Longitude values
vector

Longitude values, specified as a vector. `LatitudeData` and `LongitudeData` must have the same size.

HeightData — Height values
scalar | vector

Height values, specified as a scalar or vector. If `HeightData` is a scalar, then its value is applied to every element in `LatitudeData` and `LongitudeData`. If `HeightData` is a vector, it must be the same size as `LatitudeData` and `LongitudeData`.

HeightReference — Height reference
'geoid' (default) | 'terrain' | 'ellipsoid'

Height reference, specified as one of these values:

- 'geoid' – Height values are relative to the geoid (mean sea level).
- 'terrain' – Height values are relative to the ground.
- 'ellipsoid' – Height values are relative to the WGS84 reference ellipsoid.

For more information about terrain, geoid, and ellipsoid height, see "Find Ellipsoidal Height from Orthometric and Geoid Height".

Interactivity

Visible — State of visibility
'on' (default) | 'off'

State of visibility, specified as one of these values:

- 'on' — Display the object.
- 'off' — Hide the object without deleting it. You still can access the properties of an invisible object.

Parent/Child

Parent — Parent
GeographicGlobe object

Parent, specified as a GeographicGlobe object.

Children — Children
empty GraphicsPlaceholder array

The object has no children. You cannot set this property.

HandleVisibility — Visibility of object handle
'on' (default) | 'off' | 'callback'

Visibility of the object handle in the Children property of the parent, specified as one of these values:

- 'on' — Object handle is always visible.
- 'off' — Object handle is invisible at all times. This option is useful for preventing unintended changes by another function. Set the HandleVisibility to 'off' to temporarily hide the handle during the execution of that function.
- 'callback' — Object handle is visible from within callbacks or functions invoked by callbacks, but not from within functions invoked from the command line. This option blocks access to the object at the command line, but permits callback functions to access it.

If the object is not listed in the Children property of the parent, then functions that obtain object handles by searching the object hierarchy or querying handle properties cannot return it. Examples of such functions include the get, findobj, and close functions.

Hidden object handles are still valid. Set the root `ShowHiddenHandles` property to `'on'` to list all object handles regardless of their `HandleVisibility` property setting.

Identifiers

Type — Type of graphics object
`'line'`

Type of graphics object, returned as `'line'`. Use this property to find all objects of a given type within a plotting hierarchy, for example, searching for the type using `findobj`.

Tag — Object identifier
`''` (default) | character vector | string scalar

Object identifier, specified as a character vector or string scalar. You can specify a unique `Tag` value to serve as an identifier for an object. When you need access to the object elsewhere in your code, you can use the `findobj` function to search for the object based on the `Tag` value.

UserData — User data
`[]` (default) | array

User data, specified as any MATLAB array. For example, you can specify a scalar, vector, matrix, cell array, character array, table, or structure. Use this property to store arbitrary data on an object.

If you are working in App Designer, create public or private properties in the app to share data instead of using the `UserData` property. For more information, see "Share Data Within App Designer Apps".

See Also
`geoplot3` | `geoglobe`

Introduced in R2020a

linecirc

Intersections of circles and lines in Cartesian plane

Syntax

```
[xout,yout] = linecirc(slope,intercpt,centerx,centery,radius)
```

Description

`[xout,yout] = linecirc(slope,intercpt,centerx,centery,radius)` finds the points of intersection given a circle defined by a center and radius in *x-y* coordinates, and a line defined by slope and *y*-intercept, or a slope of "inf" and an *x*-intercept. Two points are returned. When the objects do not intersect, `NaN`s are returned.

When the line is tangent to the circle, two identical points are returned. All inputs must be scalars.

See Also
```
circcirc
```

Introduced before R2006a

linem

Project line object on map axes

Syntax

```
h = linem(lat,lon)
h = linem(lat,lon,LineSpec)
h = linem(lat,lon,PropertyName,PropertyValue,...)
h = linem(lat,lon,z)
```

Description

`h = linem(lat,lon)` displays projected line objects on the current map axes. `lat` and `lon` are the latitude and longitude coordinates, respectively, of the line object to be projected. Note that this ordering is conceptually reversed from the MATLAB `line` function, because the *vertical* (*y*) coordinate comes first. However, the ordering latitude, then longitude, is standard geographic usage. `lat` and `lon` must be the same size and in the `AngleUnits` of the map axes. The object handle for the displayed line can be returned in `h`.

`h = linem(lat,lon,LineSpec)` specifies the line style, marker, and color on page 1-819.

`h = linem(lat,lon,PropertyName,PropertyValue,...)` allows the specification of any number of property name/property value pairs for any properties recognized by the MATLAB `line` function except for `XData`, `YData`, and `ZData`.
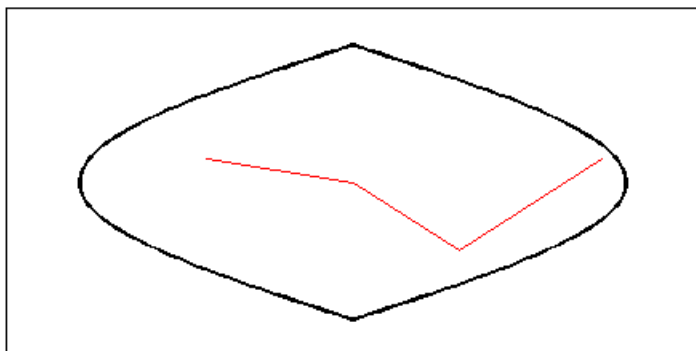
`h = linem(lat,lon,z)` displays a line object in three dimensions, where `z` is the same size as `lat` and `lon` and contains the desired altitude data. `z` is independent of `AngleUnits`. If omitted, all points are assigned a *z*-value of 0 by default.

The units of `z` are arbitrary, except when using the `globe` projection. In the case of `globe`, `z` should have the same units as the radius of the earth or semimajor axis specified in the `'geoid'` (reference ellipsoid) property of the map axes. This implies that when the reference ellipsoid is a unit sphere, the units of `z` are earth radii.

`linem` is the mapping equivalent of the MATLAB `line` function. It is a low-level graphics function for displaying line objects in map projections. Ordinarily, it is not used directly. Use `plotm` or `plot3m` instead.

Examples

```
axesm sinusoid; framem
linem([15; 0; -45; 15],[-100; 0; 100; 170],'r-')
```

More About

Line Style, Marker, and Color

Specify the line style, marker, and color as a character vector or string containing symbols. The symbols can appear in any order. You do not need to specify all three characteristics (line style, marker, and color). For example, specify a red dashed line using '--r'.

Line Style	Description	Resulting Line
'-'	Solid line	————
'--'	Dashed line	- — — — —
':'	Dotted line	....
'-.'	Dash-dotted line	—·—·—·—··

Marker	Description	Resulting Marker
'o'	Circle	○
'+'	Plus sign	+
'*'	Asterisk	✳
'.'	Point	•
'x'	Cross	×
'_'	Horizontal line	—
'|'	Vertical line		
's'	Square	☐
'd'	Diamond	◇

Marker	Description	Resulting Marker
'^'	Upward-pointing triangle	△
'v'	Downward-pointing triangle	▽
'>'	Right-pointing triangle	▷
'<'	Left-pointing triangle	◁
'p'	Pentagram	☆
'h'	Hexagram	✡

Color Name	Short Name	RGB Triplet	Appearance
'red'	'r'	[1 0 0]	
'green'	'g'	[0 1 0]	
'blue'	'b'	[0 0 1]	
'cyan'	'c'	[0 1 1]	
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

See Also
line | plot3m | plotm

longitudeToIntrinsicX

Package: `map.rasterref`

Convert from longitude to intrinsic *x* coordinates

Syntax

```
xIntrinsic = longitudeToIntrinsicX(R,lon)
```

Description

`xIntrinsic = longitudeToIntrinsicX(R,lon)` returns the *x*-coordinate in the intrinsic coordinate system corresponding to longitude `lon` in the geographic coordinate system, based on the relationship defined by geographic raster R.

Examples

Find Intrinsic *x*-Coordinate from Longitude

Import a GeoTIFF image [1] on page 1-0 of the Gulf of Mexico as an array and a `GeographicCellsReference` object. Find the intrinsic *x*-coordinate corresponding to a longitude of `-90.0226` degrees.

```
[Z,R] = readgeoraster('katrina.tif');
xIntrinsic = longitudeToIntrinsicX(R,-90.0226)
```

```
xIntrinsic = 341.0619
```

The result means that coordinates with a longitude of `-90.0226` degrees fall between columns 341 and 342 of the posting points.

[1] The image used in this example is derived from imagery obtained from the NASA Goddard Scientific Visualization Studio (SVS) Image Server.

Input Arguments

R — Geographic raster
`GeographicCellsReference` or `GeographicPostingsReference` object

Geographic raster, specified as a `GeographicCellsReference` or `GeographicPostingsReference` object.

`lon` — Longitude coordinates
numeric array

Longitude coordinates, specified as a numeric array. `lon` coordinates can be outside the bounds of the raster R.

Data Types: `single` | `double`

Output Arguments

xIntrinsic — *x*-coordinates in intrinsic coordinate system
numeric array

x-coordinates in intrinsic coordinate system, returned as a numeric array. `xIntrinsic` is the same size as `lat`.

When `lon`(*k*) is outside the bounds of raster R, `xIntrinsic`(*k*) is extrapolated in the intrinsic coordinate system. Elements of `lon` with value `NaN` map to `NaN` in `xIntrinsic`.

Data Types: `double`

See Also
`geographicToIntrinsic` | `latitudeToIntrinsicY` | `intrinsicXToLongitude`

Introduced in R2013b

lookAtSpheroid

Line of sight intersection with oblate spheroid

Syntax

```
[lat,lon,slantrange] = lookAtSpheroid(lat0,lon0,h0,az,tilt,spheroid)
```

Description

`[lat,lon,slantrange] = lookAtSpheroid(lat0,lon0,h0,az,tilt,spheroid)` computes the latitude and longitude (`lat` and `lon`) of the first point of intersection of the line-of-sight from a viewpoint in space with the surface of an oblate spheroid. If the line-of-sight does not intersect with the spheroid, `lat` and `lon` contain `NaN`s.

`lat0` and `lon0` are the geodetic coordinates of the viewpoint on the reference spheroid, `spheroid`. `h0` specifies the geodetic height of the viewpoint in space above the spheroid. The `az` and `tilt` arguments specify the direction of the view (line-of-sight) as the azimuth angle, measured clockwise from North, and a tilt angle. The following figure illustrates these measurements.
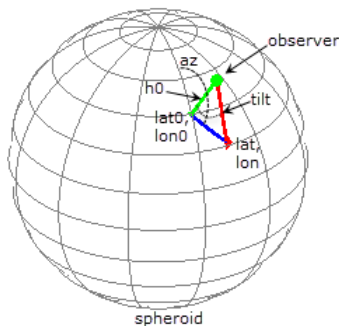
The optional `slantrange` output argument returns the (3-D Euclidean) distance from the viewpoint to the intersection. All angles are in degrees.

Examples

Calculate Intersection with Spheroid of View from Geostationary Orbit

Create a reference spheroid. Specify the length unit as kilometers.

```
spheroid = wgs84Ellipsoid('km');
```

Define location of view in space. The units for `h0` match the units of the spheroid (kilometers).

```
lat0 = 0;
lon0 = -100;
h0 = 35786;
```

Define the view from space in terms of the azimuth angle and tilt.

```
az = 45;
tilt = 6;
```

Calculate the point on the spheroid at which the view first intersects with the spheroid. The example also returns the distance in kilometers between the viewpoint in space and the first point of intersection with the spheroid.

```
[lat,lon,slantrange] = lookAtSpheroid(lat0,lon0,h0,az,tilt,spheroid)

lat = 25.7991

lon = -71.3039

slantrange = 3.7328e+04
```

Input Arguments

lat0 — Geodetic latitude of the viewpoint on the spheroid
scalar | vector | matrix | N-D array

Geodetic latitude of the viewpoint on the spheroid, specified as a scalar value, vector, matrix, or N-D array.

Data Types: `single` | `double`

lon0 — Geodetic longitude of the viewpoint on the spheroid
scalar | vector | matrix | N-D array

Geodetic longitude of the viewpoint on the spheroid, specified as a scalar value, vector, matrix, or N-D array.

Data Types: `single` | `double`

h0 — Height of the viewpoint in space above the spheroid
scalar | vector | matrix | N-D array

Height of the viewpoint in space above the spheroid, specified as a scalar value, vector, matrix, or N-D array. `h0` must be in units that match the spheroid input.

Data Types: `single` | `double`

az — Azimuth angle of view from space
scalar | vector | matrix | N-D array

Azimuth angle of view from space, specified as a scalar value, vector, matrix, or N-D array. Measured in degrees, clockwise from north.

Data Types: `single` | `double`

tilt — Tilt angle of view from space
scalar | vector | matrix | N-D array

Tilt angle of view from space, specified as scalar value, vector, matrix, or N-D array. Measured in degrees, relative to a vector pointing downward toward the nadir point. The nadir point is the point on the spheroid directly below the viewpoint, with geodetic coordinates (lat0, lon0, 0). When the tilt is zero (0), the line-of-sight is directed at the nadir point itself. `Tilt` can be in the range `[0 180]` but

for large angles and all angles greater than or equal to 90 degrees, there is no intersection with the spheroid.

Data Types: `single` | `double`

spheroid — Reference spheroid
referenceEllipsoid | oblateSpheroid | referenceSphere

Reference spheroid, specified as a `referenceEllipsoid`, `oblateSpheroid`, or `referenceSphere` object. Use the constructor for one of these three classes, or the `wgs84Ellipsoid` function, to construct a spheroid object. You cannot directly pass in the name of the reference spheroid. Instead, pass the name to `referenceEllipsoid` or `referenceSphere` and use the resulting object.

Output Arguments

`lat` — Latitude of the first point of intersection with the spheroid
scalar | vector | matrix | N-D array

Latitude of the first point of intersection with the spheroid, returned as a scalar value, vector, matrix, or N-D array. If the line-of-sight does not intersect with the spheroid, `lat` contains NaNs.

`lon` — Longitude of the first point of intersection with the spheroid
scalar | vector | matrix | N-D array

Longitude of the first point of intersection with the spheroid, returned as a scalar value, vector, matrix, or N-D array. If the line-of-sight does not intersect with the spheroid, `lon` contains NaNs.

`slantrange` — Distance from the viewpoint to the first intersection with the spheroid
scalar | vector | matrix | N-D array

Distance from the viewpoint to the intersection with the spheroid, returned as a scalar value, vector, matrix, or N-D array. Units match the `LengthUnit` property of the input spheroid object.

See Also
`geodetic2aer`

Introduced in R2016b

los2

Line-of-sight visibility between two points in terrain

Syntax

```
vis = los2(Z,R,lat1,lon1,lat2,lon2)
vis = los2(Z,R,lat1,lon1,lat2,lon2,alt1)
vis = los2(Z,R,lat1,lon1,lat2,lon2,alt1,alt2)
vis = los2(Z,R,lat1,lon1,lat2,lon2,alt1,alt2,alt1opt)
vis = los2(Z,R,lat1,lon1,lat2,lon2,alt1,alt2,alt1opt,alt2opt)
vis = los2(Z,R,lat1,lon1,lat2,lon2,alt1,alt2,alt1opt, ...
  alt2opt,actualradius)
vis = los2(Z,R,lat1,lon1,lat2,lon2,alt1,alt2,alt1opt, ...
  alt2opt,actualradius,effectiveradius)
[vis,visprofile,dist,H,lattrk,lontrk] = los2(...)
los2(...)
```

Description

`los2` computes the mutual visibility between two points on a displayed digital elevation map. `los2` uses the current object if it is a regular data grid, or the first regular data grid found on the current axes. The grid's `zdata` is used for the profile. The color data is used in the absence of data in `z`. The two points are selected by clicking on the map. The result is displayed in a new figure. Markers indicate visible and obscured points along the profile. The profile is shown in a Cartesian coordinate system with the origin at the observer's location. The displayed `z` coordinate accounts for the elevation of the terrain and the curvature of the body.

`vis = los2(Z,R,lat1,lon1,lat2,lon2)` computes the mutual visibility between pairs of points on a digital elevation map. The elevations are provided as a regular data grid Z containing elevations in units of meters. The two points are provided as vectors of latitudes and longitudes in units of degrees. The resulting logical variable `vis` is equal to one when the two points are visible to each other, and zero when the line of sight is obscured by terrain. If any of the input arguments are empty, `los2` attempts to gather the data from the current axes. With one or more output arguments, no figures are created and only the data is returned.

R can be a geographic raster reference object, a referencing vector, or a referencing matrix. If R is a geographic raster reference object, its `RasterSize` property must be consistent with `size(Z)`.

If R is a referencing vector, it must be a 1-by-3 with elements:

`[cells/degree northern_latitude_limit western_longitude_limit]`

If R is a referencing matrix, it must be 3-by-2 and transform raster row and column indices to or from geographic coordinates according to:

`[lon lat] = [row col 1] * R`

If R is a referencing matrix, it must define a (non-rotational, non-skewed) relationship in which each column of the data grid falls along a meridian and each row falls along a parallel. Nearest-neighbor interpolation is used by default. NaN is returned for points outside the grid limits or for which `lat` or `lon` contain NaN. All angles are in units of degrees.

vis = los2(Z,R,lat1,lon1,lat2,lon2,alt1) places the first point at the specified altitude in meters above the surface (on a tower, for instance). This is equivalent to putting the point on a tower. If omitted, point 1 is assumed to be on the surface. alt1 may be either a scalar or a vector with the same length as lat1, lon1, lat2, and lon2.

vis = los2(Z,R,lat1,lon1,lat2,lon2,alt1,alt2) places both points at a specified altitudes in meters above the surface. alt2 may be either a scalar or a vector with the same length as lat1, lon1, lat2, and lon2. If alt2 is omitted, point 2 is assumed to be on the surface.

vis = los2(Z,R,lat1,lon1,lat2,lon2,alt1,alt2,alt1opt) controls the interpretation of alt1 as either a relative altitude (alt1opt equals 'AGL', the default) or an absolute altitude (alt1opt equals 'MSL'). If the altitude option is 'AGL', alt1 is interpreted as the altitude of point 1 in meters above the terrain ("above ground level"). If alt1opt is 'MSL', alt1 is interpreted as altitude above zero ("mean sea level").

vis = los2(Z,R,lat1,lon1,lat2,lon2,alt1,alt2,alt1opt,alt2opt) controls the interpretation ALT2.

vis = los2(Z,R,lat1,lon1,lat2,lon2,alt1,alt2,alt1opt, ...
   alt2opt,actualradius) does the visibility calculation on a sphere with the specified radius. If omitted, the radius of the earth in meters is assumed. The altitudes, elevations and the radius should be in the same units. This calling form is most useful for computations on bodies other than the earth.

vis = los2(Z,R,lat1,lon1,lat2,lon2,alt1,alt2,alt1opt, ...
   alt2opt,actualradius,effectiveradius) assumes a larger radius for propagation of the line of sight. This can account for the curvature of the signal path due to refraction in the atmosphere. For example, radio propagation in the atmosphere is commonly treated as straight line propagation on a sphere with 4/3 the radius of the earth. In that case the last two arguments would be R_e and 4/3*R_e, where R_e is the radius of the earth. Use Inf as the effective radius for flat earth visibility calculations. The altitudes, elevations and radii should be in the same units.

[vis,visprofile,dist,H,lattrk,lontrk] = los2(...), for scalar inputs (lat1, lon1, etc.), returns vectors of points along the path between the two points. visprofile is a logical vector containing true (logical(1)) where the intermediate points are visible and false (logical(0)) otherwise. dist is the distance along the path (in meters or the units of the radius). H contains the terrain profile relative to the vertical datum along the path. lattrk and lontrk are the latitudes and longitudes of the points along the path. For vector inputs los2 returns visprofile, dist, H, lattrk, and lontrk as cell arrays, with one cell per element of lat1,lon1, etc.

los2(...), with no output arguments, displays the visibility profile between the two points in a new figure.

Examples

Determine Line-of-Sight

Determine the line-of-sight for two points. Display the visibility profile in 2-D and 3-D.

Import terrain data for an area around South Boulder Peak in Colorado. Prepare the data for analysis by specifying the output type as 'double'.

```
[Z,R] = readgeoraster("n39_w106_3arc_v2.dt1","OutputType","double");
```

Specify the coordinates of South Boulder Peak and a nearby point. Determine whether the nearby point is visible from a location 100 meters above the peak by assigning the result of `los2` to a variable.

```
latPeak = 39.95384;
lonPeak = -105.29916;
latNearby = 39.96955;
lonNearby = -105.29867;
vis = los2(Z,R,latPeak,lonPeak,latNearby,lonNearby,100)

vis = logical
   0
```

The value of `vis` is 0, which means the point is not visible from the peak.

To display the visibility profile in 2-D, call `los2` without an output argument.

```
los2(Z,R,latPeak,lonPeak,latNearby,lonNearby,100)
```
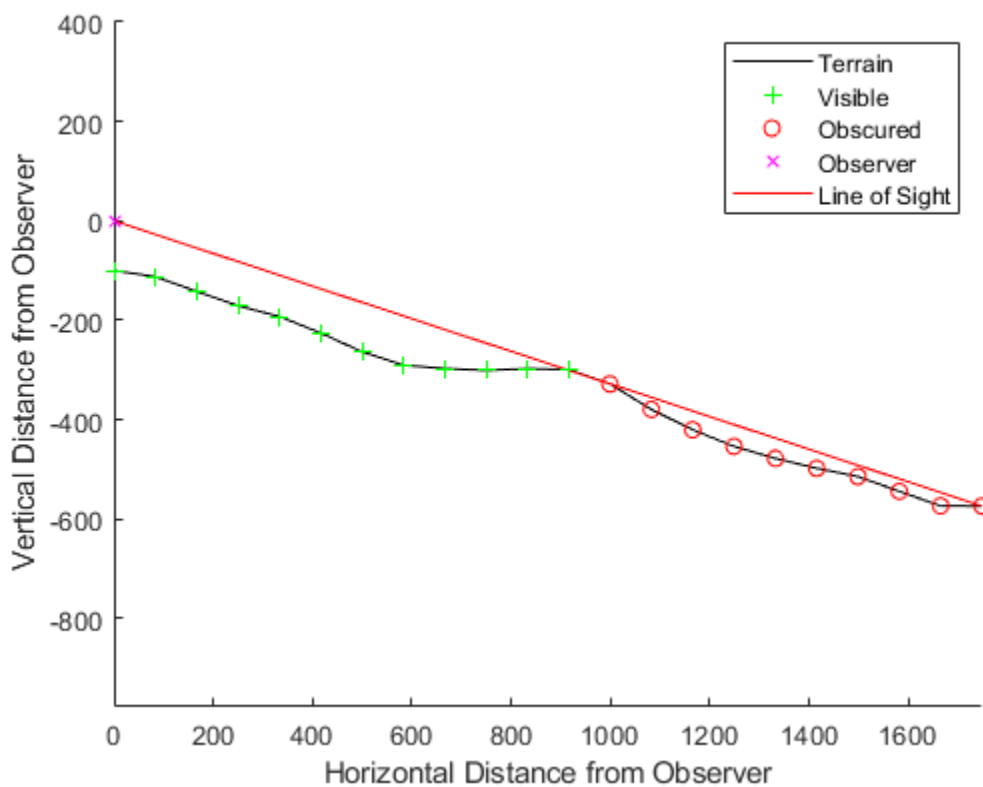

To display the visibility profile in 3-D, return visibility information about points between the coordinates you specified. The outputs `lattrk` and `lontrk` contain the coordinates of the points. The output `visprofile` contains logical values indicating whether the points are visible.

```
[~,visprofile,~,~,lattrk,lontrk] = los2(Z,R,latPeak,lonPeak,latNearby,lonNearby,100);
```

Create a geographic globe that includes custom terrain for the area around South Boulder Peak.

```
addCustomTerrain("southboulder","n39_w106_3arc_v2.dt1")
uif = uifigure;
g = geoglobe(uif,"Terrain","southboulder");
```

Plot the line-of-sight between the coordinates you specified. Use a magenta line to indicate the point is not visible from above the peak.

```
hold(g,"on")
geoplot3(g,[latPeak latNearby],[lonPeak lonNearby],[100 0],"m","LineWidth",2, ...
    "HeightReference","terrain")
```

Plot the intermediate points. Use magenta markers for points that are not visible and green markers for points that are visible. Plot each value 10 meters above the terrain so the markers are not obscured by the terrain.

```
geoplot3(g,lattrk(~visprofile),lontrk(~visprofile),10, ...
    "mo","MarkerSize",2,"LineWidth",2,"HeightReference","terrain")
geoplot3(g,lattrk(visprofile),lontrk(visprofile),10,"go", ...
    "MarkerSize",2,"LineWidth",2,"HeightReference","terrain")
```

Adjust the view by setting the camera position, pitch, roll, and heading.

```
campos(g,39.9519,-105.3039,2.9424e+03)
campitch(g,-31.7850)
camroll(g,0.3603)
camheading(g,36.5618)
```
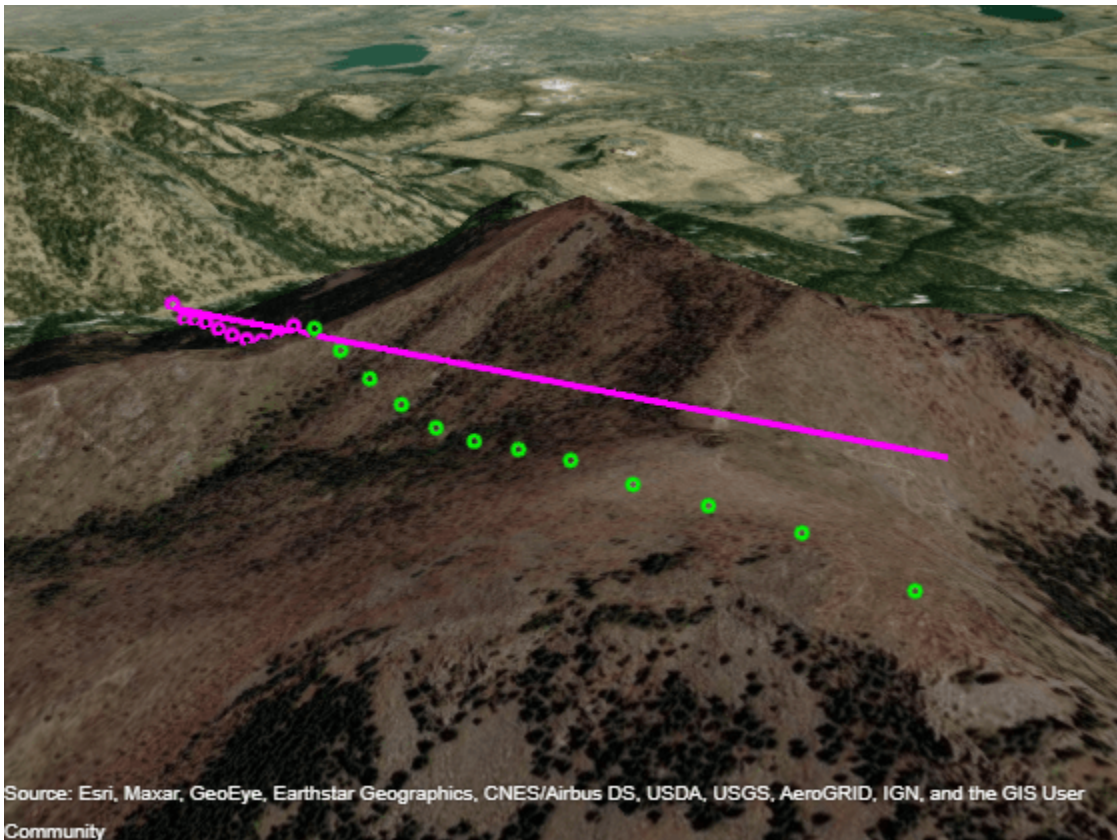

Close the figure containing the geographic globe and remove the custom terrain.

```
close(uif)
removeCustomTerrain("southboulder")
```

Find Elevation Angle of Point 90 Degrees from Observer

Find the elevation angle of a point 90 degrees from an observer assuming that the observer and the target are both 1000 km above the Earth.

```
lat1 = 0;
lon1 = 0;
alt1 = 1000*1000;
lat2 = 0;
lon2 = 90;
alt2 = 1000*1000;
[az, elev, r] = geodetic2aer(lat2,lon2,alt2,lat1,lon1,alt1,referenceEllipsoid('grs 80'))

az = 90

elev = -45

r = 1.0434e+07
```

Visually check the result using the `los2` line of sight function. Construct a data grid of zeros to represent the Earth's surface. The `los2` function with no output arguments creates a figure displaying the geometry.

```
Z = zeros(181,361);
R = georefpostings([-90 90],[-180 180], size(Z))

R =
  GeographicPostingsReference with properties:

              LatitudeLimits: [-90 90]
             LongitudeLimits: [-180 180]
                  RasterSize: [181 361]
        RasterInterpretation: 'postings'
            ColumnsStartFrom: 'south'
               RowsStartFrom: 'west'
      SampleSpacingInLatitude: 1
     SampleSpacingInLongitude: 1
       RasterExtentInLatitude: 180
      RasterExtentInLongitude: 360
            XIntrinsicLimits: [1 361]
            YIntrinsicLimits: [1 181]
        CoordinateSystemType: 'geographic'
               GeographicCRS: []
                   AngleUnit: 'degree'
```

```
los2(Z,R,lat1,lon1,lat2,lon2,alt1,alt1);
```
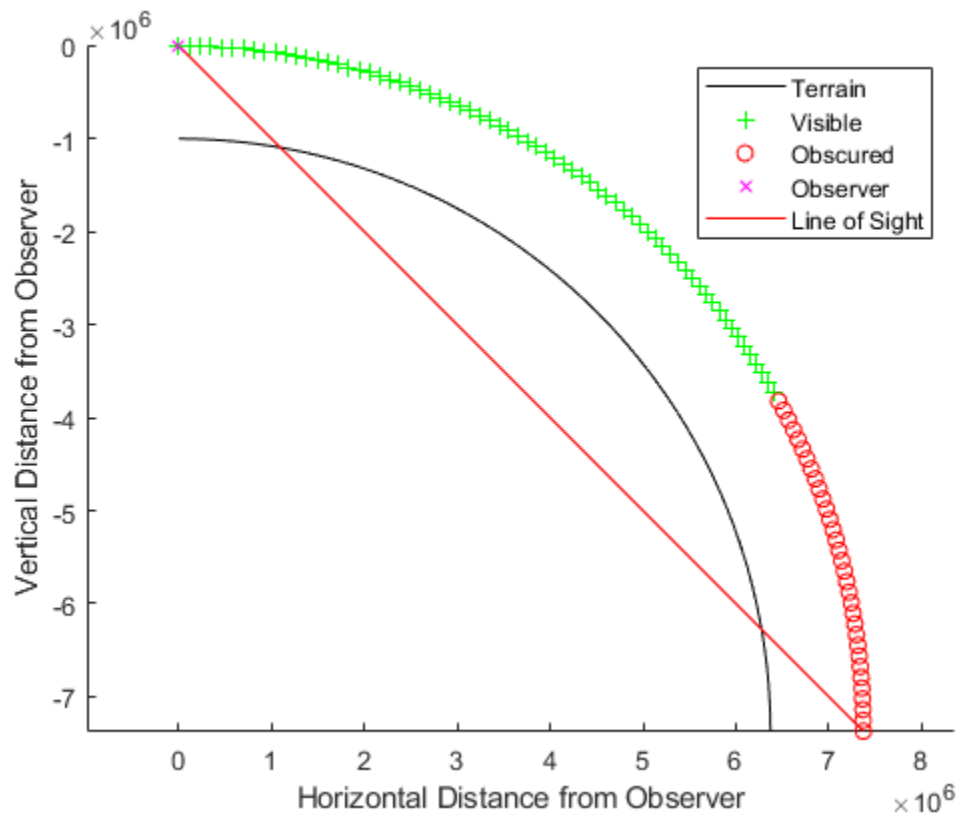
See Also

`viewshed` | `mapprofile`

Introduced before R2006a

ltln2val

(To be removed) Extract data grid values for specified locations

---

Note `ltln2val` will be removed in a future release. Use the `geointerp` function instead. For more information, see "Compatibility Considerations".

Syntax

```
val = ltln2val(Z, R, lat, lon)
val = ltln2val(Z, R, lat, lon, method)
```

Description

`val = ltln2val(Z, R, lat, lon)` interpolates a regular data grid Z with referencing vector R at the points specified by vectors of latitude and longitude, `lat` and `lon`. R can be a geographic raster reference object, a referencing vector, or a referencing matrix.

If R is a geographic raster reference object, its `RasterSize` property must be consistent with `size(Z)`.

If R is a referencing vector, it must be a 1-by-3 with elements:

`[cells/degree northern_latitude_limit western_longitude_limit]`

If R is a referencing matrix, it must be 3-by-2 and transform raster row and column indices to or from geographic coordinates according to:

`[lon lat] = [row col 1] * R`

If R is a referencing matrix, it must define a (non-rotational, non-skewed) relationship in which each column of the data grid falls along a meridian and each row falls along a parallel. Nearest-neighbor interpolation is used by default. NaN is returned for points outside the grid limits or for which `lat` or `lon` contain NaN. All angles are in units of degrees.

`val = ltln2val(Z, R, lat, lon, method)` where `method` specifies the type of interpolation: `'bilinear'` for linear interpolation, `'bicubic'` for cubic interpolation, or `'nearest'` for nearest neighbor interpolation.

Examples

Load elevation raster data and a geographic cells reference object. Then, find the elevation in meters associated with Milan, Bern, and Prague.

```
load topo60c
lat = [45.45 46.95 50.1];
lon = [9.2 7.4 14.45];
elevations = ltln2val(topo60c,topo60cR,lat,lon)
```

```
elevations =

        313        1660        297
```

Compatibility Considerations

ltln2val will be removed
Not recommended starting in R2020b

Some functions that accept referencing vectors or referencing matrices as input will be removed, including the `ltln2val` function. Use a geographic reference object and the `geointerp` function instead. Reference objects have several advantages over referencing vectors and matrices.

- Unlike referencing vectors and matrices, reference objects have properties that document the size of the associated raster, its geographic limits, and the direction of its rows and columns. For more information about reference object properties, see the `GeographicCellsReference` and `GeographicPostingsReference` objects.
- You can manipulate the limits of geographic rasters associated with reference objects using the `geocrop` function.
- You can manipulate the size and resolution of geographic rasters associated with reference objects using the `georesize` function.

To update your code, first create a reference object for either a raster of cells using the `georefcells` function or a raster of regularly posted samples using the `georefpostings` function. Alternatively, convert from a referencing vector or a referencing matrix to a reference object using the `refvecToGeoRasterReference` or `refmatToGeoRasterReference` function, respectively.

Then, replace uses of the `ltln2val` function with the `geointerp` function according to these patterns. Note that the default method of interpolation for the `geointerp` function is `'linear'` instead of `'nearest'`. In addition, replace the interpolation methods of `'bilinear'` and `'bicubic'` for the `ltln2val` function with `'linear'` and `'cubic'` for the `geointerp` function.

Will Be Removed	Recommended
`val = ltln2val(Z,R,lat,lon);`	`val = geointerp(Z,R,lat,lon,'nearest');`
`val = ltln2val(Z,R,lat,lon,method);`	`val = geointerp(Z,R,lat,lon,method);`

See Also
findm | imbedm | geointerp

lv2ecef

Convert local vertical to geocentric (ECEF) coordinates

Note `lv2ecef` will be removed in a future release. Use `enu2ecef` instead. In `enu2ecef`, the latitude and longitude of the local origin are in degrees by default, so the optional `angleUnit` input should be included, with the value `'radians'`.

Syntax

`[x,y,z] = lv2ecef(xl,yl,zl,phi0,lambda0,h0,ellipsoid)`

Description

`[x,y,z] = lv2ecef(xl,yl,zl,phi0,lambda0,h0,ellipsoid)` converts arrays `xl`, `yl`, and `zl` in the local vertical coordinate system to arrays `x`, `y`, and `z` in the geocentric coordinate system. The origin of the local vertical system is at geodetic latitude `phi0`, geodetic longitude `lambda0`, and ellipsoidal height `h0`. The arrays `xl`, `yl`, and `zl` may have any shape, as long as they are all the same size. They are measured in the same length units as the semimajor axis. `phi0` and `lambda0` are scalars measured in radians; `h0` is a scalar with the same length units as the semimajor axis; and `ellipsoid` is a `referenceEllipsoid` (`oblateSpheroid`) object, a `referenceSphere` object, or a vector of the form `[semimajor axis, eccentricity]`). The coordinates `x`, `y`, and `z` also have the same units as the semimajor axis.

More About

Local Vertical System

In the local vertical Cartesian system defined by `phi0`, `lambda0`, `h0`, and `ellipsoid`, the `xl` axis is parallel to the plane tangent to the ellipsoid at (`phi0`,`lambda0`) and points due east. The `yl` axis is parallel to the same plane and points due north. The `zl` axis is normal to the ellipsoid at (`phi0`,`lambda0`) and points outward into space. The local vertical system is sometimes referred to as East-North-Up or ENU.

Geocentric System

The geocentric Cartesian coordinate system, also known as Earth-Centered, Earth-Fixed (ECEF), is fixed with respect to the Earth, with its origin at the center of the spheroid and its positive X-, Y-, and Z axes intersecting the surface at the following points:

	Latitude	Longitude	Notes
X-axis	0	0	Equator at the Prime Meridian
Y-axis	0	90	Equator at 90-degrees East
Z-axis	90	0	North Pole

See Also
ecef2enu

Introduced before R2006a

majaxis

Semimajor axis of ellipse

Note Support for nonscalar input, including the syntax `a = majaxis(vec)`, will be removed in a future release.

Syntax

```
a = majaxis(semiminor,e)
a = majaxis(vec)
```

Description

`a = majaxis(semiminor,e)` computes the semimajor axis of an ellipse (or ellipsoid of revolution) given the semiminor axis and eccentricity. The input data can be scalar or matrices of equal dimensions.

`a = majaxis(vec)` assumes a 2 element vector (`vec`) is supplied, where `vec = [semiminor, e]`.

See Also
axes2ecc | flat2ecc | minaxis | n2ecc

Introduced before R2006a

makeattribspec

Create attribute specification

Syntax

```
attribspec = makeattribspec(S)
```

Description

`attribspec = makeattribspec(S)` creates an attribute specification for use with the `kmlwrite`, `wmmarker`, `wmline`, and `wmpolygon` functions based on the vector geographic features stored in `S`.

The `kmlwrite` function creates an HTML table from attribute fields. The first column of the table includes attribute labels and the second column includes attribute values as text. Use an attribute specification with the `kmlwrite` function and `Description` name-value argument to specify which attribute fields to include in the HTML table and the format of the attribute values.

When you plot features using `wmmarker`, `wmline`, or `wmpolygon`, you can view feature balloons with attribute information by clicking on features. Use an attribute specification with `wmmarker`, `wmline`, or `wmpolygon` and the `Description` name-value argument to specify which attribute fields to include in the feature balloons and the format of the attribute values.

Examples

Create Attribute Specification

Import a shapefile representing tsunami (tidal wave) events reported over several decades, tagged geographically by source location, as a geospatial table. Create an attribute specification from the table.

```
GT = readgeotable("tsunamis.shp","CoordinateSystemType","geographic");
attribspec = makeattribspec(GT)

attribspec = struct with fields:
          Year: [1x1 struct]
         Month: [1x1 struct]
           Day: [1x1 struct]
          Hour: [1x1 struct]
        Minute: [1x1 struct]
        Second: [1x1 struct]
      Val_Code: [1x1 struct]
      Validity: [1x1 struct]
    Cause_Code: [1x1 struct]
         Cause: [1x1 struct]
        Eq_Mag: [1x1 struct]
       Country: [1x1 struct]
      Location: [1x1 struct]
    Max_Height: [1x1 struct]
      Iida_Mag: [1x1 struct]
     Intensity: [1x1 struct]
```

```
      Num_Deaths: [1x1 struct]
      Desc_Deaths: [1x1 struct]
```

Remove all attributes from the specification, except for the attributes describing the maximum height, cause, year, location, and country.

```
desiredAttributes = {'Max_Height','Cause','Year','Location','Country'};
allAttributes = fieldnames(attribspec);
attributes = setdiff(allAttributes,desiredAttributes);
attribspec = rmfield(attribspec,attributes)
```

```
attribspec = struct with fields:
          Year: [1x1 struct]
         Cause: [1x1 struct]
       Country: [1x1 struct]
      Location: [1x1 struct]
    Max_Height: [1x1 struct]
```

Rename the `Max_Height` attribute to `Maximum Height` and highlight each attribute label in bold font.

```
attribspec.Max_Height.AttributeLabel = "<b>Maximum Height</b>";
attribspec.Cause.AttributeLabel = "<b>Cause</b>";
attribspec.Year.AttributeLabel = "<b>Year</b>";
attribspec.Location.AttributeLabel = "<b>Location</b>";
attribspec.Country.AttributeLabel = "<b>Country</b>";
```

Add `Meters` to the format of the maximum height attribute. Set the format of the year attribute to include no decimal places.

```
attribspec.Max_Height.Format = "%.1f Meters";
attribspec.Year.Format = "%.0f";
```

Export the selected attributes and tsunami source locations to a KML file.

```
kmlwrite("tsunamis.kml",GT,"Description",attribspec,"Name",GT.Location)
```

Display Marker Overlay Using Attribute Specification

Import a shapefile representing tsunami (tidal wave) events reported over several decades, tagged geographically by source location.

```
GT = readgeotable("tsunamis.shp","CoordinateSystemType","geographic");
```

Create an attribute specification for use in the feature balloons. Modify the specification so that it defines a table of values, including year, cause, country, location, and maximum height. The attribute specification defines the format of the expected value for each field.

```
attribspec = makeattribspec(GT);

desiredAttributes = ...
      {'Max_Height','Cause','Year','Location','Country'};
allAttributes = fieldnames(attribspec);
```

```
attributes = setdiff(allAttributes,desiredAttributes);
attribspec = rmfield(attribspec,attributes);
attribspec.Max_Height.AttributeLabel = "<b>Maximum Height</b>";
attribspec.Max_Height.Format = "%.1f Meters";
attribspec.Cause.AttributeLabel = "<b>Cause</b>";
attribspec.Year.AttributeLabel = "<b>Year</b>";
attribspec.Year.Format = "%.0f";
attribspec.Location.AttributeLabel = "<b>Location</b>";
attribspec.Country.AttributeLabel = "<b>Country</b>";
```

Create a web map with a base layer containing ocean details. Add the marker overlay. Display the table data you specified in the attribute specification by selecting a marker.

```
webmap("ocean basemap");
wmmarker(GT,"Description",attribspec,...
        "OverlayName","Tsunami Events")
wmzoom(2)
```


Input Arguments

S — Vector geographic features
geospatial table | `geopoint` vector | `geoshape` vector | `geostruct` vector

Vector geographic features, specified as one of these:

- A geospatial table containing `geopointshape`, `geolineshape`, or `geopolyshape` objects. The `makeattribspec` function does not support geospatial tables containing more than one type of shape object. For more information about geospatial tables, see "Create Geospatial Tables".

- A `geopoint` vector.
- A `geoshape` vector with no dynamic vertex properties.
- A geographic structure array (`geostruct`) with `Lat` and `Lon` fields.

Output Arguments

`attribspec` — Attribute specification
structure array

Attribute specification, returned as a structure array with two levels. The top level contains a field for each attribute in `S`. Each field in the top level contains a scalar structure with two fields.

Field	Description
AttributeLabel	Attribute label for the `kmlwrite` function to use in the first column of the HTML table, returned as a character vector. By default, the label is identical to the name of the corresponding attribute field.
Format	Format for the `kmlwrite` function to use when converting the attribute value to text, returned as a character vector. For more information about formatting data, see the `sprintf` function.

You can change the structure array before using it with the `kmlwrite` function. For example, you can remove attributes from the structure or change the `AttributeLabel` field to include spaces.

See Also

Functions
kmlwrite | wmmarker | wmline | wmpolygon | makedbfspec | shapewrite

Introduced before R2006a

makedbfspec

Create DBF specification

Syntax

```
dbfspec = makedbfspec(S)
```

Description

`dbfspec = makedbfspec(S)` creates a DBF specification for use with the `shapewrite` function based on the vector geographic features stored in `S`.

The `shapewrite` function creates a main file (`.shp`), an index file (`.shx`), and an attribute file (`.dbf`). Use a DBF specification with the `shapewrite` function and `DbfSpec` name-value argument to specify which attribute fields to include in the DBF file, the length of each field, and the numerical precision of each field.

Examples

Create DBF Specification

Import a shapefile representing a network of road segments in Concord, MA, as a geospatial table. Create a DBF specification from the table.

```
GT = readgeotable("concord_roads.shp");
dbfspec = makedbfspec(GT)

dbfspec = struct with fields:
    STREETNAME: [1x1 struct]
     RT_NUMBER: [1x1 struct]
         CLASS: [1x1 struct]
    ADMIN_TYPE: [1x1 struct]
        LENGTH: [1x1 struct]
```

Remove the `ADMIN_TYPE` field from the specification, rename the `STREETNAME` field to `Street Name`, and reduce the number of decimal places used to store road lengths.

```
dbfspec = rmfield(dbfspec,"ADMIN_TYPE");
dbfspec.STREETNAME.FieldName = "Street Name";
dbfspec.LENGTH.FieldDecimalCount = 1;
```

Export the road network to a new shapefile.

```
shapewrite(GT,"concord_roads_new.shp","DbfSpec",dbfspec)
```

Input Arguments

S — Vector geographic features
geospatial table | `mappoint` vector | `mapshape` vector | `mapstruct` vector | `geopoint` vector | `geoshape` vector | `geostruct` vector

Vector geographic features, specified as one of these:

- A geospatial table. The `makedbfspec` function does not support geospatial tables containing more than one type of shape object. For more information about geospatial tables, see "Create Geospatial Tables".

- A `mappoint` vector.

- A `mapshape` vector.

- A map structure array (`mapstruct`) with X and Y coordinate fields.

- A `geopoint` vector.

- A `geoshape` vector.

- A geographic structure array (`geostruct`) with Lat and Lon fields.

Output Arguments

dbfspec — DBF specification
structure array

DBF specification, returned as a structure array with two levels. The top level contains a field for each attribute in S. Each field in the top level contains a scalar structure with four fields.

Field	Description
FieldName	Field name for the `shapewrite` function to use in the DBF file. By default, the field name is identical to the name of the corresponding attribute.
FieldType	Field type for the `shapewrite` function to use in the DBF file, specified as `'N'` (numeric) or `'C'` (character).
FieldLength	Number of bytes each instance of the field occupies in the file.
FieldDecimalCount	Number of digits to the right of the decimal place. Specify the number of digits as `0` for integer-valued fields and character fields. The default value for noninteger numeric fields is `6`.

You can change the structure array before using it with the `shapewrite` function. For example, you can remove attributes from the structure or change the `FieldName` field to include spaces.

See Also

Functions
shapeinfo | shapewrite | readgeotable

Introduced before R2006a

makemapped

Convert ordinary graphics object to mapped object

Note makemapped will be removed in a future release.

Syntax

```
makemapped(h)
```

Description

makemapped(h) modifies the graphic object(s) associated with h such that upon subsequent modification of map axes properties, they are automatically reprojected appropriately. The object's coordinates are not changed by makemapped, but will change should you modify the map projection. h can be a handle, vector of handles, or any name recognized by handlem. The objects are then considered to be geographic data. You should first trim objects extending outside the map frame to the map frame using trimcart.

Examples

```
axesm('miller','geoid',[25 0])
framem
plot(humps,'b+-')

h = plot(humps,'r+-');
trimcart(h)
makemapped(h)

setm(gca,'MapProjection','sinusoid')
```
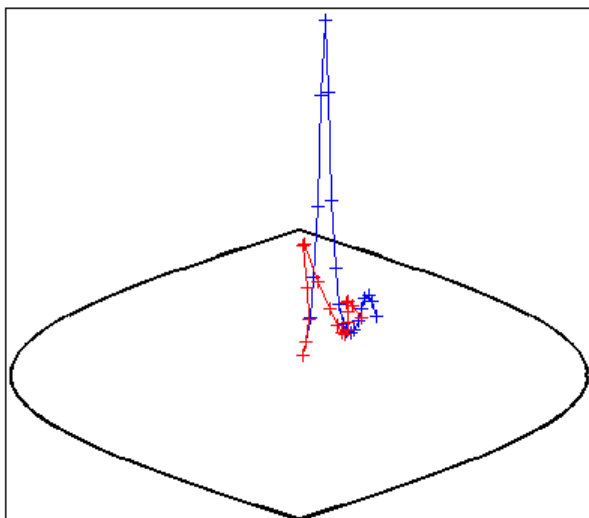
Tips

Objects should first be trimmed to the map frame using `trimcart`. This avoids problems in taking inverse map projections with out-of-range data.

See Also

makerefmat

(To be removed) Construct affine spatial-referencing matrix

Note makerefmat will be removed in a future release. Create a raster reference object using the georefcells, georefpostings, georasterref, maprefcells, maprefpostings, or maprasterref function instead. For more information, see "Compatibility Considerations".

Syntax

```
R = makerefmat(x11, y11, dx, dy)
R = makerefmat(lon11, lat11, dlon, dlat)
R = makerefmat(param1, val1, param2, val2, ...)
```

Description

R = makerefmat(x11, y11, dx, dy), with scalars dx and dy, constructs a referencing matrix that aligns image or data grid rows to map *x* and columns to map *y*. Scalars x11 and y11 specify the map location of the center of the first (1,1) pixel in the image or the first element of the data grid.

dx is the difference in *x* (or longitude) between pixels in successive columns, and dy is the difference in *y* (or latitude) between pixels in successive rows.

Pixels cover squares on the map when abs(dx) = abs(dy). To achieve the most typical kind of alignment, where *x* increases from column to column and *y* decreases from row to row, make dx positive and dy negative. In order to specify such an alignment along with square pixels, make dx positive and make dy equal to -dx:

```
 R = makerefmat(x11, y11, dx, -dx)
```

R = makerefmat(x11, y11, dx, dy), with two-element vectors dx and dy, constructs the most general possible kind of referencing matrix.

In this general case, each pixel can become a parallelogram on the map, with neither edge necessarily aligned to map *x* or *y*. The vector [dx(1) dy(1)] is the difference in map location between a pixel in one row and its neighbor in the preceding row. Likewise, [dx(2) dy(2)] is the difference in map location between a pixel in one column and its neighbor in the preceding column.

To specify pixels that are rectangular or square (but possibly rotated), choose dx and dy such that prod(dx) + prod(dy) = 0. To specify square (but possibly rotated) pixels, choose dx and dy such that the 2-by-2 matrix [dx(:) dy(:)] is a scalar multiple of an orthogonal matrix (that is, its two eigenvalues are real, nonzero, and equal in absolute value). This amounts to either rotation, a mirror image, or a combination of both. Note that for scalars dx and dy,

```
R = makerefmat(x11, y11, [0 dx], [dy 0])
```

is equivalent to

```
R = makerefmat(x11, y11, dx, dy)
```

R = makerefmat(lon11, lat11, dlon, dlat), with longitude preceding latitude, constructs a referencing matrix for use with geographic coordinates.

R = makerefmat(param1, val1, param2, val2, ...) uses parameter name-value pairs to construct a referencing matrix for an image or raster grid that is referenced to and aligned with a geographic coordinate system. There can be no rotation or skew: each column must fall along a meridian, and each row must fall along a parallel. Each parameter name must be specified exactly as shown, including case.

Parameter Name	Data Type	Value
RasterSize	Two-element size vector [M N]	The number of rows (M) and columns (N) of the raster or image to be used with the referencing matrix. With 'RasterSize', you may also provide a size vector having more than two elements. This enables usage such as: R = makerefmat('RasterSize', ... size(RGB), ...) where RGB is M-by-N-by-3. However, in cases like this, only the first two elements of the size vector will actually be used. The higher (non-spatial) dimensions will be ignored. The default value is [1 1].
LatitudeLimits	Two-element row vector of the form: [southern_limit, northern_limit], in units of degrees.	The limits in latitude of the geographic quadrangle bounding the georeferenced raster. The default value is [0 1].
LongitudeLimits	Two-element row vector of the form: [western_limit, eastern_limit], in units of degrees.	The limits in longitude of the geographic quadrangle bounding the georeferenced raster. The elements of the 'LongitudeLimits' vector must be ascending in value. In other words, the limits must be unwrapped. The default value is [0 1].
ColumnsStartFrom	String scalar or character vector	Indicates the column direction of the raster (south-to-north vs. north-to-south) in terms of the edge from which row indexing starts. Values are 'south' or 'north' and they can be shortened, and are case-insensitive. In a typical terrain grid, row indexing starts at southern edge. In images, row indexing starts at northern edge. The default value is 'south'.
RowsStartFrom	String scalar or character vector	Indicates the row direction of the raster (west-to-east vs. east-to-west) in terms of the edge from which column indexing starts. Values are: 'west' or 'east' and they can be shortened, and are case-insensitive. Rows almost always run from west to east. The default value is 'west'.

Examples

Create a referencing matrix for an image with square, four-meter pixels and with its upper left corner (in a map coordinate system) at $x = 207000$ meters, $y = 913000$ meters. The image follows the typical orientation: x increasing from column to column and y decreasing from row to row.

```
x11 = 207002;  % Two meters east of the upper left corner
y11 = 912998;  % Two meters south of the upper left corner
dx =  4;
dy = -4;
R = makerefmat(x11, y11, dx, dy)
```

More About

Spatial Referencing Matrix

A spatial referencing matrix R ties the row and column subscripts of an image or regular data grid to 2-D map coordinates or to geographic coordinates (longitude and geodetic latitude). R is a 3-by-2 affine transformation matrix. R either transforms pixel subscripts (row, column) to/from map coordinates (x,y) according to

```
[x y] = [row col 1] * R
```

or transforms pixel subscripts to/from geographic coordinates according to

```
[lon lat] = [row col 1] * R
```

To construct a referencing matrix for use with geographic coordinates, use longitude in place of X and latitude in place of Y, as shown in the `R = makerefmat(X11, Y11, dx, dy)` syntax. This is one of the few places where longitude precedes latitude in a function call.

Compatibility Considerations

makerefmat will be removed
Not recommended starting in R2020b

Some functions that return referencing matrices will be removed, including the `makerefmat` function. Instead, create a geographic raster reference object by using the `georefcells`, `georefpostings`, or `georasterref` function, or a map raster reference object by using the `maprefcells`, `maprefpostings`, or `maprasterref` function. Reference objects have several advantages over referencing matrices.

- Unlike referencing matrices, reference objects have properties that document the size of the associated raster, its limits, and the direction of its rows and columns. For more information about reference object properties, see the `GeographicCellsReference` and `MapPostingsReference` objects.
- You can manipulate the limits of rasters associated with reference objects using the `geocrop` or `mapcrop` function.
- You can manipulate the size and resolution of rasters associated with reference objects using the `georesize` or `mapresize` function.
- Most functions that accept referencing matrices as inputs also accept reference objects.

Depending on the `makerefmat` function syntax, there are different ways to update your code.

- To specify the first data point of A and scalar values of dx and dy or dlon and dlat, use these replacement patterns. Use the geograsterref function for geographic coordinates and the maprasterref function for planar map coordinates.

Will Be Removed	Recommended
R = makerefmat(x11,y11,dx,dy);	W = [dx 0 x11; 0 dy y11]; R = maprasterref(W,size(A));
R = makerefmat(lon11,lat11,dlon,dlat);	W = [dlon 0 lon11; 0 dlat lat11]; R = georasterref(W,size(A));

- To specify the first data point of A and vector values of dx and dy or dlon and dlat, use these replacement patterns. Use the georasterref function for geographic coordinates and the maprasterref function for planar map coordinates.

Will Be Removed	Recommended
R = makerefmat(x11,y11,dx,dy);	W = [dx x11; dy y11]; R = maprasterref(W,size(A));
R = makerefmat(lon11,lat11,dlon,dlat);	W = [dlon lon11; dlat lat11]; R = georasterref(W,size(A));

- To specify arguments such as the raster size or latitude and longitude limits, use this replacement pattern. Use the georefcells function for a raster of cells and the georefpostings function for a raster of regularly posted samples.

Will Be Removed	Recommended

R = makerefmat('RasterSize',size, ...
          'LatitudeLimits',latlim,
          'LongitudeLimits',lonlim);
	R = georefcells(latlim,lonlim,size);
	...

You can also create a map raster reference object using the maprefcells or maprefpostings function. These functions are useful if you have information such as world limits or cell extents. Use the maprefcells function for a raster of cells and the maprefpostings function for a raster of regularly posted samples.

See Also

maprasterref | georasterref | georefpostings | georefcells | maprefcells | maprefpostings

makesymbolspec

Create vector layer symbol specification

Syntax

```
symbolspec = makesymbolspec(geometry,rule1,rule2,...,ruleN)
```

Description

`symbolspec = makesymbolspec(geometry,rule1,rule2,...,ruleN)` creates a symbol specification `symbolspec` for the shape `geometry` according to the rules specified in one or more `rule` arguments.

Use symbol specifications to customize the appearance of vector shapes for maps you create using the `mapshow` and `geoshow` functions.

Examples

Specify Default Line Color

Import a shapefile containing road data for Concord, MA, as a geospatial table.

```
roads = readgeotable("concord_roads.shp");
```

Create a symbol specification that specifies the default line color as purple.

```
blueRoads = makesymbolspec("Line",{'Default','Color','#7E2F8E'});
```

Display the roads using the symbol specification.

```
mapshow(roads,"SymbolSpec",blueRoads)
```
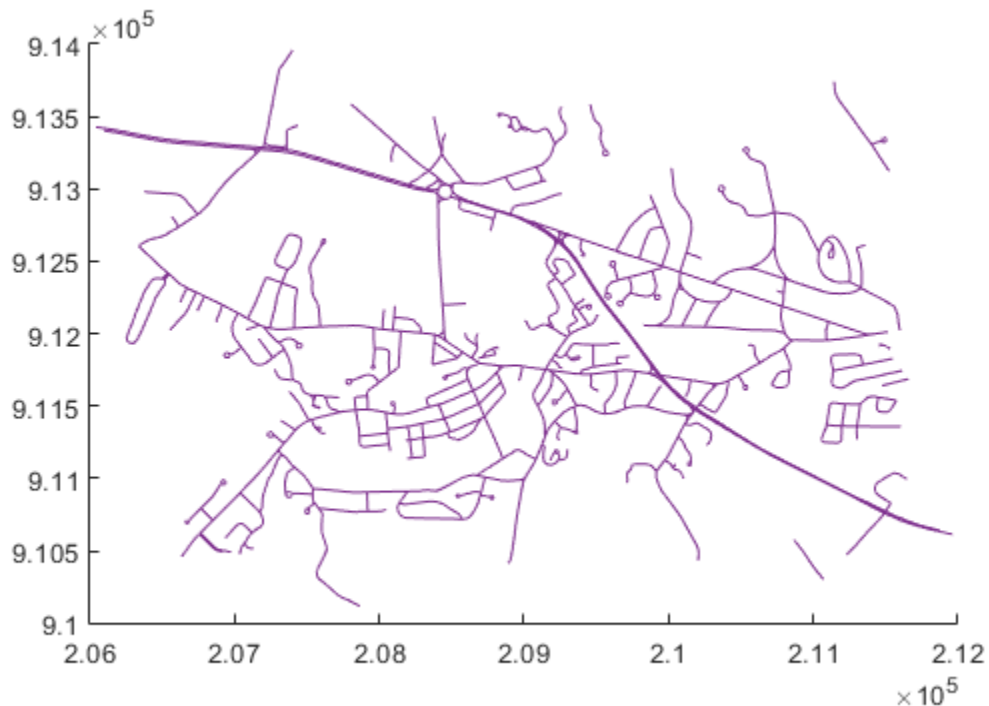
Specify Line Color Based on Attribute Values

Import a shapefile containing road data for Concord, MA, as a geospatial table.

```
roads = readgeotable("concord_roads.shp");
```

The CLASS attribute includes information about road types. For example, a value of 2 indicates a multilane highway and a value of 6 indicates a minor road. Create a symbol specification that specifies road colors based on the value of CLASS.

```
roadColors = makesymbolspec("Line", ...
    {'CLASS',2,'Color','#A2142F'}, ...
    {'CLASS',3,'Color','#77AC30'}, ...
    {'CLASS',6,'Color','#0072BD'}, ...
    {'Default','Color','k'});
```

Display the roads using the symbol specification.

```
mapshow(roads,"SymbolSpec",roadColors)
```

Specify Line Style Using Range of Attribute Values

Import a shapefile containing road data for Concord, MA, as a geospatial table.

```
roads = readgeotable("concord_roads.shp");
```

The CLASS attribute includes information about road types. For example, a value of 2 indicates a multilane highway and a value of 6 indicates a minor road. Create a symbol specification that specifies line styles and widths based on the value of CLASS such that major roads have a thick dashed lines and minor roads have thin dash-dotted lines.

```
lineStyles = makesymbolspec("Line", ...
    {'CLASS',[1 3],'LineStyle',':','LineWidth',2}, ...
    {'CLASS',[4 6],'LineStyle','-.','LineWidth',0.25});
```

Display the roads using the symbol specification.

```
mapshow(roads,"SymbolSpec",lineStyles)
```

Specify Line Style Using Range of Attribute and Property Values

Import a shapefile containing road data for Concord, MA, as a geospatial table.
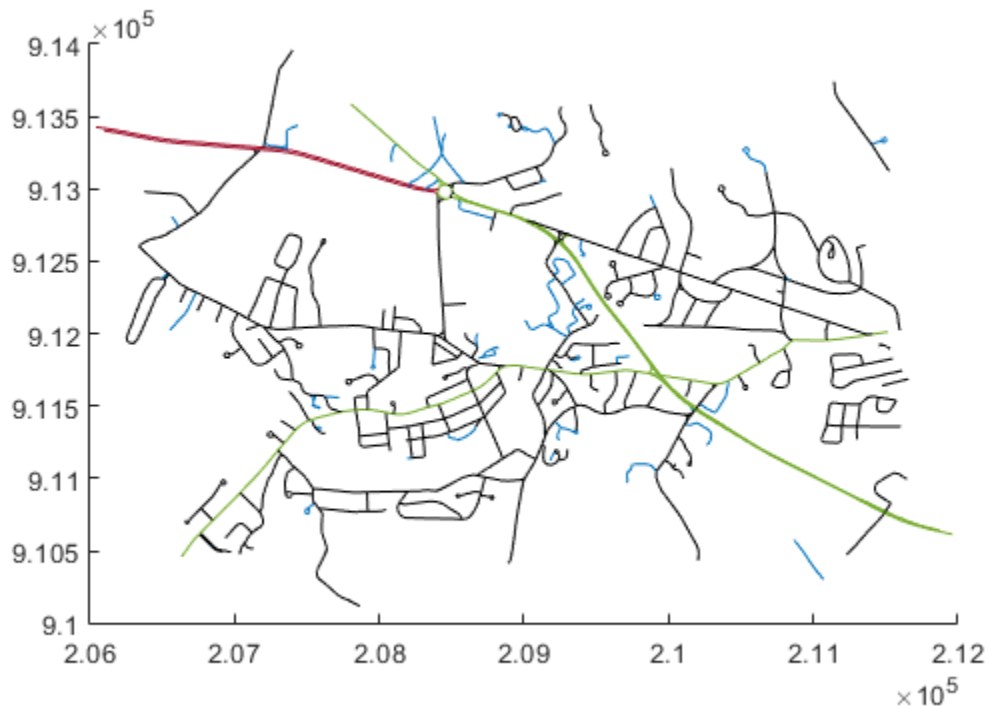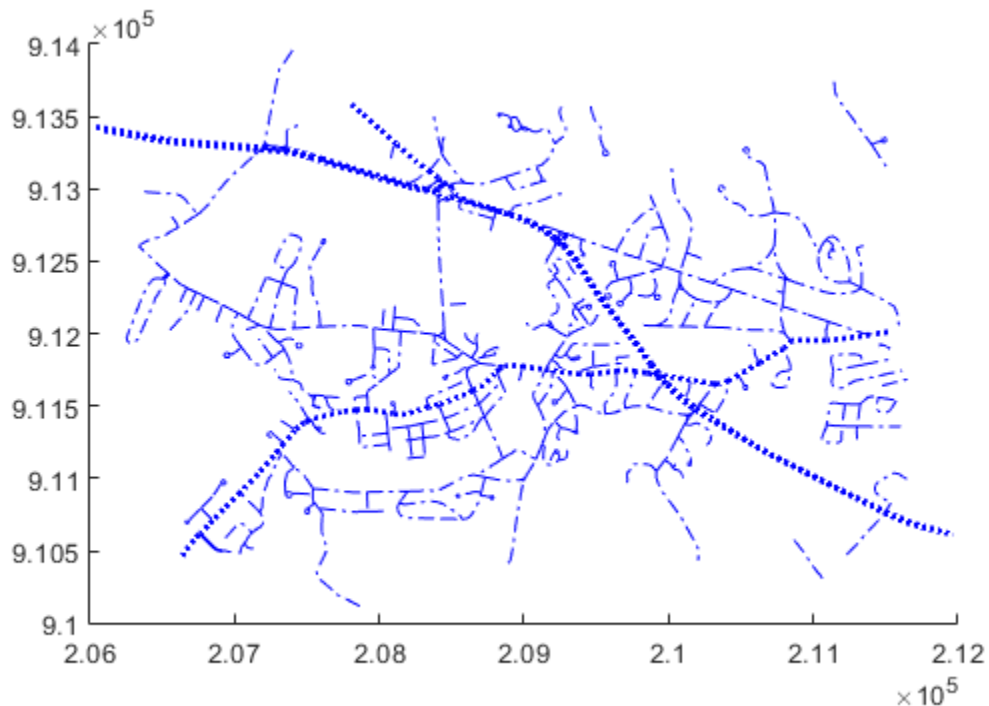
```
roads = readgeotable("concord_roads.shp");
```

The CLASS attribute includes information about road types. For example, a value of 2 indicates a multilane highway, a value of 3 indicates a major road, and a value of 6 indicates a minor road. Create a symbol specification that specifies line colors in the turbo colormap based on the value of CLASS so that highways are blue, major roads are green, and minor roads are red.

```
cmap = turbo(5);
colorRange = makesymbolspec("Line", ...
    {'CLASS',[2 6],'Color',cmap});
```

Display the roads using the symbol specification.

```
mapshow(roads,"SymbolSpec",colorRange)
```

Add a colorbar to the map by setting the colormap and colormap limits of the axes.

```
colormap(cmap)
caxis([2 7])
colorbar
```
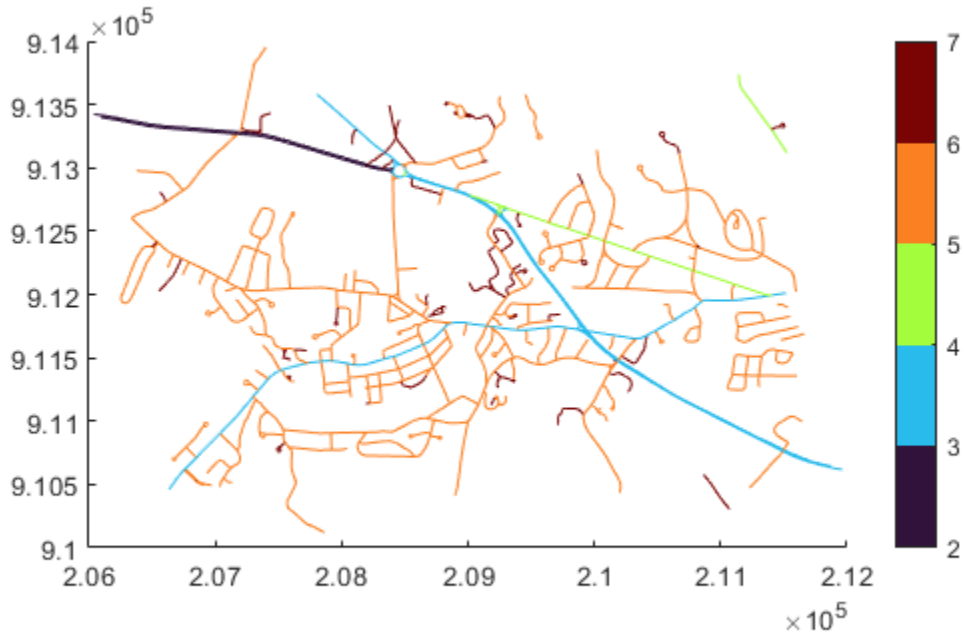
Input Arguments

geometry — Shape type
"Point" | "Line" | "PolyLine" | "Polygon" | "Patch"

Shape type, specified as one of these options:

- "Point" — Point shape
- "Line" — Line shape
- "PolyLine" — Series of connected line shapes
- "Polygon" — Polygon shape
- "Patch" — Patch shape

Data Types: char | string

rule — Shape customization rule
cell array

Shape customization rule, specified as a cell array.

To create a default rule for all features with the specified geometry, specify rule using this pattern:

{'Default',Property1,Value1,Property2,Value2,...,PropertyN,ValueN},

where `Property1`, `Property2`, and `PropertyN` are graphics properties and `Value1`, `Value2`, and `ValueN` are values. Specify properties using character vectors. Valid properties depend on the value of `geometry`:

- `"Point"` — Valid properties are `Marker`, `Color`, `MarkerEdgeColor`, `MarkerFaceColor`, `MarkerSize`, and `Visible`. For a description of each property, see Line Properties.
- `"Line"` and `"PolyLine"` — Valid properties are `Color`, `LineStyle`, `LineWidth`, and `Visible`. For a description of each property, see Line Properties.
- `"Polygon"` and `"Patch"` — Valid properties are `FaceColor`, `FaceAlpha`, `LineStyle`, `LineWidth`, `EdgeColor`, and `EdgeAlpha`, and `Visible`. For a description of each property, see Patch Properties.

To create a rule based on the values of an attribute, specify `rule` using this pattern:

`{AttributeName,AttributeValue,Property1,Value1,Property2,Value2,...,PropertyN`
`,ValueN},`

where `AttributeName` is the attribute and `AttributeValue` is the value or range of values. The size of `AttributeValue` and `Value1`, `Value2`, and `ValueN` do not need to match.

When a feature does not match a rule, the `mapshow` and `geoshow` functions display the feature using default graphics properties.

Data Types: `cell`

See Also

Functions
mapshow | geoshow | readgeotable

Introduced before R2006a

map2pix

(To be removed) Convert map coordinates to pixel coordinates

Note The map2pix function will be removed in a future release. Use the worldToIntrinsic function instead. For more information, see "Compatibility Considerations".

Syntax

```
[row,col] = map2pix(R,x,y)
p = map2pix(R,x,y)
[...] = map2pix(R,s)
```

Description

`[row,col] = map2pix(R,x,y)` calculates pixel coordinates `row,col` from map coordinates `x,y`. `R` is either a 3-by-2 referencing matrix defining a 2-dimensional affine transformation from intrinsic pixel coordinates to map coordinates, or a map raster reference object. `x` and `y` are vectors or arrays of matching size. The outputs `row` and `col` have the same size as `x` and `y`.

`p = map2pix(R,x,y)` combines `row` and `col` into a single array `p`. If `x` and `y` are column vectors of length n, then `p` is an n-by-2 matrix and each `p(k,:)` specifies the pixel coordinates of a single point. Otherwise, `p` has size `[size(row) 2]`, and `p(k1,k2,...,kn,:)` contains the pixel coordinates of a single point.

`[...] = map2pix(R,s)` combines `x` and `y` into a single array `s`. If `x` and `y` are column vectors of length n, the `s` should be an n-by-2 matrix such that each row (`s(k,:)`) specifies the map coordinates of a single point. Otherwise, `s` should have size `[size(X) 2]`, and `s(k1,k2,...,kn,:)` should contain the map coordinates of a single point.

Examples

```
% Find the pixel coordinates for the spatial coordinates
% (207050, 912900)
[X,cmap] = imread('concord_ortho_w.tif');
R = worldfileread('concord_ortho_w.tfw','planar',size(X));
[r,c] = map2pix(R,207050,912900);
```

Compatibility Considerations

map2pix will be removed
Not recommended starting in R2021b

Some functions that accept referencing matrices as input will be removed, including the map2pix function. Use a map raster reference object and the worldToIntrinsic function instead. Reference objects have several advantages over referencing matrices.

- Unlike referencing matrices, reference objects have properties that document the size of the associated raster, its limits, and the direction of its rows and columns. For more information about reference object properties, see the MapPostingsReference and MapCellsReference objects.

- You can manipulate the limits of rasters associated with reference objects using the `mapcrop` function.
- You can manipulate the size and resolution of rasters associated with reference objects using the `mapresize` function.
- Most functions that accept referencing matrices as inputs also accept reference objects.

To update your code, first create a reference object for either a raster of cells using the `maprefcells` function or a raster of regularly posted samples using the `maprefpostings` function. Alternatively, convert from a referencing matrix to a reference object using the `refmatToMapRasterReference` function.

Then, replace uses of the `map2pix` function with the `worldToIntrinsic` function according to these patterns.

Will Be Removed	Recommended
`[row,col] = map2pix(R,x,y);`	`[col,row] = worldToIntrinsic(R,x,y);`

`p = map2pix(R,x,y);`	If x and y are column vectors:
	
	`[col,row] = worldToIntrinsic(R,x,y);`
	`p = [row col];`
	
	If `row` and `col` have size *M*-by-*N* or 1-by-*N*:
	
	`[col,row] = worldToIntrinsic(R,x,y);`
	`p = cat(3,row,col);`

`[row,col] = map2pix(R,s);`	If `s` has size *N*-by-2: `[col,row] = worldToIntrinsic(R,s(:,1),s(:,2));`

See Also
`intrinsicToWorld` | `worldfileread` | `worldToIntrinsic`

Introduced before R2006a

mapbbox

(To be removed) Compute bounding box of georeferenced image or data grid

---

Note mapbbox will be removed in a future release. Instead, create a map raster reference object, and query its XWorldLimits and YWorldLimits properties. For more information, see "Compatibility Considerations".

Syntax

```
bbox = mapbbox(R,height,width)
bbox = mapbbox(R, sizea)
BBOX = mapbbox(info)
```

Description

bbox = mapbbox(R,height,width) computes the 2-by-2 bounding box of a georeferenced image or regular gridded data set. R is either a 3-by-2 referencing matrix defining a 2-dimensional affine transformation from intrinsic pixel coordinates to map coordinates, or a MapCellsReference object. (If you are using a raster reference object, you can calculate the bounding box from the object limit properties of the object rather than using this function.) height and width are the image dimensions. bbox bounds the outer edges of the image in map coordinates:

```
[minX minY
 maxX maxY]
```

bbox = mapbbox(R, sizea) accepts sizea = [height, width, ...] instead of height and width.

BBOX = mapbbox(info) accepts a scalar structure array with the fields

'RefMatrix'	3-by-2 referencing matrix
'Height'	Scalar number
'Width'	Scalar number

Compatibility Considerations

mapbbox will be removed
Not recommended starting in R2020b

Some functions that accept referencing matrices as input will be removed, including the mapbbox function. Instead, create a map raster reference object, and query its XWorldLimits and YWorldLimits properties. Reference objects have several advantages over referencing matrices.

- Unlike referencing matrices, reference objects have properties that document the size of the associated raster, its world limits, and the direction of its rows and columns. For more information about reference object properties, see the MapCellsReference and MapPostingsReference objects.

- You can manipulate the limits of rasters associated with reference objects using the `mapcrop` function.
- You can manipulate the size and resolution of rasters associated with reference objects using the `mapresize` function.

To update your code, first create a reference object for either a raster of cells using the `maprefcells` function or a raster of regularly posted samples using the `maprefpostings` function. Alternatively, convert from a referencing matrix to a reference object using the `refmatToMapRasterReference` function.

Then, find the limits of the raster by querying the `XWorldLimits` and `YWorldLimits` properties of the reference object, R. Create a bounding box matrix using the property values.

```
xlimits = R.XWorldLimits;
ylimits = R.YWorldLimits;
bbox = [xlimits' ylimits'];
```

See Also

Functions
geotiffinfo | mapoutline | worldGrid | intrinsicToWorld

Objects
MapCellsReference | MapPostingsReference

Introduced before R2006a
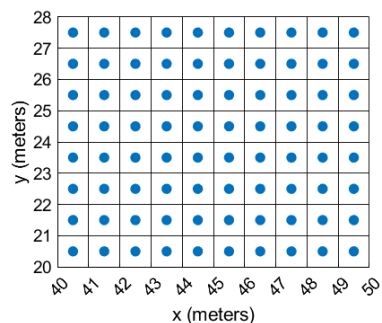
MapCellsReference

Reference raster cells to map coordinates

Description

A map cells reference object contains information that describes the relationship between a planar map coordinate system and an intrinsic coordinate system on page 1-864 that is anchored to the columns and rows of a 2-D spatially referenced raster grid or image of cells.

Typically, the raster is sampled regularly in the planar world *x* and world *y* coordinates of the map system, such that the intrinsic *x* and world *x* axes align and the intrinsic *y* and world *y* axes align. When this is true, the relationship between the two systems is rectilinear. More generally, and much more rarely, their relationship is affine. The affine relationship allows for a possible rotation (and skew). In either case, rectilinear or affine, the sample spacing from row to row need not equal the sample spacing from column to column. The cells or pixels need not be square. In the most general case, they could conceivably be parallelograms, but in practice they are always rectangular.

This image illustrates a 8-by-10 planar raster of cells. The raster cells have an extent of 1 meter in *x* and *y*, cover *x* values in the range [40,50] in meters, and cover *y* values in the range [20,28] in meters. The boundary of the raster of cells is made up of the outermost boundaries of the outermost cells.

To reference a raster of postings to planar map coordinates, use a `MapPostingsReference` object instead.

Creation

You can use these functions to create `MapCellsReference` objects:

- `readgeoraster` — Import a raster data file as an array and a reference object.
- `maprefcells` — Create a map raster reference object by specifying the *x* and *y* world limits of the raster and either the size of the raster or the extent of the cells.
- `maprasterref` — Convert a world file to a map raster reference object.
- `refmatToMapRasterReference` — Convert a referencing matrix to a map raster reference object.

Properties

XWorldLimits — Limits of raster in world *x*-coordinates
[0.5 2.5] (default) | two-element row vector

Limits of raster in world *x*-coordinates, specified as a two-element row vector of the form [xMin xMax].

The value of the ProjectedCRS property determines the length units for the raster. This code shows how to find the length units for a raster associated with the map cells reference object R.

R.ProjectedCRS.LengthUnit

Example: [207000 209000]

Data Types: double

YWorldLimits — Limits of raster in world *y*-coordinates
[0.5 2.5] (default) | two-element row vector

Limits of raster in world *y*-coordinates, specified as a two-element row vector of the form [yMin yMax].

The value of the ProjectedCRS property determines the length units for the raster. This code shows how to find the length units for a raster associated with the map cells reference object R.

R.ProjectedCRS.LengthUnit

Example: [911000 913000]

Data Types: double

RasterSize — Number of rows and columns of the raster or image associated with the referencing object
[2 2] (default) | two-element vector of positive integers

Number of rows and columns of the raster or image associated with the referencing object, specified as a two-element vector, [*m n*], where *m* represents the number of rows and *n* the number of columns.

For convenience, you can assign a size vector having more than two elements. This enables assignments like R.RasterSize = size(RGB), where RGB is *m*-by-*n*-by-3. In cases like this, the object stores only the first two elements of the size vector and ignores the higher (nonspatial) dimensions.

Example: [200 300]

Data Types: double

RasterInterpretation — Geometric nature of the raster
'cells' (default)

This property is read-only.

Geometric nature of the raster, specified as 'cells'. The value 'cells' indicates that the raster comprises a grid of quadrangular cells, and is bounded on all sides by cell edges. For an *m*-by-*n* raster, points with an intrinsic *x*-coordinate of 1 or *n* or an intrinsic *x*-coordinate of 1 or *m* fall within the raster, not on its edges.

Data Types: char

ColumnsStartFrom — Edge from which column indexing starts
'south' (default) | 'north'

Edge from which column indexing starts, specified as 'south' or 'north'.

Example: 'south'

Data Types: char

RowsStartFrom — Edge from which row indexing starts
'west' (default) | 'east'

Edge from which row indexing starts, specified as 'west' or 'east'.

Example: 'east'

Data Types: char

CellExtentInWorldX — Extent in world *x*-coordinates of individual cells
1 (default) | positive numeric scalar

Extent in world *x*-coordinates of individual cells, specified as a positive numeric scalar. Distance between the eastern and western limits of a single raster cell. The value is the same for all cells in the raster.

Example: 2.5

Data Types: double

CellExtentInWorldY — Extent in world *y*-coordinates of individual cells
1 (default) | positive numeric scalar

Extent in world *y*-coordinates of individual cells, specified as a positive numeric scalar. Distance between the northern and southern limits of a single raster cell. The value is the same for all cells in the raster.

Example: 2.5

Data Types: double

RasterExtentInWorldX — Extent of the full raster or image as measured in the world system in a direction parallel to its rows
2 (default) | positive numeric scalar

This property is read-only.

Extent of the full raster or image as measured in the world system in a direction parallel to its rows, specified as a positive numeric scalar. In the case of a rectilinear geometry, which is most typical, this is the horizontal direction (east-west).

Data Types: double

RasterExtentInWorldY — Extent of the full raster or image as measured in the world system in a direction parallel to its columns
2 (default) | positive numeric scalar

This property is read-only.

Extent of the full raster or image as measured in the world system in a direction parallel to its columns, specified as a positive numeric scalar. In the case of a rectilinear geometry, which is most typical, this is the vertical direction (north-south).

Data Types: `double`

XIntrinsicLimits — Raster limits in intrinsic *x*-coordinates
[0.5 2.5] (default) | two-element row vector of positive integers

This property is read-only.

Raster limits in intrinsic *x*-coordinates, specified as a two-element row vector of positive integers, [xMin xMax]. For an *m*-by-*n* raster, XIntrinsicLimits equals [0.5, m+0.5], because the RasterInterpretation is 'cells'.

Data Types: `double`

YIntrinsicLimits — Raster limits in intrinsic *y*-coordinates
[0.5 2.5] (default) | two-element row vector of positive integers

This property is read-only.

Raster limits in intrinsic *y*-coordinates, specified as a two-element row vector of positive integers, [yMin yMax]. For an *m*-by-*n* raster, YIntrinsicLimits equals [0.5, m+0.5], because the RasterInterpretation is 'cells'.

Data Types: `double`

TransformationType — Type of geometric relationship between intrinsic and world systems
'rectilinear' (default) | 'affine'

This property is read-only.

Type of geometric relationship between the intrinsic coordinate system and the world coordinate system, specified as either 'rectilinear' or 'affine'. Its value is 'rectilinear' when world *x* depends only on intrinsic *x* and vice versa, and world *y* depends only on intrinsic *y* and vice versa. When the value is 'rectilinear', the image displays without rotation in the world system, although it might be flipped. Otherwise, the value is 'affine'.

Data Types: `char`

CoordinateSystemType — Type of coordinate system to which the image or raster is referenced
'planar' (default)

This property is read-only.

Type of coordinate system to which the image or raster is referenced, specified as 'planar'.

Data Types: `char`

ProjectedCRS — Projected coordinate reference system
[] (default) | `projcrs` object

Projected coordinate reference system (CRS), specified as a `projcrs` object. A projected CRS consists of a geographic CRS and several parameters that are used to transform coordinates to and from the geographic CRS.

The value of `ProjectedCRS` determines the length units for the raster. To find the length units, query the `LengthUnit` property of the `projcrs` object.

Object Functions

contains	Determine if geographic or map raster contains points
firstCornerX	Return world x-coordinate of map raster index (1,1)
firstCornerY	Return world y-coordinate of map raster index (1,1)
intrinsicToWorld	Transform intrinsic to planar world coordinates
sizesMatch	Determine if geographic or map raster object and image or raster are size-compatible
worldFileMatrix	Return world file parameters for transformation
worldGrid	World coordinates of raster elements
worldToDiscrete	Transform planar world to discrete coordinates
worldToIntrinsic	Transform planar world to intrinsic coordinates

Examples

Display Planar Raster with Regularly Spaced Cell Centers

Import a land cover classification of Oahu, Hawaii as an array, a `MapCellsReference` object, and a colormap by using the `readgeoraster` function.

```
[Z,R,cmap] = readgeoraster('oahu_landcover.img');
```

View the properties of the `MapCellsReference` object.

```
R
```
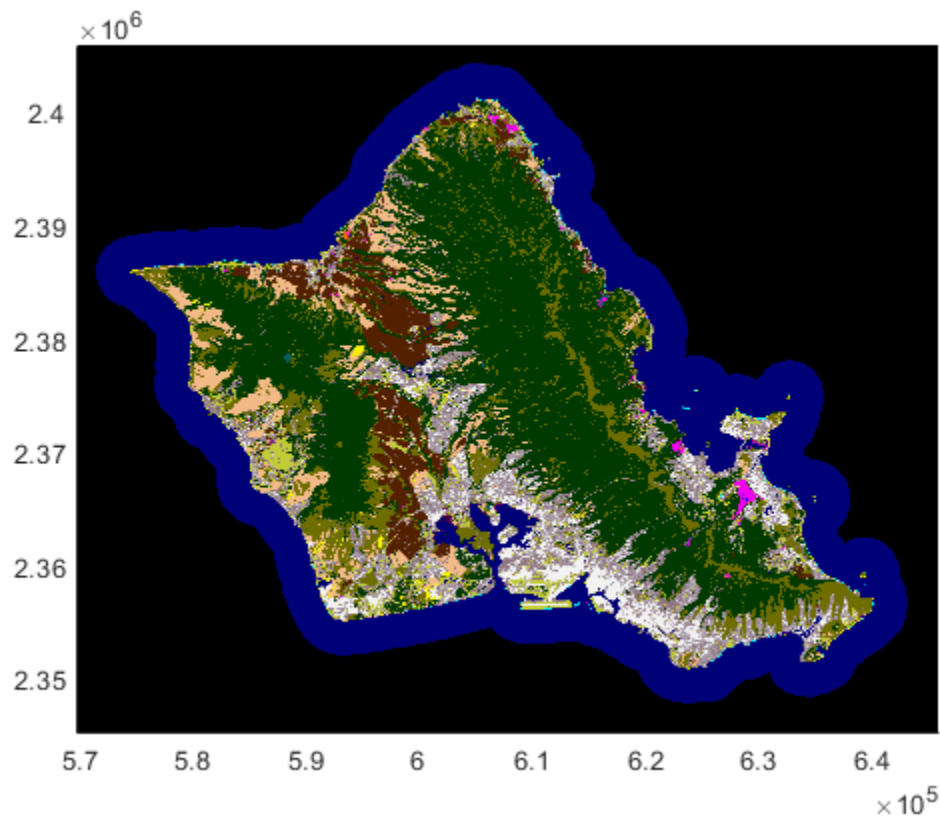
```
R =
  MapCellsReference with properties:

            XWorldLimits: [569857 645907]
            YWorldLimits: [2345284 2405884]
              RasterSize: [2020 2535]
    RasterInterpretation: 'cells'
        ColumnsStartFrom: 'north'
           RowsStartFrom: 'west'
      CellExtentInWorldX: 30
      CellExtentInWorldY: 30
    RasterExtentInWorldX: 76050
    RasterExtentInWorldY: 60600
         XIntrinsicLimits: [0.5 2535.5]
         YIntrinsicLimits: [0.5 2020.5]
       TransformationType: 'rectilinear'
     CoordinateSystemType: 'planar'
             ProjectedCRS: [1x1 projcrs]
```

Display the land cover classification on a map.

```
mapshow(Z,cmap,R)
```

More About

Intrinsic Coordinate System

A 2-D Cartesian system with its *x*-axis running parallel to the rows of a raster or image and its *y*-axis running parallel to the columns. *x* increases by 1 from column to column, and *y* increases by 1 from row to row.

The Mapping Toolbox and Image Processing Toolbox use the convention for the location of the origin relative to the raster cells or sampling points such that, at a sample location or at the center of a cell, *x* has an integer value equal to the column index. Likewise, at a sample location or at the center of a cell, *y* has an integer value equal to the row index. For details, see Image Coordinate Systems (Image Processing Toolbox).

See Also

Functions
maprefpostings | maprasterref | readgeoraster

Objects
MapPostingsReference | GeographicPostingsReference | GeographicCellsReference

Topics
"Spatially Reference Imported Rasters"

Introduced in R2013b

mapcrop

Crop projected map raster

Syntax

```
[B,RB] = mapcrop(A,RA,xlimits,ylimits)
```

Description

`[B,RB] = mapcrop(A,RA,xlimits,ylimits)` crops the projected map raster specified by A and raster reference RA and returns the cropped raster B and raster reference RB. The returned raster is cropped to limits in world coordinates close to those specified by `xlimits` and `ylimits`.

Examples

Crop Projected Raster Image

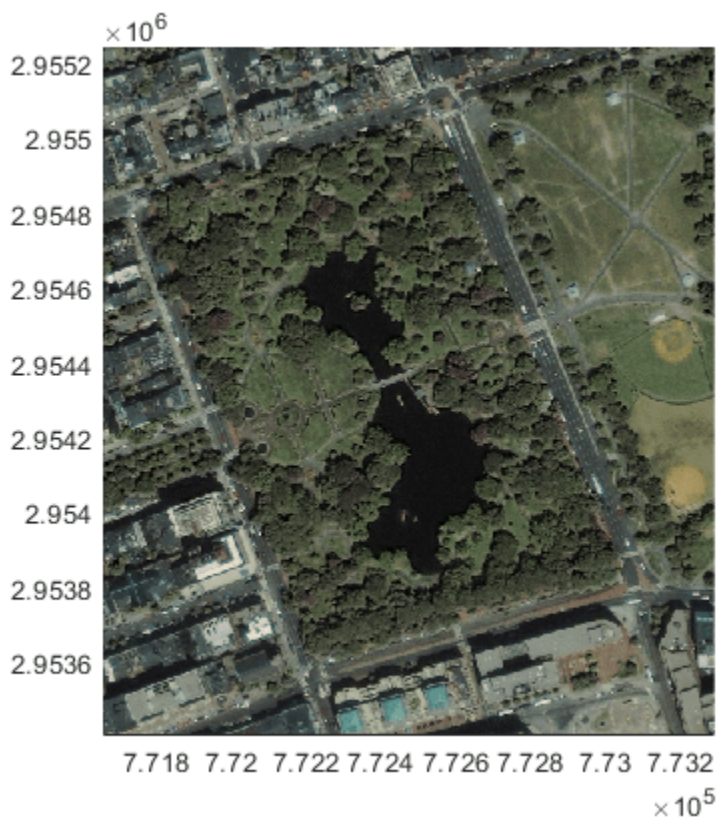Crop a projected raster image and display the cropped image.

First, import a projected raster image of Boston and a map cells reference object. Then, crop the image to the limits specified by `xlimits` and `ylimits`.

```
[A,RA] = readgeoraster('boston.tif');
xlimits = [771660  773290];
ylimits = [2953410 2955240];
[B,RB] = mapcrop(A,RA,xlimits,ylimits);
```

Display the cropped image.

```
mapshow(B,RB)
```

Input Arguments

A — Projected map raster
array

Projected map raster, specified as an *M*-by-*N* or *M*-by-*N*-by-*P* numeric or logical array.

RA — Raster reference
`MapCellsReference` object | `MapPostingsReference` object

Raster reference for A, specified as a `MapCellsReference` object or `MapPostingsReference` object. The `TransformationType` property of RA must be `'rectilinear'`.

xlimits — Minimum and maximum *x* limits
two-element vector

Minimum and maximum *x* limits, specified as a two-element numeric vector of the form `[xmin xmax]`, where `xmax` is greater than `xmin`.

ylimits — Minimum and maximum *y* limits
two-element vector

Minimum and maximum *y* limits, specified as a two-element numeric vector of the form `[ymin ymax]`, where `ymax` is greater than `ymin`.

Output Arguments

B — Cropped projected map raster
array

Cropped projected map raster, returned as a numeric or logical array. The data type and size of B matches the data type and size of A.

If the limits specified by `xlimits` and `ylimits` do not intersect the raster specified by A and RA, then B is empty.

RB — Raster reference
`MapCellsReference` object | `MapPostingsReference` object

Raster reference for B, returned as `MapCellsReference` object or `MapPostingsReference` object. The object type of RB matches the object type of RA.

The exact x and y limits of RB do not match the limits specified by `xlimits` and `ylimits`, unless they coincide with a cell boundary or posting location. Otherwise, the limits of RB are slightly larger than `xlimits` and `ylimits`.

If the limits specified by `xlimits` and `ylimits` do not intersect the raster specified by A and RA, then RB is empty.

See Also
`geocrop` | `mapresize`

Introduced in R2020a

mapinterp

Map raster interpolation

Syntax

```
Vq = mapinterp(V,R,xq,yq)
Vq = mapinterp( ___ ,method)
```

Description

`Vq = mapinterp(V,R,xq,yq)` interpolates the spatially referenced raster V, using bilinear interpolation. The function returns a value in `Vq` for each of the query points in arrays `xq` and `yq`. R is a map raster reference object that specifies the location and extent of data in `V`.

`Vq = mapinterp( ___ ,method)` specifies alternate interpolation methods.

Examples

Interpolate Spatially Referenced Raster Grid at Defined Coordinates

Load projected elevation data and a map cells reference object for an area around Mount Washington. Specify the points you want to interpolate. Then, interpolate the values.

```
[Z,R] = readgeoraster('MtWashington-ft.grd');
x = [ 312829  317447  316083  311150];
y = [4913618 4912253 4904329 4904172];
Vinterpolated = mapinterp(Z,R,x,y)

Vinterpolated = 1x4 int32 row vector

   1524    3678    6236    2365
```

Input Arguments

V — Spatially referenced raster grid
numeric or logical array

Spatially referenced raster grid, specified as numeric or logical array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

R — Map raster
`MapCellsReference` or `MapPostingsReference` object

Map raster, specified as a `MapCellsReference` or `MapPostingsReference` object.

To convert a referencing matrix to a map raster reference object, use `refmatToMapRasterReference`.

xq — Query point coordinates in *x* dimension
numeric array

Query point coordinates in *x* dimension, specified as a numeric array.

Data Types: `single` | `double`

yq — Query point coordinates in *y* dimension
numeric array

Query point coordinates in *y* dimension, specified as a numeric array.

Data Types: `single` | `double`

method — Interpolation methods
`'linear'` (default) | `'nearest'` | `'cubic'` | `'spline'`

Interpolation methods, specified as one of these values:

Method	Description
`'nearest'`	Nearest neighbor interpolation
`'linear'`	Bilinear interpolation
`'cubic'`	Bicubic interpolation
`'spline'`	Spline interpolation

Data Types: `char` | `string`

Output Arguments

Vq — Interpolated values
numeric array

Interpolated values, returned as a numeric array.

See Also
geointerp | interp2 | griddedInterpolant

Introduced in R2017a

maplist

Map projection support for map axes and map projection structures

Syntax

```
list = maplist
[list,defproj] = maplist
```

Description

`list = maplist` returns a structure that lists the IDs and classifications of map projections for use with map axes and map projection structures. The structure `list` contains the fields `Name`, `IdString`, `Classification`, and `ClassCode`. The `maps` and `axesmui` functions use the structure when processing map projection identifiers during operation of the toolbox functions.

`[list,defproj] = maplist` also returns the default projection's `IdString`.

`list.Name` defines the full name of the projection. This entry is used in the command-line table display and in the Projection Control Box.

`list.IdString` provides the name of the MATLAB function that computes the projection.

`list.Classification` defines the projection classification that is used in the command-line table display.

`list.ClassCode` defines the character vector that is used to label the classes of projections in the Projection Control Box. The eight class codes are

- `Azim` — Azimuthal
- `Coni` — Conic
- `Cyln` — Cylindrical
- `Mazi` — Modified azimuthal
- `Pazi` — Pseudoazimuthal
- `Pcon` — Pseudoconic
- `Pcy` — Pseudocylindrical
- `Poly` — Polyconic

See Also

maps | axesmui

mapoutline

Compute outline of georeferenced image or data grid

Syntax

```
[x,y] = mapoutline(R,height,width)
[x,y] = mapoutline(R, sizea)
[x,y] = mapoutline(info)
[x,y] = mapoutline(...,'close')
[lon,lat] = mapoutline(R,...)
outline = mapoutline(...)
```

Description

`[x,y] = mapoutline(R,height,width)` computes the outline of a georeferenced image or regular gridded data set in map coordinates. `R` is either a 3-by-2 referencing matrix defining a 2-dimensional affine transformation from intrinsic pixel coordinates to map coordinates, or a `MapCellsReference` or `MapPostingsReference` object. `height` and `width` are the image dimensions. `x` and `y` are 4-by-1 column vectors containing the map coordinates of the outer corners of the corner pixels, in the following order:

`(1,1), (height,1), (height, width), (1, width)`.

`[x,y] = mapoutline(R, sizea)` accepts `sizea = [height, width, ...]` instead of `height` and `width`.

`[x,y] = mapoutline(info)` accepts a scalar structure array with the fields

`'RefMatrix'`	3-by-2 referencing matrix
`'Height'`	Scalar number
`'Width'`	Scalar number

`[x,y] = mapoutline(...,'close')` returns `x` and `y` as 5-by-1 vectors, appending the coordinates of the first of the four corners to the end.

`[lon,lat] = mapoutline(R,...)`, where `R` georeferences pixels to longitude and latitude rather than map coordinates, returns the outline in geographic coordinates. Longitude must precede latitude in the output argument list.

`outline = mapoutline(...)` returns the corner coordinates in a 4-by-2 or 5-by-2 array.

Examples

Draw a red outline delineating the Boston GeoTIFF image, which is referenced to the Massachusetts Mainland State Plane coordinate system with units of survey feet.

```
figure
info = georasterinfo('boston.tif');
R = info.RasterReference;
```

```
[x,y] = mapoutline(R,R.RasterSize,'close');

hold on
plot(x,y,'r')
xlabel('MA Mainland State Plane easting, survey feet')
ylabel('MA Mainland State Plane northing, survey feet')
```

Draw a black outline delineating a TIFF image of Concord, Massachusetts, while lies roughly 25 km north west of Boston. Convert world file units to survey feet from meters to be consistent with the Boston image.

```
info  = imfinfo('concord_ortho_w.tif');
R = worldfileread('concord_ortho_w.tfw','planar', ...
        [info.Height info.Width]);
[x,y] = mapoutline(R, info.Height, info.Width, 'close');
x = x * unitsratio('sf','meter');
y = y * unitsratio('sf','meter');
plot(x,y,'k')
```
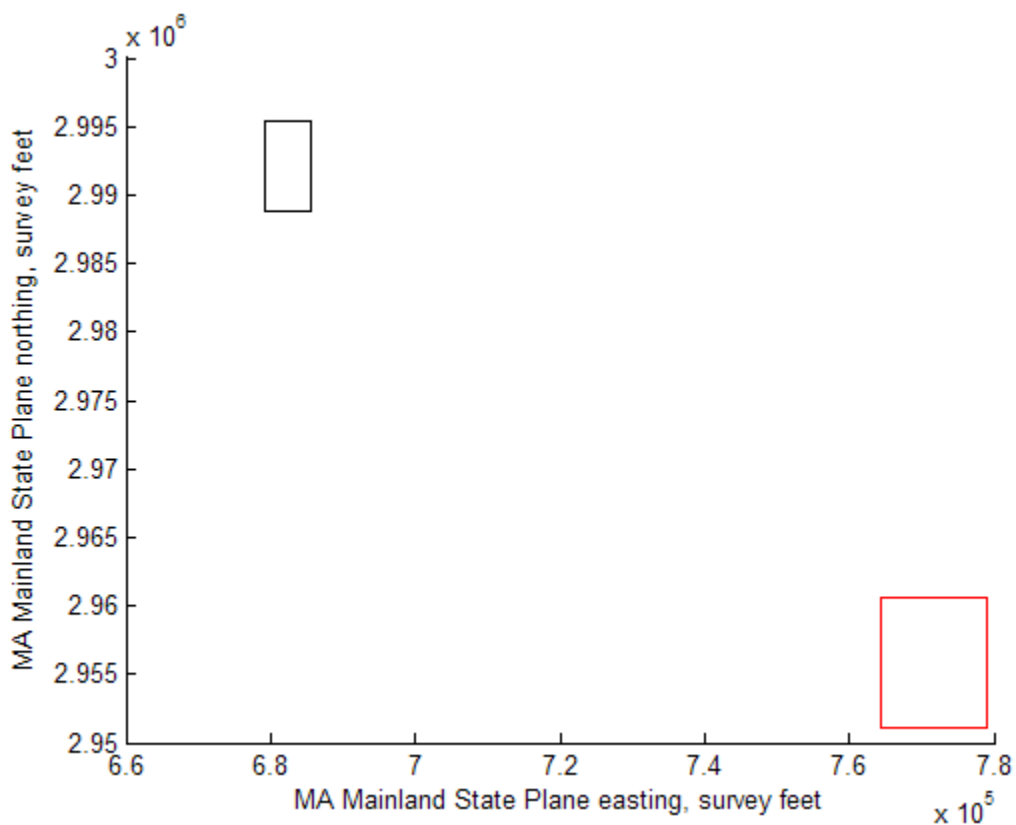

See Also

Functions
worldGrid | intrinsicToWorld

Objects
MapCellsReference | MapPostingsReference

Introduced before R2006a

mappoint

Planar point vector

Description

A mappoint vector is a container object that holds planar point coordinates and attributes. The points are coupled, such that the size of the *x*- and *y*-coordinate arrays are always equal and match the size of any dynamically added attribute arrays. Each entry of a coordinate pair and associated attributes, if any, represents a discrete element in the mappoint vector.

To create a planar point shape for use with a geospatial table, create a `mappointshape` object instead.

Creation

Syntax

```
p = mappoint()
p = mappoint(x,y)
p = mappoint(x,y,Name,Value)
p = mappoint(structArray)
p = mappoint(x,y,structArray)
```

Description

`p = mappoint()` constructs an empty mappoint vector, `p`, with these default property settings:

```
p =

 0x1 mappoint vector with properties:

 Collection properties:
    Geometry: 'point'
    Metadata: [1x1 struct]
 Feature properties:
           X: []
           Y: []
```

`p = mappoint(x,y)` constructs a new mappoint vector and assigns the X and Y properties to the numeric array inputs, `x` and `y`.

`p = mappoint(x,y,Name,Value)` constructs a mappoint vector, then adds dynamic properties to the mappoint vector using `Name`, `Value` argument pairs. You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

`p = mappoint(structArray)` constructs a new `mappoint` vector, assigning the fields of the structure array, `structArray`, as dynamic properties. Field values in `structArray` that are not numeric values, string scalar, string array, character vectors, or cell arrays of numeric values or character vectors are ignored.

`p = mappoint(x,y,structArray)` constructs a new `mappoint` vector, sets the X and Y properties equal to the numeric arrays x and y, and sets dynamic properties from the field values of `structArray`.

Properties

Each element in a mappoint vector is considered a feature. For more about the property types in `mappoint`, see "Collection Properties" on page 1-890 and "Feature Properties" on page 1-890.

Dynamic properties are new features that are added to a mappoint vector and that apply to each individual feature in the mappoint vector. You can attach dynamic Feature properties to a mappoint object during construction with a `Name,Value` pair or after construction using dot (.) notation. This is similar to adding dynamic fields to a structure. For an example of adding Feature properties dynamically, see "Construct a Mappoint Vector for Multiple Features and Examine Autosizing" on page 1-880.

Geometry — Type of geometry
`'point'`

Type of geometry, specified as `'point'`. For `mappoint`, Geometry is always `'point'`.

Data Types: `char` | `string`

Metadata — Information for the entire set of mappoint vector elements
scalar structure

Information for the entire set of mappoint vector elements, specified as a scalar structure. You can add any data type to the structure.

- If `Metadata` is provided as a dynamic property `Name` in the constructor, and the corresponding `Value` is a scalar structure, then `Value` is copied to the `Metadata` property. Otherwise, an error is issued.

- If a `Metadata` field is provided by `structArray`, and both `Metadata` and `structArray` are scalar structures, then the `Metadata` field value is copied to the `Metadata` property value. If `structArray` is a scalar but the `Metadata` field is not a structure, then an error is issued. If `structArray` is not scalar, then the `Metadata` field is ignored.

Data Types: `struct`

X — Planar *x*-coordinates
numeric row or column vector

Planar *x*-coordinate, specified as a numeric row or column vector.

Data Types: `double` | `single`

Y — Planar *y*-coordinates
numeric row or column vector

Planar *y*-coordinates, specified as a numeric row or column vector.

Data Types: `double` | `single`

Object Functions

append	Append features to geographic or planar vector
cat	Concatenate geographic or planar vector
disp	Display geographic or planar vector
fieldnames	Return dynamic property names of geographic or planar vector
isempty	Determine if geographic or planar vector is empty
isfield	Determine if dynamic property exists in geographic or planar vector
isprop	Determine if property exists in geographic or planar vector
length	Return number of elements in geographic or planar vector
properties	Return property names of geographic or planar vector
rmfield	Remove dynamic property from geographic or planar vector
rmprop	Remove property from geographic or planar vector
size	Return size of geographic or planar vector
struct	Convert geographic or planar vector to scalar structure
vertcat	Vertically concatenate geographic or planar vectors

Examples

Construct a Default Mappoint Vector

Dynamically set the X and Y property values, and dynamically add Vertex property Z.

```
p = mappoint();
p.X = 1:3;
p.Y = 1:3;
p.Z = [10 10 10]

p =

 3x1 mappoint vector with properties:

 Collection properties:
    Geometry: 'point'
    Metadata: [1x1 struct]
 Feature properties:
           X: [1 2 3]
           Y: [1 2 3]
           Z: [10 10 10]
```

Construct a Mappoint Vector from X and Y Values

Define x and y coordinates. Use them to create a mappoint.

```
x = [40 50 60];
y = [10, 11, 12];
p = mappoint(x, y)

p =

 3x1 mappoint vector with properties:

 Collection properties:
    Geometry: 'point'
```

```
        Metadata: [1x1 struct]
  Feature properties:
          X: [40 50 60]
          Y: [10 11 12]
```

Construct a Mappoint Vector from X, Y, and Temperature Values

```
x = 41:43;
y = 1:3;
temperature = 61:63;
p = mappoint(x, y, 'Temperature', temperature)

p =

 3x1 mappoint vector with properties:

 Collection properties:
      Geometry: 'point'
      Metadata: [1x1 struct]
 Feature properties:
             X: [41 42 43]
             Y: [1 2 3]
   Temperature: [61 62 63]
```

Construct a Mappoint Vector from a Structure Array

Create a structure array and then create a mappoint vector, specifying the array as input.

```
structArray = shaperead('boston_placenames')
p = mappoint(structArray)

structArray =

13x1 struct array with fields:
    Geometry
    X
    Y
    NAME
    FEATURE
    COORD


p =

 13x1 mappoint vector with properties:

 Collection properties:
    Geometry: 'point'
    Metadata: [1x1 struct]
 Feature properties:
           X: [1x13 double]
           Y: [1x13 double]
        NAME: {1x13 cell}
```

```
    FEATURE: {1x13 cell}
      COORD: {1x13 cell
```

Construct a Mappoint Vector from X and Y Numeric Arrays and a Structure Array

```
[structArray, A] = shaperead('boston_placenames');
x = [structArray.X];
y = [structArray.Y];
p = mappoint(x, y, A)

p =

 13x1 mappoint vector with properties:

 Collection properties:
    Geometry: 'point'
    Metadata: [1x1 struct]
 Feature properties:
           X: [1x13 double]
           Y: [1x13 double]
        NAME: {1x13 cell}
     FEATURE: {1x13 cell}
       COORD: {1x13 cell}
```

Construct a Mappoint Vector for One Feature

This example shows how to add a single feature to an empty mappoint vector after construction.

```
x = 1;
y = 1;
p = mappoint(x, y)

p =

 1x1 mappoint vector with properties:

 Collection properties:
    Geometry: 'point'
    Metadata: [1x1 struct]
 Feature properties:
           X: 1
           Y: 1
```

Add a dynamic Feature property with a character vector value.

```
p.FeatureName = 'My Feature'

p =

 1x1 mappoint vector with properties:

 Collection properties:
      Geometry: 'point'
      Metadata: [1x1 struct]
 Feature properties:
```

```
          X: 1
          Y: 1
FeatureName: 'My Feature'
```

Construct a Mappoint Vector for Multiple Features and Examine Autosizing

This example show how mappoint vectors autoresize all properties lengths to ensure they are equal in size when a new dynamic property is added or an existing property is appended or shortened.

Create a mappoint vector.

```
x = [1 2];
y = [10 10];
p = mappoint(x,y)

p =

 2x1 mappoint vector with properties:

 Collection properties:
    Geometry: 'point'
    Metadata: [1x1 struct]
 Feature properties:
           X: [1 2]
           Y: [10 10]
```

Add a dynamic Feature property.

```
p.FeatureName = {'Feature 1','Feature 2'}

p =

 2x1 mappoint vector with properties:

 Collection properties:
        Geometry: 'point'
        Metadata: [1x1 struct]
 Feature properties:
               X: [1 2]
               Y: [10 10]
     FeatureName: {'Feature 1' 'Feature 2'}
```

Add a numeric dynamic Feature property.

```
p.ID = [1 2]

p =

 2x1 mappoint vector with properties:

 Collection properties:
        Geometry: 'point'
        Metadata: [1x1 struct]
 Feature properties:
               X: [1 2]
               Y: [10 10]
```

```
    FeatureName: {'Feature 1' 'Feature 2'}
             ID: [1 2]
```

Add a third feature. All properties are autosized so that all vector lengths match.

```
p(3).X = 3
p(3).Y = 10

p =

 3x1 mappoint vector with properties:

 Collection properties:
      Geometry: 'point'
      Metadata: [1x1 struct]
 Feature properties:
             X: [1 2 3]
             Y: [10 10 10]
   FeatureName: {'Feature 1' 'Feature 2' ''}
            ID: [1 2 0]
```

Set the values for the ID feature dynamic property with more values than contained in X or Y. All properties are expanded to match in size.

```
p.ID = 1:4

p =

 4x1 mappoint vector with properties:

 Collection properties:
      Geometry: 'point'
      Metadata: [1x1 struct]
 Feature properties:
             X: [1 2 3 0]
             Y: [10 10 10 0]
   FeatureName: {'Feature 1' 'Feature 2' '' ''}
            ID: [1 2 3 4]
```

Set the values for the ID dynamic Feature property with fewer values than contained in X or Y. The ID property values expand to match the length of X and Y.

```
p.ID = 1:2

p =

 4x1 mappoint vector with properties:

 Collection properties:
      Geometry: 'point'
      Metadata: [1x1 struct]
 Feature properties:
             X: [1 2 3 0]
             Y: [10 10 10 0]
   FeatureName: {'Feature 1' 'Feature 2' '' ''}
            ID: [1 2 0 0]
```

Set the values of either coordinate property (X or Y) with fewer values. All properties shrink in size to match the new length.

```
p.X = 1:2

p =

 2x1 mappoint vector with properties:

 Collection properties:
        Geometry: 'point'
        Metadata: [1x1 struct]
  Feature properties:
               X: [1 2]
               Y: [10 10]
     FeatureName: {'Feature 1' 'Feature 2'}
              ID: [1 2]
```

Remove the `FeatureName` property by setting its value to [ ].

```
p.FeatureName = []

p =

 2x1 mappoint vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
  Feature properties:
            X: [1 2]
            Y: [10 10]
           ID: [1 2]
```

Remove all dynamic properties and set the object to empty by setting a coordinate property value to [ ].

```
p.X = []

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
  Feature properties:
            X: []
            Y: []
```

Construct a Mappoint Specifying Two Features

This example shows how to include multiple dynamic features during object construction.

```
point = mappoint([42 44],[10, 11],'Temperature',[63 65], ...
    'TemperatureUnits','Fahrenheit')

point =

 2x1 mappoint vector with properties:

 Collection properties:
        Geometry: 'point'
        Metadata: [1x1 struct]
```

```
 Feature properties:
              X: [42 44]
              Y: [10 11]
    Temperature: [63 65]
TemperatureUnits: 'Fahrenheit'
```

Create a Mappoint Vector from a MAT-File

This example shows how to construct a mappoint vector using data from a MAT-file containing oceanic depths.

Load data from the `seamount` MAT-file and construct a mappoint vector to hold the coordinates.

```
seamount = load('seamount');
p = mappoint(seamount.x, seamount.y, 'Z', seamount.z);
```

Create a level list to use to bin the `z` values and create a list of color values for each level.

```
levels = [unique(floor(seamount.z/1000)) * 1000; 0];
colors = {'red','green','blue','cyan','black'};
```

Add a `MinLevel` and `MaxLevel` feature property to indicate the lowest and highest binned level. Add a dynamic feature property to indicate the *z*-coordinate. Add a dynamic Feature property to indicate a binned level value and a color value for a given level. Include metadata information from the MAT-file.

```
for k = 1:length(levels) - 1
   index = levels(k) <= p.Z & p.Z < levels(k+1);
   p(index).MinLevel = levels(k);
   p(index).MaxLevel = levels(k+1) - 1;
   p(index).Color = colors{k};
end
```

Add metadata information. `Metadata` is a scalar structure containing information for the entire set of properties. You can add any type of data to the structure.

```
p.Metadata.Caption = seamount.caption;
p.Metadata
```

```
ans =

    Caption: [1x229 char]
```
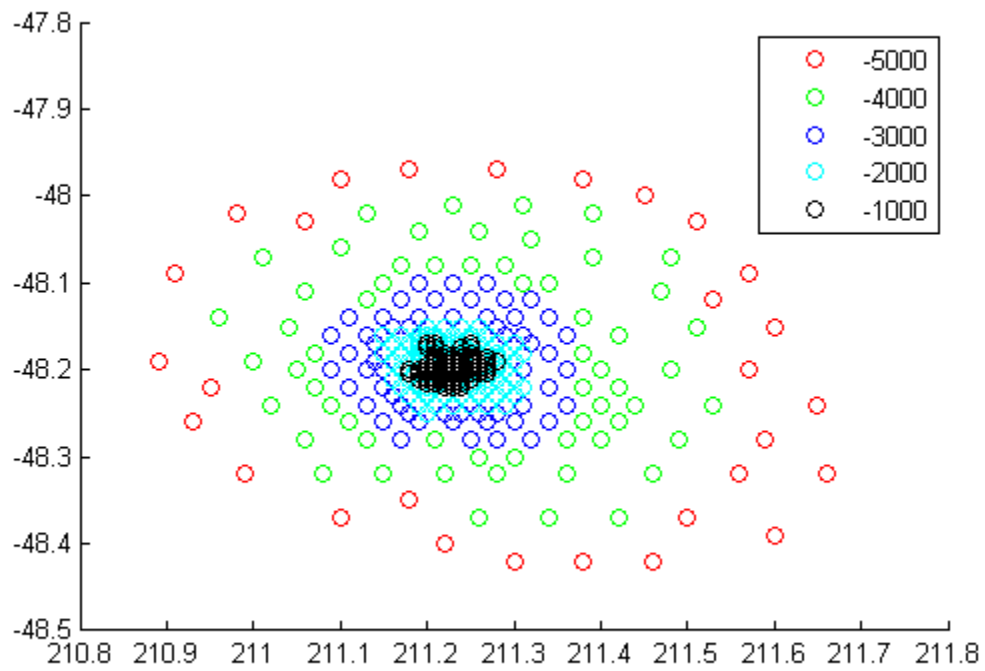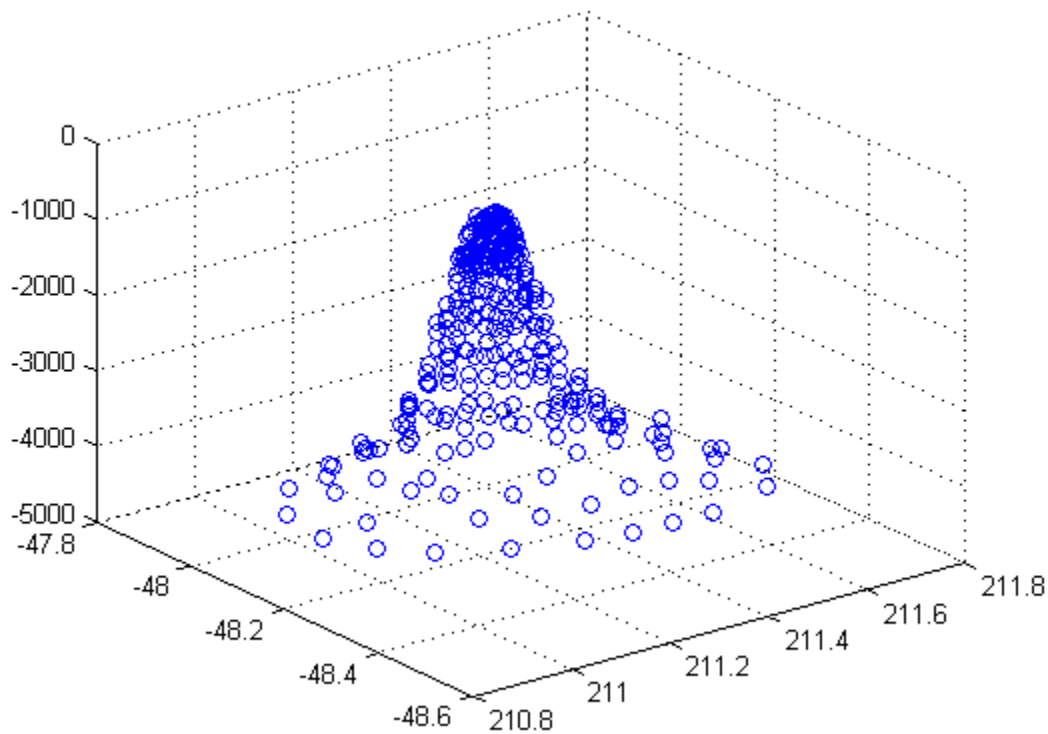
Display the point data as a 2-D plot.

```
figure
minLevels = unique(p.MinLevel);
for k=1:length(minLevels)
  index = p.MinLevel == minLevels(k);
  mapshow(p(index).X, p(index).Y, ...
   'MarkerEdgeColor', p(find(index,1)).Color, ...
   'Marker', 'o', ...
   'DisplayType', 'point')
end
legend(num2str(minLevels'))
```

Display the point data as a 3-D scatter plot.

```
figure
scatter3(p.X, p.Y, p.Z)
```

Assign Dynamic Features to Mappoint Vector from a Structure Array

This example shows how to create a mappoint vector from a structure array, and how to add features and metadata to the mappoint vector.

```
structArray = shaperead('boston_placenames');
p = mappoint();
p.X = [structArray.X];
p.Y = [structArray.Y];
p.Name = {structArray.NAME}


p =

 13x1 mappoint vector with properties:

 Collection properties:
    Geometry: 'point'
    Metadata: [1x1 struct]
 Feature properties:
          X: [1x13 double]
          Y: [1x13 double]
       Name: {1x13 cell}
```

Construct a mappoint vector from a structure array using the constructor syntax.

```
filename = 'boston_placenames.shp';
structArray = shaperead(filename);
p = mappoint(structArray)

p =

 13x1 mappoint vector with properties:

 Collection properties:
    Geometry: 'point'
    Metadata: [1x1 struct]
 Feature properties:
           X: [1x13 double]
           Y: [1x13 double]
        NAME: {1x13 cell}
     FEATURE: {1x13 cell}
       COORD: {1x13 cell}
```

Add a `Filename` field to the `Metadata` structure. Display the first five points and the `Metadata` structure.

```
p.Metadata.Filename = filename;
p(1:5)
p.Metadata

ans =

 5x1 mappoint vector with properties:

 Collection properties:
    Geometry: 'point'
    Metadata: [1x1 struct]
 Feature properties:
           X: [2.3403e+05 2.3357e+05 2.3574e+05 2.3627e+05 2.3574e+05]
           Y: [900038 9.0019e+05 9.0113e+05 9.0097e+05 9.0036e+05]
        NAME: {1x5 cell}
     FEATURE: {'PPL-SUBDVSN' ' MARSH' ' HILL' ' PPL' ' PENINSULA'}
       COORD: {1x5 cell}

ans =

    Filename: 'boston_placenames.shp'
```

Append a Point by Indexing

This example show how to add a feature to the mappoint vector using linear indexing.

Append Paderborn, Germany to the vector of world cities.

```
p = mappoint(shaperead('worldcities.shp'));
x = 51.715254;
y = 8.75213;
p = append(p, x, y, 'Name', 'Paderborn');
p(end)

ans =
```

```
    1x1 mappoint vector with properties:

  Collection properties:
      Geometry: 'point'
      Metadata: [1x1 struct]
  Feature properties:
             X: 51.7153
             Y: 8.7521
          Name: 'Paderborn'
```

You can also add a point to the end of the mappoint vector using linear indexing. Add Arlington, Virginia to the end of the vector.
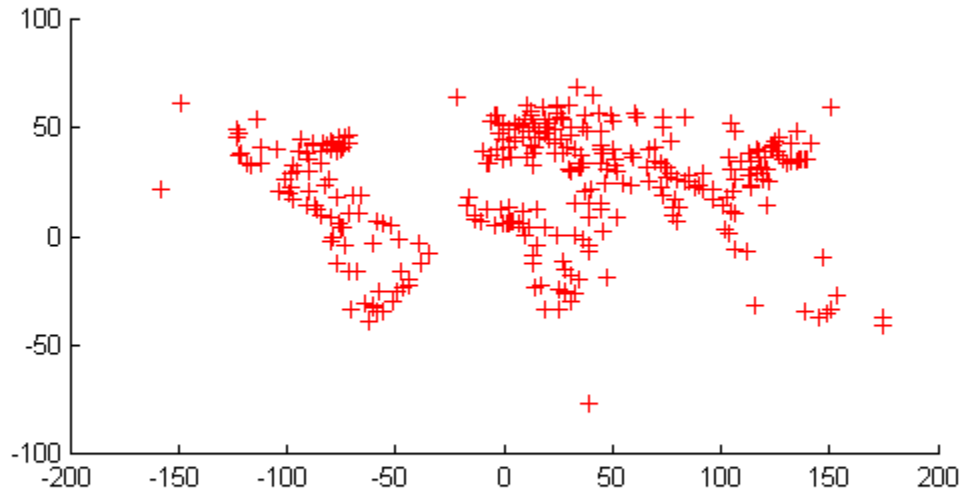
```
p(end+1).X = 38.880043;
p(end).Y = -77.196676;
p(end).Name = 'Arlington';
p(end-1:end)

ans =

  2x1 mappoint vector with properties:

  Collection properties:
      Geometry: 'point'
      Metadata: [1x1 struct]
  Feature properties:
             X: [51.7153 38.8800]
             Y: [8.7521 -77.1967]
          Name: {'Paderborn' 'Arlington'}

% Plot the points
figure
mapshow(p.X, p.Y, 'DisplayType', 'point')
```

Sort Dynamic Properties

This example show how features can be sorted by using the indexing behavior of the mappoint class.

Construct a mappoint vector and sort the dynamic properties.

```
p = mappoint(shaperead('tsunamis'));
p = p(:, sort(fieldnames(p)))

p =

 162x1 mappoint vector with properties:

 Collection properties:
       Geometry: 'point'
       Metadata: [1x1 struct]
 Feature properties:
              X: [1x162 double]
              Y: [1x162 double]
          Cause: {1x162 cell}
     Cause_Code: [1x162 double]
        Country: {1x162 cell}
            Day: [1x162 double]
    Desc_Deaths: [1x162 double]
         Eq_Mag: [1x162 double]
           Hour: [1x162 double]
```

```
        Iida_Mag: [1x162 double]
       Intensity: [1x162 double]
        Location: {1x162 cell}
      Max_Height: [1x162 double]
          Minute: [1x162 double]
           Month: [1x162 double]
      Num_Deaths: [1x162 double]
          Second: [1x162 double]
        Val_Code: [1x162 double]
        Validity: {1x162 cell}
            Year: [1x162 double]
```

Modify the mappoint vector to contain only the dynamic properties, `'Year'`, `'Month'`, `'Day'`, `'Hour'`, `'Minute'`.

```
p = p(:, {'Year', 'Month', 'Day', 'Hour', 'Minute'})

p =

 162x1 mappoint vector with properties:

 Collection properties:
    Geometry: 'point'
    Metadata: [1x1 struct]
 Feature properties:
           X: [1x162 double]
           Y: [1x162 double]
        Year: [1x162 double]
       Month: [1x162 double]
         Day: [1x162 double]
        Hour: [1x162 double]
      Minute: [1x162 double]
```

Display the first five elements.

```
p(1:5)

ans =

 5x1 mappoint vector with properties:

 Collection properties:
    Geometry: 'point'
    Metadata: [1x1 struct]
 Feature properties:
           X: [128.3000 -156 157.9500 143.8500 -155]
           Y: [-3.8000 19.5000 -9.0200 42.1500 19.1000]
        Year: [1950 1951 1951 1952 1952]
       Month: [10 8 12 3 3]
         Day: [8 21 22 4 17]
        Hour: [3 10 NaN 1 3]
      Minute: [23 57 NaN 22 58]
```

Row and Column Input Arguments

This example demonstrates that input arguments x and y can be either row or column vectors.

If you typically store *x*- and *y*-coordinate values in an *n*-by-2 or 2-by-*m* array, you can assign a mappoint object to these numeric values. If the values are stored in an *n*-by-2 array, then the X property values are assigned to the first column and the Y property values are assigned to the second column.

```
x = 1:10;
y = 21:30;
pts = [x' y'];
p = mappoint;
p(1:length(pts)) = pts

p =

 10x1 mappoint vector with properties:

 Collection properties:
    Geometry: 'point'
    Metadata: [1x1 struct]
 Feature properties:
          X: [1 2 3 4 5 6 7 8 9 10]
          Y: [21 22 23 24 25 26 27 28 29 30]
```

If the values are stored in a 2-by-*m* array, then the X property values are assigned to the first row and the Y property values are assigned to the second row.

```
pts = [x; y];
p(1:length(pts)) = pts

p =

 10x1 mappoint vector with properties:

 Collection properties:
    Geometry: 'point'
    Metadata: [1x1 struct]
 Feature properties:
          X: [1 2 3 4 5 6 7 8 9 10]
          Y: [21 22 23 24 25 26 27 28 29 30]
```

Observe that in both cases, X and Y are stored as row vectors.

More About

Collection Properties

Collection properties contain only one value per class instance. In contrast, the Feature property type has attribute values associated with each feature. Geometry and Metadata are the only two Collection properties.

Feature Properties

Feature properties contain one value (a scalar number, string scalar, or a character vector) for each feature in a mappoint vector. They are suitable for properties such as name, owner, serial number, or age, that describe a given feature (an element of a mappoint vector) as a whole. The X and Y coordinate properties are feature properties as there is one value for each element in the mappoint vector.

Feature properties can be added dynamically using dot notation. This is similar to adding dynamic fields to a structure.

Tips

- If X, Y, or a dynamic property is set with more values than features in the mappoint vector, then all other properties expand in size using 0 for numeric values and an empty character vector (' ') for cell values.
- If a dynamic property is set with fewer values than the number of features, then this dynamic property expands to match the size of the other properties, by inserting a 0 if the value is numeric or an empty character vector (' '), if the value is a cell array.
- If the X or Y property of the mappoint vector is set with fewer values than contained in the object, then all other properties shrink in size.
- If either X or Y is set to [ ], then both coordinate properties are set to [ ] and all dynamic properties are removed.
- If a dynamic property is set to [ ], then it is removed from the object.
- The mappoint vector can be indexed like any MATLAB vector. You can access any element of the vector to obtain a specific feature. The following examples demonstrate this behavior:

  "Append a Point by Indexing" on page 1-886

  "Sort Dynamic Properties" on page 1-888

See Also

Functions
gpxread | shaperead

Objects
geopoint | geoshape | mapshape

Introduced in R2012a

mappointshape

Point shape in planar coordinates

Description

A `mappointshape` object represents a point or multipoint in planar coordinates. A multipoint is an individual point shape that contains a set of point locations.

To represent a point or multipoint in geographic coordinates, use a `geopointshape` object instead.

Creation

To create `mappointshape` objects, either:

- Import point data in planar coordinates as a geospatial table using the `readgeotable` function, and then query the `Shape` variable of the table.
- Use the `mappointshape` function (described here).

Syntax

shape = mappointshape(x,y)

Description

shape = mappointshape(x,y) creates a `mappointshape` object or array of `mappointshape` objects with the specified *x*- and *y*-coordinates. The sizes of `x`, `y`, and the `mappointshape` object array `shape` match.

Input Arguments

x — *x*-coordinates
numeric array | cell array of numeric arrays

x-coordinates, specified as a numeric array or a cell array of numeric arrays.

- Create a point by specifying a scalar, such as `39`.
- Create a multipoint by specifying an array within a cell, such as `{[38 -30 29]}`.
- Create an array of points by specifying an array, such as `[38 -30 29]`.
- Create an array of points and multipoints by specifying a cell array of numeric arrays, such as `{39,[38 -30 29]}`.

Create placeholders for points with missing data by including `NaN` values. The `NaN` values in `x` must correspond to the `NaN` values in `y`.

The size of `x` must match the size of `y`. For cell arrays, the size of the array in each cell of `x` must match the size of the array in the corresponding cell of `y`.

Data Types: `double` | `cell`

y — *y*-coordinates
numeric array | cell array of numeric arrays

y-coordinates, specified as a numeric array or a cell array of numeric arrays.

- Create a point by specifying a scalar, such as `-113`.
- Create a multipoint by specifying an array within a cell, such as `{[-66 -31 42]}`.
- Create an array of points by specifying an array, such as `[-66 -31 42]`.
- Create an array of points and multipoints by specifying a cell array of numeric arrays, such as `{-113,[-66 -31 42]}`.

Create placeholders for points with missing data by including `NaN` values. The `NaN` values in `x` must correspond to the `NaN` values in `y`.

The size of `x` must match the size of `y`. For cell arrays, the size of the array in each cell of `x` must match the size of the array in the corresponding cell of `y`.

Properties

NumPoints — Number of points
array of nonnegative integers

This property is read-only.

Number of points, returned as an array of nonnegative integers.

For a `mappointshape` scalar, the value of `NumPoints` is `1` when the `mappointshape` object represents a single point and more than `1` when the object represents a multipoint.

For a `mappointshape` array, the size of `NumPoints` matches the size of the array.

Data Types: `double`

X — *x*-coordinates
array

x-coordinates, specified as an array.

For a `mappointshape` scalar, the size of `X` matches the value of `NumPoints`.

For a `mappointshape` array, the size of `X` matches the size of `NumPoints`. If the array contains `mappointshape` objects with multipoints, then accessing the `X` property of the array is not supported. Instead, access the `X` property of individual objects within the array.

This property is read-only for arrays when any element of `NumPoints` is greater than `1`.

`X` and `Y` must be the same size.

Data Types: `double`

Y — *y*-coordinates
array

y-coordinates, specified as an array.

For a `mappointshape` scalar, the size of Y matches the value of `NumPoints`.

For a `mappointshape` array, the size of Y matches the size of `NumPoints`. If the array contains `mappointshape` objects with multipoints, then accessing the Y property of the array is not supported. Instead, access the Y property of individual objects within the array.

This property is read-only for arrays when any element of `NumPoints` is greater than `1`.

X and Y must be the same size.

Data Types: `double`

Geometry — Geometric type
`"point"` (default)

This property is read-only.

Geometric type, returned as `"point"`.

Data Types: `string`

CoordinateSystemType — Coordinate system type
`"planar"` (default)

This property is read-only.

Coordinate system type, returned as `"planar"`.

Data Types: `string`

ProjectedCRS — Projected coordinate reference system
`[]` (default) | `projcrs` object

Projected coordinate reference system (CRS), specified as a `projcrs` object. A projected CRS consists of a geographic CRS and several parameters that are used to transform coordinates to and from the geographic CRS.

Examples

Get Information about Imported Planar Points

Import a shapefile containing the coordinates of locations in Boston as a geospatial table. The shapefile represents the locations using points. Get information about the points by querying the `Shape` variable of the table.
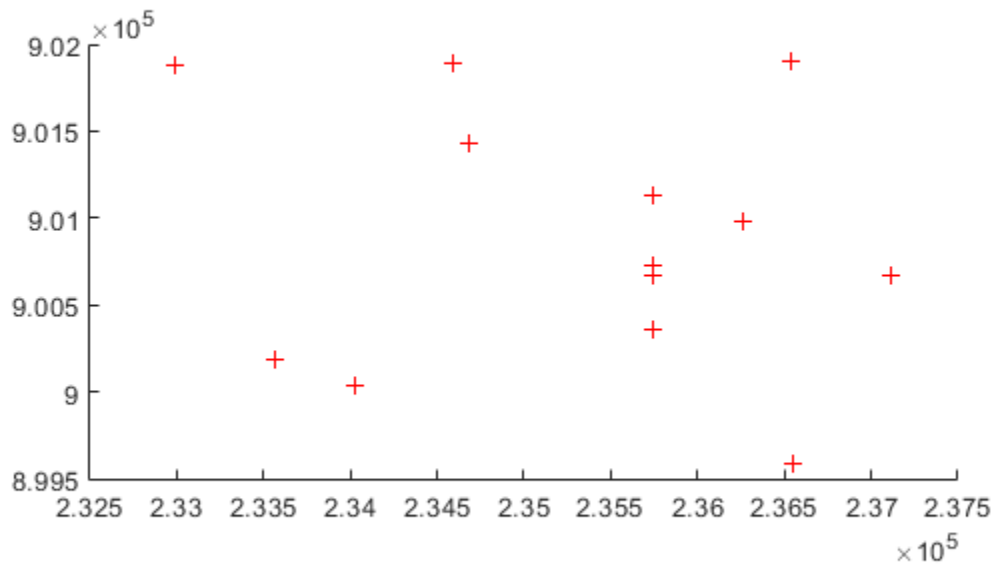
```
GT = readgeotable("boston_placenames.shp");
GT.Shape

ans =
  13x1 mappointshape array with properties:

              NumPoints: [13x1 double]
                      X: [13x1 double]
                      Y: [13x1 double]
               Geometry: "point"
    CoordinateSystemType: "planar"
```

```
        ProjectedCRS: [1x1 projcrs]
```

Display the locations.

```
mapshow(GT)
```


Create Planar Points and Multipoints

Create an individual point as a `mappointshape` scalar. Specify the projected CRS as World Equidistant Cylindrical, which has the EPSG authority code `4087`.

```
point = mappointshape(-113,39);
p = projcrs(4087);
point.ProjectedCRS = p
```

```
point =
  mappointshape with properties:

             NumPoints: 1
                     X: -113
                     Y: 39
              Geometry: "point"
    CoordinateSystemType: "planar"
          ProjectedCRS: [1x1 projcrs]
```

Create a multipoint as a `mappointshape` scalar.

```
multipoint = mappointshape({[-66 -31 42]},{[38 -30 29]});
multipoint.ProjectedCRS = p

multipoint =
  mappointshape with properties:

                NumPoints: 3
                        X: [-66 -31 42]
                        Y: [38 -30 29]
                 Geometry: "point"
    CoordinateSystemType: "planar"
            ProjectedCRS: [1x1 projcrs]
```

Create three individual points as a 1-by-3 `mappointshape` array.

```
pointVector = mappointshape([-66 -31 42],[38 -30 29]);
pointVector.ProjectedCRS = p

pointVector =
  1x3 mappointshape array with properties:

                NumPoints: [1 1 1]
                        X: [-66 -31 42]
                        Y: [38 -30 29]
                 Geometry: "point"
    CoordinateSystemType: "planar"
            ProjectedCRS: [1x1 projcrs]
```

Create one individual point and one multipoint as a 1-by-2 `mappointshape` array.

```
pointMultipoint = mappointshape({-113,[-66 -31 42]},{39, [38 -30 29]});
pointMultipoint.ProjectedCRS = p

pointMultipoint =
  1x2 mappointshape array with properties:

                NumPoints: [1 3]
                 Geometry: "point"
    CoordinateSystemType: "planar"
            ProjectedCRS: [1x1 projcrs]
```

See Also

Functions
`readgeotable`

Objects
`maplineshape` | `mappolyshape` | `geopointshape`

Topics
"Create Geospatial Tables"

Introduced in R2021b

maplineshape

Line shape in planar coordinates

Description

A `maplineshape` object represents a line or multiline in planar coordinates. A multiline is an individual line shape that contains a set of separate lines.

To represent a line or multiline in geographic coordinates, use a `geolineshape` object instead.

Creation

To create `maplineshape` objects, either:

- Import line data in planar coordinates as a geospatial table using the `readgeotable` function, and then query the `Shape` variable of the table.
- Use the `maplineshape` function (described here).

Syntax

shape = maplineshape(x,y)

Description

shape = maplineshape(x,y) creates a `maplineshape` object or array of `maplineshape` objects with vertices at the specified *x*- and *y*-coordinates. The sizes of x, y, and the `maplineshape` object array `shape` match.

Input Arguments

x — *x*-coordinates
numeric vector | cell array of numeric vectors

x-coordinates, specified as a numeric vector or a cell array of numeric vectors.

- Create a line by specifying a vector, such as [65 62 53 66].
- Create a multiline by specifying a vector and including line breaks as `NaN` values, such as [55 34 18 NaN 14 19 42 26].
- Create an array of lines and multilines by specifying a cell array of vectors, such as {[55 34 18],[14 19 NaN 42 26 31]}.

The `NaN` values in x must correspond to the `NaN` values in y.

The size of x must match the size of y. For cell arrays, the size of the vector in each cell of x must match the size of the vector in the corresponding cell of y.

Data Types: `double` | `cell`

y — *y*-coordinates
numeric vector | cell array of numeric vectors

y-coordinates, specified as a numeric vector or a cell array of numeric vectors.

- Create a line by specifying a vector, such as `[4 59 121 98]`.
- Create a multiline by specifying a vector and including line breaks as `NaN` values, such as `[78 56 63 NaN 83 106 104 126]`.
- Create an array of lines and multilines by specifying a cell array of vectors, such as `{[78 56 63],[83 106 NaN 104 126 131]}`.

The `NaN` values in `x` must correspond to the `NaN` values in `y`.

The size of `x` must match the size of `y`. For cell arrays, the size of the vector in each cell of `x` must match the size of the vector in the corresponding cell of `y`.

Properties

NumParts — Number of line parts
array of nonnegative integers

This property is read-only.

Number of line parts, returned as an array of nonnegative integers.

For a `maplineshape` scalar, the value of `NumParts` is `1` when the `maplineshape` object represents a single line and more than `1` when the object represents a multiline.

For a `maplineshape` array, the size of `NumParts` matches the size of the array.

Data Types: `double`

Geometry — Geometric type
`"line"` (default)

This property is read-only.

Geometric type, returned as `"line"`.

Data Types: `string`

CoordinateSystemType — Coordinate system type
`"planar"` (default)

This property is read-only.

Coordinate system type, returned as `"planar"`.

Data Types: `string`

ProjectedCRS — Projected coordinate reference system
`[]` (default) | `projcrs` object

Projected coordinate reference system (CRS), specified as a `projcrs` object. A projected CRS consists of a geographic CRS and several parameters that are used to transform coordinates to and from the geographic CRS.

Examples

### Get Information About Imported Planar Lines

Import a shapefile containing a network of road segments in Concord, MA as a geospatial table. The shapefile represents the road segments using lines. Get information about the fourth line by querying the Shape variable of the table.

```
GT = readgeotable("concord_roads.shp");
GT.Shape(4)
```

```
ans =
  maplineshape with properties:

               NumParts: 1
               Geometry: "line"
    CoordinateSystemType: "planar"
           ProjectedCRS: [1x1 projcrs]
```

Display the road segments.

```
mapshow(GT)
```
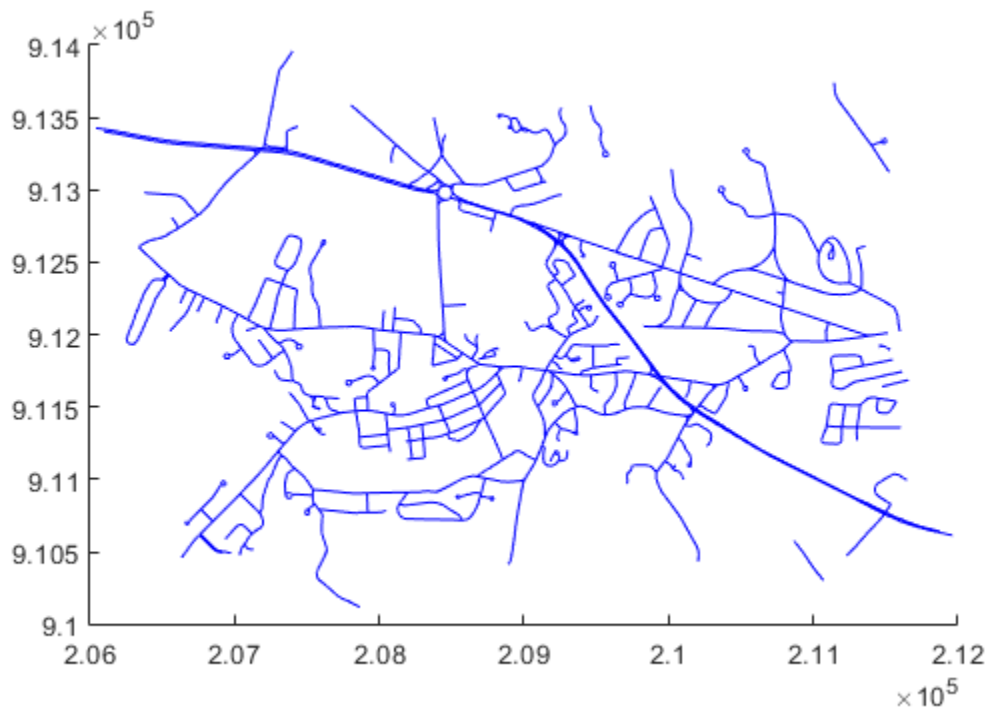
Create Planar Lines and Multilines

Create an individual line as a `maplineshape` scalar. Specify the projected CRS as World Equidistant Cylindrical, which has the EPSG authority code 4087.

```
x = [4 59 121 98];
y = [65 62 53 66];
lineshp = maplineshape(x,y);

p = projcrs(4087);
lineshp.ProjectedCRS = p

lineshp =
  maplineshape with properties:

                NumParts: 1
                Geometry: "line"
    CoordinateSystemType: "planar"
            ProjectedCRS: [1x1 projcrs]
```

Create a multiline as a `maplineshape` scalar.

```
x = [78 56 63 NaN 83 106 104 126];
y = [55 34 18 NaN 14 19 42 26];
multiline = maplineshape(x,y);
multiline.ProjectedCRS = p

multiline =
  maplineshape with properties:

                NumParts: 2
                Geometry: "line"
    CoordinateSystemType: "planar"
            ProjectedCRS: [1x1 projcrs]
```

Create one individual line and one multiline as a 1-by-2 `maplineshape` array.

```
x = {[78 56 63],[83 106 NaN 104 126 113]};
y = {[55 34 18],[14 19 NaN 42 26 37]};
lineMultiline = maplineshape(x,y);
lineMultiline.ProjectedCRS = p

lineMultiline=1×2 object
  1x2 maplineshape array with properties:

                NumParts: [1 2]
                Geometry: "line"
    CoordinateSystemType: "planar"
            ProjectedCRS: [1x1 projcrs]
```

See Also

Functions
readgeotable

Objects
mappointshape | mappolyshape | geolineshape

Topics
"Create Geospatial Tables"

Introduced in R2021b

mappolyshape

Polygon in planar coordinates

Description

A `mappolyshape` object represents a polygon or multipolygon in planar coordinates. A polygon is a region bounded by a closed curve and can include interior holes. A multipolygon is an individual polygon shape that includes multiple nonintersecting regions.

To represent a polygon or multipolygon in geographic coordinates, use a `geopolyshape` object instead.

Creation

To create `mappolyshape` objects, either:

- Import polygon data in planar coordinates as a geospatial table using the `readgeotable` function, and then query the Shape variable of the table.
- Use the `mappolyshape` function (described here).

Syntax

`shape = mappolyshape(x,y)`

Description

`shape = mappolyshape(x,y)` creates a `mappolyshape` object or array of `mappolyshape` objects with vertices at the specified *x*- and *y*-coordinates. The sizes of x, y, and the `mappolyshape` object array `shape` match.

The `mappolyshape` function assumes that x and y define polygons with valid topology. A polygon has valid topology when:

- Region interiors are to the right as you trace boundaries from vertex to vertex.
- The boundaries have no self-intersections.

In general, the outer boundaries of polygons with valid topology have vertices in a clockwise order and the interior holes have vertices in a counterclockwise order.

Input Arguments

x — *x*-coordinates
numeric vector | cell array of numeric vectors

x-coordinates, specified as a numeric vector or a cell array of numeric vectors.

- Create a polygon by specifying a vector, such as `[-113 -49 -100 -113]`.

- Create a polygon with holes or a multipolygon by specifying a vector and including breaks between the hole and region boundaries as NaN values, such as [69 90 105 79 69 NaN 6 52 43 14 6 NaN 18 32 22 18].
- Create an array of polygons and multipolygons by specifying a cell array of vectors, such as {[69 90 105 79 69],[6 52 43 14 6 NaN 18 32 22 18]}.

The NaN values in x must correspond to the NaN values in y.

The size of x must match the size of y. For cell arrays, the size of the vector in each cell of x must match the size of the vector in the corresponding cell of y.

Data Types: double | cell

y — *y*-coordinates
numeric vector | cell array of numeric vectors

y-coordinates, specified as a numeric vector or a cell array of numeric vectors.

- Create a polygon by specifying a vector, such as [39 45 19 39].
- Create a polygon with holes or a multipolygon by specifying a vector and including breaks between the hole and region boundaries as NaN values, such as [37 46 31 20 37 NaN 45 49 35 32 45 NaN 35 40 42 35].
- Create an array of polygons and multipolygons by specifying a cell array of vectors, such as {[37 46 31 20 37],[45 49 35 32 45 NaN 35 40 42 35]}.

The NaN values in x must correspond to the NaN values in y.

The size of x must match the size of y. For cell arrays, the size of the vector in each cell of x must match the size of the vector in the corresponding cell of y.

Properties

NumRegions — Number of regions
array of nonnegative integers

This property is read-only.

Number of regions, returned as an array of nonnegative integers. A region is a connected area such that any two points within the area can be connected by a path entirely within the area. Regions may contain holes.

For a mappolyshape scalar, the value of NumRegions is 1 when the mappolyshape object represents a single polygon and more than 1 when the object represents a multipolygon.

For a mappolyshape array, the size of NumRegions matches the size of the array.

Data Types: double

NumHoles — Number of holes
array of nonnegative integers

This property is read-only.

Number of holes in the polygon or multipolygon, returned as an array of nonnegative integers.

For a `mappolyshape` array, the size of `NumHoles` matches the size of the array.

Data Types: `double`

Geometry — Geometric type
`"polygon"` (default)

This property is read-only.

Geometric type, returned as `"polygon"`.

Data Types: `string`

CoordinateSystemType — Coordinate system type
`"planar"` (default)

This property is read-only.

Coordinate system type, returned as `"planar"`.

Data Types: `string`

ProjectedCRS — Projected coordinate reference system
`[]` (default) | `projcrs` object

Projected coordinate reference system (CRS), specified as a `projcrs` object. A projected CRS consists of a geographic CRS and several parameters that are used to transform coordinates to and from the geographic CRS.

Examples

Get Information About Imported Planar Polygons

Import a shapefile containing hydrographic data for Concord, MA as a geospatial table. The shapefile represents the data using polygons.

```
hydro = readgeotable("concord_hydro_area.shp");
```

Create a subtable that contains a polygon representing a pond. Get information about the polygon by querying the `Shape` variable of the table.

```
pond = hydro(14,:);
pond.Shape

ans =
  mappolyshape with properties:

            NumRegions: 1
              NumHoles: 3
              Geometry: "polygon"
   CoordinateSystemType: "planar"
            ProjectedCRS: [1×1 projcrs]
```

Import an orthophoto of Concord, MA, along with a map cells reference object and a colormap. Display the orthophoto and polygon on a map.

```
[ortho,R,cmap] = readgeoraster("concord_ortho_w.tif");
mapshow(ortho,cmap,R)
mapshow(pond,FaceColor="c")
```
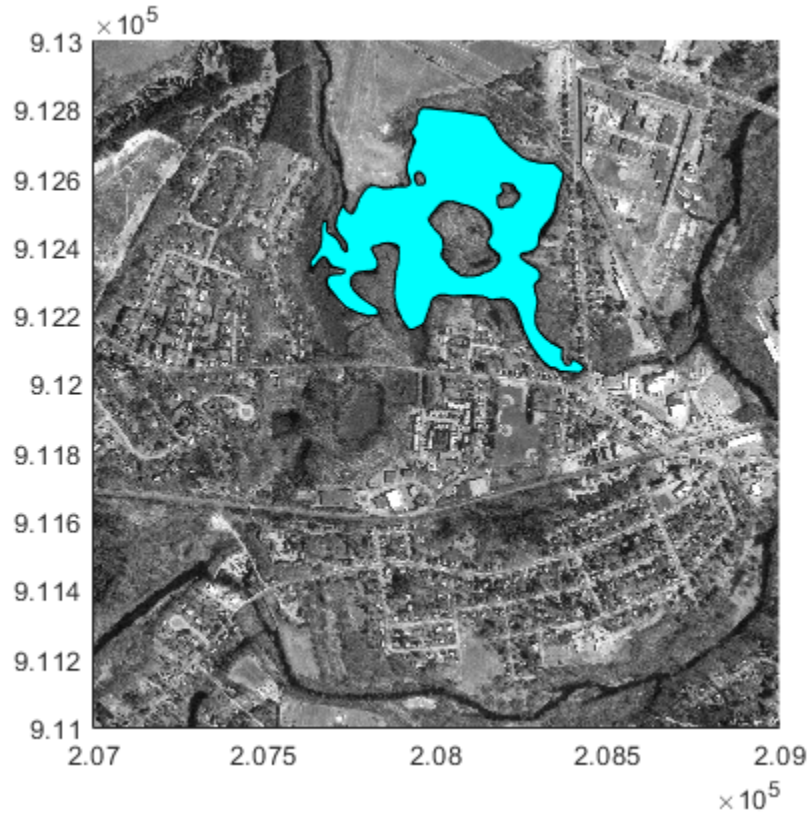

Create Planar Polygons and Multipolygons

Create an individual polygon with no holes as a `mappolyshape` scalar. Specify the projected CRS as World Equidistant Cylindrical, which has the EPSG authority code `4087`.

```
x = [-113 -49 -100 -113];
y = [39 45 19 39];
polyshp = mappolyshape(x,y);

p = projcrs(4087);
polyshp.ProjectedCRS = p

polyshp =
  mappolyshape with properties:

               NumRegions: 1
                 NumHoles: 0
                 Geometry: "polygon"
       CoordinateSystemType: "planar"
             ProjectedCRS: [1x1 projcrs]
```

Create a multipolygon with two regions and one hole as a `mappolyshape` scalar.

```
x = [69 90 105 79 69 NaN 6 52 43 14 6 NaN 18 32 22 18];
y = [37 46 31 20 37 NaN 45 49 35 32 45 NaN 35 40 42 35];
multipolygon = mappolyshape(x,y);
multipolygon.ProjectedCRS = p

multipolygon =
  mappolyshape with properties:

              NumRegions: 2
                NumHoles: 1
                Geometry: "polygon"
    CoordinateSystemType: "planar"
            ProjectedCRS: [1x1 projcrs]
```

Create two individual polygons as a 1-by-2 `mappolyshape` array. The second polygon contains a hole.

```
x = {[69 90 105 79 69],[6 52 43 14 6 NaN 18 32 22 18]};
y = {[37 46 31 20 37],[45 49 35 32 45 NaN 35 40 42 35]};
polyArray = mappolyshape(x,y);
polyArray.ProjectedCRS = p

polyArray=1×2 object
  1x2 mappolyshape array with properties:

              NumRegions: [1 1]
                NumHoles: [0 1]
                Geometry: "polygon"
    CoordinateSystemType: "planar"
            ProjectedCRS: [1x1 projcrs]
```

See Also

Functions
`readgeotable`

Objects
`mappointshape` | `maplineshape` | `geopolyshape`

Topics
"Create Geospatial Tables"

Introduced in R2021b

MapPostingsReference

Reference raster postings to map coordinates

Description

A map postings raster reference object contains information that describes the relationship between a planar map coordinate system and an intrinsic coordinate system on page 1-913 that is anchored to the columns and rows of a 2-D spatially referenced grid of point samples (or postings).

Typically, the raster is sampled regularly in the planar world *x*- and world *y*-coordinates of the map system, such that the intrinsic *x* and world *x* axes align and the intrinsic *y* and world *y* axes align. When this is true, the relationship between the two systems is rectilinear. More generally, and much more rarely, their relationship is affine. The affine relationship allows for a possible rotation (and skew). In either case, rectilinear or affine, the sample spacing from row to row need not equal the sample spacing from column to column.

This image illustrates a 9-by-11 planar raster of postings. The postings are spaced 1 meter apart in *x* and *y*, cover *x* values in the range [40,50] in meters, and cover *y* values in the range [20,28] in meters. The boundary of the raster of postings is a rectangle with edges that pass through the outermost postings.
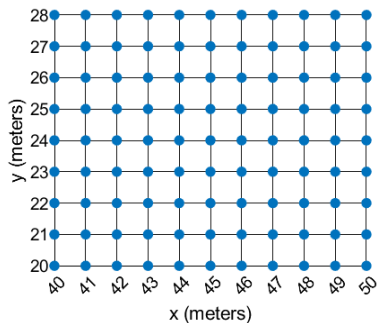
To reference a raster of cells to planar map coordinates, use a `MapCellsReference` object instead.

Creation

You can use these functions to create `MapPostingsReference` objects:

- `readgeoraster` — Import a raster data file as an array and a reference object.
- `maprefpostings` — Create a map raster reference object by specifying the *x* and *y* world limits of the raster and either the size of the raster or the spacing of the postings.
- `maprasterref` — Convert a world file to a map raster reference object.
- `refmatToMapRasterReference` — Convert a referencing matrix to a map raster reference object.

Properties

XWorldLimits — Limits of raster in world *x*-coordinates
[0.5 2.5] (default) | two-element row vector

Limits of raster in world *x*-coordinates, specified as a two-element row vector of the form [xMin xMax].

The value of the ProjectedCRS property determines the length units for the raster. This code shows how to find the length units for a raster associated with the map postings reference object R.

R.ProjectedCRS.LengthUnit

Example: [207000 209000]

Data Types: double

YWorldLimits — Limits of raster in world *y*-coordinates
[0.5 2.5] (default) | two-element row vector

Limits of raster in world *y*-coordinates, specified as a two-element row vector of the form [yMin yMax].

The value of the ProjectedCRS property determines the length units for the raster. This code shows how to find the length units for a raster associated with the map postings reference object R.

R.ProjectedCRS.LengthUnit

Example: [911000 913000]

Data Types: double

RasterSize — Number of rows and columns of the raster or image associated with the referencing object
[2 2] (default) | two-element vector of positive integers

Number of rows and columns of the raster or image associated with the referencing object, specified as a two-element vector, [*m n*], where *m* represents the number of rows and *n* the number of columns. For convenience, you can assign a size vector having more than two elements. This enables assignments like R.RasterSize = size(RGB), where RGB is *m*-by-*n*-by-3. In cases like this, the object stores only the first two elements of the size vector and ignores the higher (nonspatial) dimensions. *m* and *n* must be positive in all cases and must be 2 or greater.

Example: [200 300]

Data Types: double

RasterInterpretation — Geometric nature of the raster
'postings' (default)

Geometric nature of the raster, specified as 'postings'. The value 'postings' indicates that the raster comprises a grid of sample points, where rows or columns of samples run along the edge of the grid. For an *m*-by-*n* raster, points with an intrinsic *x*-coordinate of 1 or *n* or an intrinsic *y*-coordinate of 1 or *m* fall right on an edge (or corner) of the raster.

Cannot be set.

Data Types: char

ColumnsStartFrom — Edge from which column indexing starts
'south' (default) | 'north'

Edge from which column indexing starts, specified as 'south' or 'north'.

Example: ColumnsStartFrom: 'south'

Data Types: char

RowsStartFrom — Edge from which row indexing starts
'west' (default) | 'east'

Edge from which row indexing starts, specified as 'west' or 'east'.

Example: RowsStartFrom: 'east'

Data Types: char

SampleSpacingInWorldX — East-west distance between adjacent postings
2 (default) | positive numeric scalar

East-west distance between adjacent postings, specified as a positive numeric scalar. The value is constant throughout the raster.

Example: 2.5

Data Types: double

SampleSpacingInWorldY — North-south distance between adjacent postings
2 (default) | positive numeric scalar

North-south distance between adjacent postings, specified as a positive numeric scalar. The value is constant throughout the raster.

Example: 2.5

Data Types: double

RasterExtentInWorldX — Extent of the full raster or image as measured in the world system in a direction parallel to its rows
2 (default) | positive numeric scalar

This property is read-only.

Extent of the full raster or image as measured in the world system in a direction parallel to its rows, specified as a positive numeric scalar. In the case of a rectilinear geometry, which is most typical, this is the horizontal direction (east-west).

Data Types: double

RasterExtentInWorldY — Extent of the full raster or image as measured in the world system in a direction parallel to its columns
2 (default) | positive numeric scalar

This property is read-only.

Extent of the full raster or image as measured in the world system in a direction parallel to its columns. In the case of a rectilinear geometry, which is most typical, this is the vertical direction (north-south).

Data Types: `double`

XIntrinsicLimits — Raster limits in intrinsic *x*-coordinates
[1 2] (default) | two-element row vector of positive integers

This property is read-only.

Raster limits in intrinsic *x*-coordinates, specified as a two-element row vector of positive integers, [xMin xMax]. For an *m*-by-*n* raster, `XIntrinsicLimits` equals [1 *m*], because the `RasterInterpretation` is `'postings'`.

Example: [2 4]

Data Types: `double`

YIntrinsicLimits — Raster limits in intrinsic *y*-coordinates
[1 2] (default) | two-element row vector of positive integers

This property is read-only.

Raster limits in intrinsic *y*-coordinates, specified as a two-element row vector of positive integers, [yMin yMax]. For an *m*-by-*n* raster with `RasterInterpretation` equal to `'postings'`, `YIntrinsicLimits` equals [1 *m*].

Data Types: `double`

TransformationType — Type of geometric relationship between intrinsic and world systems
`'rectilinear'` (default) | `'affine'`

This property is read-only.

Type of geometric relationship between the intrinsic coordinate system and the world coordinate system, specified as either `'rectilinear'` or `'affine'`. Its value is `'rectilinear'` when world *x* depends only on intrinsic *x* and vice versa, and world *y* depends only on intrinsic *y* and vice versa. When the value is `'rectilinear'`, the image displays without rotation in the world system, although it might be flipped. Otherwise, the value is `'affine'`.

Data Types: `char`

CoordinateSystemType — Type of coordinate system to which the image or raster is referenced
`'planar'` (default)

This property is read-only.

Type of coordinate system to which the image or raster is referenced, specified as `'planar'`.

Data Types: `char`

ProjectedCRS — Projected coordinate reference system
[] (default) | `projcrs` object

Projected coordinate reference system (CRS), specified as a `projcrs` object. A projected CRS consists of a geographic CRS and several parameters that are used to transform coordinates to and from the geographic CRS.

The value of `ProjectedCRS` determines the length units for the raster. To find the length units, query the `LengthUnit` property of the `projcrs` object.

Object Functions

contains	Determine if geographic or map raster contains points
firstCornerX	Return world x-coordinate of map raster index (1,1)
firstCornerY	Return world y-coordinate of map raster index (1,1)
intrinsicToWorld	Transform intrinsic to planar world coordinates
sizesMatch	Determine if geographic or map raster object and image or raster are size-compatible
worldFileMatrix	Return world file parameters for transformation
worldGrid	World coordinates of raster elements
worldToDiscrete	Transform planar world to discrete coordinates
worldToIntrinsic	Transform planar world to intrinsic coordinates

Examples

Display Planar Raster of Regularly Spaced Posting Points

Import elevation data as an array and a `MapPostingsReference` object. Then, display the data on a map.

For this example, specify the file to import by extracting a GZ file into a temporary directory. Then, set the name of the first extracted file to a variable.

```
filenames = gunzip('sanfranciscos.dem.gz',tempdir);
filename1 = filenames{1};
```

Import the data by using the `readgeoraster` function. Prepare the data for plotting by specifying the output type as `'double'` and by replacing missing data with `NaN` values.

```
[Z,R] = readgeoraster(filename1,'OutputType','double');
info = georasterinfo(filename1);
Z = standardizeMissing(Z,info.MissingDataIndicator);
```

View the properties of the `MapPostingsReference` object.

```
R
```

```
R =
  MapPostingsReference with properties:

            XWorldLimits: [544050 555150]
            YWorldLimits: [4164120 4178070]
              RasterSize: [466 371]
    RasterInterpretation: 'postings'
        ColumnsStartFrom: 'north'
           RowsStartFrom: 'west'
   SampleSpacingInWorldX: 30
   SampleSpacingInWorldY: 30
    RasterExtentInWorldX: 11100
    RasterExtentInWorldY: 13950
        XIntrinsicLimits: [1 371]
        YIntrinsicLimits: [1 466]
```

```
    TransformationType: 'rectilinear'
 CoordinateSystemType: 'planar'
         ProjectedCRS: [1x1 projcrs]
```

Display the elevation data as a surface on a map. Apply a colormap that is appropriate for elevation data.

```matlab
mapshow(Z,R,'DisplayType','surface')
demcmap(Z)
```


More About

Intrinsic Coordinate System

A 2-D Cartesian system with its *x*-axis running parallel to the rows of a raster or image and its *y*-axis running parallel to the columns. *x* increases by 1 from column to column, and *y* increases by 1 from row to row.

The Mapping Toolbox and Image Processing Toolbox use the convention for the location of the origin relative to the raster cells or sampling points such that, at a sample location or at the center of a cell, *x* has an integer value equal to the column index. Likewise, at a sample location or at the center of a cell, *y* has an integer value equal to the row index. For details, see Image Coordinate Systems (Image Processing Toolbox).

See Also

Functions
maprefpostings | maprasterref | readgeoraster

Objects
GeographicCellsReference | GeographicPostingsReference | MapCellsReference

Topics
"Spatially Reference Imported Rasters"

Introduced in R2013b

mapprofile

Interpolate between waypoints on regular data grid

Syntax

```
[zi,ri,lat,lon] = mapprofile
[zi,ri,lat,lon] = mapprofile(Z,R,lat,lon)
[zi,ri,lat,lon] = mapprofile(Z,R,lat,lon,units)
[zi,ri,lat,lon] = mapprofile(Z,R,lat,lon,ellipsoid)
[zi,ri,lat,lon] = mapprofile( ___ ,'trackmethod','interpmethod')
```

Description

`mapprofile` plots a profile of values between waypoints on a displayed regular data grid. `mapprofile` uses the current object if it is a regular data grid, or the first regular data grid found on the current axes. The grid's `zdata` is used for the profile. The color data is used in the absence of `zdata`. The result is displayed in a new figure.

`[zi,ri,lat,lon] = mapprofile` returns the values of the profile without displaying them. The output `zi` contains interpolated values along great circles between the waypoints. `ri` is a vector of associated distances from the first waypoint in units of degrees of arc along the surface. `lat` and `lon` are the corresponding latitudes and longitudes.

`[zi,ri,lat,lon] = mapprofile(Z,R,lat,lon)` accepts as input a regular data grid and waypoint vectors. No displayed grid is required. Sets of waypoints may be separated by NaNs into line sequences. The output ranges are measured from the first waypoint within a sequence. R can be a geographic raster reference object, a referencing vector, or a referencing matrix.

If R is a geographic raster reference object, its `RasterSize` property must be consistent with `size(Z)`.

If R is a referencing vector, it must be a 1-by-3 with elements:

```
[cells/degree northern_latitude_limit western_longitude_limit]
```

If R is a referencing matrix, it must be 3-by-2 and transform raster row and column indices to or from geographic coordinates according to:

```
[lon lat] = [row col 1] * R
```

If R is a referencing matrix, it must define a (non-rotational, non-skewed) relationship in which each column of the data grid falls along a meridian and each row falls along a parallel.

`[zi,ri,lat,lon] = mapprofile(Z,R,lat,lon,units)` specifies the units of the output ranges along the profile. Valid range units inputs are any distance value recognized by `unitsratio`. Surface distances are computed using the default radius of the earth. If omitted, `'degrees'` are assumed.

`[zi,ri,lat,lon] = mapprofile(Z,R,lat,lon,ellipsoid)` uses the provided ellipsoid definition in computing the range along the profile. `ellipsoid` is a `referenceSphere`, `referenceEllipsoid`, or `oblateSpheroid` object, or a vector of the form [`semimajor_axis eccentricity`]. The output range is reported in the same distance units as the semimajor axes of

the ellipsoid. If you do not specify `ellipsoid` and R is a reference object with a nonempty `GeographicCRS` property, then `mapprofile` uses the ellipsoid contained in the `Spheroid` property of the `geocrs` object in the `GeographicCRS` property of R. Otherwise, `mapprofile` uses the unit sphere.

`[zi,ri,lat,lon] = mapprofile( ___ ,'trackmethod','interpmethod')` control the interpolation methods used. Valid track methods are `'gc'` for great circle tracks between waypoints, and `'rh'` for rhumb lines. Valid methods for interpolation within the matrix are `'bilinear'` for linear interpolation, `'bicubic'` for cubic interpolation, and `'nearest'` for nearest neighbor interpolation. If omitted, `'gc'` and `'bilinear'` are assumed.

Examples

Interpolate Between Waypoints

Load elevation raster data and a geographic cells reference object for the Korean peninsula. Specify an elevation profile. Then, plot the profile along with coastline and city marker data. When you select more than two waypoints, the automatically generated figure displays the result in 3-D.

```
load korea5c
plat = [ 43  43  41  38];
plon = [116 120 126 128];
mapprofile(korea5c,korea5cR,plat,plon)
load coastlines
plotm(coastlat,coastlon)
geoshow('worldcities.shp','Marker','.','Color', 'red')
```
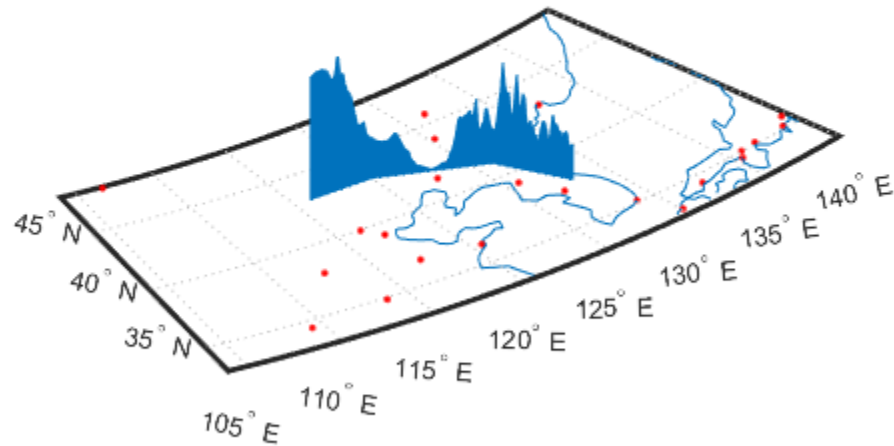
Interactively Select Waypoints

This example shows the relative sizes of the mountains in northern China (upper-left) compared to the depths of the Sea of Japan (lower-right).

First, load elevation raster data and a geographic cells reference object for the Korean peninsula. Create a map axes object with appropriate limits and display the data as a surface. Then, interactively pick waypoints by calling `mapprofile` without input arguments. Select the two waypoints, one in the upper-left corner and one in the lower-right corner, by clicking your mouse. Press **Enter** after you select the last point.

```
load korea5c
worldmap(korea5c,korea5cR)
meshm(korea5c,korea5cR)
demcmap(korea5c)
[zi,ri,lat,lon] = mapprofile;
```
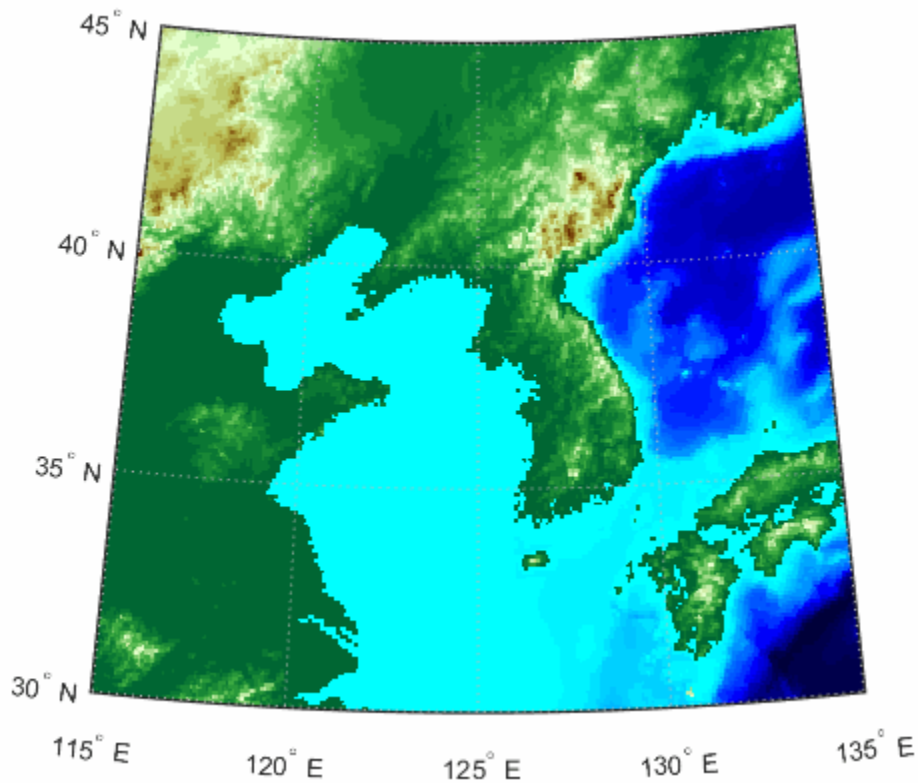
When you call `mapprofile` using output arguments, the results do not display in a new figure. Instead, use the results in further calculations or display the results yourself. For this example, convert ranges and elevations to kilometers and display them in a new figure. Set the vertical exaggeration factor to 50. Otherwise, the changes in elevation would be almost too small to see.

```
figure
plot(deg2km(ri),zi/1000)
daspect([ 1 1/50 1 ]);
xlabel 'Range (km)'
ylabel 'Elevation (km)'
```
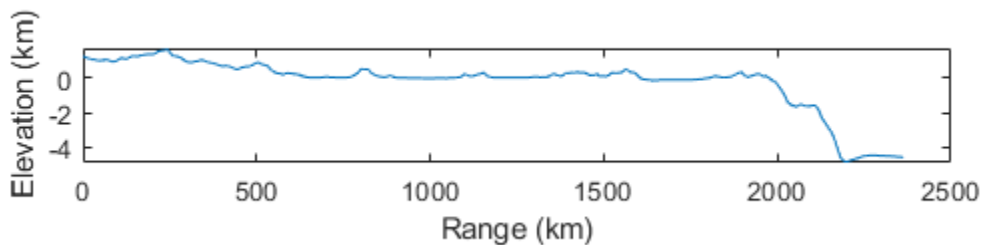

The profile you get depends on the transect locations you pick.

Specify Units and Interpolation Method

You can compute values along a path without reference to an existing figure by providing a regular data grid and vectors of waypoint coordinates. Optional arguments allow control over the units of the range output and interpolation methods between waypoints and data grid elements.

Show what land and ocean areas lie under a great circle track from Frankfurt to Seattle:

```
cities = shaperead('worldcities.shp', 'UseGeoCoords', true);
Seattle = strcmp('Seattle', {cities(:).Name});
Frankfurt = strcmp('Frankfurt', {cities(:).Name});
lat = [cities(Seattle).Lat cities(Frankfurt).Lat];
lon = [cities(Seattle).Lon cities(Frankfurt).Lon];
load topo60c
[valp,rp,latp,lonp] = ...
   mapprofile(topo60c,topo60cR, ...
   lat,lon,'km','gc','nearest');
figure
worldmap([40 80],[-135 20])
land = shaperead('landareas.shp', 'UseGeoCoords', true);
faceColors = makesymbolspec('Polygon',...
   {'INDEX', [1 numel(land)], 'FaceColor', ...
   polcmap(numel(land))});
geoshow(land,'SymbolSpec',faceColors)
plotm(latp,lonp,'r')
plotm(lat,lon,'ro')
axis off
```
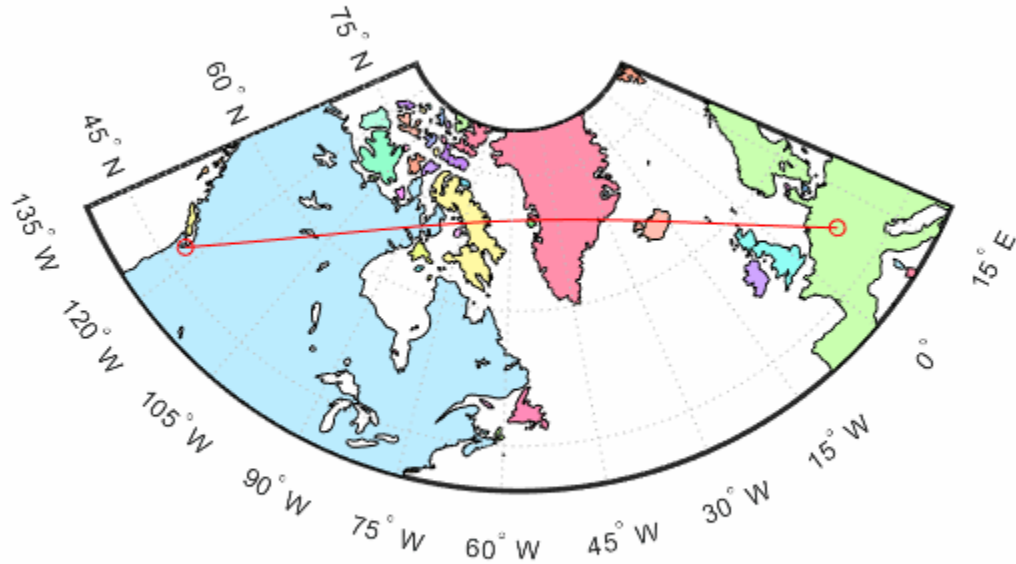
Compute Elevation Profile Along Straight Line

Load elevation data and a geographic cells reference object for the Korean peninsula. View the region on geographic axes using a satellite basemap.

```
load korea5c
figure
geolimits(korea5cR.LatitudeLimits,korea5cR.LongitudeLimits)
geobasemap('satellite')
```
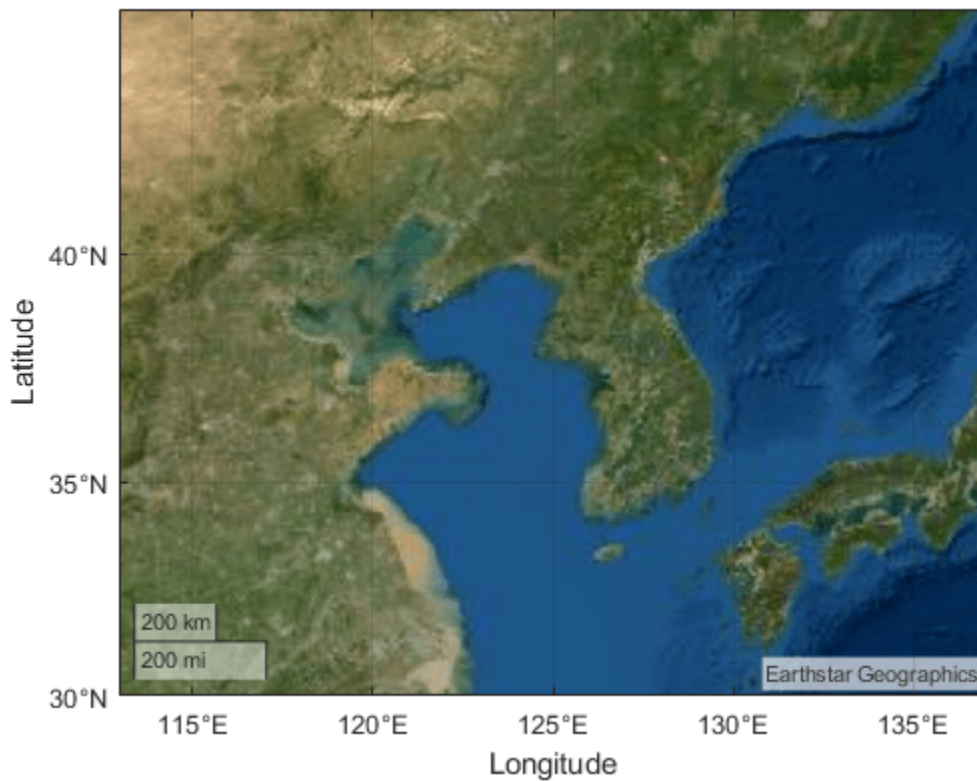
Define endpoints for a straight-line transect through the region. Compute the elevation profile by using the `mapprofile` function. By default, the `mapprofile` function uses bilinear interpolation along a great circle track.

```
plat = [40.5 30.7];
plon = [121.5 133.5];
[z,rng,lat,lon] = mapprofile(korea5c,korea5cR,plat,plon);
```

Preserve the basemap and limits by using `hold on`. Plot the transect on the geographic axes.

```
hold on
geoplot(lat,lon,'w','LineWidth',2)
```
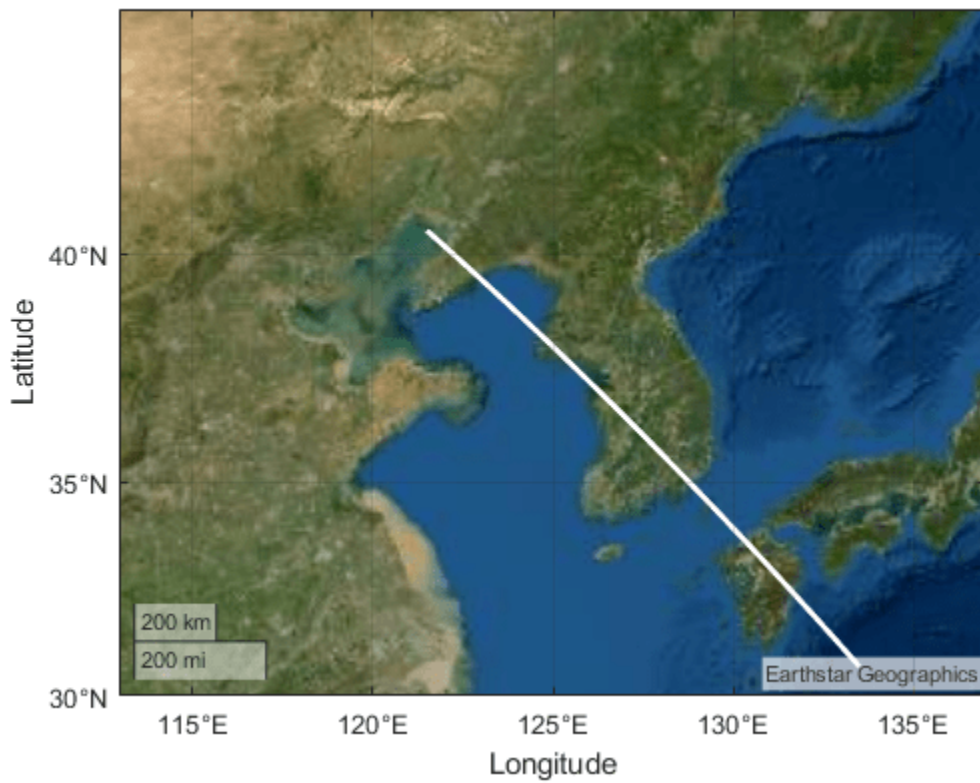
Plot the transect range and elevation on Cartesian axes.

```
figure
plot(rng,z,'r')
```
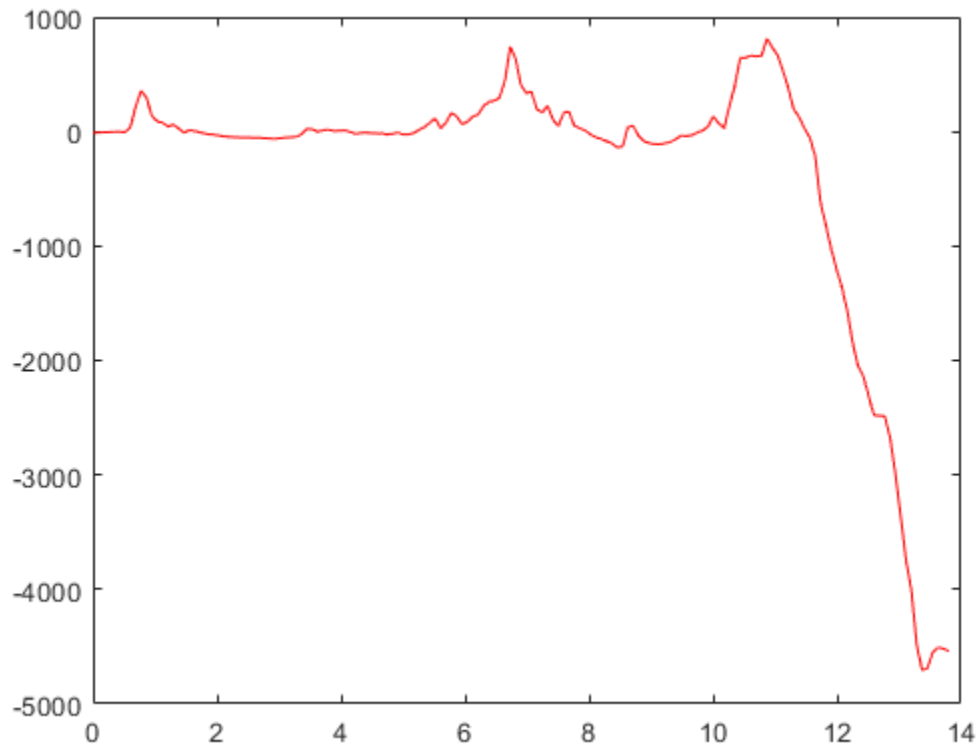
See Also

geointerp | los2

maprasterref

Construct map raster reference object

Note Use the `maprefcells` function or the `maprefpostings` function instead, except when constructing a raster reference object from a world file matrix.

Syntax

```
R = maprasterref(W,rasterSize)
R = maprasterref(W,rasterSize,rasterInterpretation)
R = maprasterref(Name,Value)
```

Description

`R = maprasterref(W,rasterSize)` creates a reference object for a regular raster of cells in planar map coordinates using the specified world file matrix `W` and raster size `rasterSize`.

`R = maprasterref(W,rasterSize,rasterInterpretation)`, where `rasterInterpretation` is `'postings'`, specifies that the raster contains regularly posted samples in planar map coordinates. The default for `rasterInterpretation` is `'cells'`, which specifies a regular raster of cells.

`R = maprasterref(Name,Value)` accepts a list of name-value pairs that are used to assign selected properties when initializing a map raster reference object.

Input Arguments

`W` — World file matrix
2-by-3 numeric array

World file matrix, specified as a 2-by-3 numeric array. Each of the six elements in `W` matches one of the lines in a world file that defines the transformation in raster referencing object R.

Data Types: `double`

`rasterSize` — Number of rows and columns of the raster
two-element vector

Number of rows (*m*) and columns (*n*) of the raster or image associated with the referencing object, specified as a two-element vector [*m n*]. For convenience, you may assign a size vector having more than two elements to `RasterSize`. This flexibility enables assignments like `R.RasterSize = size(RGB)`, for example, where RGB is *m*-by-*n*-by-3. However, in such cases, only the first two elements of the size vector are actually stored. The higher (non-spatial) dimensions are ignored

`rasterInterpretation` — Control to handle raster edges
`'cells'` (default) | `'postings'`

Controls handling of raster edges. The `rasterInterpretation` input is optional, and can equal either `'cells'` or `'postings'`.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

You can include any of the following properties, overriding their default values as needed. Alternatively, you may omit any or all properties when constructing your map raster reference object. Then, you can customize the result by resetting properties from this list one at a time. This name-value syntax always results in an object with a `'rectilinear'` `TransformationType`. If your image is rotated with respect to the world coordinate axes, you need an object with a `TransformationType` of `'affine'`. Alternately, you can provide an appropriate world file matrix as input, as shown in the third syntax. You cannot do it by resetting properties of an existing rectilinear map raster reference object.

XLimWorld

Limits of raster in world *x*

Two-element row vector of the form `[xMin xMax]`.

Default: [0.5 2.5]

YLimWorld

Limits of raster in world *y*

Two-element row vector of the form `[yMin yMax]`.

Default: [0.5 2.5]

RasterSize

Two-element vector `[M N]` specifying the number of rows (*M*) and columns (*N*) of the raster or image associated with the referencing object. For convenience, you may assign a size vector having more than two elements to `RasterSize`. This flexibility enables assignments like `R.RasterSize = size(RGB)`, for example, where RGB is *M*-by-*N*-by-3. However, in such cases, only the first two elements of the size vector are actually stored. The higher (non-spatial) dimensions are ignored.

Default: [2 2]

RasterInterpretation

Controls handling of raster edges, specified as either `'cells'` or `'postings'`.

Default: `'cells'`

ColumnsStartFrom

Edge where column indexing starts, specified as either `'south'` or `'north'`.

Default: `'south'`

RowsStartFrom

Edge from which row indexing starts, specified as either `'west'` or `'east'`.

Default: `'west'`

Output Arguments

R — Map raster
`MapCellsReference` or `MapPostingsReference` object

Map raster, specified as a `MapCellsReference` or `MapPostingsReference` object.

Examples

Construct Raster Referencing Object Specifying Limits

Construct a referencing object for an 1000-by-2000 image with square, half-meter pixels referenced to a planar map coordinate system (the "world" system). The X-limits in the world system are 207000 and 208000. The Y-limits are 912500 and 913000. The image follows the popular convention in which world X increases from column to column and world Y decreases from row to row.

```
R = maprasterref('RasterSize', [1000 2000], ...
        'YWorldLimits', [912500 913000], 'ColumnsStartFrom','north', ...
        'XWorldLimits', [207000 208000])

R =
  MapCellsReference with properties:

            XWorldLimits: [207000 208000]
            YWorldLimits: [912500 913000]
              RasterSize: [1000 2000]
    RasterInterpretation: 'cells'
        ColumnsStartFrom: 'north'
           RowsStartFrom: 'west'
       CellExtentInWorldX: 1/2
       CellExtentInWorldY: 1/2
     RasterExtentInWorldX: 1000
     RasterExtentInWorldY: 500
         XIntrinsicLimits: [0.5 2000.5]
         YIntrinsicLimits: [0.5 1000.5]
      TransformationType: 'rectilinear'
     CoordinateSystemType: 'planar'
             ProjectedCRS: []
```

Construct Default Raster Reference Object and Set Fields

Create a default raster reference object.

```
R = maprasterref

R =
  MapCellsReference with properties:
```

```
            XWorldLimits: [0.5 2.5]
            YWorldLimits: [0.5 2.5]
               RasterSize: [2 2]
      RasterInterpretation: 'cells'
         ColumnsStartFrom: 'south'
            RowsStartFrom: 'west'
       CellExtentInWorldX: 1
       CellExtentInWorldY: 1
     RasterExtentInWorldX: 2
     RasterExtentInWorldY: 2
          XIntrinsicLimits: [0.5 2.5]
          YIntrinsicLimits: [0.5 2.5]
       TransformationType: 'rectilinear'
     CoordinateSystemType: 'planar'
              ProjectedCRS: []
```

Set fields in the raster reference object.

```
R.XWorldLimits = [207000 208000];
R.YWorldLimits = [912500 913000];
R.ColumnsStartFrom = 'north';
R.RasterSize = [1000 2000]

R =
  MapCellsReference with properties:

            XWorldLimits: [207000 208000]
            YWorldLimits: [912500 913000]
               RasterSize: [1000 2000]
      RasterInterpretation: 'cells'
         ColumnsStartFrom: 'north'
            RowsStartFrom: 'west'
       CellExtentInWorldX: 1/2
       CellExtentInWorldY: 1/2
     RasterExtentInWorldX: 1000
     RasterExtentInWorldY: 500
          XIntrinsicLimits: [0.5 2000.5]
          YIntrinsicLimits: [0.5 1000.5]
       TransformationType: 'rectilinear'
     CoordinateSystemType: 'planar'
              ProjectedCRS: []
```

Construct Raster Reference Object Using World File Matrix

Create a world file matrix.

```
W = [0.5   0.0   207000.25; ...
     0.0  -0.5   912999.75];
```

Specify the size of the image.

```
rasterSize = [1000 2000];
```

Create the map raster reference object.

```
R = maprasterref(W, rasterSize)

R =
  MapCellsReference with properties:

             XWorldLimits: [207000 208000]
             YWorldLimits: [912500 913000]
               RasterSize: [1000 2000]
    RasterInterpretation: 'cells'
        ColumnsStartFrom: 'north'
           RowsStartFrom: 'west'
      CellExtentInWorldX: 1/2
      CellExtentInWorldY: 1/2
    RasterExtentInWorldX: 1000
    RasterExtentInWorldY: 500
         XIntrinsicLimits: [0.5 2000.5]
         YIntrinsicLimits: [0.5 1000.5]
       TransformationType: 'rectilinear'
     CoordinateSystemType: 'planar'
             ProjectedCRS: []
```

See Also

Functions
maprefcells | maprefpostings | georasterref | worldFileMatrix

Objects
MapCellsReference | MapPostingsReference

maprefcells

Reference raster cells to map coordinates

Syntax

```
R = maprefcells()
R = maprefcells(xlimits,ylimits,rasterSize)
R = maprefcells(xlimits,ylimits,xcellextent,ycellextent)
R = maprefcells(xlimits,ylimits, ___ ,Name,Value)
```

Description

`R = maprefcells()` returns a default referencing object for a regular raster of cells in planar (map) coordinates.

`R = maprefcells(xlimits,ylimits,rasterSize)` constructs a referencing object for a raster of cells spanning the specified limits in planar coordinates, with the numbers of rows and columns specified by `rasterSize`.

`R = maprefcells(xlimits,ylimits,xcellextent,ycellextent)` allows the cell extents to be set precisely. If necessary, `maprefcells` adjusts the limits of the raster slightly to ensure an integer number of cells in each dimension.

`R = maprefcells(xlimits,ylimits, ___ ,Name,Value)` allows the directions of the columns and rows to be specified via name-value pairs.

Examples

Construct Referencing Object with Raster Interpretation of Cells

Define latitude and longitude limits and dimensions of the image. The image follows the popular convention in which world *x* coordinates increase from column to column and world *y* coordinates decrease from row to row.

```
xlimits = [207000 208000];
ylimits = [912500 913000];
rasterSize = [1000 2000]

rasterSize = 1×2

        1000        2000
```

Create the referencing object specifying the raster size.

```
R = maprefcells(xlimits,ylimits,rasterSize, ...
    'ColumnsStartFrom','north')

R =
  MapCellsReference with properties:
```

```
        XWorldLimits: [207000 208000]
        YWorldLimits: [912500 913000]
          RasterSize: [1000 2000]
RasterInterpretation: 'cells'
   ColumnsStartFrom: 'north'
      RowsStartFrom: 'west'
  CellExtentInWorldX: 1/2
  CellExtentInWorldY: 1/2
RasterExtentInWorldX: 1000
RasterExtentInWorldY: 500
     XIntrinsicLimits: [0.5 2000.5]
     YIntrinsicLimits: [0.5 1000.5]
   TransformationType: 'rectilinear'
CoordinateSystemType: 'planar'
         ProjectedCRS: []
```

Obtain the same result by specifying the cell extents. For this example, the pixels are 1/2 meter square, referenced to a planar map coordinate system (the "world" system).

```
extent = 1/2;

R = maprefcells(xlimits,ylimits,extent,extent, ...
    'ColumnsStartFrom','north')

R =
  MapCellsReference with properties:

        XWorldLimits: [207000 208000]
        YWorldLimits: [912500 913000]
          RasterSize: [1000 2000]
RasterInterpretation: 'cells'
   ColumnsStartFrom: 'north'
      RowsStartFrom: 'west'
  CellExtentInWorldX: 1/2
  CellExtentInWorldY: 1/2
RasterExtentInWorldX: 1000
RasterExtentInWorldY: 500
     XIntrinsicLimits: [0.5 2000.5]
     YIntrinsicLimits: [0.5 1000.5]
   TransformationType: 'rectilinear'
CoordinateSystemType: 'planar'
         ProjectedCRS: []
```

Input Arguments

xlimits — Limits in the *x* direction
[0.5 2.5] (default) | 1-by-2 numeric vector

Limits in the *x* direction, specified as a 1-by-2 numeric vector. The value of `xlimits` determines the `XWorldLimits` property of R.

Example: `xlimits = [207000 208000];`

Data Types: double

ylimits — Limits in the *y* direction
[0.5 2.5] (default) | 1-by-2 numeric vector

Limits in the *y* direction, specified as a 1-by-2 numeric vector. The value of ylimits determines the YWorldLimits property of R.

Example: ylimits = [912500 913000];

Data Types: double

rasterSize — Size of the raster
[2 2] (default) | 1-by-2 numeric vector

Size of the raster, specified as a 1-by-2 numeric vector.

Example: rasterSize = [180 360];

Data Types: double

xcellextent — Width of cells
1 (default) | numeric scalar

Width of cells, specified as a numeric scalar. The value of xcellextent determines the CellExtentInWorldX property of R.

Example: xcellextent = 1.5

Data Types: double

ycellextent — Height of cells
1 (default) | numeric scalar

Height of cells, specified as a numeric scalar. The value of ycellextent determines the CellExtentInWorldY property of R.

Example: ycellextent = 1.5

Data Types: double

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: R = maprefcells(latlim,lonlim,rasterSize,'ColumnsStartFrom','north')

ColumnsStartFrom — Edge from which column indexing starts
'south' (default) | 'north'

Edge from which column indexing starts, specified as 'north' or 'south'.

Example: R = maprefcells(latlim,lonlim,rasterSize,'ColumnsStartFrom','north')

Data Types: char | string

RowsStartFrom — Edge from which row indexing starts
'west' (default) | 'east'

Edge from which column indexing starts, specified as `'west'` or `'east'`.

Example: R = maprefcells(latlim,lonlim,rasterSize,'RowsStartFrom','east')

Data Types: char | string

Output Arguments

R — Object that references raster cells to map coordinates
MapCellsReference raster reference object

Object that references raster cells to map coordinates, returned as a `MapCellsReference` raster reference object.

Tips

- To construct a map raster reference object from a world file matrix, use the `maprasterref` function.

See Also

maprefpostings | MapCellsReference | georefcells

Introduced in R2015b

maprefpostings

Reference raster postings to map coordinates

Syntax

```
R = maprefpostings()
R = maprefpostings(xlimits,ylimits,rasterSize)
R = maprefpostings(xlimits,ylimits,xspacing,yspacing)
R = maprefpostings(xlimits,ylimits, ___ , Name,Value)
```

Description

`R = maprefpostings()` returns a default referencing object for a raster of regularly posted samples in planar (map) coordinates.

`R = maprefpostings(xlimits,ylimits,rasterSize)` constructs a referencing object for a raster spanning the specified limits in planar coordinates, with the numbers of rows and columns specified by `rasterSize`.

`R = maprefpostings(xlimits,ylimits,xspacing,yspacing)` allows the sample spacings to be set precisely. If necessary, `maprefpostings` adjusts the limits of the raster slightly to ensure an integer number of samples in each dimension.

`R = maprefpostings(xlimits,ylimits, ___ , Name,Value)` allows the directions of the columns and rows to be specified via name-value pairs.

Examples

Construct Referencing Object for Grid

Define latitude and longitude limits and dimension of a grid. The example uses postings separated by 1/2 meter, referenced to a planar map coordinate system (the "world" system).

```
xlimits = [207000 208000];
ylimits = [912500 913000];
rasterSize = [1001 2001]

rasterSize = 1×2

      1001      2001
```

Create the referencing object specifying the raster size.

```
R = maprefpostings(xlimits,ylimits,rasterSize)

R =
  MapPostingsReference with properties:

          XWorldLimits: [207000 208000]
```

```
            YWorldLimits: [912500 913000]
              RasterSize: [1001 2001]
    RasterInterpretation: 'postings'
        ColumnsStartFrom: 'south'
           RowsStartFrom: 'west'
   SampleSpacingInWorldX: 1/2
   SampleSpacingInWorldY: 1/2
    RasterExtentInWorldX: 1000
    RasterExtentInWorldY: 500
         XIntrinsicLimits: [1 2001]
         YIntrinsicLimits: [1 1001]
      TransformationType: 'rectilinear'
     CoordinateSystemType: 'planar'
             ProjectedCRS: []
```

Obtain the same result by specifying the sample spacing.

```
spacing = 1/2;
```

```
R = maprefpostings(xlimits,ylimits,spacing,spacing)
```

```
R =
  MapPostingsReference with properties:

            XWorldLimits: [207000 208000]
            YWorldLimits: [912500 913000]
              RasterSize: [1001 2001]
    RasterInterpretation: 'postings'
        ColumnsStartFrom: 'south'
           RowsStartFrom: 'west'
   SampleSpacingInWorldX: 1/2
   SampleSpacingInWorldY: 1/2
    RasterExtentInWorldX: 1000
    RasterExtentInWorldY: 500
         XIntrinsicLimits: [1 2001]
         YIntrinsicLimits: [1 1001]
      TransformationType: 'rectilinear'
     CoordinateSystemType: 'planar'
             ProjectedCRS: []
```

Input Arguments

xlimits — Limits in the *x* direction
[0.5 2.5] (default) | 1-by-2 numeric vector

Limits in the *x* direction, specified as a 1-by-2 numeric vector. The value of `xlimits` determines the `XWorldLimits` property of R.

Example: `xlimits = [207000 208000];`

Data Types: `double`

ylimits — Limits in the *y* direction
[0.5 2.5] (default) | 1-by-2 numeric vector

Limits in the *y* direction, specified as a 1-by-2 numeric vector. The value of `ylimits` determines the `YWorldLimits` property of R.

Example: `ylimits = [912500 913000];`

Data Types: `double`

rasterSize — Size of the raster
[2 2] (default) | 1-by-2 numeric vector

Size of the raster, specified as a 1-by-2 numeric vector.

Example: `rasterSize = [180 360];`

Data Types: `double`

xspacing — Horizontal spacing of posting
1 (default) | numeric scalar

Horizontal spacing of posting, specified as a numeric scalar. The value of `xspacing` determines the `SampleSpacingInWorldX` property of R.

Example: `xspacing = 1.5`

Data Types: `double`

yspacing — Vertical spacing of postings
1 (default) | numeric scalar

Vertical spacing of postings, specified as a numeric scalar. The value of `yspacing` determines the `SampleSpacingInWorldY` property of R.

Example: `yspacing = 1.5`

Data Types: `double`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `R = maprefpostings(latlim,lonlim,rasterSize,'ColumnsStartFrom','north')`

ColumnsStartFrom — Edge from which column indexing starts
'south' (default) | 'north'

Edge from which column indexing starts, specified as `'north'` or `'south'`.

Example: `R = maprefpostings(latlim,lonlim,rasterSize,'ColumnsStartFrom','north')`

Data Types: `char` | `string`

RowsStartFrom — Edge from which row indexing starts
'west' (default) | 'east'

Edge from which row indexing starts, specified as `'east'` or `'west'`.

Example: R = maprefpostings(latlim,lonlim,rasterSize,'RowsStartFrom','east')

Data Types: char | string

Output Arguments

R — Object that references raster postings to map coordinates
MapPostingsReference raster reference object

Object that references raster postings to map coordinates, returned as a `MapPostingsReference` raster reference object.

Tips

- To construct a map raster reference object from a world file matrix, use the `maprasterref` function.

See Also

maprefcells | MapPostingsReference | georefpostings

Introduced in R2015b

mapresize

Resize projected raster

Syntax

```
[B,RB] = mapresize(A,RA,scale)
[B,RB] = mapresize( ___ ,method)
[B,RB] = mapresize( ___ ,'Antialiasing',TF)
```

Description

`[B,RB] = mapresize(A,RA,scale)` returns a raster B that is `scale` times the size of raster A. RA is a raster reference object that specifies the location and extent of data in A. `mapresize` returns the raster reference object RB that is associated with the returned raster B. By default, `mapresize` uses cubic interpolation.

`[B,RB] = mapresize( ___ ,method)` returns a resized raster where `method` specifies the interpolation method.

`[B,RB] = mapresize( ___ ,'Antialiasing',TF)` specifies whether to perform antialiasing when shrinking a raster. When `true`, `mapresize` performs antialiasing. The default value depends on the type of interpolation specified. For nearest-neighbor interpolation, the default value is `false`. For all other interpolation methods, the default is `true`.

Examples

Resize Projected Raster

Import a sample projected raster and map cells reference object.

```
[Z,R] = readgeoraster('map_sample.tif');
```

Resize the raster using `mapresize`. Double the length and width of the raster by specifying the scale as 2. Use nearest neighbor interpolation by specifying the interpolation method as `'nearest'`.

```
[Z2,R2] = mapresize(Z,R,2,'nearest');
```

Verify the raster has been resized by comparing the size of the original raster with the size of the updated raster.

```
R.RasterSize
```

ans = *1×2*

```
     2      2
```

```
R2.RasterSize
```

ans = *1×2*

```
        4       4
```

If the rasters are small, you can compare them directly.

Z

Z = 2×2

```
        1       2
        3       4
```

Z2

Z2 = 4×4

```
        1       1       2       2
        1       1       2       2
        3       3       4       4
        3       3       4       4
```

Resize Projected Raster Data Set

Read a projected raster data set and map cells reference object into the workspace.

```
[boston,R] = readgeoraster('boston.tif');
```

Display the raster with mapshow.

```
mapshow(boston,R)
```
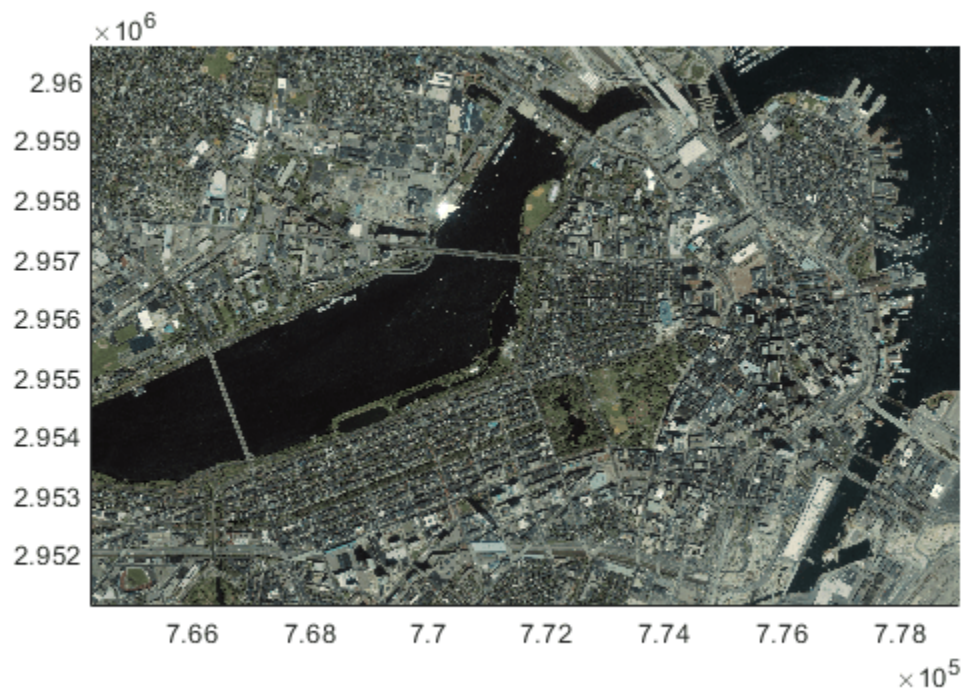
Resize the projected raster data set. For this example, reduce the raster to one sixteenth of the original size.

```
[resizedBoston,resizedR] = mapresize(boston,R,1/16);
```

Display the resized raster. Note that `mapshow` preserves the original limits of the map in the display so that, at first glance, the resized raster appears to be the same size as the original. A closer look reveals that the size of pixels in the resized raster are larger than the pixels in the original.

```
figure
mapshow(resizedBoston,resizedR)
```
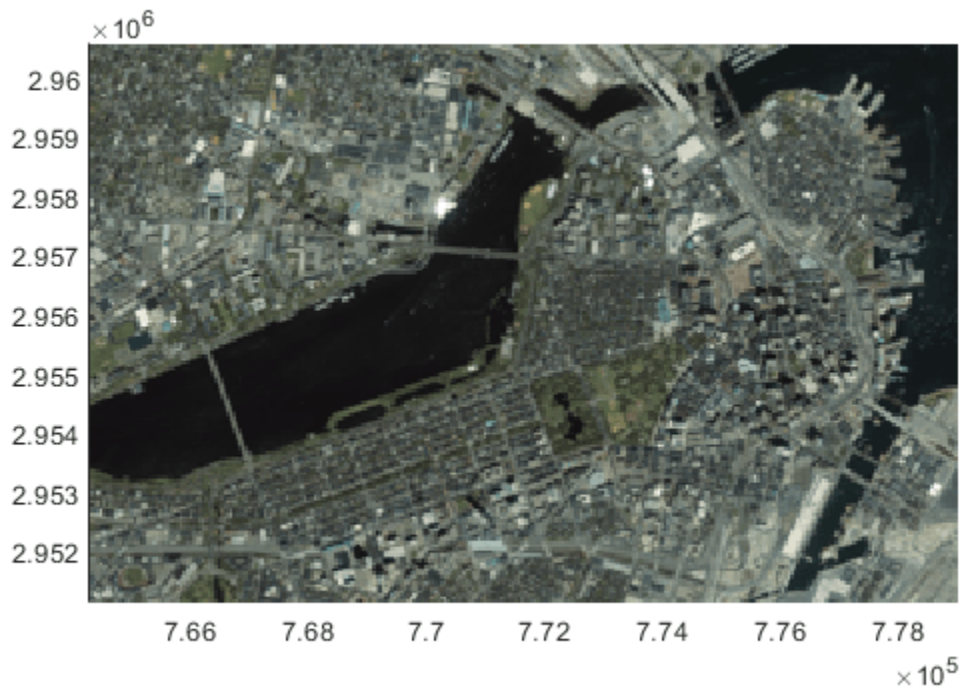
Input Arguments

A — Raster to be resized
numeric or logical array

Raster to be resized, specified as a numeric or logical array. If A has more than two dimensions, `mapresize` only resizes the first two dimensions.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

RA — Information about location and extent of raster
map raster reference object

Information about location and extent of raster, specified as a map raster reference object. To convert a raster matrix into a map raster reference object, use the `refmatToMapRasterReference` function.

scale — Amount of resizing
numeric scalar

Amount of resizing, specified as numeric scalar. If `scale` is in the range `[0 1]`, B is smaller than A. If `scale` is greater than 1, B is larger than A.

Example: `0.5`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

method — Interpolation method
cubic (default) | `'nearest'` | `'bilinear'`

Interpolation method, specified as one of the following values.

Value	Description
`'nearest'`	Nearest-neighbor interpolation
`'bilinear'`	Bilinear interpolation
`'cubic'`	Cubic interpolation

Data Types: `char` | `string`

Output Arguments

B — Resized raster
numeric or logical array

Resized raster, returned as a numeric or logical array.

RB — Information about location and extent of raster
map raster reference object

Information about location and extent of raster, returned as a map raster reference object.

Tips

- Use `mapresize` with raster data in *x*- and *y*-coordinates. To work with geographic raster data in latitude and longitude coordinates, use `georesize`.

See Also
`georesize` | `maprefcells` | `mapinterp` | `maprefpostings`

Introduced in R2019a

maps

List map projections for map axes and map projection structures

Syntax

```
strmat = maps('namelist')
strmat = maps('idlist')
stdstr = maps('proj_id_abbrev')
```

Description

`maps` displays in the Command Window a table describing all projections available for use with map axes and map projection structures.

`strmat = maps('namelist')` returns the English names for the available projections as a matrix of character vectors.

`strmat = maps('idlist')` returns the standard projection identifiers for the available projections as a matrix of character vectors.

`stdstr = maps('proj_id_abbrev')` returns the specific standard projection identification associated with a unique abbreviation.

Examples

To show the first five entries of the projections name list,

```
str1 = maps('namelist');
str1(1:5,:)
ans =
Balthasart Cylindrical
Behrmann Cylindrical
Bolshoi Sovietskii Atlas Mira
Braun Perspective Cylindrical
Cassini Cylindrical
```

The corresponding shorthand names are

```
str2 = maps('idlist');
str2(1:5,:)
ans =
balthsrt
behrmann
bsam
braun
cassini
```

These shorthand names can be used, for example, when setting the `axesm` property `MapProjection`.

The functions `setm` and `axesm` recognize unique abbreviations (truncations) of these names. The `maps` function can be used to convert such an abbreviation to the standard ID:

```
stdstr = maps('merc')
stdstr =
mercator
```

When the function name alone is used,

```
maps
```

```
MapTools Projections
CLASS              NAME                             ID STRING
Cylindrical        Balthasart Cylindrical           balthsrt
Cylindrical        Behrmann Cylindrical             behrmann
Cylindrical        Bolshoi Sovietskii Atlas Mira*   bsam
Cylindrical        Braun Perspective Cylindrical*   braun
Cylindrical        Cassini Cylindrical              cassini
Cylindrical        Central Cylindrical*             ccylin
Cylindrical        Equal Area Cylindrical           eqacylin
Cylindrical        Equidistant Cylindrical          eqdcylin
Cylindrical        Gall Isographic                  giso...
```

The actual result contains all defined projections.

See Also
axesm | setm

mapshape

Planar shape vector

Description

A mapshape vector is an object that represents planar vector features with either point, line, or polygon topology. The features consist of *x*- and *y*-coordinates and associated attributes.

Attributes that vary spatially are termed Vertex properties. These elements of the mapshape vector are coupled such that the length of the *x*- and *y*-coordinate property values are always equal in length to any additional dynamic Vertex properties.

Attributes that only pertain to the overall feature (point, line, polygon) are termed Feature properties. Feature properties are not linked to the autosizing mechanism of the Vertex properties. Both property types can be added to a mapshape vector after construction using standard dot (.) notation.

To create a planar point, line, or polygon shape for use with a geospatial table, create a `mappointshape`, `maplineshape`, or `mappolyshape` object instead.

Creation

Syntax

```
s = mapshape()
s = mapshape(x,y)
s = mapshape(x,y,Name,Value)
s = mapshape(structArray)
s = mapshape(x,y,structArray)
```

Description

`s = mapshape()` constructs an empty mapshape vector, `s`, with these default property settings.

```
s =

 0x1 mapshape vector with properties:

 Collection properties:
    Geometry: 'line'
    Metadata: [1x1 struct]
 Vertex properties:
           X: []
           Y: []
```

`s` is always a column vector.

`s = mapshape(x,y)` constructs a mapshape vector and sets the X and Y property values equal to vectors `x` and `y`.

s = mapshape(x,y,Name,Value) constructs a mapshape vector, then adds dynamic properties to the mapshape vector using `Name`, `Value` argument pairs. You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

s = mapshape(structArray) constructs a `mapshape` vector, assigning the fields of the structure array, `structArray`, as dynamic properties. Field values in `structArray` that are not numeric, logical, string scalars, string arrays, character vectors, cell arrays of character vectors, or cell arrays of numeric, logical, or cell array of character vectors values are ignored. You can specify vectors within cell arrays as either row or column vectors.

s = mapshape(x,y,structArray) constructs a new `mapshape` vector, sets the X and Y properties equal to vectors x and y, and sets dynamic properties from the field values of `structArray`.

Properties

`mapshape` class is a general class that represents a variety of planar features. The class permits features to have more than one vertex and can thus represent lines and polygons in addition to multipoints. For more about the property types in `mapshape`, see "Collection Properties" on page 1-960, "Vertex Properties" on page 1-960, and "Feature Properties" on page 1-960.

Dynamic properties are new features and vertices that are added to a mapshape vector. You can attach dynamic Feature and Vertex properties to a mapshape vector during construction with a `Name,Value` pair or after construction using dot (.) notation after construction. This is similar to adding new fields to a structure. For an example of adding dynamic Feature properties, see "Construct a Mapshape Vector with Dynamic Properties" on page 1-947.

Geometry — Shape of all the features in the mapshape vector
`'line'` (default) | `'point'` | `'polygon'`

Shape of all the features in the mapshape vector, specified as `'line'`, `'point'`, or `'polygon'`. As a Collection Property there can be only one value per object instance and its purpose is purely informational. The three allowable values for `Geometry` do not change class behavior. The class does not provide validation for line or polygon topologies.

Data Types: `char` | `string`

Metadata — Information for the entire set of features
scalar structure

Information for all the features, specified as a scalar structure. You can add any data type to the structure. As a Collection Property type, only one instance per object is allowed.

- If `Metadata` is provided as a dynamic property `Name` in the constructor, and the corresponding `Value` is a scalar structure, then `Value` is copied to the `Metadata` property. Otherwise, an error is issued.

- If a `Metadata` field is provided by `structArray`, and both `Metadata` and `structArray` are scalar structures, then the `Metadata` field value is copied to the `Metadata` property value. If `structArray` is a scalar but the `Metadata` field is not a structure, then an error is issued. If `structArray` is not scalar, then the `Metadata` field is ignored.

Data Types: `struct`

X — Planar *x*-coordinates
numeric row or column vector

Planar *x*-coordinates, specified as a numeric row or column vector, stored as a row vector.

Data Types: `double` | `single`

Y — Planar *y*-coordinates
numeric row or column vector

Planar *y*-coordinates, specified as a numeric row or column vector, stored as a row vector.

Data Types: `double` | `single`

Object Functions

append	Append features to geographic or planar vector
cat	Concatenate geographic or planar vector
disp	Display geographic or planar vector
fieldnames	Return dynamic property names of geographic or planar vector
isempty	Determine if geographic or planar vector is empty
isfield	Determine if dynamic property exists in geographic or planar vector
isprop	Determine if property exists in geographic or planar vector
length	Return number of elements in geographic or planar vector
properties	Return property names of geographic or planar vector
rmfield	Remove dynamic property from geographic or planar vector
rmprop	Remove property from geographic or planar vector
size	Return size of geographic or planar vector
struct	Convert geographic or planar vector to scalar structure
vertcat	Vertically concatenate geographic or planar vectors

Examples

Construct a Default Mapshape Vector and Set and Add Properties

Create default mapshape vector.

```
s = mapshape()

0x1 mapshape vector with properties:

 Collection properties:
    Geometry: 'line'
    Metadata: [1x1 struct]
 Vertex properties:
           X: []
           Y: []
```

Set the values of the existing X and Y properties and dynamically add the Vertex property Z.

```
s(1).X = 0:45:90;
s(1).Y= [10 10 10];
s(1).Z = [10 20 30]

s =

 1x1 mapshape vector with properties:
```

```
Collection properties:
    Geometry: 'line'
    Metadata: [1x1 struct]
Vertex properties:
          X: [0 45 90]
          Y: [10 10 10]
          Z: [10 20 30]
```

Construct a Mapshape Vector Specifying X and Y Values

Create a mapshape vector specifying x and y.

```
x = [40, 50, 60];
y = [10, 20, 30];
shape = mapshape(x, y)

shape =

 1x1 mapshape vector with properties:

 Collection properties:
    Geometry: 'line'
    Metadata: [1x1 struct]
 Vertex properties:
          X: [40 50 60]
          Y: [10 20 30]
```

Construct a Mapshape Vector with Dynamic Properties

Create mapshape vector specifying a Name-Value pair.

```
x = 1:10;
y = 21:30;
temperature = {61:70};
shape = mapshape(x, y, 'Temperature', temperature)

shape =

 1x1 mapshape vector with properties:

 Collection properties:
      Geometry: 'line'
      Metadata: [1x1 struct]
 Vertex properties:
             X: [1 2 3 4 5 6 7 8 9 10]
             Y: [21 22 23 24 25 26 27 28 29 30]
   Temperature: [61 62 63 64 65 66 67 68 69 70]
```

When `Value` is a cell array containing numeric, logical, or cell array of character vectors, it is designated as a Vertex property. Otherwise the Name-Value pair is designated as being a Feature property.

Construct a Mapshape Vector from a Structure Array

Create structure array and then create mapshape vector with array.

```
structArray = shaperead('concord_roads');
shape = mapshape(structArray)
```

```
shape =

 609x1 mapshape vector with properties:

 Collection properties:
      Geometry: 'line'
      Metadata: [1x1 struct]
 Vertex properties:
  (609 features concatenated with 608 delimiters)
             X: [1x5422 double]
             Y: [1x5422 double]
 Feature properties:
    STREETNAME: {1x609 cell}
     RT_NUMBER: {1x609 cell}
         CLASS: [1x609 double]
    ADMIN_TYPE: [1x609 double]
        LENGTH: [1x609 double]
```

Construct a Mapshape Vector Using Cell Arrays and Structures to Define Multiple Features and Properties

Read data from a shapefile into a structure.

```
[structArray, A] = shaperead('concord_hydro_area');
```

```
structArray =

98x1 struct array with fields:

    Geometry
    BoundingBox
    X
    Y
```

```
A =

98x1 struct array with fields:

    AREA
    PERIMETER
```

Create a mapshape vector specifying the structure.

```
shape = mapshape({structArray.X}, {structArray.Y}, A);
shape.Geometry = structArray(1).Geometry
```

```
shape =

 98x1 mapshape vector with properties:
```

```
Collection properties:
    Geometry: 'polygon'
    Metadata: [1x1 struct]
Vertex properties:
  (98 features concatenated with 97 delimiters)
          X: [1x4902 double]
          Y: [1x4902 double]
Feature properties:
        AREA: [1x98 double]
   PERIMETER: [1x98 double]
```

Construct a Mapshape Vector and Add a Feature Property

This example shows how to add a single feature after construction of the mapshape vector using dot (.) notation.

Create a mapshape vector.

```
x = 0:10:100;
y = 0:10:100;
shape = mapshape(x, y)

shape =

 1x1 mapshape vector with properties:

 Collection properties:
    Geometry: 'line'
    Metadata: [1x1 struct]
 Vertex properties:
          X: [0 10 20 30 40 50 60 70 80 90 100]
          Y: [0 10 20 30 40 50 60 70 80 90 100]
```

Add a dynamic Feature property.

```
shape.FeatureName = 'My Feature'

shape =

 1x1 mapshape vector with properties:

 Collection properties:
      Geometry: 'line'
      Metadata: [1x1 struct]
 Vertex properties:
             X: [0 10 20 30 40 50 60 70 80 90 100]
             Y: [0 10 20 30 40 50 60 70 80 90 100]
 Feature properties:
    FeatureName: 'My Feature'
```

Add a dynamic Vertex property to the first feature.

```
shape(1).Temperature = [60 61 63 65 66 68 69 70 72 75 80];

shape =
```

```
  1x1 mapshape vector with properties:

Collection properties:
      Geometry: 'line'
      Metadata: [1x1 struct]
Vertex properties:
             X: [0 10 20 30 40 50 60 70 80 90 100]
             Y: [0 10 20 30 40 50 60 70 80 90 100]
   Temperature: [60 61 63 65 66 68 69 70 72 75 80]
Feature properties:
   FeatureName: 'My Feature
```

Construct a Mapshape Vector and Manipulate Features

This extended example adds multiple features that are both Vertex and Feature properties. It also demonstrates property behaviors when vector lengths are either changed or set to [ ].

Create a mapshape vector.

```
x = {1:3, 4:6};
y = {[0 0 0], [1 1 1]};
shape = mapshape(x, y)
```

```
shape =

 2x1 mapshape vector with properties:

Collection properties:
   Geometry: 'line'
   Metadata: [1x1 struct]
Vertex properties:
 (2 features concatenated with 1 delimiter)
        X: [1 2 3 NaN 4 5 6]
        Y: [0 0 0 NaN 1 1 1]
```

Add a two element dynamic Feature property.

```
shape.FeatureName = {'Feature 1', 'Feature 2'}
```

```
shape =

 2x1 mapshape vector with properties:

Collection properties:
      Geometry: 'line'
      Metadata: [1x1 struct]
Vertex properties:
 (2 features concatenated with 1 delimiter)
            X: [1 2 3 NaN 4 5 6]
            Y: [0 0 0 NaN 1 1 1]
Feature properties:
   FeatureName: {'Feature 1' 'Feature 2'}
```

Add a dynamic Vertex property.

```
z = {101:103, [115, 114, 110]}
shape.Z = z

z =

    [1x3 double]    [1x3 double]


shape =

 2x1 mapshape vector with properties:

 Collection properties:
      Geometry: 'line'
      Metadata: [1x1 struct]
 Vertex properties:
  (2 features concatenated with 1 delimiter)
             X: [1 2 3 NaN 4 5 6]
             Y: [0 0 0 NaN 1 1 1]
             Z: [101 102 103 NaN 115 114 110]
 Feature properties:
    FeatureName: {'Feature 1' 'Feature 2'}
```

Display the second feature.

```
shape(2)
```

```
ans =

 1x1 mapshape vector with properties:

 Collection properties:
      Geometry: 'line'
      Metadata: [1x1 struct]
 Vertex properties:
             X: [4 5 6]
             Y: [1 1 1]
             Z: [115 114 110]
 Feature properties:
    FeatureName: 'Feature 2'
```

Add a third feature. The lengths of all the properties are synchronized.

```
shape(3).X = 5:9
```

```
shape =

 3x1 mapshape vector with properties:

 Collection properties:
      Geometry: 'line'
      Metadata: [1x1 struct]
 Vertex properties:
  (3 features concatenated with 2 delimiters)
             X: [1 2 3 NaN 4 5 6 NaN 5 6 7 8 9]
             Y: [0 0 0 NaN 1 1 1 NaN 0 0 0 0 0]
             Z: [101 102 103 NaN 115 114 110 NaN 0 0 0 0 0]
 Feature properties:
    FeatureName: {'Feature 1' 'Feature 2' ''}
```

Set the values for the Z vertex property with fewer values than contained in X or Y. The Z values expand to match the length of X and Y.

```
shape(3).Z = 1:3

shape =

 3x1 mapshape vector with properties:

 Collection properties:
      Geometry: 'line'
      Metadata: [1x1 struct]
 Vertex properties:
  (3 features concatenated with 2 delimiters)
             X: [1 2 3 NaN 4 5 6 NaN 5 6 7 8 9]
             Y: [0 0 0 NaN 1 1 1 NaN 0 0 0 0 0]
             Z: [101 102 103 NaN 115 114 110 NaN 1 2 3 0 0]
 Feature properties:
    FeatureName: {'Feature 1' 'Feature 2' ''}
```

Set the values for either coordinate property (X or Y) and all properties shrink in size to match the new vertex length of that feature.

```
shape(3).Y = 1

shape =

 3x1 mapshape vector with properties:

 Collection properties:
      Geometry: 'line'
      Metadata: [1x1 struct]
 Vertex properties:
  (3 features concatenated with 2 delimiters)
             X: [1 2 3 NaN 4 5 6 NaN 5]
             Y: [0 0 0 NaN 1 1 1 NaN 1]
             Z: [101 102 103 NaN 115 114 110 NaN 1]
 Feature properties:
    FeatureName: {'Feature 1' 'Feature 2' ''}
```

Set the values for the Z vertex property with more values than contained in X or Y. All properties expand in length to match Z.

```
shape(3).Z = 1:6

shape =

 3x1 mapshape vector with properties:

 Collection properties:
      Geometry: 'line'
      Metadata: [1x1 struct]
 Vertex properties:
  (3 features concatenated with 2 delimiters)
             X: [1 2 3 NaN 4 5 6 NaN 5 0 0 0 0 0]
             Y: [0 0 0 NaN 1 1 1 NaN 1 0 0 0 0 0]
             Z: [101 102 103 NaN 115 114 110 NaN 1 2 3 4 5 6]
 Feature properties:
    FeatureName: {'Feature 1' 'Feature 2' ''}
```

Remove the `FeatureName` property.

```
shape.FeatureName = []

shape =

 3x1 mapshape vector with properties:

 Collection properties:
    Geometry: 'line'
    Metadata: [1x1 struct]
 Vertex properties:
  (3 features concatenated with 2 delimiters)
          X: [1 2 3 NaN 4 5 6 NaN 5 0 0 0 0 0]
          Y: [0 0 0 NaN 1 1 1 NaN 1 0 0 0 0 0]
          Z: [101 102 103 NaN 115 114 110 NaN 1 2 3 4 5 6]
```

Remove all dynamic properties and set the object to empty.

```
shape.X = []

shape =

 0x1 mapshape vector with properties:

 Collection properties:
    Geometry: 'line'
    Metadata: [1x1 struct]
 Vertex properties:
          X: []
          Y: []
```

Construct a Mapshape Vector Specifying Several Name-Value Pairs

This example shows how to include multiple dynamic features during object construction.

Create a mapshape vector specifying several name-value pairs.

```
x = {1:3, 4:6};
y = {[0 0 0], [1 1 1]};
z = {41:43, [56 50 59]};
name = {'Feature 1', 'Feature 2'};
id = [1 2];
shape = mapshape(x, y, 'Z', z, 'Name', name, 'ID', id)

shape =

 2x1 mapshape vector with properties:

 Collection properties:
    Geometry: 'line'
    Metadata: [1x1 struct]
 Vertex properties:
  (2 features concatenated with 1 delimiter)
          X: [1 2 3 NaN 4 5 6]
          Y: [0 0 0 NaN 1 1 1]
          Z: [41 42 43 NaN 56 50 59]
```

```
Feature properties:
      Name: {'Feature 1' 'Feature 2'}
        ID: [1 2]
```

Construct a Mapshape Vector Containing Multiple Features and Indexing Behaviors

Load the data and create x, y, and z arrays. Create a level list to use to bin the z values.

```
seamount = load('seamount');
x = seamount.x; y = seamount.y; z = seamount.z;

levels = [unique(floor(seamount.z/1000)) * 1000; 0];
```

Construct a mapshape object and assign the X and Y Vertex properties to the binned x and y values. Create a new Z Vertex property to contain the binned z values. Add a `Levels` Feature property to contain the lowest level value per feature.

```
shape = mapshape;
for k = 1:length(levels) - 1
    index = z >= levels(k) & z < levels(k+1);
    shape(k).X = x(index);
    shape(k).Y = y(index);
    shape(k).Z = z(index);
    shape(k).Level = levels(k);
end
```

Add a `Color` Feature property to denote a color for that feature, and specify that the geometry is `'point'`

```
shape.Color = {'red', 'green', 'blue', 'cyan', 'black'};
shape.Geometry = 'point'

shape =

 5x1 mapshape vector with properties:

 Collection properties:
    Geometry: 'point'
    Metadata: [1x1 struct]
 Vertex properties:
  (5 features concatenated with 4 delimiters)
          X: [1x298 double]
          Y: [1x298 double]
          Z: [1x298 double]
 Feature properties:
      Level: [-5000 -4000 -3000 -2000 -1000]
      Color: {'red' 'green' 'blue' 'cyan' 'black'}
```

Add metadata information. `Metadata` is a scalar structure containing information for the entire set of properties. Any type of data may be added to the structure.

```
shape.Metadata.Caption = seamount.caption;
shape.Metadata

ans =

    Caption: [1x229 char]
```

Display the point data in 2-D.

```
figure
for k=1:length(shape)
  mapshow(shape(k).X, shape(k).Y, ...
    'MarkerEdgeColor', shape(k).Color, ...
    'Marker', 'o', ...
    'DisplayType', shape.Geometry)
end
legend(num2str(shape.Level'))
```
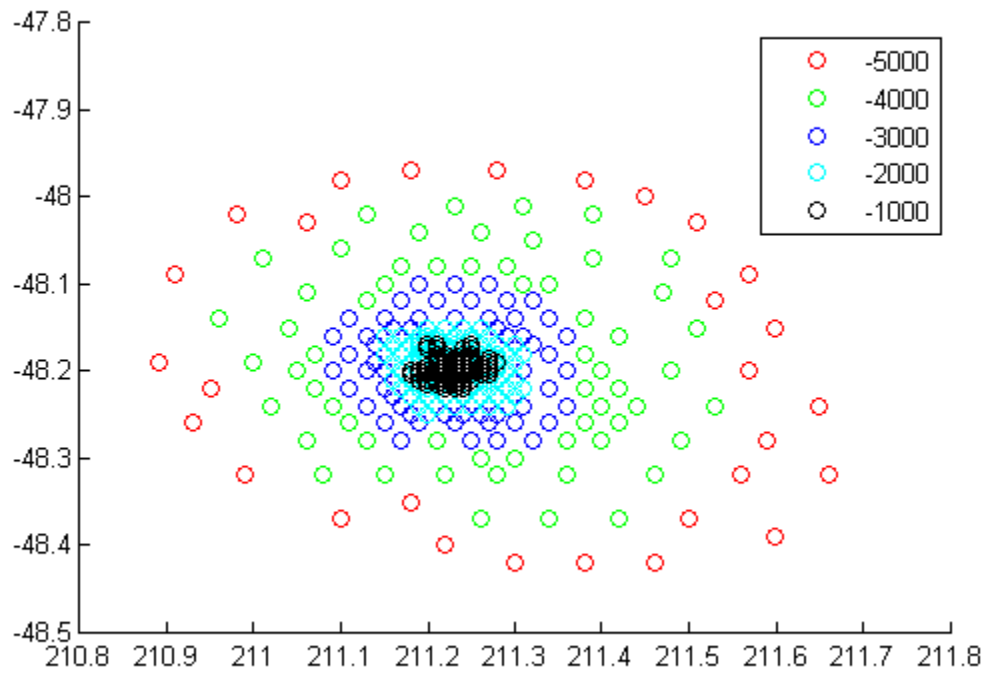

Display data as a 3-D scatter plot.

```
figure
scatter3(shape.X, shape.Y, shape.Z)
```
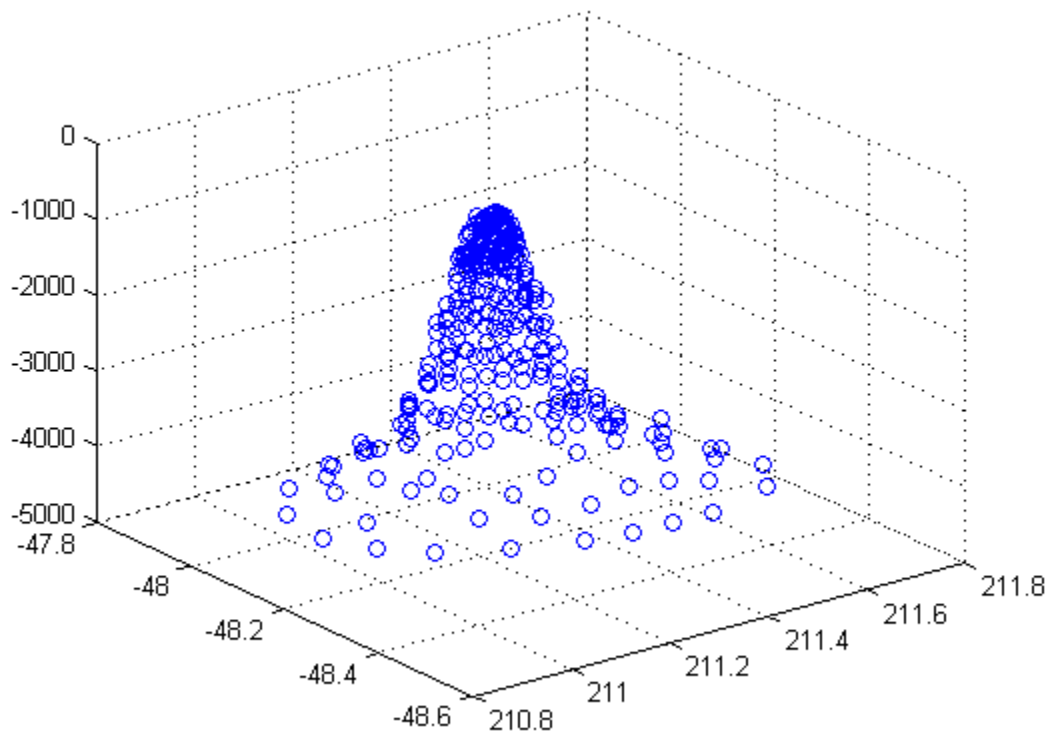
Construct a Mapshape Vector and Add Metadata and Indexing

This example shows how to use selective indexing behavior of a mapshape vector, and how to add a
`Metadata` property.

Construct a mapshape vector from a structure array

```
filename = 'concord_roads.shp';
S = shaperead(filename);
shape = mapshape(S)
```

```
shape =

 609x1 mapshape vector with properties:

 Collection properties:
     Geometry: 'line'
     Metadata: [1x1 struct]
 Vertex properties:
  (609 features concatenated with 608 delimiters)
           X: [1x5422 double]
           Y: [1x5422 double]
 Feature properties:
    STREETNAME: {1x609 cell}
     RT_NUMBER: {1x609 cell}
         CLASS: [1x609 double]
```

```
    ADMIN_TYPE: [1x609 double]
        LENGTH: [1x609 double]
```

Add a `Filename` field to the `Metadata` structure and then construct a new mapshape object with only CLASS 4 (major road) designation.

```
shape.Metadata.Filename = filename;
class4 = shape(shape.CLASS == 4)

class4 =

 26x1 mapshape vector with properties:

 Collection properties:
     Geometry: 'line'
     Metadata: [1x1 struct]
 Vertex properties:
  (26 features concatenated with 25 delimiters)
            X: [1x171 double]
            Y: [1x171 double]
 Feature properties:
    STREETNAME: {1x26 cell}
     RT_NUMBER: {1x26 cell}
         CLASS: [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
    ADMIN_TYPE: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
        LENGTH: [1x26 double]
```

Construct a Mapshape Vector and Sort the Dynamic Properties

This example show how features can be sorted by using the indexing behavior of the mapshape class.

You can create a new mapshape vector that contains a subset of dynamic properties by adding the name of a property or a cell array of property names to the last index in the ( ) operator.

Read data from file directly in mapshape constructor.

```
shape = mapshape(shaperead('tsunamis'))

shape =

 162x1 mapshape vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
 Vertex properties:
  (162 features concatenated with 161 delimiters)
            X: [1x323 double]
            Y: [1x323 double]
 Feature properties:
         Cause: {1x162 cell}
    Cause_Code: [1x162 double]
       Country: {1x162 cell}
           Day: [1x162 double]
   Desc_Deaths: [1x162 double]
         Eq_Mag: [1x162 double]
```

```
          Hour: [1x162 double]
      Iida_Mag: [1x162 double]
     Intensity: [1x162 double]
      Location: {1x162 cell}
    Max_Height: [1x162 double]
        Minute: [1x162 double]
         Month: [1x162 double]
    Num_Deaths: [1x162 double]
        Second: [1x162 double]
      Val_Code: [1x162 double]
      Validity: {1x162 cell}
          Year: [1x162 double]
```

Alphabetize the Feature properties.

```
shape = shape(:, sort(fieldnames(shape)))

shape =

 162x1 mapshape vector with properties:

 Collection properties:
      Geometry: 'point'
      Metadata: [1x1 struct]
 Vertex properties:
  (162 features concatenated with 161 delimiters)
             X: [1x323 double]
             Y: [1x323 double]
 Feature properties:
         Cause: {1x162 cell}
    Cause_Code: [1x162 double]
       Country: {1x162 cell}
           Day: [1x162 double]
   Desc_Deaths: [1x162 double]
        Eq_Mag: [1x162 double]
          Hour: [1x162 double]
      Iida_Mag: [1x162 double]
     Intensity: [1x162 double]
      Location: {1x162 cell}
    Max_Height: [1x162 double]
        Minute: [1x162 double]
         Month: [1x162 double]
    Num_Deaths: [1x162 double]
        Second: [1x162 double]
      Val_Code: [1x162 double]
      Validity: {1x162 cell}
          Year: [1x162 double]
```

Modify the mapshape vector to contain only the specified dynamic properties.

```
shape = shape(:, {'Year', 'Month', 'Day', 'Hour', 'Minute'})

shape =

 162x1 mapshape vector with properties:

 Collection properties:
    Geometry: 'point'
    Metadata: [1x1 struct]
```

```
Vertex properties:
  (162 features concatenated with 161 delimiters)
          X: [1x323 double]
          Y: [1x323 double]
Feature properties:
       Year: [1x162 double]
      Month: [1x162 double]
        Day: [1x162 double]
       Hour: [1x162 double]
     Minute: [1x162 double]
```

Create a new mapshape vector in which each feature contains the points for the same year. Copy the data from a mappoint vector to ensure that NaN feature separators are not included. Create a subsection of data to include only Year and Country dynamic properties.

```matlab
points = mappoint(shaperead('tsunamis'));
points = points(:, {'Year', 'Country'});
years = unique(points.Year);
multipoint = mapshape();
multipoint.Geometry = 'point';
for k = 1:length(years)
   index = points.Year == years(k);
   multipoint(k).X = points(index).X;
   multipoint(k).Y = points(index).Y;
   multipoint(k).Year = years(k);
   multipoint(k).Country = points(index).Country;
end
multipoint             % Display
```

```
multipoint =

 53x1 mapshape vector with properties:

 Collection properties:
    Geometry: 'point'
    Metadata: [1x1 struct]
 Vertex properties:
  (53 features concatenated with 52 delimiters)
          X: [1x214 double]
          Y: [1x214 double]
    Country: {1x214 cell}
 Feature properties:
       Year: [1x53 double]
```

Display the third from the end feature.

```
multipoint(end-3)
```

```
ans =

 1x1 mapshape vector with properties:

 Collection properties:
    Geometry: 'point'
    Metadata: [1x1 struct]
 Vertex properties:
          X: [3.6340 -62.1800 143.9100]
          Y: [36.9640 16.7220 41.8150]
```

```
        Country: {'ALGERIA' 'MONTSERRAT' 'JAPAN'}
     Feature properties:
             Year: 2003
```

More About

Collection Properties

Collection properties contain only one value per class instance. In contrast, the Feature and Vertex property types have attribute values associated with each feature or with each vertex in a set that defines a feature. `Geometry` and `Metadata` are the only two Collection properties.

Vertex Properties

Vertex properties provide a scalar number or a character vector for each vertex in a mapshape object. Vertex properties are suitable for attributes that vary spatially from point to point (vertex to vertex) along a line. Examples of such spatially varying attributes could be elevation, speed, temperature, or time. X and Y are Vertex properties since they contain a scalar number for each vertex in a mapshape vector.

Attribute values are associated with each vertex during construction or by using dot notation after construction. This process is similar to adding dynamic fields to a structure. Dynamic Vertex property values of an individual feature match its X and Y values in length.

Feature Properties

Feature properties provide one value (a scalar number, string scalar, or a character vector) for each feature in a mapshape vector. They are suitable for properties, such as name, owner, serial number, or age, that describe a given feature (an element of a mapshape vector) as a whole. Like Vertex properties, Feature properties can be added during construction or by using dot notation after construction.

Tips

- The `mapshape` function separates features using `NaN` values. If you display a feature by using a scalar to index into the mapshape vector, such as `s(1)`, then `NaN` values that separate the features do not display.

- If X, Y, or a dynamic property is set with more values than features in the mapshape vector, then all other properties expand in size using 0 for numeric values and an empty character vector (`' '`) for cell values.

- If a dynamic property is set with fewer values than the number of features, then this dynamic property expands to match the size of the other properties.

- If the X or Y property of the mapshape vector is set with fewer values than contained in the object, then all other properties shrink in size.

- If either X or Y is set to [ ], then both coordinate properties are set to [ ] and all dynamic properties are removed.

- If a dynamic property is set to [ ], then it is removed from the object.

- The mapshape vector can be indexed like any MATLAB vector. You can access any element of the vector to obtain a specific feature. The following example demonstrates this behavior:

"Construct a Mapshape Vector Containing Multiple Features and Indexing Behaviors" on page 1-954

This example builds a mapshape vector from a structure array; adds a Metadata property and demonstrates selective indexing behavior. "Construct a Mapshape Vector and Add Metadata and Indexing" on page 1-956

"Construct a Mapshape Vector and Sort the Dynamic Properties" on page 1-957

See Also

Functions
gpxread | shaperead

Objects
geopoint | geoshape | mappoint

Topics
"Create and Display Polygons"

Introduced in R2012a

mapshow

Display map data without projection

Syntax

```
mapshow(x,y)
mapshow(S)

mapshow(x,y,Z)
mapshow(Z,R)

mapshow(x,y,I)
mapshow(x,y,X,cmap)
mapshow(I,R)
mapshow(X,cmap,R)

mapshow(filename)

mapshow( ___ ,Name,Value)
mapshow(ax, ___ )
h = mapshow( ___ )
```

Description

mapshow(x,y) displays the coordinate vectors x and y as lines. You can optionally display the coordinate vectors as points or polygons by using the DisplayType name-value pair argument.

mapshow(S) displays the vector geographic features stored in S as points, multipoints, lines, multilines, polygons, or multipolygons according to the geometry of S.

You can optionally specify symbolization rules using the SymbolSpec name-value pair argument.

mapshow(x,y,Z) displays a geolocated data grid, Z. You can optionally display the data as a surface, mesh, texture map, or contour by using the DisplayType name-value pair argument.

mapshow(Z,R) displays a regular data grid, Z, with referencing object R. You can optionally display the data as a surface, mesh, texture map, or contour by using the DisplayType name-value pair argument. If DisplayType is 'texturemap', then mapshow displays the image as a texture map on a zero-elevation surface (by setting ZData values to 0).

mapshow(x,y,I) and

mapshow(x,y,X,cmap) display a geolocated image as a texture map on a zero-elevation surface. The geolocated image can be a truecolor, grayscale, or binary image, I, or an indexed image X with colormap cmap. x and y are geolocation arrays in map coordinates. Examples of geolocated images include a color composite from a satellite swath or an image originally referenced to a different coordinate system.

mapshow(I,R) and

mapshow(X,cmap,R) display an image georeferenced to map coordinates through the referencing object R. The mapshow function constructs an image object if the display geometry permits.

Otherwise, `mapshow` displays the image as a texture map on a zero-elevation surface (by setting `ZData` values to 0).

`mapshow(filename)` displays data from the file specified according to the type of file format.

`mapshow( ___ ,Name,Value)` modifies the displayed map by using name-value pair arguments to set the `DisplayType` and `SymbolSpec` parameters. You can also use name-value pairs to set any MATLAB graphics properties. Parameter names can be abbreviated, and case does not matter.

`mapshow(ax, ___ )` sets the parent axes to `ax`.

`h = mapshow( ___ )` returns a handle to a MATLAB graphics object.

Examples

Overlay Vector on Orthophoto

Overlay Boston roads on an orthophoto. Note that `mapshow` draws a new layer in the axes rather than replacing its contents.

Display image.

```
figure
mapshow boston.tif
axis image off manual
```

Convert Boston roads to units of survey feet and overlay on orthophoto.

```
S = shaperead('boston_roads.shp');
surveyFeetPerMeter = unitsratio('sf','meter');
x = surveyFeetPerMeter * [S.X];
y = surveyFeetPerMeter * [S.Y];
mapshow(x,y)
```


Display Vector Data Customizing Line Style

Import a shapefile containing a network of road segments in Concord, MA as a geospatial table. Display the road segments using dotted lines.

```
roads = readgeotable("boston_roads.shp");
figure
mapshow(roads,"LineStyle",":")
```
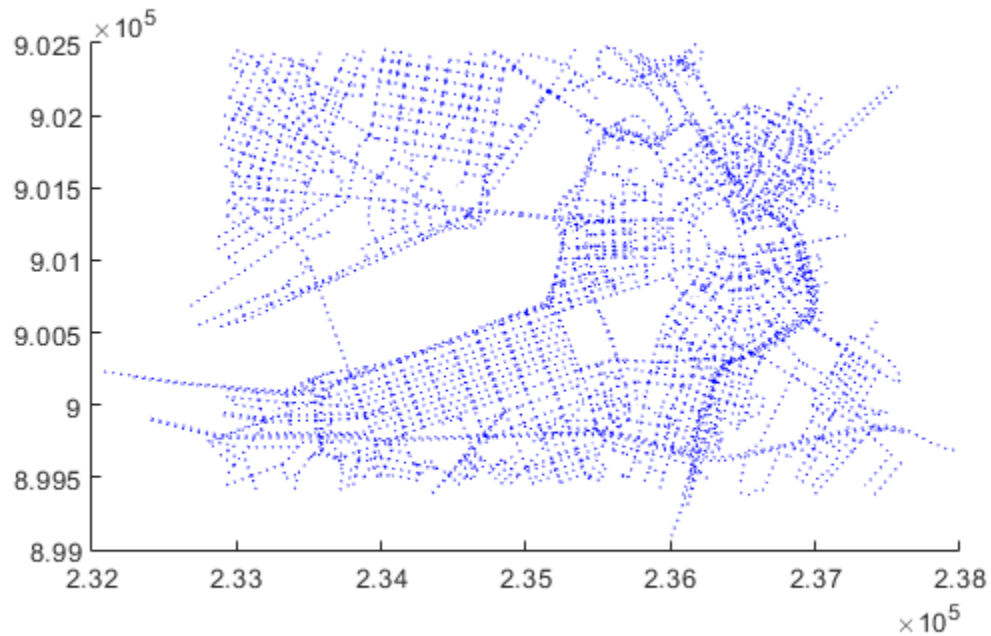
Display Vector Data Using Symbol Specification

Create a symbol specification to distinguish between different types of roads. For example, you can hide very minor roads (CLASS=6) by turning off their visibility and make major roads (CLASS=1-4) more visible by increasing their line widths. This symbol specification also uses color to distinguish between types of roads.

```
roadspec = makesymbolspec('Line',...
                          {'ADMIN_TYPE',0,'Color','cyan'}, ...
                          {'ADMIN_TYPE',3,'Color','red'},...
                          {'CLASS',6,'Visible','off'},...
                          {'CLASS',[1 4],'LineWidth',2});
```

Display the vector data using the symbol specification.

```
figure
mapshow('boston_roads.shp','SymbolSpec',roadspec);
```
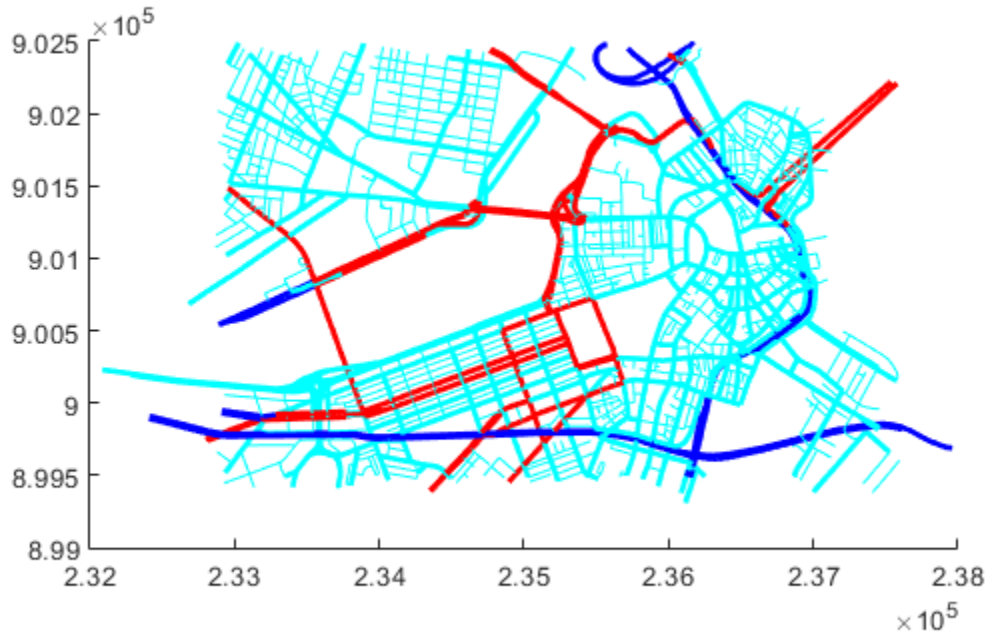
Override Default Properties of the Line

Create a symbol specification and specify the default color used for lines. As seen in the previous example, the default is blue. This example sets the default to black.

```
roadspec = makesymbolspec('Line',...
                          {'Default', 'Color', 'black'}, ...
                          {'ADMIN_TYPE',0,'Color','c'}, ...
                          {'ADMIN_TYPE',3,'Color','r'},...
                          {'CLASS',6,'Visible','off'},...
                          {'CLASS',[1 4],'LineWidth',2});
```

Display the vector data, using the symbol specification. Note how the major roads displayed in blue in the previous example are now black.

```
figure
mapshow('boston_roads.shp','SymbolSpec',roadspec);
```
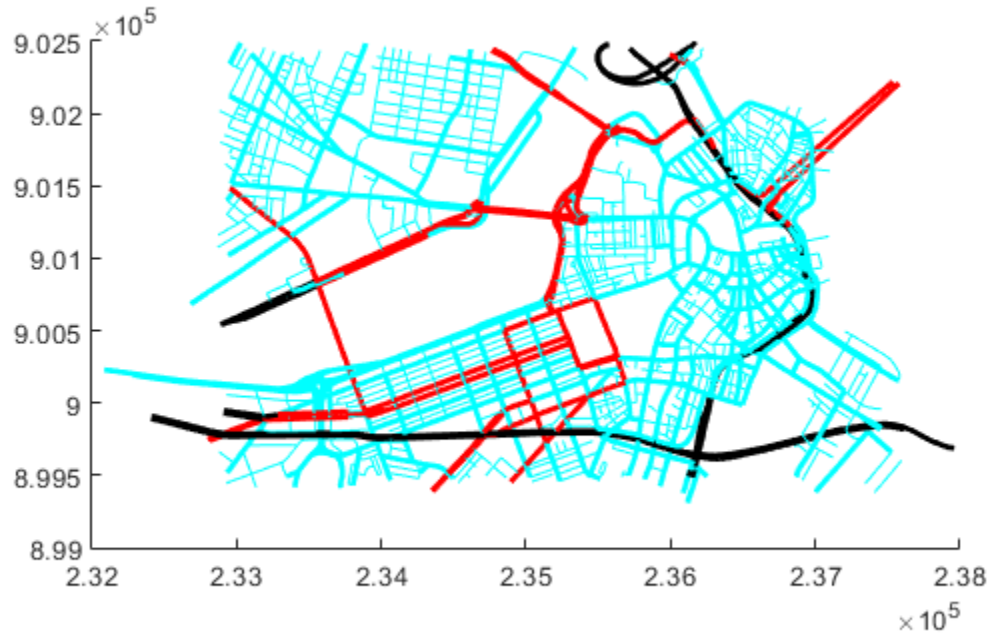
Override Symbol Specification on Command Line

Create a symbol specification, setting various properties.

```
roadspec = makesymbolspec('Line',...
                          {'Default', 'Color', 'yellow'}, ...
                          {'ADMIN_TYPE',0,'Color','c'}, ...
                          {'ADMIN_TYPE',3,'Color','r'},...
                          {'CLASS',6,'Visible','off'},...
                          {'CLASS',[1 4],'LineWidth',2});
```

Display the vector data, specifying the color on the command line.

```
figure
mapshow('boston_roads.shp', 'Color', 'black', 'SymbolSpec', roadspec);
```

Display Polygon Over Orthophoto

Import an orthophoto of Concord, MA, along with a map cells reference object and a colormap. Display the orthophoto using the `mapshow` function.

```
[ortho,R,cmap] = readgeoraster("concord_ortho_w.tif");
mapshow(ortho,cmap,R)
```
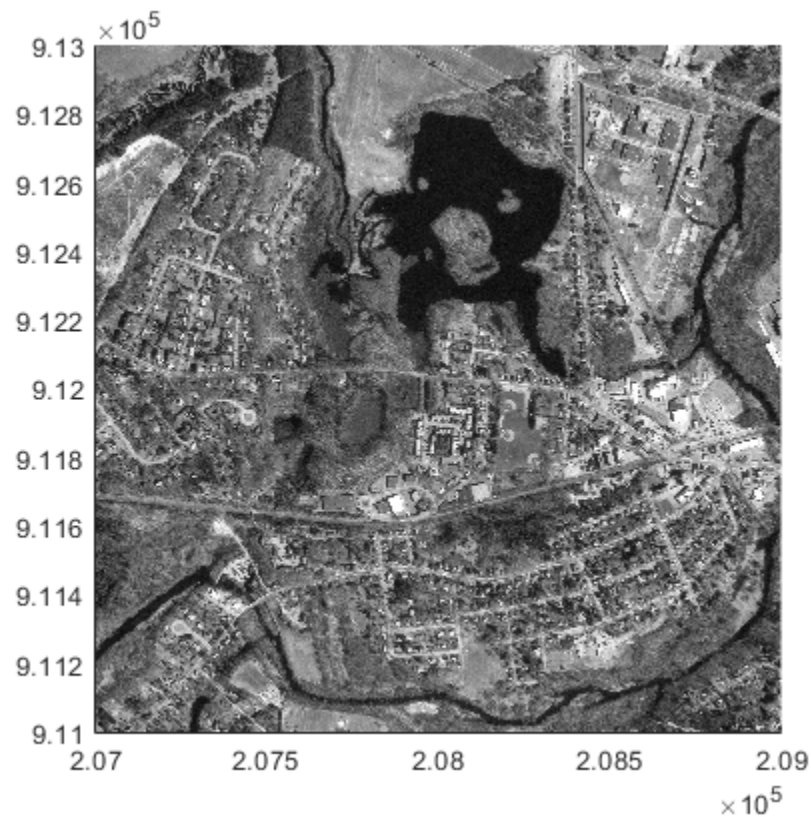
Import and display a polygon representing a pond in the middle of the orthophoto.

Import a shapefile containing hydrographic data for Concord as a geospatial table. Create a subtable that contains the 14th row of the table, which is the row containing the pond. Verify that the subtable contains polygon data by querying the Shape variable of the table.

```
hydro = readgeotable("concord_hydro_area.shp");
pond = hydro(14,:);
pond.Shape
```

```
ans =
  mappolyshape with properties:

              NumRegions: 1
                NumHoles: 3
                Geometry: "polygon"
    CoordinateSystemType: "planar"
            ProjectedCRS: [1x1 projcrs]
```

Display the polygon over the orthophoto.

```
mapshow(pond,"FaceColor","c")
```
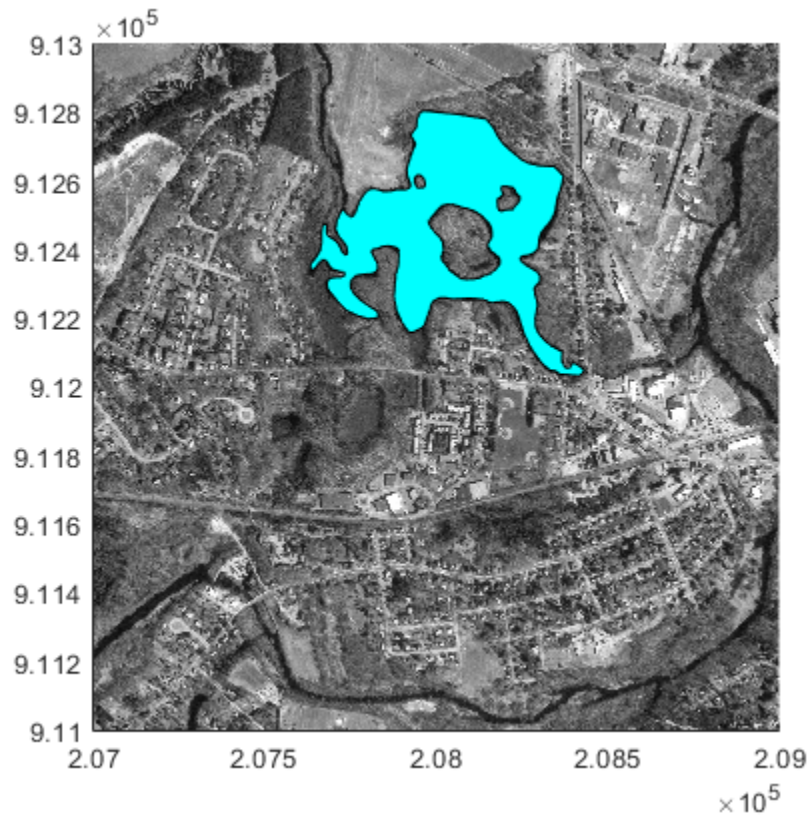
Display Terrain Data as Mesh, Surface, and 3-D Surface

Read SDTS terrain data for Mount Washington. Get information such as missing data indicators using `georasterinfo`. Replace the missing data with `NaN` values using `standardizeMissing`.

```
[Z,R] = readgeoraster('sdts/9129CATD.DDF','OutputType','double');
info = georasterinfo('sdts/9129CATD.DDF');
Z = standardizeMissing(Z,info.MissingDataIndicator);
```

View the terrain data as a mesh. Apply a colormap appropriate for terrain data using `demcmap`.

```
figure
mapshow(Z,R,'DisplayType','mesh');
demcmap(Z)
```
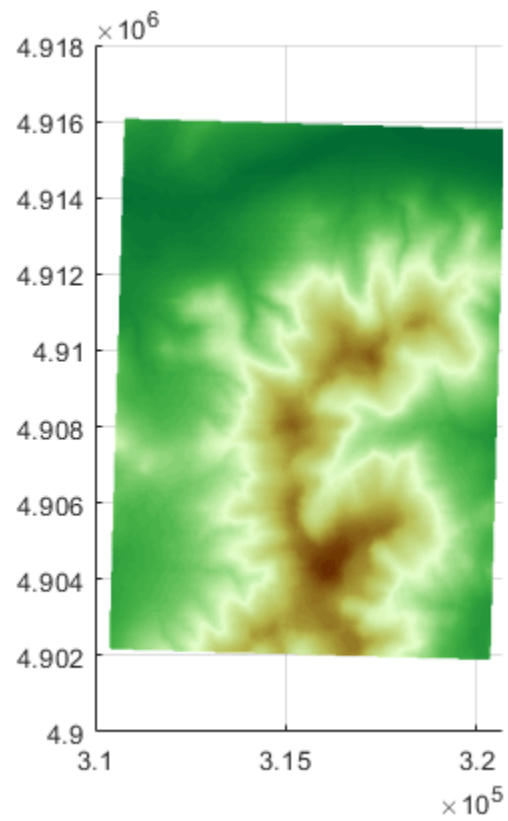
View the terrain data as a surface.

```
figure
mapshow(Z,R,'DisplayType','surface');
demcmap(Z)
```
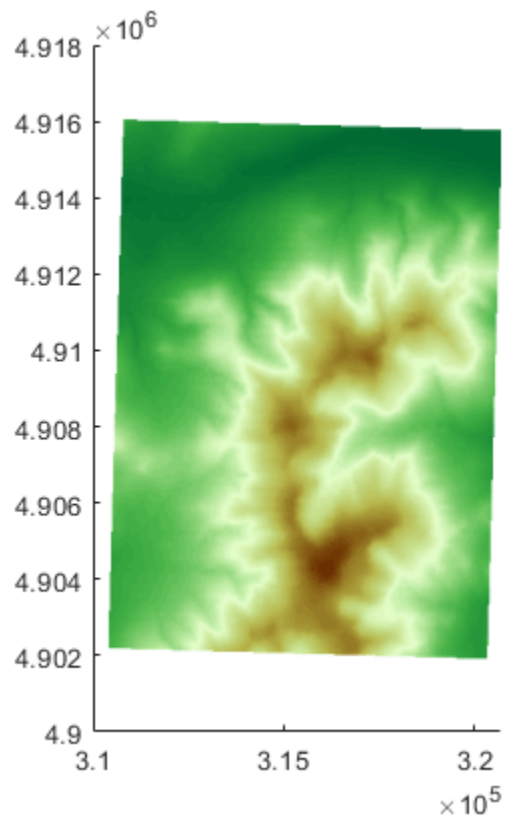
View the terrain data as a 3-D surface.

```
view(3);
axis normal
```
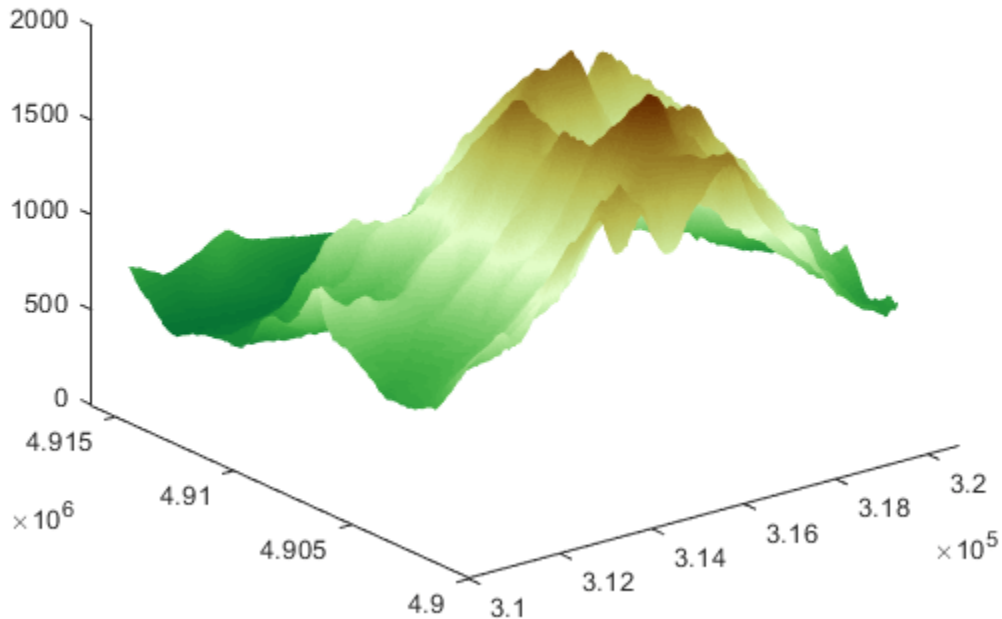
Display Grid and Contour Lines

Read the terrain data files for Mount Washington and Mount Dartmouth. To plot the data as a surface using mapshow, the raster must be of type single or double. Specify the data type for the raster using the 'OutputType' name-value pair.

```
[ZWash,RWash] = readgeoraster('MtWashington-ft.grd','OutputType','double');
[ZDart,RDart] = readgeoraster('MountDartmouth-ft.grd','OutputType','double');
```

Find missing data using the georasterinfo function. The function returns an object with a MissingDataIndicator property that indicates which value represents missing data. Replace the missing data with NaN values using the standardizeMissing function.

```
infoWash = georasterinfo('MtWashington-ft.grd');
ZWash = standardizeMissing(ZWash,infoWash.MissingDataIndicator);

infoDart = georasterinfo('MountDartmouth-ft.grd');
ZDart = standardizeMissing(ZDart,infoDart.MissingDataIndicator);
```

Display the terrain data under the contour lines and labels by specifying the 'ZData' name-value pair as a matrix of zeros. Apply a colormap appropriate for terrain data using demcmap.

```
hold on
mapshow(ZWash,RWash,'DisplayType','surface','ZData',zeros(RWash.RasterSize))
```

```
mapshow(ZDart,RDart,'DisplayType','surface','ZData',zeros(RDart.RasterSize))
axis equal
demcmap(ZWash)
```
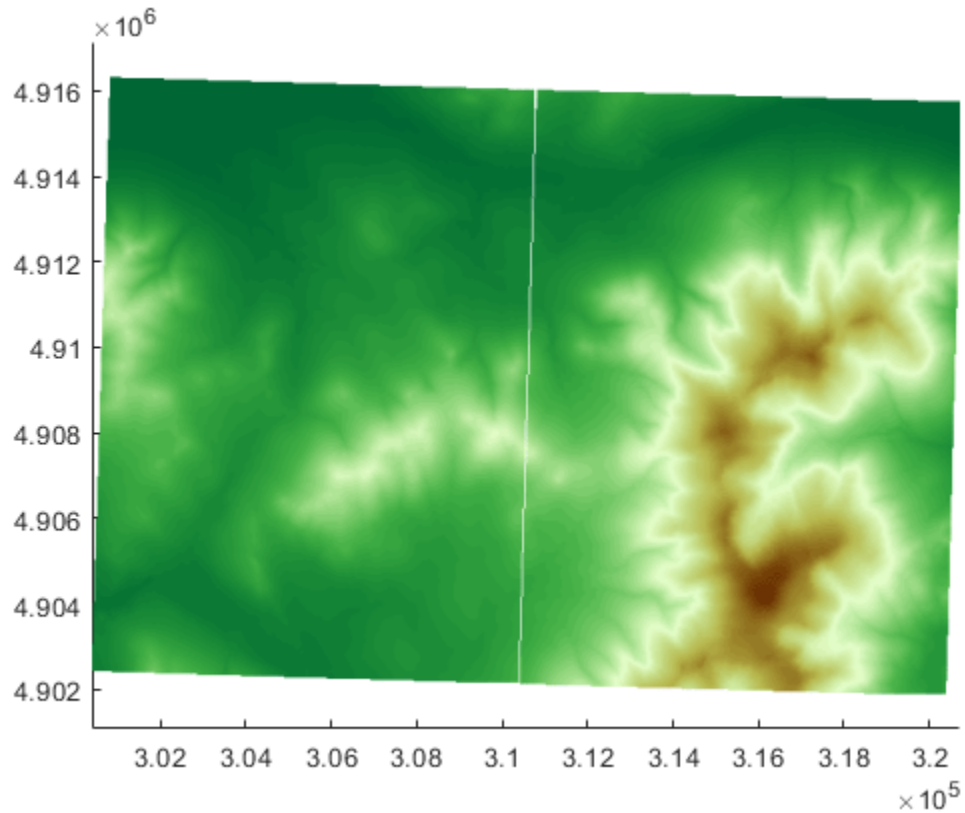

Overlay black contour lines and labels.

```
mapshow(ZWash,RWash,'DisplayType','contour',...
    'LineColor','k','ShowText','on');
mapshow(ZDart,RDart,'DisplayType','contour',...
    'LineColor','k','ShowText','on');
```
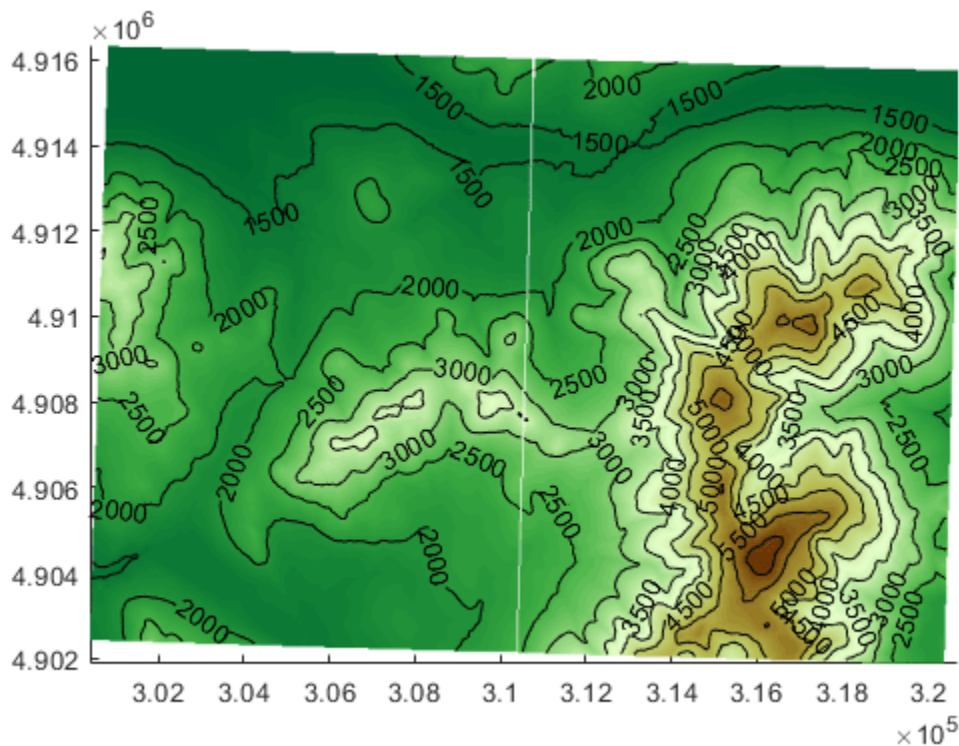
Input Arguments

x, y — *x*- or *y*-coordinates
numeric vector | *M*-by-*N* numeric matrix

x- or *y* coordinates, specified as a numeric vector or an *M*-by-*N* numeric matrix. x and y must be the same size. If x and y are matrices, they represent coordinate arrays or a geolocation array in map coordinates. x and y may contain embedded NaNs to delimit individual lines or polygon parts.

S — Geographic features
geospatial table | mappoint vector | mapshape vector | map structure array

Geographic features, specified as one of the following:

- A geospatial table. The Shape variable of the table must contain mappointshape, maplineshape, or mappolyshape objects. The mapshow function does not support geospatial tables containing more than one type of shape object. For more information about geospatial tables, see "Create Geospatial Tables".

- A mappoint vector.

- A mapshape vector.

- A map structure array (mapstruct) with X and Y coordinate fields.

The mapshow function assumes the coordinates are map coordinates.

If you specify a `geopoint` vector, a `geoshape` vector, or a geographic structure array (`geostruct`) with `Lat` and `Lon` coordinate fields, then the `mapshow` function projects the coordinates using a Plate Carrée projection and issues a warning. If you specify a geospatial table containing `geopointshape`, `geolineshape`, or `geopolyshape` objects, then the `mapshow` function throws an error. In these cases, use the `geoshow` function instead.

Z — Data grid
M-by-*N* array

Data grid, specified as an *M*-by-*N* array that may contain `NaN` values. Z is either a georeferenced data grid, or a regular data grid associated with a geographic reference R. The size of x and y must match the size of Z.

Data Types: `double`

R — Map reference
map raster reference object | matrix

Map reference, specified as one of the following. For more information about referencing matrices, see "Georeferenced Raster Data".

Type	Description
Map raster reference object	`MapCellsReference` or `MapPostingsReference` or `GeographicPostingsReference` map raster reference object that relates the subscripts of Z to map coordinates. The `RasterSize` property must be consistent with the size of the data grid, `size(Z)`. If R is a `MapPostingsReference` object, then the `'image'` and `'texturemap'` values of `DisplayType` are not accepted.
Matrix	3-by-2 numeric matrix that transforms raster row and column indices to or from map coordinates according to: `[x y] = [row col 1] * R`

I — Truecolor, grayscale, or binary image
M-by-*N*-by-3 array | *M*-by-*N* array

Truecolor, grayscale, or binary image, specified as an *M*-by-*N*-by-3 array for truecolor images, or an *M*-by-*N* array for grayscale or binary images. x and y must be *M*-by-*N* arrays.

X — Indexed image
M-by-*N* array

Indexed image with color map defined by `cmap`, specified as an *M*-by-*N* array. x and y must be *M*-by-*N* arrays.

cmap — Color map
c-by-3 matrix

Color map of indexed image X, specified as an *c*-by-3 numeric matrix. There are *c* colors in the color map, each represented by a red, green, and blue pixel value.

filename — File name
character vector | string scalar

File name, specified as a string scalar or character vector. `mapshow` automatically sets the `DisplayType` parameter according to the format of the data.

Format	DisplayType
Shape file	`'point'`, `'multipoint'`, `'line'`, or `'polygon'`
GeoTIFF	`'image'`
TIFF/JPEG/PNG with a world file	`'image'`
ARC ASCII GRID	`'surface'` (can be overridden)
SDTS raster	`'surface'` (can be overridden)

Data Types: `char` | `scalar`

ax — Parent axes
axes object

Parent axes, specified as an axes object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'DisplayType','point'`

DisplayType — Display type
`'point'` | `'multipoint'` | `'line'` | `'polygon'` | `'image'` | `'surface'` | `'mesh'` | `'texturemap'` | `'contour'`

Type of graphic display for the data, specified as the comma-separated pair consisting of `'DisplayType'` and one of the following values.

Data Format	Display Type	Type of Property
Vector	`'point'`	*line marker*
	`'multipoint'`	*line marker*
	`'line'`	*line*
	`'polygon'`	*patch*
Image	`'image'`	*surface*
Grid	`'surface'`	*surface*
	`'mesh'`	*surface*
	`'texturemap'`	*surface*
	`'contour'`	*contour*

Valid values of `DisplayType` depend on the format of the map data. For example, if the map data is a geolocated image or georeferenced image, then the only valid value of `DisplayType` is `'image'`.

Different display types support different map data class types:

Display Type	Supported Class Types
Image	
Surface	`single` and `double`
Texture map	All numeric types and `logical`

SymbolSpec — Symbolization rules
structure

Symbolization rules to be used for displaying vector data, specified as a `symbolSpec` structure returned by `makesymbolspec`. When both `SymbolSpec` and one or more graphics properties are specified, the graphics properties will override any settings in the `symbolSpec` structure.

To change the default symbolization rule for a `Name,Value` pair in the `symbolSpec` structure, prefix the word `'Default'` to the graphics property name.

Output Arguments

h — Parent axes
handle object | modified patch object

Parent axes, returned as a handle to a MATLAB graphics object or, in the case of polygons, a modified patch object. If a mapstruct or shapefile name is input, `mapshow` returns the handle to an `hggroup` object with one child per feature in the mapstruct or shapefile. In the case of a polygon mapstruct or shapefile, each child is a modified patch object; otherwise it is a line object.

Tips

*   If you do not want `mapshow` to draw on top of an existing map, create a new figure or subplot before calling it.

*   You can use `mapshow` to display vector data in an `axesm` figure. However, you should not subsequently change the map projection using `setm`.

*   If you display a polygon, do not set `'EdgeColor'` to either `'flat'` or `'interp'`. This combination may result in a warning.

See Also

Functions
geoshow | makesymbolspec | mapview | shaperead

Objects
MapCellsReference | MapPostingsReference

Topics
"Create and Display Polygons"

Introduced before R2006a

maptriml

Trim lines to latitude-longitude quadrangle

Syntax

```
[lat,lon] = maptriml(lat0,lon0,latlim,lonlim)
```

Description

`[lat,lon] = maptriml(lat0,lon0,latlim,lonlim)` returns *filtered* NaN-delimited vector map data sets from which all points lying outside the desired latitude and longitude limits have been discarded. These limits are specified by the two-element vectors `latlim` and `lonlim`, which have the form `[south-limit north-limit]` and `[west-limit east-limit]`, respectively.

Examples

Trim Lines to Region of Interest

It is not unusual for vector data to extend beyond the geographic region currently of interest. For example, you might have coastline data for the entire world, but are interested in mapping Australia only. In this and other situations, you might want to eliminate unnecessary data from the workspace and from calculations in order to save memory or to speed up processing and display. You can trim line data by simply removing points outside the region of interest by clipping lines at the map frame or to some other defined region. The `maptriml` function returns variables containing only those points that lie within the defined region.

Load the `coastlines` MAT-file. This file contains data for the entire world.

```
load coastlines
```

Define a region of interest centered on Australia.

```
latlim = [-50 0];
lonlim = [105 160];
```

Delete all line data outside these limits, producing line vectors, by using the `maptriml` function.

```
[linelat,linelon] = maptriml(coastlat,coastlon,latlim,lonlim);
```

Examine the variables to see how much data has been reduced. The clipped data is only 10% as large as the original data set.

```
whos
```

Name	Size	Bytes	Class	Attributes
coastlat	9865x1	78920	double	
coastlon	9865x1	78920	double	
latlim	1x2	16	double	
linelat	977x1	7816	double	

```
    linelon        977x1                7816  double
    lonlim           1x2                  16  double
```

Plot the trimmed line data using a Miller projection.

```
axesm('MapProjection', 'miller', 'Frame', 'on',...
'FlatLimit', latlim, 'FlonLimit', lonlim)
geoshow(linelat,linelon)
```


See Also

`maptrimp` | `geocrop`

Introduced before R2006a

maptrimp

Trim polygons to latitude-longitude quadrangle

Syntax

```
[latTrimmed,lonTrimmed] = maptrimp(lat,lon,latlim,lonlim)
```

Description

[latTrimmed,lonTrimmed] = maptrimp(lat,lon,latlim,lonlim) trims the polygons in lat and lon to the quadrangle specified by latlim and lonlim. latlim and lonlim are two-element vectors, defining the latitude and longitude limits respectively. lat and lon must be vectors that represent valid polygons.

Examples

Trim Dataset to Specific Geographic Area

Read coastline data and display it on a map.

```
load coastlines
figure
mapshow(coastlon,coastlat,'DisplayType','polygon');
```
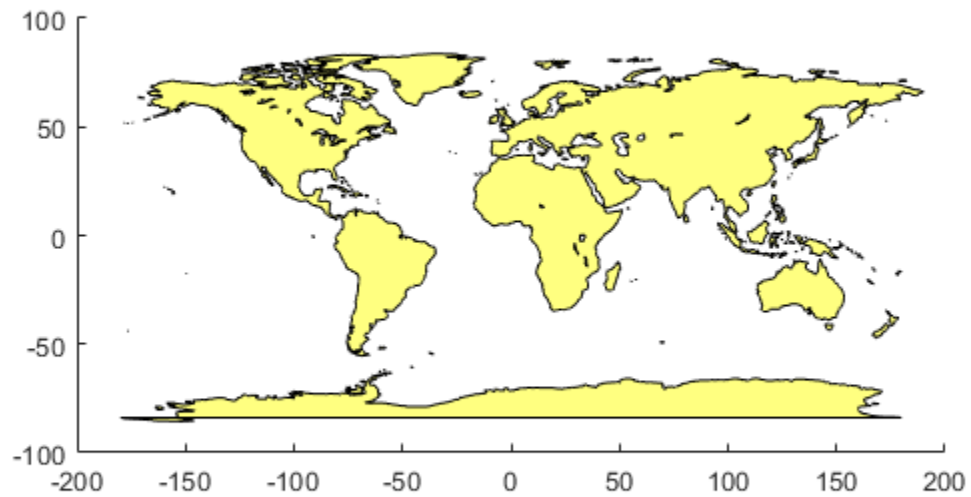
Trim the dataset.

```
latlim = [-50 50];
lonlim = [-100 50];
[latTrimmed,lonTrimmed] = maptrimp(coastlat,coastlon, ...
    latlim, lonlim);
```

Display the trimmed dataset.

```
figure
mapshow(lonTrimmed,latTrimmed, 'DisplayType', 'polygon');
```
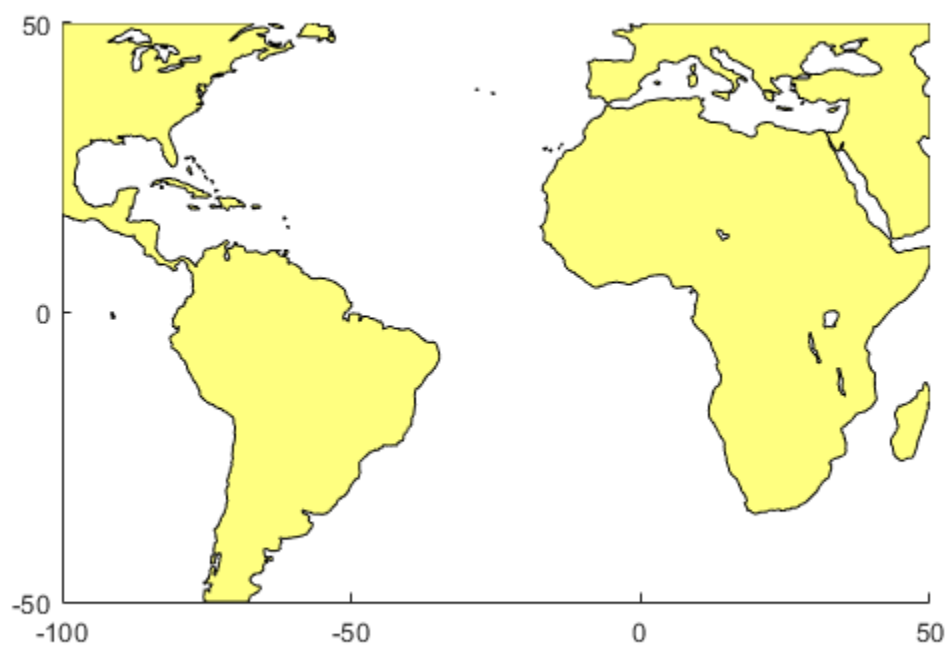
Tips

`maptrimp` conditions the longitude limits such that:

- `lonlim(2)` always exceeds `lonlim(1)`
- `lonlim(2)` never exceeds `lonlim(1)` by more than `360`
- `lonlim(1)` `<` `180` or `lonlim(2)` `>` `-180`
- Ensure that if the quadrangle span the Greenwich meridian, then that meridian appears at longitude `0`.

See Also

`maptriml` | `geocrop`

Introduced before R2006a

maptrims

(To be removed) Trim regular data grid to latitude-longitude quadrangle

---

Note `maptrims` will be removed in a future release. Use the `geocrop` function instead. For more information, see "Compatibility Considerations".

Syntax

```
[Z_trimmed] = maptrims(Z,R,latlim,lonlim)
[Z_trimmed] = maptrims(Z,R,latlim,lonlim,cellDensity)
[Z_trimmed, R_trimmed] = maptrims(...)
```

Description

`[Z_trimmed] = maptrims(Z,R,latlim,lonlim)` trims a regular data grid Z to the region specified by `latlim` and `lonlim`. By default, the output grid `Z_trimmed` has the same sample size as the input. R can be a geographic raster reference object, a referencing vector, or a referencing matrix. If R is a geographic raster reference object, its `RasterSize` property must be consistent with `size(Z)` and its `RasterInterpretation` must be `'cells'`.

If R is a referencing vector, it must be a 1-by-3 vector with elements:

```
[cells/degree northern_latitude_limit western_longitude_limit]
```

If R is a referencing matrix, it must be 3-by-2 and transform raster row and column indices to/from geographic coordinates according to:

```
[lon lat] = [row col 1] * R
```

If R is a referencing matrix, it must define a (non-rotational, non-skewed) relationship in which each column of the data grid falls along a meridian and each row falls along a parallel. `latlim` and `lonlim` are two-element vectors, defining the latitude and longitude limits, respectively. The `latlim` vector has the form:

```
[southern_limit northern_limit]
```

Likewise, the `lonlim` vector has the form:

```
[western_limit eastern_limit]
```

When an individual value in `latlim` or `lonlim` corresponds to a parallel or meridian that runs precisely along cell boundaries, the output grid will extend all the way to that limit. But if a limiting parallel or meridian cuts through a column or row of input cells, then the limit will be adjusted inward. In other words, the requested limits will be truncated as necessary to avoid partial cells.

`[Z_trimmed] = maptrims(Z,R,latlim,lonlim,cellDensity)` uses the scalar `cellDensity` to reduce the size of the output. If R is a referencing vector, then `R(1)` must be evenly divisible by `cellDensity`. If R is a referencing matrix, then the inverse of each element in the first two rows (containing "`deltaLat`" and "`deltaLon`") must be evenly divisible by `cellDensity`.

[Z_trimmed, R_trimmed] = maptrims(...) returns a referencing vector, matrix, or object for the trimmed data grid. If R is a referencing vector, then R_trimmed is a referencing vector. If R is a referencing matrix, then R_trimmed is a referencing matrix. If R is a geographic raster reference object, then R_trimmed is either a geographic raster reference object (when Z_trimmed is non-empty) or [] (when Z_trimmed is empty).

Examples

Load elevation raster data and a geographic cells reference object. Then, trim the data to the specified latitude and longitude limits.

```
load topo60c
[subgrid,subR] = maptrims(topo60c,topo60cR,...
                          [80.25 85.3],[165.2 170.7])

subgrid =
        -2826         -2810         -2802         -2793
        -2915         -2913         -2905         -2884
        -3192         -3186         -3165         -3122
        -3399         -3324         -3273         -3214

subR =

  GeographicCellsReference with properties:

             LatitudeLimits: [81 85]
            LongitudeLimits: [166 170]
                 RasterSize: [4 4]
       RasterInterpretation: 'cells'
           ColumnsStartFrom: 'south'
              RowsStartFrom: 'west'
        CellExtentInLatitude: 1
       CellExtentInLongitude: 1
      RasterExtentInLatitude: 4
     RasterExtentInLongitude: 4
            XIntrinsicLimits: [0.5 4.5]
            YIntrinsicLimits: [0.5 4.5]
        CoordinateSystemType: 'geographic'
                   AngleUnit: 'degree'
```

The upper left corner of the grid might differ slightly from that of the requested region. The maptrims function uses the corner coordinates of the first cell inside the limits.

Compatibility Considerations

maptrims will be removed
Not recommended starting in R2020b

Some functions that accept referencing vectors or referencing matrices as input will be removed, including the maptrims function. Use a geographic reference object and the geocrop function instead. Reference objects have several advantages over referencing vectors and matrices.

- Unlike referencing vectors and matrices, reference objects have properties that document the size of the associated raster, its geographic limits, and the direction of its rows and columns. For more information about reference object properties, see the GeographicCellsReference and GeographicPostingsReference objects.

- Manipulate the limits of geographic rasters associated with reference objects using the `geocrop` function.
- Manipulate the size and resolution of geographic rasters associated with reference objects using the `georesize` function.

To update your code, first create a reference object for either a raster of cells using the `georefcells` function or a raster of regularly posted samples using the `georefpostings` function. Alternatively, convert from a referencing vector or a referencing matrix to a reference object using the `refvecToGeoRasterReference` or `refmatToGeoRasterReference` function, respectively.

Then, replace instances of the `maptrims` function with the `geocrop` function using these patterns.

Will Be Removed	Recommended
`[B,RB] = maptrims(A,R,latlim,lonlim);`	`[B,RB] = geocrop(A,R,latlim,lonlim)`

`[B,RB] = maptrims(A,R,latlim,lonlim,cellDensity);`	`[B,RB] = geocrop(A,R,latlim,lonlim)`
	`latscale = cellDensity * RB.CellExtentInLatitude;`
	`lonscale = cellDensity * RB.CellExtentInLongitude;`
	`[B,RB] = georesize(B,RB,latscale,lonscale,'nearest');`

The limits of the reference object returned by the `geocrop` function may be larger than the limits returned by the `maptrims` function.

See Also
`maptriml` | `maptrimp` | `geocrop`

Introduced before R2006a

mapview

Interactive map viewer

Syntax

```
mapview
```

Description

`mapview` opens the Map Viewer app in an empty state. Use the Map Viewer to view geospatial data in map (*x*-*y*) coordinates. The Map Viewer works with vector, image, and raster data grids in a map coordinate system. You can load data, pan and zoom on the map, control the map scale of your screen display, control the order, visibility, and symbolization of map layers, annotate your map, and click to learn more about individual vector features. `mapview` complements `mapshow` and `geoshow`, which are for constructing maps in ordinary figure windows in a less interactive, script-oriented way.

For usage information, see the following sections. You can also work through the Map Viewer tutorial, "Tour Boston with the Map Viewer App".

Importing Data

The Map Viewer opens with no data loaded and an empty map display window. The first step is to import a data set. Use the options in the **File** menu to select data from a file or from the MATLAB workspace:

Import From File

Use the file browsing dialog to open a file in one of the following formats: Shapefile, GeoTIFF, SDTS DEM, Arc ASCII Grid, TIFF, JPEG, or PNG with world file. This option imports the data into the viewer but does not add it to your workspace.

To view standard-format geodata files provided with the toolbox, set your working folder or navigate the Map Viewer Open dialog to *matlabroot*/examples/map/data or *matlabroot*/toolbox/map/mapdata.

Import From Workspace

Images.   Use the **Raster Data > Image** import dialog to select a **Referencing matrix or object name** and **Raster data name** for the image from the list of workspace variables. If the image type is truecolor (RGB), specify which band represents the red, green, and blue intensities. (The `RasterInterpretation` of the referencing object must be `'cells'`.)

Data grids.   Use the **Raster Data > Grid** import dialog to select X and Y geolocation and data grid array names from the list of workspace variables.

Vector data.   Use the **Vector Data > Map coordinates** import dialog to select X and Y variables for map coordinates from the list of workspace variables and identify the type of geometry to be displayed (**Point**, **Line**, or **Polygon**). The X and Y variables can specify multiple line segments or multiple polygons if they contain `NaN`s at matching locations in the coordinate vectors.

Vector geographic data structure. Use the **Vector Data > Geographic data structure** import dialog to select the structure that contains vector map data from the list of workspace variables.

Once you import your first data set, the Map Viewer automatically sets the limits of its map display window to the spatial extent of the imported data.

Working in Map Coordinates

As you move any of the Map Viewer cursors across the map display area, the coordinate readout in the lower left corners shows you the cursor position in map X and Y coordinates.

The Map Viewer requires that all currently viewed data sets possess the same coordinate system and length units. This is likely to be the case for data sets that originated from a common source. If it is not the case, you will need to adjust coordinates before importing data into the Map Viewer.

If some or all of your data is in geographic coordinates, use `projfwd` to project latitudes and longitudes to your desired map coordinate system before you import it. When starting from a different projection, you must first unproject to latitude and longitude using `projinv`, then reproject with `projfwd`. You might also need to adjust the horizontal datum of your data using, for example, the free GEOTRANS (Geographic Translator) application from the Geospatial Sciences Division of the U.S. National Geospatial-Intelligence Agency (NGA). If you simply need a change of units, multiply by the appropriate conversion factor obtained from `unitsratio`.

`mapview` can also display data in unprojected geographic coordinates, if you consistently substitute longitude for map X and latitude for map Y. Geographic coordinates must be consistently expressed in either degrees or radians (not both at once). When using geographic coordinates, do not specify the viewer's map units (see below); you can only use the Map Viewer's map scale display when working in linear units of length.

Setting Map Units and Scale

If you tell the Map Viewer which length unit you are using, it can calculate an approximate map scale for your onscreen display. Set the map units with either the drop-down menu at the bottom of the display or the **Set Map Units** item in the **Tools** menu.

The scale computed by the Map Viewer is displayed in the window just above the map units drop-down. To change your display scale while keeping the center of the map display fixed, simply edit this text box.

Make sure to format your text in the standard way (`1:`*N*, where *N* is a positive number such that a distance on the ground is *N* times the same distance on your screen, e.g., `1:24000`).

The scale is approximate because it depends on the MATLAB estimate of the size of your screen pixels. It is also approximate if your projection introduces significant distortion. If your data falls in a fairly small area and you use a conformal projection (e.g., UTM with all data in a single zone), the scale will be very consistent across your entire map.

Navigating Your Map

By default, the Map Viewer sets the limits of your map window to match the extent of the first data set that you load. You will probably want to adjust this to see some areas in greater detail.

The Map Viewer provides several tools to control the limits of your map window and the map scale of the data display. Some are familiar from standard MATLAB figure windows.

- **Zoom in**: Drag a box to zoom in on a specific area or click a point to zoom in with that point centered in the map display.

- **Zoom out**: Click a point to zoom out with that point centered in the map display.

- **Pan tool**: Click, hold, and drag to reposition the selected point in the display window, while holding the map scale fixed. Release when you are satisfied with new display limits.

- **Fit to window**: Set the map display to enclose all currently loaded data layers. This is equivalent to selecting **Fit to Window** in the **View** menu.

- **Back to previous view**: Click this button once to return the map scale and display center to their values prior to the most recent zoom, pan, or scale change. Click repeatedly to undo earlier changes. This is equivalent to selecting **Previous View** in the **View** menu.

Another way to zoom in or out while keeping the center of the view fixed at the same map coordinates is to directly edit the map scale box at the bottom of the screen.

Managing Map Layers

Each time you import a set of vectors, an image, or a data grid into the Map Viewer, the new data is stored in a new map layer. The layers form an ordered stack. Each layer is listed as an item in the **Layers** menu, with its position in the menu indicating its position in the stack.

When you import a new layer, the Map Viewer automatically places it at the top of the layer stack. To reposition a layer in the stack, select it in the **Layers** menu, slide right, and select **To Top**, **To Bottom**, **Move Up**, or **Move Down** from the pop-up submenu.

The vector features or raster in a given layer obscure coincident elements of any underlying layers. To control layers that are obscuring one another, you can also toggle layer visibility on and off. Use the item **Visible** in the slide-right menu. Or, simply remove a layer from the Map Viewer via the **Remove** item in the slide-right menu. Remember that even if a layer's visibility is *on*, the layer does not appear if its contents are located completely outside the current display limits or are obscured by another layer.

Symbolizing Vector Features

When point, line, and polygon layers are loaded, the Map Viewer initializes their graphics properties as follows:

Geometry	Properties

Point (line objects)	`LineStyle = 'none`
	`Marker = 'x'`
	`MarkerEdgeColor = <randomly generated value>`
	`MarkerFaceColor = 'none'`

Line (line objects)	`Color = <randomly generated value>`
	`LineStyle = '-'`
	`Marker = 'none'`

Polygon (patch objects)	`EdgeColor = [0 0 0] FaceColor = <randomly generated value>`

To override symbolism defaults for a vector layer, use `makesymbolspec` to create a symbol specification in the workspace. A symbolspec contains a set of rules for setting vector graphics properties based on the values of feature attributes. For instance, if you have a line layer representing roads of various classes (e.g., major highway, secondary road, etc.), you can create a symbolspec to use a different color, line width, or line style for each road class. See the `makesymbolspec` help for examples and to learn how to construct a symbolspec. If you regularly work with data sets sharing a common set of feature attributes, you might want to save one or more symbolspecs in a MAT-file (or save calls to `makesymbolspec` in a MATLAB program file).

Once you have a symbolspec in your workspace, select your vector layer in the **Layers** menu, then slide right and click **Set Symbol Spec**, which opens a dialog box. Use the dialog box to select the symbolspec from your workspace.

Getting Information About Vector Features

The **Datatip** tool and the **Info** tool provide different ways to check the attributes of vector features that you select graphically. Before using either tool you must designate one of your vector layers as *active*. (The default active layer is the first one that you imported.) Either use the **Active Layer** drop-down menu at the bottom of your screen or select the layer in the **Layers** menu, slide right, and select **Active**. Having a designated active layer ensures that when you click a feature you don't inadvertently select an overlapping feature from a different layer.

- **Datatip tool**: The **Datatip** tool displays a feature attribute in a text label each time you click a vector feature. By default the attribute is the first one in the layer's attribute list. To change which attribute is used, select the layer in the **Layers** menu, slide right, and click `Set Layer Attribute`. In the dialog that follows, select a different attribute, or `Index`. If you choose `Index`, the Map Viewer displays the one-based index value corresponding to a given feature—based on its position in the input file or workspace array. To remove a text label, right-click it and choose **Delete datatip** from the context menu. Or choose **Delete all datatips** from the context menu or the **Tools** menu.

- **Info tool**: The **Info** tool opens a separate text window each time you click a vector feature. The window displays all the attribute names and values for that feature, in contrast to the **Datatip** tool, which displays only the value of a single attribute. If you need to compare two or more features, simply click each one and view the info windows together. Use its close button to close an info window when you're done with it, or choose **Close All Info Windows** from the **Tools** menu.

Annotating Your Map

Use the **text**, **line**, or **arrow** annotation tools to mark and highlight points of interest on your map, or select the corresponding items in the **Insert** menu. Use the **Select annotations** tool and **Edit** menu to modify or remove your annotations. The Map Viewer manages annotations separately from data layers; annotations always stay on top. Note that annotations cannot be saved as graphic objects, although you can export maps containing annotations to an image format as described below.

Creating and Using Additional Views

Use **New View** on the **File** menu to create an additional Map Viewer window linked to an existing window. Consider using an additional window when you want to see your map at different scales at the same time (e.g., a detailed view plus an overview), or when you want to simultaneously see

different areas of the map at large scale. You can create as many additional windows as you need, and close them when you want. Your `mapview` session ends when you close the last window.

Options for creating a new viewer window include: **Duplicate Current View**, **Full Extent**, **Full Extent of Active Layer**, and **Selected Area**. Click and drag with the **Select area** tool to define a selected area.

A new viewer window differs from existing windows mainly in terms of the visible map extent and scale (it also omits annotations and any labels you added with the datatip tool). You will see the same layers in the same order with the same settings (including the active layer). Updates to layers (insertion/removal, order, visibility, label attribute, and symbolization) in one viewer window are propagated automatically to all the windows with which it is linked. Updates to annotations and datatip labels are not propagated between viewers. If you need two different layer configurations in different windows, launch a second `mapview` from the command line instead of creating an additional window. The views it contains will not be linked to previous ones.

Exporting Your Map

The Map Viewer allows you to export all or part of your map for use in a publication or on a Web page. Use **File > Save As Raster Map** to export an image of either the current display extent or an area outlined with the **Select area** tool. Select a format (PNG, TIFF, JPEG) from the drop-down menu in the export dialog. For maps including vector layers, PNG (Portable Network Graphics) is often the best choice. This format provides excellent quality, good compression, and is well supported by modern Web browsers. The export process automatically creates a world file (ending with suffix `tfw`, `jgw`, or `pgw`) as well; the pair of files constitute a georeferenced image that itself can be displayed with `mapview`, `mapshow`, and many external GIS packages.

Limitations

The **Select area** tool is not supported in MATLAB Online. To view a particular region on the map, use the **Zoom in**, **Zoom out**, and **Pan** tools instead.

See Also
geoshow | readgeoraster | makesymbolspec | mapshow | shaperead | updategeostruct | worldfileread

Introduced before R2006a

mdistort

Display contours of constant map distortion

Syntax

```
mdistort
mdistort off
mdistort parameter
mdistort(parameter,levels)
mdistort(parameter,levels,gsize)
h = mdistort(...)
```

Description

`mdistort`, with no input arguments, toggles the display of contours of projection-induced distortion on the current map axes. The magnitude of the distortion is reported in percent.

`mdistort off` removes the contours.

`mdistort parameter` displays contours of distortion for the specified parameter. Units are in percent deviation (except for angles for which degrees are used).

Parameter	Value
`'area'`	
`'angles'`	maximum angular distortion of right angles
`'scale'` or `'maxscale'`	maximum scale (the default)
`'minscale'`	minimum scale
`'parscale'`	scale along the parallels
`'merscale'`	scale along the meridians
`'scaleratio'`	ratio of maximum and minimum scale

`mdistort(parameter,levels)` specifies the levels for which the contours are drawn. `levels` is a vector of values as used by `contour`. If empty, the default levels are used.

`mdistort(parameter,levels,gsize)` controls the size of the underlying graticule matrix used to compute the contours. `gsize` is a two-element vector containing the number of rows and columns. If omitted, the default Mapping Toolbox graticule size of `[50 100]` is assumed.

`h = mdistort(...)` returns a handle to the contour group object containing the contours and text.

Background

Map projections inevitably introduce distortions in the shape and size of objects as they are transformed from three-dimensional spherical coordinates to two-dimensional Cartesian coordinates. The amount and type of distortion vary between projections, over the projection, and with the selection of projection parameters such as standard parallels. This function provides a quantitative graphical display of distortion parameters.

mdistort is not intended for use with UTM. Distortion is minimal within a given UTM zone. mdistort issues a warning if a UTM projection is encountered.

Examples

View Extreme Area Distortion of Mercator Projection

The extreme area distortion of the Mercator projection. This makes it ill-suited for global displays.

```
figure
axesm mercator
load coastlines
framem;
plotm(coastlat,coastlon,'color',.5*[1 1 1])
mdistort('area', [1 30 100 200 500 1000])
```
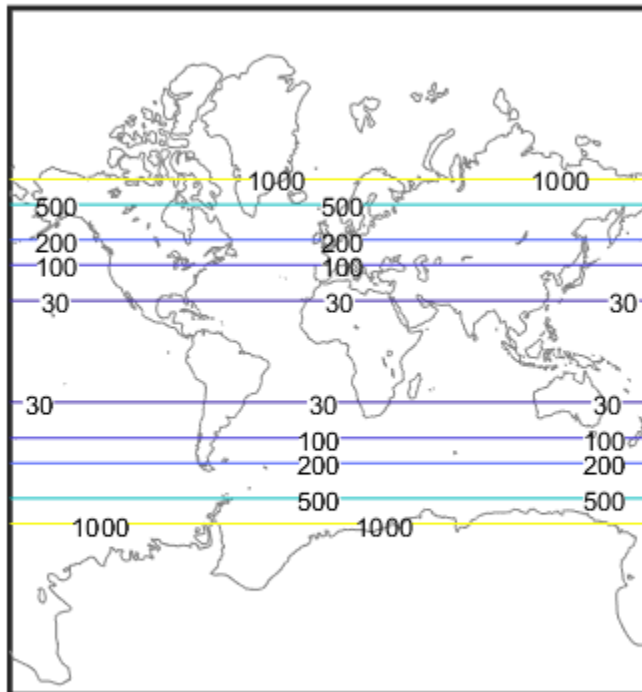

View Lines of Zero Distortion for Bonne Projection

The lines of zero distortion for the Bonne projection follow the central meridian and the standard parallel.

```
figure
axesm bonne
```

```
load coastlines
framem;
plotm(coastlat,coastlon,'color',.5*[1 1 1])
mdistort('angles', 0:15:90)
parallelui
```
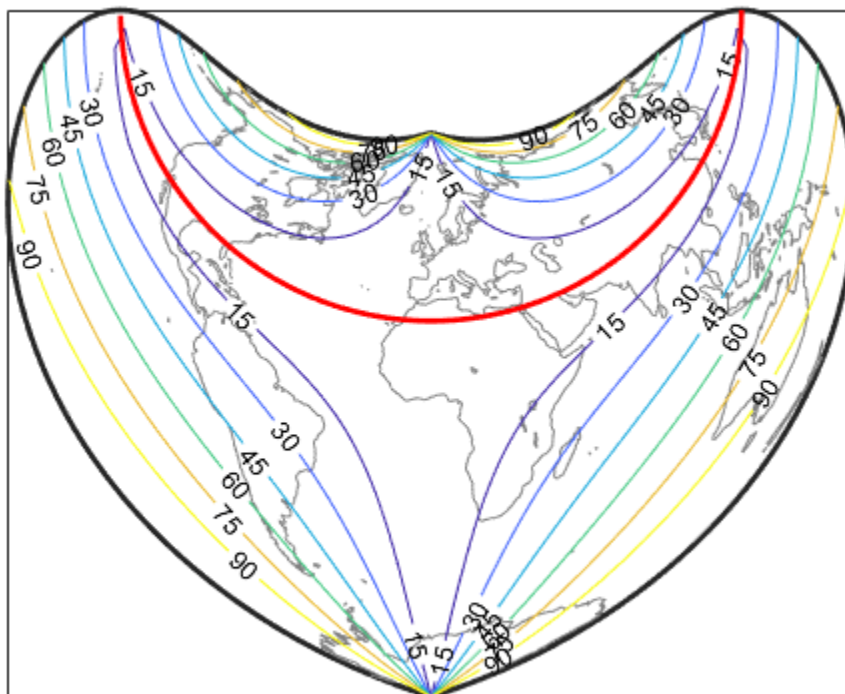

View Distortion of Conic Projection with Properly Chosen Parallels

An equidistant conic projection with properly chosen parallels can map the conterminous United States with less than 1.5% distortion.

```
figure
usamap conus
load conus
patchm(uslat, uslon, [1 0.7 0])
plotm(statelat, statelon)
patchm(gtlakelat, gtlakelon, 'cyan')
framem off; gridm off; mlabel off; plabel off
mdistort('parscale', -2:.5:2)
parallelui
```
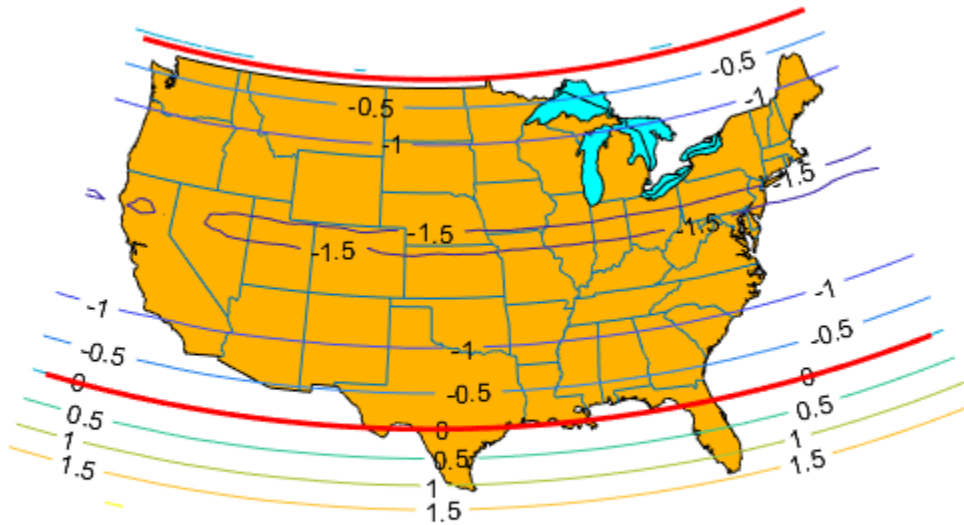
View Contours of Scale Ratios

Create a Hammer projection map axes in normal aspect and plot a graticule and frame. Load the coast data set and plot it as gray patches.

```
figure
axesm('MapProjection','hammer','Grid','on','Frame','on')
load coastlines
patchm(coastlat,coastlon,'FaceColor',[0.95 0.95 0.95])
```

Plot contours of minimum-to-maximum scale ratios by using the `mdistort` function. Notice that the region of minimum distortion is centered around (0,0).

```
mdistort('scaleratio')
```
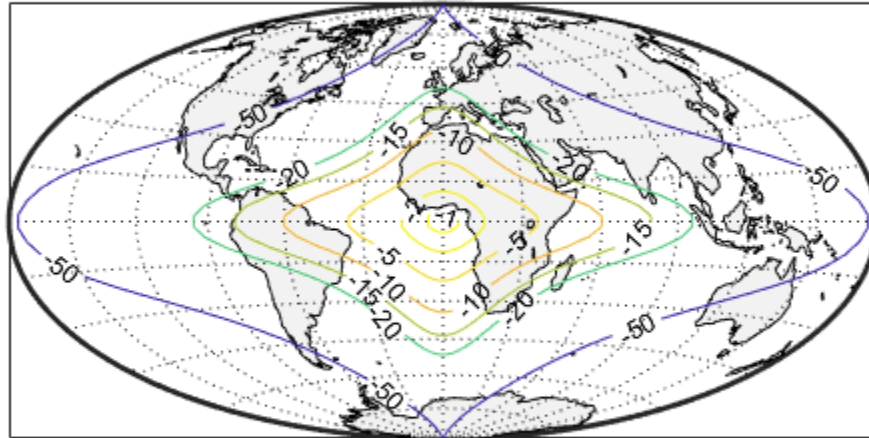
Repeat this diagram with a Bonne projection in a new figure window. Notice that the region of minimum distortion is centered around (30,0) which is where the single standard parallel is.

```
figure
axesm('MapProjection','bonne','Grid','on','Frame','on')
patchm(coastlat,coastlon,'FaceColor',[0.95 0.95 0.95])
mdistort('scaleratio')
```
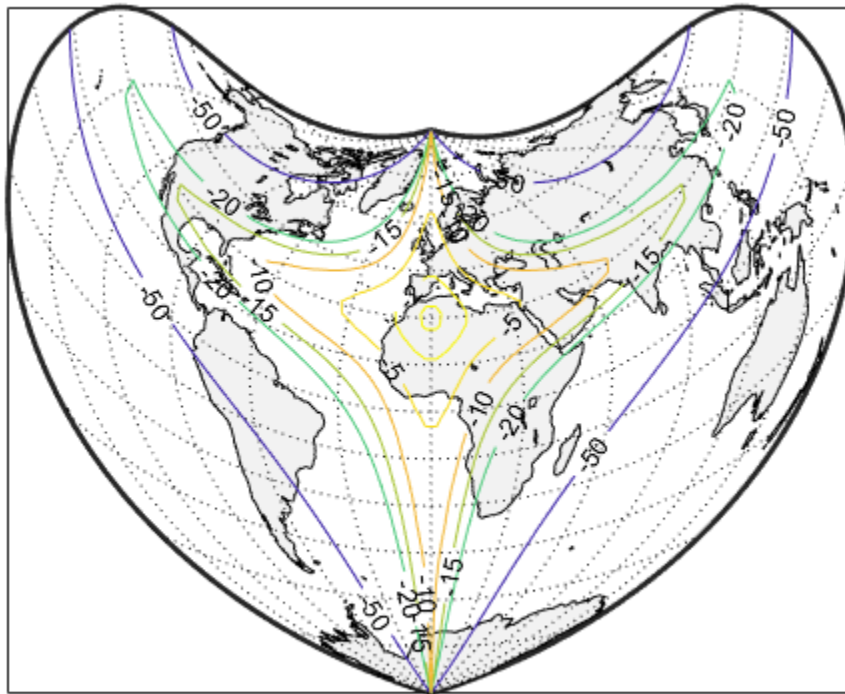
Tips

`mdistort` can help in the placement of standard parallels for projections. Standard parallels are generally placed to minimize distortion over the region of interest. The default parallel locations might not be appropriate for maps of smaller regions. By using `mdistort` and `parallelui`, you can immediately see how the movement of parallels reduces distortion.

See Also

`tissot` | `distortcalc` | `vfwdtran`

meanm

Mean location of geographic coordinates

Syntax

```
[latmean,lonmean] = meanm(lat,lon)
[latmean,lonmean] = meanm(lat,lon,units)
[latmean,lonmean] = meanm(lat,lon,ellipsoid)
```

Description

`[latmean,lonmean] = meanm(lat,lon)` returns row vectors of the geographic mean positions of the columns of the input latitude and longitude points.

`[latmean,lonmean] = meanm(lat,lon,units)` indicates the angular units of the data. The default angle unit is `'degrees'`.

`[latmean,lonmean] = meanm(lat,lon,ellipsoid)` specifies the shape of the Earth using `ellipsoid`, which can be a `referenceSphere`, `referenceEllipsoid`, or `oblateSpheroid` object, or a vector of the form `[semimajor_axis eccentricity]`. The default ellipsoid model is a spherical Earth, which is sufficient for most applications.

If a single output argument is used, then `geomeans = [latmean,longmean]`. This is particularly useful if the original `lat` and `lon` inputs are column vectors.

Background

Finding the mean position of geographic points is more complicated than simply averaging the latitudes and longitudes. `meanm` determines mean position through three-dimensional vector addition. See "Geographic Statistics for Point Locations on a Sphere" in the *Mapping Toolbox User's Guide*.

Examples

Find Mean Position of Geographic Points

Create some random latitudes.

```
rng(0,'twister')
lats = rand(3)
```

```
lats = 3×3

    0.8147    0.9134    0.2785
    0.9058    0.6324    0.5469
    0.1270    0.0975    0.9575
```

Create some random longitudes.

```
lons = rand(3)
```

```
lons = 3×3

    0.9649    0.9572    0.1419
    0.1576    0.4854    0.4218
    0.9706    0.8003    0.9157
```

Calculate the mean positions of the input geographic positions.

```
[latmean,lonmean] = meanm(lats,lons,'radians');
```

```
[latmean,lonmean]
```

```
ans = 1×6

    0.6519    0.5581    0.6146    0.7587    0.7351    0.4250
```

See Also
filterm | hista | histr | stdist | stdm

meridianarc

Ellipsoidal distance along meridian

Syntax

```
s = meridianarc(phi1,phi2,ellipsoid)
```

Description

`s = meridianarc(phi1,phi2,ellipsoid)` calculates the (signed) distance `s` between latitudes `phi1` and `phi2` along a meridian on the ellipsoid defined by `ellipsoid`, which can be a `referenceSphere`, `referenceEllipsoid`, or `oblateSpheroid` object, or a vector of the form `[semimajor_axis eccentricity]`. Latitudes `phi1` and `phi2` are in radians. The distance `s` has the same units as the semimajor axis of the ellipsoid. If `phi2` is less than `phi1`, `s` is negative.

See Also

meridianfwd

Introduced in R2007a

meridianfwd

Reckon position along meridian

Syntax

```
phi2 = meridianfwd(phi1,s,ellipsoid)
```

Description

`phi2 = meridianfwd(phi1,s,ellipsoid)` determines the geodetic latitude `phi2` reached by starting at geodetic latitude `phi1` and traveling distance `s` north (positive `s`) or south (negative `s`) along a meridian on the specified `ellipsoid`. `ellipsoid` is a `referenceSphere`, `referenceEllipsoid`, or `oblateSpheroid` object, or a vector of the form `[semimajor_axis eccentricity]`. Latitudes `phi1` and `phi2` are in radians, and `s` has the same units as the semimajor axis of the ellipsoid.

See Also

meridianarc

Introduced in R2007a

meshgrat

(To be removed) Construct map graticule for surface object display

---

Note The `meshgrat` function will be removed in a future release. Use the `geographicGrid`, `linspace`, or `ndgrid` function instead. For more information, see "Compatibility Considerations".

Syntax

```
[lat, lon] = meshgrat(Z, R)
[lat, lon] = meshgrat(Z, R, gratsize)
[lat, lon] = meshgrat(lat, lon)
[lat, lon] = meshgrat(latlim, lonlim, gratsize)
[lat, lon] = meshgrat(lat, lon, angleunits)
[lat, lon] = meshgrat(latlim, lonlim, angleunits)
[lat, lon] = meshgrat(latlim, lonlim, gratsize, angleunits)
```

Description

`[lat, lon] = meshgrat(Z, R)` constructs a graticule for use in displaying a regular data grid, Z. In typical usage, a latitude-longitude graticule is projected, and the grid is warped to the graticule using MATLAB graphics functions. In this two-argument calling form, the graticule size is equal to the size of Z. R can be a geographic raster reference object, a referencing vector, or a referencing matrix.

If R is a geographic raster reference object, its `RasterSize` property must be consistent with `size(Z)`.

If R is a referencing vector, it must be 1-by-3 with elements:

```
[cells/degree northern_latitude_limit western_longitude_limit]
```

If R is a referencing matrix, it must be 3-by-2 and transform raster row and column indices to/from geographic coordinates according to:

```
[lon lat] = [row col 1] * R
```

If R is a referencing matrix, it must define a (non-rotational, non-skewed) relationship in which each column of the data grid falls along a meridian and each row falls along a parallel.

`[lat, lon] = meshgrat(Z, R, gratsize)` produces a graticule of size `gratsize`. `gratsize` is a two-element vector of the form `[number_of_parallels number_of_meridians]`. If `gratsize = []`, then the graticule returned has the default size 50-by-100. (But if `gratsize` is omitted, a graticule of the same size as Z is returned.) A finer graticule uses larger arrays and takes more memory and time but produces a higher fidelity map.

`[lat, lon] = meshgrat(lat, lon)` takes the vectors `lat` and `lon` and returns graticule arrays of size `numel(lat)-by-numel(lon)`. In this form, `meshgrat` is similar to the MATLAB function `meshgrid`.

`[lat, lon] = meshgrat(latlim, lonlim, gratsize)` returns a graticule mesh of size `gratsize` that covers the geographic limits defined by the two-element vectors `latlim` and `lonlim`.

[lat, lon] = meshgrat(lat, lon, angleunits), [lat, lon] = meshgrat(latlim, lonlim, angleunits), and [lat, lon] = meshgrat(latlim, lonlim, gratsize, angleunits) where angleunits can be either 'degrees' (the default) or 'radians'.

The graticule mesh is a grid of points that are projected on a map axes and to which surface map objects are warped. The fineness, or resolution, of this grid determines the quality of the projection and the speed of plotting. There is no hard and fast rule for sufficient graticule resolution, but in general, cylindrical projections need very few graticules in the longitudinal direction, while complex curve-generating projections require more.

Examples

Make a (coarse) graticule for the entire world:

```
latlim = [-90 90];
lonlim = [-180 180];
[lat,lon] = meshgrat(latlim,lonlim,[3 6])

lat =

   -90.0000   -90.0000   -90.0000   -90.0000   -90.0000   -90.0000
         0          0          0          0          0          0
    90.0000    90.0000    90.0000    90.0000    90.0000    90.0000


lon =

  -180.0000 -108.0000   -36.0000    36.0000   108.0000   180.0000
  -180.0000 -108.0000   -36.0000    36.0000   108.0000   180.0000
  -180.0000 -108.0000   -36.0000    36.0000   108.0000   180.0000
```

These paired coordinates are the graticule vertices, which are projected according to the requirements of the desired map projection. Then data such as the elevation data in topo60c can be warped to the grid.

Compatibility Considerations

meshgrat will be removed
Not recommended starting in R2021a

Some functions that accept referencing vectors or referencing matrices as input will be removed, including the meshgrat function. Depending on the meshgrat function syntax, use the geographicGrid, linspace, or ndgrid function instead.

- If your code uses the meshgrat function and a referencing vector or referencing matrix, replace the code by using a geographic raster reference object and the geographicGrid function. Reference objects have several advantages over referencing vectors and matrices.

  - Unlike referencing vectors and matrices, reference objects have properties that document the size of the associated raster, its geographic limits, and the direction of its rows and columns. For more information about reference object properties, see the GeographicCellsReference and GeographicPostingsReference objects.

  - You can manipulate the limits of rasters associated with reference objects using the geocrop function.

- You can manipulate the size and resolution of rasters associated with reference objects using the `georesize` function.
- Most functions that accept referencing vectors or matrices as inputs also accept reference objects.

Create a reference object for either a raster of cells using the `georefcells` function or a raster of regularly posted samples using the `georefpostings` function. Alternatively, convert from a referencing vector or a referencing matrix to a reference object using the `refvecToGeoRasterReference` or `refmatToGeoRasterReference` function, respectively.

Replace uses of the `meshgrat` function with the `geographicGrid` function according to these patterns.

Will Be Removed	Recommended
`[lat,lon] = meshgrat(A,R);`	`[lat,lon] = geographicGrid(R);`

`[lat,lon] = meshgrat(A,R,gratsize);`	`Rg = R;`
	`Rg.RasterSize = gratsize;`
	`[lat,lon]= geographicGrid(Rg);`

- If your code uses the `meshgrat` function and latitude and longitude vectors, replace the code by using the `ndgrid` function.

Will Be Removed	Recommended
`[lat,lon] = meshgrat(lat,lon);`	`[lat,lon] = ndgrid(lat,lon);`

- If your code uses the `meshgrat` function, latitude and longitude limits, and a graticule size, replace the code by using the `linspace` and `ndgrid` functions.

Will Be Removed	Recommended

`[lat,lon] = meshgrat(latlim,lonlim,gratsize);`	`latv = linspace(latlim(1),latlim(2),gratsize(1));`
	`lonv = linspace(lonlim(1),lonlim(2),gratsize(2));`
	`[lat,lon] = ndgrid(latv,lonv);`

See Also

`meshgrid` | `meshm` | `surfacem` | `surfm` | `geographicGrid` | `ndgrid` | `linspace`

meshlsrm

3-D lighted shaded relief of regular data grid

Syntax

```
meshlsrm(Z,R)
meshlsrm(Z,R,[azim elev])
meshlsrm(Z,R,[azim elev],cmap)
meshlsrm(Z,R,[azim elev],cmap,clim)
h = meshlsrm(...)
```

Description

`meshlsrm(Z,R)` displays the regular data grid Z colored according to elevation and surface slopes. R can be a referencing vector, a referencing matrix, or a geographic raster reference object.

If R is a geographic raster reference object, its `RasterSize` property must be consistent with `size(Z)`.

If R is a referencing vector, it must be a 1-by-3 with elements:

```
[cells/degree northern_latitude_limit western_longitude_limit]
```

If R is a referencing matrix, it must be 3-by-2 and transform raster row and column indices to/from geographic coordinates according to:

```
[lon lat] = [row col 1] * R
```

If R is a referencing matrix, it must define a (non-rotational, non-skewed) relationship in which each column of the data grid falls along a meridian and each row falls along a parallel. By default, shading is based on a light to the east (90 deg.) at an elevation of 45 degrees. Also by default, the colormap is constructed from 16 colors and 16 grays. Lighting is applied before the data is projected. The current axes must have a valid map projection definition.

`meshlsrm(Z,R,[azim elev])` displays the regular data grid Z with the light coming from the specified azimuth and elevation. Angles are specified in degrees, with the azimuth measured clockwise from North, and elevation up from the zero plane of the surface.

`meshlsrm(Z,R,[azim elev],cmap)` displays the regular data grid Z using the specified colormap. The number of grayscales is chosen to keep the size of the shaded colormap below 256. If the vector of `azimuth` and `elevation` is empty, the default locations are used. Color axis limits are computed from the data.

`meshlsrm(Z,R,[azim elev],cmap,clim)` uses the provided color axis limits, which by default are computed from the data.

`h = meshlsrm(...)` returns the handle to the surface drawn.

Examples

Load elevation data and a geographic cells reference object for the Korean peninsula. Create a world map using appropriate latitude and longitude limits for the peninsula. Then, display a lighted shaded relief map. By default, `meshlsrm` applies a colormap appropriate for elevation data.

```
load korea5c
worldmap(korea5c,korea5cR)
meshlsrm(korea5c,korea5cR,[45 65])
```
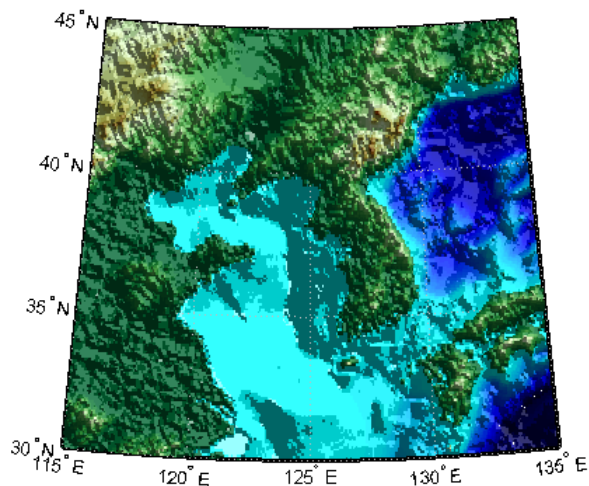

Tips

This function effectively multiplies two colormaps, one with color based on elevation, the other with a grayscale based on the slope of the surface, to create a new colormap. This produces an effect similar to using a light on a surface, but with all of the visible colors actually in the colormap. Lighting calculations are performed on the unprojected data.

See Also

geographicGrid | ndgrid | meshm | pcolorm | surfacem | surflm | surflsrm

Introduced before R2006a

meshm

Project regular data grid on map axes

Syntax

```
meshm(Z,R)
meshm(Z,R,gratsize)
meshm(Z,R,gratsize,alt)
meshm( ___ ,param1,val1,param2,val2,...)
H = meshm( ___ )
```

Description

meshm(Z,R) will display the regular data grid Z warped to the default projection graticule. R can be a referencing vector, a referencing matrix, or a geographic raster reference object.

If R is a geographic raster reference object, its RasterSize property must be consistent with size(Z).

If R is a referencing vector, it must be a 1-by-3 with elements:

```
[cells/degree northern_latitude_limit western_longitude_limit]
```

If R is a referencing matrix, it must be 3-by-2 and transform raster row and column indices to/from geographic coordinates according to:

```
[lon lat] = [row col 1] * R
```

If R is a referencing matrix, it must define a (non-rotational, non-skewed) relationship in which each column of the data grid falls along a meridian and each row falls along a parallel. The current axes must have a valid map projection definition.

meshm(Z,R,gratsize) displays a regular data grid warped to a graticule mesh defined by the 1-by-2 vector gratsize. gratsize(1) indicates the number of lines of constant latitude (parallels) in the graticule, and gratsize(2) indicates the number of lines of constant longitude (meridians).

meshm(Z,R,gratsize,alt) displays the regular surface map at the altitude specified by alt. If alt is a scalar, then the grid is drawn in the z = alt plane. If alt is a matrix, then size(alt) must equal gratsize, and the graticule mesh is drawn at the altitudes specified by alt. If the default graticule is desired, set gratsize = [].

meshm(___ ,param1,val1,param2,val2,...) uses optional parameter name-value pairs to control the properties of the surface object constructed by meshm. (If data is placed in the UserData property of the surface, then the projection of this object cannot be altered once displayed.)

H = meshm(___) returns the handle to the surface drawn.

Examples

Load elevation data and a geographic cells reference object for the Korean peninsula. Then, display the data on a world map. Apply a colormap appropriate for elevation data using demcmap.

```
load korea5c
worldmap(korea5c,korea5cR)
meshm(korea5c,korea5cR)
demcmap(korea5c)
```
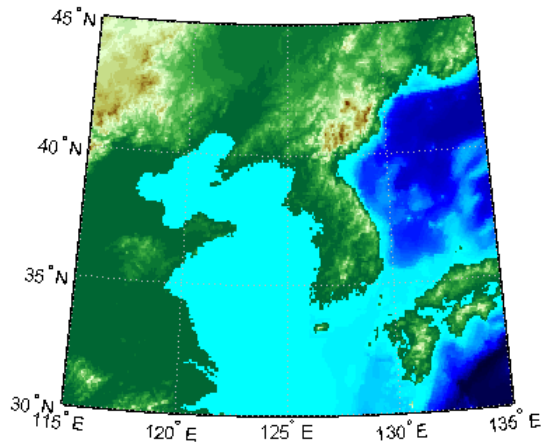

See Also

ndgrid | geoshow | mapshow | geographicGrid | pcolorm | surfacem | surfm

Introduced before R2006a

mfwdtran

(To be removed) Project geographic features to map coordinates

---

Note `mfwdtran` will be removed in a future release. In most cases, use the `projfwd` function instead. If the `mapprojection` property of the current map axes or specified map projection structure is `'globe'`, then use the `geodetic2ecef` function instead. For more information, see "Compatibility Considerations".

Syntax

```
[x,y] = mfwdtran(lat,lon)
[x,y,z] = mfwdtran(lat,lon,alt)
[...] = mfwdtran(mstruct,...)
```

Description

`[x,y] = mfwdtran(lat,lon)` applies the forward transformation defined by the map projection in the current map axes. You can use this function to convert point locations and line and polygon vertices given in latitudes and longitudes to a planar, projected map coordinate system.

`[x,y,z] = mfwdtran(lat,lon,alt)` applies the forward projection to 3-D input, resulting in 3-D output. If the input `alt` is empty or omitted, then `alt = 0` is assumed.

`[...] = mfwdtran(mstruct,...)` requires a valid map projection structure as the first argument. In this case, no map axes is needed.

Examples

Transform Geographic Data into Map Coordinates

Get geographic location data for the District of Columbia.

```
dc = shaperead('usastatelo', 'UseGeoCoords', true,...
    'Selector',{@(name) strcmpi(name,'District of Columbia'),...
    'Name'});
lat = [dc.Lat]';
lon = [dc.Lon]';
[lat lon]

ans =

   38.9000   -77.0700
   38.9500   -77.1200
   39.0000   -77.0300
   38.9000   -76.9000
   38.7800   -77.0300
   38.8000   -77.0200
   38.8700   -77.0200
   38.9000   -77.0700
```

```
        38.9000  -77.0700
           NaN       NaN
```

Obtain the UTM zone for this point.

```
dczone = utmzone(mean(lat,'omitnan'),mean(lon,'omitnan'))

dczone =

    '18S'
```

Set up the UTM coordinate system based on this information.

```
utmstruct = defaultm('utm');
utmstruct.zone = dczone;
utmstruct.geoid = wgs84Ellipsoid;
utmstruct = defaultm(utmstruct);
```

Project the District of Columbia data from geographic coordinates into map coordinates for UTM zone 18S.

```
[x,y] = mfwdtran(utmstruct,lat,lon)

x =

   1.0e+05 *

    3.2049
    3.1629
    3.2421
    3.3524
    3.2367
    3.2459
    3.2476
    3.2049
    3.2049
       NaN


y =

   1.0e+06 *

    4.3077
    4.3134
    4.3187
    4.3074
    4.2943
    4.2965
    4.3043
    4.3077
    4.3077
       NaN
```

Compatibility Considerations

mfwdtran will be removed
Not recommended starting in R2020b

The mfwdtran function will be removed in a future release. In most cases, use the projfwd function instead. If the mapprojection field of the current map axes or specified map projection structure is 'globe', then use the geodetic2ecef function instead.

This table shows some typical usages of the mfwdtran function and how to update your code to use the projfwd function instead.

Will Be Removed	Recommended
[x,y,z] = mfwdtran(lat,lon);	mstruct = gcm; [x,y] = projfwd(mstruct,lat,lon);
[x,y,z] = mfwdtran(mstruct,lat,lon);	[x,y] = projfwd(mstruct,lat,lon);

This table shows some typical usages of the mfwdtran function and how to update your code to use the geodetic2ecef function instead.

Will Be Removed	Recommended
[x,y,z] = mfwdtran(lat,lon,alt);	mstruct = gcm; [x,y,z] = geodetic2ecef(lat,lon,alt,mstruct.geoid);
[x,y,z] = mfwdtran(mstruct,lat,lon,alt);	[x,y,z] = geodetic2ecef(lat,lon,alt,mstruct.geoid);

If the value of the mapprojection field of the map projection structure is listed in this table, then all linear units in the map projection structure must be in meters. In addition, the projfwd function returns coordinates in meters. You can convert the coordinates to other units using the unitsratio function.

Value	Projection Name
'tranmerc'	Transverse Mercator
'mercator'	Mercator
'lambertstd'	Lambert Conformal Conic
'eqaazim'	Lambert Azimuthal Equal Area
'eqaconicstd'	Albers Equal-Area Conic
'eqdazim'	Azimuthal Equidistant
'eqdconicstd'	Equidistant Conic
'ups'	Polar Stereographic
'stereo'	Oblique Stereographic
'eqdcylin'	Equirectangular
'cassinistd'	Cassini-Soldner
'gnomonic'	Gnomonic
'miller'	Miller Cylindrical
'ortho'	Orthographic
'polyconstd'	Polyconic
'robinson'	Robinson
'sinusoid'	Sinusoidal
'vgrint1'	Van der Grinten

Value	Projection Name
`'eqacylin'`	Equal Area Cylindrical

See Also
defaultm | gcm | projfwd | projinv | vfwdtran | vinvtran | geodetic2ecef

Introduced before R2006a

minaxis

Semiminor axis of ellipse

Note Support for nonscalar input, including the syntax `b = minaxis(vec)`, will be removed in a future release.

Syntax

```
b = minaxis(semimajor,e)
b = minaxis(vec)
```

Description

`b = minaxis(semimajor,e)` computes the semiminor axis of an ellipse (or ellipsoid of revolution) given the semimajor axis and eccentricity. The input data can be scalar or matrices of equal dimensions.

`b = minaxis(vec)` assumes a 2 element vector (`vec`) is supplied, where `vec = [semimajor, e]`.

See Also
axes2ecc | flat2ecc | majaxis | n2ecc

Introduced before R2006a

minvtran

(To be removed) Unproject features from map to geographic coordinates

---

Note `minvtran` will be removed in a future release. In most cases, use the `projinv` function instead. If the `mapprojection` field of the current map axes or specified map projection structure is `'globe'`, then use the `ecef2geodetic` function instead. For more information, see "Compatibility Considerations".

Syntax

```
[lat,lon] = minvtran(x,y)
[lat,lon,alt] = minvtran(x,y,z)
[...] = minvtran(mstruct,...)
```

Description

`[lat,lon] = minvtran(x,y)` applies the inverse transformation defined by the map projection in the current map axes. Using `minvtran`, you can convert point locations and line and polygon vertices in a planar, projected map coordinate system to latitudes and longitudes.

`[lat,lon,alt] = minvtran(x,y,z)` applies the inverse projection to 3-D input, resulting in 3-D output. If the input Z is empty or omitted, then `Z = 0` is assumed.

`[...] = minvtran(mstruct,...)` takes a valid map projection structure as the first argument. In this case, no map axes is needed.

Examples

Before using `minvtran`, it is necessary to create a map projection structure. You can do this with `axesm` or the `defaultm` function:

```
mstruct = defaultm('mercator');
mstruct.origin = [38.89 -77.04 0];
mstruct = defaultm(mstruct);
```

The following latitude and longitude data for the District of Columbia is obtained from the `usastatelo` shapefile:

```
dc = shaperead('usastatelo', 'UseGeoCoords', true,...
     'Selector',{@(name) strcmpi(name,'District of Columbia'),...
     'Name'});
lat = [dc.Lat]';
lon = [dc.Lon]';
[lat lon]

ans =

   38.9000   -77.0700
   38.9500   -77.1200
   39.0000   -77.0300
```

```
    38.9000   -76.9000
    38.7800   -77.0300
    38.8000   -77.0200
    38.8700   -77.0200
    38.9000   -77.0700
    38.9000   -77.0700
       NaN        NaN
```

This data can be projected into Cartesian coordinates of the Mercator projection using the `projfwd` function:

```
[x,y] = projfwd(mstruct,lat,lon);
[x y]

ans =

   -0.0004    0.5745
   -0.0011    0.5753
    0.0001    0.5762
    0.0019    0.5745
    0.0001    0.5724
    0.0003    0.5727
    0.0003    0.5739
   -0.0004    0.5745
   -0.0004    0.5745
      NaN       NaN
```

To transform the projected *x-y* data back into the unprojected geographic system, use the `minvtran` function:

```
[lat2,lon2] = minvtran(mstruct,x,y);
[lat2 lon2]

ans =

   70.1302   -77.0987
   70.1729   -77.1969
   70.2157   -77.0204
   70.1300   -76.7659
   70.0276   -77.0205
   70.0447   -77.0010
   70.1046   -77.0009
   70.1302   -77.0987
   70.1302   -77.0987
      NaN       NaN
```

Compatibility Considerations

`minvtran` will be removed
Not recommended starting in R2020b

The `minvtran` function will be removed in a future release. In most cases, use the `projinv` function instead. If the `mapprojection` field of the current map axes or specified map projection structure is `'globe'`, then use the `ecef2geodetic` function instead.

This table shows some typical uses of the `minvtran` function and how to update your code to use the `projinv` function instead.

Will Be Removed	Recommended
`[lat,lon,alt] = minvtran(x,y);`	`mstruct = gcm;` `[lat,lon] = projinv(mstruct,x,y);`
`[lat,lon,alt] = minvtran(mstruct,x,y);`	`[lat,lon] = projinv(mstruct,x,y);`

This table shows some typical uses of the `minvtran` function and how to update your code to use the `ecef2geodetic` function instead.

Will Be Removed	Recommended
`[lat,lon,alt] = minvtran(x,y,z);`	`mstruct = gcm;` `[lat,lon,alt] = ecef2geodetic(x,y,z,mstruct.geoid);`
`[lat,lon,alt] = minvtran(mstruct,x,y,z);`	`[lat,lon,alt] = ecef2geodetic(x,y,z,mstruct.geoid);`

If the value of the `mapprojection` field of the map projection structure is listed in this table, then you must specify all linear units in meters, including coordinates and map projection structure fields. After you have projected the coordinates, you can convert them to other units using the `unitsratio` function.

Value	Projection Name
`'tranmerc'`	Transverse Mercator
`'mercator'`	Mercator
`'lambertstd'`	Lambert Conformal Conic
`'eqaazim'`	Lambert Azimuthal Equal Area
`'eqaconicstd'`	Albers Equal-Area Conic
`'eqdazim'`	Azimuthal Equidistant
`'eqdconicstd'`	Equidistant Conic
`'ups'`	Polar Stereographic
`'stereo'`	Oblique Stereographic
`'eqdcylin'`	Equirectangular
`'cassinistd'`	Cassini-Soldner
`'gnomonic'`	Gnomonic
`'miller'`	Miller Cylindrical
`'ortho'`	Orthographic
`'polyconstd'`	Polyconic
`'robinson'`	Robinson
`'sinusoid'`	Sinusoidal
`'vgrint1'`	Van der Grinten
`'eqacylin'`	Equal Area Cylindrical

See Also

axesm | defaultm | gcm | projfwd | projinv | vfwdtran | vinvtran | ecef2geodetic

Introduced before R2006a

mlabel

Toggle and control display of meridian labels

Syntax

```
mlabel
mlabel('on')
mlabel('off')
mlabel('reset')
mlabel(parallel)
mlabel(MapAxesPropertyName,PropertyValue,...)
```

Description

`mlabel` toggles the visibility of meridian labeling on the current map axes.

`mlabel('on')` sets the visibility of meridian labels to `'on'`.

`mlabel('off')` sets the visibility of meridian labels to `'off'`.

`mlabel('reset')` resets the displayed meridian labels using the currently defined meridian label properties.

`mlabel(parallel)` sets the value of the `MLabelParallel` property of the map axes to the value of `parallel`. This determines the parallel upon which the labels are placed (see `axesm`). The options for `parallel` are a scalar latitude or `'north'`, `'south'`, or `'equator'`.

`mlabel(MapAxesPropertyName,PropertyValue,...)` allows paired map axes' property names and property values to be passed in. For a complete description of map axes properties, see the `axesm` reference page in this guide.

Meridian label handles can be returned in `h` if desired.

See Also

axesm | mlabelzero22pi | plabel | setm

Introduced before R2006a

mlabelzero22pi

Convert meridian labels to 0-360 degree range

Syntax

```
mlabelzero22pi
```

Description

`mlabelzero22pi` displays longitude labels in the range of 0 to 360 degrees east of the prime meridian.

Examples

```
% create a map
figure('color','w'); axesm('miller','grid','on'); tightmap;
mlabel on; plabel on
```
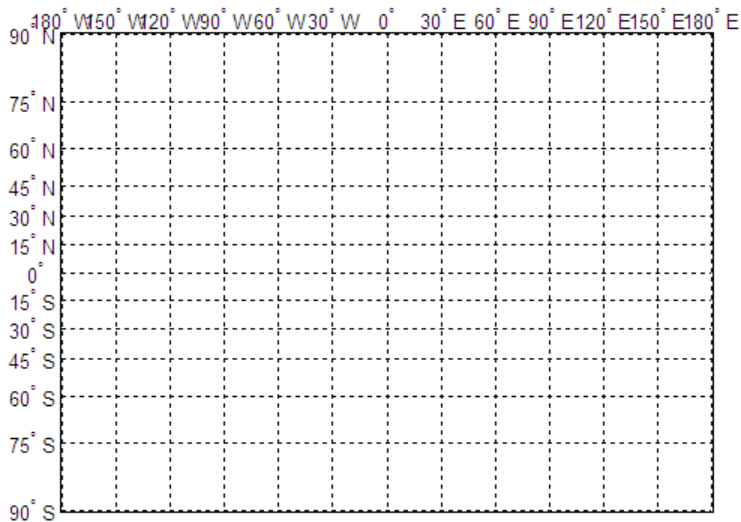


```
% Display longitude labels in the range of 0 to 360 degrees
mlabelzero22pi
```
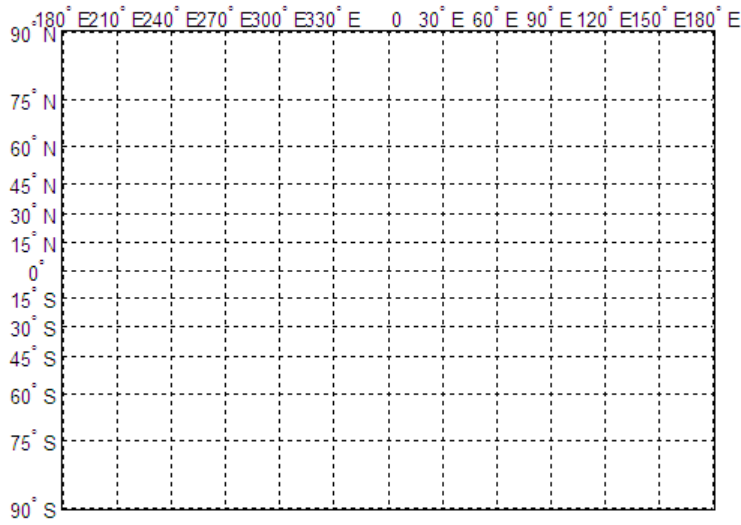
See Also
mlabel

Introduced before R2006a

n2ecc

Eccentricity of ellipse from third flattening

Note Support for nonscalar input, including the special two-column syntax described below, will be removed in a future release.

Syntax

Description

ecc = n2ecc(n) computes the eccentricity of an ellipse (or ellipsoid of revolution) given the parameter n (the "third flattening"). n is defined as (a-b)/(a+b), where a is the semimajor axis and b is the semiminor axis. Except when the input has 2 columns (or is a row vector), each element is assumed to be a third flattening and the output ecc has the same size as n.

ecc = n2ecc(n), where n has two columns (or is a row vector), assumes that the second column is a third flattening, and a column vector is returned.

See Also
axes2ecc | ecc2n

namem

Names of graphics objects

Syntax

```
obj_names = namem
obj_names = namem(h)
```

Description

`obj_names = namem` returns the names of all the objects on the current axes. The name of an object is the value of its `Tag` property, if specified. Otherwise, the name of the object is the value of its `Type` property. The values of these properties are either set at object creation or defined using the `tagm` function.

`obj_names = namem(h)` returns the name of the object (or objects) specified by the array, `h`.

Examples

Get List of Graphics Objects in Map Display

Load elevation data and a geographic cells reference object for the Korean peninsula. Then, display the data on a world map.

```
load korea5c
worldmap(korea5c,korea5cR);
geoshow(korea5c,korea5cR,'DisplayType','texturemap')
demcmap(korea5c)
```
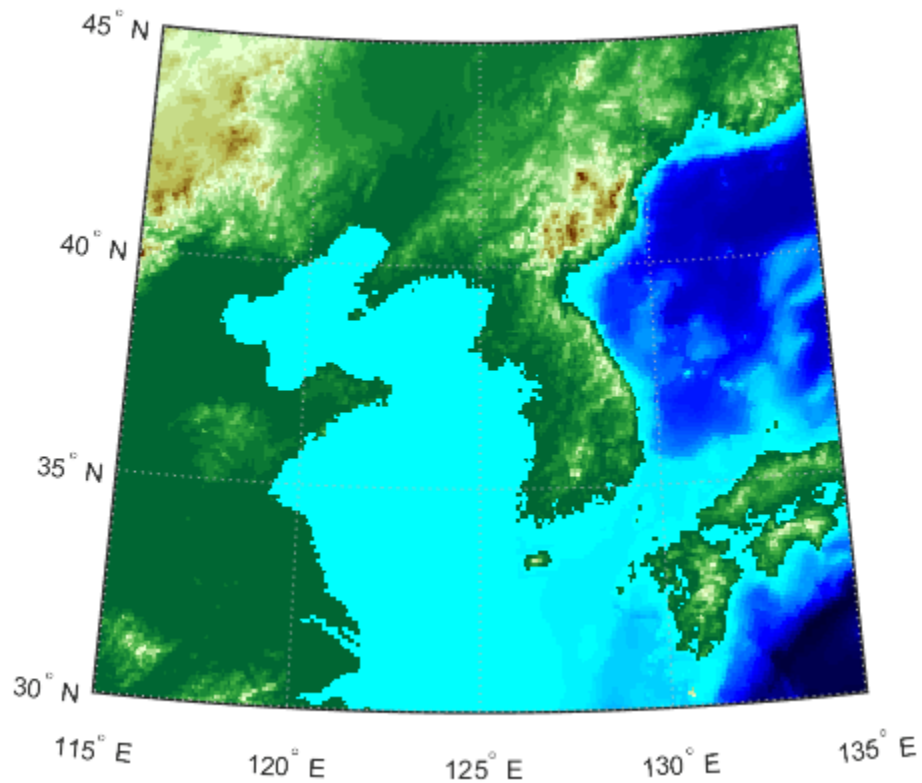
Find the names of graphics objects in the figure. The `namem` function returns the names as a character array. Convert the array into a cell array of character vectors using `cellstr`.

```
namesChar = namem
```

```
namesChar = 6x8 char array
    'PLabel  '
    'MLabel  '
    'Parallel'
    'Meridian'
    'surface '
    'Frame   '
```

```
namesCell = cellstr(namesChar)
```

```
namesCell = 6x1 cell
    {'PLabel'  }
    {'MLabel'  }
    {'Parallel'}
    {'Meridian'}
    {'surface' }
    {'Frame'   }
```

Input Arguments

h — Graphics objects
array

Graphics objects, specified as an array.

Output Arguments

obj_names — Names of graphics objects
character array

Names of graphics objects, returned as a character array. `namem` removes duplicate object names from the array.

Note: Use `cellstr(obj_names)` to convert `obj_names` to a cell array of character vectors.

See Also
`clma` | `clmo` | `handlem` | `hidem` | `showm` | `tagm`

Introduced before R2006a

nanclip

Clip vector data with NaNs at specified pen-down locations

Syntax

```
dataout = nanclip(datain)
dataout = nanclip(datain,pendowncmd)
```

Description

`dataout = nanclip(datain)` and `dataout = nanclip(datain,pendowncmd)` return the pen-down delimited data in the matrix `datain` as NaN-delimited data in `dataout`. When the first column of `datain` equals `pendowncmd`, a segment is started and a NaN is inserted in all columns of `dataout`. The default `pendowncmd` is `-1`.

Pen-down delimited data is a matrix with a first column consisting of pen commands. At the beginning of each segment in the data, this first column has an entry corresponding to a pen-down command. Other entries indicate that the segment is continuing. NaN-delimited data consists of columns of data, each segment of which ends in a NaN in every data column. Since there is no pen command column, the NaN-delimited format can represent the same data in one fewer columns; the remaining columns have more entries, one for each NaN (that is, for each segment).

Examples

```
datain = [-1 45 67; 0 23 54; 0 28 97; -1 47 89; 0 56 12]

datain =
    -1    45    67          % Begin first segment
     0    23    54
     0    28    97
    -1    47    89          % Begin second segment
     0    56    12

dataout = nanclip(datain)

dataout =
    45    67
    23    54
    28    97
   NaN   NaN                % End first segment
    47    89
    56    12
   NaN   NaN                % End second segment
```

See Also
`spcread`

Introduced before R2006a

nanm

(To be removed) Construct regular data grid of NaNs

Note nanm will be removed in a future release. Use the georefcells and NaN functions instead. For more information, see "Compatibility Considerations".

Syntax

```
[Z,refvec] = nanm(latlim,lonlim,scale)
```

Description

[Z,refvec] = nanm(latlim,lonlim,scale) returns a regular data grid consisting entirely of NaNs and a three-element referencing vector for the returned Z. The two-element vectors latlim and lonlim define the latitude and longitude limits of the geographic region. They should be of the form [south north] and [west east], respectively. The scalar scale specifies the number of rows and columns per degree of latitude and longitude.

Examples

```
[Z,refvec] = nanm([46,51],[-79,-75],1)

Z =
    NaN    NaN    NaN    NaN
    NaN    NaN    NaN    NaN
    NaN    NaN    NaN    NaN
    NaN    NaN    NaN    NaN
    NaN    NaN    NaN    NaN
refvec =
     1     51    -79
```

Compatibility Considerations

nanm will be removed
Not recommended starting in R2015b

Some functions that return referencing vectors will be removed, including the nanm function. Instead, create a geographic raster reference object using the georefcells function and a matrix of NaN values using the NaN function. Reference objects have several advantages over referencing vectors.

- Unlike referencing vectors, reference objects have properties that document the size of the associated raster, its geographic limits, and the direction of its rows and columns. For examples of reference object properties, see the GeographicPostingsReference object.

- You can manipulate the limits of rasters associated with geographic reference objects using the geocrop function.

- You can manipulate the size and resolution of rasters associated with geographic reference objects using the georesize function.

- Most functions that accept referencing vectors as inputs also accept reference objects.

This table shows how to update your code to use the `georefcells` and `NaN` functions instead of the `nanm` function.

Will Be Removed	Recommended
`[Z,refvec] = nanm(latlim,lonlim,scale);`	`R = georefcells(latlim,lonlim,1/scale,1/scale);` `Z = NaN(R.RasterSize);`

See Also
ones | sparse | zeros | NaN | georefcells

Introduced before R2006a

navfix

Mercator-based navigational fix

Syntax

```
[latfix,lonfix] = navfix(lat,long,az)
[latfix,lonfix] = navfix(lat,long,range,casetype)
[latfix,lonfix] = navfix(lat,long,az_range,casetype)
[latfix,lonfix] = navfix(lat,long,az_range,casetype,drlat,drlon)
```

Description

`[latfix,lonfix] = navfix(lat,long,az)` returns the intersection points of rhumb lines drawn parallel to the observed bearings, `az`, of the landmarks located at the points `lat` and `long` and passing through these points. One bearing is required for each landmark. Each possible pairing of the n landmarks generates one intersection, so the total number of resulting intersection points is the combinatorial *n choose 2*. The calculation time therefore grows rapidly with n.

`[latfix,lonfix] = navfix(lat,long,range,casetype)` returns the intersection points of Mercator projection circles with radii defined by `range`, centered on the landmarks located at the points `lat` and `long`. One range value is required for each landmark. Each possible pairing of the *n* landmarks generates up to two intersections (circles can intersect twice), so the total number of resulting intersection points is the combinatorial *2 times (n choose 2)*. The calculation time therefore grows rapidly with *n*. In this case, the variable `casetype` is a vector of `0`s the same size as the variable `range`.

`[latfix,lonfix] = navfix(lat,long,az_range,casetype)` combines ranges and bearings. For each element of `casetype` equal to 1, the corresponding element of `az_range` represents an azimuth to the associated landmark. Where `casetype` is a `0`, `az_range` is a range.

`[latfix,lonfix] = navfix(lat,long,az_range,casetype,drlat,drlon)` returns for each possible pairing of landmarks only the intersection that lies closest to the dead reckoning position indicated by `drlat` and `drlon`. When this syntax is used, all included landmarks' bearing lines or range arcs must intersect. If any possible pairing fails, the warning `No Fix` is displayed.

Background

This is a navigational function. It assumes that all latitudes and longitudes are in degrees and all distances are in nautical miles. In navigation, piloting is the practice of fixing one's position based on the observed bearing and ranges *to* fixed landmarks (points of land, lighthouses, smokestacks, etc.) *from* the navigator's vessel. In conformance with navigational practice, bearings are treated as rhumb lines and ranges are treated as the radii of circles on a Mercator projection.

In practice, at least three azimuths (bearings) and/or ranges are required for a usable fix. The resulting intersections are unlikely to coincide exactly. Refer to "Navigation" in the *Mapping Toolbox User's Guide* for a more complete description of the use of this function.

Examples

For a fully illustrated example of the application of this function, refer to the "Navigation" section in the *Mapping Toolbox User's Guide*.

Imagine you have two landmarks, at (15ºN,30.4ºW) and (14.8ºN,30.1ºW). You have a visual bearing to the first of 280º and to the second of 160º. Additionally, you have a range to the second of 12 nm. Find the intersection points:

```
[latfix,lonfix] = navfix([15 14.8 14.8],[-30.4 -30.1 -30.1],...
                         [280 160 12],[1 1 0])

latfix =
    14.9591       NaN
    14.9680   14.9208
    14.9879       NaN
lonfix =
   -30.1599       NaN
   -30.2121   -29.9352
   -30.1708       NaN
```

Here is an illustration of the geometry:

Small dots are intersection points.
A dead reckoning position could be
used to eliminate the inconsistent
intersection.
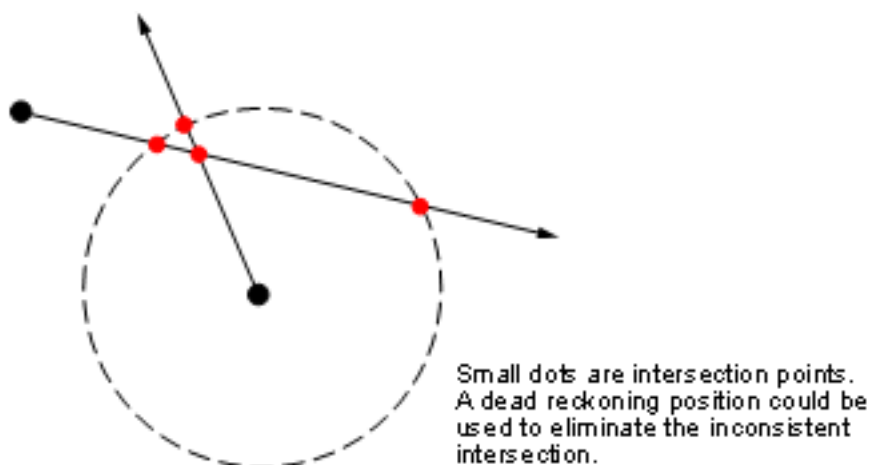
Limitations

Traditional plotting and the `navfix` function are limited to relatively short distances. Visual bearings are in fact great circle azimuths, not rhumb lines, and range arcs are actually arcs of small circles, not of the planar circles plotted on the chart. However, the mechanical ease of the process and the practical limits of visual bearing ranges and navigational radar ranges (~30 nm) make this limitation moot in practice. The error contributed because of these assumptions is minuscule at that scale.

Tips

The outputs of this function are matrices providing the locations of the intersections for all possible pairings of the n entered lines of bearing and range arcs. These matrices therefore have *n-choose-2* rows. In order to allow for two intersections per combination, these matrices have two columns. Whenever there are fewer than two intersections for that combination, one or two NaNs are returned in that row.

When a dead reckoning position is included, these matrices are column vectors.

See Also
crossfix | gcxgc | gcxsc | scxsc | rhxrh | polyxpoly | dreckon | gcwaypts | legs | track

Introduced before R2006a

ned2aer

Transform local north-east-down coordinates to local spherical

Syntax

```
[az,elev,slantRange] = ned2aer(xNorth,yEast,zDown)
[ ___ ] = ned2aer( ___ ,angleUnit)
```

Description

`[az,elev,slantRange] = ned2aer(xNorth,yEast,zDown)` transforms the local north-east-down (NED) Cartesian coordinates specified by `xNorth`, `yEast`, and `zDown` to the local azimuth-elevation-range (AER) spherical coordinates specified by `az`, `elev`, and `slantRange`. Both coordinate systems use the same local origin. Each input argument must match the others in size or be scalar.

`[ ___ ] = ned2aer( ___ ,angleUnit)` specifies the units for azimuth and elevation. Specify `angleUnit` as `'degrees'` (the default) or `'radians'`.

Examples

Calculate AER Coordinates from NED Coordinates

Find the AER coordinates of a landmark with respect to an aircraft, using the NED coordinates of the landmark with respect to the same aircraft.

First, specify the NED coordinates of the landmark. For this example, specify the coordinate values in kilometers.

```
xNorth = -9.1013;
yEast = 4.1617;
zDown = 4.2812;
```

Then, calculate the AER coordinates of the landmark. The azimuth and elevation are specified in degrees. The units for the slant range match the units specified by the NED coordinates. Thus, the slant range is specified in kilometers.

```
[az,elev,slantRange] = ned2aer(xNorth,yEast,zDown)

az = 155.4271

elev = -23.1609

slantRange = 10.8849
```

Reverse the transformation using the `aer2ned` function.

```
[xNorth,yEast,zDown] = aer2ned(az,elev,slantRange)

xNorth = -9.1013
```

```
yEast = 4.1617

zDown = 4.2812
```

Input Arguments

xNorth — NED *x*-coordinates
scalar | vector | matrix | N-D array

NED *x*-coordinates of one or more points in the local NED system, specified as a scalar, vector, matrix, or N-D array.

Data Types: `single` | `double`

yEast — NED *y*-coordinates
scalar | vector | matrix | N-D array

NED *y*-coordinates of one or more points in the local NED system, specified as a scalar, vector, matrix, or N-D array.

Data Types: `single` | `double`

zDown — NED *z*-coordinates
scalar | vector | matrix | N-D array

NED *z*-coordinates of one or more points in the local NED system, specified as a scalar, vector, matrix, or N-D array.

Data Types: `single` | `double`

angleUnit — Angle units
`'degrees'` (default) | `'radians'`

Angle units, specified as `'degrees'` (the default) or `'radians'`.

Output Arguments

az — Azimuth angles
scalar | vector | matrix | N-D array

Azimuth angles of one or more points in the local AER system, returned as a scalar, vector, matrix, or N-D array. Azimuths are measured clockwise from north. Values are specified in degrees within the half-open interval [0 360). To use values in radians, specify the `angleUnit` argument as `'radians'`.

elev — Elevation angles
scalar | vector | matrix | N-D array

Elevation angles of one or more points in the local AER system, returned as a scalar, vector, matrix, or N-D array. Elevations are calculated with respect to the `xNorth-yEast` plane that contains the local origin. If the local origin is on the surface of the spheroid, then the `xNorth-yEast` plane is tangent to the spheroid.

Values are specified in degrees within the closed interval [-90 90]. Positive elevations correspond to negative `zDown` values, and negative elevations correspond to positive `zDown` values. An elevation of

0 indicates that the point lies in the xNorth-yEast plane. To use values in radians, specify the angleUnit argument as 'radians'.

slantRange — Distances from local origin
scalar | vector | matrix | N-D array

Distances from the local origin, returned as a scalar, vector, matrix, or N-D array. Each distance is calculated along a straight, 3-D, Cartesian line. Values are returned in the units specified by xNorth, yEast, and zDown.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also
aer2ned | enu2aer | ned2geodetic | ned2ecef

Topics
"Choose a 3-D Coordinate System"

Introduced in R2012b

ned2ecef

Transform local north-east-down coordinates to geocentric Earth-centered Earth-fixed

Syntax

```
[X,Y,Z] = ned2ecef(xNorth,yEast,zDown,lat0,lon0,h0,spheroid)
[___] = ned2ecef(___,angleUnit)
```

Description

[X,Y,Z] = ned2ecef(xNorth,yEast,zDown,lat0,lon0,h0,spheroid) transforms the local north-east-down (NED) Cartesian coordinates specified by xNorth, yEast, and zDown to the geocentric Earth-centered Earth-fixed (ECEF) Cartesian coordinates specified by X, Y, and Z. Specify the origin of the local NED system with the geodetic coordinates lat0, lon0, and h0. Each coordinate input argument must match the others in size or be scalar. Specify spheroid as the reference spheroid for the geodetic coordinates.

[___] = ned2ecef(___,angleUnit) specifies the units for latitude and longitude. Specify angleUnit as 'degrees' (the default) or 'radians'.

Examples

Calculate ECEF Coordinates from NED Coordinates

Find the ECEF coordinates of Mount Mansfield with respect to a nearby aircraft, using the NED coordinates of Mount Mansfield with respect to the geodetic coordinates of the aircraft.

First, specify the reference ellipsoid as WGS84 with length units measured in kilometers. For more information about WGS84, see "Reference Spheroids". The units for ellipsoidal height, NED coordinates, and ECEF coordinates must match the units specified by the LengthUnit property of the reference spheroid.

```
wgs84 = wgs84Ellipsoid('kilometer');
```

Specify the geodetic coordinates of the local origin. In this example, the local origin is the aircraft. Specify h0 as ellipsoidal height in kilometers.

```
lat0 = 44.532;
lon0 = -72.782;
h0 = 1.699;
```

Specify the NED coordinates of the point of interest. In this example, the point of interest is Mount Mansfield.

```
xNorth = 1.3343;
yEast = -2.5444;
zDown = 0.3600;
```

Then, calculate the ECEF coordinates of Mount Mansfield. In this example, the results display in scientific notation.

```
[x,y,z] = ned2ecef(xNorth,yEast,zDown,lat0,lon0,h0,wgs84)

x = 1.3457e+03

y = -4.3509e+03

z = 4.4523e+03
```

Reverse the transformation using the `ecef2ned` function.

```
[xNorth,yEast,zDown] = ecef2ned(x,y,z,lat0,lon0,h0,wgs84)

xNorth = 1.3343

yEast = -2.5444

zDown = 0.3600
```

Input Arguments

xNorth — NED *x*-coordinates
scalar | vector | matrix | N-D array

NED *x*-coordinates of one or more points in the local NED system, specified as a scalar, vector, matrix, or N-D array. Specify values in units that match the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Data Types: `single` | `double`

yEast — NED *y*-coordinates
scalar | vector | matrix | N-D array

NED *y*-coordinates of one or more points in the local NED system, specified as a scalar, vector, matrix, or N-D array. Specify values in units that match the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Data Types: `single` | `double`

zDown — NED *z*-coordinates
scalar | vector | matrix | N-D array

NED *z*-coordinates of one or more points in the local NED system, specified as a scalar, vector, matrix, or N-D array. Specify values in units that match the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Data Types: `single` | `double`

lat0 — Geodetic latitude of local origin
scalar | vector | matrix | N-D array

Geodetic latitude of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

`lon0` — Geodetic longitude of local origin
scalar | vector | matrix | N-D array

Geodetic longitude of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

`h0` — Ellipsoidal height of local origin
scalar | vector | matrix | N-D array

Ellipsoidal height of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify values in units that match the `LengthUnit` property of the `spheroid` object. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

For more information about ellipsoidal height, see "Find Ellipsoidal Height from Orthometric Height".

Data Types: `single` | `double`

`spheroid` — Reference spheroid
`referenceEllipsoid` object | `oblateSpheroid` object | `referenceSphere` object

Reference spheroid, specified as a `referenceEllipsoid` object, `oblateSpheroid` object, or `referenceSphere` object. The term reference spheroid is used synonymously with reference ellipsoid. To create a reference spheroid, use the creation function for the object. To specify the reference ellipsoid for WGS84, use the `wgs84Ellipsoid` function.

For more information about reference spheroids, see "Reference Spheroids".

Example: `spheroid = referenceEllipsoid('GRS 80');`

`angleUnit` — Angle units
`'degrees'` (default) | `'radians'`

Angle units, specified as `'degrees'` (the default) or `'radians'`.

Output Arguments

`X` — ECEF *x*-coordinates
scalar | vector | matrix | N-D array

ECEF *x*-coordinates of one or more points in the geocentric ECEF system, returned as a scalar, vector, matrix, or N-D array. Units are specified by the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

`Y` — ECEF *y*-coordinates
scalar | vector | matrix | N-D array

ECEF *y*-coordinates of one or more points in the geocentric ECEF system, returned as a scalar, vector, matrix, or N-D array. Units are specified by the `LengthUnit` property of the `spheroid`

argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Z — ECEF *z*-coordinates
scalar | vector | matrix | N-D array

ECEF *z*-coordinates of one or more points in the geocentric ECEF system, returned as a scalar, vector, matrix, or N-D array. Units are specified by the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Tips

To transform vectors instead of coordinate locations, use the `ned2ecefv` function.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also
aer2ecef | ned2geodetic | ecef2ned | enu2ecef

Topics
"Choose a 3-D Coordinate System"

Introduced in R2012b

ned2ecefv

Rotate local north-east-down vector to geocentric Earth-centered Earth-fixed

Syntax

```
[U,V,W] = ned2ecefv(uNorth,vEast,wDown,lat0,lon0)
[ ___ ] = ned2ecefv( ___ ,angleUnit)
```

Description

`[U,V,W] = ned2ecefv(uNorth,vEast,wDown,lat0,lon0)` returns vector components U, V, and W in a geocentric Earth-centered Earth-fixed (ECEF) system corresponding to vector components `uNorth`, `vEast`, and `wDown` in a local north-east-down (NED) system. Specify the origin of the system with the geodetic coordinates `lat0` and `lon0`. Each coordinate input argument must match the others in size or be scalar.

`[ ___ ] = ned2ecefv( ___ ,angleUnit)` specifies the units for latitude and longitude. Specify `angleUnit` as `'degrees'` (the default) or `'radians'`.

Examples

Calculate ECEF Vector Components from NED Components

Find the ECEF velocity components of an aircraft using its NED velocity components.

Specify the geodetic coordinates of the aircraft in degrees and the NED velocity components in kilometers per hour.

```
lat0 = 61.64;
lon0 = 30.70;

uNorth = -434.0403;
vEast = 152.4451;
wDown = -684.6964;
```

Calculate the ECEF components of the aircraft. The units for the ECEF components match the units for the NED components. Thus, the ECEF components are returned in kilometers per hour. The rotation performed by `ned2ecefv` does not affect the speed of the aircraft.

```
[U,V,W] = ned2ecefv(uNorth,vEast,wDown,lat0,lon0)
```

```
U = 530.2445
```

```
V = 492.1283
```

```
W = 396.3459
```

Reverse the rotation using the `ecef2nedv` function.

```
[uNorth,vEast,wDown] = ecef2nedv(U,V,W,lat0,lon0)
```

```
uNorth = -434.0403

vEast = 152.4451

wDown = -684.6964
```

Input Arguments

uNorth — NED *x*-components
scalar value | vector | matrix | N-D array

NED *x*-components of one or more vectors in the local NED system, specified as a scalar value, vector, matrix, or N-D array.

Data Types: `single` | `double`

vEast — NED *y*-components
scalar value | vector | matrix | N-D array

NED *y*-components of one or more vectors in the local NED system, specified as a scalar value, vector, matrix, or N-D array.

Data Types: `single` | `double`

wDown — NED *z*-components
scalar value | vector | matrix | N-D array

NED *z*-components of one or more vectors in the local NED system, specified as a scalar value, vector, matrix, or N-D array.

Data Types: `single` | `double`

lat0 — Geodetic latitude of local origin
scalar | vector | matrix | N-D array

Geodetic latitude of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

lon0 — Geodetic longitude of local origin
scalar | vector | matrix | N-D array

Geodetic longitude of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

angleUnit — Angle units
`'degrees'` (default) | `'radians'`

Angle units, specified as `'degrees'` (the default) or `'radians'`.

Output Arguments

U — ECEF *x*-components
scalar value | vector | matrix | N-D array

ECEF *x*-components of one or more vectors, returned as a scalar value, vector, matrix, or N-D array. Values are returned in the units specified by `uNorth`, `vEast`, and `wDown`.

V — ECEF *y*-components
scalar value | vector | matrix | N-D array

ECEF *y*-components of one or more vectors, returned as a scalar value, vector, matrix, or N-D array. Values are returned in the units specified by `uNorth`, `vEast`, and `wDown`.

W — ECEF *z*-components
scalar value | vector | matrix | N-D array

ECEF *z*-components of one or more vectors, returned as a scalar value, vector, matrix, or N-D array. Values are returned in the units specified by `uNorth`, `vEast`, and `wDown`.

Tips

To transform coordinate locations instead of vectors, use the `ned2ecef` function.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also
`enu2ecefv` | `ecef2nedv` | `ecef2enuv`

Topics
"Vectors in 3-D Coordinate Systems"

Introduced in R2012b

ned2geodetic

Transform local north-east-down coordinates to geodetic

Syntax

```
[lat,lon,h] = ned2geodetic(xNorth,yEast,zDown,lat0,lon0,h0,spheroid)
[ ___ ] = ned2geodetic( ___ ,angleUnit)
```

Description

`[lat,lon,h] = ned2geodetic(xNorth,yEast,zDown,lat0,lon0,h0,spheroid)` transforms the local north-east-down (NED) Cartesian coordinates specified by `xNorth`, `yEast`, and `zDown` to the geodetic coordinates specified by `lat`, `lon`, and `h`. Specify the origin of the local NED system with the geodetic coordinates `lat0`, `lon0`, and `h0`. Each coordinate input argument must match the others in size or be scalar. Specify `spheroid` as the reference spheroid for the geodetic coordinates.

`[ ___ ] = ned2geodetic( ___ ,angleUnit)` specifies the units for latitude and longitude. Specify `angleUnit` as `'degrees'` (the default) or `'radians'`.

Examples

Calculate Geodetic Coordinates from NED Coordinates

Find the geodetic coordinates of Mount Mansfield with respect to a nearby aircraft, using the NED coordinates of Mount Mansfield with respect to the geodetic coordinates of the aircraft.

First, specify the reference spheroid as WGS84. For more information about WGS84, see "Reference Spheroids". The units for ellipsoidal height and NED coordinates must match the units specified by the `LengthUnit` property of the reference spheroid. The default length unit for the reference spheroid created by `wgs84Ellipsoid` is `'meter'`.

```
wgs84 = wgs84Ellipsoid;
```

Specify the geodetic coordinates of the local origin. In this example, the local origin is the aircraft. Specify `h0` as ellipsoidal height in meters.

```
lat0 = 44.532;
lon0 = -72.782;
h0 = 1699;
```

Specify the NED coordinates of the point of interest. In this example, the point of interest is Mount Mansfield.

```
xNorth = 1334.3;
yEast = -2543.6;
zDown = 359.65;
```

Then, calculate the geodetic coordinates of Mount Mansfield. The result `h` is the ellipsoidal height of the mountain in meters. To view the results in standard notation, specify the display format as `shortG`.

```
format shortG
[lat,lon,h] = ned2geodetic(xNorth,yEast,zDown,lat0,lon0,h0,wgs84)

lat =
        44.544


lon =
       -72.814


h =
         1340
```

Reverse the transformation using the `geodetic2ned` function.

`[xNorth,yEast,zDown] = geodetic2ned(lat,lon,h,lat0,lon0,h0,wgs84)`

```
xNorth =
        1334.3


yEast =
       -2543.6


zDown =
        359.65
```

Input Arguments

xNorth — NED *x*-coordinates
scalar | vector | matrix | N-D array

NED *x*-coordinates of one or more points in the local NED system, specified as a scalar, vector, matrix, or N-D array. Specify values in units that match the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Data Types: `single` | `double`

yEast — NED *y*-coordinates
scalar | vector | matrix | N-D array

NED *y*-coordinates of one or more points in the local NED system, specified as a scalar, vector, matrix, or N-D array. Specify values in units that match the `LengthUnit` property of the `spheroid` argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Data Types: `single` | `double`

zDown — NED *z*-coordinates
scalar | vector | matrix | N-D array

NED *z*-coordinates of one or more points in the local NED system, specified as a scalar, vector, matrix, or N-D array. Specify values in units that match the `LengthUnit` property of the `spheroid`

argument. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

Data Types: `single` | `double`

`lat0` — Geodetic latitude of local origin
scalar | vector | matrix | N-D array

Geodetic latitude of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

`lon0` — Geodetic longitude of local origin
scalar | vector | matrix | N-D array

Geodetic longitude of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify the values in degrees. To use values in radians, specify the `angleUnit` argument as `'radians'`.

Data Types: `single` | `double`

`h0` — Ellipsoidal height of local origin
scalar | vector | matrix | N-D array

Ellipsoidal height of the local origin, specified as a scalar, vector, matrix, or N-D array. The local origin can refer to one point or a series of points (for example, a moving platform). Specify values in units that match the `LengthUnit` property of the `spheroid` object. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

For more information about ellipsoidal height, see "Find Ellipsoidal Height from Orthometric Height".

Data Types: `single` | `double`

`spheroid` — Reference spheroid
`referenceEllipsoid` object | `oblateSpheroid` object | `referenceSphere` object

Reference spheroid, specified as a `referenceEllipsoid` object, `oblateSpheroid` object, or `referenceSphere` object. The term reference spheroid is used synonymously with reference ellipsoid. To create a reference spheroid, use the creation function for the object. To specify the reference ellipsoid for WGS84, use the `wgs84Ellipsoid` function.

For more information about reference spheroids, see "Reference Spheroids".

Example: `spheroid = referenceEllipsoid('GRS 80');`

`angleUnit` — Angle units
`'degrees'` (default) | `'radians'`

Angle units, specified as `'degrees'` (the default) or `'radians'`.

Output Arguments

`lat` — Geodetic latitude
scalar | vector | matrix | N-D array

Geodetic latitude of one or more points, returned as a scalar, vector, matrix, or N-D array. Values are specified in degrees within the interval `[-90 90]`. To use values in radians, specify the `angleUnit` argument as `'radians'`.

lon — Geodetic longitude
scalar | vector | matrix | N-D array

Geodetic longitude of one or more points, returned as a scalar, vector, matrix, or N-D array. Values are specified in degrees within the interval `[-180 180]`. To use values in radians, specify the `angleUnit` argument as `'radians'`.

h — Ellipsoidal height
scalar | vector | matrix | N-D array

Ellipsoidal height of one or more points, returned as a scalar, vector, matrix, or N-D array. Values are specified in units that match the `LengthUnit` property of the `spheroid` object. For example, the default length unit for the reference ellipsoid created by `wgs84Ellipsoid` is `'meter'`.

For more information about ellipsoidal height, see "Find Ellipsoidal Height from Orthometric Height".

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also
`aer2geodetic` | `enu2geodetic` | `geodetic2ned` | `ned2ecef`

Topics
"Choose a 3-D Coordinate System"

Introduced in R2012b

neworig

Orient regular data grid to oblique aspect

Syntax

```
[Z,lat,lon] = neworig(Z0,R,origin)
[Z,lat,lon] = neworig(Z0,R,origin,'forward')
[Z,lat,lon] = neworig(Z0,R,origin,'inverse')
```

Description

`[Z,lat,lon] = neworig(Z0,R,origin)` and `[Z,lat,lon] = neworig(Z0,R,origin,'forward')` will transform regular data grid `Z0` into an oblique aspect, while preserving the matrix storage format. In other words, the oblique map origin is not necessarily at (0,0) in the Greenwich coordinate frame. This allows operations to be performed on the matrix representing the oblique map. For example, azimuthal calculations for a point in a data grid become row and column operations if the data grid is transformed so that the north pole of the oblique map represents the desired point on the globe.

`R` can be a geographic raster reference object, a referencing vector, or a referencing matrix. If `R` is a geographic raster reference object, its `RasterSize` property must be consistent with `size(Z)`.

If `R` is a referencing vector, it must be a 1-by-3 with elements:

```
[cells/degree northern_latitude_limit western_longitude_limit]
```

If `R` is a referencing matrix, it must be 3-by-2 and transform raster row and column indices to or from geographic coordinates according to:

```
[lon lat] = [row col 1] * R
```

If `R` is a referencing matrix, it must define a (non-rotational, non-skewed) relationship in which each column of the data grid falls along a meridian and each row falls along a parallel. Nearest-neighbor interpolation is used by default. NaN is returned for points outside the grid limits or for which `lat` or `lon` contain NaN. All angles are in units of degrees.

`[Z,lat,lon] = neworig(Z0,R,origin,'inverse')` transforms the regular data grid from the oblique frame to the Greenwich coordinate frame.

The `neworig` function transforms a regular data grid into a new matrix in an altered coordinate system. An analytical use of the new matrix can be realized in conjunction with the `newpole` function. If a selected point is made the *north pole* of the new system, then when a new matrix is created with `neworig`, each row of the new matrix is a constant distance from the selected point, and each column is a constant azimuth from that point.

Limitations

`neworig` only supports data grids that cover the entire globe.

Examples

Load elevation raster data and a geographic cells reference object. Then, transform the map so that Sri Lanka is at the North Pole.

```
load topo60c
origin = newpole(7,80);
Z = neworig(topo60c,topo60cR,origin);
axesm miller
gridm on
lat = linspace(-90,90,90);
lon = linspace(-180,180,180);
surfm(lat,lon,Z)
demcmap(topo60c)
tightmap
```
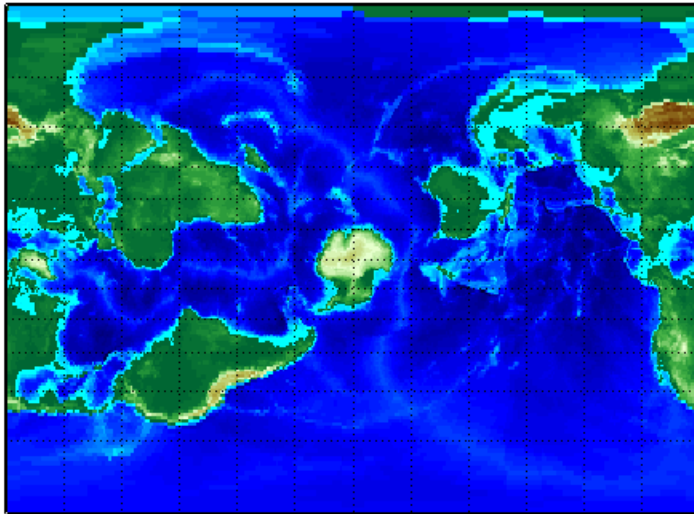

See Also

org2pol | rotatem | geographicToDiscrete

Introduced before R2006a

newpole

Origin vector to place specific point at pole

Syntax

```
origin = newpole(polelat,polelon)
origin = newpole(polelat,polelon,units)
```

Description

`origin = newpole(polelat,polelon)` provides the `origin` vector for a transformed coordinate system based upon moving the point (`polelat`, `polelon`) to become the north pole singularity in the new system. The origin is a three-element vector of the form `[latitude longitude orientation]`, where the latitude and longitude are the coordinates the new center (`origin`) had in the untransformed system, and the orientation is the azimuth of the true North Pole from the new origin point. For the `newpole` calculation, this orientation is constrained to be always 0º.

`origin = newpole(polelat,polelon,units)` specifies the units of the inputs and output, where `units` is any valid angle unit. The default is `'degrees'`.

When developing transverse or oblique projections, you need transformed coordinate systems. One way to define these systems is to establish the point in the original (untransformed) system that will become the new (transformed) *north pole*.

Examples

Take a point and make it the new North Pole:

```
origin = newpole(60,180)

origin =
   30.0000         0         0
```

This makes sense: as a point 30º beyond the true North Pole on the original origin's meridian is pulled up to become the *pole*, the point originally 30º above the origin is pulled down into the origin spot.

See Also
neworig | org2pol | putpole

nm2deg

Convert spherical distance from nautical miles to degrees

Syntax

```
deg = nm2deg(nm)
deg = nm2deg(nm,radius)
deg = nm2deg(nm,sphere)
```

Description

`deg = nm2deg(nm)` converts distances from nautical miles to degrees, as measured along a great circle on a sphere with a radius of 3440.065 nm, the mean radius of the Earth.

`deg = nm2deg(nm,radius)` converts distances from nautical miles to degrees, as measured along a great circle on a sphere having the specified radius.

`deg = nm2deg(nm,sphere)` converts distances from nautical miles to degrees, as measured along a great circle on a sphere approximating an object in the Solar System.

Examples

Convert Nautical Miles to Degrees on Earth

Convert 500 nautical miles to degrees, as measured along a great circle on the Earth. By default, the `nm2deg` function assumes your input is a measurement of the Earth, which is represented as a sphere with a radius of 3440.065 nautical miles.

```
deg = nm2deg(500)
```

```
deg = 8.3277
```

Convert Nautical Miles to Degrees on Sphere with Specified Radius

Convert 500 nautical miles to degrees, as measured along a great circle on a sphere with a radius of 3432 nautical miles.

```
deg = nm2deg(500,3432)
```

```
deg = 8.3473
```

Input Arguments

nm — Distance in nautical miles
numeric array

Distance in nautical miles, specified as a numeric array.

Data Types: `single` | `double`

radius — Radius
3440.065 (default) | numeric scalar

Radius of sphere in units of nautical miles, specified as a numeric scalar.

sphere — Sphere
`'sun'` | `'moon'` | `'mercury'` | `'venus'` | `'earth'` | ...

Sphere approximating an object in the Solar System, specified as one of the following values: `'sun'`, `'moon'`, `'mercury'`, `'venus'`, `'earth'`, `'mars'`, `'jupiter'`, `'saturn'`, `'uranus'`, `'neptune'`, or `'pluto'`. The value of `sphere` is case-insensitive.

Output Arguments

deg — Distance in degrees
numeric array

Distance in degrees, returned as a numeric array.

Data Types: `single` | `double`

See Also

deg2rad | rad2deg | deg2nm | km2deg | sm2deg | nm2rad

Introduced in R2007a

nm2km

Convert nautical miles to kilometers

Syntax

km = nm2km(nm)

Description

km = nm2km(nm) converts distances from nautical miles to kilometers.

Examples

How fast is 30 knots (nautical miles per hour) in kph?

km = nm2km(30)

km =
    55.5600

See Also

deg2km | km2deg | km2rad | rad2km | deg2nm | nm2deg | nm2rad | rad2nm | deg2sm | sm2deg | deg2sm | sm2rad | rad2sm

Introduced in R2007a

nm2rad

Convert spherical distance from nautical miles to radians

Syntax

```
rad = nm2rad(nm)
rad = nm2rad(nm,radius)
rad = nm2rad(nm,sphere)
```

Description

`rad = nm2rad(nm)` converts distances from nautical miles to radians, as measured along a great circle on a sphere with a radius of 3440.065 nm, the mean radius of the Earth.

`rad = nm2rad(nm,radius)` converts distances from nautical miles to radians, as measured along a great circle on a sphere having the specified radius.

`rad = nm2rad(nm,sphere)` converts distances from nautical miles to radians, as measured along a great circle on a sphere approximating an object in the Solar System.

Examples

Convert Nautical Miles to Radians on Earth

Convert 500 nautical miles to radians, as measured along a great circle on the Earth. By default, the `nm2rad` function assumes your input is a measurement of the Earth, which is represented as a sphere with a radius of 3440.065 nautical miles.

```
rad = nm2rad(500)
```

```
rad = 0.1453
```

Convert Nautical Miles to Radians on Sphere with Specified Radius

Convert 500 nautical miles to radians, as measured along a great circle on a sphere with a radius of 3432 nautical miles.

```
rad = nm2rad(500,3432)
```

```
rad = 0.1457
```

Input Arguments

nm — Distance in nautical miles
numeric array

Distance in nautical miles, specified as a numeric array.

Data Types: `single` | `double`

radius — Radius
3440.065 (default) | numeric scalar

Radius of sphere in units of nautical miles, specified as a numeric scalar.

sphere — Sphere
`'sun'` | `'moon'` | `'mercury'` | `'venus'` | `'earth'` | ...

Sphere approximating an object in the Solar System, specified as one of the following values: `'sun'`, `'moon'`, `'mercury'`, `'venus'`, `'earth'`, `'mars'`, `'jupiter'`, `'saturn'`, `'uranus'`, `'neptune'`, or `'pluto'`. The value of `sphere` is case-insensitive.

Output Arguments

rad — Distance in radians
numeric array

Distance in radians, returned as a numeric array.

Data Types: `single` | `double`

See Also
deg2rad | rad2deg | rad2nm | nm2deg | sm2rad | km2rad

Introduced in R2007a

nm2sm

Convert nautical to statute miles

Syntax

```
sm = nm2sm(nm)
```

Description

`sm = nm2sm(nm)` converts distances from nautical miles to statute miles.

See Also

deg2km | km2deg | km2rad | rad2km | deg2nm | nm2deg | nm2rad | rad2nm | deg2sm | sm2deg | deg2sm | sm2rad | rad2sm

Introduced in R2007a

northarrow

Add graphic element pointing to geographic North Pole

Syntax

```
northarrow
northarrow('property',value,...)
```

Description

`northarrow` creates a default north arrow.

`northarrow('property',value,...)` creates a north arrow using the specified property/value pairs. Valid entries for properties are `'latitude'`, `'longitude'`, `'facecolor'`, `'edgecolor'`, `'linewidth'`, and `'scaleratio'`. The `'latitude'` and `'longitude'` properties specify the location of the north arrow. The `'facecolor'`, `'edgecolor'`, and `'linewidth'` properties control the appearance of the north arrow. The `'scaleratio'` property represents the size of the north arrow as a fraction of the size of the axes. A `'scaleratio'` value of 0.10 creates a north arrow one-tenth (1/10) the size of the axes. You can change the appearance (`'facecolor'`, `'edgecolor'`, and `'linewidth'`) of the north arrow using the `set` command.

The `northarrow` function creates a north arrow symbol at the map origin on the displayed map. You can reposition the north arrow symbol by clicking and dragging its icon. Alternatively, open a dialog box that you can use to change the location of the north arrow by right-clicking or **Ctrl**-clicking the arrow. The orientation of the north arrow is computed for you, and does not need manual adjustment, no matter where you move the symbol.

Modifying some of the properties of the north arrow results in replacement of the original object. Use `HANDLEM('NorthArrow')` to get the handles associated with the north arrow.

Examples

Add North Arrow to Map of Texas

Create a map of Texas and add the north arrow in the lower left of the map.

```
figure; usamap('texas')
states = shaperead('usastatelo.shp','UseGeoCoords',true);
faceColors = makesymbolspec('Polygon',...
        {'INDEX', [1 numel(states)], 'FaceColor', ...
        polcmap(numel(states))});
geoshow(states, 'DisplayType', 'polygon', ...
        'SymbolSpec', faceColors)
northarrow('latitude',25,'longitude',-105,'linewidth',1.5);
```
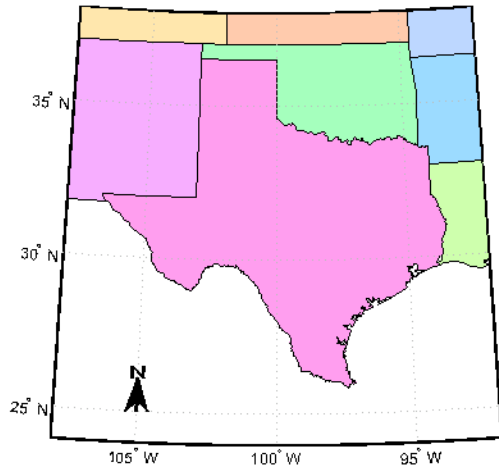
Change the `'FaceColor'` and `'EdgeColor'` properties of the north arrow.

```
h = handlem('NorthArrow');
set(h,'FaceColor',[1.000 0.8431 0.0000],...
      'EdgeColor',[0.0100 0.0100 0.9000])
```
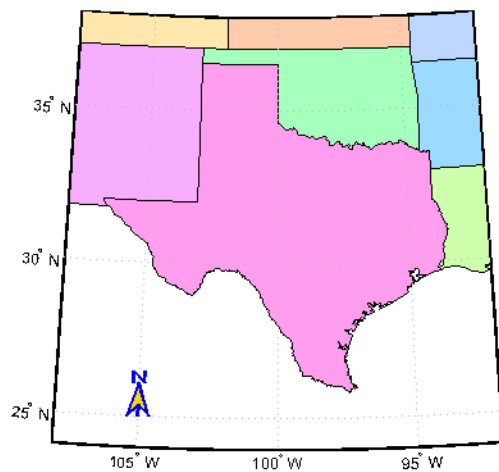

Add North Arrows to Map of South Pole

Create a map centered at the South Pole. Add a north arrow at a specified geographic position.

```
antarctica = shaperead('landareas.shp', 'UseGeoCoords', true, ...
    'Selector',{@(name) strcmpi(name,{'Antarctica'}), 'Name'});
figure
worldmap('south pole')
geoshow(antarctica)
northarrow('latitude',-57,'longitude',135);
```
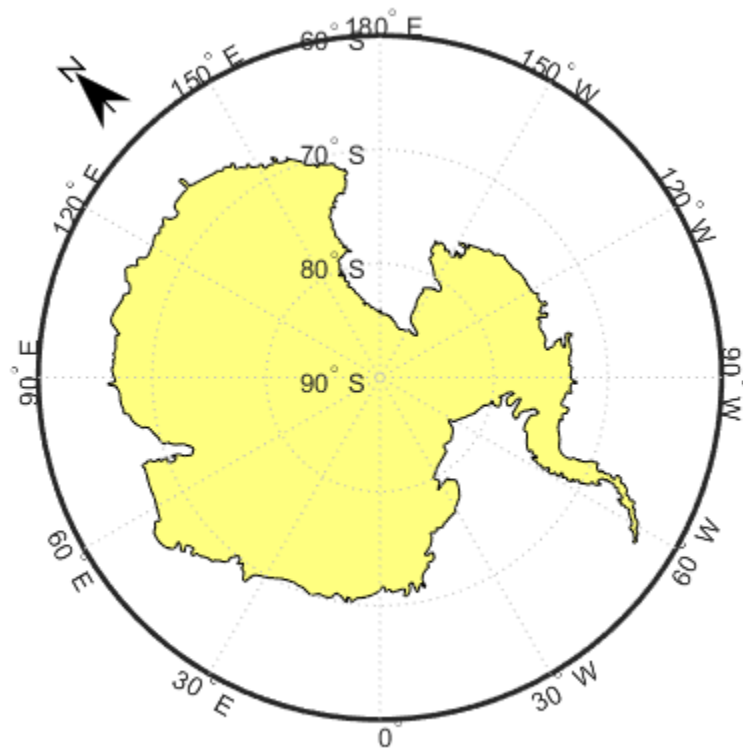
Change the appearance of the north arrow.

```
h = handlem('NorthArrow');
set(h,'FaceColor', [1.000 0.8431 0.0000], ...
      'EdgeColor', [0.0100 0.0100 0.9000])
```
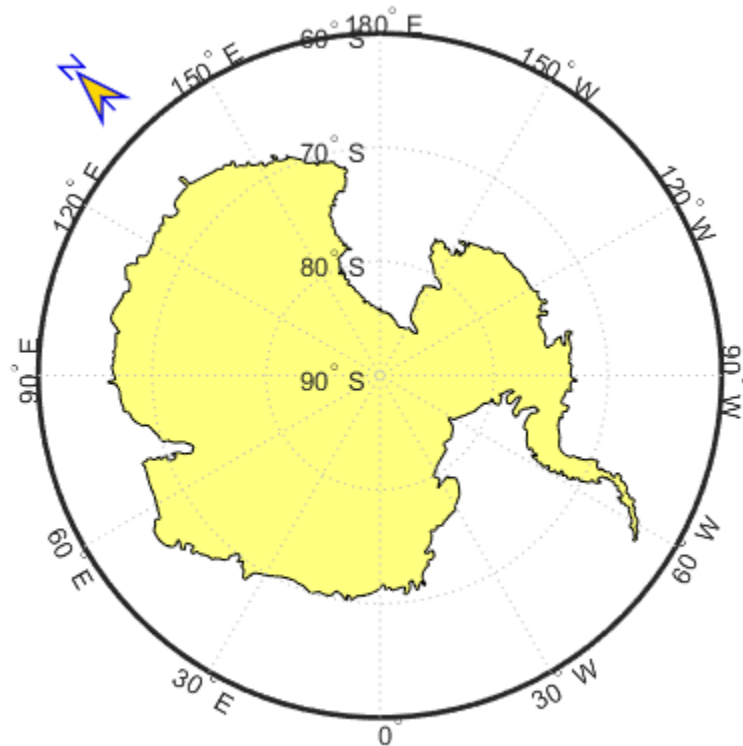
Show that every direction from the South Pole is north by creating three more north arrows. Note that north arrows are objects in the axes with Cartesian coordinates, not mapping objects. As a result, if you create more than one north arrow, any Mapping Toolbox™ function that manipulates a north arrow will affect only the last one drawn.

```
northarrow('latitude',-57,'longitude', 45);
northarrow('latitude',-57,'longitude',225);
northarrow('latitude',-57,'longitude',315);
```
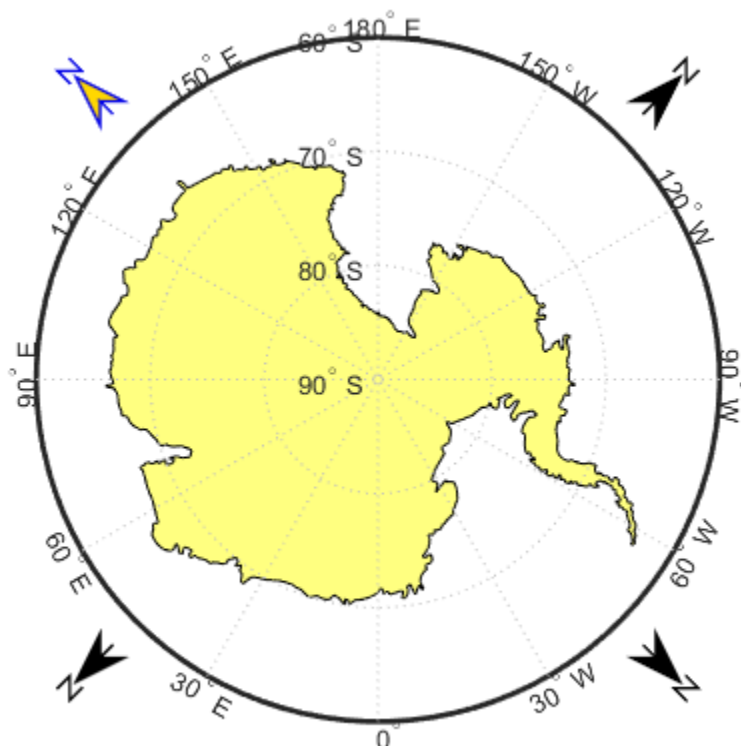
Limitations

You can draw multiple north arrows on the map. However, the callbacks will only work with the most recently created north arrow. In addition, since it can be displayed outside the map frame limits, the north arrow is not converted into a "mapped" object. Hence, the location and orientation of the north arrow have to be updated manually if the map origin or projection changes.

See Also
scaleruler

Introduced before R2006a

npi2pi

Wrap longitudes to [-180 180] degree interval

Compatibility

Note The npi2pi function has been replaced by wrapTo180 and wrapToPi.

Syntax

```
anglout = npi2pi(anglin)
anglout = npi2pi(anglin,units)
anglout = npi2pi(anglin,units,method)
```

Description

anglout = npi2pi(anglin) wraps the input angle anglin (typically representing a longitude) to lie on the range -180 to 180 (e.g., 270º is renamed -90º).

anglout = npi2pi(anglin,*units*) specifies the angle units with any valid angle units character vector *units*. The default is 'degrees'.

anglout = npi2pi(anglin,*units*,*method*) allows special alternative computations to be used when npi2pi is called from within certain Mapping Toolbox functions. *method* can be one of the following:

- 'exact', for exact wrapping (the default value)
- 'inward', where angles are scaled by a factor of (1 - epsm('radians')) before wrapping
- 'outward', where angles are scaled by a factor of (1 + epsm('radians')) before wrapping

Examples

```
npi2pi(315)

ans =
    -45

npi2pi(181)

ans =
  -179
```

See Also
wrapToPi | wrapTo180

Introduced before R2006a

oblateSpheroid

Oblate ellipsoid of revolution

Description

An `oblateSpheroid` object encapsulates the interrelated intrinsic properties of an oblate ellipsoid of revolution. An oblate spheroid is symmetric about its polar axis and flattened at the poles, and includes the perfect sphere as a special case.

Creation

You can create an `oblateSpheroid` object, `s`, by entering:

`s = oblateSpheroid;`

on the command line.

Properties

SemimajorAxis — Equatorial radius of spheroid
1 (default) | positive, finite scalar

Equatorial radius of spheroid, specified as a positive, finite scalar. The value of `SemimajorAxis` is expressed in meters.

When the `SemimajorAxis` property is changed, the `SeminimorAxis` property scales as needed to preserve the shape of the spheroid and the values of shape-related properties including `InverseFlattening` and `Eccentricity`. The only way to change the `SemimajorAxis` property is to set it directly, using dot notation.

Example: 6378137

Data Types: `double`

SeminimorAxis — Distance from center of spheroid to pole
1 (default) | nonnegative, finite scalar

Distance from center of spheroid to pole, specified as a nonnegative, finite scalar. The value of `SeminimorAxis` is always less than or equal to `SemimajorAxis`, and is expressed in meters.

When the `SeminimorAxis` property is changed, the `SemimajorAxis` property remains unchanged, but the shape of the spheroid changes, which is reflected in changes in the values of `InverseFlattening`, `Eccentricity`, and other shape-related properties.

Example: 6356752

Data Types: `double`

InverseFlattening — Reciprocal of flattening
`Inf` (default) | positive scalar in the range [1, `Inf`]

Reciprocal of flattening, specified as positive scalar in the range [1, `Inf`].

The value of inverse flattening, 1/*f*, is calculated using `SemimajorAxis` value *a* and `SemiminorAxis` value *b* according to 1/*f* = *a*/(*a-b*). A value 1/*f* of `Inf` designates a perfect sphere. As 1/*f* approaches 1, the reference spheroid approaches a flattened disk.

When the `InverseFlattening` property is changed, other shape-related properties update, including `Eccentricity`. The `SemimajorAxis` property remains unchanged, but the value of `SemiminorAxis` adjusts to reflect the new shape.

Example: 300

Data Types: `double`

`Eccentricity` — First eccentricity of spheroid
0 (default) | nonnegative scalar in the range [0, 1]

First eccentricity of the spheroid, specified as nonnegative scalar in the range [0, 1].

The value of eccentricity, *ecc*, is calculated using `SemimajorAxis` value *a* and `SemiminorAxis` value *b* according to *ecc* = `sqrt`(a^2 - b^2)/*a*. A value *ecc* of 0 designates a perfect sphere.

When the `Eccentricity` property is changed, other shape-related properties update, including `InverseFlattening`. The `SemimajorAxis` property remains unchanged, but the value of `SemiminorAxis` adjusts to reflect the new shape.

Example: 0.08

Data Types: `double`

`Flattening` — Flattening of spheroid
nonnegative scalar in the range [0, 1]

This property is read-only.

Flattening of the spheroid, specified as nonnegative scalar in the range [0, 1].

The value of flattening, *f*, is calculated using `SemimajorAxis` value *a* and `SemiminorAxis` value *b* according to *f* = (*a-b*)/*a*.

Data Types: `double`

`ThirdFlattening` — Third flattening of spheroid
nonnegative scalar in the range [0, 1]

This property is read-only.

Third flattening of the spheroid, specified as nonnegative scalar in the range [0, 1].

The value of the third flattening, *n*, is calculated using `SemimajorAxis` value *a* and `SemiminorAxis` value *b* according to *n* = (*a-b*)/(*a+b*).

Data Types: `double`

`MeanRadius` — Mean radius of the spheroid
positive, finite scalar

This property is read-only.

Mean radius of the spheroid, specified as positive, finite scalar. The `MeanRadius` property is expressed in meters.

The mean radius of the spheroid, *r,* is calculated using `SemimajorAxis` value *a* and `SemiminorAxis` value *b* according to $r = (2a+b)/3$.

Data Types: `double`

SurfaceArea — Surface area of the spheroid
positive, finite scalar

This property is read-only.

Surface area of the spheroid, specified as positive, finite scalar. The `SurfaceArea` property is expressed in square meters.

Data Types: `double`

Volume — Volume of the spheroid
positive, finite scalar

This property is read-only.

Volume of the spheroid, specified as positive, finite scalar. The `Volume` property is expressed in cubic meters.

Data Types: `double`

Examples

Create GRS 80 Ellipsoid

Create a GRS 80 ellipsoid using the `oblateSpheroid` class.

Start with a unit sphere by default.

```
s = oblateSpheroid

s =

oblateSpheroid with defining properties:

       SemimajorAxis: 1
       SemiminorAxis: 1
   InverseFlattening: Inf
        Eccentricity: 0

  and additional properties:

    Flattening
    ThirdFlattening
    MeanRadius
    SurfaceArea
    Volume
```

Reset the semimajor axis to match the equatorial radius of the GRS 80 reference ellipsoid.

```
s.SemimajorAxis = 6378137

s =

oblateSpheroid with defining properties:

       SemimajorAxis: 6378137
       SemiminorAxis: 6378137
    InverseFlattening: Inf
        Eccentricity: 0

  and additional properties:

    Flattening
    ThirdFlattening
    MeanRadius
    SurfaceArea
    Volume
```

The result is a sphere with radius 6,378,137 meters.

Reset the inverse flattening to the standard value for GRS 80, resulting in an oblate spheroid with a semiminor axis consistent with the value, 6,356,752.3141, tabulated in DMA Technical Memorandum 8358.1, "Datums, Ellipsoids, Grids, and Grid Reference Systems."

```
s.InverseFlattening = 298.257222101

s =

oblateSpheroid with defining properties:

       SemimajorAxis: 6378137
       SemiminorAxis: 6356752.31414036
    InverseFlattening: 298.257222101
        Eccentricity: 0.0818191910428158

  and additional properties:

    Flattening
    ThirdFlattening
    MeanRadius
    SurfaceArea
    Volume
```

Tips

- When you define a spheroid in terms of semimajor and semiminor axes (rather than semimajor axis and inverse flattening, or semimajor axis and eccentricity), a small loss of precision in the last few digits of `Flattening`, `Eccentricity`, and `ThirdFlattening` may occur. This is unavoidable, but does not affect the results of practical computation.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

referenceEllipsoid | referenceSphere | validateLengthUnit

Introduced in R2012a

onem

(To be removed) Construct regular data grid of 1s

Note onem will be removed in a future release. Use the georefcells and ones functions instead. For more information, see "Compatibility Considerations".

Syntax

```
[Z,refvec] = onem(latlim,lonlim,scale)
```

Description

[Z,refvec] = onem(latlim,lonlim,scale) returns a regular data grid consisting entirely of 1s and a three-element referencing vector for the returned data grid, Z. The two-element vectors latlim and lonlim define the latitude and longitude limits of the geographic region. They should be of the form [south north] and [west east], respectively. The scalar scale specifies the number of rows and columns per degree of latitude and longitude.

Examples

```
[Z,refvec] = onem([46,51],[-79,-75],1)

Z =
    1    1    1    1
    1    1    1    1
    1    1    1    1
    1    1    1    1
    1    1    1    1
refvec =
    1    51   -79
```

Compatibility Considerations

onem will be removed
Not recommended starting in R2015b

Some functions that return referencing vectors will be removed, including the onem function. Instead, create a geographic raster reference object using the georefcells function and a matrix of ones using the ones function. Reference objects have several advantages over referencing vectors.

- Unlike referencing vectors, reference objects have properties that document the size of the associated raster, its geographic limits, and the direction of its rows and columns. For examples of reference object properties, see the GeographicPostingsReference object.

- You can manipulate the limits of rasters associated with geographic reference objects using the geocrop function.

- You can manipulate the size and resolution of rasters associated with geographic reference objects using the georesize function.

- Most functions that accept referencing vectors as inputs also accept reference objects.

This table shows how to update your code to use the `georefcells` and `ones` functions instead of the `onem` function.

Will Be Removed	Recommended
`[Z,refvec] = onem(latlim,lonlim,scale);`	`R = georefcells(latlim,lonlim,1/scale,1/scale);` `Z = ones(R.RasterSize);`

See Also
NaN | sparse | zeros | georefcells | ones

Introduced before R2006a

org2pol

Location of north pole in rotated map

Syntax

```
pole = org2pol(origin)
pole = org2pol(origin,units)
```

Description

`pole = org2pol(origin)` returns the location of the North Pole in terms of the coordinate system after transformation based on the input `origin`. The `origin` is a three-element vector of the form `[latitude longitude orientation]`, where `latitude` and `longitude` are the coordinates that the new center (origin) had in the untransformed system, and `orientation` is the azimuth of the true North Pole from the new origin point in the transformed system. The output `pole` is a three-element vector of the form `[latitude longitude meridian]`, which gives the latitude and longitude point in terms of the original untransformed system of the new location of the true North Pole. The meridian is the longitude from the original system upon which the new system is centered.

`pole = org2pol(origin,units)` allows the specification of the angular units of the `origin` vector, where `units` is any valid angle unit. The default is `'degrees'`.

When developing transverse or oblique projections, transformed coordinate systems are required. One way to define these systems is to establish the point at which, in terms of the original (untransformed) system, the (transformed) true North Pole will lie.

Examples

Perhaps you want to make (30ºN,0º) the new origin. Where does the North Pole end up in terms of the original coordinate system?

```
pole = org2pol([30 0 0])

pole =
   60.0000        0        0
```

This makes sense: pull a point 30º down to the origin, and the North Pole is pulled down 30º. A little less obvious example is the following:

```
pole = org2pol([5 40 30])

pole =
   59.6245   80.0750   40.0000
```

See Also
`neworig | putpole`

outlinegeoquad

Polygon outlining geographic quadrangle

Syntax

```
[lat,lon] = outlinegeoquad(latlim,lonlim,dlat,dlon)
```

Description

`[lat,lon] = outlinegeoquad(latlim,lonlim,dlat,dlon)` constructs a polygon that traces the outline of the geographic quadrangle defined by `latlim` and `lonlim`. Such a polygon can be useful for displaying the quadrangle graphically, especially on a projection where the meridians and/or parallels do not project to straight lines. `latlim` is a two-element vector of the form: `[southern-limit northern-limit]` and `lonlim` is a two-element vector of the form: `[western-limit eastern-limit]`. `dlat` is a positive scalar that specifies a minimum vertex spacing in degrees to be applied along the meridians that bound the eastern and western edges of the quadrangle. Likewise, `dlon` is a positive scalar that specifies a minimum vertex spacing in degrees of longitude to be applied along the parallels that bound the northern and southern edges of the quadrangle. The outputs `lat` and `lon` contain the vertices of a simple closed polygon with clockwise vertex ordering.

Examples

Display the outlines of three geographic quadrangles having very different qualities on top of a simple base map:

```
figure('Color','white')
axesm('ortho','Origin',[-45 110],'frame','on','grid','on')
axis off
load coastlines
geoshow(coastlat, coastlon)

% Quadrangle covering Australia and vicinity
[lat, lon] = outlinegeoquad([-45 5],[110 175],5,5);
geoshow(lat,lon,'DisplayType','polygon','FaceAlpha',0.5);

% Quadrangle covering Antarctic region
antarcticCircleLat = dms2degrees([-66 33 39]);
[lat, lon] = outlinegeoquad([-90 antarcticCircleLat], ...
    [-180 180],5,5);
geoshow(lat,lon,'DisplayType','polygon', ...
        'FaceColor','cyan','FaceAlpha',0.5);

% Quadrangle covering nominal time zone 9 hours ahead of UTC
[lat, lon] = outlinegeoquad([-90 90], 135 + [-7.5 7.5], 5, 5);
geoshow(lat,lon,'DisplayType','polygon', ...
        'FaceColor','green','FaceAlpha',0.5);
```
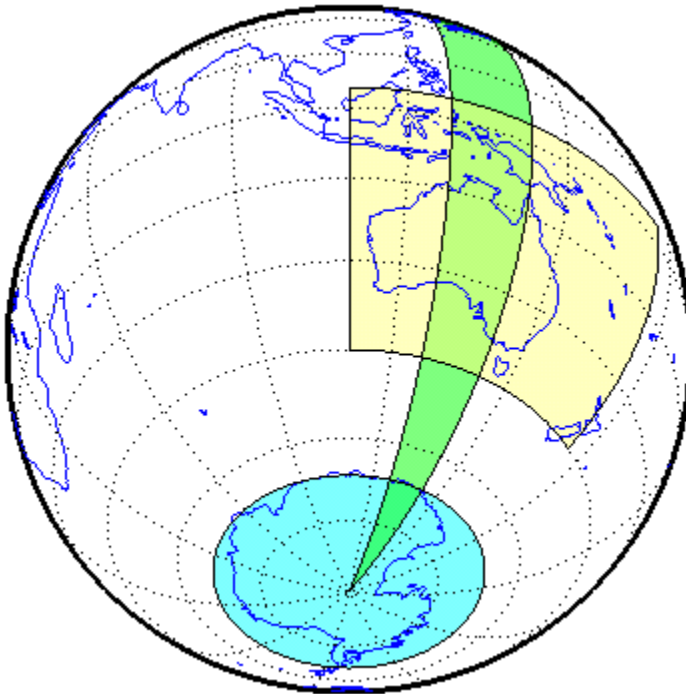
Tips

All input and output angles are in units of degrees. Choose a reasonably small value for `dlat` (a few degrees, perhaps) when using a projection with curved meridians or curved parallels.

To avoid interpolating extra vertices along meridians or parallels, set `dlat` or `dlon` to a value of `Inf`.

Special Cases

The insertion of additional vertices is suppressed at the poles (that is, if `latlim(1) == -90` or `latlim(2) == 90`. If `lonlim` corresponds to a quadrangle width of exactly 360 degrees (`lonlim == [-180 180]`, for example), then it covers a full latitudinal zone and includes two separate, NaN-separated parts, unless either

- `latlim(1) == -90` *or* `latlim(2) == 90`, so that only one part is needed—a polygon that follows a parallel clockwise around one of the poles.
- `latlim(1) == -90` *and* `latlim(2) == 90`, so that the quadrangle encompasses the entire planet. In this case, the quadrangle cannot be represented by a latitude-longitude polygon, and an error results.

See Also

ingeoquad | intersectgeoquad

Introduced in R2008a

paperscale

Set figure properties for printing at specified map scale

Syntax

```
paperscale(paperdist,punits,surfdist,sunits)
paperscale(paperdist,punits,surfdist,sunits,lat,long)
paperscale(paperdist,punits,surfdist,sunits,lat,long,az)
paperscale(paperdist,punits,surfdist,sunits,lat,long,az,gunits)
paperscale(paperdist,punits,surfdist,sunits,lat,long,az,gunits,
radius)
paperscale(scale,...)
[paperXdim,paperYdim] = paperscale(...)
```

Description

paperscale(paperdist,*punits*,surfdist,*sunits*) sets the figure paper position to print the map in the current axes at the desired scale. The scale is described by the geographic distance that corresponds to a paper distance. For example, a scale of 1 inch = 10 kilometers is specified as degrees(1,'inch',10,'km'). See below for an alternate method of specifying the map scale. The surface distance units *sunits* can be any unit recognized by unitsratio. The paper units *punits* can be any dimensional units recognized for the figure PaperUnits property.

paperscale(paperdist,*punits*,surfdist,*sunits*,lat,long) sets the paper position so that the scale is correct at the specified geographic location. If omitted, the default is the center of the map limits.

paperscale(paperdist,*punits*,surfdist,*sunits*,lat,long,az) also specifies the direction along which the scale is correct. If omitted, 90 degrees (east) is assumed.

paperscale(paperdist,*punits*,surfdist,*sunits*,lat,long,az,*gunits*) also specifies the units in which the geographic position and direction are given. If omitted, 'degrees' is assumed.

paperscale(paperdist,*punits*,surfdist,*sunits*,lat,long,az,*gunits*, radius) uses the last input to determine the radius of the sphere. radius can be one of the values supported by km2deg, or it can be the (numerical) radius of the desired sphere in zunits. If omitted, the default radius of the Earth is used..

paperscale(scale,...), where the numeric scale replaces the two property/value pairs, specifies the scale as a ratio between distance on the sphere and on paper. This is commonly notated on maps as 1:scale (e.g. 1:100 000, or 1:1 000 000). For example, paperscale(100000) or paperscale(100000,lat,long).

[paperXdim,paperYdim] = paperscale(...) returns the computed paper dimensions. The dimensions are in the paper units specified. For the scale calling form, the returned dimensions are in centimeters.

Background

Maps are usually printed at a size that allows an easy comparison of distances measured on paper to distances on the Earth. The relationship of geographic distance and paper distance is termed *scale*. It is usually expressed as a ratio, such as 1 to 100,000 or 1:100,000 or 1 cm = 1 km.

Examples

The small circle measures 10 cm across when printed.

```
axesm mercator
[lat,lon] = scircle1(0,0,km2deg(5));
plotm(lat,lon)
[x,y] = paperscale(1,'centimeter',1,'km'); [x y]

ans =
      13.154        12.509

set(gca,'pos', [ 0 0 1 1])
[x,y] = paperscale(1,'centimeter',1,'km'); [x y]

ans =
      10.195        10.195
```

Limitations

The relationship between the paper and geographic coordinates holds only as long as there are no changes to the display that affect the axes limits or the relationship between geographic coordinates and projected coordinates. Changes of this type include the ellipsoid or scale factor properties of the map axes, or adding elements to the display that cause MATLAB to modify the axes autoscaling. To be sure that the scale is correct, execute `paperscale` just before printing.

See Also
printpreview | axesscale | daspectm

Introduced before R2006a

parametricLatitude

Convert geodetic to parametric latitude

Syntax

```
beta = parametricLatitude(phi,F)
beta = parametricLatitude(phi,F,angleUnit)
```

Description

`beta = parametricLatitude(phi,F)` returns the parametric latitude corresponding to geodetic latitude `phi` on an ellipsoid with flattening `F`.

`beta = parametricLatitude(phi,F,angleUnit)` specifies the units of input `phi` and output `beta`.

Examples

Convert Geodetic Latitude to Parametric Latitude

Create a reference ellipsoid and then convert the geodetic latitude to parametric latitude. The reference ellipsoid contains a flattening factor.

```
s = wgs84Ellipsoid;

parametricLatitude(45, s.Flattening)

ans =

    44.9038
```

Convert Geodetic Latitude Expressed in Radians to Parametric Latitude

Create a reference ellipsoid and then convert a parametric latitude expressed in radians to geodetic latitude. The reference ellipsoid contains a flattening factor.

```
s = wgs84Ellipsoid;

parametricLatitude(pi/3, s.Flattening, 'radians')

ans =

    1.0457
```

Input Arguments

`phi` — Geodetic latitude of one or more points
scalar value, vector, matrix, or N-D array

Geodetic latitude of one or more points, specified as a scalar value, vector, matrix, or N-D array. Values must be in units that match the input argument `angleUnit`, if supplied, and in degrees, otherwise.

Data Types: `single` | `double`

F — Flattening of reference spheroid
scalar

Flattening of reference spheroid, specified as a scalar value.

Data Types: `single` | `double`

angleUnit — Unit of measurement for angle
`'degrees'` (default) | `'radians'`

Unit of measurement for angle, specified as `'degrees'` or `'radians'`.

Data Types: `char` | `string`

Output Arguments

beta — Parametric latitudes of one or more points
scalar value, vector, matrix, or N-D array

Parametric latitudes of one or more points, returned as a scalar value, vector, matrix, or N-D array. Values are in units that match the input argument `angleUnit`, if supplied, and in degrees, otherwise.

See Also

Functions
geocentricLatitude | geodeticLatitudeFromParametric

Objects
AuthalicLatitudeConverter | ConformalLatitudeConverter | IsometricLatitudeConverter | RectifyingLatitudeConverter

Introduced in R2013a

patchesm

Project patches on map axes as individual objects

Syntax

```
patchesm(lat,lon,cdata)
patchesm(lat,lon,z,cdata)
patchesm(...,'PropertyName',PropertyValue,...)
h = patchesm(...)
```

Description

`patchesm(lat,lon,cdata)` projects 2-D patch objects onto the current map axes. The input latitude and longitude data must be in the same units as specified in the current map axes. The input `cdata` defines the patch face color. If the input vectors are `NaN` clipped, then multiple patches are drawn each with a single face. Unlike `fillm` and `fill3m`, `patchesm` will always add the patches to the current map regardless of the current hold state.

`patchesm(lat,lon,z,cdata)` projects 3-D planar patches at the uniform elevation given by scalar `z`.

`patchesm(...,'PropertyName',PropertyValue,...)` uses the patch properties supplied to display the patch. Except for `xdata`, `ydata`, and `zdata`, all patch properties available through `patch` are supported by `patchesm`.

`h = patchesm(...)` returns the handles to the patch objects drawn.

Examples

Plot Distinct Patch Objects on Current Map Axes

Plot coastline data as distinct patch objects.

```
load coastlines
axesm sinusoid;
framem
h = patchesm(coastlat,coastlon,'b');
```
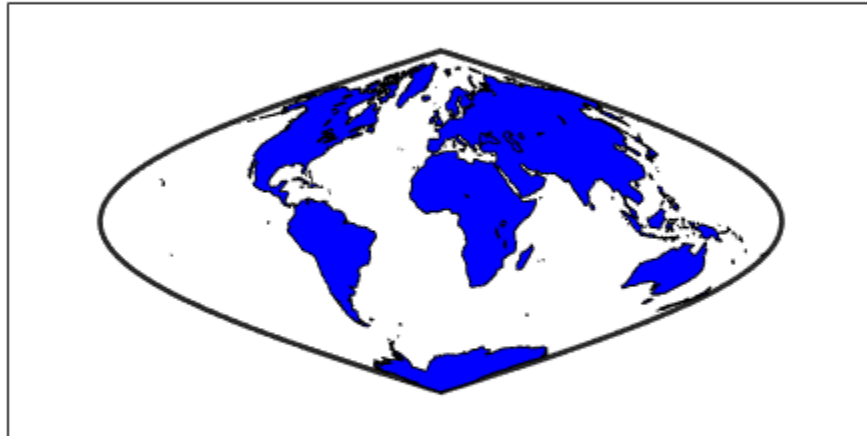
```
% Number of objects created.
length(h)
```

```
ans = 241
```

Tips

Differences between `patchesm` and `patchm`

The `patchesm` function is very similar to the `patchm` function. The significant difference is that in `patchesm`, separate patches (delineated by NaNs in the inputs `lat` and `lon`) are separated and plotted as distinct patch objects on the current map axes. The advantage to this is that less memory is required. The disadvantage is that multifaced objects cannot be treated as a single object. For example, the archipelago of the Philippines cannot be treated and handled as a single Handle Graphics® object.

When Patches Are Completely Trimmed Away

Removing graphic objects that fall outside the map frame is called trimming. If, after trimming no polygons remain to be seen within it, `patchesm` creates no patches and returns an empty 1-by-0 list of handles. When this occurs, automatic reprojection of the patch data (by changing the projection or any of its parameters) is not possible. In cases where some polygons are completely trimmed away but not others, handles returned for the trimmed polygons will be empty. No polygons or rings that have been totally trimmed away can be reprojected; to plot them again, you will need to call `patchesm` again with the original data.

See Also

geoshow | fill3m | fillm | patchm

Introduced before R2006a

patchm

Project patch objects on map axes

Syntax

```
h = patchm(lat,lon,cdata)
h = patchm(lat,lon,cdata,PropertyName,PropertyValue,...)
h = patchm(lat,lon,PropertyName,PropertyValue,...)
h = patchm(lat,lon,z,cdata)
h = patchm(lat,lon,z,cdata, PropertyName,PropertyValue,...)
```

Description

`h = patchm(lat,lon,cdata)` and `h = patchm(lat,lon,cdata,PropertyName,PropertyValue,...)` project and display patch (polygon) objects defined by their vertices given in `lat` and `lon` on the current map axes. `lat` and `lon` must be vectors. The color data, `cdata`, can be any color data designation supported by the standard MATLAB `patch` function. The object handle or handles, h, can be returned.

`h = patchm(lat,lon,PropertyName,PropertyValue,...)` allows any property name/property value pair supported by `patch` to be assigned to the `patchm` object. Except for `XData`, `YData`, and `ZData`, all line properties and styles available through `patch` are supported by `patchm`.

`h = patchm(lat,lon,z,cdata)` and `h = patchm(lat,lon,z,cdata, PropertyName,PropertyValue,...)` allow the assignment of an altitude, z, to each patch object. The default altitude is `z = 0`.

Examples

Project Patch Object on Map Axes

Project coastline data as single patch object on map axes.

```
load coastlines
axesm sinusoid;
framem
h = patchm(coastlat,coastlon,'b');
```
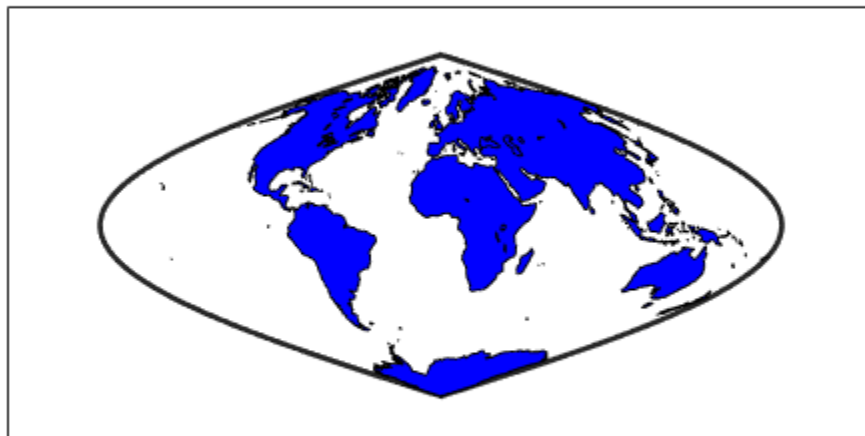
```
length(h)
```

```
ans = 1
```

Tips

How patchm Works

This Mapping Toolbox function is very similar to the standard MATLAB `patch` function. Like its analog, and unlike higher level functions such as `fillm` and `fill3m`, `patchm` adds patch objects to the current map axes regardless of hold state.

When A Patch Is Completely Trimmed Away

Removing graphic objects that fall outside the map frame is called trimming. If, after trimming to the map frame no polygons remain to be seen within it, `patchm` creates no patches and returns an empty 0-by-1 handle. When this occurs, automatic reprojection of the patch data (by changing the projection or any of its parameters) will not be possible. Instead, after changing the projection, call `patchm` again.

See Also
patchesm | fill3m | fillm

Introduced before R2006a

pcolorm

Project regular data grid on map axes in z = 0 plane

Syntax

```
pcolorm(lat,lon,Z)
pcolorm(latlim,lonlim,Z)
pcolorm(...,prop1,val1,prop2,val2,...)
h = pcolorm(...)
```

Description

pcolorm(lat,lon,Z) constructs a surface to represent the data grid Z in the current map axes. The surface lies flat in the horizontal plane with its CData property set to Z. Lat and lon are vectors or 2-D arrays that define the latitude-longitude graticule mesh on which Z is displayed. For a complete description of the various forms that lat and lon can take, see surfm. If the hold state is 'off', pcolorm clears the current map.

pcolorm(latlim,lonlim,Z) defines the graticule using the latitude and longitude limits latlim and lonlim. These limits should match the geographic extent of Z, the data grid. Latlim is a two-element vector of the form:

```
[southern_limit northern_limit]
```

Likewise, lonlim has the form:

```
[western_limit eastern_limit]
```

A latitude-longitude graticule of size 50-by-100 is constructed. The surface FaceColor property is 'texturemap', except when Z is precisely 50-by-100, in which case it is 'flat'.

pcolorm(...,prop1,val1,prop2,val2,...) applies additional MATLAB graphics properties to the surface via property/value pairs. Any property accepted by the surface may be specified, except for XData, YData, and ZData.

h = pcolorm(...) returns a handle to the surface object.

Examples

Load elevation raster data and a geographic cells reference object. Then, display the data as a surface.

```
load topo60c
axesm miller
axis off
framem on
gridm on
[topo60c,topo60cR] = geocrop(topo60c,topo60cR,[-90 90],[-180,180]);
[lat,lon] = geographicGrid(topo60cR);
pcolorm(lat,lon,topo60c)
```

```
demcmap(topo60c)
tightmap
```
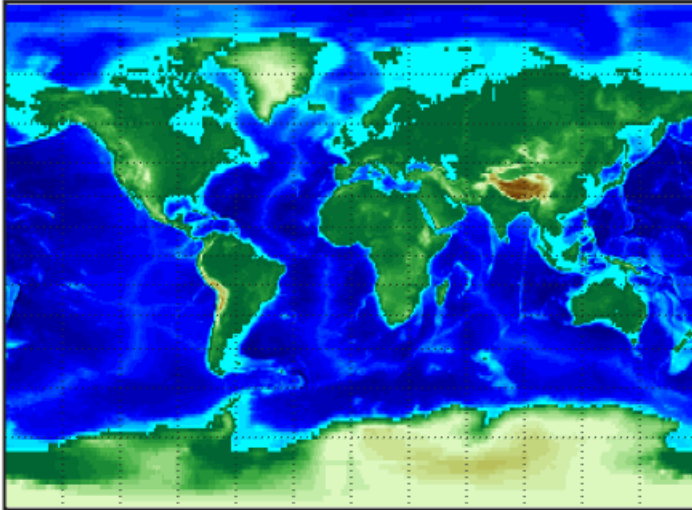

Tips

This function warps a data grid to a graticule mesh, which is projected according to the map axes property `MapProjection`. The fineness, or resolution, of this grid determines the quality of the projection and the speed of plotting it. There is no hard and fast rule for sufficient graticule resolution, but in general, cylindrical projections need fewer graticule points in the longitudinal direction than do complex curve-generating projections.

See Also

geoshow | geographicGrid | meshm | surfacem | surfm

Introduced before R2006a

pix2latlon

(To be removed) Convert pixel coordinates to latitude-longitude coordinates

---

Note The pix2latlon function will be removed in a future release. Use the intrinsicToGeographic function instead. For more information, see "Compatibility Considerations".

Syntax

```
[lat, lon] = pix2latlon(R,row,col)
```

Description

[lat, lon] = pix2latlon(R,row,col) calculates latitude-longitude coordinates lat, lon from pixel coordinates row, col. R is either a 3-by-2 referencing matrix that transforms intrinsic pixel coordinates to geographic coordinates, or a geographic raster reference object. row and col are vectors or arrays of matching size. The outputs lat and lon have the same size as row and col.

Examples

Find the latitude and longitude of the upper left outer corner of a gridded data set.

```
load korea5c
[UL_lat,UL_lon] = pix2latlon(korea5cR,0.5,0.5)
```

The output appears as follows:

```
UL_lat =

    30


UL_lon =

   115
```

Compatibility Considerations

pix2latlon will be removed
Not recommended starting in R2021b

Some functions that accept referencing matrices as input will be removed, including the pix2latlon function. Use a geographic raster reference object and the intrinsicToGeographic function instead. Reference objects have several advantages over referencing matrices.

- Unlike referencing matrices, reference objects have properties that document the size of the associated raster, its geographic limits, and the direction of its rows and columns. For more information about reference object properties, see the GeographicCellsReference and GeographicPostingsReference objects.

- You can manipulate the limits of rasters associated with reference objects using the `geocrop` function.
- You can manipulate the size and resolution of rasters associated with reference objects using the `georesize` function.
- Most functions that accept referencing matrices as inputs also accept reference objects.

To update your code, first create a reference object for either a raster of cells using the `georefcells` function or a raster of regularly posted samples using the `georefpostings` function. Alternatively, convert from a referencing matrix to a reference object using the `refmatToGeoRasterReference` function.

Then, replace uses of the `pix2latlon` function with the `intrinsicToGeographic` function according to this pattern.

Will Be Removed	Recommended
`[lat,lon] = pix2latlon(R,row,col);`	`[lat,lon] = intrinsicToGeographic(R,col,row);`

See Also
geographicToIntrinsic | intrinsicToWorld | intrinsicToGeographic

Introduced before R2006a

pix2map

(To be removed) Convert pixel coordinates to map coordinates

Note The pix2map function will be removed in a future release. Use the intrinsicToWorld function instead. For more information, see "Compatibility Considerations".

Syntax

```
[x,y] = pix2map(R,row,col)
s = pix2map(R,row,col)
[...] = pix2map(R,p)
```

Description

[x,y] = pix2map(R,row,col) calculates map coordinates x,y from pixel coordinates row,col. R is either a 3-by-2 referencing matrix defining a two-dimensional affine transformation from intrinsic pixel coordinates to map coordinates, or a map raster reference object. row and col are vectors or arrays of matching size. The outputs x and y have the same size as row and col.

s = pix2map(R,row,col) combines x and y into a single array s. If row and col are column vectors of length n, then s is an n-by-2 matrix and each row (s(k,:)) specifies the map coordinates of a single point. Otherwise, s has size [size(row) 2], and s(k1,k2,...,kn,:) contains the map coordinates of a single point.

[...] = pix2map(R,p) combines row and col into a single array p. If row and col are column vectors of length n, then p should be an n-by-2 matrix such that each row (p(k,:)) specifies the pixel coordinates of a single point. Otherwise, p should have size [size(row) 2], and p(k1,k2,...,kn,:) should contain the pixel coordinates of a single point.

Examples

```
% Find the map coordinates for the pixel at (100,50).
[X,cmap] = imread('concord_ortho_w.tif');
R = worldfileread('concord_ortho_w.tfw','planar',size(X));
[x,y] = pix2map(R,100,50);
```

Compatibility Considerations

pix2map will be removed
Not recommended starting in R2021b

Some functions that accept referencing matrices as input will be removed, including the pix2map function. Use a map raster reference object and the intrinsicToWorld function instead. Reference objects have several advantages over referencing matrices.

- Unlike referencing matrices, reference objects have properties that document the size of the associated raster, its limits, and the direction of its rows and columns. For more information about reference object properties, see the MapPostingsReference and MapCellsReference objects.

- You can manipulate the limits of rasters associated with reference objects using the `mapcrop` function.
- You can manipulate the size and resolution of rasters associated with reference objects using the `mapresize` function.
- Most functions that accept referencing matrices as inputs also accept reference objects.

To update your code, first create a reference object for either a raster of cells using the `maprefcells` function or a raster of regularly posted samples using the `maprefpostings` function. Alternatively, convert from a referencing matrix to a reference object using the `refmatToMapRasterReference` function.

Then, replace uses of the `pix2map` function with the `intrinsicToWorld` function according to these patterns.

Will Be Removed	Recommended
`[x,y] = pix2map(R,row,col);`	`[x,y] = intrinsicToWorld(R,col,row);`

`s = pix2map(R,row,col);`	If `row` and `col` are column vectors:
	
	`[x,y] = intrinsicToWorld(R,col,row);`
	`s = [x y];`
	
	If `row` and `col` have size *M*-by-*N* or 1-by-*N*:
	
	`[x,y] = intrinsicToWorld(R,col,row);`
	`s = cat(3,x,y);`

`[x,y] = pix2map(R,p);`	If `p` has size *N*-by-2: `[x,y] = intrinsicToWorld(R,p(:,2),p(:,1));`

See Also
`worldToIntrinsic` | `intrinsicToGeographic` | `worldfileread` | `intrinsicToWorld`

Introduced before R2006a

pixcenters

(To be removed) Compute pixel centers for georeferenced image or data grid

---

Note The `pixcenters` function will be removed in a future release. Use the `geographicGrid` or `worldGrid` function instead. For more information, see "Compatibility Considerations".

Syntax

```
[x,y] = pixcenters(R,height,width)
[x,y] = pixcenters(r,sizea)
[x,y] = pixcenters(..., 'makegrid')
```

Description

`[x,y] = pixcenters(R,height,width)` returns the spatial coordinates of a spatially-referenced image or regular gridded data set. `R` is either a 3-by-2 referencing matrix defining a 2-dimensional affine transformation from intrinsic pixel coordinates to map coordinates, or a `MapCellsReference` object. `height` and `width` are the image dimensions. If `r` does not include a rotation (i.e., `r(1,1) = r(2,2) = 0`), then `x` is a 1-by-`width` vector and `y` is a 1-by-`height` vector. In this case, the spatial coordinates of the pixel in row `row` and column `col` are given by `x(col)`, `y(row)`. Otherwise, `x` and `y` are each a `height`-by-`width` matrix such that `x(col,row)`, `y(col,row)` are the coordinates of the pixel with subscripts (`row`,`col`).

`[x,y] = pixcenters(r,sizea)` accepts the size vector `sizea = [height, width, ...]` instead of `height` and `width`.

`[x,y] = pixcenters(info)` accepts a scalar structure array with the fields

`'RefMatrix'`	3-by-2 referencing matrix
`'Height'`	Scalar number
`'Width'`	Scalar number

`[x,y] = pixcenters(..., 'makegrid')` returns `x` and `y` as `height`-by-`width` matrices even if `r` is irrotational. This syntax can be helpful when you call `pixcenters` from within a function or script.

Examples

```
[Z,R] = readgeoraster('MtWashington-ft.grd','OutputType','double');
info = georasterinfo('MtWashington-ft.grd');
Z = standardizeMissing(Z,info.MissingDataIndicator);
figure
[x,y] = pixcenters(R,size(Z));
h = surf(x,y,Z);
axis equal
demcmap(Z)
set(h,'EdgeColor','none')
xlabel('x (easting in meters)')
```

```
ylabel('y (northing in meters)')
zlabel('elevation in feet')
```

Tips

`pixcenters` is useful for working with `surf`, `mesh`, or `surface`, and for coordinate transformations.

Compatibility Considerations

pixcenters will be removed
Not recommended starting in R2021a

Some functions that accept referencing matrices as input will be removed, including the `pixcenters` function. Use a raster reference object and the `geographicGrid` or `worldGrid` function instead. Reference objects have several advantages over referencing matrices.

- Unlike referencing matrices, reference objects have properties that document the size of the associated raster, its limits, and the direction of its rows and columns. For more information about reference object properties, see the `GeographicCellsReference` and `MapCellsReference` objects.
- You can manipulate the limits of rasters associated with reference objects using the `geocrop` or `mapcrop` function.
- You can manipulate the size and resolution of rasters associated with reference objects using the `georesize` or `mapresize` function.
- Most functions that accept referencing matrices as inputs also accept reference objects.

Depending on the `pixcenters` function syntax, there are different ways to update your code.

- If the raster is referenced to projected *x-y* coordinates, use a map raster reference object and the `worldGrid` function. This table shows replacement patterns for the `pixcenters` function syntaxes that specify a referencing matrix `refmat`, scalar structure `info`, or GeoTIFF information structure `geoinfo`. Note that the default behavior of the `pixcenters` function is to return row vectors and the default behavior of the `worldGrid` function is to return 2-D arrays. To return row vectors using the `worldGrid` function, specify the second argument as `'gridvectors'`.

Will Be Removed	Recommended
`[X,Y] = pixcenters(refmat,height,width);`	`R = refmatToMapRasterReference(refmat,[height widt` `[X,Y] = worldGrid(R,'gridvectors');`
`[X,Y] = pixcenters(refmat,rasterSize);`	`R = refmatToMapRasterReference(refmat,rasterSize);` `[X,Y] = worldGrid(R,'gridvectors');`
`[X,Y] = pixcenters(info);`	`R = refmatToMapRasterReference( ...` `    info.RefMatrix, [info.Height info.Width]);` `[X,Y] = worldGrid(R,'gridvectors');`
`[X,Y] = pixcenters(geoinfo);`	`[X,Y] = worldGrid(geoinfo.SpatialRef,'gridvectors'`

If the `TransformationType` property of the reference object `R` is `'affine'`, the `[x,y] = worldgrid(R,'gridvectors')` syntax displays a warning. Temporarily suppress the warning by including this code before the call to the `worldGrid` function.

```
w = warning('off','map.spatialref.fullGridForAffine');
c = onCleanup(@() warning(w));
```

- If the raster is referenced to latitude and longitude coordinates, use a geographic raster reference object and the `geographicGrid` function. This table shows replacement patterns for function syntaxes that specify a referencing matrix `refmat` or scalar structure `info`. Note that the default behavior of the `pixcenters` function is to return row vectors and the default behavior of the `geographicGrid` function is to return 2-D arrays. To return row vectors using the `geographicGrid` function, specify the second argument as `'gridvectors'`.

Will Be Removed	Recommended
`[lon,lat] = pixcenters(refmat,height,width);`	`R = refmatToGeoRasterReference(refmat,[height widt` `[lat,lon] = geographicGrid(R,'gridvectors');`
`[lon,lat] = pixcenters(refmat,rasterSize);`	`R = refmatToGeoRasterReference(refmat,rasterSize);` `[lat,lon] = geographicGrid(R,'gridvectors');`

`[lon,lat] = pixcenters(info);`	`R = refmatToGeoRasterReference( ...`
	`    info.RefMatrix, [info.Height info.Width]);`
	`[lat,lon] = geographicGrid(R,'gridvectors');`

See Also

Functions
readgeoraster | mapoutline | intrinsicToWorld | worldfileread | mapshow | geographicGrid | worldGrid

Objects
MapCellsReference

Introduced before R2006a

plabel

Toggle and control display of parallel labels

Syntax

```
plabel
plabel('on')
plabel('off')
plabel(meridian)
plabel(MapAxesPropertyName,PropertyValue,...)
```

Description

`plabel` toggles the visibility of parallel labeling on the current map axes.

`plabel('on')` sets the visibility of parallel labels to `'on'`.

`plabel('off')` sets the visibility of parallel labels to `'off'`.

`plabel('reset')` resets the displayed parallel labels using the currently defined parallel label properties.

`plabel(meridian)` sets the value of the `PLabelMeridian` property of the map axes to the value meridian. This determines the meridian upon which the labels are placed (see `axesm`). The options for `meridian` are a scalar longitude or `'east'`, `'west'`, or `'prime'`.

`plabel(MapAxesPropertyName,PropertyValue,...)` allows paired map axes property names and property values to be passed in. For a complete description of map axes properties, see the `axesm` reference page.

Parallel label handles can be returned in `h` if desired.

See Also
`axesm` | `setm` | `mlabel`

Introduced before R2006a

plot3m

Project 3-D lines and points on map axes

Syntax

```
h = plot3m(lat,lon,z)
h = plot3m(lat,lon,LineSpec)
h = plot3m(lat,lon,PropertyName,PropertyValue,...)
```

Description

`h = plot3m(lat,lon,z)` displays projected line objects on the current map axes. `lat` and `lon` are the latitude and longitude coordinates, respectively, of the line object to be projected. Note that this ordering is conceptually reversed from the MATLAB `line` function, because the *vertical* (*y*) coordinate comes first. However, the ordering latitude, then longitude, is standard geographic usage. `lat` and `lon` must be the same size, and in the `AngleUnits` of the map axes. `z` is the altitude data associated with each point in `lat` and `lon`. The object handle for the displayed line can be returned in `h`.

The units of `z` are arbitrary, except when using the globe projection. In the case of `globe`, `z` should have the same units as the radius of the earth or semimajor axis specified in the `'geoid'` (reference ellipsoid) property of the map axes. This implies that when the reference ellipsoid is a unit sphere, the units of `z` are earth radii.

`h = plot3m(lat,lon,LineSpec)` specifies the line style, marker, and color on page 1-1091.

`h = plot3m(lat,lon,PropertyName,PropertyValue,...)` allows the specification of any number of property name/property value pairs for any properties recognized by the MATLAB `line` function except for `XData`, `YData`, and `ZData`.

Examples

```
axesm sinusoid; framem; view(3)
[lats,longs] = interpm([45 -45 -45 45 45 -45]',...
                        [-100 -100 100 100 -100 -100]',1);
z = (1:671)'/100;
plot3m(lats,longs,z,'m')
```

More About

Line Style, Marker, and Color

Specify the line style, marker, and color as a character vector or string containing symbols. The symbols can appear in any order. You do not need to specify all three characteristics (line style, marker, and color). For example, specify a red dashed line using `--r`.

Line Style	Description	Resulting Line
`'-'`	Solid line	————————
`'--'`	Dashed line	– – – – –
`':'`	Dotted line	....
`'-.'`	Dash-dotted line	—·—·—·—

Marker	Description	Resulting Marker
`'o'`	Circle	○
`'+'`	Plus sign	+
`'*'`	Asterisk	✳
`'.'`	Point	•
`'x'`	Cross	×
`'_'`	Horizontal line	—
`'|'`	Vertical line		

Marker	Description	Resulting Marker
`'s'`	Square	□
`'d'`	Diamond	◇
`'^'`	Upward-pointing triangle	△
`'v'`	Downward-pointing triangle	▽
`'>'`	Right-pointing triangle	▷
`'<'`	Left-pointing triangle	◁
`'p'`	Pentagram	☆
`'h'`	Hexagram	✩

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	[1 0 0]	
`'green'`	`'g'`	[0 1 0]	
`'blue'`	`'b'`	[0 0 1]	
`'cyan'`	`'c'`	[0 1 1]	
`'magenta'`	`'m'`	[1 0 1]	
`'yellow'`	`'y'`	[1 1 0]	
`'black'`	`'k'`	[0 0 0]	
`'white'`	`'w'`	[1 1 1]	

Tips

`plot3m` is the mapping equivalent of the MATLAB `plot3` function.

See Also

`linem` | `plot3` | `plotm`

plotm

Project 2-D lines and points on map axes

Syntax

```
plotm(lat,lon)
plotm([lat lon])
plotm(lat,lon,LineSpec)
plotm(lat,lon,Name,Value)
h = plotm( ___ )
```

Description

`plotm` is the mapping equivalent of the MATLAB `plot` function.

`plotm(lat,lon)` displays projected line objects on the current map axes. `lat` and `lon` are the latitude and longitude coordinates, respectively, of the line object to be projected. This ordering of latitude then longitude is standard geographic usage. However, this ordering is conceptually reversed from the MATLAB `line` function, in which the *horizontal* (*x*) coordinate comes first.

`plotm([lat lon])` allows the latitude and longitude coordinates to be packed into a single two-column matrix.

`plotm(lat,lon,LineSpec)` specifies the line style, marker, and color.

`plotm(lat,lon,Name,Value)` uses name-value pair arguments to specify any number of Line Properties except for `XData`, `YData`, and `ZData`. `Name` is the property name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`. Property names can be abbreviated, and case does not matter.

`h = plotm( ___ )` returns a handle to the displayed line.

Examples

Plot Coastlines on Map

First, load coastline data into the workspace and create a set of map axes. Then, plot the data. The `plotm` function uses the `NaN` values in `coastlat` and `coastlon` to break the data into separate lines.

```
load coastlines
ax = axesm('sinusoid','Frame','on');
plotm(coastlat,coastlon,'g')
```
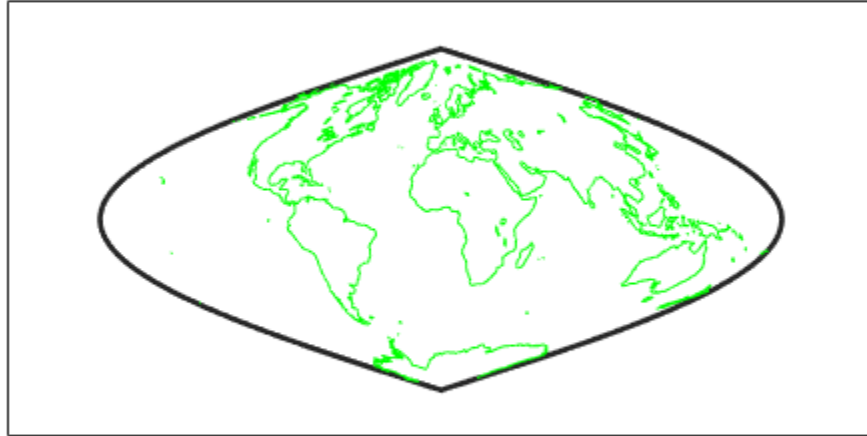
Input Arguments

`lat, lon` — Latitude or longitude vector
numeric vector

Latitude or longitude vector, specified as a numeric vector. Specify values in units that match the `AngleUnits` property of the map axes. `lat` and `lon` must be the same size.

Create breaks in lines or polygons using `NaN` values. For example, this code plots the first three elements, skips the fourth element, and draws another line using the last three elements.
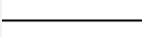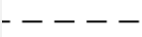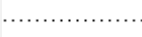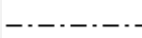
```
lat = [0 1 2 NaN 4 5 6];
lon = [0 1 2 NaN 3 4 5];
axesm('UTM','Zone','31N','Frame','on')
plotm(lat,lon)
```

`LineSpec` — Line style, marker, and color
character vector | string

Line style, marker, and color, specified as a character vector or string containing symbols. The symbols can appear in any order. You do not need to specify all three characteristics (line style, marker, and color). For example, if you omit the line style and specify the marker, then the plot shows only the marker and no line.

Example: `'--or'` is a red dashed line with circle markers

Line Style	Description	Resulting Line
'-'	Solid line	————————
'--'	Dashed line	- — — — —
':'	Dotted line	....
'-.'	Dash-dotted line	—·—·—·—·—

Marker	Description	Resulting Marker
'o'	Circle	○
'+'	Plus sign	+
'*'	Asterisk	✳
'.'	Point	•
'x'	Cross	×
'_'	Horizontal line	—
'|'	Vertical line		
's'	Square	□
'd'	Diamond	◇
'^'	Upward-pointing triangle	△
'v'	Downward-pointing triangle	▽
'>'	Right-pointing triangle	▷
'<'	Left-pointing triangle	◁
'p'	Pentagram	☆
'h'	Hexagram	✡

Color Name	Short Name	RGB Triplet	Appearance
'red'	'r'	[1 0 0]	▬
'green'	'g'	[0 1 0]	▬
'blue'	'b'	[0 0 1]	▬
'cyan'	'c'	[0 1 1]	▬
'magenta'	'm'	[1 0 1]	▬

Color Name	Short Name	RGB Triplet	Appearance
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Output Arguments

h — Handle to displayed line
handle object

Handle to the displayed line, returned as a handle to a MATLAB graphics object.

Tips

When you pass two points to the `plotm` function, the function projects the locations of each point on the map and displays a straight line that connects them, regardless of the map projection or the distance between the points. To display a projected geodesic or rhumbline curve between two points, first insert intermediate points using the `track2` function.

See Also

linem | plot | plot3m

Introduced before R2006a

polcmap

Create colormap appropriate to political regions

Syntax

```
polcmap
polcmap(ncolors)
polcmap(ncolors,maxsat)
polcmap(ncolors,huelimits,saturationlimits,valuelimits)
cmap = polcmap( ___ )
```

Description

`polcmap` applies a random, muted colormap to the current figure. The size of the colormap is the same as the existing colormap.

`polcmap(ncolors)` creates a colormap with the specified number of colors.

`polcmap(ncolors,maxsat)` controls the maximum saturation of the colors.

`polcmap(ncolors,huelimits,saturationlimits,valuelimits)` controls the hue, saturation, and value of the colors. `polcmap` randomly selects values within the limit vectors. These are two-element vectors of the form `[min max]`. Valid values range from 0 to 1. As the hue varies from 0 to 1, the resulting color varies from red, through yellow, green, cyan, blue, and magenta, back to red. When the saturation is 0, the colors are unsaturated; they are simply shades of gray. When the saturation is 1, the colors are fully saturated; they contain no white component. As the value varies from 0 to 1, the brightness increases.

`cmap = polcmap( ___ )` returns the colormap without applying it to the figure.

Examples

Color Patches with SymbolSpec Created Using `polcmap`

Create an empty map axes with a Lambert conformal conic projection and map limits covering Texas.

```
figure
usamap("texas")
```
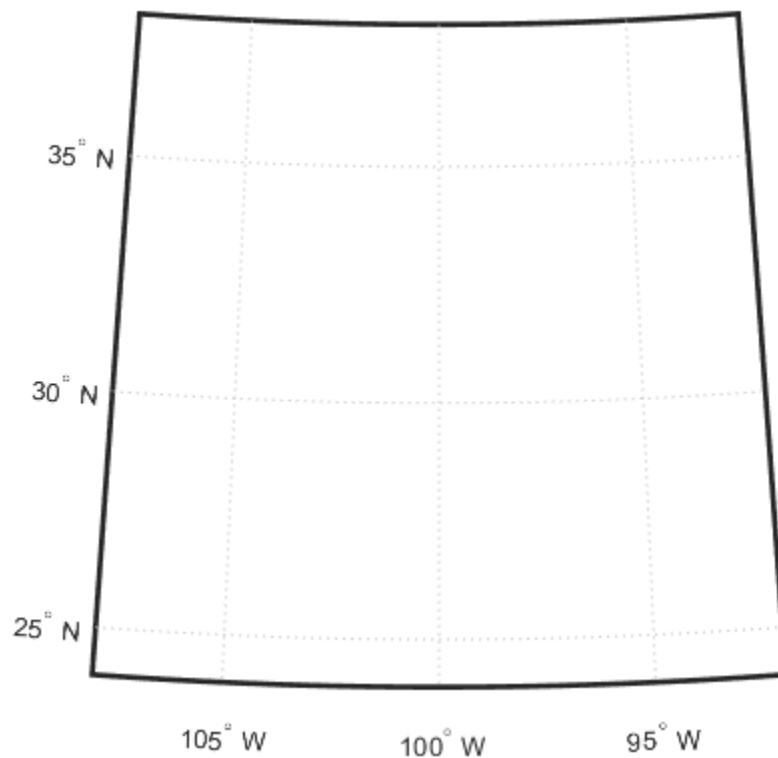
Read state boundaries from a shapefile into a geospatial table.

```
states = readgeotable("usastatelo.shp");
```

Define the colors you want to apply to the shapes (`states`) in a symbol specification. Use `polcmap` to create a colormap the same size as the number of rows in the geospatial table. `polcmap` creates a palette of muted colors.

```
h = height(states);
faceColors = makesymbolspec("Polygon", ...
    {'INDEX',[1 h],'FaceColor',polcmap(h)});
```

Display the map using the symbol specification to color the patches (`states`) in the map. The colors you obtain for this example can vary each time you run this example because `polcmap` computes them randomly.

```
geoshow(states,"DisplayType","polygon","SymbolSpec",faceColors)
```
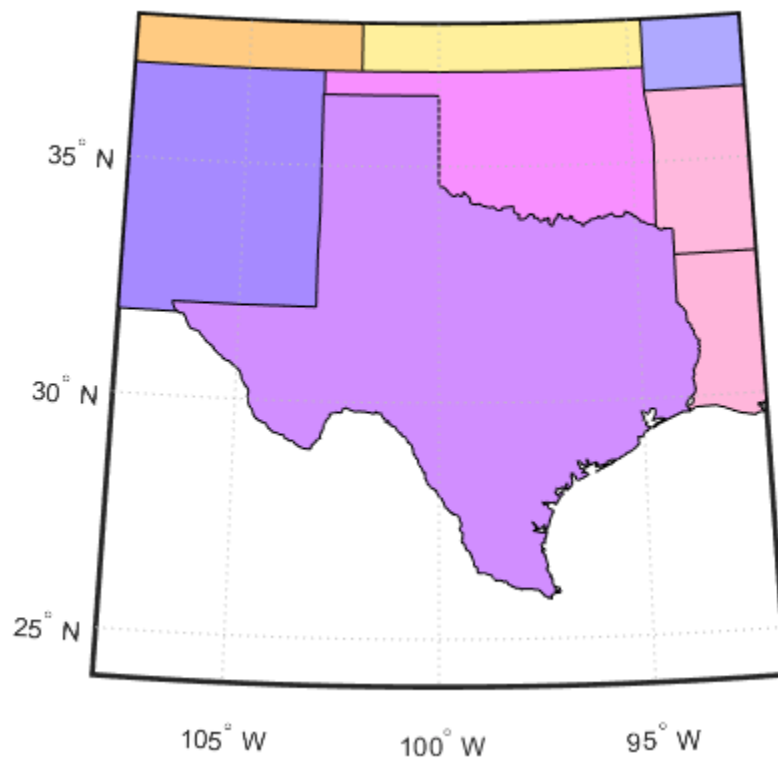
Input Arguments

`ncolors` — Number of colors in the color map
numeric scalar

Number of colors in the color map, specified as a numeric scalar.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`maxsat` — Maximum saturation of colors
`0.5` (default) | numeric scalar

Maximum saturation of colors, specified as a numeric scalar. Larger maximum saturation values produce brighter, more saturated colors.

Data Types: `single` | `double` | `logical` | `char`

`huelimits` — Color range limits
`[0 1]` (default) | two-element vector

Color range limits, specified as a two-element vector of the form `[min max]`. Values range from 0 to 1.0. As the hue varies from 0 to 1, the resulting color varies from red, through yellow, green, cyan, blue, and magenta, back to red.

Data Types: `single` | `double`

saturationlimits — Color saturation limits
[.25 .5] (default) | two-element vector

Color saturation limits, specified as a two-element vector of the form [min max]. Values range from 0 to 1.0. When the saturation is 0, the colors are unsaturated; they are simply shades of gray. When the saturation is 1, the colors are fully saturated; they contain no white component.

Data Types: single | double

valuelimits — Brightness of colors
[1 1] (default) | two-element vector

Brightness of colors, specified as a two-element vector of the form [min max]. Values range from 0 to 1.0. As the value varies from 0 to 1, the brightness increases.

Data Types: single | double

Output Arguments

cmap — Colormap
m-by-3 numeric array

Colormap, returned as an *m*-by-3 numeric array of class double or class single, depending on the type of the input.

Tips

- You cannot use polcmap to alter the colors of displayed patches drawn by geoshow or mapshow. The patches must have been rendered by displaym. However, you can color patches using polcmap when you call geoshow or mapshow (see "Color Patches with SymbolSpec Created Using polcmap" on page 1-1097).

See Also
demcmap | colormap

Introduced before R2006a

poly2ccw

Convert polygon contour to counterclockwise vertex ordering

Syntax

```
[x2,y2] = poly2ccw(x1,y1)
```

Description

`[x2,y2] = poly2ccw(x1,y1)` arranges the Cartesian vertices in the polygonal contour (`x1,y1`) in counterclockwise order, returning the result in `x2` and `y2`. If `x1` and `y1` can contain multiple contours, represented either as `NaN`-separated vectors or as cell arrays, then each contour is converted to clockwise ordering. `x2` and `y2` have the same format (`NaN`-separated vectors or cell arrays) as `x1` and `y1`.

Examples

Convert a clockwise-ordered square to counterclockwise ordering.

```
x1 = [0 0 1 1 0];
y1 = [0 1 1 0 0];
ispolycw(x1, y1)

ans =
        1

[x2, y2] = poly2ccw(x1, y1);
ispolycw(x2, y2)
ans =
        0
```

Tips

You can use `poly2ccw` for geographic coordinates if the polygon does not cross the Antimeridian or contain a pole. A polygon contains a pole if the longitude data spans 360 degrees. To use `poly2ccw` with geographic coordinates, specify the longitude vector as `x1` and the latitude vector as `y1`.

See Also
`poly2cw` | `ispolycw` | `polyshape`

Topics
"Create and Display Polygons"

Introduced before R2006a

poly2cw

Convert polygon contour to clockwise vertex ordering

Syntax

```
[x2, y2] = poly2cw(x1, y1)
```

Description

`[x2, y2] = poly2cw(x1, y1)` arranges the Cartesian vertices in the polygonal contour (`x1, y1`) in clockwise order, returning the result in `x2` and `y2`. If `x1` and `y1` can contain multiple contours, represented either as `NaN`-separated vectors or as cell arrays, then each contour is converted to clockwise ordering. `x2` and `y2` have the same format (`NaN`-separated vectors or cell arrays) as `x1` and `y1`.

Examples

Convert a counterclockwise-ordered square to clockwise ordering.

```
x1 = [0 1 1 0 0];
y1 = [0 0 1 1 0];
ispolycw(x1, y1)

ans =
        0

[x2, y2] = poly2cw(x1, y1);
ispolycw(x2, y2)

ans =
        1
```

Tips

You can use `poly2cw` for geographic coordinates if the polygon does not cross the Antimeridian or contain a pole. A polygon contains a pole if the longitude data spans 360 degrees. To use `poly2cw` with geographic coordinates, specify the longitude vector as `x1` and the latitude vector as `y1`.

See Also
`poly2ccw` | `ispolycw` | `polyshape`

Topics
"Create and Display Polygons"

Introduced before R2006a

poly2fv

Convert polygonal region to patch faces and vertices

Syntax

```
[F,V] = poly2fv(x,y)
```

Description

`[F,V] = poly2fv(x,y)` converts the polygonal region represented by the contours `(x,y)` into a faces matrix, `F`, and a vertices matrix, `V`, that can be used with the `patch` function to display the region. If the polygon represented by `x` and `y` has multiple parts, either the `NaN`-separated vector format or the cell array format may be used. The `poly2fv` function creates triangular faces.

Most Mapping Toolbox functions adhere to the convention that individual contours with clockwise-ordered vertices are external contours and individual contours with counterclockwise-ordered vertices are internal contours. Although the `poly2fv` function ignores vertex order, you should follow the convention when creating contours to ensure consistency with other functions.

Examples

Display a rectangular region with two holes using a single patch object.

```
% External contour, rectangle.
x1 = [0 0 6 6 0];
y1 = [0 3 3 0 0];

% First hole contour, square.
x2 = [1 2 2 1 1];
y2 = [1 1 2 2 1];

% Second hole contour, triangle.
x3 = [4 5 4 4];
y3 = [1 1 2 1];

% Compute face and vertex matrices.
[f, v] = poly2fv({x1, x2, x3}, {y1, y2, y3});

% Display the patch.
patch('Faces', f, 'Vertices', v, 'FaceColor', 'r', ...
 'EdgeColor', 'none');
axis off, axis equal
```

See Also

Functions
`ispolycw` | `patch` | `poly2cw` | `poly2ccw` | `polyshape`

Topics
"Create and Display Polygons"

Introduced before R2006a

polybool

Set operations on polygonal regions

Note `polybool` is not recommended. Use `polyshape` instead.

To replace calls to `polybool`, create `polyshape` objects to represent the shapes, call the appropriate `polyshape` object function for the equivalent Boolean operation, and then call the `polyshape boundary` object function. For example, this call to `polybool` specifies the union operation as the first argument.

```
[Cx,Cy] = polybool('union',Ax,Ay,Bx,By)
```

To update this usage, create separate `polyshape` objects for each shape and then use the `union` object function associated with the `polyshape` object. `polyshape` supports the Boolean operations supported by `polybool`: union, intersection, subtraction, and exclusive OR. Use the `polyshape boundary` object function to return `Cx` and `Cy`. See `polyshape` for a complete list of object functions, including `plot`.

```
A = polyshape(Ax,Ay,'Simplify',false);
B = polyshape(Bx,By,'Simplify',false);
C = union(A,B);
[Cx,Cy] = boundary(C);
```

Note that the polygon vertex order is likely to differ between the output from `polybool` and the output from the call to `boundary`, because there is no single right answer. (Even in a simple one-region polygon, the vertices can be permuted cyclically without affecting the underlying geometry.) In addition, if the geometries of the inputs are not perfectly clean (free from self-intersections, etc.), then the `polyshape union` operation may make small changes that are not necessarily performed in `polybool`.

Syntax

```
[x,y] = polybool(flag,x1,y1,x2,y2)
```

Description

`[x,y] = polybool(flag,x1,y1,x2,y2)` performs the polygon set operation identified by `flag`. The result is output using the same format as the input. Geographic data that encompasses a pole cannot be used directly. Use `flatearthpoly` to convert polygons that contain a pole to Cartesian coordinates.

Most Mapping Toolbox functions adhere to the convention that individual contours with clockwise-ordered vertices are external contours and individual contours with counterclockwise-ordered vertices are internal contours. Although the `polybool` function ignores vertex order, follow this convention when creating contours to ensure consistency with other functions.

Examples

Set Operations on Two Overlapping Circular Regions

```matlab
theta = linspace(0, 2*pi, 100);
x1 = cos(theta) - 0.5;
y1 = -sin(theta);     % -sin(theta) to make a clockwise contour
x2 = x1 + 1;
y2 = y1;
[xa, ya] = polybool('union', x1, y1, x2, y2);
[xb, yb] = polybool('intersection', x1, y1, x2, y2);
[xc, yc] = polybool('xor', x1, y1, x2, y2);
[xd, yd] = polybool('subtraction', x1, y1, x2, y2);

subplot(2, 2, 1)
patch(xa, ya, 1, 'FaceColor', 'r')
axis equal, axis off, hold on
plot(x1, y1, x2, y2, 'Color', 'k')
title('Union')

subplot(2, 2, 2)
patch(xb, yb, 1, 'FaceColor', 'r')
axis equal, axis off, hold on
plot(x1, y1, x2, y2, 'Color', 'k')
title('Intersection')

subplot(2, 2, 3)
% The output of the exclusive-or operation consists of disjoint
% regions.  It can be plotted as a single patch object using the
% face-vertex form.  Use poly2fv to convert a polygonal region
% to face-vertex form.
[f, v] = poly2fv(xc, yc);
patch('Faces', f, 'Vertices', v, 'FaceColor', 'r', ...
   'EdgeColor', 'none')
axis equal, axis off, hold on
plot(x1, y1, x2, y2, 'Color', 'k')
title('Exclusive Or')

subplot(2, 2, 4)
patch(xd, yd, 1, 'FaceColor', 'r')
axis equal, axis off, hold on
plot(x1, y1, x2, y2, 'Color', 'k')
title('Subtraction')
```
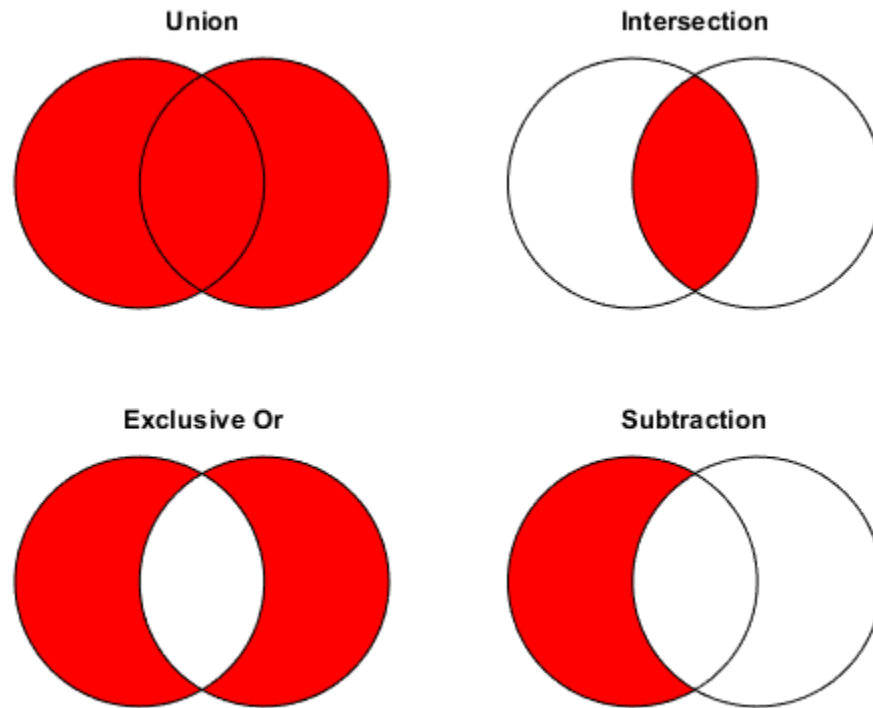
Set Operations on Regions with Holes

```
Ax = {[1 1 6 6 1], [2 5 5 2 2], [2 5 5 2 2]};
Ay = {[1 6 6 1 1], [2 2 3 3 2], [4 4 5 5 4]};
subplot(2, 3, 1)
[f, v] = poly2fv(Ax, Ay);
patch('Faces', f, 'Vertices', v, 'FaceColor', 'r', ...
  'EdgeColor', 'none')
axis equal, axis off, axis([0 7 0 7]), hold on
for k = 1:numel(Ax), plot(Ax{k}, Ay{k}, 'Color', 'k'), end
title('A')

Bx = {[0 0 7 7 0], [1 3 3 1 1], [4 6 6 4 4]};
By = {[0 7 7 0 0], [1 1 6 6 1], [1 1 6 6 1]};
subplot(2, 3, 4);
[f, v] = poly2fv(Bx, By);
patch('Faces', f, 'Vertices', v, 'FaceColor', 'r', ...
  'EdgeColor', 'none')
axis equal, axis off, axis([0 7 0 7]), hold on
for k = 1:numel(Bx), plot(Bx{k}, By{k}, 'Color', 'k'), end
title('B')

subplot(2, 3, 2)
[Cx, Cy] = polybool('union', Ax, Ay, Bx, By);
[f, v] = poly2fv(Cx, Cy);
```

```matlab
patch('Faces', f, 'Vertices', v, 'FaceColor', 'r', ...
  'EdgeColor', 'none')
axis equal, axis off, axis([0 7 0 7]), hold on
for k = 1:numel(Cx), plot(Cx{k}, Cy{k}, 'Color', 'k'), end
title('A \cup B')

subplot(2, 3, 3)
[Dx, Dy] = polybool('intersection', Ax, Ay, Bx, By);
[f, v] = poly2fv(Dx, Dy);
patch('Faces', f, 'Vertices', v, 'FaceColor', 'r', ...
  'EdgeColor', 'none')
axis equal, axis off, axis([0 7 0 7]), hold on
for k = 1:numel(Dx), plot(Dx{k}, Dy{k}, 'Color', 'k'), end
title('A \cap B')

subplot(2, 3, 5)
[Ex, Ey] = polybool('subtraction', Ax, Ay, Bx, By);
[f, v] = poly2fv(Ex, Ey);
patch('Faces', f, 'Vertices', v, 'FaceColor', 'r', ...
  'EdgeColor', 'none')
axis equal, axis off, axis([0 7 0 7]), hold on
for k = 1:numel(Ex), plot(Ex{k}, Ey{k}, 'Color', 'k'), end
title('A - B')

subplot(2, 3, 6)
[Fx, Fy] = polybool('xor', Ax, Ay, Bx, By);
[f, v] = poly2fv(Fx, Fy);
patch('Faces', f, 'Vertices', v, 'FaceColor', 'r', ...
  'EdgeColor', 'none')
axis equal, axis off, axis([0 7 0 7]), hold on
for k = 1:numel(Fx), plot(Fx{k}, Fy{k}, 'Color', 'k'), end
title('XOR(A, B)')
```
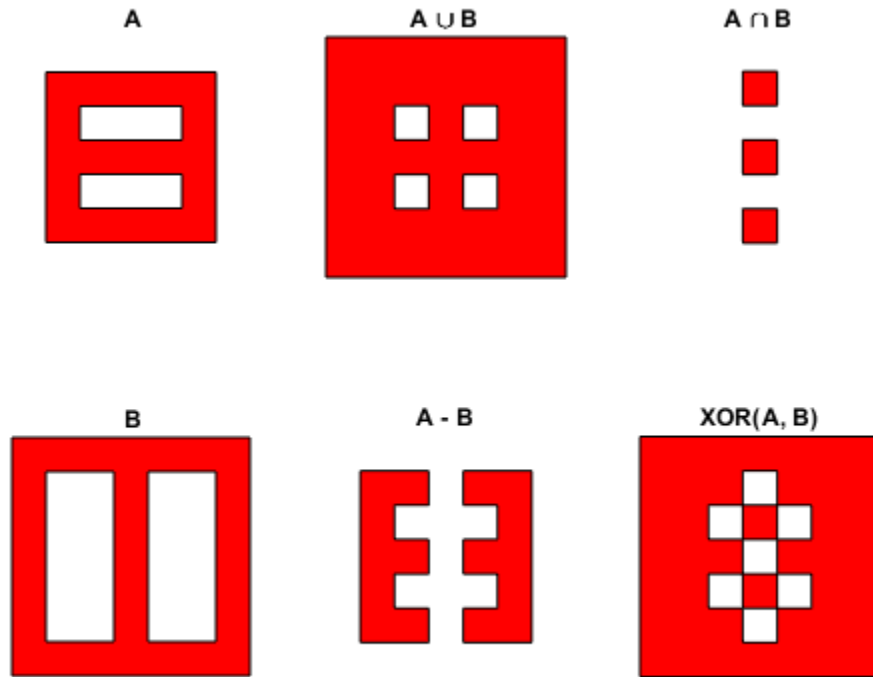
Input Arguments

flag — Polygon set operation
`'intersection'|'and'|'&'|'union'|'or'|'|'|'+'|'plus'|'subtraction'|`
`'minus'|'-'|'exclusiveor'|'xor'`

Polygon set operation, specified as one of the following values.

Operation					
Region intersection	`'intersection'`	`'and'`	`'&'`		
Region union	`'union'`	`'or'`	`'|'`	`'+'`	`'plus'`
Region subtraction	`'subtraction'`	`'minus'`	`'-'`		
Region exclusive or	`'exclusiveor'`	`'xor'`			

Data Types: `char` | `string`

x1 — Polygon contours
`NaN`-delimited vector | cell array

Polygon contours, specified as a `NaN`-delimited vector or cell array.

Data Types: `double`

x2 — Polygon contours
`NaN`-delimited vector | cell array

Polygon contours, specified as a `NaN`-delimited vector or cell array.

Data Types: `double`

y1 — Polygon contours
`NaN`-delimited vector | cell array

Polygon contours, specified as a `NaN`-delimited vector or cell array.

Data Types: `double`

y2 — Polygon contours
`NaN`-delimited vector | cell array

Polygon contours, specified as a `NaN`-delimited vector or cell array.

Data Types: `double`

Output Arguments

x — Polygon contour after set operation
`NaN`-delimited vector | cell array

Polygon contour after set operation, returned as a `NaN`-delimited vector or cell array. The output returns in the same format as the input.

y — Polygon contour after set operation
`NaN`-delimited vector | cell array

Polygon contour after set operation, returned as a `NaN`-delimited vector or cell array. The output returns in the same format as the input.

Tips

* Numerical problems can occur when the polygons have a large offset from the origin. To avoid this issue, translate the coordinates to a location closer to the origin before performing the operation. Then, undo the translation after completing the operation. For example:

```
[x,y] = polybool(flag,x1-xt,y1-yt,x2-xt,y2-yt);

x = x+xt;

y = y+yt;
```

See Also

bufferm | flatearthpoly | ispolycw | poly2ccw | polyjoin | polysplit | poly2fv | poly2cw | polyshape

Introduced before R2006a

polycut

Polygon branch cuts for holes

Syntax

```
[lat2,long2] = polycut(lat,long)
```

Description

`[lat2,long2] = polycut(lat,long)` connects the contour and holes of polygons using optimal branch cuts. Polygons are input as `NaN`-delimited vectors, or as cell arrays containing individual polygons in each element with the outer face separated from the subsequent inner faces by `NaN`s. Multiple polygons outputs are separated by `NaN`s.

See Also
polyshape | polysplit | polyjoin

Introduced before R2006a

polyjoin

Convert line or polygon parts from cell arrays to vector form

Syntax

```
[lat,lon] = polyjoin(latcells,loncells)
```

Description

`[lat,lon] = polyjoin(latcells,loncells)` converts polygons from cell array format to column vector format. In cell array format, each element of the cell array is a vector that defines a separate polygon.

Examples

```
latcells = {[1 2 3]'; 4; [5 6 7 8 NaN 9]'};
loncells = {[9 8 7]'; 6; [5 4 3 2 NaN 1]'};
[lat,lon] = polyjoin(latcells,loncells);
[lat lon]

ans =
      1      9
      2      8
      3      7
    NaN    NaN
      4      6
    NaN    NaN
      5      5
      6      4
      7      3
      8      2
    NaN    NaN
      9      1
```

Tips

A polygon may consist of an outer contour followed by holes separated with `NaN`s. In vector format, each vector may contain multiple faces separated by `NaN`s. There is no structural distinction between outer contours and holes in vector format.

See Also

polyshape | polysplit

Topics
"Create and Display Polygons"

Introduced before R2006a

polymerge

Merge line segments with matching endpoints

Syntax

```
[latMerged, lonMerged] = polymerge(lat, lon)
[latMerged, lonMerged] = polymerge(lat, lon, tol)
[latMerged, lonMerged] = polymerge(lat, lon, tol, outputFormat)
```

Description

`[latMerged, lonMerged] = polymerge(lat, lon)` accepts a multipart line in latitude-longitude with vertices stored in arrays `lat` and `lon`, and merges the parts wherever a pair of end points coincide. For this purpose, an end point can be either the first or last vertex in a given part. When a pair of parts are merged, they are combined into a single part and the duplicate common vertex is removed. If two first vertices coincide or two last vertices coincide, then the vertex order of one of the parts will be reversed. A merge is applied anywhere that the end points of exactly two distinct parts coincide, so that an indefinite number of parts can be chained together in a single call to `polymerge`. If three or more distinct parts share a common end point, however, the choice of which parts to merge is ambiguous and therefore none of the corresponding parts are connected at that common point.

The inputs `lat` and `lon` can be column or row vectors with NaN-separated parts (and identical NaN locations in each array), or they can be cell arrays with each part in a separate cell. The form of the output arrays, `latMerged` and `lonMerged`, matches the inputs in this regard.

`[latMerged, lonMerged] = polymerge(lat, lon, tol)` combines line segments whose endpoints are separated by less than the circular tolerance, `tol`. `tol` has the same units as the polygon input.

`[latMerged, lonMerged] = polymerge(lat, lon, tol, outputFormat)` allows you to request either the NaN-separated vector form for the output (set `outputFormat` to `'vector'`), or the cell array form (set `outputFormat` to `'cell'`).

Examples

Link Line Segments with Common Endpoints into Polygons

Construct column vectors representing coordinate values. The vectors use `NaN` separators to define four line segments.

```
lat = [3 2 NaN 1 2 NaN 5 6 NaN 3 4]';
lon = [13 12 NaN 11 12 NaN 15 16 NaN 13 14]';
```

Concatenate the segments with matching endpoints. Three of the line segments have overlapping end points, so `polymerge` returns two line segments.

```
[latm, lonm] = polymerge(lat,lon)
```

```
latm = 8×1

       1
       2
       3
       4
     NaN
       5
       6
     NaN


lonm = 8×1

      11
      12
      13
      14
     NaN
      15
      16
     NaN
```

See Also
polyjoin | polysplit

Topics
"Create and Display Polygons"

Introduced before R2006a

polysplit

Convert line or polygon parts from vector form to cell arrays

Syntax

```
[latcells,loncells] = polysplit(lat,lon)
```

Description

`[latcells,loncells] = polysplit(lat,lon)` returns the NaN-delimited segments of the vectors `lat` and `lon` as N-by-1 cell arrays with one polygon segment per cell. `lat` and `lon` must be the same size and have identically-placed NaNs. The polygon segments are column vectors if `lat` and `lon` are column vectors, and row vectors otherwise.

Examples

Extract and Join Polygons or Line Segments

Create two NaN-delimited arrays in the form of column vectors.

```
lat = [45.6 -23.47 78 NaN 43.9 -67.14 90 -89]';
lon = [13 -97.45 165 NaN 0 -114.2 -18 0]';
```

Split the column vectors into individual line segment cell arrays at the NaN separators using `polysplit`.

```
[latc,lonc] = polysplit(lat,lon)
```

```
latc=2×1 cell array
    {3x1 double}
    {4x1 double}


lonc=2×1 cell array
    {3x1 double}
    {4x1 double}
```

Inspect the contents of the cell arrays. Note that each cell array element contains a segment of the original line.

```
[latc{1} lonc{1}]
```

```
ans = 3×2

   45.6000    13.0000
  -23.4700   -97.4500
   78.0000   165.0000
```

```
[latc{2} lonc{2}]
```

```
ans = 4×2

   43.9000          0
  -67.1400  -114.2000
   90.0000   -18.0000
  -89.0000          0
```

You can reverse the process by using the `polyjoin` function.

```
[lat2,lon2] = polyjoin(latc,lonc);
```

Compare the original and joined segments. By definition, the logical comparison is false for the `NaN` delimiters.

```
[lat lon] == [lat2 lon2]
```

```
ans = 8x2 logical array

   1   1
   1   1
   1   1
   0   0
   1   1
   1   1
   1   1
   1   1
```

Test for global equality, including `NaN` values.

```
isequaln(lat,lat2) & isequaln(lon,lon2)
```

```
ans = logical
   1
```

See Also
isShapeMultipart | polyshape | polyjoin

Topics
"Create and Display Polygons"

Introduced before R2006a

polyxpoly

Intersection points for lines or polygon edges

Syntax

```
[xi,yi] = polyxpoly(x1,y1,x2,y2)
[xi,yi,ii] = polyxpoly( ___ )
[xi,yi] = polyxpoly( ___ ,'unique')
```

Description

`[xi,yi] = polyxpoly(x1,y1,x2,y2)` returns the intersection points of two polylines in a planar, Cartesian system, with vertices defined by `x1`, `y1`, `x2` and `y2`. The output arguments, `xi` and `yi`, contain the *x*- and *y*-coordinates of each point at which a segment of the first polyline intersects a segment of the second. In the case of overlapping, collinear segments, the intersection is actually a line segment rather than a point, and both endpoints are included in `xi`, `yi`.

`[xi,yi,ii] = polyxpoly( ___ )` returns a two-column array of line segment indices corresponding to the intersection points. The *k*-th row of `ii` indicates which polyline segments give rise to the intersection point `xi(k)`, `yi(k)`.

To remember how these indices work, just think of segments and vertices as fence sections and posts. The *i*-th fence section connects the *i*-th post to the (*i*+1)-th post. In general, letting *i* and *j* denote the scalar values comprised by the *k*-th row of `ii`, the intersection indicated by that row occurs where the *i*-th segment of the first polyline intersects the *j*-th segment of the second polyline. But when an intersection falls precisely on a vertex of the first polyline, then *i* is the index of that vertex. Likewise with the second polyline and the index *j*. In the case of an intersection at the *i*-th vertex of the first line, for example, `xi(k)` equals `x1(i)` and `yi(k)` equals `y1(i)`. In the case of intersections between vertices, *i* and *j* can be interpreted as follows: the segment connecting `x1(i)`, `y1(i)` to `x1(i+1)`, `y1(i+1)` intersects the segment connecting `x2(j)`, `y2(j)` to `x2(j+1)`, `y2(j+1)` at the point `xi(k)`, `yi(k)`.

`[xi,yi] = polyxpoly( ___ ,'unique')` filters out duplicate intersections, which may result if the input polylines are self-intersecting.

Examples

Find Intersection Points Between Rectangle and Polyline

Define and fill a rectangular area in the plane.

```
xlimit = [3 13];
ylimit = [2  8];
xbox = xlimit([1 1 2 2 1]);
ybox = ylimit([1 2 2 1 1]);
mapshow(xbox,ybox,'DisplayType','polygon','LineStyle','none')
```

Define and display a two-part polyline.

```
x = [0 6  4  8 8 10 14 10 14 NaN 4 4 6 9 15];
y = [4 6 10 11 7  6 10 10  6 NaN 0 3 4 3  6];
mapshow(x,y,'Marker','+')
```

Intersect the polyline with the rectangle.

```
[xi,yi] = polyxpoly(x,y,xbox,ybox);
mapshow(xi,yi,'DisplayType','point','Marker','o')
```
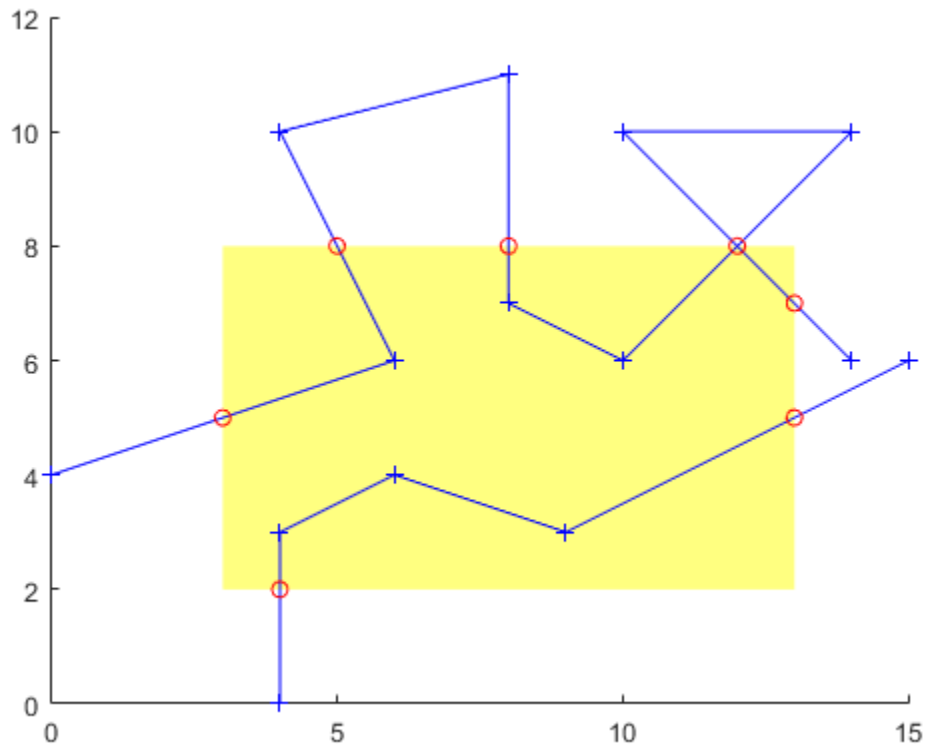

Display the intersection points; note that the point (12, 8) appears twice because of a self-intersection near the end of the first part of the polyline.

```
[xi yi]
```

ans = 8×2

```
     3.0000     5.0000
     5.0000     8.0000
     8.0000     8.0000
    12.0000     8.0000
    12.0000     8.0000
    13.0000     7.0000
    13.0000     5.0000
     4.0000     2.0000
```

You can suppress this duplicate point by using the 'unique' option.

```
[xi,yi] = polyxpoly(x,y,xbox,ybox,'unique');
[xi yi]

ans = 7×2

     3.0000     5.0000
     5.0000     8.0000
     8.0000     8.0000
    12.0000     8.0000
    13.0000     7.0000
    13.0000     5.0000
     4.0000     2.0000
```

Find Intersection Points Between State Border and Small Circle

Read state polygons into a geospatial table. Create a subtable that contains the California polygon. Display the polygon on a map.

```
states = readgeotable("usastatehi.shp");
row = states.Name == "California";
california = states(row,:);

figure
usamap("california")
geoshow(california,"FaceColor","none")
```

Define a small circle centered off the coast of California.

```
lat0 = 37;
lon0 = -122;
rad = 500;
[latc,lonc] = scircle1(lat0,lon0,km2deg(rad));
plotm(lat0,lon0,"r*")
plotm(latc,lonc,"r")
```

Extract the latitude and longitude coordinates of the California polygon from the geospatial table.

```
T = geotable2table(california,["Latitude","Longitude"]);
[lat,lon] = polyjoin(T.Latitude',T.Longitude');
```

Find the intersection points between the state of California and the small circle.

```
[loni,lati] = polyxpoly(lon,lat,lonc,latc);
plotm(lati,loni,"bo")
```
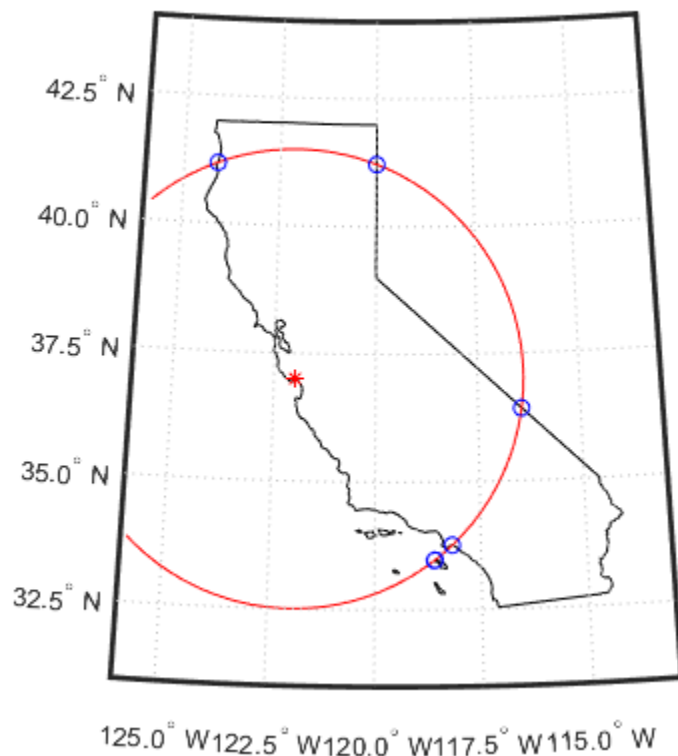
Input Arguments

x1, y1, x2, y2 — Coordinates of polylines
numeric vector

x- or *y*-coordinates of points in the first or second polyline, specified as a numeric vector. For a given polyline, the *x*- and *y*-coordinate vectors must be the same length.

Output Arguments

xi, yi — Coordinates of intersection points
numeric column vector

x- or *y*-coordinates of intersection points, specified as a numeric column vector.

ii — line segment indices
numeric vector

Line segment indices of intersection points, specified as a numeric vector.

Tips

• If the spacing between points is large, the intersections calculated by the `polyxpoly` function and the intersections shown on a map display might be different. This is a result of differences

between straight lines in the unprojected and projected coordinates. Similarly, there might be differences between the `polyxpoly` result and intersections that assume great circles or rhumb lines between points.

See Also

crossfix | gcxgc | gcxsc | navfix | rhxrh | scxsc

Introduced before R2006a

previewmap

View map at printed size

Description

The appearance of a map onscreen can differ from the final printed output. This results from the difference in the size and shape of the figure window and the area the figure occupies on the printed page. A map that appears readable on screen might be cluttered when the printed output is smaller. Likewise, the relative position of multiple axes can appear different when printed. This function resizes the figure to the printed size.

Examples

Is the text small enough to avoid overlapping in a map of Europe?

```
figure
worldmap europe
land=shaperead('landareas.shp','UseGeoCoords',true);
geoshow([land.Lat],[land.Lon])
m=gcm;
latlim = m.maplatlimit;
lonlim = m.maplonlimit;
BoundingBox = [lonlim(1) latlim(1);lonlim(2) latlim(2)];
cities=shaperead('worldcities.shp', ...
    'BoundingBox',BoundingBox,'UseGeoCoords',true);
for index=1:numel(cities)
    h=textm(cities(index).Lat, cities(index).Lon, ...
            cities(index).Name);
    trimcart(h)
    rotatetext(h)
end
orient landscape
tightmap
axis off
previewmap
```
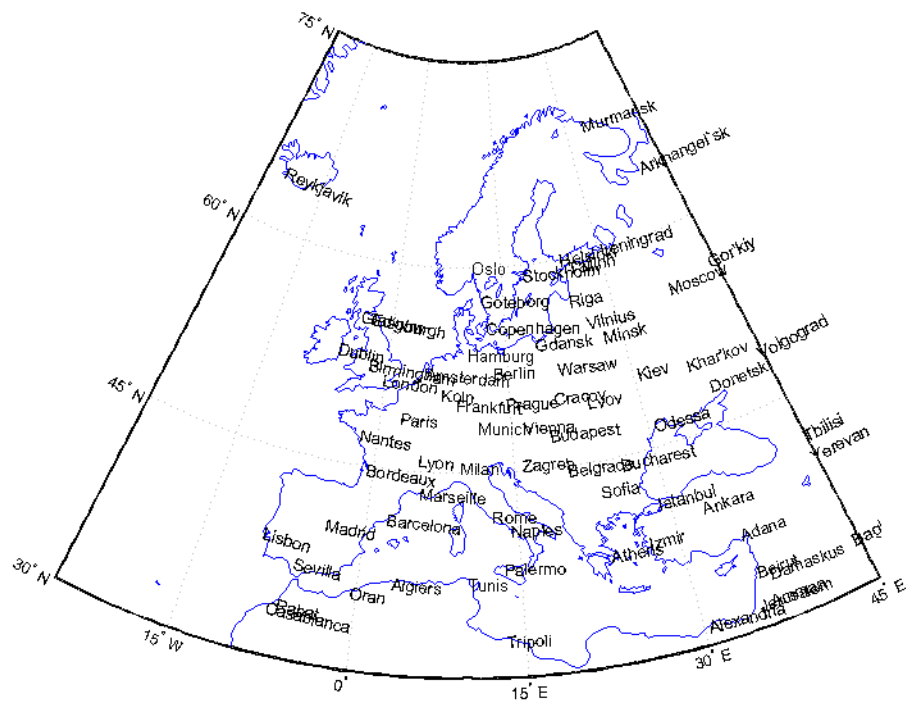
Limitations

The figure cannot be made larger than the screen.

Tips

`previewmap` changes the size of the current figure to match the printed output. If the resulting figure size exceeds the screen size, the figure is enlarged as much as possible.

See Also

`printpreview` | `paperscale` | `axesscale`

Introduced before R2006a

projcrs

Projected coordinate reference system object

Description

A projected coordinate reference system (CRS) provides information that assigns Cartesian *x* and *y* map coordinates to physical locations. Projected CRSs consist of a geographic CRS and several parameters that are used to transform coordinates to and from the geographic CRS. For more information about geographic CRSs, see `geocrs`.

Creation

There are several ways to create projected CRS objects, including:

- Import raster data by using `readgeoraster`, and then query the `ProjectedCRS` property of the returned raster reference object.
- Get information about a shapefile by using the `shapeinfo` function, and then query the `CoordinateReferenceSystem` field of the returned structure.
- Use the `projcrs` function (described here).

Syntax

```
p = projcrs(code)
p = projcrs(code,'Authority',authority)
p = projcrs(wkt)
```

Description

`p = projcrs(code)` creates a projected CRS object using the EPSG code specified by `code`.

`p = projcrs(code,'Authority',authority)` creates a projected CRS object using the specified `code` and `authority`.

`p = projcrs(wkt)` creates a projected CRS object using the well-known text (WKT) string representation specified by `wkt`.

Input Arguments

code — Projected CRS code
positive integer | string scalar | character vector

Projected CRS code, specified as a positive integer, string scalar, or character vector. By default, the `projcrs` function assumes the `code` argument is an EPSG code. To specify other types of codes, use the `'Authority'` name-value pair.

If referring to an EPSG or ESRI code, specify this argument as a positive integer. If referring to an IGNF code, specify this argument as a string scalar or character vector.

For information on valid EPSG codes, see the EPSG home page.

`authority` — Authority
`'EPSG'` (default) | `'ESRI'` | `'IGNF'`

Authority, specified as `'EPSG'`, `'ESRI'`, or `'IGNF'`. This argument specifies which authority the `projcrs` function uses to determine the properties of the created projected CRS object. If you do not specify an authority, then the `projcrs` function uses `'EPSG'`.

`wkt` — Well-known text
string scalar | character vector

Well-known text (WKT), specified as a string scalar or character vector. You can specify WKT using either the WKT 1 or WKT 2 standard.

The parameters listed in the `ProjectionParameters` property use the WKT 2 standard, even if the argument uses the WKT 1 standard.

Properties

`Name` — CRS name
string scalar

This property is read-only.

CRS name, returned as a string scalar.

Data Types: `string`

`GeographicCRS` — Geographic CRS
`geocrs` object

This property is read-only.

Geographic CRS, returned as a `geocrs` object. A geographic CRS consists of a datum (including its ellipsoid), prime meridian, and angular unit of measurement.

`LengthUnit` — Length unit
string scalar

This property is read-only.

Length unit, returned as a string scalar. Possible values include `"meter"` and `"U.S. survey foot"`.

Data Types: `string`

`ProjectionMethod` — Projection method
string scalar

This property is read-only.

Projection method, returned as a string scalar. Possible values include `"Lambert Conic Conformal (2SP)"` and `"Transverse Mercator"`.

Data Types: `string`

ProjectionParameters — Projection parameters
ProjectionParameters object

This property is read-only.

Projection parameters, returned as a `ProjectionParameters` object. The parameters in a `ProjectionParameters` object use the WKT 2 standard, even if the supplied `wkt` argument uses the WKT 1 standard.

You can query individual projection parameters using dot notation. For example, create a `projcrs` object and access the `LatitudeOfFalseOrigin` parameter.

```
p = projcrs(26986);
p.ProjectionParameters.LatitudeOfFalseOrigin
```

This table describes common projection parameters, including those used by the Lambert Conformal Conic and Transverse Mercator projection methods. Different projections may have parameters other than the ones listed here.

Parameter	Description
EastingAtFalseOrigin	Easting at the false origin, returned as a number in the units specified by `LengthUnit`. A projected CRS often uses a false origin such that all coordinates within the CRS have positive values. The easting at the false origin is with respect to the grid origin at (0, 0).
FalseEasting	False easting, returned as a number in the units specified by `LengthUnit`. A projected CRS often uses a false easting to shift the *y*-axis of the map grid so that the *x*-coordinates have positive values.
FalseNorthing	False northing, returned as a number in the units specified by `LengthUnit`. A projected CRS often uses a false northing to shift the *x*-axis of the map grid so that the *y*-coordinates have positive values.
LatitudeOf1stStandardParallel	Latitude of the first standard parallel, returned as a number. Units are typically in degrees. Standard parallels are the parallels at which the cone or cylinder used in a conic or cylindrical projection intersects the reference spheroid.
LatitudeOf2ndStandardParallel	Latitude of the second standard parallel, returned as a number. Units are typically in degrees. Standard parallels are the parallels at which the cone or cylinder used in a conic or cylindrical projection intersects the reference spheroid.
LatitudeOfFalseOrigin	Latitude of false origin, returned as a number. A projected CRS typically uses a false origin such that all coordinates within the CRS have positive values.

Parameter	Description
LatitudeOfNaturalOrigin	Latitude of the natural origin, returned as a number. Units are typically in degrees. The natural origin is the grid origin without the shift by a false northing or easting.
LongitudeOfFalseOrigin	Longitude of false origin, returned as a number. A projected CRS typically uses a false origin such that all coordinates within the CRS have positive values.
LongitudeOfNaturalOrigin	Longitude of the natural origin, returned as a number. Units are typically in degrees. The natural origin is the grid origin without the shift by a false northing or easting.
NorthingAtFalseOrigin	Northing at the false origin, returned as a number in the units specified by `LengthUnit`. A projected CRS typically uses a false origin such that all coordinates within the CRS have positive values. The northing at the false origin is with respect to the grid origin at (0, 0).
ScaleFactorAtNaturalOrigin	Scale factor at natural origin, returned as a number with no units. The natural origin is the grid origin without the shift by a false northing or easting. A projected CRS typically uses a scale factor (a number close to 1) to balance out scale distortion across the area covered by the coordinate system.

Object Functions

projfwd   Project latitude-longitude coordinates to x-y map coordinates
projinv   Unproject x-y map coordinates to latitude-longitude coordinates
wktstring   Well-known text string
isequal   Compare two projcrs or geocrs objects for equivalence

Examples

Get Projected CRS from EPSG Code

Create a projected CRS object by specifying an EPSG code.

```
p = projcrs(5325)

p =
  projcrs with properties:

                   Name: "ISN2004 / Lambert 2004"
           GeographicCRS: [1x1 geocrs]
        ProjectionMethod: "Lambert Conic Conformal (2SP)"
              LengthUnit: "meter"
    ProjectionParameters: [1x1 map.crs.ProjectionParameters]
```

Get Projected CRS from ESRI Code

Create a projected CRS object from an ESRI code by using the `'Authority'` name-value pair.

```
p = projcrs(53026,'Authority','ESRI')

p =
  projcrs with properties:

                     Name: "Sphere_Stereographic"
           GeographicCRS: [1x1 geocrs]
        ProjectionMethod: "Stereographic"
              LengthUnit: "meter"
    ProjectionParameters: [1x1 map.crs.ProjectionParameters]
```

Get Projected CRS from IGNF Code

Create a projected CRS object from an IGNF code by using the `'Authority'` name-value pair. Specify the code using a string or character vector.

```
p = projcrs('UTM39SW84','Authority','IGNF')

p =
  projcrs with properties:

                     Name: "WGS84 UTM SUD FUSEAU 39"
           GeographicCRS: [1x1 geocrs]
        ProjectionMethod: "Transverse Mercator"
              LengthUnit: "meter"
    ProjectionParameters: [1x1 map.crs.ProjectionParameters]
```

Get Projected CRS from Projection File

Import a WKT projection file as a character vector by using the `fileread` function. Then create a projected CRS object by specifying the vector.

```
wkt = fileread('MtWashington-ft.prj');
p = projcrs(wkt)

p =
  projcrs with properties:

                     Name: "UTM Zone 19, Northern Hemisphere"
           GeographicCRS: [1x1 geocrs]
        ProjectionMethod: "Transverse Mercator"
              LengthUnit: "meter"
    ProjectionParameters: [1x1 map.crs.ProjectionParameters]
```

Get Projected CRS from Imported Raster Data

Import raster data as an array and a map reference object using the `readgeoraster` function. Then, get the projected CRS by querying the `ProjectedCRS` property of the reference object.

```
[Z,R] = readgeoraster('boston.tif');
R.ProjectedCRS
```

```
ans =
  projcrs with properties:

                     Name: "NAD83 / Massachusetts Mainland"
            GeographicCRS: [1×1 geocrs]
         ProjectionMethod: "Lambert Conic Conformal (2SP)"
               LengthUnit: "U.S. survey foot"
     ProjectionParameters: [1×1 map.crs.ProjectionParameters]
```

Alternatively, return information about the same file as a `RasterInfo` object using the `georasterinfo` function. Then, get the projected CRS by querying the `CoordinateReferenceSystem` property of the object.

```
info = georasterinfo('boston.tif');
info.CoordinateReferenceSystem
```

```
ans =
  projcrs with properties:

                     Name: "NAD83 / Massachusetts Mainland"
            GeographicCRS: [1×1 geocrs]
         ProjectionMethod: "Lambert Conic Conformal (2SP)"
               LengthUnit: "U.S. survey foot"
     ProjectionParameters: [1×1 map.crs.ProjectionParameters]
```

Get Projection Parameters for Projected CRS

Get projection parameters for a projected CRS by creating a `projcrs` object and querying its `ProjectionParameters` property.

```
p = projcrs(26986);
parameters = p.ProjectionParameters
```

```
parameters =
  ProjectionParameters object with parameters:

            LatitudeOfFalseOrigin: 41
           LongitudeOfFalseOrigin: -71.5
    LatitudeOf1stStandardParallel: 42.6833333333333
    LatitudeOf2ndStandardParallel: 41.7166666666667
             EastingAtFalseOrigin: 200000
            NorthingAtFalseOrigin: 750000
```

Query individual projection parameters by using dot notation.

```
f = parameters.EastingAtFalseOrigin
```

```
f = 200000
```

Tips

When you compare two `ProjectionParameters` objects by using the `isequal` function, the function might return `0` (`false`), even when the parameter values are the same. Instead, compare `ProjectionParameters` objects by comparing the `projcrs` objects or by comparing the parameter values directly.

See Also

Functions
`fileread` | `projfwd` | `projinv`

Objects
`geocrs`

External Websites
EPSG home page

Introduced in R2020b

project

Project displayed map graphics object

Note `project` will be removed in a future release.

Syntax

```
project(h)
project(h,'xy')
project(h,'yx')
```

Description

`project(h)` takes unprojected objects with handles `h` that are displayed on map axes and projects them. For example, `project` takes a line created on a map axes with the `plot` function and projects it as though it had been created with the `plotm` function. This can be useful if a standard MATLAB function was accidentally executed. The map structure of the existing map axes determines the specifics of the projection. If `h` is the handle of the map axes, then all the children of `h` are projected. Do not attempt this if any children of `h` have already been projected!

`project(h,'xy')` specifies that the `XData` of the unprojected objects corresponds to longitudes and the `YData` to latitudes. This is the default assumption.

`project(h,'yx')` specifies that the `XData` of the unprojected objects corresponds to latitudes and the `YData` to longitudes.

Examples

Create an axes, plot a line, then project it:

```
axesm('bonne','AngleUnits','radians');framem;
h = plot([-1 -.5 0 .5 1],[-1 -.5  0 .5 1]);
```
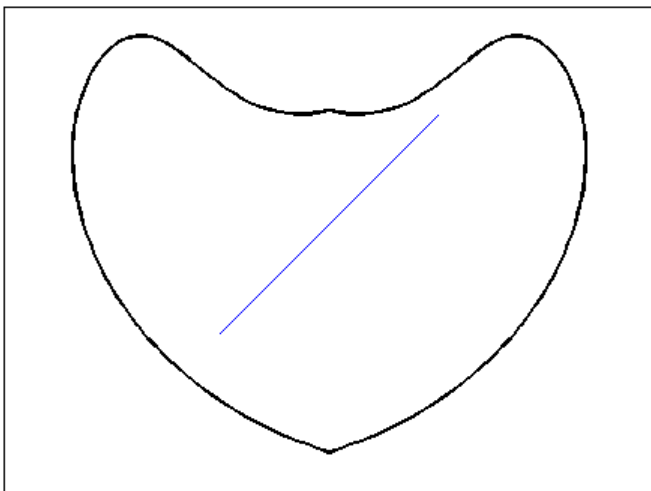
```
project(h)
```
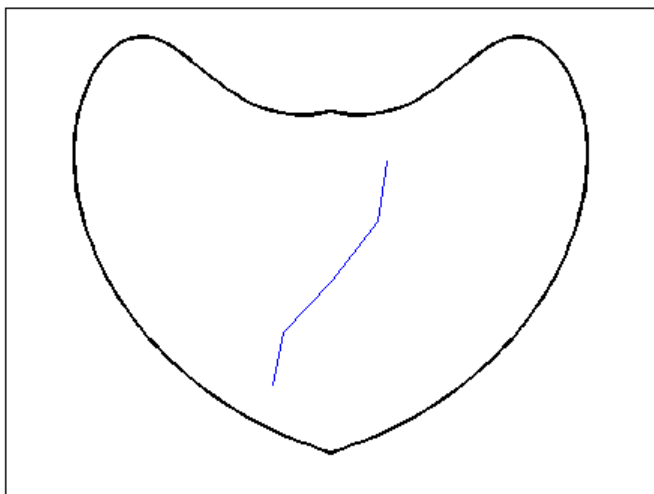

The line is straight in *x-y* space, but when converted to a projected map object, it bends with the projection.

See Also

Introduced before R2006a

projfwd

Project latitude-longitude coordinates to *x-y* map coordinates

Syntax

```
[x,y] = projfwd(proj,lat,lon)
```

Description

`[x,y] = projfwd(proj,lat,lon)` transforms the latitude-longitude coordinates specified by `lat` and `lon` to the `x` and `y` map coordinates in the projected coordinate reference system specified by `proj`. Specify `proj` using a `projcrs` object *(since R2020b)*, a map projection structure, or a GeoTIFF information structure.

Examples

Project Latitude-Longitude Coordinates to *x-y*

Project latitude-longitude coordinates to *x-y* coordinates by specifying a map projection. Then, display the projected coordinates on a map.

To do this, first specify the latitude and longitude coordinates of landmarks in Boston. Specify the coordinates in the NAD83 geographic CRS.

```
lat = [42.3604 42.3691 42.3469 42.3480 42.3612];
lon = [-71.0580 -71.0710 -71.0623 -71.0968 -71.0941];
```

Then, import a GeoTIFF image of Boston as an array and a map cells reference object. Get information about the map projection by querying the `ProjectedCRS` property of the reference object. Verify that the geographic CRS underlying the projected CRS is NAD83.

```
[A,R] = readgeoraster('boston.tif');
proj = R.ProjectedCRS;
proj.GeographicCRS.Name

ans =
"NAD83"
```

Project the latitude-longitude coordinates to *x-y* coordinates using the same projected CRS as the GeoTIFF image.

```
[x,y] = projfwd(proj,lat,lon);
```

Display the GeoTIFF image and the projected coordinates on the same map. Change the marker symbol and color of the coordinates so they are more visible. Then, add axis labels.

```
mapshow(A,R)
mapshow(x,y,'DisplayType','point','Marker','o', ...
    'MarkerFaceColor','y','MarkerEdgeColor','none')
xlabel('x (survey feet)')
ylabel('y (survey feet)')
```
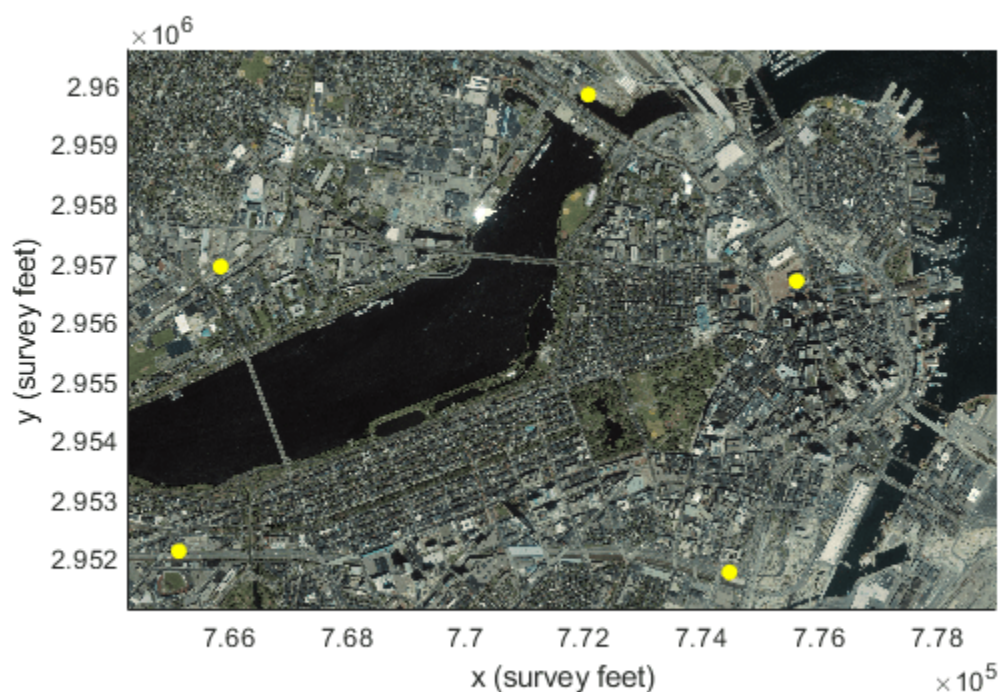
Input Arguments

`proj` — Map projection
`projcrs` object | scalar map projection structure | scalar GeoTIFF information structure

Map projection, specified as a `projcrs` object *(since R2020b)*, a scalar map projection structure (`mstruct`), or a GeoTIFF information structure. For more information about map projection structures, see `defaultm`. For more information about GeoTIFF information structures, see `geotiffinfo`.

Data Types: `struct`

`lat` — Geodetic latitudes
scalar value | vector | matrix | N-D array

Geodetic latitudes, specified as a scalar value, vector, matrix, or N-D array, in units of degrees. The size of `lat` and `lon` must match.

Data Types: `single` | `double`

`lon` — Geodetic longitudes
scalar value | vector | matrix | N-D array

Geodetic longitudes, specified as a scalar value, vector, matrix, or N-D array, in units of degrees. The size of `lat` and `lon` must match.

Data Types: `single` | `double`

Output Arguments

x — Projected *x*-coordinates
scalar value | vector | matrix | N-D array

Projected *x*-coordinates, returned as a scalar value, vector, matrix, or N-D array.

y — Projected *y*-coordinates
scalar value | vector | matrix | N-D array

Projected *y*-coordinates, returned as a scalar value, vector, matrix, or N-D array.

Tips

If the geographic CRS of `lat` and `lon` does not match the geographic CRS of `proj`, then the values of `x` and `y` may be inaccurate. When `proj` is a `projcrs` object, you can find its geographic CRS by querying its `GeographicCRS` property. For example, this code shows how to create a `projcrs` object from EPSG code 32610 and find the associated geographic CRS.

```
proj = projcrs(32610);
proj.GeographicCRS.Name
```

```
ans =

    "WGS 84"
```

See Also

Functions
`geotiffinfo` | `projinv` | `projlist`

Objects
`projcrs`

Topics
"Project and Display Raster Data"
"Transform Coordinates to a Different Projected CRS"

Introduced before R2006a

projinv

Unproject *x-y* map coordinates to latitude-longitude coordinates

Syntax

```
[lat,lon] = projinv(proj,x,y)
```

Description

`[lat,lon] = projinv(proj,x,y)` transforms the map coordinates specified by `x` and `y` in the projected coordinate reference system specified by `proj` to the latitude-longitude coordinates `lat` and `lon`. Specify `proj` using a `projcrs` object *(since R2020b)*, a map projection structure, or a GeoTIFF information structure.

Examples

Unproject *x-y* Coordinates to Latitude-Longitude

Unproject *x-y* coordinates to latitude-longitude coordinates by specifying the projected CRS of the *x-y* coordinates. Then, display the latitude-longitude coordinates on geographic axes.

To do this, first import a shapefile containing the *x*- and *y*-coordinates of roads in Concord, MA. Get information about the shapefile as a structure. Find the projected CRS for the coordinates by accessing the `CoordinateReferenceSystem` field of the structure.

```
roads = shaperead('concord_roads.shp');
x = [roads.X];
y = [roads.Y];
info = shapeinfo('concord_roads.shp');
proj = info.CoordinateReferenceSystem;
```

Unproject the *x-y* coordinates to latitude-longitude coordinates.

```
[lat,lon] = projinv(proj,x,y);
```

Display the coordinates on geographic axes.

```
figure
geoplot(lat,lon)
hold on
geobasemap('streets')
```
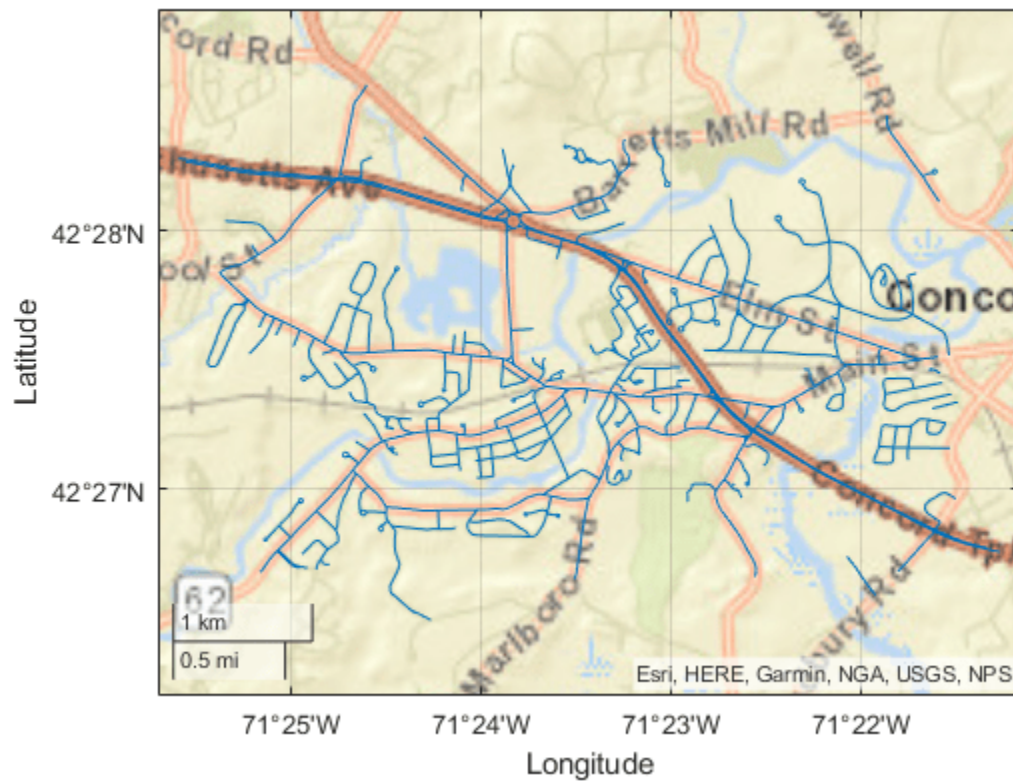
The geographic CRS of the *x-y* coordinates used in this example is NAD83. You can find the geographic CRS that underlies a projected CRS by querying the `GeographicCRS` property.

```
proj.GeographicCRS.Name
```

```
ans =
"NAD83"
```

The geographic CRS underlying the `'streets'` basemap is WGS84. NAD83 and WGS84 are similar, but not identical. Therefore, at high zoom levels the coordinates and basemap may appear misaligned.

Input Arguments

`proj` — Map projection
`projcrs` object | scalar map projection structure | scalar GeoTIFF information structure

Map projection, specified as a `projcrs` object *(since R2020b)*, a scalar map projection structure (`mstruct`), or a GeoTIFF information structure. For more information about map projection structures, see `defaultm`. For more information about GeoTIFF information structures, see `geotiffinfo`.

Data Types: `struct`

`x` — Projected *x*-coordinates
scalar value | vector | matrix | N-D array

Projected *x*-coordinates, specified as a scalar value, vector, matrix, or N-D array. The size of `x` and `y` must match.

Data Types: `single` | `double`

y — Projected *y*-coordinates
scalar value | vector | matrix | N-D array

Projected *y*-coordinates, specified as a scalar value, vector, matrix, or N-D array. The size of `x` and `y` must match.

Data Types: `single` | `double`

Output Arguments

`lat` — Geodetic latitudes
scalar value | vector | matrix | N-D array

Geodetic latitudes, returned as a scalar value, vector, matrix, or N-D array, in units of degrees.

The geographic CRS of `lat` matches the geographic CRS of `proj`. If `proj` is a `projcrs` object, then you can find its geographic CRS by querying its `GeographicCRS` property. For example, this code shows how to create a `projcrs` object from EPSG code 32610 and find the associated geographic CRS.

```
proj = projcrs(32610);
proj.GeographicCRS.Name
```

```
ans =

    "WGS 84"
```

`lon` — Geodetic longitudes
scalar value | vector | matrix | N-D array

Geodetic longitudes, returned as a scalar value, vector, matrix, or N-D array, in units of degrees.

The geographic CRS of `lat` matches the geographic CRS of `proj`. If `proj` is a `projcrs` object, then you can find its geographic CRS by querying its `GeographicCRS` property. For example, this code shows how to create a `projcrs` object from EPSG code 32610 and find the associated geographic CRS.

```
proj = projcrs(32610);
proj.GeographicCRS.Name
```

```
ans =

    "WGS 84"
```

See Also

Functions
`geotiffinfo` | `projfwd` | `projlist`

Objects
`projcrs`

Topics
"Project and Display Raster Data"
"Transform Coordinates to a Different Projected CRS"

Introduced before R2006a

projlist

GeoTIFF info structure support for `projfwd` and `projinv`

Syntax

```
projlist(listmode)
S = projlist(listmode)
```

Description

`projlist(listmode)` displays a table of projection names, IDs, and availability. `listmode` can be `'mapprojection'`, `'geotiff'`, `'geotiff2mstruct'`, or `'all'`. The default value is `'mapprojection'`.

`S = projlist(listmode)` returns a structure array containing projection names, IDs, and availability. The output of `projlist` for each `listmode` is described below:

*   `mapprojection` — Lists the projection IDs that `projfwd` and `projinv` use the PROJ library to implement. Starting in R2020b, you can use any valid map projection ID with `projfwd` and `projinv`. Return a list of valid map projection IDs using the `maplist` function. The output structure contains these fields.

    *   `Name` — Projection name
    *   `MapProjection` — Projection ID

*   `geotiff` — Lists the GeoTIFF projection IDs that are available for use with `projfwd` and `projinv`. The output structure contains these fields.

    *   `GeoTIFF` — GeoTIFF projection ID
    *   `Available`— Logical array with values 1 or 0

*   `geotiff2mstruct` — Lists the GeoTIFF projection IDs that are available for use with `geotiff2mstruct`. The output structure contains these fields.

    *   `GeoTIFF` — GeoTIFF projection ID
    *   `MapProjection` — Projection ID

*   `all` — Lists the map and GeoTIFF projection IDs that are available for use with `projfwd` and `projinv`. The output structure contains these fields.

    *   `GeoTIFF` — GeoTIFF projection ID
    *   `MapProjection` — Projection ID
    *   `info` — Logical array with values 1 or 0
    *   `mstruct` — Logical array with values 1 or 0

Examples

```
s=projlist

s =
```

```
1x19 struct array with fields:
    Name
    MapProjection

s=projlist('geotiff2mstruct')

s =
1x19 struct array with fields:
    GeoTIFF
    MapProjection
```

See Also
geotiff2mstruct | projfwd | projinv | maplist | maps

Introduced before R2006a

properties

Return property names of geographic or planar vector

Syntax

```
prop = properties(v)
```

Description

`prop = properties(v)` returns the property names of the geographic or planar vector `v`.

Examples

View All Properties of a Mapshape Vector

Create a mapshape vector.

```
ms = mapshape(shaperead('tsunamis.shp', 'UseGeoCoords', true));
```

Display all properties of the mapshape vector. This includes the `Geometry` and `Metadata` collection properties, the `X` and `Y` required mapshape Vertex properties, and all dynamic properties.

```
properties(ms)
```

```
Properties for class mapshape:

    Geometry
    Metadata
    X
    Y
    Lon
    Lat
    Year
    Month
    Day
    Hour
    Minute
    Second
    Val_Code
    Validity
    Cause_Code
    Cause
    Eq_Mag
    Country
    Location
    Max_Height
    Iida_Mag
    Intensity
    Num_Deaths
    Desc_Deaths
```

Input Arguments

v — Geographic or planar vector
geopoint, geoshape, mappoint, or mapshape object

Geographic or planar vector, specified as a geopoint, geoshape, mappoint, or mapshape object.

Output Arguments

prop — Property names
cell array

Property names of geographic or planar vector v, returned as a cell array.

See Also
disp | fieldnames

Introduced in R2012a

putpole

Origin vector to place north pole at specified point

Syntax

```
origin = putpole(pole)
origin = putpole(pole,units)
```

Description

`origin = putpole(pole)` returns an `origin` vector required to transform a coordinate system in such a way as to put the true North Pole at a point specified by the three- (or two-) element vector `pole`. This vector is of the form `[latitude longitude meridian]`, specifying the coordinates in the original system at which the true North Pole is to be placed in the transformed system. The meridian is the longitude upon which the new system is to be centered, which is the new pole longitude if omitted. The output is a three-element vector of the form `[latitude longitude orientation]`, where the latitude and longitude are the coordinates in the untransformed system of the new origin, and the orientation is the azimuth of the true North Pole in the transformed system.

`origin = putpole(pole,units)` allows the specification of the angular units of the `origin` vector, where `units` is any valid angle unit. The default is `'degrees'`.

Examples

Pull the North Pole down the 0º meridian by 30º to 60ºN. What is the resulting origin vector?

```
origin = putpole([60 0])

origin =
   30.0000        0        0
```

This makes sense: when the pole slid down 30º, the point that was 30º north of the origin slid down to become the origin. Following is a less obvious transformation:

```
origin = putpole([60 80 0])   % constrain to original central
                              % meridian

origin =
    4.9809        0   29.6217

origin = putpole([60 80 40])  % constrain to arbitrary meridian

origin =
    4.9809   40.0000   29.6217
```

Tips

When developing transverse or oblique projections, you need transformed coordinate systems. One way to define these systems is to establish the point in the original (untransformed) system that will become the new (transformed) origin.

See Also

neworig | org2pol

Introduced before R2006a

quiver3m

Project 3-D quiver plot on map axes

Syntax

```
h = quiver3m(lat,lon,alt,dlat,dlon,dalt)
h = quiver3m(lat,lon,alt,dlat,dlon,dalt,LineSpec)
h = quiver3m(lat,lon,alt,dlat,dlon,dalt,LineSpec,'filled')
h = quiver3m(lat,lon,alt,dlat,dlon,dalt,scale)
h = quiver3m(lat,lon,alt,dlat,dlon,dalt,LineSpec,scale)
h = quiver3m(lat,lon,alt,dlat,dlon,dalt,LineSpec,scale,'filled')
```

Description

`h = quiver3m(lat,lon,alt,dlat,dlon,dalt)` displays *velocity* vectors with components `(dlat,dlon,dalt)` at the geographic points `(lat,lon)` and altitude `alt` on a displayed map axes. The inputs `dlat`, `dlon`, and `dalt` determine the direction of the vectors in latitude, longitude, and altitude, respectively. The function automatically determines the length of these vectors to make them as long as possible without overlap. The object handles of the displayed vectors can be returned in `h`.

`h = quiver3m(lat,lon,alt,dlat,dlon,dalt,LineSpec)` specifies the line style, marker, and color on page 1-1148. If you use markers, they are plotted at the start points of the vectors, i.e., the input points `(lat,lon,alt)`.

`h = quiver3m(lat,lon,alt,dlat,dlon,dalt,LineSpec,'filled')` results in the filling in of any symbols specified by `LineSpec`.

`h = quiver3m(lat,lon,alt,dlat,dlon,dalt,scale)`, `h = quiver3m(lat,lon,alt,dlat,dlon,dalt,LineSpec,scale)` and `h = quiver3m(lat,lon,alt,dlat,dlon,dalt,LineSpec,scale,'filled')` alter the automatically calculated vector lengths by multiplying them by the scalar value `scale`. For example, if `scale` is 2, the displayed vectors are twice as long as they would be if `scale` were 1 (the default). When scale is set to 0, the automatic scaling is suppressed and the length of the vectors is determined by the inputs. In this case, the vectors are plotted from `(lat,lon,alt)` to `(lat+dlat,lon+dlon,alt+dalt)`.

Examples

Plot 3-D Quiver Vectors

Plot 3-D quiver vectors from London (51.5°N,0°) and New Delhi (29°N,77.5°E), both at an altitude of 0. Suppress the automatic scaling. Terminate both vectors at an altitude of 1; the London vector should terminate 100° southward and 70° eastward, while the New Delhi vector should terminate 50° northward and 10° eastward.

```
load coastlines
axesm miller;
view(3)
plotm(coastlat,coastlon)
lat0 = [51.5,29];
```

```
lon0 = [0 77.5];
alt = [0 0];
dlat = [-40 50];
dlon = [-70 10];
dalt = [1 1];
quiver3m(lat0,lon0,alt,dlat,dlon,dalt,'m')
tightmap
```
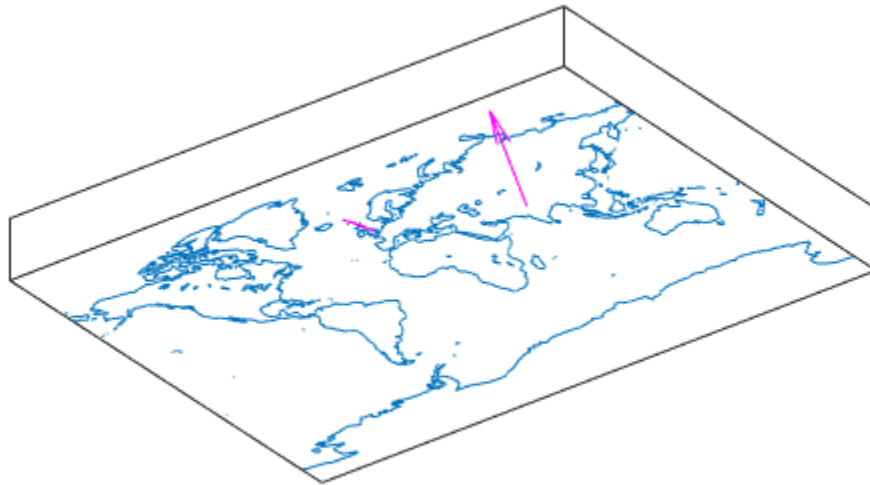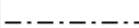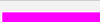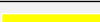

More About

Line Style, Marker, and Color

Specify the line style, marker, and color as a character vector or string containing symbols. The symbols can appear in any order. You do not need to specify all three characteristics (line style, marker, and color). For example, specify a red dashed line using `'--r'`.

Line Style	Description	Resulting Line
`'-'`	Solid line	
`'--'`	Dashed line	
`':'`	Dotted line	

Line Style	Description	Resulting Line
'-.'	Dash-dotted line	—.—.—.—..

Marker	Description	Resulting Marker
'o'	Circle	
'+'	Plus sign	
'*'	Asterisk	
'.'	Point	
'x'	Cross	
'_'	Horizontal line	
'|'	Vertical line	
's'	Square	
'd'	Diamond	
'^'	Upward-pointing triangle	
'v'	Downward-pointing triangle	
'>'	Right-pointing triangle	
'<'	Left-pointing triangle	
'p'	Pentagram	
'h'	Hexagram	

Color Name	Short Name	RGB Triplet	Appearance
'red'	'r'	[1 0 0]	
'green'	'g'	[0 1 0]	
'blue'	'b'	[0 0 1]	
'cyan'	'c'	[0 1 1]	
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

See Also
quiverm | quiver3

Introduced before R2006a

quiverm

Project 2-D quiver plot on map axes

Syntax

```
h = quiverm(lat,lon,deltalat,deltalon)
h = quiverm(lat,lon,deltalat,deltalon,LineSpec)
h = quiverm(lat,lon,deltalat,deltalon,LineSpec,'filled')
h = quiverm(lat,lon,deltalat,deltalon,scale)
h = quiverm(lat,lon,deltalat,deltalon,LineSpec,scale,'filled')
```

Description

`h = quiverm(lat,lon,deltalat,deltalon)` displays *velocity* vectors with components (`deltalat`,`deltalon`) at the geographic points (`lat`,`lon`) on displayed map axes. All four inputs should be in the `AngleUnits` of the map axes. The inputs `deltalat` and `deltalon` determine the direction of the vectors in latitude and longitude, respectively. The function automatically determines the length of these vectors to make them as long as possible without overlap. The object handles of the displayed vectors can be returned in `h`.

`h = quiverm(lat,lon,deltalat,deltalon,LineSpec)` specifies the line style, marker, and color on page 1-1153. If you use markers, they are plotted at the start points of the vectors, i.e., the input points (`lat`,`lon`).

`h = quiverm(lat,lon,deltalat,deltalon,LineSpec,'filled')` results in the filling in of any symbols specified by `LineSpec`.

`h = quiverm(lat,lon,deltalat,deltalon,scale)` and `h = quiverm(lat,lon,deltalat,deltalon,LineSpec,scale,'filled')` alter the automatically calculated vector lengths by multiplying them by the scalar value `scale`. For example, if `scale` is 2, the displayed vectors are twice as long as they would be if `scale` were 1 (the default). When scale is set to 0, the automatic scaling is suppressed, and the length of the vectors is determined by the inputs. In this case, the vectors are plotted from (`lat`,`lon`) to (`lat+deltalat`,`lon+deltalon`).

Examples

Plot Quiver Vectors Corresponding to Latitude and Longitude

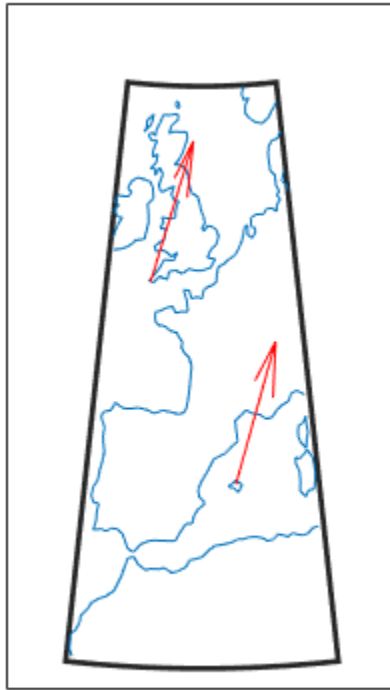Load the coast lines dataset.

```
load coastlines
```

Setup an axes specifying latitude and longitude limits. Plot the coast line data.

```
axesm('eqaconic','MapLatLimit',[30 60],'MapLonLimit',[-10 10])
framem;
plotm(coastlat,coastlon)
```
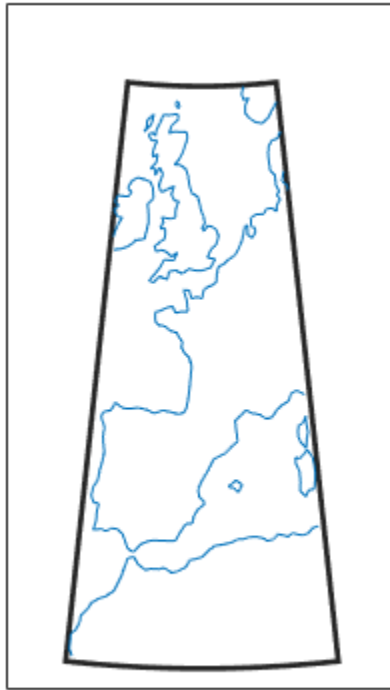
Plot quiver vectors in a direction corresponding to +5 degrees latitude and +3 degrees longitude. Use automatic scaling.

```
lat0 = [50 39.7];
lon0 = [-5.4 2.9];
deltalat = [5 5];
deltalon = [3 3];
quiverm(lat0,lon0,deltalat,deltalon,'r')
```
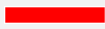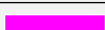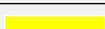
More About

Line Style, Marker, and Color

Specify the line style, marker, and color as a character vector or string containing symbols. The symbols can appear in any order. You do not need to specify all three characteristics (line style, marker, and color). For example, specify a red dashed line using '--r'.

Line Style	Description	Resulting Line
'-'	Solid line	————
'--'	Dashed line	– – – –
':'	Dotted line	....
'-.'	Dash-dotted line	—·—·—·—

Marker	Description	Resulting Marker
'o'	Circle	○
'+'	Plus sign	+

Marker	Description	Resulting Marker
'*'	Asterisk	✳
'.'	Point	•
'x'	Cross	✕
'_'	Horizontal line	—
'|'	Vertical line	|
's'	Square	▢
'd'	Diamond	◇
'^'	Upward-pointing triangle	△
'v'	Downward-pointing triangle	▽
'>'	Right-pointing triangle	▷
'<'	Left-pointing triangle	◁
'p'	Pentagram	☆
'h'	Hexagram	✡

Color Name	Short Name	RGB Triplet	Appearance
'red'	'r'	[1 0 0]	
'green'	'g'	[0 1 0]	
'blue'	'b'	[0 0 1]	
'cyan'	'c'	[0 1 1]	
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

See Also

quiver3m | quiver

Introduced before R2006a

rad2km

Convert spherical distance from radians to kilometers

Syntax

```
km = rad2km(rad)
km = rad2km(rad,radius)
km = rad2km(rad,sphere)
```

Description

`km = rad2km(rad)` converts distances from radians to kilometers, as measured along a great circle on a sphere with a radius of 6371 km, the mean radius of the Earth.

`km = rad2km(rad,radius)` converts distances from radians to kilometers, as measured along a great circle on a sphere having the specified radius.

`km = rad2km(rad,sphere)` converts distances from radians to kilometers, as measured along a great circle on a sphere approximating an object in the Solar System.

Examples

Convert Radians to Kilometers on Earth

Find the circumference of the Earth in kilometers. By default, the `rad2km` function assumes your input is a measurement of the Earth, which is represented as sphere with a radius of 6371 kilometers.

```
km = rad2km(2*pi)
```

```
km = 4.0030e+04
```

Convert Radians to Kilometers on Mars

Find the circumference of Mars in kilometers.

```
km = rad2km(2*pi,'mars')
```

```
km = 2.1300e+04
```

Input Arguments

rad — Distance in radians
numeric array

Distance in radians, specified as a numeric array.

Data Types: `single` | `double`

radius — Radius
6371 (default) | numeric scalar

Radius of sphere in units of kilometers, specified as a numeric scalar.

sphere — Sphere
'sun' | 'moon' | 'mercury' | 'venus' | 'earth' | …

Sphere approximating an object in the Solar System, specified as one of the following values: 'sun', 'moon', 'mercury', 'venus', 'earth', 'mars', 'jupiter', 'saturn', 'uranus', 'neptune', or 'pluto'. The value of sphere is case-insensitive.

Output Arguments

km — Distance in kilometers
numeric array

Distance in kilometers, returned as a numeric array.

Data Types: single | double

See Also
deg2rad | rad2deg | rad2nm | rad2sm | deg2km | km2rad

Introduced in R2007a

rad2nm

Convert spherical distance from radians to nautical miles

Syntax

```
nm = rad2nm(rad)
nm = rad2nm(rad,radius)
nm = rad2nm(rad,sphere)
```

Description

`nm = rad2nm(rad)` converts distances from radians to nautical miles, as measured along a great circle on a sphere with a radius of 3440.065 nm, the mean radius of the Earth.

`nm = rad2nm(rad,radius)` converts distances from radians to nautical miles, as measured along a great circle on a sphere having the specified radius.

`nm = rad2nm(rad,sphere)` converts distances from radians to nautical miles, as measured along a great circle on a sphere approximating an object in the Solar System.

Examples

Convert Radians to Nautical Miles on Earth

Find the circumference of the Earth in nautical miles. By default, the `rad2nm` function assumes your input is a measurement of the Earth, which is represented as a sphere with a radius of 3440.065 nautical miles.

```
nm = rad2nm(2*pi)
```

```
nm = 2.1615e+04
```

Convert Radians to Nautical Miles on Sphere with Specified Radius

Find the circumference of a sphere with a radius of 3432 nautical miles.

```
nm = rad2nm(2*pi,3432)
```

```
nm = 2.1564e+04
```

Input Arguments

rad — Distance in radians
numeric array

Distance in radians, specified as a numeric array.

Data Types: `single` | `double`

radius — Radius
3440.065 (default) | numeric scalar

Radius of sphere in units of nautical miles, specified as a numeric scalar.

sphere — Sphere
`'sun'` | `'moon'` | `'mercury'` | `'venus'` | `'earth'` | ...

Sphere approximating an object in the Solar System, specified as one of the following values: `'sun'`, `'moon'`, `'mercury'`, `'venus'`, `'earth'`, `'mars'`, `'jupiter'`, `'saturn'`, `'uranus'`, `'neptune'`, or `'pluto'`. The value of `sphere` is case-insensitive.

Output Arguments

nm — Distance in nautical miles
numeric array

Distance in nautical miles, returned as a numeric array.

Data Types: `single` | `double`

See Also
deg2rad | rad2deg | rad2sm | rad2km | nm2rad | deg2nm

Introduced in R2007a

rad2sm

Convert spherical distance from radians to statute miles

Syntax

```
sm = rad2sm(rad)
sm = rad2sm(rad,radius)
sm = rad2sm(rad,sphere)
```

Description

`sm = rad2sm(rad)` converts distances from radians to statute miles, as measured along a great circle on a sphere with a radius of 3958.748 sm, the mean radius of the Earth.

`sm = rad2sm(rad,radius)` converts distances from radians to statute miles, as measured along a great circle on a sphere having the specified radius.

`sm = rad2sm(rad,sphere)` converts distances from radians to statute miles, as measured along a great circle on a sphere approximating an object in the Solar System.

Examples

Convert Arc Length to Statute Miles

How long is a trip around the equator in statute miles?

```
sm = rad2sm(2*pi)
```

```
sm = 2.4874e+04
```

How about on Jupiter?

```
sm = rad2sm(2*pi,'jupiter')
```

```
sm = 2.7283e+05
```

Input Arguments

rad — Distance in radians
numeric array

Distance in radians, specified as a numeric array.

Data Types: `single` | `double`

radius — Radius
3958.748 (default) | numeric scalar

Radius of sphere in units of statute miles, specified as a numeric scalar.

sphere — Sphere
'sun' | 'moon' | 'mercury' | 'venus' | 'earth' | …

Sphere approximating an object in the Solar System, specified as one of the following values: 'sun', 'moon', 'mercury', 'venus', 'earth', 'mars', 'jupiter', 'saturn', 'uranus', 'neptune', or 'pluto'. The value of sphere is case-insensitive.

Output Arguments

sm — Distance in statute miles
numeric array

Distance in statute miles, returned as a numeric array.

Data Types: single | double

See Also
deg2rad | rad2deg | rad2km | deg2sm | rad2nm | sm2rad

Introduced in R2007a

radtodeg

(Not recommended) Convert angles from radians to degrees

> **Note** `radtodeg` is not recommended. Use the `rad2deg` function instead. For more information, see "Compatibility Considerations".

Syntax

```
angleInDegrees = radtodeg(angleInRadians)
```

Description

`angleInDegrees = radtodeg(angleInRadians)` converts angle units from radians to degrees.

Examples

Convert Radians to Degrees

Convert `pi` radians to degrees.

```
angleInDegrees = radtodeg(pi)

angleInDegrees =

   180
```

Input Arguments

angleInRadians — Angle in radians
scalar | vector | matrix | N-D array

Angle in radians, specified as a scalar, vector, matrix, or N-D array.

Compatibility Considerations

radtodeg is not recommended
Not recommended starting in R2015b

The `radtodeg` function is not recommended. Use the `rad2deg` function instead.

To update your code, replace instances of the `radtodeg` function with the `rad2deg` function.

While the `radtodeg` function accepts input arguments of integer types such as `uint8`, the `rad2deg` function accepts only single- and double-precision types. You can convert integer types to single- or double-precision types by using the `single` or `double` function.

See Also

rad2deg

Introduced in R2009b

RasterInfo

Information about geospatial raster data file

Description

`RasterInfo` objects contain information about geographic or projected raster data files, such as their file format, their native data type, and how they represent missing data.

Creation

Create a `RasterInfo` object using `georasterinfo`.

Properties

Filename — Path to data file
`string` array

This property is read-only.

Path to data file and any supporting files, returned as a `string` array.

Data Types: `string`

FileModifiedDate — Modification date of files
`datetime` array

This property is read-only.

Modification date of the data file and any supporting files, returned as a `datetime` array. The order of modification dates in `FileModifiedDate` corresponds to the order of files in `Filename`.

Data Types: `datetime`

FileSize — File size
`array`

This property is read-only.

File size of the data file and any supporting files, returned as an array. The order of file sizes in `FileSize` corresponds to the order of files in `Filename`.

Data Types: `double`

FileFormat — File format name
`string` scalar

This property is read-only.

File format name, returned as a `string` scalar. For a list of supported file formats, see `georasterinfo`.

Data Types: `string`

RasterSize — Dimensions of raster data
two-element vector

This property is read-only.

Dimensions of the raster data embedded in the file, returned as a two-element vector, `[m n]`, where `m` is the number of rows and `n` is the number of columns.

Data Types: `double`

NumBands — Number of bands
positive integer

This property is read-only.

Number of bands, returned as a positive integer.

When you read raster data using the `readgeoraster` function, the array it returns is of size *M*-by-*N*-by-*P*, where *P* is the value of `NumBands`.

Data Types: `double`

NativeFormat — Data type embedded in file
`string` scalar

This property is read-only.

Data type of the data embedded in the file, returned as a `string` scalar. To read data using a different data type, call `readgeoraster` and specify the `'OutputType'` name-value pair.

Data Types: `string`

MissingDataIndicator — Value indicating missing data
integer

This property is read-only.

Value indicating missing data, returned as an integer. You can replace missing data with `NaN` values using the `standardizeMissing` function.

```
[A,R] = readgeoraster('MtWashington-ft.grd');
info = georasterinfo('MtWashington-ft.grd');
m = info.MissingDataIndicator;
A = standardizeMissing(A,m);
```

Data Types: `double`

Categories — Category names
`string` array

This property is read-only.

Category names, returned as a `string` array.

The value of `Categories` may be empty, even when the data is grouped into categories.

Data Types: `string`

ColorType — Color type of image
`'indexed'` | `'grayscale'` | `'truecolor'` | `'CMYK'` | `'HSL'` | `'unknown'`

This property is read-only.

Color type of image, returned as one of these values:

- `'indexed'` – Indexed image.
- `'grayscale'` – Grayscale intensity image.
- `'truecolor'` – True color image using RGB color space.
- `'CMYK'` – Image using CMYK color space.
- `'HSL'` – Image using HSL color space.
- `'unknown'` – Unknown color type, or raster data does not represent an image.

Data Types: `string`

Colormap — Colormap
n-by-3 matrix

This property is read-only.

Colormap associated with an indexed image, returned as a *n*-by-3 matrix with values in the range [0,1]. Each row of `Colormap` is a three-element RGB triplet that specifies the red, green, and blue components of a single color of the colormap. The value of `Colormap` is empty unless the value of `ColorType` is `'indexed'`.

Data Types: `double`

RasterReference — Spatial reference
`GeographicCellsReference` object | `GeographicPostingsReference` object | `MapCellsReference` object | `MapPostingsReference` object

This property is read-only.

Spatial reference for the raster data, returned as a `GeographicCellsReference` object, `GeographicPostingsReference` object, `MapCellsReference` object, or `MapPostingsReference` object. The value of `RasterReference` depends on the raster data contained in the file:

- If the raster data is referenced to a geographic coordinate system, then `RasterReference` is a `GeographicCellsReference` object or `GeographicPostingsReference` object.
- If the raster data is referenced to a projected coordinate system, then `RasterReference` is a `MapCellsReference` object or `MapPostingsReference` object.

If the file does not contain enough information to determine whether the data is projected or geographic, then `RasterReference` is a `MapCellsReference` or `MapPostingsReference` object. If a file contains no valid spatial reference information, then `RasterReference` is empty.

Regardless of the file format, the `ColumnsStartFrom` property of the reference object returned by `RasterReference` has a value of `'north'`.

CoordinateReferenceSystem — Coordinate reference system
`[]` (default) | `projcrs` object | `geocrs` object

This property is read-only.

Coordinate reference system (CRS), returned as a `geocrs` or `projcrs` object. The value of `CoordinateReferenceSystem` depends on the raster data contained in the file:

- If the raster data is referenced to a geographic coordinate system, then `CoordinateReferenceSystem` is a `geocrs` object.
- If the raster data is referenced to a projected coordinate system, then `CoordinateReferenceSystem` is a `projcrs` object.
- If the file does not contain valid coordinate reference system information, then `CoordinateReferenceSystem` is empty.

Metadata — Metadata
`struct`

This property is read-only.

Metadata, returned as a `struct` of additional information that is specific to the data file. Each field of `Metadata` is returned as a string. The formatting of field names and values is dependent on the data contained in the file.

This property only applies to file formats that have metadata. For example, DTED files may contain metadata such as the datum, coordinates of the data origin, and absolute accuracy.

Examples

Get GeoTIFF Image Information

Get information about a GeoTIFF image by creating a `RasterInfo` object. Find the native data type embedded in the file by accessing the `NativeFormat` property of the `RasterInfo` object.

```
info = georasterinfo('boston.tif');
info.NativeFormat
```

```
ans =
"uint8"
```

The data used in this example includes material copyrighted by GeoEye, all rights reserved.

See Also
`readgeoraster` | `georasterinfo`

Topics
"Find Geospatial Raster Data"

Introduced in R2020a

rcurve

Ellipsoidal radii of curvature

Syntax

```
r = rcurve(ellipsoid,lat)
r = rcurve('parallel',ellipsoid,lat)
r = rcurve('meridian',ellipsoid,lat)
r = rcurve('transverse',ellipsoid,lat)
r = rcurve(..., angleunits)
```

Description

`r = rcurve(ellipsoid,lat)` and `r = rcurve('parallel',ellipsoid,lat)` return the parallel radius of curvature at the latitude `lat` for a reference ellipsoid defined by `ellipsoid`, which can be a `referenceSphere`, `referenceEllipsoid`, or `oblateSpheroid` object, or a vector of the form `[semimajor_axis eccentricity]`. `r` is in units of length consistent with those used for the semimajor axis. `lat` is in `'degrees'`.

`r = rcurve('meridian',ellipsoid,lat)` returns the meridional radius of curvature, which is the radius of curvature in the plane of a meridian at the latitude `lat`.

`r = rcurve('transverse',ellipsoid,lat)` returns the transverse radius of curvature, which is the radius of a curvature in a plane normal to the surface of the ellipsoid and normal to a meridian, at the latitude `lat`.

`r = rcurve(..., angleunits)` specifies the units of the input `lat`. `angleunits` can be `'degrees'` or `'radians'`.

Examples

The radii of curvature of the default ellipsoid at 45º, in kilometers:

```
r = rcurve('transverse',referenceEllipsoid('earth','km'),...
           45,'degrees')

r =
   6.3888e+03

r = rcurve('meridian',referenceEllipsoid('earth','km'),...
           45,'degrees')

r =
   6.3674e+03

r = rcurve('parallel',referenceEllipsoid('earth','km'),...
           45,'degrees')

r =
   4.5024e+03
```

See Also

`rsphere`

Introduced before R2006a

readfields

Read fields or records from fixed-format files

Syntax

```
struc = readfields(fname,fstruc)
struc = readfields(fname,fstruc,recordIDs)
struc = readfields(fname,fstruc,fieldIDs)
struc = readfields(fname,fstruc,recordIDs,mformat)
struc = readfields(fname,fstruc,recordIDs,mformat,fid)
struc = readfields(fname,fstruc,recordIDs,mformat,fid,'sparse')
```

Description

`struc = readfields(fname,fstruc)` reads all the records from a fixed format file. *fname* is a character vector containing the name of the file. If it is empty, the file is selected interactively. `fstruc` is a structure defining the format of the file. The contents of `fstruc` are described below. The result is returned in a structure.

`struc = readfields(fname,fstruc,recordIDs)` reads only the records specified in the vector `recordIDs`. For example, `recordIDs = [1 2 3 4]`. All the fields in the selected records are read.

`struc = readfields(fname,fstruc,fieldIDs)` reads only the fields specified in the cell array `fieldIDs`. For example, `fieldIDs = {1 2 4}`. The selected fields are read from all the records. `fieldIDs` can be used in place of `recordIDs` in all calling forms.

`struc = readfields(fname,fstruc,recordIDs,mformat)` opens the file with the specified machine format. *mformat* must be recognized by `fopen`.

`struc = readfields(fname,fstruc,recordIDs,mformat,fid)` reads from a file that is already open. `fid` is the file identifier returned by `fopen`. The records are read starting from the current location in the file.

`struc = readfields(fname,fstruc,recordIDs,mformat,fid,'sparse')` disables error messages when the number of elements read does not agree with the stated format of the file. This is useful for formatted files with empty fields. Use `fid = []` for files that are not already open. This option is only compatible with reading selected records.

Background

Map data is often provided as binary or ASCII files with a fixed format. Writing your own functions to read the data into the MATLAB workspace can be difficult and time-consuming, particularly for binary files. This function allows you to read the data by simply specifying the format of the file.

Examples

Write a binary file and read it.

```
fid = fopen('testbin','wb');
for i = 1:3
```

```
      fwrite(fid,['character' num2str(i) ],'char');
      fwrite(fid,i,'int8');
      fwrite(fid,[i i],'int16');
      fwrite(fid,i,'integer*4');
      fwrite(fid,i,'real*8');
end
fclose(fid);

fs(1).length = 10;fs(1).type = 'char';fs(1).name = 'field 1';
fs(2).length = 1;fs(2).type = 'int8';    fs(2).name = 'field 2';
fs(3).length = 2;fs(3).type = 'int16';fs(3).name = 'field 3';
fs(4).length = 1;fs(4).type = 'integer*4';fs(4).name = 'field 4';
fs(5).length = 1;fs(5).type = 'float64'; fs(5).name = 'field 5';

s = readfields('testbin',fs);

s(1)
ans =
    field1: 'character1'
    field2: 1
    field3: [1 1]
    field4: 1
    field5: 1
```

Limitations

Formatted numbers must stay within the width specified for them. Files must have a size that is an integer multiple of the computed record length. This is potentially a problem for formatted files on DOS platforms that use a carriage return/linefeed line ending everywhere except the last record. File sizes are not checked when an open file is provided.

Tips

The format of the file is described in the input argument `fstruc`. `fstruc` is a structure with one entry for every field in the file. `fstruc` has three required fields: `length`, `name`, and `type`. For fields containing binary data of the type that would be read by `fread`, `length` is the number of elements to be read, `name` is a character vector containing the field name under which the read data is stored in the output structure, and `type` is a format recognized by `fread`. Repetition modifiers such as `'40*char'` are *not* supported. Fields with empty field names are omitted from the output.

The following `fstruc` definition is for a file with a 40-character field, a field containing two integers, and a field with a single-precision floating-point number.

```
fstruc(1).length = 40;
fstruc(1).name = 'character Field'; % spaces will be suppressed
filestruc(1).type = 'char';

fstruc(2).length = 2;
fstruc(2).name = 'integer Field';   % spaces will be suppressed
fstruc(2).type = 'int16';

fstruc(3).length = 1;
fstruc(3).name = 'float Field';      % spaces will be suppressed
fstruc(3).type = 'real*4';
```

The type can also be a `fscanf` and `sscanf`-style format of the form `'%nX'`, where n is the number of characters within which the formatted data is found, and X is the conversion character such as `'g'` or `'d'`. For formatted fields, the length entry in `fstruc` is the number of elements, each of which has the width specified in the type. Fortran-style double-precision output such as `'0.0D00'` can be read using a type such as `'%nD'`, where n is the number of characters per element. This is an extension to the C-style formats accepted by `sscanf`. Users unfamiliar with C should note that `'%d'` is preferred over `'%i'` for formatted integers. MATLAB syntax follows C in interpreting `'%i'` integers with

leading zeros as octal. Line-ending characters in ASCII files must also be counted in the `fstruc` specification. Note that the number of line-ending characters differs across platforms.

A field specification for a formatted field with two integers each six characters wide would be of the form

```
fstruc(4).length = 2;
fstruc(4).name = 'Elevation Units';
fstruc(4).type = '%6d'
```

To summarize, `length` is the number of elements for binary numbers, the number of characters, and the number of elements for formatted data.

You can omit fields from all output by providing an empty character vector (`''`) for the `fstruc` name field.

See Also
`grepfields` | `readmtx` | `textread` | `spcread` | `dlmread`

Introduced before R2006a

readfk5

Read Fifth Fundamental Catalog of Stars

Note readfk5 will be removed in a future release.

Syntax

```
struc = readfk5(filename)
struc = readfk5(filename,struc)
```

Description

struc = readfk5(*filename*) reads the FK5 file and returns the contents in a structure. Each star is an element in the structure, with the different data items stored in appropriately named fields.

struc = readfk5(*filename*,struc) appends the data in the file to the existing structure struc.

Background

The Fifth Fundamental Catalog of Stars (FK5), Parts I and II, is a compilation of data on more than 4500 stars. The catalog contains positions, errors in positions, proper motions, and characteristics such as magnitudes, spectral types, parallaxes, and radial velocities. There are also cross-references to the identities of stars in other catalogs. It was compiled by researchers at the Astronomisches Rechen-Institut in Heidelberg.

Examples

```
FK5 = readfk5('FK5.dat');
FK5e = readfk5('FK5_ext.dat');
whos

  Name        Size          Bytes  Class
  FK5         1x1535      5042752  struct array
  FK5e        1x3117     10226424  struct array
FK5e(1)

ans =
        FK5: 2003
        RAh: 0
        RAm: 5
        RAs: 1.1940
       pmRA: 0.6230
        DEd: 27
        DEm: 40
        DEs: 29.0100
       pmDE: -1.1100
    RAh1950: 0
    RAm1950: 2
    RAs1950: 26.5900
```

```
pmRA1950: 0.6210
 DEd1950: 27
 DEm1950: 23
 DEs1950: 47.4400
pmDE1950: -1.1100
EpRA1900: 51.7200
   e_RAs: 2
  e_pmRA: 9
EpDE1900: 46.8200
   e_DEs: 3.4000
  e_pmDE: 14
    Vmag: 6.4700
  n_Vmag: ''
  SpType: 'G5'
     plx: []
      RV: 12
   AGK3R: '38'
     SRS: ''
      HD: '225292'
      DM: 'BD+26 4744'
      GC: '48'
```

Tips

Positions are given in terms of right ascension and declination. The Fifth Fundamental Catalog of Stars (FK5), Parts I and II data and documentation are available over the Internet by anonymous ftp.

See Also

dms2degrees | scatterm

Introduced before R2006a

readgeoraster

Read geospatial raster data file

Syntax

```
[A,R] = readgeoraster(filename)
[A,R] = readgeoraster( ___ ,Name,Value)
[ ___ ,cmap] = readgeoraster( ___ )
```

Description

`[A,R] = readgeoraster(filename)` creates an array by reading geographic or projected raster data from a file. The output argument R contains spatial referencing information for the array. Supported file formats include Esri Binary Grid, Esri GridFloat, GeoTIFF, and DTED. For a full list of supported formats, see "Supported Formats and Extensions" on page 1-1180.

`[A,R] = readgeoraster( ___ ,Name,Value)` specifies options using one or more `Name,Value` pair arguments.

`[ ___ ,cmap] = readgeoraster( ___ )` also returns the colormap of A.

Examples

Read and Display GeoTIFF Image

Read a GeoTIFF image of Boston as an array and a map cells reference object. The array is of size 2881-by-4481-by-3 and specifies the red, green, and blue components of the image. Display the image using the `mapshow` function.

```
[A,R] = readgeoraster('boston.tif');
mapshow(A,R)
```
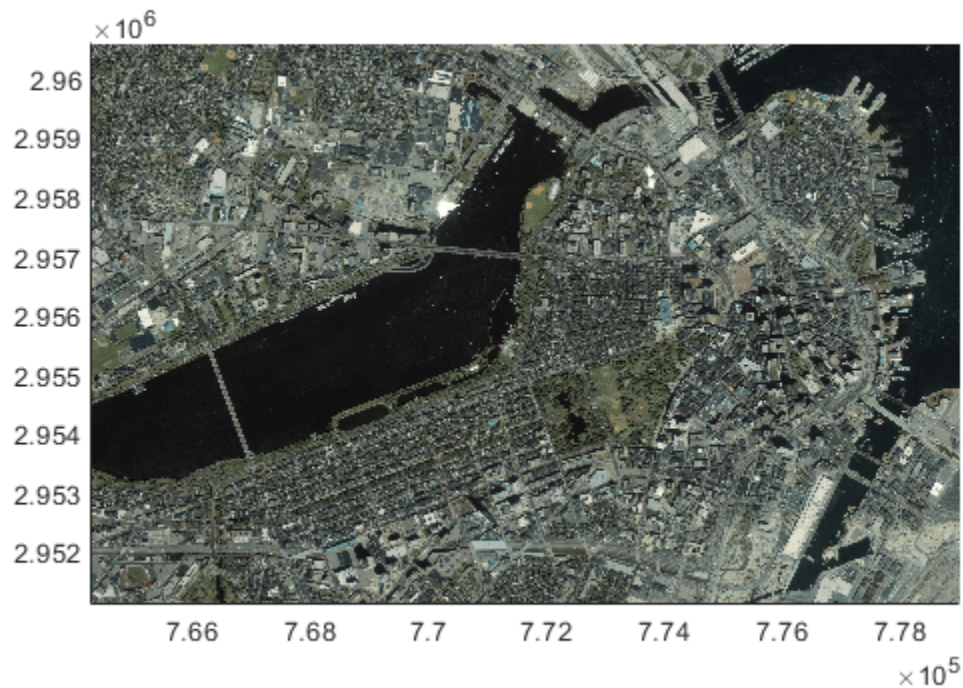
The data used in this example includes material copyrighted by GeoEye, all rights reserved.

Read and Display Land Cover Classification

Read and display a land cover classification of Oahu, Hawaii.

First, read the land cover data as an array, a map cells reference object, and a colormap. The elements of A index into the colormap. Each row of the colormap specifies the red, green, and blue components of a single color. Then, display the land cover data.

```
[A,R,cmap] = readgeoraster('oahu_landcover.img');
mapshow(A,cmap,R)
```

The data used in this example is courtesy of the National Oceanic and Atmospheric Administration (NOAA).

Read and Display Elevation Data

Read and display elevation data for an area around South Boulder Peak in Colorado.

First, read the elevation data as an array and a geographic postings reference object. To display the data as a surface, the `geoshow` function requires data of type `double` or `single`. In this case, preserve precision by specifying the output type as `'double'`.

```
[A,R] = readgeoraster('n39_w106_3arc_v2.dt1','OutputType','double');
```

Create a map. First, create map axes by specifying the latitude and longitude limits of the data. Then, display the data as a surface using the `geoshow` function. Apply a colormap appropriate for elevation data using the `demcmap` function.

```
latlim = R.LatitudeLimits;
lonlim = R.LongitudeLimits;
usamap(latlim,lonlim)
geoshow(A,R,'DisplayType','surface')
demcmap(A)
```
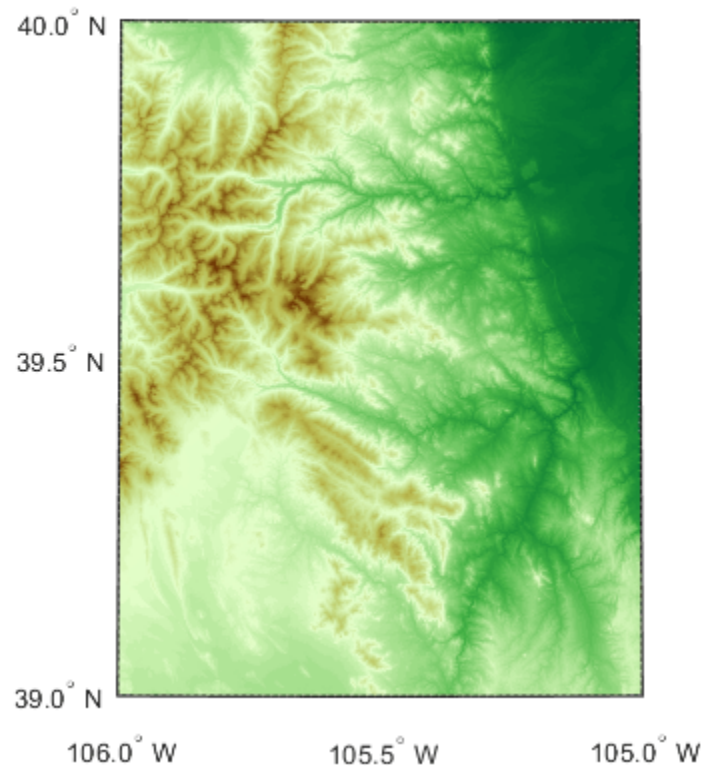
The elevation data used in this example is courtesy of the US Geological Survey.

Replace Missing Data with NaN Values

Raster data sets sometimes indicate missing data values using a large negative number. Import raster data, find the missing data indicator, and then replace missing data with NaN values.

Import raster data and a reference object using the `readgeoraster` function. Find the missing data indicator using the `georasterinfo` function.

```
[A,R] = readgeoraster('MtWashington-ft.grd');
info = georasterinfo('MtWashington-ft.grd');
m = info.MissingDataIndicator
```

```
m = -32766
```

Verify the raster data contains missing data using the `ismember` function. The `ismember` function returns logical 1 (`true`) if the raster contains the missing data indicator.

```
ismember(m,A)
```

```
ans = logical
   1
```

Replace the missing data with `NaN` values using the `standardizeMissing` function.

```
A = standardizeMissing(A,m);
```

Input Arguments

`filename` — Name of file to read
character vector | string scalar

Name of the file to read, specified as a character vector or string scalar. The form of `filename` depends on the location of your file.

- If the file is in your current folder or in a folder on the MATLAB path, then specify the name of the file, such as `'myFile.dem'`.
- If the file is not in the current folder or in a folder on the MATLAB path, then specify the full or relative path name, such as `'C:\myfolder\myFile.tif'` or `'dataDir\myFile.dat'`.

For a list of supported file formats, see "Supported Formats and Extensions" on page 1-1180.

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'OutputType','double','Bands',1:2`

`OutputType` — Data type for A
`'native'` (default) | `'single'` | `'double'` | `'int32'` | …

Data type for A, specified as the comma-separated pair consisting of `'OutputType'` and a character vector or string scalar containing one of these values: `'native'`, `'single'`, `'double'`, `'int16'`, `'int32'`, `'int64'`, `'uint8'`, `'uint16'`, `'uint32'`, `'uint64'`, or `'logical'`.

The default for `'OutputType'` is `'native'`, which returns A using the native data type embedded in `filename`. Using a data type other than `'native'` may result in a loss of precision.

Example: `'OutputType','double'`

Data Types: `char` | `string`

`Bands` — Bands to read
`'all'` (default) | positive integer | vector of positive integers

Bands to read, specified as the comma-separated pair consisting of `'Bands'` and `'all'`, a positive integer, or a vector of positive integers. For example, if you specify the value 3, `readgeoraster` reads the third band in the file. Bands are returned in the specified order.

The default for `'Bands'` is `'all'`, where `readgeoraster` reads all bands in the file.

Example: `'Bands',3`

`CoordinateSystemType` — Coordinate system type for R
`'auto'` (default) | `'geographic'` | `'planar'`

Coordinate system type for R, specified as one of these values:

- `'auto'` – Returns R as a raster reference object determined by the contents of the file.
- `'geographic'` – Returns R as a geographic cells or postings reference object.
- `'planar'` – Returns R as a map cells or postings reference object.

Specify the coordinate system type when your data does not contain projection information.

Example: `'CoordinateSystemType','geographic'`

Output Arguments

A — Georeferenced image or data grid
numeric array

Georeferenced image or data grid, returned as an *M*-by-*N* or *M*-by-*N*-by-*P* numeric array.

By default, the data type of A matches the native data type embedded in `filename`. Specify a data type using the `'OutputType'` name-value pair.

Regardless of how the data is encoded, the first row of A represents the northernmost data, and the last row of A represents the southernmost data.

R — Spatial reference
GeographicCellsReference object | GeographicPostingsReference object | MapCellsReference object | MapPostingsReference object

Spatial reference for A, returned as a `GeographicCellsReference` object, `GeographicPostingsReference` object, `MapCellsReference` object, or `MapPostingsReference` object. The value of R depends on the data in `filename`:

- If the data in `filename` is referenced to a geographic coordinate system, then R is a `GeographicCellsReference` object or `GeographicPostingsReference` object.
- If the data in `filename` is referenced to a projected coordinate system, then R is a `MapCellsReference` object or `MapPostingsReference` object.

If the file does not contain enough information to determine whether the data is projected or geographic, then R is a `MapCellsReference` or `MapPostingsReference` object. If a file contains no valid spatial reference information, then R is empty. You can specify the spatial reference as `'geographic'` or `'planar'` using the `'CoordinateSystemType'` name-value pair.

cmap — Colormap
n-by-3 matrix

Colormap associated with an indexed image, returned as a *n*-by-3 numeric matrix with values in the range [0,1]. Each row of `cmap` is a three-element RGB triplet that specifies the red, green, and blue components of a single color in the colormap. The value of `cmap` is empty unless A is an indexed image.

More About

Supported Formats and Extensions

The `readgeoraster` and `georasterinfo` functions support these file formats and extensions. In some cases, you can read supported file formats using extensions other than the ones listed.

File Format	Extension
GeoTIFF	`.tif` or `.tiff`
Esri Binary Grid	`.adf`
Esri ASCII Grid	`.asc` or `.grd`
Esri GridFloat	`.flt`
DTED	`.dt0`, `.dt1`, or `.dt2`
SDTS	`.DDF`
USGS DEM	`.dem`
ER Mapper ERS	`.ers`
ENVI	`.dat`
ERDAS IMAGINE	`.img`
Vertical Mapper Numeric Grid	`.grd`
Vertical Mapper Classified Grid	`.grc`
SRTM Height	`.hgt`

Some file formats consist of a data file and multiple supporting files. For example, Esri GridFloat files may have supporting header files (`.hdr`). When you read a data file with supporting files using `readgeoraster` or `georasterinfo`, specify the extension of the data file.

File formats may be referred to using different names. For example, the Esri GridFloat format may also be referred to as Esri `.hdr` Labelled or ITT ESRI `.hdr` RAW Raster. The Esri Binary Grid format may also be referred to as ArcGrid Binary, Esri ArcGIS Binary Grid, or Esri ArcInfo Grid.

Tips

- Some functions require input arguments of type `single` or `double`, such as the `geoshow` function for displaying surfaces. To use the output of `readgeoraster` with these functions, specify the output type as `'single'` or `'double'` using the `'OutputType'` name-value pair.

- Regardless of the file format, the array returned by `readgeoraster` has columns starting from north and the `ColumnsStartFrom` property of the reference object has a value of `'north'`.

See Also

Functions
`georasterinfo`

Topics
"Find Geospatial Raster Data"

Introduced in R2020a

readgeotable

Read geospatial table from vector data file

Syntax

```
T = readgeotable(filename)
T = readgeotable(filename,Name=Value)
```

Description

`T = readgeotable(filename)` creates a geospatial table by reading geographic or projected vector data from a file. Supported file formats include shapefile, KML, and Esri file geodatabase. For a full list of supported formats, see "Supported Formats and Extensions" on page 1-1188.

`T = readgeotable(filename,Name=Value)` specifies options using name-value arguments. For example, specify the coordinate system type using the `CoordinateSystemType` name-value argument.

Examples

Read Shapefile as Geospatial Table

Read a shapefile, containing a network of roads in Concord, MA, into the workspace as a geospatial table.

```
T = readgeotable("concord_roads.shp");
```

The `Shape` variable of the table contains information about the road shapes, including the coordinate reference system (CRS). The road shapes in this shapefile use a projected CRS.

```
T.Shape.ProjectedCRS
```

```
ans =
  projcrs with properties:

                   Name: "NAD83 / Massachusetts Mainland"
          GeographicCRS: [1x1 geocrs]
       ProjectionMethod: "Lambert Conic Conformal (2SP)"
             LengthUnit: "meter"
   ProjectionParameters: [1x1 map.crs.ProjectionParameters]
```

Display the roads on a map.

```
mapshow(T)
xlabel("x (meters)")
ylabel("y (meters)")
```
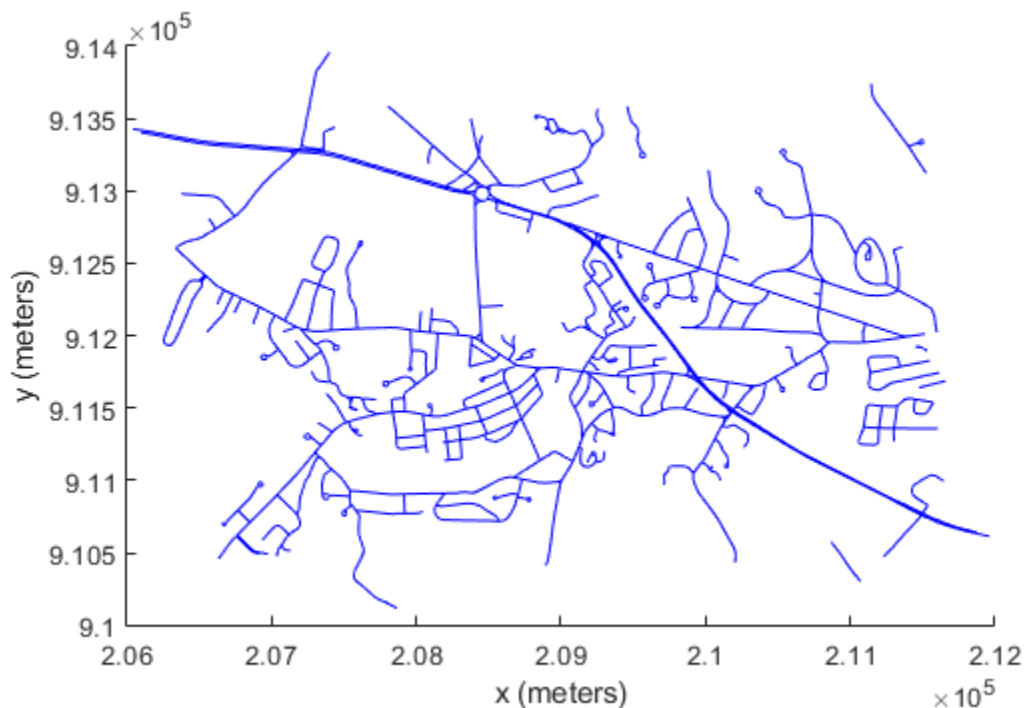
Read Shapefile with No Projection File

An optional projection file (`.prj`) determines the coordinate system type for a shapefile. When your shapefile does not have a projection file, but you know the coordinate system type, you can specify it by using the `CoordinateSystemType` name-value argument.

Read a shapefile called `tsunamis.shp`, containing information about tsunami events, into the workspace. The metadata accompanying the file indicates that the shapefile uses geographic coordinates.

```
T = readgeotable("tsunamis.shp",CoordinateSystemType="geographic");
```

View the `Shape` variable of the geospatial table. The tsunami source locations are stored as points.

```
T.Shape
```

```
ans =
  162×1 geopointshape array with properties:

              NumPoints: [162×1 double]
               Latitude: [162×1 double]
              Longitude: [162×1 double]
               Geometry: "point"
    CoordinateSystemType: "geographic"
```

```
        GeographicCRS: []
```

Plot the source locations on a web map.

```
wmmarker(T)
```
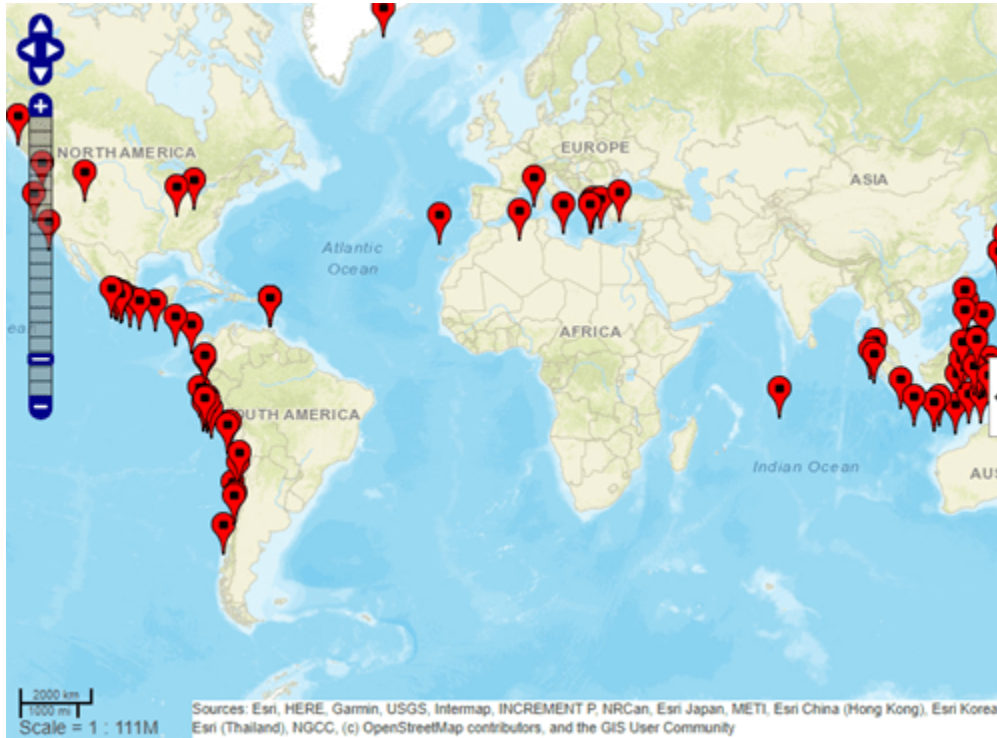

Read Layers from GPX Files

GPX files can contain up to five layers: waypoints, tracks, track points, routes, and route points. When you read a layer containing track points or route points, the geospatial table contains an ID variable that associates the points with a track or route.

Import the tracks layer of a GPX file with two tracks. The `Shape` variable for each track is a `geolineshape` object.

```
T = readgeotable("sample_tracks.gpx",Layer="tracks")
```

T=*2×3 table*

Shape	Name
geolineshape	"Track logs from walking the perimeter of the MathWorks campus in Natick on M
geolineshape	"Track logs from biking from Concord to the MathWorks campus in Natick on Jui

View the shape of each track. The first track has one segment and the second track has five segments.

```
T.Shape(1)

ans =
  geolineshape with properties:

                 NumParts: 1
                 Geometry: "line"
     CoordinateSystemType: "geographic"
            GeographicCRS: [1×1 geocrs]


T.Shape(2)

ans =
  geolineshape with properties:

                 NumParts: 5
                 Geometry: "line"
     CoordinateSystemType: "geographic"
            GeographicCRS: [1×1 geocrs]
```

Import the track points layer. The `Shape` variable for each point is a `geopointshape` object.

```
T2 = readgeotable("sample_tracks.gpx",Layer="track_points");
T2.Shape

ans =
  2586×1 geopointshape array with properties:

                NumPoints: [2586×1 double]
                 Latitude: [2586×1 double]
                Longitude: [2586×1 double]
                 Geometry: "point"
     CoordinateSystemType: "geographic"
            GeographicCRS: [1×1 geocrs]
```

Create a subtable that contains points in the second track only. For this file, points in the second track have a `TrackFID` value of `1`.

```
rows = (T2.TrackFID == 1);
T3 = T2(rows,:);
```

Plot the points in the subtable as a blue line.

```
wmline(T3,Color="b")
```
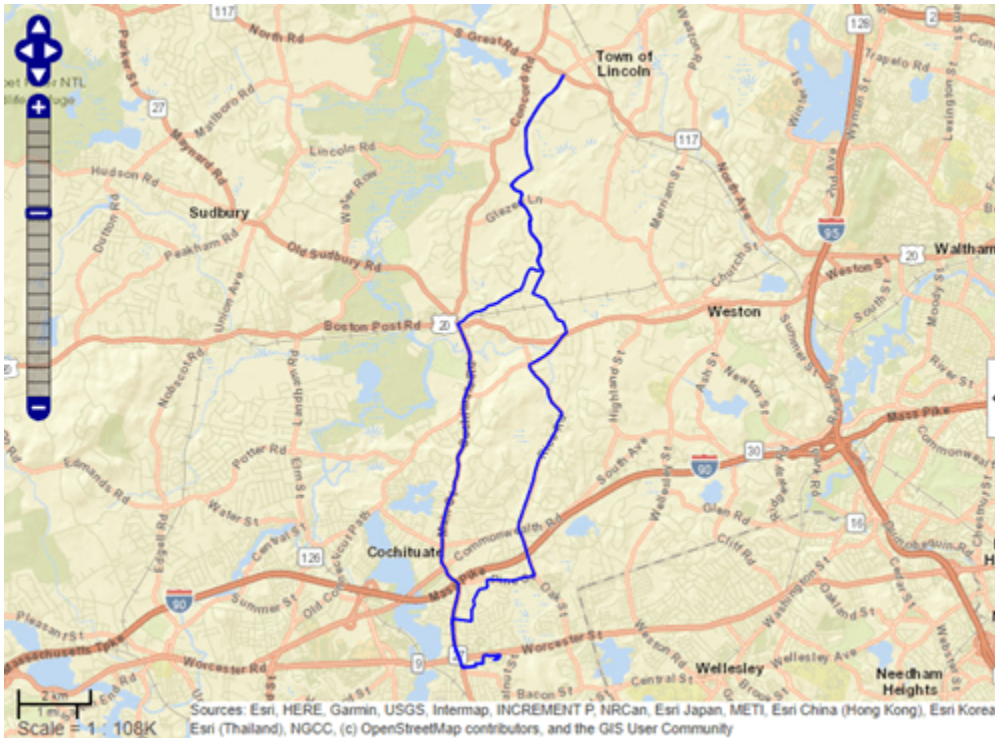
Input Arguments

`filename` — Name of file to read
character vector | string scalar

Name of the file to read, specified as a character vector or string scalar. The form of `filename` depends on the location of your file.

- If the file is in your current folder or in a folder on the MATLAB path, then specify the name of the file, such as `"myFile.shp"`.
- If the file is not in the current folder or in a folder on the MATLAB path, then specify the full or relative path name, such as `"C:\myfolder\myFile.gpx"` or `"dataDir\myFile.kml"`.

For a list of supported file formats, see "Supported Formats and Extensions" on page 1-1188.

To read an Esri file geodatabase, you must either specify the full or relative path name or include the geodatabase in your current folder.

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `readgeotable("sample_tracks.gpx",Layer="track_points")` reads the track points layer from the GPX file.

Layer — Layer to read
positive integer | string scalar | character vector

Layer to read when the file contains multiple layers, specified as a positive integer, a string scalar, or a character vector. If you specify an integer, then it must be less than or equal to the number of vector layers in the file.

The layers contained in a file depend on the file format. Shapefiles have one layer. The content of Esri file geodatabases and GeoJSON files determines the number of layers and the names of the layers. Each folder and subfolder of KML files corresponds to layers, and the folder names match the layer names.

For GPX files, specify the layer as one of these options:

- `"waypoints"` — Waypoints.
- `"tracks"` — Track lines.
- `"track_points"` — Track points. When you read track points, the geospatial table contains an ID variable, `TrackFID`, that associates each point with a track.
- `"routes"` — Route lines.
- `"route_points"` — Route points. When you read route points, the geospatial table contains an ID variable, `RouteFID`, that associates each point with a route.

Data Types: `single` | `double` | `char` | `string`

CoordinateSystemType — Coordinate system type
`"auto"` (default) | `"geographic"` | `"planar"`

Coordinate system type, specified as one of these values:

- `"auto"` — Create the `Shape` table variable based on the contents of the file.
- `"geographic"` — Create the `Shape` table variable using `geopointshape`, `geolineshape`, and `geopolyshape` objects.
- `"planar"` — Create the `Shape` table variable using `mappointshape`, `maplineshape`, and `mappolyshape` objects.

Specify the coordinate system type when the `readgeotable` function is unable to detect the coordinate system type.

For shapefiles, the coordinate system type is determined by an optional projection file (`.prj`). If your shapefile does not have a projection file, you can try to determine the coordinate system type using one of these options:

- Refer to the metadata.
- Ask your data provider.
- Return information about the shapefile as a structure by using the `shapeinfo` function. Then, view the limits of the data by querying the `BoundingBox` field of the structure. The limits might help you predict the coordinate system type.

VariableNamingRule — Flag to preserve variable names
`"preserve"` (default) | `"modify"`

Flag to preserve variable names, specified as one of these options:

- `"preserve"` — Preserve variable names that are not valid MATLAB identifiers, such as variable names that include spaces and non-ASCII characters.
- `"modify"` — Convert invalid variable names, as identified by the `isvarname` function, to valid MATLAB identifiers.

Data Types: `char` | `string`

Output Arguments

T — Geospatial table
geospatial table

Geospatial table, returned as a geospatial table. A geospatial table is a `table` object with a `Shape` variable and attribute variables.

- The `Shape` variable contains 2-D information about point, line, and polygon shapes. The `Shape` variable can contain combinations of shape types. All the shapes have the same coordinate reference system (CRS).

  The `readgeotable` function represents shapes with coordinates in geographic CRSs by using `geopointshape`, `geolineshape`, and `geopolyshape` objects.

  The `readgeotable` function represents shapes with coordinates in projected CRSs by using `mappointshape`, `maplineshape`, and `mappolyshape` objects.

- The attribute variables contain data such as names, classifications, and measurements.

When the data file contains CRS information, the `readgeotable` function stores the information as a `projcrs` or `geocrs` object within each shape object.

More About

Supported Formats and Extensions

The `readgeotable` function supports these file formats and extensions.

File Format	Extension
Esri file geodatabase	`.gdb`
GeoJSON	`.json` or `.geojson`
GPX	`.gpx`
KML	`.kml`
Shapefile	`.shp`

Some file formats consist of a main file and multiple supporting files. For example, shapefiles include a main file (`.shp`), an index file (`.shx`), and an attribute file (`.dbf`). When you read a data file with supporting files using the `readgeotable` function, specify the extension of the main file.

When the file contains 3-D points, lines, or polygons, the function reads only the 2-D information into the `Shape` variable of the geospatial table.

For GeoJSON files, all attributes apply to all rows in the geospatial table, even when the GeoJSON file does not apply an attribute to all elements. When the file does not apply an attribute to an element, the corresponding cell in the geospatial table contains a missing data value.

For GPX files, Garmin® extensions are not supported.

For KML files, the `readgeotable` function reads only shapes, names, and descriptions into the geospatial table.

Tips

- You can save KML or GeoJSON files from a web service to a file by using the `websave` function.

See Also

Functions
`readgeoraster` | `readtable`

Topics
"Create Geospatial Tables"
"Find Geospatial Vector Data"

Introduced in R2021b

readmtx

Read matrix stored in file

Syntax

```
mtx = readmtx(fname,nrows,ncols,precision)
mtx = readmtx(fname,nrows,ncols,precision,readrows,readcols)
mtx = readmtx(fname,nrows,ncols,precision,... readrows,readcols,mformat)
mtx = readmtx(fname,nrows,ncols,precision,...
readrows,readcols,mformat,nheadbytes)
mtx = readmtx(fname,nrows,ncols,precision,...
readrows,readcols,mformat,nheadbytes,nRowHeadBytes)
mtx = readmtx(fname,nrows,ncols,precision,...
readrows,readcols,mformat,nheadbytes,nRowHeadBytes,nRowTrailBytes)
mtx = readmtx(fname,nrows,ncols,precision,...
readrows,readcols,mformat,nheadbytes,nRowHeadBytes,...
nRowTrailBytes,nFileTrailBytes)
mtx = readmtx(fname,nrows,ncols,precision,...
readrows,readcols,mformat,nheadbytes,nRowHeadBytes,...
nRowTrailBytes,nFileTrailBytes,recordlen)
```

Description

mtx = readmtx(*fname*,nrows,ncols,*precision*) reads a matrix stored in a file. The file contains only a matrix of numbers with the dimensions nrows by ncols stored with the specified *precision*. Recognized *precision* values are described below.

mtx = readmtx(*fname*,nrows,ncols,*precision*,readrows,readcols) reads a subset of the matrix. readrows and readcols specify which rows and columns are to be read. They can be vectors containing the row or column numbers, or two-element vectors of the form [start end], which are expanded using the colon operator to start:end. To read just two rows or columns, without expansion by the colon operator, provide the indices as a column matrix.

mtx = readmtx(*fname*,nrows,ncols,*precision*,... readrows,readcols,*mformat*) specifies the machine format used to write the file. *mformat* can be any recognized by fopen. This option is used to automatically swap bytes for files written on platforms with a different byte ordering.

mtx = readmtx(*fname*,nrows,ncols,*precision*,...
readrows,readcols,*mformat*,nheadbytes) skips the file header, whose length is specified in bytes.

mtx = readmtx(*fname*,nrows,ncols,*precision*,...
readrows,readcols,*mformat*,nheadbytes,nRowHeadBytes) also skips a header that precedes every row of the matrix. The length of the header is specified in bytes.

mtx = readmtx(*fname*,nrows,ncols,*precision*,...
readrows,readcols,*mformat*,nheadbytes,nRowHeadBytes,nRowTrailBytes) also skips a trailer that follows every row of the matrix. The length of the trailer is specified in bytes.

```
mtx = readmtx(fname,nrows,ncols,precision,...
readrows,readcols,mformat,nheadbytes,nRowHeadBytes,...
nRowTrailBytes,nFileTrailBytes)
```
accounts for the length of data following the matrix. The sizes of the components of the matrix are used to compute an expected file size, which is compared to the actual file size.

```
mtx = readmtx(fname,nrows,ncols,precision,...
readrows,readcols,mformat,nheadbytes,nRowHeadBytes,...
nRowTrailBytes,nFileTrailBytes,recordlen)
```
overrides the record length calculated from the precision and number of columns, and instead uses the record length given in bytes. This is used for formatted data with extra spaces or line breaks in the matrix.

Background

Map data is often provided as binary or ASCII files with a fixed format. Writing your own functions to read the data into the MATLAB workspace can be difficult and time-consuming, particularly for binary files. This function allows you to read the data by simply specifying the format of the file.

Examples

Write and read a binary matrix file:

```
fid = fopen('binmat','w');
fwrite(fid,1:100,'int16');
fclose(fid);
mtx = readmtx('binmat',10,10,'int16')

mtx =
     1     2     3     4     5     6     7     8     9    10
    11    12    13    14    15    16    17    18    19    20
    21    22    23    24    25    26    27    28    29    30
    31    32    33    34    35    36    37    38    39    40
    41    42    43    44    45    46    47    48    49    50
    51    52    53    54    55    56    57    58    59    60
    61    62    63    64    65    66    67    68    69    70
    71    72    73    74    75    76    77    78    79    80
    81    82    83    84    85    86    87    88    89    90
    91    92    93    94    95    96    97    98    99   100

mtx = readmtx('binmat',10,10,'int16',[2 5],3:2:9)

mtx =
    13    15    17    19
    23    25    27    29
    33    35    37    39
    43    45    47    49
```

Limitations

Every row of the matrix must have the same number of elements.

Tips

This function reads files that have a general format consisting of a header, a matrix, and a trailer. Each row of the matrix can have a certain number of bytes of extraneous information preceding or following the matrix data.

Both binary and formatted data files can be read. If the file is binary, the precision argument is a format recognized by `fread`. Repetition modifiers such as `'40*char'` are *not* supported. If the file is formatted, precision is a `fscanf` and `sscanf`-style format of the form `'%nX'`, where `n` is the number of characters within which the formatted data is found, and `X` is the conversion character such as `'g'` or `'d'`. Fortran-style double-precision output such as `'0.0D00'` can be read using a precision such as `'%nD'`, where `n` is the number of characters per element. This is an extension to the C-style format s accepted by `sscanf`. Users unfamiliar with C should note that `'%d'` is preferred over `'%i'` for formatted integers. MATLAB syntax follows C in interpreting `'%i'` integers with leading zeros as octal. Formatted files with line endings need to provide the number of trailing bytes per row, which can be 1 for platforms with carriage returns *or* linefeed (Macintosh, UNIX), or 2 for platforms with carriage returns *and* linefeeds (DOS).

See Also
`readfields` | `textread` | `spcread` | `dlmread`

Introduced before R2006a

reckon

Point at specified azimuth, range on sphere or ellipsoid

Syntax

```
[latout,lonout] = reckon(lat,lon,arclen,az)
[latout,lonout] = reckon(lat,lon,arclen,az,units)
[latout,lonout] = reckon(lat,lon,arclen,az,ellipsoid)
[latout,lonout] = reckon(lat,lon,arclen,az,ellipsoid,units)
[latout,lonout] = reckon(track,...)
```

Description

`[latout,lonout] = reckon(lat,lon,arclen,az)`, for scalar inputs, calculates a position (`latout,lonout`) at a given range, `arclen`, and azimuth, `az`, along a great circle from a starting point defined by `lat` and `lon`. `lat` and `lon` are in degrees. `arclen` must be expressed as degrees of arc on a sphere, and equals the length of a great circle arc connecting the point (`lat`, `lon`) to the point (`latout`, `lonout`). `az`, also in degrees, is measured clockwise from north. `reckon` calculates multiple positions when given four arrays of matching size. When given a combination of scalar and array inputs, the scalar inputs are automatically expanded to match the size of the arrays.

`[latout,lonout] = reckon(lat,lon,arclen,az,units)`, where `units` is either `'degrees'` or `'radians'`, specifies the units of the inputs and outputs, including `arclen`. The default value is `'degrees'`.

`[latout,lonout] = reckon(lat,lon,arclen,az,ellipsoid)` calculates positions along a geodesic on an ellipsoid, as specified by `ellipsoid`. `ellipsoid` is a `referenceSphere`, `referenceEllipsoid`, or `oblateSpheroid` object, or a vector of the form [`semimajor_axis eccentricity`]. The range, `arclen`, must be expressed same unit of length as the semimajor axis of the `ellipsoid`.

`[latout,lonout] = reckon(lat,lon,arclen,az,ellipsoid,units)` calculates positions on the specified ellipsoid with `lat`, `lon`, `az`, `latout`, and `lonout` in the specified angle units.

`[latout,lonout] = reckon(track,...)` calculates positions on great circles (or geodesics) if `track` is `'gc'` and along rhumb lines if `track` is `'rh'`. The default value is `'gc'`.

Examples

Find the coordinates of the point 600 nautical miles northwest of London, UK (51.5ºN,0º) in a great circle sense:

```
% Convert nm distance to degrees.
dist = nm2deg(600)
dist =
    9.9933

% Northwest is 315 degrees.
pt1 = reckon(51.5,0,dist,315)
```

```
pt1 =
   57.8999  -13.3507
```

Now, determine where a plane from London traveling on a constant northwesterly course for 600 nautical miles would end up:

```
pt2 = reckon('rh',51.5,0,dist,315)
```

```
pt2 =
   58.5663  -12.3699
```

How far apart are the points above (distance in great circle sense)?

```
separation = distance('gc',pt1,pt2)
```

```
separation =
    0.8430
```

```
% Convert answer to nautical miles.
nmsep = deg2nm(separation)
nmsep =
   50.6156
```

Over 50 nautical miles separate the two points.

See Also
azimuth | distance | km2deg | dreckon | track | track1 | track2

Introduced before R2006a

map.geodesy.RectifyingLatitudeConverter

Convert between geodetic and rectifying latitudes

Description

A `RectifyingLatitudeConverter` object provides conversion methods between geodetic and rectifying latitudes for an ellipsoid with a given third flattening.

The rectifying latitude maps an ellipsoid (oblate spheroid) to a sphere while preserving the distances along the meridians. Rectifying latitudes are used when implementing map projections, such as Equidistant Cylindrical, that preserve such distances.

Creation

Syntax

```
converter = map.geodesy.RectifyingLatitudeConverter
converter = map.geodesy.RectifyingLatitudeConverter(spheroid)
```

Description

`converter = map.geodesy.RectifyingLatitudeConverter` returns a `RectifyingLatitudeConverter` object for a sphere and sets the `ThirdFlattening` property to 0.

`converter = map.geodesy.RectifyingLatitudeConverter(spheroid)` returns a rectifying latitude converter object and sets the `ThirdFlattening` property to match the specified spheroid object.

Input Arguments

spheroid — Reference spheroid
referenceEllipsoid object | oblateSpheroid object | referenceSphere object

Reference spheroid, specified as a `referenceEllipsoid` object, `oblateSpheroid` object, or `referenceSphere` object. The term reference spheroid is used synonymously with reference ellipsoid. To create a reference spheroid, use the creation function for the object. To specify the reference ellipsoid for WGS84, use the `wgs84Ellipsoid` function.

For more information about reference spheroids, see "Reference Spheroids".

Example: `spheroid = referenceEllipsoid('GRS 80');`

Properties

ThirdFlattening — Third flattening of an ellipsoid
numeric scalar

Third flattening of an ellipsoid, specified as a numeric scalar. `ThirdFlattening` is in the interval [0, `ecc2n(0.5)`], or approximately [0, 0.071797]. (Flatter spheroids are possible in theory, but do not occur in practice and are not supported.)

Data Types: `double`

Object Functions

forward   Convert geodetic latitude to authalic, conformal, isometric, or rectifying latitude
inverse   Convert authalic, conformal, isometric, or rectifying latitude to geodetic latitude

Examples

Create a Rectifying Latitude Converter Object and Set Property

```
grs80 = referenceEllipsoid('GRS 80');

conv1 = map.geodesy.RectifyingLatitudeConverter;
conv1.ThirdFlattening = grs80.ThirdFlattening

conv1 =

  RectifyingLatitudeConverter with properties:

    ThirdFlattening: 0.0017
```

Create a Rectifying Latitude Converter Object, Specifying Spheroid

```
grs80 = referenceEllipsoid('GRS 80');

conv2 = map.geodesy.RectifyingLatitudeConverter(grs80)

conv2 =

  RectifyingLatitudeConverter with properties:

    ThirdFlattening: 0.0017
```

See Also

Functions
geocentricLatitude | parametricLatitude

Objects
AuthalicLatitudeConverter | ConformalLatitudeConverter |
IsometricLatitudeConverter

Introduced in R2013a

reducem

Reduce density of points in vector data

Syntax

```
[latout,lonout] = reducem(latin,lonin)
[latout,lonout] = reducem(latin,lonin,tol)
[latout,lonout,cerr] = reducem(...)
[latout,lonout,cerr,tol] = reducem(...)
```

Description

`[latout,lonout] = reducem(latin,lonin)` reduces the number of points in vector map data. In this case the tolerance is computed automatically.

`[latout,lonout] = reducem(latin,lonin,tol)` uses the provided tolerance. The units of the tolerance are degrees of arc on the surface of a sphere.

`[latout,lonout,cerr] = reducem(...)` in addition returns a measure of the error introduced by the simplification. The output `cerr` is the difference in the arc length of the original and reduced data, normalized by the original length.

`[latout,lonout,cerr,tol] = reducem(...)` also returns the tolerance used in the reduction, which is useful when the tolerance is computed automatically.

Examples

Compare Original and Reduced Borders

Compare the original and reduced borders of the District of Columbia from the shapefile `usastatehi.shp`.

Extract the borders of the District of Colombia.

```
dc = shaperead('usastatehi.shp',...
    'UseGeoCoords', true,...
    'Selector',{@(name) ...
     strcmpi(name,'district of columbia'), 'Name'});
lat = extractfield(dc, 'Lat')';
lon = extractfield(dc, 'Lon')';
```

Reduce the number of points in the border.

```
[latreduced, lonreduced] = reducem(lat, lon);
```

Display the original and reduced borders.

```
lonlim = dc.BoundingBox(:,1)' + [-0.02 0.02];
latlim = dc.BoundingBox(:,2)' + [-0.02 0.02];
```

```
subplot(1,2,1)
usamap(latlim, lonlim); axis off
geoshow(lat, lon,...
    'DisplayType', 'polygon', 'FaceColor', 'blue')

subplot(1,2,2)
usamap(latlim, lonlim); axis off
geoshow(latreduced, lonreduced,...
    'DisplayType', 'polygon', 'FaceColor', 'yellow')
```
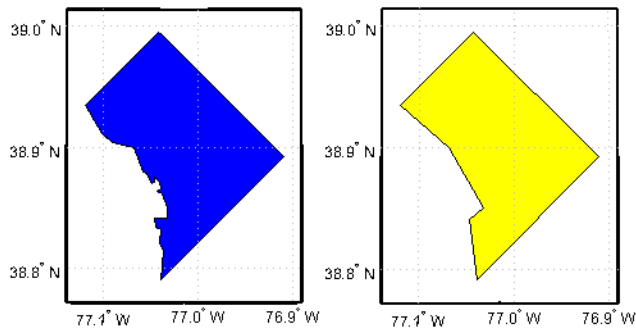

Simplify Polygon and Line Data

Simplifying polygon and line data can speed up certain calculations without making any noticeable impact on the data. One way to approach simplification is to use `reducem` with the default tolerance value and view the output. If the results do not meet your requirements, repeat the operation, increasing or decreasing the tolerance value to achieve the result you desire. The `reducem` function returns the tolerance value used.

Extract Massachusetts coastlines and state borders from the `usastatehi.shp` shapefile.

```
ma = shaperead('usastatehi.shp','UseGeoCoords',true, ...
    'Selector', {@(name)strcmpi(name,'Massachusetts'),'Name'});
masslat = ma.Lat;
masslon = ma.Lon;
```

The result is a pair of vectors that outline the state with 957 vertices.

```
numel(masslat)
```

```
ans = 957
```

Simplify Line Data Using Default Tolerance

Simplify the outline by using the `reducem` function with the default tolerance value.

```
[masslat1, masslon1, cerr, tol] = reducem(masslat', masslon');
```

Inspect the results. The number of vertices has been reduced to 252. The vectors have been reduced to about a quarter of their original lengths.

```
numel(masslat1)
```

```
ans = 252
```

```
numel(masslat1)/numel(masslat)
```

```
ans = 0.2633
```

Examine the error and tolerance values returned by `reducem`. The `cerr` value indicates that `reducem` has reduced the total length of the outline by about 3.3 percent. The tolerance that `reducem` used to achieve this reduction was 0.006 degrees, or about 660 meters.

```
[cerr tol]
```

```
ans = 1×2

    0.0331    0.0060
```

Plot the reduced outline in red over the original outline in blue, using `geoshow`. At this resolution, it's hard to see any difference between the original outline and the reduced outline.

```
figure
axesm('MapProjection', 'eqdcyl', 'MapLatLim', [41.1 43.0],...
'MapLonLim', [-73.6, -69.8], 'Frame', 'off', 'Grid', 'off');
geoshow(masslat, masslon, 'DisplayType', 'line', 'color', 'blue')
geoshow(masslat1, masslon1, 'DisplayType', 'line', 'color', 'red')
```
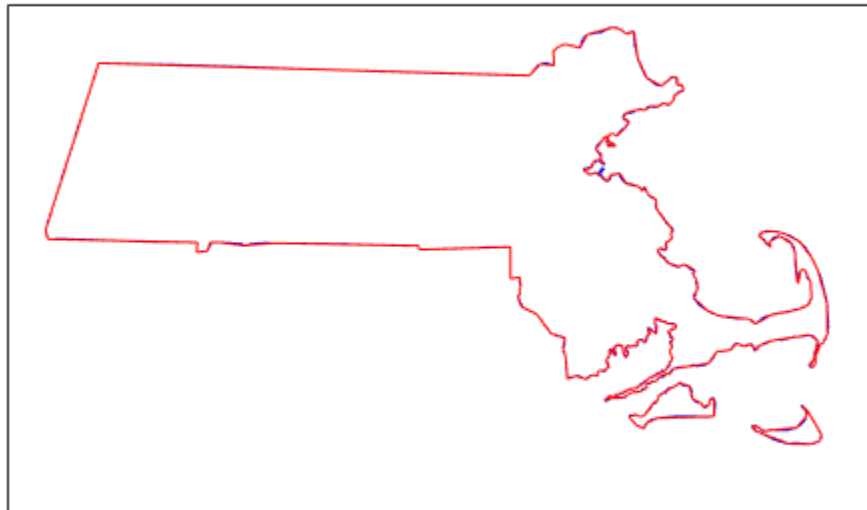
To get a better look at the two outlines, use `xlim` and `ylim` to zoom in on a portion of the map. Notice how the reduced outline conforms to the general contours of the original map but loses a lot of the detail.

```
axesm('MapProjection', 'eqdcyl', 'MapLatLim', [41.1 43.0],...
'MapLonLim', [-73.6, -69.8], 'Frame', 'off', 'Grid', 'off');
xlim([0.0104 0.0198])
ylim([0.7202 0.7264])
geoshow(masslat, masslon, 'DisplayType', 'line', 'color', 'blue')
geoshow(masslat1, masslon1, 'DisplayType', 'line', 'color', 'red')
```


Simplify Line Data Using Specified Tolerance

Perform the operation again, this time doubling the tolerance value.

```
[masslat2,masslon2,cerr2,tol2] = reducem(masslat', masslon', 0.012);
numel(masslat2)
```

```
ans = 157
```

```
numel(masslat2)/numel(masslat)
```

```
ans = 0.1641
```

Examine the error and tolerance values returned by `reducem`. This time, the `cerr` value indicates that `reducem` has reduced the total length of the outline by about 5.2 percent. The tolerance that `reducem` used to achieve this reduction was 0.012 degrees.

```
[cerr2 tol2]
```

```
ans = 1×2

    0.0517    0.0120
```

Plot this reduced outline in dark green over the original outline in blue. Note how this reduced outline maintains the general shape of the original map but loses much of the fine detail.

```
geoshow(masslat2, masslon2, 'DisplayType', 'line', 'color', [0 .6 0])
```
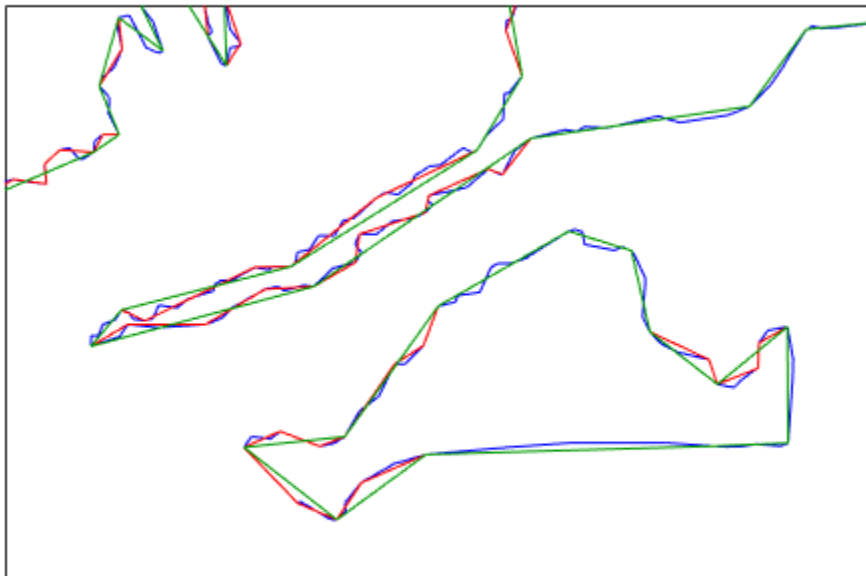

Tips

Vector data is reduced using the Douglas-Peucker line simplification algorithm. This method recursively subdivides a polygon until a run of points can be replaced by a straight line segment, with no point in that run deviating from the straight line by more than the tolerance. The distances used to decide on which runs of points to eliminate are computed in a Plate Carrée projection.

Reduced geographic data might not always be appropriate for display. If all intermediate points in a data set are reduced, then lines appearing straight in one projection are incorrectly displayed as straight lines in others.

See Also

Functions
interpm | georesize

Topics
"Simplify Vector Coordinate Data"

Introduced before R2006a

referenceEllipsoid

Reference ellipsoid

Description

A `referenceEllipsoid` object encapsulates a reference ellipsoid, modeled as an oblate spheroid with three additional properties: name, unit of length of the semi-major and semi-minor axes, and a numerical EPSG code.

Creation

You can create a general `referenceEllipsoid` object with the `referenceEllipsoid` function described here. You can also create a `referenceEllipsoid` with properties specific to the World Geodetic System 1984 reference ellipsoid using the `wgs84Ellipsoid` function.

Syntax

```
E = referenceEllipsoid
E = referenceEllipsoid(name)
E = referenceEllipsoid(code)
E = referenceEllipsoid(name,lengthUnit)
E = referenceEllipsoid(code,lengthUnit)
```

Description

`E = referenceEllipsoid` creates a `referenceEllipsoid` object that represents the unit sphere.

`E = referenceEllipsoid(name)` creates a `referenceEllipsoid` object corresponding to `name`. `name` is case-insensitive. The values of the `SemimajorAxis` and `SemiminorAxis` properties are in meters.

`E = referenceEllipsoid(code)` creates a `referenceEllipsoid` object corresponding to the numerical EPSG code, `code`. The unit of length used for the `SemimajorAxis` and `SemiminorAxis` properties depends on the ellipsoid selected, and is indicated in the property `LengthUnit`.

`E = referenceEllipsoid(name,lengthUnit)` and

`E = referenceEllipsoid(code,lengthUnit)` create a `referenceEllipsoid` object with the `SemimajorAxis` and `SemiminorAxis` properties in the specified unit of length, `LengthUnit`. The unit of length can be any length unit supported by the `validateLengthUnit` function.

Input Arguments

name — Name of the reference ellipsoid
`'Unit Sphere'` (default) | character vector | string scalar

Name of the reference ellipsoid, specified as one of the values in the "Names of EPSG Ellipsoids" on page 1-1209 table.

Data Types: `char` | `string`

code — Numerical EPSG code
`[]` (default) | integer between 7000 and 8000

Numerical EPSG code, specified as an empty vector or an integer between 7000 and 8000, although not all integers in this range are valid numerical EPSG codes. For information on valid EPSG codes, see the EPSG home page.

Data Types: `double`

lengthUnit — Length unit
`''` (default) | character vector | string scalar

Length unit, specified as a character vector or string scalar. You can specify any length unit that is supported by the `validateLengthUnit` function.

Data Types: `char` | `string`

Properties

Code — Numerical EPSG code
`[]` (default) | integer between 7000 and 8000

Numerical EPSG code, specified as an empty vector or an integer between 7000 and 8000, although not all integers in this range are valid numerical EPSG codes. For information on valid EPSG codes, see the EPSG home page.

When the reference ellipsoid represents the unit sphere or a custom ellipsoid, `Code` is an empty vector (`[]`).

Example: 7030

Data Types: `double`

Name — Name of the reference ellipsoid
`'Unit Sphere'` (default) | character vector | string scalar

Name of the reference ellipsoid, specified as a character vector or string scalar. When you create a reference ellipsoid by specifying its name, use one of the values in the "Names of EPSG Ellipsoids" on page 1-1209 table.

When the reference ellipsoid represents the unit sphere, `Name` is the character vector `'Unit Sphere'`.

Example: `'World Geodetic System 1984'`

Data Types: `char` | `string`

LengthUnit — Unit of length for the ellipsoid axes
`''` (default) | character vector | string scalar

Unit of length for the ellipsoid axes, specified as a character vector. The character vector can be empty, or it can be any unit of length accepted by the `validateLengthUnit` function.

When the reference ellipsoid represents the unit sphere, `LengthUnit` is the empty character vector `''`.

Example: `'km'`

Data Types: `char` | `string`

SemimajorAxis — Equatorial radius of ellipsoid
1 (default) | positive, finite scalar

Equatorial radius of ellipsoid, specified as a positive, finite scalar. The `SemimajorAxis` property is expressed in units of length specified by `LengthUnit`.

When the `SemimajorAxis` property is changed, the `SemiminorAxis` property scales as needed to preserve the shape of the ellipsoid and the values of shape-related properties including `InverseFlattening` and `Eccentricity`. The only way to change the `SemimajorAxis` property is to set it directly, using dot notation.

Example: 6378137

Data Types: `double`

SemiminorAxis — Distance from center of ellipsoid to pole
1 (default) | nonnegative, finite scalar

Distance from center of ellipsoid to pole, specified as a nonnegative, finite scalar. The value of `SemiminorAxis` is always less than or equal to `SemimajorAxis`, and is expressed in units of length specified by `LengthUnit`.

When the `SemiminorAxis` property is changed, the `SemimajorAxis` property remains unchanged, but the shape of the ellipsoid changes, which is reflected in changes in the values of `InverseFlattening`, `Eccentricity`, and other shape-related properties.

Example: 6356752

Data Types: `double`

InverseFlattening — Reciprocal of flattening
`Inf` (default) | positive scalar in the range [1, `Inf`]

Reciprocal of flattening, specified as positive scalar in the range [1, `Inf`].

The value of inverse flattening, $1/f$, is calculated using `SemimajorAxis` value a and `SemiminorAxis` value b according to $1/f = a/(a-b)$. A value $1/f$ of `Inf` designates a perfect sphere. As $1/f$ approaches 1, the reference ellipsoid approaches a flattened disk.

When the `InverseFlattening` property is changed, other shape-related properties update, including `Eccentricity`. The `SemimajorAxis` property remains unchanged, but the value of `SemiminorAxis` adjusts to reflect the new shape.

Example: 300

Data Types: `double`

Eccentricity — First eccentricity of ellipsoid
0 (default) | nonnegative scalar in the range [0, 1]

First eccentricity of the ellipsoid, specified as nonnegative scalar in the range [0, 1].

The value of eccentricity, ecc, is calculated using `SemimajorAxis` value a and `SemiminorAxis` value b according to $ecc = \mathtt{sqrt}(a^2 - b^2)/a$. A value ecc of 0 designates a perfect sphere.

When the `Eccentricity` property is changed, other shape-related properties update, including `InverseFlattening`. The `SemimajorAxis` property remains unchanged, but the value of `SemiminorAxis` adjusts to reflect the new shape.

Example: 0.08

Data Types: `double`

Flattening — Flattening of ellipsoid
nonnegative scalar in the range [0, 1]

This property is read-only.

Flattening of the ellipsoid, specified as nonnegative scalar in the range [0, 1].

The value of flattening, *f*, is calculated using `SemimajorAxis` value *a* and `SemiminorAxis` value *b* according to *f* = (*a-b*)/*a*.

Data Types: `double`

ThirdFlattening — Third flattening of ellipsoid
nonnegative scalar in the range [0, 1]

This property is read-only.

Third flattening of the ellipsoid, specified as nonnegative scalar in the range [0, 1].

The value of the third flattening, *n*, is calculated using `SemimajorAxis` value *a* and `SemiminorAxis` value *b* according to *n* = (*a-b*)/(*a+b*).

Data Types: `double`

MeanRadius — Mean radius of the ellipsoid
positive, finite scalar

This property is read-only.

Mean radius of the ellipsoid, specified as positive, finite scalar. The `MeanRadius` property is expressed in units of length specified by `LengthUnit`.

The mean radius of the ellipsoid, *r*, is calculated using `SemimajorAxis` value *a* and `SemiminorAxis` value *b* according to *r* = (2*a+b*)/3.

Data Types: `double`

SurfaceArea — Surface area of the ellipsoid
positive, finite scalar

This property is read-only.

Surface area of the ellipsoid, specified as positive, finite scalar. The `SurfaceArea` property is expressed in units of area consistent the unit of length specified by the `LengthUnit` property.

Data Types: `double`

Volume — Volume of the ellipsoid
positive, finite scalar

This property is read-only.

Volume of the ellipsoid, specified as positive, finite scalar. The `Volume` property is expressed in units of volume consistent with the unit of length specified by the `LengthUnit` property.

Data Types: `double`

Examples

Construct GRS80 Reference Ellipsoid

Create a reference ellipsoid object by specifying the name of the ellipsoid.

```
e = referenceEllipsoid('GRS 1980')

e =
referenceEllipsoid with defining properties:

                Code: 7019
                Name: 'GRS 1980'
          LengthUnit: 'meter'
        SemimajorAxis: 6378137
        SemiminorAxis: 6356752.31414036
    InverseFlattening: 298.257222101
         Eccentricity: 0.0818191910428158

  and additional properties:

    Flattening
    ThirdFlattening
    MeanRadius
    SurfaceArea
    Volume
```

Create the same reference ellipsoid object by specifying the EPSG code 7019.

```
e = referenceEllipsoid(7019)

e =
referenceEllipsoid with defining properties:

                Code: 7019
                Name: 'GRS 1980'
          LengthUnit: 'meter'
        SemimajorAxis: 6378137
        SemiminorAxis: 6356752.31414036
    InverseFlattening: 298.257222101
         Eccentricity: 0.0818191910428158

  and additional properties:

    Flattening
    ThirdFlattening
    MeanRadius
    SurfaceArea
```

```
      Volume
```

Create a reference ellipsoid object, specifying the units. For length unit, you can specify any unit of length accepted by the `validateLengthUnit` function.

```
e = referenceEllipsoid('GRS80','km')

e =
referenceEllipsoid with defining properties:

                Code: 7019
                Name: 'GRS 1980'
           LengthUnit: 'kilometer'
        SemimajorAxis: 6378.137
        SemiminorAxis: 6356.75231414036
     InverseFlattening: 298.257222101
         Eccentricity: 0.0818191910428158

  and additional properties:

     Flattening
     ThirdFlattening
     MeanRadius
     SurfaceArea
     Volume
```

Create a reference ellipsoid, specifying the `GeoTIFFCodes.Ellipsoid` field from a file.

```
info = geotiffinfo('boston.tif');
e = referenceEllipsoid(info.GeoTIFFCodes.Ellipsoid)

e =
referenceEllipsoid with defining properties:

                Code: 7019
                Name: 'GRS 1980'
           LengthUnit: 'meter'
        SemimajorAxis: 6378137
        SemiminorAxis: 6356752.31414036
     InverseFlattening: 298.257222101
         Eccentricity: 0.0818191910428158

  and additional properties:

     Flattening
     ThirdFlattening
     MeanRadius
     SurfaceArea
     Volume
```

Create Custom Model for Planet

Create a custom reference ellipsoid by modifying the properties of a default reference ellipsoid.

```
customPlanet = referenceEllipsoid;
customPlanet.Name = 'Tiny Planet';
customPlanet.LengthUnit = 'meter';
customPlanet.SemimajorAxis = 8000;
customPlanet.SemiminorAxis = 7900

customPlanet =
referenceEllipsoid with defining properties:

              Code: []
              Name: 'Tiny Planet'
        LengthUnit: 'meter'
     SemimajorAxis: 8000
     SemiminorAxis: 7900
  InverseFlattening: 80
      Eccentricity: 0.157619002661481

  and additional properties:

    Flattening
    ThirdFlattening
    MeanRadius
    SurfaceArea
    Volume
```

More About

Names of EPSG Ellipsoids

All of the nearly 60 codes in the EPSG ellipsoid table are supported. A subset of these ellipsoids can be created by specifying their name. The short and long names of these ellipsoids, along with their corresponding code, appear in the table. You can create a reference ellipsoid by specifying either its code, short name, or long name.

EPSG Code	Short Name	Long Name
—	'unitsphere'	'Unit Sphere'
7019	'grs80'	'GRS 1980'
7030	'wgs84'	'WGS 84'
7015	'everest'	'Everest 1830 (1837 Adjustment)'
7004	'bessel'	'Bessel 1841'
7001	'airy1830'	'Airy 1830'
7002	'airy1849'	'Airy Modified 1849'
7008	'clarke66'	'Clarke 1866'
7012	'clarke80'	'Clarke 1880 (RGS)'
7022	'international'	'International 1924'
7024	'krasovsky'	'Krassowsky 1940'
7043	'wgs72'	'WGS 72'

EPSG Code	Short Name	Long Name
—	`'wgs60'`	`'World Geodetic System 1960'`
—	`'iau65'`	`'International Astronomical Union 1965'`
—	`'wgs66'`	`'World Geodetic System 1966'`
—	`'iau68'`	`'International Astronomical Union 1968'`
7030	`'earth'`	`'WGS 84'`
—	`'sun'`	`'Sun'`
—	`'moon'`	`'Moon'`
—	`'mercury'`	`'Mercury'`
—	`'venus'`	`'Venus'`
—	`'mars'`	`'Mars'`
—	`'jupiter'`	`'Jupiter'`
—	`'saturn'`	`'Saturn'`
—	`'uranus'`	`'Uranus'`
—	`'neptune'`	`'Neptune'`
—	`'pluto'`	`'Pluto'`

Tips

- When you define an ellipsoid in terms of semimajor and semiminor axes (rather than semimajor axis and inverse flattening, or semimajor axis and eccentricity), a small loss of precision in the last few digits of `Flattening`, `Eccentricity`, and `ThirdFlattening` may occur. This is unavoidable, but does not affect the results of practical computation.

Compatibility Considerations

Name property for some ReferenceEllipsoid objects has changed
Behavior changed in R2020b

Starting in R2020b, the `Name` property of `referenceEllipsoid` objects always contains the names of the ellipsoids as they appear in the EPSG Geodetic Database. In R2020a and previous releases, the value of the `Name` property depended on the name or code you used to create the object.

For example, create two `referenceEllipsoid` objects using the WGS84 reference system. Create the first object by specifying its EPSG code, and create the second object by specifying its name. In R2020b and later releases the value of the `Name` property is the same for both `referenceEllipsoid` objects. In R2020a and earlier releases the value of the `Name` property is not the same.

```
r1 = referenceEllipsoid(7030);
r2 = referenceEllipsoid('World Geodetic System 1984');
r1.Name
r2.Name
```

R2020b and later	R2020a and earlier

ans =
 	
    'WGS 84'
 	
 ans =
 	
    'WGS 84'
	ans =
	
	    'WGS 84'
	
	 ans =
	
	    'World Geodetic System 1984'

If you have existing code in which you create a reference ellipsoid object by specifying a name to the `referenceEllipsoid` creation function, you do not need to update your code to correspond to a name in the EPSG Geodetic Database.

For more information about the EPSG Geodetic Database, see the EPSG home page.

See Also
oblateSpheroid | referenceSphere | validateLengthUnit | wgs84Ellipsoid | geocrs

Introduced in R2012a

referenceSphere

Reference sphere

Description

A `referenceSphere` object represents a sphere with a specific name and radius that you can use in map projections and other geodetic operations.

Creation

Syntax

```
S = referenceSphere
S = referenceSphere(name)
S = referenceSphere(name,lengthUnit)
```

Description

`S = referenceSphere` creates a `referenceSphere` object that represents the unit sphere.

`S = referenceSphere(name)` creates a `referenceSphere` object corresponding to the specified spherical body given by `name`. The radius of the reference sphere is in meters.

`S = referenceSphere(name,lengthUnit)` creates a `referenceSphere` object with radius in the specified unit of length, `LengthUnit`. The unit of length can be any length unit supported by the `validateLengthUnit` function.

Input Arguments

name — Name of predefined reference sphere
'Unit Sphere' (default) | 'earth' | 'sun' | 'moon' | ...

Name of predefined reference sphere, specified as `'earth'`, `'sun'`, `'moon'`, `'mercury'`, `'venus'`, `'mars'`, `'jupiter'`, `'saturn'`, `'uranus'`, `'neptune'`, `'pluto'`, or `'unit sphere'`. The `name` argument is not case-sensitive.

Data Types: `char` | `string`

lengthUnit — Length unit
'' (default) | character vector | string scalar

Length unit, specified as a character vector or string scalar. You can specify any length unit that is supported by the `validateLengthUnit` function.

Data Types: `char` | `string`

Properties

Name — Name of reference sphere
`'Unit Sphere'` (default) | character vector | string scalar

Name of the reference sphere, specified as a string scalar or character vector.

Example: `'Sun'`

Data Types: `char` | `string`

LengthUnit — Unit of length of the radius
`''` (default) | character vector | string scalar

Unit of length of the radius, specified as a string scalar or character vector. You can specify any unit of length accepted by the `validateLengthUnit` function. When the reference sphere represents the unit sphere, `LengthUnit` is the empty character vector `' '`.

Example: `'km'`

Data Types: `char` | `string`

Radius — Radius of the sphere
positive, finite scalar

Radius of the sphere, specified as positive, finite scalar.

Data Types: `double`

SemimajorAxis — Equatorial radius of the sphere
1 (default) | positive, finite scalar

This property is read-only.

Equatorial radius of the sphere, specified as a positive, finite scalar. The value of `SemimajorAxis` is equal to the value of `Radius`.

Data Types: `double`

SeminimorAxis — Distance from center of sphere to pole
1 (default) | positive, finite scalar

This property is read-only.

Distance from center of sphere to pole, specified as a positive, finite scalar. The value of `SeminimorAxis` is equal to the value of `Radius`.

Data Types: `double`

InverseFlattening — Reciprocal of flattening
`Inf` (default)

This property is read-only.

Reciprocal of flattening, specified as the value `Inf`.

Data Types: `double`

Eccentricity — First eccentricity of sphere
0 (default)

This property is read-only.

First eccentricity of the sphere, specified as the value 0.

Data Types: `double`

Flattening — Flattening of sphere
0 (default)

This property is read-only.

Flattening of the sphere, specified as the value 0.

Data Types: `double`

ThirdFlattening — Third flattening of sphere
0 (default)

This property is read-only.

Third flattening of the sphere, specified as the value 0.

Data Types: `double`

MeanRadius — Mean radius of the sphere
positive, finite scalar

This property is read-only.

Mean radius of the sphere, specified as positive, finite scalar. The value of `MeanRadius` is equal to the value of `Radius`.

Data Types: `double`

SurfaceArea — Surface area of the sphere
positive, finite scalar

This property is read-only.

Surface area of the sphere, specified as positive, finite scalar.

The `SurfaceArea` property is expressed in units of area consistent with the unit of length specified by the `LengthUnit` property.

Data Types: `double`

Volume — Volume of the sphere
positive, finite scalar

This property is read-only.

Volume of the sphere, specified as positive, finite scalar.

The `Volume` property is expressed in units of volume consistent with the unit of length specified by the `LengthUnit` property.

Data Types: double

Examples

Create Model of Earth in Kilometers

Construct a reference sphere that models the Earth as a sphere with a radius of 6371000 meters. Note that the unit of length is meters.

```
s = referenceSphere('Earth')

s =

referenceSphere with defining properties:

          Name: 'Earth'
    LengthUnit: 'meter'
        Radius: 6371000

  and additional properties:

    SemimajorAxis
    SeminimorAxis
    InverseFlattening
    Eccentricity
    Flattening
    ThirdFlattening
    MeanRadius
    SurfaceArea
    Volume
```

Switch the unit of length in the reference sphere to kilometers.

```
s.LengthUnit = 'kilometer'

s =

referenceSphere with defining properties:

          Name: 'Earth'
    LengthUnit: 'kilometer'
        Radius: 6371

  and additional properties:

    SemimajorAxis
    SeminimorAxis
    InverseFlattening
    Eccentricity
    Flattening
    ThirdFlattening
    MeanRadius
    SurfaceArea
    Volume
```

Determine the surface area of the sphere in square kilometers.

```
s.SurfaceArea

ans =

   5.1006e+08
```

Find the volume of the sphere in cubic kilometers.

```
s.Volume

ans =

   1.0832e+12
```

Create Custom Model for Planet

Create a custom reference sphere for a planet by modifying the properties of a default reference sphere.

```
customPlanet = referenceSphere;
customPlanet.Name = 'Tiny Planet';
customPlanet.LengthUnit = 'meter';
customPlanet.Radius = 8000

customPlanet =

referenceSphere with defining properties:

         Name: 'Tiny Planet'
   LengthUnit: 'meter'
       Radius: 8000

  and additional properties:

    SemimajorAxis
    SemiminorAxis
    InverseFlattening
    Eccentricity
    Flattening
    ThirdFlattening
    MeanRadius
    SurfaceArea
    Volume
```

See Also
oblateSpheroid | referenceEllipsoid | validateLengthUnit

Introduced in R2012a

refine

Refine search of WMS layers

Syntax

```
refined = refine(layers,querystr)
refined = refine( ___ ,Name,Value,...)
```

Description

`refined = refine(layers,querystr)` searches fields of Web map service layers, `layers`, for a partial match with the string or character vector in *querystr*. By default, `refine` searches the `Layer` or `LayerName` properties but you can include other fields in the search using the `SearchFields` parameter.

`refined = refine( ___ ,Name,Value,...)` modifies the search based on the values of the named parameters.

Examples

Refine Search by Text

Find layers in the WMS Database that contain information about temperature.

```
temperature = wmsfind('temperature');
```

Refine the search of temperature layers to find only those layers that contain annual temperature information.

```
annual = refine(temperature,'annual');
```

Refine the search of temperature layers to find layers containing only sea surface temperatures.

```
sst = refine(temperature,'sea surface');
```

Refine the search of sea surface temperature layers to include only layers that include annual information.

```
annual_and_sst = refine(sst,'annual');
annual_or_sst = [sst;annual];
```

Input Arguments

`layers` — Layers to search
array of `WMSLayer` objects

Layers to search, specified as an array of `WMSLayer` objects.

querystr — Characters to search for in `WMSLayer` object fields
character vector | string

Characters to search for in `WMSLayer` object fields, specified as a string or character vector. `querystr` can contain the asterisk wildcard character (*).

Example: `'temperature'`

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'IgnoreCase',false`

SearchFields — Fields to search in the `WMSLayer` objects
`'layer'` (default) | `'server'` | `'layertitle'` | `'layername'` | `'servertitle'` | `'serverurl'` | `'any'` | `'abstract'`

Fields to search in the `WMSLayer` objects, specified as a string, string array, character vector, or cell array of character vectors. The function searches the values of the specified fields in the `WMSLayer` objects for a partial or exact match with `querystr`. Multiple options may be included in a string array or cell array of character vectors.

The table lists valid values of `searchFields`.

Field	Behavior
`'layer'`	Search both the `LayerTitle` and the `LayerName` fields.
`'layername'`	Search the `LayerName` field in the `WMSLayer` objects. The layer name is an abbreviated form of the `LayerTitle` field and is the keyword the server uses to retrieve the layer.
`'layertitle'`	Search the `LayerTitle` field in the `WMSLayer` objects. The layer title includes descriptive information about a layer and facilitates understanding the meaning of the raster values of the layer.
`'server'`	Search the `ServerURL` in the `WMSLayer` objects. The server URL and layer information facilitate the reading of raster layers by the function `wmsread`.
`'servertitle'`	Search the `ServerTitle` field in the `WMSLayer` objects. A server title includes descriptive information about the server.
`'serverurl'`	Search the `ServerURL` in the `WMSLayer` objects. The server URL and layer information facilitate the reading of raster layers by the function `wmsread`.
`'abstract'`	Search the `abstract` field in the `WMSLayer` objects.
`'any'`	Search all fields.

Data Types: `char` | `string`

MatchType — Strictness of match
`'partial'` (default) | `'exact'`

Strictness of match, specified as the string or character vector `'partial'` or `'exact'`. If `'MatchType'` is `'exact'` and `querystr` is `'*'`, a match occurs when the search field matches the character `'*'`.

Data Types: `char` | `string`

IgnoreCase — Ignore case when comparing field values to `querystr`
`true` (default) | `false`

Ignore case when comparing field values to `querystr`, specified as the logical value `true` or `false`.

Data Types: `logical`

Output Arguments

`refined` — Refined layers
array of `WMSLayer` objects

Refined layers, returned as an array of `WMSLayer` objects. Each layer in the array has a searched field that matches the text query, `querystr`.

See Also
`wmsfind` | `refineLimits`

Introduced in R2009b

refineLimits

Refine search of WMS layers based on geographic limits

Syntax

```
refined = refineLimits(layers,Name,Value,...)
```

Description

`refined = refineLimits(layers,Name,Value,...)` searches for elements of Web map service layers, `layers`, that match specific latitude or longitude limits. The results include a given layer only if the quadrangle specified by the optional `'Latlim'` and `'Lonlim'` parameters fully contains the boundary quadrangle, as defined by the `Latlim` and `Lonlim` properties. Partial overlap does not result in a match. All angles are in units of degrees.

Examples

Refine Search by Geographic Limits

Find layers in the WMS Database that contain information about hurricanes.

```
layers = wmsfind('hurricane');
num_layers = length(layers)
```

```
num_layers = 79
```

Refine the search by finding layers that are in the western hemisphere.

```
refined = refineLimits(layers, ...
    'Latlim',[-90 90],'Lonlim',[-180 0]);
num_refined = length(refined)
```

```
num_refined = 32
```

Note that the refined array contains fewer layers.

Your results might be different because the WMS Database changes each release.

Find Layers Containing Global Elevation Data

Search the WMS Database for layers containing elevation data.

```
elevation = wmsfind('elevation');
```

Refine the search by specifying latitude and longitude limits.

```
latlim = [-90, 90];
lonlim = [-180, 180];
```

```
globalElevation = ...
    refineLimits(elevation,'Latlim',latlim,'Lonlim',lonlim);
```

Display the server titles as a column vector. Your results might be different because the WMS Database changes each release.

```
serverTitles(globalElevation)'
```

```
ans = 7x1 cell
    {'NASA WorldWind WMS'            }
    {'CubeSERV WMS'                  }
    {'NASA WorldWind WMS'            }
    {'NASA Earth Observations (NEO) WMS'}
    {'NASA Earth Observations (NEO) WMS'}
    {'World Map'                     }
    {'World Map'                     }
```

Input Arguments

layers — Layers to search
array of `WMSLayer` object

Layers to search, specified as an array of `WMSLayer` objects.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Latlim',[0 90]`

Latlim — Latitudinal limits to search
two-element vector

Latitudinal limits to search, specified as a two-element vector. `Latlim` is in the form `[southern_limit northern_limit]` or a scalar value representing the latitude of a single point.

Example: `[-90, 90]`

Lonlim — Longitudinal limits to search
numeric scalar | two-element numeric vector

Longitudinal limits to search, specified as a numeric scalar or two-element numeric vector. `Lonlim` is in the form `[western_limit eastern_limit]` or a scalar value representing the longitude of a single point.

Example: `[-180, 180]`

Output Arguments

refined — Refined layers
array of `WMSLayer` objects

Refined layers, specified as an array of `WMSLayer` objects. Each layer in the array has a boundary quadrangle that is fully contained in the quadrangle defined by the specified `'Latlim'` and `'Lonlim'` parameters.

Tips

- The default value of `[]` for either `'Latlim'` or `'Lonlim'` implies that all layers match the criteria. For example, if you specify the following, then the results include all the layers that cover the northern hemisphere.

  ```
  refineLimits(layer,'Latlim', [0 90], 'Lonlim', [])
  ```

See Also

wmsfind | refine

Introduced in R2009b

refmat2vec

(To be removed) Convert referencing matrix to referencing vector

---

Note `refmat2vec` will be removed in a future release. Instead, convert referencing matrices to geographic raster reference objects using the `refmatToGeoRasterReference` function. For more information, see "Compatibility Considerations".

Syntax

```
refvec = refmat2vec(R,s)
```

Description

`refvec = refmat2vec(R,s)` converts a referencing matrix, R, to the three-element referencing vector `refvec`. R is a 3-by-2 referencing matrix defining a two-dimensional affine transformation from pixel coordinates to spatial coordinates. `s` is the size of the array (data grid) that is being referenced. `refvec` is a 1-by-3 referencing vector having elements `[cells/degree north-latitude west-longitude]` with latitude and longitude limits specified in degrees.

Examples

```
% Convert a sample referencing matrix to a
% referencing vector.
N = rand(180,360);
refmat = [0 1; 1 0; -0.5 -90.5];
V = refmat2vec(refmat,size(N));
```

Compatibility Considerations

refmat2vec will be removed
Not recommended starting in R2013b

Some functions that return referencing vectors will be removed, including the `refmat2vec` function. Instead, convert referencing matrices to geographic raster reference objects using the `refmatToGeoRasterReference` function. Reference objects have several advantages over referencing vectors.

- Unlike referencing vectors, reference objects have properties that document the size of the associated raster, its geographic limits, and the direction of its rows and columns. For examples of reference object properties, see the `GeographicPostingsReference` object.

- You can manipulate the limits of rasters associated with geographic reference objects using the `geocrop` function.

- You can manipulate the size and resolution of rasters associated with geographic reference objects using the `georesize` function.

- Most functions that accept referencing vectors as inputs also accept reference objects.

To update your code, replace instances of the `refmat2vec` function with the `refmatToGeoRasterReference` function.

```
rasterRef = refmatToGeoRasterReference(refmat,rasterSize);
```

See Also
`refmatToGeoRasterReference`

Introduced before R2006a

refvec2mat

(To be removed) Convert referencing vector to referencing matrix

---

Note `refvec2mat` will be removed in a future release. Instead, convert referencing vectors to geographic raster reference objects using the `refvecToGeoRasterReference` function. For more information, see "Compatibility Considerations".

Syntax

```
R = refvec2mat(refvec,s)
```

Description

`R = refvec2mat(refvec,s)` converts a referencing vector, `refvec`, to the referencing matrix R. `refvec` is a 1-by-3 referencing vector having elements `[cells/degree north-latitude west-longitude]` with latitude and longitude limits specified in degrees. `s` is the size of the array (data grid) that is being referenced. `R` is a 3-by-2 referencing matrix defining a two-dimensional affine transformation from pixel coordinates to spatial coordinates.

Examples

```
% Create a sample data set and referencing vector
N = rand(180,360);
refvec = [1 90 0];
% Convert the referencing vector to a referencing matrix
R = refvec2mat(refvec,size(N));
```

Compatibility Considerations

refvec2mat will be removed
Not recommended starting in R2013b

Some functions that return referencing matrices will be removed, including the `refvec2mat` function. Instead, convert referencing vectors to geographic raster reference objects using the `refvecToGeoRasterReference` function. Reference objects have several advantages over referencing matrices.

- Unlike referencing matrices, reference objects have properties that document the size of the associated raster, its geographic limits, and the direction of its rows and columns. For examples of reference object properties, see the `GeographicPostingsReference` object.

- You can manipulate the limits of rasters associated with geographic reference objects using the `geocrop` function.

- You can manipulate the size and resolution of rasters associated with geographic reference objects using the `georesize` function.

- Most functions that accept referencing matrices as inputs also accept reference objects.

To update your code, replace instances of the `refvec2mat` function with the `refvecToGeoRasterReference` function.

```
R = refvecToGeoRasterReference(refvec,rasterSize);
```

See Also
`refvecToGeoRasterReference`

Introduced before R2006a

refmatToGeoRasterReference

Referencing matrix to geographic raster reference object

Syntax

```
R = refmatToGeoRasterReference(refmat,rasterSize)
R = refmatToGeoRasterReference( ___ ,rasterInterpretation)
R = refmatToGeoRasterReference( ___ ,funcName,varName,argIndex)
R = refmatToGeoRasterReference(Rin,rasterSize, ___ )
```

Description

`R = refmatToGeoRasterReference(refmat,rasterSize)` constructs a cell-oriented geographic raster reference object, R, from a referencing matrix, `refmat`, and a size vector, `rasterSize`.

`R = refmatToGeoRasterReference( ___ ,rasterInterpretation)` uses the `rasterInterpretation` input to determine which type of geographic raster reference object to construct. The `rasterInterpretation` input indicates basic geometric nature of the raster, and can equal either `'cells'` or `'postings'`.

`R = refmatToGeoRasterReference( ___ ,funcName,varName,argIndex)` uses up to three optional arguments to provide additional information. This information is used to construct error messages if either the `refmat` or `rasterSize` inputs turn out to be invalid. Thus, you can use `refmatToGeoRasterReference` for both validating and converting a referencing matrix. The optional inputs work just like their counterparts in the function `validateattributes`.

`R = refmatToGeoRasterReference(Rin,rasterSize, ___ )` verifies that size of the geographic raster reference object, `Rin` is consistent with the size specified by `rasterSize`, and then copies `Rin` to R. `refmatToGeoRasterReference` gets size information from the `Rin.RasterSize` property.

Input Arguments

refmat

Any valid referencing matrix. The matrix must lead to valid latitude and longitude limits when combined with `rasterSize`, and the matrix columns and rows must be aligned with meridians and parallels, respectively.

rasterSize

Size vector `[M N ...]` specifying the number of rows (M) and columns (N) in the raster or image to be associated with the geographic raster reference object, R. For convenience, `rasterSize` may be a row vector with more than two elements. This flexibility allows you to specify the size in the following way:

```
R = refmatToGeoRasterReference(refmat, size(RGB))
```

where RGB is *M*-by-*N*-by-3. However, in such cases, only the first two elements of the size vector are actually used. The higher (non-spatial) dimensions are ignored.

rasterInterpretation

Basic geometric nature of the raster, specified as either `'cells'` or `'postings'`.

funcName

Name used in the formatted error message to identify the function checking the input, specified as a character vector.

varName

Name used in the formatted error message to identify the referencing matrix, specified as a character vector.

argIndex

Positive integer that indicates the position of the referencing matrix checked in the function argument list. `refmatToGeoRasterReference` includes this information in the formatted error message.

Rin

Geographic raster reference object.

Output Arguments

R

Geographic raster reference object.

Examples

Convert a referencing matrix to a geographic raster reference object:

```
% Specify the size of a sample raster and referencing matrix.
rasterSize = [180 360];
refmat = [0 1; 1 0; -0.5 -90.5];

% Convert the referencing matrix to a
% geographic raster reference object.
R = refmatToGeoRasterReference(refmat,rasterSize);

% For comparison, construct a referencing object directly.
R2 = georasterref( ...
    'RasterSize',rasterSize,'Latlim',[-90 90],'Lonlim',[0 360]);
```

See Also
georasterref | refvecToGeoRasterReference

Introduced in R2011a

refmatToMapRasterReference

Referencing matrix to map raster reference object

Syntax

```
R = refmatToMapRasterReference(refmat,rasterSize)
R = refmatToMapRasterReference( ___ ,rasterInterpretation)
R = refmatToMapRasterReference( ___ ,func_name, var_name, arg_pos)
R = refmatToMapRasterReference(Rin,rasterSize, ___ )
```

Description

`R = refmatToMapRasterReference(refmat,rasterSize)` constructs a map raster reference object, R, from a referencing matrix, `refmat`, and a size vector, `rasterSize`.

`R = refmatToMapRasterReference( ___ ,rasterInterpretation)` uses the `rasterInterpretation` input to determine which type of map raster reference object to construct. The `rasterInterpretation` input indicates basic geometric nature of the raster, and can equal either `'cells'` or `'postings'`.

`R = refmatToMapRasterReference( ___ ,func_name, var_name, arg_pos)` uses up to three optional arguments to provide additional information. This information is used to construct error messages if either the `refmat` or `rasterSize` inputs turn out to be invalid. Thus, you can use `refmatToMapRasterReference` for both validating and converting a referencing matrix. The optional inputs work just `like` their counterparts in the function `validateattributes`.

`R = refmatToMapRasterReference(Rin,rasterSize, ___ )` verifies that `Rin.RasterSize` is consistent with `rasterSize`, then copies Rin to R.

Input Arguments

refmat

Referencing matrix

rasterSize

Size vector [M N ...] specifying the number of rows (*M*) and columns (*N*) in the raster or image to be associated with the `MapRasterReference` object, R. For convenience, `rasterSize` may be a row vector with more than two elements. This flexibility allows you to specify the size in the following way:

```
R = refmatToMapRasterReference(refmat, size(RGB))
```

where RGB is *M*-by-*N*-by-3. However, in such cases, only the first two elements of the size vector are actually used. The higher (non-spatial) dimensions are ignored.

rasterInterpretation

Basic geometric nature of the raster, specified as either `'cells'` or `'postings'`.

func_name

Name used in the formatted error message to identify the function checking the input, specified as a character vector.

var_name

Name used in the formatted error message to identify the referencing matrix, specified as a character vector.

arg_pos

Positive integer that indicates the position of the referencing matrix checked in the function argument list. refmatToMapRasterReference includes this information in the formatted error message.

Rin

Map raster reference object.

Output Arguments

R

Map raster reference object.

Examples

Convert a referencing matrix manually versus using the maprasterref function.

```
% Create a sample referencing matrix for a 2000-by-2000
% orthoimage referenced to the Massachusetts State
% Plane Mainland coordinate system.
refmat = [0 -1; 1 0; 208999.5 913000.5];

% Import the corresponding TIFF image and use its size to
% help convert the referencing matrix to a referencing object.
[X, cmap] = imread('concord_ortho_e.tif');
R = refmatToMapRasterReference(refmat, size(X));

% Obtain the map limits.
xLimWorld = R.XWorldLimits;
yLimWorld = R.YWorldLimits;

% Construct a referencing object directly, for comparison.
R2 = maprasterref('RasterSize',size(X),'ColumnsStartFrom','north', ...
   'XLimWorld',xLimWorld,'YLimWorld',yLimWorld);
```

See Also
maprasterref | refmatToGeoRasterReference

Introduced in R2011a

refmatToWorldFileMatrix

Convert referencing matrix to world file matrix

Syntax

```
W = refmatToWorldFileMatrix(refmat)
```

Description

`W = refmatToWorldFileMatrix(refmat)` converts the 3-by-2 referencing matrix `refmat` to a 2-by-3 world file matrix `W`.

For a definition of a world file matrix, see the `worldFileMatrix` method of the map raster reference and geographic raster reference classes.

See Also

georasterref | maprasterref | georefpostings | georefcells

Introduced in R2011a

refvecToGeoRasterReference

Referencing vector to geographic raster reference object

Syntax

```
R = refvecToGeoRasterReference(refvec,rasterSize)
R = refvecToGeoRasterReference( ___ ,funcName,varName,argIndex)
R = refvecToGeoRasterReference(Rin,rasterSize, ___ )
```

Description

`R = refvecToGeoRasterReference(refvec,rasterSize)` constructs a geographic raster reference object, R, from a referencing vector, `refvec`, and a size vector, `rasterSize`.

`R = refvecToGeoRasterReference( ___ ,funcName,varName,argIndex)` uses up to three optional arguments to provide additional information. This information is used to construct error messages if either the `refvec` or `rasterSize` inputs turn out to be invalid. Thus, you can use `refvecToGeoRasterReference` for both validating and converting a referencing vector. The optional inputs work just like their counterparts in the MATLAB function `validateattributes`.

`R = refvecToGeoRasterReference(Rin,rasterSize, ___ )` verifies that `Rin.RasterSize` is consistent with `rasterSize`, then copies `Rin` to R.

Input Arguments

refvec

Any valid 1-by-3 referencing vector, as long as the cell size `1/refvec(1)`, northwest corner latitude `refvec(2)`, and northwest corner longitude `refvec(3)` lead to valid latitude and longitude limits when combined with the `rasterSize` vector.

rasterSize

Size vector `[M N  ...]` specifying the number of rows (M) and columns (N) in the raster or image to be associated with the `GeoRasterReference` object, R.

funcName

Name used in the formatted error message to identify the function checking the input, specified as a character vector.

varName

Name used in the formatted error message to identify the referencing vector, specified as a character vector.

argIndex

Positive integer that indicates the position of the referencing vector checked in the function argument list. `refvecToGeoRasterReference` includes this information in the formatted error message.

Rin

Geographic raster reference object.

Output Arguments

R

Geographic raster reference object.

Examples

Convert a referencing vector manually versus using the `georasterref` function.

```
% Construct a referencing vector for a regular 180-by-240 grid
% covering an area that includes the Korean Peninsula, with 12 cells
% per degree.
refvec = [12  45  115];

% Convert to a geographic raster reference object:
rasterSize = [180 240];
R = refvecToGeoRasterReference(refvec,rasterSize);

% Construct a reference object with the same limits.
latlim = R.LatitudeLimits;
lonlim = R.LongitudeLimits;
R2 = georasterref('RasterSize',rasterSize,'Latlim',latlim, ...
    'Lonlim',lonlim);
```

See Also
georasterref

Introduced in R2011a

removeCustomBasemap

Remove custom basemap

Syntax

```
removeCustomBasemap(basemapName)
```

Description

removeCustomBasemap(basemapName) removes the custom basemap specified by basemapName from the list of available basemaps.

If the custom basemap specified by basemapName has not been previously added using the addCustomBasemap function, the removeCustomBasemap function returns an error.

Examples

Remove Custom Basemap

Add a custom basemap to view locations as bubbles on an OpenStreetMap® basemap.

```
name = 'openstreetmap';
url = 'a.tile.openstreetmap.org';
copyright = char(uint8(169));
attribution = copyright + "OpenStreetMap contributors";
addCustomBasemap(name,url,'Attribution',attribution)
```

Use the custom basemap with a geographic bubble chart.

```
lat = [42.3501 42.3515 42.3598 42.3584 42.3529];
lon = [-71.0870 -71.0926 -71.0662 -71.0598 -71.0662];
gb = geobubble(lat,lon,'Basemap','openstreetmap', ...
    'BubbleWidthRange',25,'MapLayout','maximized', ...
    'ZoomLevel',14);
```
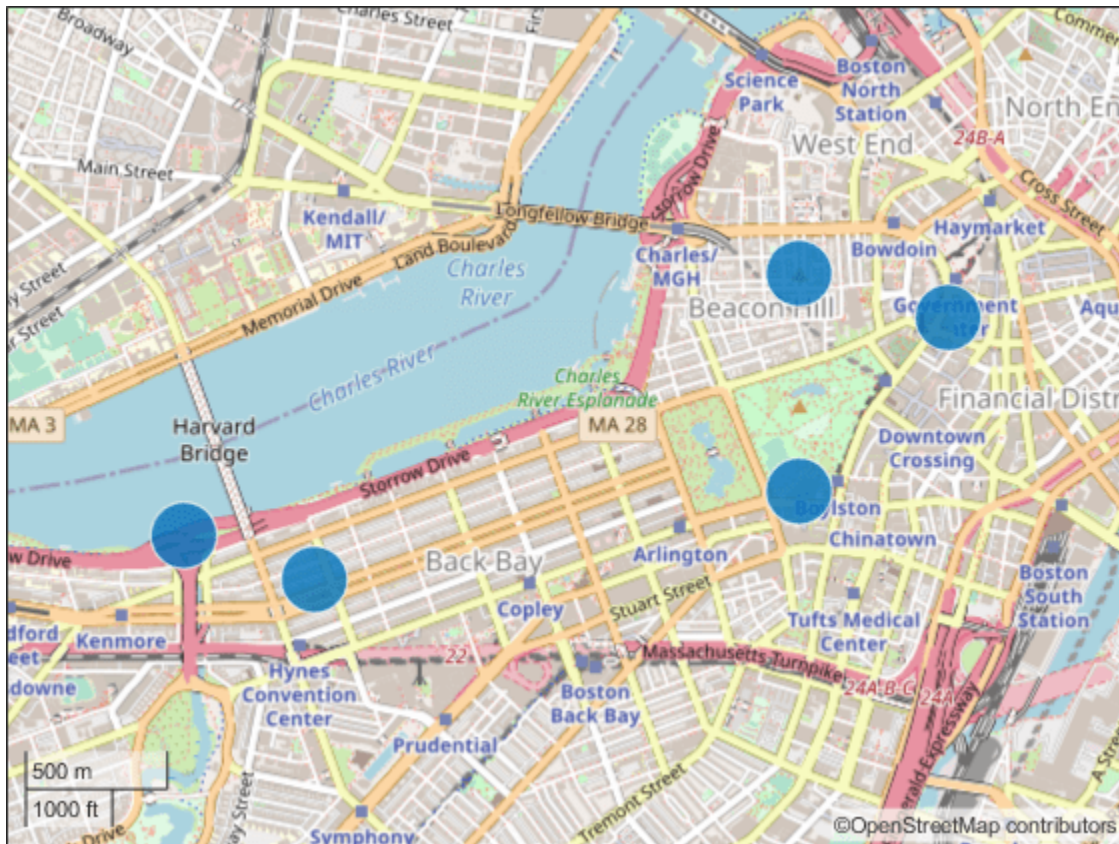
Remove the custom basemap.

```
removeCustomBasemap(name)
```

Input Arguments

basemapName — Name of custom basemap
string scalar | character vector

Name of the custom basemap to remove, specified as a string scalar or character vector. You define the basemap name when you add the basemap using the `addCustomBasemap` function.

Data Types: `string` | `char`

See Also
`geoaxes` | `geobasemap` | `geobubble` | `geodensityplot` | `geoplot` | `geoscatter` | `addCustomBasemap` | `webmap`

Introduced in R2018b

removeCustomTerrain

Remove custom terrain data

Syntax

```
removeCustomTerrain(terrainName)
```

Description

removeCustomTerrain(terrainName) removes the custom terrain data specified by the user-defined terrainName. You can use this function to remove terrain data that is no longer needed. The terrain data to be removed must have been previously added using addCustomTerrain.

Examples

Display Custom Terrain Using Geographic Globe

Display a line from the surface of Gross Reservoir to a point above South Boulder Peak using custom terrain.

First, add terrain for an area around South Boulder Peak by calling addCustomTerrain and specifying a DTED file. The DTED file is from the SRTM Void Filled data set, available from the US Geological Survey. Name the terrain 'southboulderpeak'.

```
addCustomTerrain('southboulderpeak','n39_w106_3arc_v2.dt1')
```

Create a geographic globe. Specify the terrain by name, using the 'Terrain' argument of the geoglobe function. Preserve the terrain after plotting by calling the hold function. Then, plot the line. Tilt the view by holding **Ctrl** and dragging.

```
uif = uifigure;
g = geoglobe(uif,'Terrain','southboulderpeak');
hold(g,'on')

lat = [39.95384 39.95];
lon = [-105.29916 -105.3608];
hTerrain = [10 0];
geoplot3(g,lat,lon,hTerrain,'y','HeightReference','Terrain', ...
     'LineWidth',3)
```

Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community

Close the geographic globe and remove the custom terrain.

```
close(uif)
removeCustomTerrain('southboulderpeak')
```

Input Arguments

terrainName — User-defined identifier for terrain data
string scalar | character vector

User-defined identifier for terrain data previously added using `addCustomTerrain`, specified as a string scalar or a character vector.

Data Types: `char` | `string`

See Also
`addCustomTerrain` | `geoglobe`

Topics
"Access Basemaps and Terrain for Geographic Globe"

Introduced in R2020a

removeExtraNanSeparators

Clean up NaN separators in polygons and lines

Syntax

```
[xdata, ydata] = removeExtraNanSeparators(xdata,ydata)
[xdata, ydata, zdata] = removeExtraNanSeparators(xdata,ydata,zdata)
```

Description

[xdata, ydata] = removeExtraNanSeparators(xdata,ydata) removes NaNs from the vectors xdata and ydata, leaving only isolated NaN separators. If present, one or more leading NaNs are removed entirely. If present, a single trailing NaN is preserved. NaNs are removed, but never added, so if the input lacks a trailing NaN, so will the output. xdata and ydata must match in size and have identical NaN locations.

[xdata, ydata, zdata] = removeExtraNanSeparators(xdata,ydata,zdata) removes NaNs from the vectors xdata, ydata, and zdata, leaving only isolated NaN separators and optionally, if consistent with the input, a single trailing NaN.

Examples

```
xin = [NaN NaN 1:3 NaN 4:5 NaN NaN NaN 6:9 NaN NaN];
yin = xin;
[xout, yout] = removeExtraNanSeparators(xin, yin);
xout

xout =
  1   2   3   NaN   4   5 NaN   6   7   8   9   NaN

xin = [NaN 1:3 NaN NaN 4:5 NaN NaN NaN 6:9]'
yin = xin;
zin = xin;
[xout, yout, zout] = removeExtraNanSeparators(xin, yin, zin);
xout

xout =
     1
     2
     3
   NaN
     4
     5
   NaN
     6
     7
     8
     9
```

Introduced in R2006a

removeToolbarMapButton

Remove map button from toolbar

Syntax

```
removeToolbarMapButton(tb)
removeToolbarMapButton(tb,"basemap")
```

Description

removeToolbarMapButton(tb) removes all map buttons from the axes toolbar tb.

removeToolbarMapButton(tb,"basemap") remove the basemap picker from the axes toolbar.

Examples

Add and Remove Map Buttons from Toolbar

Create a figure by using the uifigure function. Place a geographic axes in the figure.

```
uif = uifigure;
gx = geoaxes(uif);
```

Create a custom axes toolbar. Add all map buttons to the toolbar.

```
tb = axtoolbar(gx,"default");
addToolbarMapButton(tb)
```

Remove all map buttons from the toolbar.

```
removeToolbarMapButton(tb)
```
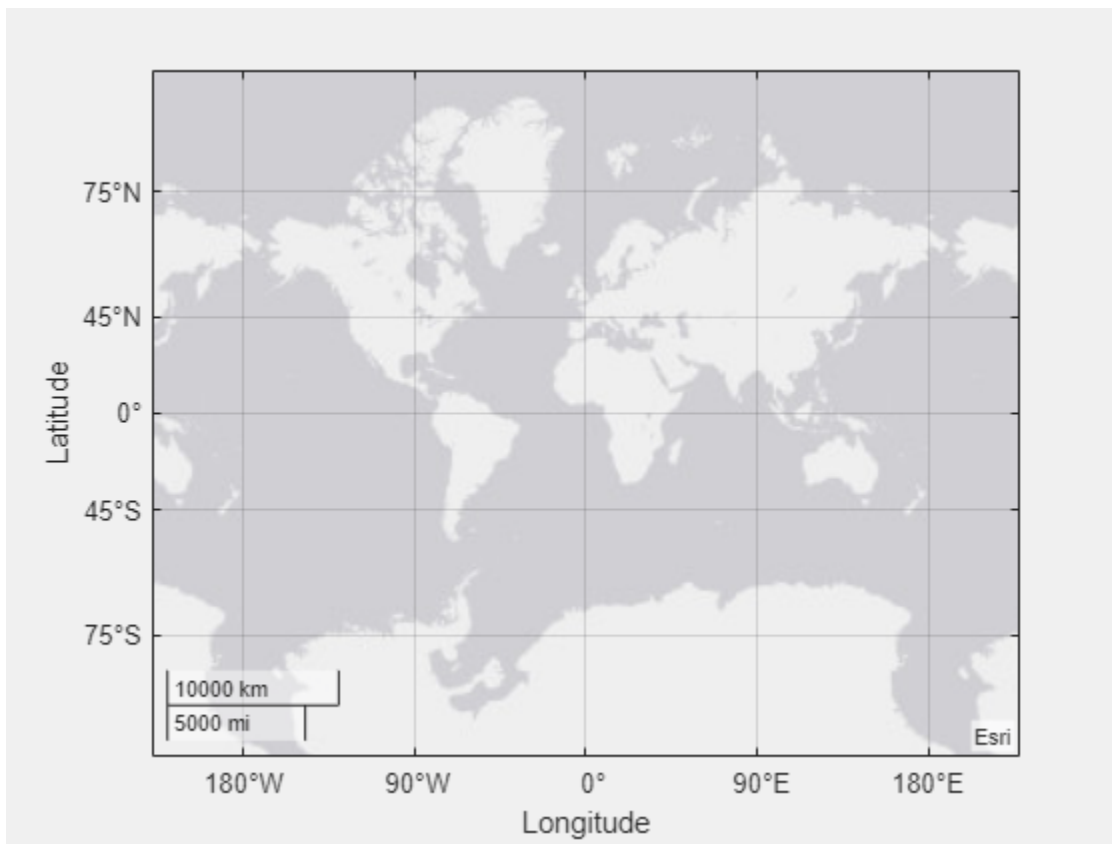
Input Arguments

tb — Axes toolbar
AxesToolbar object | array of AxesToolbar objects

Axes toolbar, specified as an AxesToolbar object or an array of AxesToolbar objects.

See Also

Functions
addToolbarMapButton | axtoolbar | geoaxes | geobasemap

Topics
"Access Basemaps for Geographic Axes and Charts"

Introduced in R2021b

resizem

(To be removed) Resize regular data grid

Note `resizem` will be removed in a future release. Use the `georesize` or `imresize` function instead. For more information, see "Compatibility Considerations".

Syntax

```
Zscaled = resizem(Z,scale)
Zscaled = resizem(Z,[numrows numcols])
[Zscaled,Rscaled] = resizem(Z,scale,R)
[Zscaled,Rscaled] = resizem(Z,[numrows numcols],R)
[ ___ ] = resizem( ___ ,method)
[ ___ ] = resizem( ___ ,method,n)
[ ___ ] = resizem( ___ ,h)
```

Description

`Zscaled = resizem(Z,scale)` returns a regular data grid `Zscaled` that is `scale` times the size of the input, `Z`. `resizem` uses interpolation to resample to a new sample density (cell size). By default, `resizem` uses nearest neighbor interpolation.

`Zscaled = resizem(Z,[numrows numcols])` resizes `Z` to have `numrows` rows and `numcols` columns.

`[Zscaled,Rscaled] = resizem(Z,scale,R)` and

`[Zscaled,Rscaled] = resizem(Z,[numrows numcols],R)` resizes a regular data grid that is spatially referenced by `R`.

`[ ___ ] = resizem( ___ ,method)` specifies alternate interpolation methods.

`[ ___ ] = resizem( ___ ,method,n)` applies a low-pass filter of size n-by-n before bilinear or bicubic interpolation to reduce aliasing.

`[ ___ ] = resizem( ___ ,h)` applies 2-D FIR filter h to the data grid before resizing, for all interpolation methods.

Examples

Resize Regular Data Grid

Define a sample data grid.

```
Z = [1 2; 3 4]
```

Z = 2×2

```
     1     2
     3     4
```

Double the size of the grid using nearest neighbor interpolation.

```
neargrid = resizem(Z,2)
```

neargrid = *4×4*

```
     1     1     2     2
     1     1     2     2
     3     3     4     4
     3     3     4     4
```

Double the size of the grid using bilinear interpolation.

```
bilingrid = resizem(Z,2,'bilinear')
```

bilingrid = *4×4*

```
    1.0000    1.3333    1.6667    2.0000
    1.6667    2.0000    2.3333    2.6667
    2.3333    2.6667    3.0000    3.3333
    3.0000    3.3333    3.6667    4.0000
```

Resize the grid to have three rows and two columns using bicubic interpolation.

```
bicubgrid = resizem(bilingrid,[3 2],'bicubic')
```

bicubgrid = *3×2*

```
    0.7406    1.2994
    1.6616    2.3462
    1.9718    2.5306
```

Input Arguments

Z — Regular data grid
M-by-*N* numeric array

Regular data grid, specified as an *M*-by-*N* numeric array that may contain NaN values. Z is either a georeferenced data grid, or a regular data grid associated with a geographic reference R.

scale — Resizing scale factor
positive scalar

Resizing scale factor, specified as a positive scalar. If scale is between 0 and 1, then the size of Zscaled is smaller than the size of Z. If scale is greater than 1, then the size of Zscaled is larger. For example, if scale is 0.5, then the number of rows and the number of columns are halved.

[numrows numcols] — Output grid size
1-by-2 vector of positive integers

Output grid size, specified as a 1-by-2 vector of positive integers.

R — Geographic reference
geographic raster reference object | vector | matrix

Geographic reference, specified as one of the following. For more information about referencing vectors and matrices, see "Georeferenced Raster Data".

Type	Description
Geographic raster reference object	`GeographicCellsReference` geographic raster reference object that relates the subscripts of Z to geographic coordinates. The `RasterSize` property must be consistent with the size of the data grid, `size(Z)`. The `RasterInterpretation` must be `'cells'`.
Vector	1-by-3 numeric vector with elements: `[cells/degree northern_latitude_limit western_longitude_limit]` **Note** When R is a referencing vector, then the argument `[nrows ncols]` is not supported and the resizing factor `scale` must be a scalar.
Matrix	3-by-2 numeric matrix that transforms raster row and column indices to or from geographic coordinates according to: `[lon lat] = [row col 1] * R` R defines a (non-rotational, non-skewed) relationship in which each column of the data grid falls along a meridian and each row falls along a parallel. Nearest-neighbor interpolation is used by default. `NaN` is returned for points outside the grid limits or for which `lat` or `lon` contain `NaN`. All angles are in units of degrees.

method — Interpolation method
`'nearest'` (default) | `'bilinear'` | `'bicubic'`

Interpolation method, specified as one of the following.

Method	Description
`'nearest'`	Nearest neighbor interpolation
`'bilinear'`	Bilinear interpolation
`'bicubic'`	Bicubic interpolation

Note If the grid size is being reduced (that is, when `scale` is less than 1 or `[numrows numcols]` is less than the size of the input grid) and `method` is `'bilinear'` or `'bicubic'`, then `resizem` applies a low-pass filter before interpolation to reduce aliasing. The default filter size is 11-by-11. You can specify a different length for the default filter using the `n` argument. You can specify a nondefault filter using the `h` argument.

Data Types: `char` | `string`

n — Low-pass filter size
11 (default) | nonnegative integer

Low-pass filter size, specified as a nonnegative integer. The filter size is n-by-n. If `n` is `0`, or if `method` is `'nearest'`, then `resizem` does not perform low-pass filtering.

h — 2-D FIR filter
numeric matrix

2-D FIR filter, specified as a numeric matrix. You can define a FIR filter using Image Processing Toolbox functions such as `ftrans2`, `fwind1`, `fwind2`, or `fsamp2`.

Output Arguments

`Zscaled` — Rescaled data grid
numeric array

Rescaled data grid, returned as a numeric array.

`Rscaled` — Rescaled geographic reference
geographic raster reference object | vector | matrix

Rescaled geographic reference, returned as a geographic raster reference object, numeric vector, or numeric matrix, consistent with the format of `R`.

Compatibility Considerations

`resizem` will be removed
Not recommended starting in R2020b

Some functions that accept referencing vectors or referencing matrices as input will be removed, including the `resizem` function. Use a geographic reference object and the `georesize` function instead. If your data is not geographically referenced, then use the `imresize` function instead. Reference objects have several advantages over referencing vectors and matrices.

- Unlike referencing vectors and matrices, reference objects have properties that document the size of the associated raster, its geographic limits, and the direction of its rows and columns. For more information about reference object properties, see the `GeographicCellsReference` and `GeographicPostingsReference` objects.
- You can manipulate the limits of geographic rasters associated with reference objects using the `geocrop` function.
- You can manipulate the size and resolution of geographic rasters associated with reference objects using the `georesize` function.

To update your code, first create a reference object for either a raster of cells using the `georefcells` function or a raster of regularly posted samples using the `georefpostings` function. Alternatively, convert from a referencing vector or referencing matrix to a reference object using the `refvecToGeoRasterReference` or `refmatToGeoRasterReference` function, respectively.

Then, replace uses of the `resizem` function with the `georesize` function. This table shows typical uses of the `resizem` function and how to update your code to use the `georesize` function instead. Note that the default method of interpolation for the `georesize` function is `'cubic'` instead of `'nearest'`.

Will Be Removed	Recommended
`[B,RB] = resizem(A,scale,R);`	`[B,RB] = georesize(A,R,scale,'nearest');`

`[B,RB] = resizem(A,[numrows numcols],R);`	`latscale = numrows / R.RasterSize(1);`
	`lonscale = numcols / R.RasterSize(2);`
	`[B,RB] = georesize(A,R,latscale,lonscale,'nearest');`

`[B,RB] = resizem(A,scale,R,method);`	`[B,RB] = georesize(A,R,scale,method);`

`[B,RB]  = resizem(A,[numrows numcols],R,method);`	`latscale = numrows / R.RasterSize(1);`
	`lonscale = numcols / R.RasterSize(2);`
	`[B,RB] = georesize(A,R,latscale,lonscale,method);`

If your data is not geographically referenced, then use the `imresize` function instead. This table shows typical uses of the `resizem` function without reference object information and how to update your code to use the `imresize` function instead.

Will Be Removed	Recommended
`B = resizem(A,scale,R);`	`B = imresize(A,scale);`
`B = resizem(A,[numrows numcols],R);`	`B = imresize(A,[numrows numcols]);`
`B = resizem(A,scale,R,method);`	`B = imresize(A,scale,method);`
`B = resizem(A,[numrows numcols],R,method);`	`B = imresize(A,[numrows numcols],method);`

See Also
filter2 | georesize | imresize

Introduced before R2006a

restack

Restack objects within map axes

Note `restack` will be removed in a future release. Use `uistack` instead.

Syntax

`restack(h,`*`position`*`)`

Description

`restack(h,`*`position`*`)` changes the stacking position of the object h within the axes. h can be a handle, a vector of handles to graphics objects, or a name recognized by `handlem`. The *position* argument can have any of the following values: `'top'`, `'bottom'`, `'bot'`, `'up'`, or `'down'`.

See Also
`uistack`

Introduced before R2006a

rhxrh

Intersection points for pairs of rhumb lines

Syntax

```
[newlat,newlong] = rhxrh(lat1,lon1,az1,lat2,lon2,az2)
[newlat,newlon] = rhxrh(lat1,lon1,az1,lat2,lon2,az2,units)
```

Description

`[newlat,newlong] = rhxrh(lat1,lon1,az1,lat2,lon2,az2)` returns in `newlat` and `newlon` the location of the intersection point for each pair of rhumb lines input in *rhumb line notation*. For example, the first line in the pair passes through the point (`lat1,lon1`) and has a constant azimuth of `az1`. When the two rhumb lines are identical or do not intersect (conditions that are not, in general, apparent by inspection), two `NaN`s are returned instead and a warning is displayed. The inputs must be column vectors.

`[newlat,newlon] = rhxrh(lat1,lon1,az1,lat2,lon2,az2,units)` specifies the units used, where units is any valid `units`. The default units are `'degrees'`.

For any pair of rhumb lines, there are three possible intersection conditions: the lines are identical, they intersect once, or they do not intersect at all (except at the poles, where all nonequatorial rhumb lines meet—this is not considered an intersection). `rhxrh` does not allow multiple rhumb line intersections, although it is possible to construct cases in which such a condition occurs. See the following discussion of Limitations on page 1-1247.

Rhumb line notation consists of a point on the line and the constant azimuth of the line.

Examples

Calculate Point of Intersection of Two Paths

Given a starting point at (10ºN,56ºW), a plane maintains a constant heading of 35º. Another plane starts at (0º,10ºW) and proceeds at a constant heading of 310º (–50º). Where would their two paths cross each other?

Calculate Point of Intersection

Use the `rhxrh` function to calculate the point of intersection of two paths.

```
[newlat,newlon] = rhxrh(10,-56,35,0,-10,310)

newlat = 26.9774

newlon = -43.4088
```

Limitations

Rhumb lines are specifically helpful in navigation because they represent lines of constant heading, whereas great circles have, in general, continuously changing heading. In fact, the Mercator

projection was originally designed so that rhumb lines plot as straight lines, which facilitates both manual plotting with a straightedge and numerical calculations using a Cartesian planar representation. When a rhumb line proceeds off the left or right *edge* of this representation at some latitude, it reappears on the other edge at the same latitude and continues on the same slope. For rhumb lines where this occurs—for example, one with a heading of 85º—it is easy to imagine another rhumb line, say one with a heading of 0º, repeatedly intersecting the first. The real-world uses of rhumb lines make this merely an intellectual exercise, however, for in practice it is always clear which *crossing* line segment is relevant. The function `rhxrh` returns at most one intersection, selecting in each case that line segment containing the input starting point for its computation.

See Also

`gcxgc` | `gcxsc` | `scxsc` | `crossfix` | `polyxpoly` | `navfix`

Introduced before R2006a

rmfield

Remove dynamic property from geographic or planar vector

Syntax

```
vout = rmfield(vin,name)
vout = rmfield(vin,names)
```

Description

`vout = rmfield(vin,name)` removes the dynamic property specified by `name` from the geographic or planar vector `vin`.

`vout = rmfield(vin,names)` removes all dynamic properties specified by `names` from `vin`.

Note `rmfield` cannot remove `Metadata`, and `Geometry` properties from any geographic or planar vector. Further, it cannot remove `Latitude` and `Longitude` properties from `geopoint` and `geoshape` objects, nor X and Y properties from `mappoint` and `mapshape` objects.

Examples

Remove a Single Property from a Geopoint Vector

Create a geopoint vector with dynamic properties.

```
gp = geopoint([42 42.2],[-110.5 -110.7],'Temperature',[65.6 63.2],'Humidity',[44 41])

gp =
 2x1 geopoint vector with properties:

 Collection properties:
       Geometry: 'point'
       Metadata: [1x1 struct]
 Feature properties:
       Latitude: [42 42.2000]
      Longitude: [-110.5000 -110.7000]
    Temperature: [65.6000 63.2000]
       Humidity: [44 41]
```

Remove only the `Humidity` property from the geopoint vector.

```
gp2 = rmfield(gp,'Humidity')

gp2 =
 2x1 geopoint vector with properties:

 Collection properties:
       Geometry: 'point'
       Metadata: [1x1 struct]
```

```
    Feature properties:
          Latitude: [42 42.2000]
         Longitude: [-110.5000 -110.7000]
       Temperature: [65.6000 63.2000]
```

Remove Multiple Properties from a Mapshape Vector

Create a mapshape vector.

```
ms = mapshape(shaperead('tsunamis.shp'))
```

```
ms =
 162x1 mapshape vector with properties:

 Collection properties:
       Geometry: 'point'
       Metadata: [1x1 struct]
 Vertex properties:
  (162 features concatenated with 161 delimiters)
              X: [128.3000 NaN -156 NaN 157.9500 NaN 143.8500 NaN -155 NaN ... ]
              Y: [-3.8000 NaN 19.5000 NaN -9.0200 NaN 42.1500 NaN 19.1000 NaN ... ]
 Feature properties:
           Year: [1950 1951 1951 1952 1952 1952 1952 1953 1953 1953 1953 1954 ... ]
          Month: [10 8 12 3 3 5 11 3 6 9 11 6 10 4 11 3 6 7 1 4 7 11 11 5 8 8 ... ]
            Day: [8 21 22 4 17 6 4 18 27 14 25 26 NaN 19 2 9 22 28 19 22 10 6 12 ... ]
           Hour: [3 10 NaN 1 3 NaN 16 NaN NaN 0 17 NaN NaN 20 NaN 14 23 8 14 ... ]
         Minute: [23 57 NaN 22 58 NaN 58 NaN NaN 26 48 NaN NaN 24 NaN 22 50 ... ]
         Second: [NaN NaN NaN 41 NaN NaN NaN NaN NaN 36 NaN NaN NaN NaN NaN ... ]
       Val_Code: [2 4 2 4 4 1 4 3 3 4 4 1 2 4 3 4 3 4 4 2 4 4 2 4 4 4 4 4 4 ... ]
       Validity: {1x162 cell}
     Cause_Code: [1 1 6 1 1 9 1 1 6 3 1 9 0 1 1 1 1 1 1 1 3 1 1 1 0 3 1 1 ... ]
          Cause: {1x162 cell}
         Eq_Mag: [7.6000 6.9000 NaN 8.1000 4.5000 NaN 9 5.8000 NaN 6.8000 ... ]
        Country: {1x162 cell}
       Location: {1x162 cell}
     Max_Height: [2.8000 3.6000 6 6.5000 1 1.5200 18 1.5000 1.4000 3 3 3 ... ]
       Iida_Mag: [1.5000 1.8000 2.6000 2.7000 NaN NaN 4.2000 0.6000 0.5000 ... ]
      Intensity: [1.5000 NaN NaN 2 NaN NaN 4 NaN 1 1.5000 1.5000 NaN NaN ... ]
     Num_Deaths: [NaN NaN NaN 33 NaN NaN 2236 NaN NaN NaN NaN 7 NaN 3 NaN ... ]
     Desc_Deaths: [NaN NaN NaN 1 NaN NaN 3 NaN NaN NaN NaN 1 NaN 1 NaN ... ]
```

Attempt to remove multiple properties from the mapshape vector.

```
s2 = rmfield(ms,{'Geometry','Second','Minute','intensity'})
```

```
s2 =
 162x1 mapshape vector with properties:

 Collection properties:
       Geometry: 'point'
       Metadata: [1x1 struct]
 Vertex properties:
  (162 features concatenated with 161 delimiters)
              X: [128.3000 NaN -156 NaN 157.9500 NaN 143.8500 NaN -155 NaN ... ]
```

```
              Y: [-3.8000 NaN 19.5000 NaN -9.0200 NaN 42.1500 NaN 19.1000 NaN ... ]
 Feature properties:
           Year: [1950 1951 1951 1952 1952 1952 1952 1953 1953 1953 1953 1954 ... ]
          Month: [10 8 12 3 3 5 11 3 6 9 11 6 10 4 11 3 6 7 1 4 7 11 11 5 8 8 ... ]
            Day: [8 21 22 4 17 6 4 18 27 14 25 26 NaN 19 2 9 22 28 19 22 10 6 12 ... ]
           Hour: [3 10 NaN 1 3 NaN 16 NaN NaN 0 17 NaN NaN 20 NaN 14 23 8 14 ... ]
       Val_Code: [2 4 2 4 4 1 4 3 3 4 4 1 2 4 3 4 3 4 4 2 4 4 2 4 4 4 4 4 4 ... ]
       Validity: {1x162 cell}
     Cause_Code: [1 1 6 1 1 9 1 1 6 3 1 9 0 1 1 1 1 1 1 1 3 1 1 1 0 3 1 1 ... ]
          Cause: {1x162 cell}
         Eq_Mag: [7.6000 6.9000 NaN 8.1000 4.5000 NaN 9 5.8000 NaN 6.8000 ... ]
        Country: {1x162 cell}
       Location: {1x162 cell}
     Max_Height: [2.8000 3.6000 6 6.5000 1 1.5200 18 1.5000 1.4000 3 3 3 ... ]
       Iida_Mag: [1.5000 1.8000 2.6000 2.7000 NaN NaN 4.2000 0.6000 0.5000 ... ]
      Intensity: [1.5000 NaN NaN 2 NaN NaN 4 NaN 1 1.5000 1.5000 NaN NaN ... ]
     Num_Deaths: [NaN NaN NaN 33 NaN NaN 2236 NaN NaN NaN NaN 7 NaN 3 NaN ... ]
    Desc_Deaths: [NaN NaN NaN 1 NaN NaN 3 NaN NaN NaN NaN 1 NaN 1 NaN ... ]
```

The `Second` and `Minute` properties have been removed successfully. Note that the `Geometry` property still exists because it cannot be removed. Also, `Intensity` has not been removed because property names are case-sensitive.

Input Arguments

vin — Input geographic or planar vector
geopoint, geoshape, mappoint, or mapshape object

Input geographic or planar vector, specified as a `geopoint`, `geoshape`, `mappoint`, or `mapshape` object.

name — Name of single property to remove
character vector

Name of a single property to remove, specified as a character vector. The property in `name` is case sensitive.

names — Name of multiple properties to remove
cell array of character vectors

Name of multiple properties to remove, specified as a cell array of character vectors. The properties in `names` are case-sensitive.

Output Arguments

vout — Output geographic or planar vector
geopoint, geoshape, mappoint, or mapshape object

Output geographic or planar vector, returned as a `geopoint`, `geoshape`, `mappoint`, or `mapshape` object. The object type of `vout` matches the object type of `vin`.

See Also
append | isfield | rmprop

Introduced in R2012a

rmprop

Remove property from geographic or planar vector

Syntax

```
vout = rmprop(vin,name)
vout = rmprop(vin,names)
```

Description

`vout = rmprop(vin,name)` removes the property specified by `name` from the geographic or planar vector `vin`.

`vout = rmprop(vin,names)` removes all properties specified by `names` from `vin`.

Note `rmprop` cannot remove `Metadata`, and `Geometry` properties from any geographic or planar vector. Further, it cannot remove `Latitude` and `Longitude` properties from `geopoint` and `geoshape` objects, nor `X` and `Y` properties from `mappoint` and `mapshape` objects.

Examples

Remove Single Property from a Geoshape Vector

Create a geoshape vector with dynamic properties.

```
gs = geoshape(shaperead('worldcities.shp', 'UseGeoCoords', true))

gs =
 318x1 geoshape vector with properties:

 Collection properties:
     Geometry: 'point'
     Metadata: [1x1 struct]
 Vertex properties:
  (318 features concatenated with 317 delimiters)
     Latitude: [5.2985 NaN 24.6525 NaN 5.6106 NaN 37.0613 NaN 9.0235 NaN ... ]
    Longitude: [-3.9509 NaN 54.7589 NaN -0.2121 NaN 35.3894 NaN 38.7575 ... ]
 Feature properties:
         Name: {1x318 cell}
```

Remove only the `Name` property from the geoshape vector.

```
gs2 = rmprop(gs,'Name')

gs2 =
 318x1 geoshape vector with properties:

 Collection properties:
     Geometry: 'point'
```

```
        Metadata: [1x1 struct]
 Vertex properties:
   (318 features concatenated with 317 delimiters)
       Latitude: [5.2985 NaN 24.6525 NaN 5.6106 NaN 37.0613 NaN 9.0235 NaN ... ]
      Longitude: [-3.9509 NaN 54.7589 NaN -0.2121 NaN 35.3894 NaN 38.7575 ... ]
```

Remove Multiple Properties from a Mappoint Vector

Create a mappoint vector.

```
mp = mappoint(-33.961,18.484,'Name','Cape Town','Temperature',64)

mp =
 1x1 mappoint vector with properties:

 Collection properties:
       Geometry: 'point'
       Metadata: [1x1 struct]
 Feature properties:
              X: -33.9610
              Y: 18.4840
           Name: 'Cape Town'
    Temperature: 64
```

Attempt to remove multiple properties from the mappoint vector.

```
s2 = rmfield(mp,{'X','Name','temperature'})

s2 =
 1x1 mappoint vector with properties:

 Collection properties:
       Geometry: 'point'
       Metadata: [1x1 struct]
 Feature properties:
              X: -33.9610
              Y: 18.4840
    Temperature: 64
```

The `Name` property has been removed successfully. Note that the `X` property still exists because it cannot be removed. Also, the `Temperature` property still exists because property names are case-sensitive.

Input Arguments

`vin` — Input geographic or planar vector
geopoint, geoshape, mappoint, or mapshape object

Input geographic or planar vector, specified as a `geopoint`, `geoshape`, `mappoint`, or `mapshape` object.

name — Name of single property to remove
character vector

Name of a single property to remove, specified as a character vector. The character vector is case sensitive.

names — Name of multiple properties to remove
cell array of character vectors

Name of multiple properties to remove, specified as a cell array of character vectors. The character vectors are case-sensitive.

Output Arguments

vout — Output geographic or planar vector
geopoint, geoshape, mappoint, or mapshape object

Output geographic or planar vector, returned as a geopoint, geoshape, mappoint, or mapshape object. The object type of vout matches the object type of vin.

See Also
append | isprop | rmfield

Introduced in R2012a

rootlayr

Construct cell array of workspace variables for `mlayers` tool

Note `rootlayr` will be removed in a future release.

Syntax

```
rootlayr
```

Description

`rootlayr` allows the `mlayers` tool to be used with workspace variables. It constructs a cell array that contains all the structure variables in the current workspace. This cell array is returned in the variable `ans`, which can then be an input to `mlayers`. If there is an existing variable named `ans`, it is overwritten.

The recommended calling procedure is `rootlayr;mlayers(ans);`

Examples

`rootlayr` creates a cell array named `ans`, consisting of the three structure variables in the following workspace.

```
whos
  Name          Size          Bytes  Class
  borders       1x1           38390  struct array
  lats          2345x1        18760  double array
  lons          2345x1        18760  double array
  nation        1x1           70224  struct array
  states        1x51         254970  struct array

rootlayr
ans
  ans =
    [1x1  struct]     'borders'
    [1x1  struct]     'nation'
    [1x51 struct]     'states'
```

The function `mlayers(ans)` can now be used to activate the `mlayers` tool for the structures contained in `ans`.

See Also
`mlayers`

Introduced before R2006a

rotatem

Transform vector map data to new origin and orientation

Syntax

```
[lat1,lon1] = rotatem(lat,lon,origin,'forward')
[lat1,lon1] = rotatem(lat,lon,origin,'inverse')
[lat1,lon1] = rotatem(lat,lon,origin,'forward',units)
[lat1,lon1] = rotatem(lat,lon,origin,'forward',units)
```

Description

`[lat1,lon1] = rotatem(lat,lon,origin,'forward')` transforms latitude and longitude data (`lat` and `lon`) to their new coordinates (`lat1` and `lon1`) in a coordinate system resulting from Euler angle rotations as specified by `origin`. The input `origin` is a three- (or two-) element vector having the form `[latitude longitude orientation]`. The latitude and longitude are the coordinates of the point in the original system, which is the center of the output system. The orientation is the azimuth from the new origin point to the original North Pole in the new system. If `origin` has only two elements, the orientation is assumed to be 0º. This `origin` vector might be the output of `putpole` or `newpole`.

`[lat1,lon1] = rotatem(lat,lon,origin,'inverse')` transforms latitude and longitude data (`lat` and `lon`) in a coordinate system *that has been transformed* by Euler angle rotations specified by `origin` to their coordinates (`lat1` and `lon1`) in the coordinate system *from which they were originally transformed*. In a sense, this *undoes* the `'forward'` process. Be warned, however, that if data is rotated forward and then inverted, the final data might not be identical to the original. This is because of roundoff and *data collapse* at the original and intermediate singularities (the poles).

`[lat1,lon1] = rotatem(lat,lon,origin,'forward',units)` and `[lat1,lon1] = rotatem(lat,lon,origin,'forward',units)` specify the angle units of the data, where *units* is any recognized angle unit. The default is `'radians'`. Note that this default is different from that of most functions.

The `rotatem` function transforms vector map data to a new coordinate system.

An analytical use of the new data can be realized in conjunction with the `newpole` function. If a selected point is made the *north pole* of the new system, then when new vector data is created with `rotatem`, the distance of every data point from this new north pole is its new colatitude (90º minus latitude). The absolute difference in the great circle azimuths between every pair of points from their new *pole* is the same as the difference in their new longitudes.

Examples

What are the coordinates of Rio de Janeiro (23ºS,43ºW) in a coordinate system in which New York (41ºN,74ºW) is made the North Pole? Use the `newpole` function to get the origin vector associated with putting New York at the pole:

```
nylat = 41; nylon = -74;
riolat = -23; riolon = -43;
```

```
origin = newpole(nylat,nylon);
[riolat1,riolon1] = rotatem(riolat,riolon,origin,...
                                  'forward','degrees')

riolat1 =
    19.8247
riolon1 =
 -149.7375
```

What does this mean? For one thing, the colatitude of Rio in this new system is its distance from New York. Compare the distance between the original points and the new colatitude:

```
dist = distance(nylat,nylon,riolat,riolon)

dist =
    70.1753

90-riolat1

ans =
    70.1753
```

See Also

neworig | newpole | org2pol | putpole

Introduced before R2006a

rotatetext

Rotate text to projected graticule

Syntax

```
rotatetext
rotatetext(objects)
rotatetext(objects,'inverse')
```

Description

`rotatetext` rotates displayed text objects to account for the curvature of the graticule. The objects are selected interactively from a graphical user interface.

`rotatetext(objects)` rotates the selected objects. `objects` can be a name recognized by `handlem` or a vector of handles to displayed text objects.

`rotatetext(objects,'inverse')` removes the rotation added by an earlier use of `rotatetext`. If omitted, `'forward'` is assumed.

Examples

Add text to a map and rotate the text to the graticule.

```
figure
worldmap('south america')
geoshow('landareas.shp','facecolor','yellow')
cities = shaperead('worldcities.shp', 'UseGeoCoords', true);
Santiago = strcmp('Santiago',{cities(:).Name});
h=textm(cities(Santiago).Lat, cities(Santiago).Lon, ...
    'Santiago');
rotatetext(h)
```
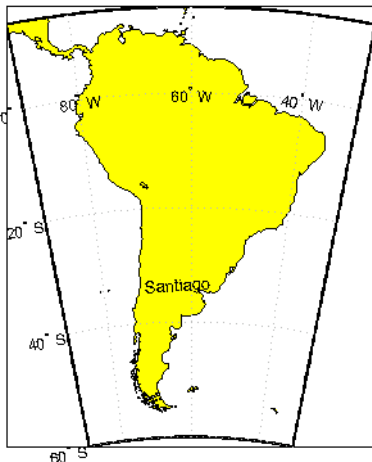
Tips

You can rotate meridian and parallel labels automatically by setting the map axes `LabelRotation` property to `'on'`.

See Also

`vfwdtran` | `vinvtran`

Introduced before R2006a

roundn

Round to multiple of 10^n

Note roundn is not recommended. Use round instead.

Syntax

```
roundn(x,n)
```

Description

roundn(x,n) rounds each element of x to the nearest multiple of 10^n. n must be scalar, and integer-valued. For complex x, the imaginary and real parts are rounded independently. For n = 0, roundn gives the same result as round. That is, roundn(x,0) == round(x).

Examples

Round pi to the nearest hundredth.

```
roundn(pi, -2)

ans =

    3.1400
```

Round the equatorial radius of the Earth, 6378137 meters, to the nearest kilometer.

```
roundn(6378137, 3)

ans =

    6378000
```

See Also

round

Introduced before R2006a

RPCCoefficientTag

Rational Polynomial Coefficients Tag

Description

`RPCCoefficientTag` contains the Rational Polynomial Coefficients (RPC) TIFF tag, which describes the relationship of latitude, longitude, and height locations with row and column locations in an image.

The `RPCCoefficientTag` object contains properties with names and permissible values corresponding to the tag elements listed in the technical note *RPCs in GeoTIFF* technical note, viewable at http://geotiff.maptools.org/rpc_prop.html. `RPCCoefficientTag` is used by the functions `geotiffinfo` and `geotiffwrite`.

Creation

You can use the `map.geotiff.RPCCoefficientTag` function to create an `RPCCoefficientTag` object.

Properties

`BiasErrorInMeters` — Root mean square bias error in meters per horizontal axis
`-1` (default) | nonnegative scalar

Root mean square bias error in meters per horizontal axis, specified as the number `-1` or a nonnegative scalar. The value is `-1` only if `BiasErrorInMeters` is not specified

Data Types: `double`

`RandomErrorInMeters` — Root mean square random error in meters per horizontal axis
`-1` (default) | nonnegative scalar

Root mean square random error in meters, specified as the number `-1` or a nonnegative scalar. The value is `-1` only if `RandomErrorInMeters` is not specified.

Data Types: `double`

`LineOffset` — Line offset in pixels
`0` (default) | nonnegative scalar

Line offset in pixels, specified as a nonnegative scalar, with a value of `0` by default.

Data Types: `double`

`SampleOffset` — Sample offset in pixels
`0` (default) | nonnegative scalar

Sample offset in pixels, specified as a nonnegative scalar, with a value of `0` by default.

Data Types: `double`

GeodeticLatitudeOffset — Geodetic latitude offset in degrees
0 (default) | numeric scalar

Geodetic latitude offset in degrees, specified as a numeric scalar. The value can range from -90 <= value <= 90.

Data Types: double

GeodeticLongitudeOffset — Geodetic longitude offset in degrees
0 (default) | numeric scalar

Geodetic longitude offset in degrees, specified as a numeric scalar. The value can range from -180 <= value <= 180.

Data Types: double

GeodeticHeightOffset — Geodetic height offset in meters
0 (default) | numeric scalar

Geodetic height offset in meters, specified as a numeric scalar.

Data Types: double

LineScale — Line scale factor in pixels
1 (default) | positive scalar

Line scale factor in pixels, specified as a positive scalar.

Data Types: double

SampleScale — Sample scale factor in pixels
1 (default) | positive scalar

Sample scale factor in pixels, specified as a positive scalar.

Data Types: double

GeodeticLatitudeScale — Geodetic latitude scale in degrees
1 (default) | positive scalar

Geodetic latitude scale in degrees, specified as a positive scalar in the range (0,90].

Data Types: double

GeodeticLongitudeScale — Geodetic longitude scale in degrees
1 (default) | positive scalar

Geodetic longitude scale in degrees, specified as positive scalar in the range (0, 180].

Data Types: double

GeodeticHeightScale — Geodetic height scale factor in meters
1 (default) | positive scalar

Geodetic height scale factor in meters, specified as a positive scalar.

Data Types: double

LineNumeratorCoefficients — Coefficients for the polynomial in the numerator of the *r(n)* equation

20-element row vector of zeros (default) | 20-element row vector

Coefficients for the polynomial in the numerator of the *r(n)* equation, specified as a 20-element row vector of class `double`.

Data Types: `double`

LineDenominatorCoefficients — Coefficients for the polynomial in the denominator of the *r(n)* equation

20-element row vector of zeros (default) | 20-element row vector

Coefficients for the polynomial in the denominator of the *r(n)* equation, specified as a 20-element row vector of class `double`.

Data Types: `double`

SampleNumeratorCoefficents — Coefficients for the polynomial in the numerator of the *c(n)* equation

20-element row vector of zeros (default) | 20-element row vector

Coefficients for the polynomial in the numerator of the *c(n)* equation, specified as a 20-element row vector of class `double`.

Data Types: `double`

SampleDenominatorCoefficents — Coefficients for the polynomial in the denominator of the *c(n)* equation

20-element row vector of zeros (default) | 20-element row vector

Coefficients for the polynomial in the denominator of the *c(n)* equation, specified as a 20-element row vector of class `double`.

Data Types: `double`

Methods

double   Convert TIFF tag property values to row vector of doubles

See Also

`geotiffinfo` | `geotiffwrite` | `Tiff`

Introduced in R2015b

rsphere

Radii of auxiliary spheres

Syntax

```
r = rsphere('biaxial',ellipsoid)
r = rsphere('biaxial',ellipsoid,method)
r = rsphere('triaxial',ellipsoid)
r = rsphere('triaxial',ellipsoid,method)
r = rsphere('eqavol',ellipsoid)
r = rsphere('authalic',ellipsoid)
r = rsphere('rectifying',ellipsoid)
r = rsphere('curve',ellipsoid,lat)
r = rsphere('curve',ellipsoid,lat,method)
r = rsphere('euler',lat1,lon1,lat2,lon2,ellipsoid)
r = rsphere('curve', ..., angleUnits)
r = rsphere('euler', ..., angleUnits)
```

Description

`r = rsphere('biaxial',ellipsoid)` computes the arithmetic mean i.e., `(a+b)/2` where `a` and `b` are the semimajor and semiminor axes of the specified ellipsoid. `ellipsoid` is a `referenceSphere`, `referenceEllipsoid`, or `oblateSpheroid` object, or a vector of the form `[semimajor_axis eccentricity]`.

`r = rsphere('biaxial',ellipsoid,method)` computes the arithmetic mean if `method` is `'mean'` and the geometric mean, `sqrt(a*b)`, if `method` is `'norm'`.

`r = rsphere('triaxial',ellipsoid)` computes the triaxial arithmetic mean of the semi-major axes, `a`, and semi-minor axes, `b` of the ellipsoid, `(2*a+b)/3`.

`r = rsphere('triaxial',ellipsoid,method)` computes the arithmetic mean if `method` is `'mean'` and the triaxial geometric mean, `(a^2*b)^(1/3)`, if `method` is `'norm'`.

`r = rsphere('eqavol',ellipsoid)` returns the radius of a sphere with a volume equal to that of the ellipsoid.

`r = rsphere('authalic',ellipsoid)` returns the radius of a sphere with a surface area equal to that of the ellipsoid.

`r = rsphere('rectifying',ellipsoid)` returns the radius of a sphere with meridional distances equal to those of the ellipsoid.

`r = rsphere('curve',ellipsoid,lat)` computes the arithmetic mean of the transverse and meridional radii of curvature at the latitude, `lat`. `lat` is in degrees.

`r = rsphere('curve',ellipsoid,lat,method)` computes an arithmetic mean if `method` is `'mean'` and a geometric mean if `method` is `'norm'`.

`r = rsphere('euler',lat1,lon1,lat2,lon2,ellipsoid)` computes the Euler radius of curvature at the midpoint of the geodesic arc defined by the endpoints (`lat1,lon1`) and (`lat2,lon2`). `lat1`, `lon1`, `lat2`, and `lon2` are in degrees.

`r = rsphere('curve', ..., angleUnits)` and `r = rsphere('euler', ..., angleUnits)` where `angleUnits` specifies the units of the `latitude` and `longitude` inputs as either `'degrees'` or `'radians'`.

Examples

Different criteria result in different spheres:

`r = rsphere('biaxial',referenceEllipsoid('earth','km'))`

```
r =
   6.3674e+03
```

`r = rsphere('triaxial',referenceEllipsoid('earth','km'))`

```
r =
   6.3710e+03
```

`r = rsphere('curve',referenceEllipsoid('earth','km'))`

```
r =
   6.3781e+03
```

See Also

`rcurve`

Introduced before R2006a

satbath

(To be removed) Read 2-minute terrain/bathymetry from Smith and Sandwell

---

Note `satbath` will be removed in a future release. Use `readgeoraster` instead. For more information, see "Compatibility Considerations".

Syntax

```
[latgrat,longrat,z] = satbath
[latgrat,longrat,z] = satbath(scalefactor)
[latgrat,longrat,z] = satbath(scalefactor,latlim,lonlim)
[latgrat,longrat,z] = satbath(scalefactor,latlim,lonlim,gsize)
```

Description

`[latgrat,longrat,z] = satbath` reads the global topography file for the entire world (`topo_8.2.img`), returning every 50th point. The result is returned as a geolocated data grid. If you use a different version of the global topography file, you need to rename it to "`topo_8.2.img`". If the file is not found on the MATLAB path, a dialog opens to request the file.

`[latgrat,longrat,z] = satbath(scalefactor)` returns the data for the entire world, subsampled by the integer `scalefactor`. A `scalefactor` of 10 returns every 10th point. The matrix at full resolution has 6336 by 10800 points.

`[latgrat,longrat,z] = satbath(scalefactor,latlim,lonlim)` returns data for the specified region. The returned data extends slightly beyond the requested area. If omitted, the entire area covered by the data file is returned. The limits are two-element vectors in units of degrees, with `latlim` in the range `[-90 90]` and `lonlim` in the range `[-180 180]`.

`[latgrat,longrat,z] = satbath(scalefactor,latlim,lonlim,gsize)` controls the size of the graticule matrices. `gsize` is a two-element vector containing the number of rows and columns desired. If omitted, a graticule the size of the data grid is returned.

Background

This is a global bathymetric model derived from ship soundings and satellite altimetry by W.H.F. Smith and D.T. Sandwell. The model was developed by iteratively adjusting gravity anomaly data from Geosat and ERS-1 against historical track line soundings. This technique takes advantage of the fact that gravity mirrors the large variations in the ocean floor as small variations in the height of the ocean's surface. The computational procedure uses the ship track line data to calibrate the scaling between the observed surface undulations and the inferred bathymetry. Land elevations are reduced-resolution versions of GTOPO30 data.

Examples

Read the data for the Falklands Islands (Islas Malvinas) at full resolution.

```
[latgrat,longrat,mat] = satbath(1,[-55 -50],[-65 -55]);
whos

  Name          Size          Bytes  Class

  latgrat       247x301       594776  double array
  longrat       247x301       594776  double array
  mat           247x301       594776  double array
```

Tips

Land elevations are given in meters above mean sea level. The data is stored in a Mercator projection grid. As a result, spatial resolution varies with latitude. The grid spacing is 2 minutes (about 4 kilometers) at the equator.

This data is available over the Internet, but subject to copyright. The data file is binary, and should be transferred with no line-ending conversion or byte swapping. This function carries out any byte swapping that might be required. The data requires about 133 MB uncompressed.

The data and documentation are available over the Internet via http and anonymous ftp. Download the latest version of file `topo_x.2.img`, where `x` is the version number, and rename it `topo_8.w.img` for compatibility with the `satbath` function.

`satbath` returns a geolocated data grid rather than a regular data grid and a referencing vector or matrix. This is because the data is in a Mercator projection, with columns evenly spaced in longitude, but with decreasing spacing for rows at higher latitudes. Referencing vectors and matrices assume that the number of cells per degrees of latitude and longitude are both constant across a data grid.

Note For details on locating map data for download over the Internet, see the following documentation at the MathWorks Web site: "Find Geospatial Data Online".

Compatibility Considerations

satbath will be removed
Warns starting in R2021b

Some raster reading functions that return latitude-longitude grids will be removed, including `satbath`. Instead, use `readgeoraster`, which returns a map raster reference object. Reference objects have several advantages over latitude-longitude grids.

- Unlike latitude-longitude grids, reference objects have properties that document the size of the associated raster, its limits, and the direction of its rows and columns. For more information about reference object properties, see `MapCellsReference` and `MapPostingsReference`.
- You can manipulate the limits of rasters associated with reference objects using the `mapcrop` function.
- You can manipulate the size and resolution of rasters associated with reference objects using the `mapresize` function.

This table shows some typical usages of `satbath` and how to update your code to use `readgeoraster` instead. Unlike `satbath`, which requires a single file with extension `.img`, the `readgeoraster` function requires both a data file with extension `.ers` and a supporting file with extension `.img`. When you call `readgeoraster`, specify the file with extension `.ers`.

Will Be Removed	Recommended
`[latgrat,longrat,z] = satbath;`	`[Z,R] = readgeoraster(filename);`
`[latgrat,longrat,z] = satbath(scalefactor);`	`[Z,R] = readgeoraster(filename);` `[Z,R] = georesize(Z,R,1/scalefactor);`

`[latgrat,longrat,z] = satbath(scalefactor,latlim,lonlim);`	`[Z,R] = readgeoraster(filename);`
	`[Z,R] = geocrop(Z,R,latlim,lonlim);`
	`[Z,R] = georesize(Z,R,1/scalefactor);`

The `readgeoraster` function returns data using the native data type embedded in the file. Return a different data type by specifying the `'OutputType'` name-value pair. For example, use `[Z,R] = readgeoraster(filename,'OutputType','double')`.

The `readgeoraster` function does not automatically replace missing data with `NaN` values. If your data set uses large negative numbers to indicate missing data, you can replace them with `NaN` values using the `standardizeMissing` function.

```
[Z,R] = readgeoraster('MtWashington-ft.grd');
info = georasterinfo('MtWashington-ft.grd');
m = info.MissingDataIndicator;
Z = standardizeMissing(Z,m);
```

See Also
egm96geoid | readgeoraster | georasterinfo

Introduced before R2006a

scaleruler

Add or modify graphic scale on map axes

Syntax

```
scaleruler
scaleruler on
scaleruler off
scaleruler(property,value,...)
h = scaleruler(...)
```

Description

`scaleruler` toggles the display of a graphic scale. If no graphic scale is currently displayed in the current map axes, one is added. If any graphic scales are currently displayed, they are removed.

`scaleruler on` adds a graphic scale to the current map axes. Multiple graphic scales can be added to the same map axes.

`scaleruler off` removes any currently displayed graphic scales.

`scaleruler(property,value,...)` adds a graphic scale and sets the properties to the values specified. You can display a list of graphic scale properties using the command `setm(h)`, where `h` is the handle to a graphic scale object. The current values for a displayed graphic scale object can be retrieved using `getm`. The properties of a displayed graphic scale object can be modified using `setm`.

`h = scaleruler(...)` returns the `hggroup` handle to the graphic scale object.

Background

Cartographers often add graphic elements to the map to indicate its scale. Perhaps the most commonly used is the graphic scale, a ruler-like object that shows distances on the ground at the correct size for the projection.

Examples

Add Graphic Scale to Map Axes

Create a map display of Florida.

```
usamap('Florida')
geoshow('usastatelo.shp','FaceColor','yellow')
```

Add a graphic scale. Change the location of the scale by setting the `XLoc` and `YLoc` properties. Specify tick locations using the `MajorTick` property.

```
scaleruler on
setm(handlem('scaleruler1'), ...
```

```
     'XLoc',-3.21e5,'YLoc',2.81e6, ...
     'MajorTick',0:200:600)
```

Add a second graphics scale that shows distance in nautical miles. Change the direction of the tick marks and text by setting the `TickDir` property.

```
scaleruler('units','nm')
setm(handlem('scaleruler2'), ...
     'XLoc',-3.2e5, ...
     'YLoc',2.78e6, ...
     'TickDir','down', ...
     'MajorTick',0:100:300, ...
     'MinorTick',0:25:50, ...
     'MajorTickLength',km2nm(25),...
     'MinorTickLength',km2nm(12.5))
```
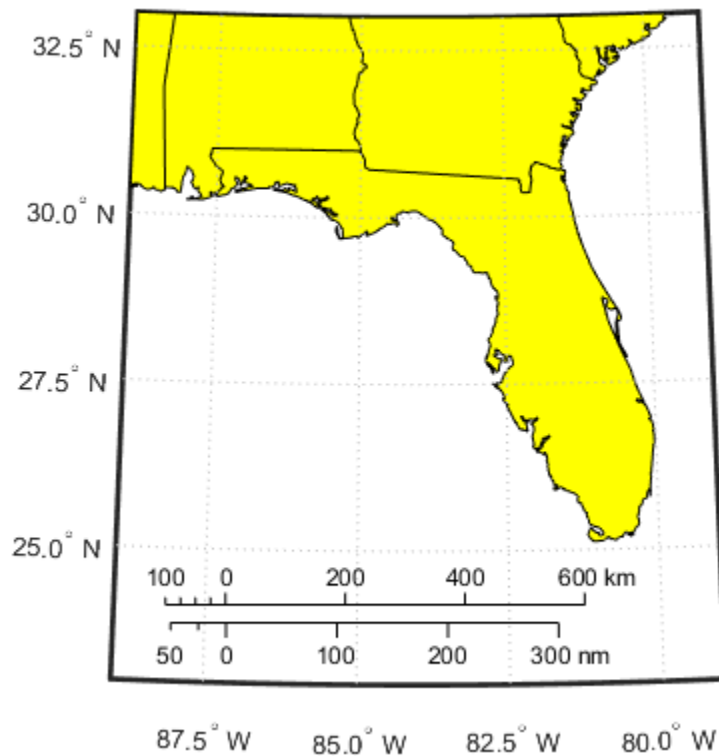

Customize Appearance of Graphic Scale

Create a map of Texas and the surrounding states. Your map might look different because the `polcmap` function randomizes colors.

```
states = readgeotable("usastatehi.shp");
figure
usamap("Texas")
```

```
h = height(states);
faceColors = makesymbolspec("Polygon",...
    {'INDEX',[1 h],'FaceColor',polcmap(h)});

geoshow(states,"DisplayType","polygon", ...
    "SymbolSpec",faceColors)
```

Add a default graphic scale. Change the location and set the font size to 8 points.

```
scaleruler on
setm(handlem("scaleruler1"), ...
    "XLoc",-6.2e5,"YLoc",3.1e6, ...
    "MajorTick",0:200:600, ...
    "FontSize",8)
```
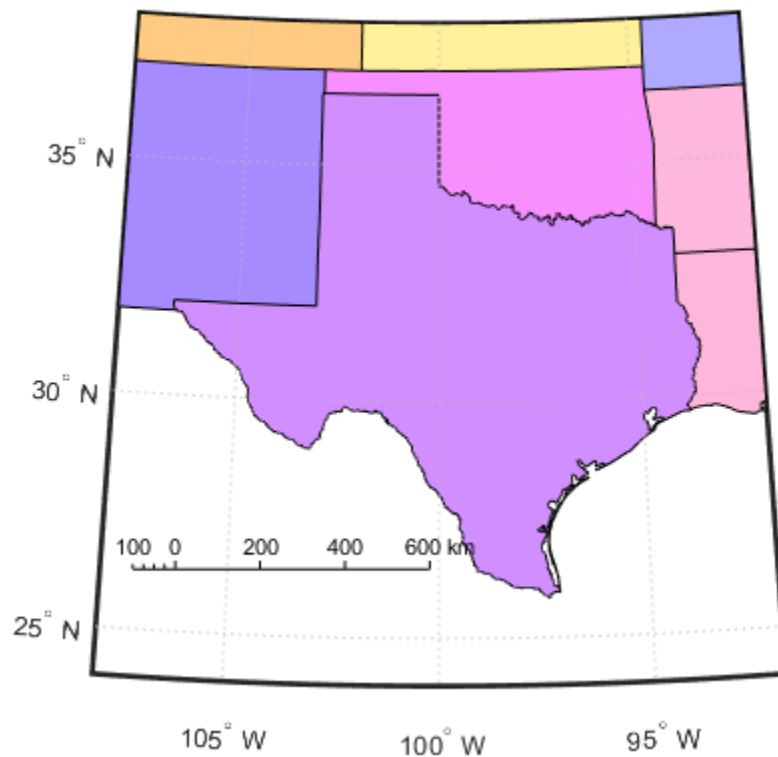

The location units of `scaleruler` default to kilometers. Note that `handlem` accepts the keyword `"scaleruler"` or `"scaleruler1"` for the first `scaleruler`, `"scaleruler2"` for the second one, etc. If there is more than one `scaleruler` on the current axes, specifying the keyword `"scaleruler"` returns a vector of handles.

Return the `scaleruler` as a `Group` object using the `handlem` function. Inspect its properties using the `getm` function.

```
s = handlem("scaleruler");
getm(s)
```

```
ans = struct with fields:
        Azimuth: 0
```

```
          Children: []
             Color: [0 0 0]
         FontAngle: 'normal'
          FontName: 'Helvetica'
          FontSize: 8
         FontUnits: 'points'
        FontWeight: 'normal'
             Label: ''
               Lat: 29.6479
              Long: -101.7263
         LineWidth: 0.5000
         MajorTick: [0 200 400 600]
    MajorTickLabel: {4x1 cell}
   MajorTickLength: 20
         MinorTick: [0 25 50 75 100]
    MinorTickLabel: '100'
   MinorTickLength: 12.5000
            Radius: 'earth'
         RulerStyle: 'ruler'
           TickDir: 'up'
          TickMode: 'manual'
             Units: 'km'
              XLoc: -620000
              YLoc: 3100000
              ZLoc: []
```

Create a second graphic scale using units of nautical miles. Modify the tick properties of the second graphic scale.

```
scaleruler("units","nm")
setm(handlem("scaleruler2"),"YLoc",3.0e6, ...
    "MajorTick",0:100:300,...
    "MinorTick",0:25:50,"TickDir","down", ...
    "MajorTickLength",km2nm(25),...
    "MinorTickLength",km2nm(12.5))
```
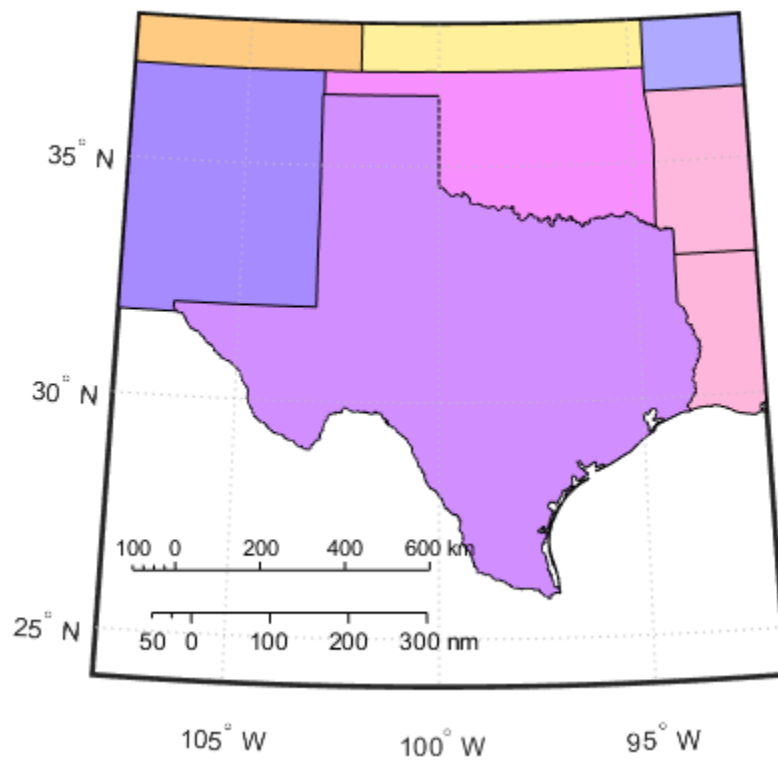
Change the ruler styles.

```
setm(handlem("scaleruler1"),"RulerStyle","lines")
setm(handlem("scaleruler2"),"RulerStyle","patches")
```
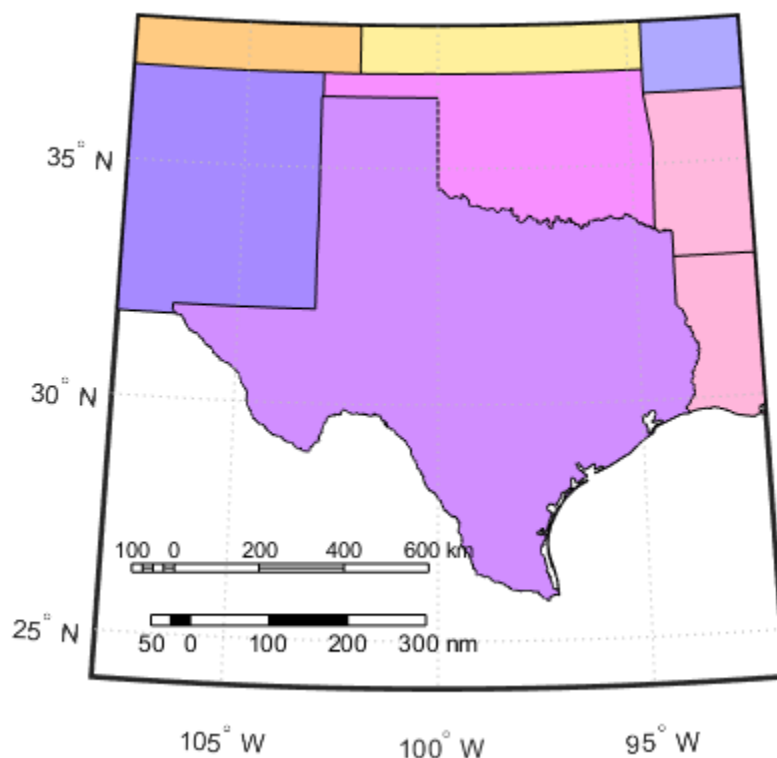
Object Properties

Properties That Control Appearance

Color

    ColorSpec {no default}

Color of the displayed graphic scale — Controls the color of the graphic scale lines and text. You can specify a color using a vector of RGB values or one of the MATLAB predefined names. By default, the graphic scale is displayed in black ([0 0 0]).

FontAngle

    {normal} | italic | oblique

Angle of the graphic scale label text — Controls the appearance of the graphic scale text components. Use any MATLAB font angles.

FontName

    courier | {helvetica} | symbol | times

Font family name for all graphic scale labels — Sets the font for all displayed graphic scale labels. To display and print properly FontName must be a font that your system supports.

FontSize

    scalar in units specified in FontUnits {9}

Font size — Specifies the font size to use for all displayed graphic scale labels, in units specified by the `FontUnits` property. The default point size is 9.

FontUnits

 inches | centimeters | normalized | {points} | pixels

Units used to interpret the FontSize property — When set to normalized, the toolbox interprets the value of `FontSize` as a fraction of the height of the axes. For example, a normalized `FontSize` of 0.16 sets the text characters to a font whose height is one-tenth of the axes' height. The default units, points, are equal to 1/72 of an inch.

FontWeight

 light | {normal} | demi | bold

Select bold or normal font — The character weight for all displayed graphic scale labels.

Label

 character vector

Label text for the graphic scale — Contains a character vector used to label the graphic scale. The text is displayed centered on the scale. The label is often used to indicate the scale of the map, for example "1:50,000,000."

LineWidth

 scalar {0.5}

Graphic scale line width — Sets the line width of the displayed scale. The value is a scalar representing points, which is 0.5 by default.

MajorTick

 vector

Graphic scale major tick locations — Sets the major tick locations for the graphic scale. The default values are chosen to give a reasonably sized scale. You can specify the locations of the tick marks by providing a vector of locations. These are usually equally spaced values as generated by `start:step:end`. The values are distances in the units of the `Units` property.

MajorTickLabel

 Cell array of character vectors

Graphic scale major tick labels — Sets the text labels associated with the major tick locations. By default, the labels are identical to the major tick locations. You can override these by providing a cell array of character vectors. There must be as many character vectors as tick locations.

MajorTickLength

 scalar

Length of the major tick lines — Controls the length of the major tick lines. The length is a distance in the units of the `Units` property.

MinorTick

 vector

Graphic scale minor tick locations — Sets the minor tick locations for the graphic scale. The default values are chosen to give a reasonably sized scale. You can specify the locations of the tick marks by providing a vector of locations. These are usually equally spaced values as generated by `start:step:end`. The values are distances in the units of the `Units` property.

MinorTickLabel

    `character vectors`

    Graphic scale minor tick labels — Sets the text labels associated with the minor tick locations. By default, the label is identical to the last minor tick location. You can override this by providing a character vector label.

MinorTickLength

    `scalar`

    Length of the minor tick lines — Controls the length of the minor tick lines. The length is a distance in the units of the `Units` property.

RulerStyle

    `{ruler} | lines | patches`

    Style of the graphic scale — Selects among three different kinds of graphic scale displays. The default `ruler` style looks like n axes' *x*-axis. The `lines` style has three horizontal lines across the tick marks. This type of graphic scale is often used on maps from the U.S. Geological Survey. The `patches` style has alternating black and white rectangles in place of lines and tick marks.

TickDir

    `{up} | down`

    Direction of the tick marks and text — Controls the direction in which the tick marks and text labels are drawn. In the default up direction, the tick marks and text labels are placed above the baseline, which is placed at the location given in the `XLoc` property. In the down position, the tick marks and labels are drawn below the baseline.

TickMode

    `{auto} | manual`

    Tick locations mode — Controls whether the tick locations and labels are computed automatically or are user-specified. Explicitly setting the tick labels or locations results in a `'manual'` tick mode. Setting any of the tick labels or locations to an empty matrix resets the tick mode to `'auto'`. Setting the tick mode to `'auto'` clears any explicitly specified tick locations and labels, which are then replaced by default values.

XLoc

    `scalar`

    X-location of the graphic scale — Controls the horizontal location of the graphic scale within the axes. The location is specified in the axes Cartesian projected coordinates. Use `showaxes` to make the Cartesian grid labels visible. You can also move the graphic scale by dragging the baseline with the mouse.

YLoc

    `scalar`

    Y-location of the graphic scale — Controls the vertical location of the graphic scale within the axes. The location is specified in the axes Cartesian projected coordinates. Use `showaxes` to make the Cartesian grid labels visible. You can also move the graphic scale by dragging the baseline with the mouse.

Properties That Control Scaling

Azimuth

    scalar

Azimuth of scale computation — The scale of a map varies, within the projection, with geographic location and azimuth. This property controls the azimuth along which the scaling between geographic and projected coordinates is computed. The azimuth is given in the current angle units of the map axes. The default azimuth is 0.

Lat

    scalar

Latitude of scale computation — The scale of a map varies, within the projection, with geographic location and azimuth. This property controls the geographic location at which the scaling between geographic and projected coordinates is computed. The latitude is given in the current angle units of the map axes. The default location is the center of the displayed map.

Long

    scalar

Longitude of scale computation — The scale of a map varies, within the projection, with geographic location and azimuth. This property controls the geographic location at which the scaling between geographic and projected coordinates is computed. The longitude is given in the current angle units of the map axes. The default location is the center of the displayed map.

Radius

    Name or radius of reference sphere

Reference sphere name or radius — The radius property controls the scaling between angular and surface distances. radius can be one of the character vectors supported by km2deg, or it can be the (numerical) radius of the desired sphere in the same units as the Units property. The default is 'earth'.

Units

    (valid distance unit)

Surface distance units — Defines the distance units displayed in the graphic scale. Units can be any distance unit recognized by unitsratio. The distance character vector is also used in the last graphic scale text label.

Other Properties

Children

    (read-only)

Name of graphic scale elements — Contains the tag assigned to the graphic elements that compose the graphic scale. All elements of the graphic scale have hidden handles except the baseline. You do not normally need to access the elements directly.

Tips

You can reposition graphic scale objects by dragging them with the mouse. You can also change their positions by modifying the XLoc and YLoc properties using setm.

Use `scaleruler off`, `clmo scaleruler`, or `clmo scalerulerN` to remove the scale rulers. You can also remove a graphic scale object with `delete(h)`, or `delete(handlem(`scalerulerN'))`, where `N` is the corresponding integer.

See Also
distance | surfdist | axesscale | paperscale | mdistort | distortcalc

Introduced before R2006a

scatterm

Project point markers with variable color and area

Syntax

```
scatterm(lat,lon,s,c)
scatterm(lat,lon)
scatterm(lat,lon,s)
scatterm(...,m)
scatterm(...,'filled')
scatterm(ax,...)
h = scatterm(...)
```

Description

scatterm(lat,lon,s,c) displays colored circles at the locations specified by the vectors lat and lon (which must be the same size). The area of each marker is determined by the values in the vector s (in points^2) and the colors of each marker are based on the values in c. s can be a scalar, in which case all the markers are drawn the same size, or a vector the same length as lat and lon.

When c is a vector the same length as lat and lon, the values in c are linearly mapped to the colors in the current colormap. When c is a length(lat)-by-3 matrix, the values in c specify the colors of the markers as RGB values. c can also be a color character vector.

scatterm(lat,lon) draws the markers in the default size and color.

scatterm(lat,lon,s) draws the markers with a single color.

scatterm(...,m) uses the marker m instead of 'o'.

scatterm(...,'filled') fills the markers.

scatterm(ax,...) plots into axes ax instead of gca. ax is a handle to a map axes.

h = scatterm(...) returns a handle to an hggroup.

Examples

Plot the seamount MATLAB data as symbols with the color proportional to the height.

```
load seamount
worldmap([-49 -47.5],[-150 -147.5])
scatterm(y,x,5,z)
scaleruler
set(gca,'Visible','off')
```
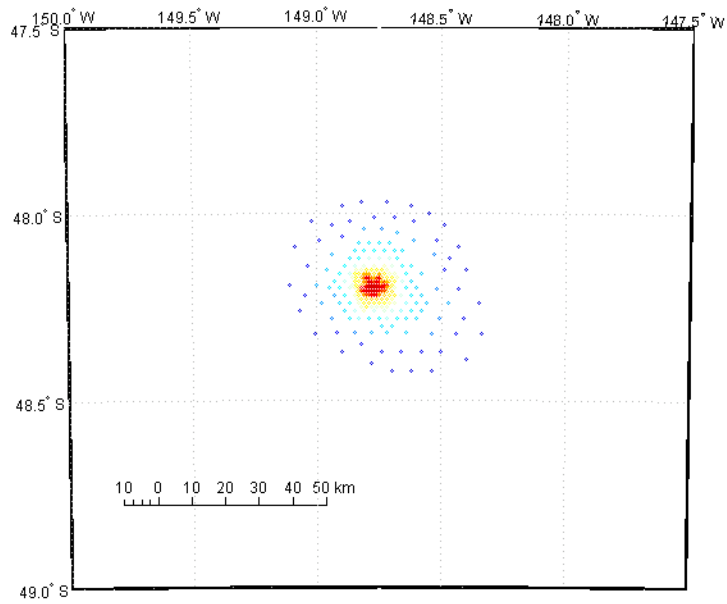
See Also
stem3m

Introduced before R2006a

scircle1

Small circles from center, range, and azimuth

Syntax

```
[lat,lon] = scircle1(lat0,lon0,rad)
[lat,lon] = scircle1(lat0,lon0,rad,az)
[lat,lon] = scircle1(lat0,lon0,rad,az,ellipsoid)
[lat,lon] = scircle1(lat0,lon0,rad,units)
[lat,lon] = scircle1(lat0,lon0,rad,az,units)
[lat,lon] = scircle1(lat0,lon0,rad,az,ellipsoid,units)
[lat,lon] = scircle1(lat0,lon0,rad,az,ellipsoid,units,npts)
[lat,lon] = scircle1(track,...)
```

Description

`[lat,lon] = scircle1(lat0,lon0,rad)` computes small circles (on a sphere) with a center at the point `lat0,lon0` and radius `rad`. The inputs can be scalar or column vectors. The input radius is in degrees of arc length on a sphere.

`[lat,lon] = scircle1(lat0,lon0,rad,az)` uses the input `az` to define the small circle arcs computed. The arc azimuths are measured clockwise from due north. If `az` is a column vector, then the arc length is computed from due north. If `az` is a two-column matrix, then the small circle arcs are computed starting at the azimuth in the first column and ending at the azimuth in the second column. If `az = []`, then a complete small circle is computed.

`[lat,lon] = scircle1(lat0,lon0,rad,az,ellipsoid)` computes small circles on the ellipsoid defined by the input `ellipsoid`, rather than by assuming a sphere. `ellipsoid` is a `referenceSphere`, `referenceEllipsoid`, or `oblateSpheroid` object, or a vector of the form `[semimajor_axis eccentricity]`. If the semimajor axis is non-zero, `rad` is assumed to be in distance units matching the units of the semimajor axis. However, if `ellipsoid = []`, or if the semimajor axis is zero, then `rad` is interpreted as an angle and the small circles are computed on a sphere as in the preceding syntax.

`[lat,lon] = scircle1(lat0,lon0,rad,units)`,
`[lat,lon] = scircle1(lat0,lon0,rad,az,units)`, and
`[lat,lon] = scircle1(lat0,lon0,rad,az,ellipsoid,units)`
are all valid calling forms, which use the input units to define the angle units of the inputs and outputs. If you omit units, `'degrees'` is assumed.

`[lat,lon] = scircle1(lat0,lon0,rad,az,ellipsoid,`*units*`,npts)` uses the scalar input `npts` to determine the number of points per small circle computed. The default value of `npts` is 100.

`[lat,lon] = scircle1(track,...)` uses track to define either a great circle or rhumb line radius. If *track* = `'gc'`, then small circles are computed. If `track = 'rh'`, then the circles with radii of constant rhumb line distance are computed. If you omit track, `'gc'` is assumed.

`mat = scircle1(...)` returns a single output argument where `mat = [lat lon]`. This is useful if a single small circle is computed.

Multiple circles can be defined from a single starting point by providing scalar `lat0,lon0` inputs and column vectors for `rad` and `az` if desired.

Examples

Create and plot a small circle centered at (0º,0º) with a radius of 10º.

```
axesm('mercator','MapLatLimit',[-30 30],'MapLonLimit',[-30 30]);
[latc,longc] = scircle1(0,0,10);
plotm(latc,longc,'g')
```

If the desired radius is known in some nonangular distance unit, use the radius returned by the `earthRadius` function as the ellipsoid input to set the range units. (Use an empty azimuth entry to indicate a full circle.)

```
[latc,longc] = scircle1(0,0,550,[],earthRadius('nm'));
plotm(latc,longc,'r')
```

For just an arc of the circle, enter an azimuth range.

```
[latc,longc] = scircle1(0,0,5,[-30 70]);
plotm(latc,longc,'m')
```
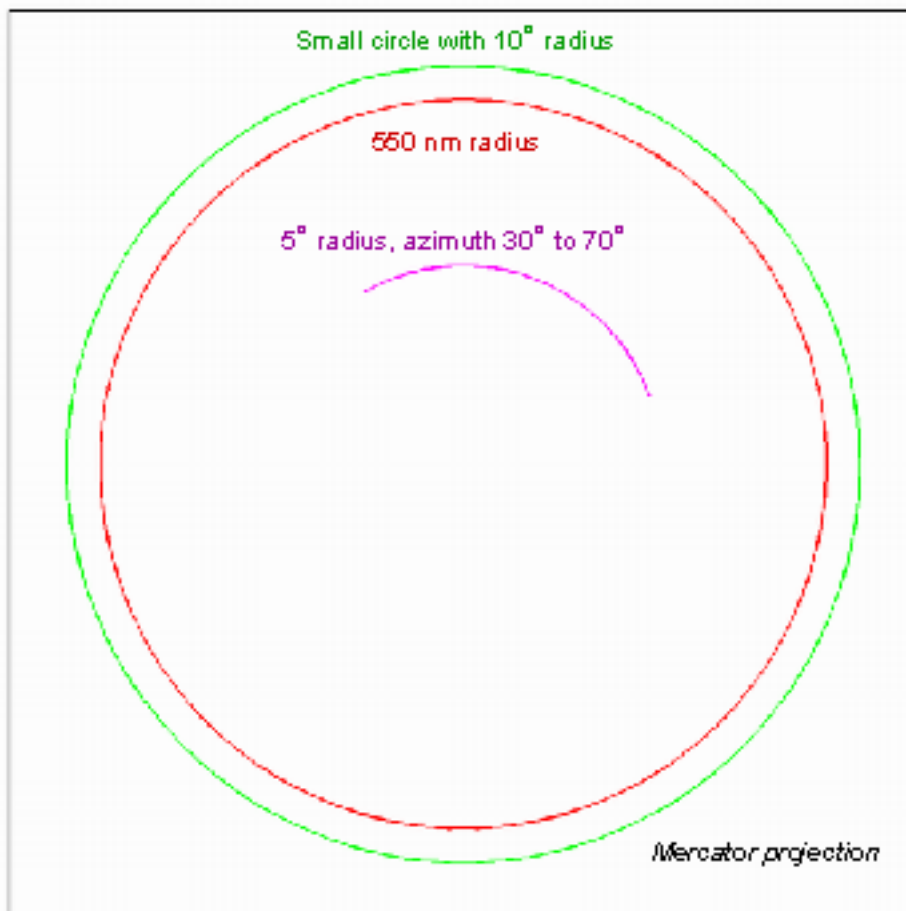
More About

Small Circle

A *small circle* is the locus of all points an equal surface distance from a given center. For true small circles, this distance is always calculated in a great circle sense; however, the `scircle1` function allows a locus to be calculated using distances in a rhumb line sense as well. An example of a small circle is *all points exactly 100 miles from the Washington Monument*. Parallels on the globe are all small circles. Great circles are a subset of small circles, specifically those with a radius of 90º or its angular equivalent, so all meridians on the globe are small circles as well.

Small Circle Notation

Small circle notation consists of a center point and a radius in units of angular arc length.

See Also

Functions
scircle2 | scircleg | track | trackg | track1 | track2

Topics
"Generate Small Circles"

Introduced before R2006a

scircle2

Small circles from center and perimeter

Syntax

```
[lat,lon] = scircle2(lat1,lon1,lat2,lon2)
[lat,lon] = scircle2(lat1,lon1,lat2,lon2,ellipsoid)
[lat,lon] = scircle2(lat1,lon1,lat2,lon2,units)
[lat,lon] = scircle2(lat1,lon1,lat2,lon2,ellipsoid,units)
[lat,lon] = scircle2(lat1,lon1,lat2,lon2,ellipsoid,units,npts)
[lat,lon] = scircle2(track,...)
mat = scircle2(...)
mat = [lat lon]
```

Description

`[lat,lon] = scircle2(lat1,lon1,lat2,lon2)` computes small circles (on a sphere) with centers at the points `lat1,lon1` and points on the circles at `lat2,lon2`. The inputs can be scalar or column vectors.

`[lat,lon] = scircle2(lat1,lon1,lat2,lon2,ellipsoid)` computes the small circle on the ellipsoid defined by the input `ellipsoid`, rather than by assuming a sphere. `ellipsoid` is a `referenceSphere`, `referenceEllipsoid`, or `oblateSpheroid` object, or a vector of the form `[semimajor_axis eccentricity]`. Default is a unit sphere.

`[lat,lon] = scircle2(lat1,lon1,lat2,lon2,units)` and
`[lat,lon] = scircle2(lat1,lon1,lat2,lon2,ellipsoid,units)` are valid calling forms, which use the input `units` to define the angle units of the inputs and outputs. If you omit `units`, `'degrees'` is assumed.

`[lat,lon] = scircle2(lat1,lon1,lat2,lon2,ellipsoid,units,npts)` uses the scalar input `npts` to determine the number of points per track computed. The default value of `npts` is 100.

`[lat,lon] = scircle2(track,...)` uses track to define either a great circle or a rhumb line radius. If `track' = 'gc'`, then small circles are computed. If `track = 'rh'`, then circles with radii of constant rhumb line distance are computed. If you omit track, `'gc'` is assumed.

`mat = scircle2(...)` returns a single output argument where `mat = [lat lon]`. This is useful if a single circle is computed.

Multiple circles can be defined from a single center point by providing scalar `lat1,lon1` inputs and column vectors for the points on the circumference, `lat2,lon2`.

Examples

Plot Locus of Points Equidistant from New Delhi

Plot the locus of all points the same distance from New Delhi as Kathmandu.

```
axesm('mercator','MapLatlimit',[0 40],'MapLonLimit',[60 110]);
load coastlines
% For reference
plotm(coastlat,coastlon,'k');
% New Delhi
lat1 = 29; lon1 = 77.5;
% Kathmandu
lat2 = 27.6; lon2 = 85.5;
% Plot the cities
plotm([lat1 lat2],[lon1 lon2],'b*')
[latc,lonc] = scircle2(lat1,lon1,lat2,lon2);
plotm(latc,lonc,'b')
```
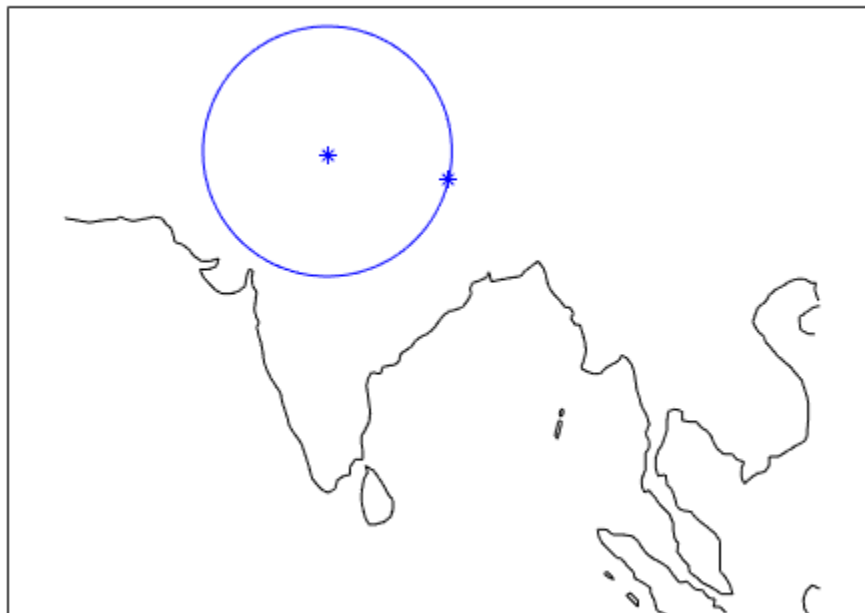

More About

Small Circle

A *small circle* is the locus of all points an equal surface distance from a given center. For true small circles, this distance is always calculated in a great circle sense. However, the `scircle2` function allows a locus to be calculated using distances in a rhumb line sense as well. An example of a small circle is *all points exactly 100 miles from the Washington Monument*.

See Also

Functions
scircle1 | track | track1 | track2

Topics
"Generate Small Circles"

Introduced before R2006a

scircleg

Small circle defined via mouse input

Syntax

```
h = scircleg(ncirc)
h = scircleg(ncirc,npts)
h = scircleg(ncirc,LineSpec)
h = scircleg(ncirc,PropertyName,PropertyValue,...)
[lat,lon] = scircleg(ncirc,npts,...)
h = scircleg(track,ncirc,...)
```

Description

`h = scircleg(ncirc)` brings forward the current map axes and waits for the user to make (2 * `ncirc`) mouse clicks. The output `h` is a vector of handles for the `ncirc` small circles, which are then displayed.

`h = scircleg(ncirc,npts)` specifies the number of plotting points to be used for each small circle. `npts` is 100 by default.

`h = scircleg(ncirc,LineSpec)` specifies the line style, marker, and color on page 1-1289.

`h = scircleg(ncirc,PropertyName,PropertyValue,...)` allows property name/property value pairs to be set, where *PropertyName* and `PropertyValue` are recognized by the `line` function.

`[lat,lon] = scircleg(ncirc,npts,...)` returns the coordinates of the plotted points rather than the handles of the small circles. Successive circles are stored in separate columns of `lat` and `lon`.

`h = scircleg(track,ncirc,...)` specifies the logic with which ranges are calculated. If track is `'gc'` (the default), great circle distance is used. If track is `'rh'`, rhumb line distance is used.

This function is used to define small circles for display using mouse clicks. For each circle, two clicks are required: one to mark the center of the circle and one to mark any point on the circle itself, thereby defining the radius.
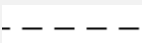
Background

A small circle is the locus of all points an equal surface distance from a given center. For true small circles, this distance is always calculated in a great circle sense; however, the `scircleg` function allows a locus to be calculated using distances in a rhumb line sense as well. You can modify the circle after creation by **shift**+clicking it. The circle is then in edit mode, during which you can change the size and position by dragging control points, or by entering values into a control panel. **Shift**+clicking again exits edit mode.

More About

Line Style, Marker, and Color

Specify the line style, marker, and color as a character vector or string containing symbols. The symbols can appear in any order. You do not need to specify all three characteristics (line style, marker, and color). For example, specify a red dashed line using '--r'.

Line Style	Description	Resulting Line
'-'	Solid line	——————
'--'	Dashed line	– — — — —
':'	Dotted line	················
'-.'	Dash-dotted line	—·—·—·—·—

Marker	Description	Resulting Marker
'o'	Circle	◯
'+'	Plus sign	+
'*'	Asterisk	✳
'.'	Point	•
'x'	Cross	×
'_'	Horizontal line	—
'|'	Vertical line	│
's'	Square	□
'd'	Diamond	◇
'^'	Upward-pointing triangle	△
'v'	Downward-pointing triangle	▽
'>'	Right-pointing triangle	▷
'<'	Left-pointing triangle	◁
'p'	Pentagram	☆
'h'	Hexagram	✡

Color Name	Short Name	RGB Triplet	Appearance
'red'	'r'	[1 0 0]	
'green'	'g'	[0 1 0]	
'blue'	'b'	[0 0 1]	
'cyan'	'c'	[0 1 1]	
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

See Also

scircle1 | scircle2

Introduced before R2006a

SCXSC

Intersection points for pairs of small circles

Syntax

```
[lat,lon] = scxsc(lat1,lon1,range1,lat2,lon2,range2)
[lat,lon] = scxsc(lat1,lon1,range1,lat2,lon2,range2,units)
latlon = scxsc( ___ )
```

Description

`[lat,lon] = scxsc(lat1,lon1,range1,lat2,lon2,range2)` returns in `lat` and `lon` the locations where pairs of small circles intersect. The small circles are defined using *small circle notation*, which consists of a center point and a radius in units of angular arc length. For example, the first small circle in a pair would be centered on the point (`lat1`,`lon1`) with a radius of `range1` (in angular units).

For any pair of small circles, there are four possible intersection conditions: the circles are identical, they do not intersect, they are tangent to each other and hence they intersect once, or they intersect twice.

`[lat,lon] = scxsc(lat1,lon1,range1,lat2,lon2,range2,units)` specifies the angular units used for all inputs, where `units` is any valid angular unit.

`latlon = scxsc( ___ )` returns a single output consisting of the concatenated latitude and longitude coordinates of the small circle intersection points.

Examples

Find Intersection Points of Two Small Circles

Given a small circle centered at (10ºS,170ºW) with a radius of 20º (~1200 nautical miles), where does it intersect with a small circle centered at (3ºN, 179ºE), with a radius of 15º (~900 nautical miles)?

```
[newlat,newlon] = scxsc(-10,-170,20,3,179,15)

newlat =
   -8.8368    9.8526

newlon =
  169.7578 -167.5637
```

Note that in this example, the two small circles cross the date line.

Input Arguments

lat1, lon1 — Center coordinate of first small circle
numeric scalar | *n*-element numeric vector

Latitude or longitude coordinate of the center of the first small circle in each pair, specified as one of these values.

- A numeric scalar to find the intersection of a single pair of small circles.
- A *n*-element numeric vector to find the intersection of *n* pairs of small circles.

`lat1` and `lon1` must have the same length.

Example: `-10`

Example: `[-10 20 90 -45]`

range1 — Radius of first small circle
positive numeric scalar | *n*-element vector of positive numbers

Radius of the first small circle of each pair, in angular units, specified as one of these values.

- A positive numeric scalar to find the intersection of a single pair of small circles.
- A *n*-element vector of positive numbers to find the intersection of *n* pairs of small circles. The length of `range1` matches the length of `lat1` and `lon1`.

Example: `20`

Example: `[20 10 45 45]`

`lat2`, `lon2` — Center coordinate of second small circle
numeric scalar | numeric vector

Latitude or longitude coordinate of the center of the second small circle in each pair, specified as one of these values.

- A numeric scalar to find the intersection of a single pair of small circles.
- A *n*-element numeric vector to find the intersection of *n* pairs of small circles.

`lat2` and `lon2` must have the same length as `lat1` and `lon1`.

Example: `3`

Example: `[3 30 85 -45]`

range2 — Radius of second small circle
positive numeric scalar | *n*-element vector of positive numbers

Radius of the second small circle of each pair, in angular units, specified as one of these values.

- A positive numeric scalar to find the intersection of a single pair of small circles.
- A *n*-element vector of positive numbers to find the intersection of *n* pairs of small circles. The length of `range2` matches the length of `lat2` and `lon2`.

Example: `15`

Example: `[15 15 45 50]`

units — Angular units
`'degrees'` (default) | `'radians'`

Angular units, specified as `'degrees'` or `'radians'`.

Output Arguments

`lat`, `lon` — Coordinates of small circle intersections
2-element vector | *n*-by-2 matrix

Coordinates of small circle intersections, returned as one of the following.

- 2-element vector when you find the intersection of a single pair of small circles.
- *n*-by-2 matrix when you find the intersection of *n* pairs of small circles.

If a pair of small circles do not intersect, or are identical, then `scxsc` displays a warning and returns NaNs for the latitude and longitude coordinates of the intersection points. If a pair of small circles are tangent, then the single intersection point is returned twice.

`latlon` — Concatenated coordinates of small circle intersections
4-element vector | *n*-by-4 matrix

Concatenated coordinates of small circle intersections, returned as one of the following. This output is identical to [`lat lon`].

- 4-element vector when you find the intersection of a single pair of small circles.
- *n*-by-4 matrix when you find the intersection of *n* pairs of small circles.

If a pair of small circles do not intersect, or are identical, then `scxsc` displays a warning and returns NaNs for the latitude and longitude coordinates of the intersection points. If a pair of small circles are tangent, then the single intersection point is returned twice.

Tips

Great circles are a subset of small circles — a great circle is just a small circle with a radius of 90º. This provides two methods of notation for defining great circles. *Great circle notation* consists of a point on the circle and an azimuth at that point. *Small circle notation* for a great circle consists of a center point and a radius of 90º (or its equivalent in radians).

See Also
`gc2sc` | `gcxgc` | `gcxsc` | `rhxrh` | `crossfix` | `polyxpoly`

Introduced before R2006a

sdtsdemread

(To be removed) Read data from SDTS raster/DEM data set

Note `sdtsdemread` will be removed in a future release. Use `readgeoraster` instead. For more information, see "Compatibility Considerations".

Syntax

```
[Z, R] = sdtsdemread(filename)
```

Description

`[Z, R] = sdtsdemread(filename)` reads data from an SDTS DEM data set. `Z` is a matrix containing the elevation values. `R` is a referencing matrix. `NaNs` are assigned to elements of `Z` corresponding to null data values or fill data values in the cell module.

`filename` can be the name of the SDTS catalog directory file (`*CATD.DDF`) or the name of any of the other files in the data set. `filename` can include the directory name; otherwise `filename` is searched for in the current directory and the MATLAB path. If any of the files specified in the catalog directory are missing, `sdtsdemread` fails.

Examples

```
filename = fullfile(matlabroot,'examples','map','data','sdts','9129CATD.DDF');
[Z,R] = sdtsdemread(filename);
mapshow(Z,R,'DisplayType','contour')
```

Tips

Elevation values can be imported with `sdtsdemread` from DEMs that use the SPRE Raster Profile (in use since January, 2001) as well as from older SDTS DEM data sets. Under this profile, elevations can be encoded either as 32-bit floating-point numbers (when their units are "decimal meters"), or as 16-bit integers (when units are "feet" or "meters"). The output class from `sdtsdemread` for both types of elevation encoding is `double`.

Note For details on locating map data for download over the Internet, see the following documentation at the MathWorks Web site: "Find Geospatial Data Online".

Compatibility Considerations

sdtsdemread will be removed
Warns starting in R2021b

Raster reading functions that return referencing matrices will be removed, including `sdtsdemread`. Instead, use `readgeoraster`, which returns a raster reference object. Reference objects have several advantages over referencing matrices.

- Unlike referencing matrices, reference objects have properties that document the size of the associated raster, its limits, and the direction of its rows and columns. For examples of reference object properties, see `GeographicCellsReference` and `MapPostingsReference`.
- You can manipulate the limits of rasters associated with reference objects using the `geocrop` or `mapcrop` functions.
- You can manipulate the size and resolution of rasters associated with reference objects using the `georesize` or `mapresize` functions.
- Most functions that accept referencing matrices as input also accept reference objects.

To update your code, change instances of the function name `sdtsdemread` to `readgeoraster` and specify an extension for the data file. The `readgeoraster` function returns data using the native data type embedded in the file. Return a different data type by specifying the `'OutputType'` name-value pair.

```
filename = fullfile(matlabroot,'examples','map','data','sdts','9129CATD.DDF');
[Z,R] = readgeoraster(filename,'OutputType','double');
```

The `readgeoraster` function does not automatically replace missing data with `NaN` values. If your data set uses large negative numbers to indicate missing data, replace them with `NaN` values using the `standardizeMissing` function.

```
filename = fullfile(matlabroot,'examples','map','data','sdts','9129CATD.DDF');
[Z,R] = readgeoraster(filename);
info = georasterinfo(filename);
m = info.MissingDataIndicator;
Z = standardizeMissing(Z,m);
```

See Also
mapshow | sdtsinfo | readgeoraster | georasterinfo

Introduced before R2006a

sdtsinfo

Information about SDTS data set

Syntax

info = sdtsinfo(filename)

Description

info = sdtsinfo(filename) returns a structure whose fields contain information about the contents of a SDTS data set.

filename is a string scalar or character vector that specifies the name of the SDTS catalog directory file, such as 7783CATD.DDF. The file name can also include the directory name. If filename does not include the directory, then it must be in the current directory or in a directory on the MATLAB path. If sdtsinfo cannot find the SDTS catalog file, it returns an error.

If any of the other files in the data set as specified by the catalog file is missing, a warning message is returned. Subsequent calls to read data from the file might also fail.

Field Descriptions

The info structure contains the following fields:

Filename	Name of the catalog directory file of the SDTS transfer set, specified as a character vector.
Title	Name of the data set, specified as a character vector.
ProfileID	Profile Identifier, e.g., 'SRPE: SDTS RASTER PROFILE and EXTENSIONS', specified as a character vector.
ProfileVersion	Profile Version Identifier, e.g., 'VER 1.1 1998 01', specified as a character vector.
MapDate	Date associated with the cartographic information contained in the data set, specified as a character vector.
DataCreationDate	Creation date of the data set, specified as a character vector.
HorizontalDatum	Horizontal datum to which the data is referenced, specified as a character vector.
MapRefSystem	Projection and reference system used, specified as one of the following values: 'GEO', 'SPCS', 'UTM', 'UPS', or ''.
ZoneNumber	Scalar value representing the zone number
XResolution	Scalar value representing the X component of the horizontal coordinate resolution
YResolution	Scalar value representing the Y component of the horizontal coordinate resolution
NumberOfRows	Scalar value representing the number of rows of the DEM

NumberOfCols	Scalar value representing the number of columns of the DEM
HorizontalUnits	Units used for the horizontal coordinate values, specified as a character vector.
VerticalUnits	Units used for the vertical coordinate values, specified as a character vector.
MinElevation	Scalar value of the minimum elevation value for the data set
MaxElevation	Scalar value of the maximum elevation value for the data set

Examples

```
filename = fullfile(matlabroot,'examples','map','data','sdts','9129CATD.DDF');
info = sdtsinfo(filename);
```

See Also

georasterinfo | readgeoraster

Introduced before R2006a

sectorg

Sector of small circle defined via mouse input

Syntax

```
sectorg
```

Description

`sectorg` prompts the user to indicate by two successive mouse clicks two points that define the center and radius of a small circle arc. By default, the angular width of the sector is 60º. The sector is constructed using the vector defined by the mouse clicks as the reference azimuth (defined to run through the center of the sector).

Once a sector has been drawn, **Shift**+clicking on it displays four control points (center point, arc resize, radial resize, and rotation controls), and the associated **Sector** control window. You can graphically interact with sectors as follows:

- To translate the circle, click and drag the center (o) control.
- To change the arc size, click and drag the resize control (square).
- To change the radial size of the sector, click and drag the radial control (down triangle).
- To rotate the arc, click and drag the rotation control (x).

You can also modify a selected sector by entering the appropriate values in the **Sector** control window and then pressing **Enter** or clicking the **Close** button. Display of the control panel is toggled by **Shift**+clicking the sector. If you select multiple sectors, a separate **Sector** control window will appear for each one.

Tips

Sector control windows are superimposed at the same location. A valid map axes must exist prior to running this function.

See Also
`scircleg` | `trackg`

Introduced before R2006a

setltln

(To be removed) Convert data grid rows and columns to latitude-longitude

---

Note The `setltln` function will be removed in a future release. Use the `intrinsicToGeographic` function instead. For more information, see "Compatibility Considerations".

Syntax

```
[lat, lon] = setltln(Z, R, row, col)
[lat, lon, indxPointOutsideGrid] = setltln(Z, R, row, col)
latlon = setltln(Z, R, row, col)
```

Description

`[lat, lon] = setltln(Z, R, row, col)` returns the latitude and longitudes associated with the input row and column coordinates of the regular data grid Z. R can be a geographic raster reference object, a referencing vector, or a referencing matrix.

If R is a geographic raster reference object, its `RasterSize` property must be consistent with `size(Z)`.

If R is a referencing vector, it must be 1-by-3 with elements:

`[cells/degree northern_latitude_limit western_longitude_limit]`

If R is a referencing matrix, it must be 3-by-2 and transform raster row and column indices to/from geographic coordinates according to:

`[lon lat] = [row col 1] * R`

If R is a referencing matrix, it must define a (non-rotational, non-skewed) relationship in which each column of the data grid falls along a meridian and each row falls along a parallel. Points falling outside the grid are ignored in `row` and `col`. All input angles are in degrees.

`[lat, lon, indxPointOutsideGrid] = setltln(Z, R, row, col)` returns the indices of the elements of the `row` and `col` vectors that lie outside the input grid. The outputs `lat` and `lon` always ignore these points; the third output accounts for them.

`latlon = setltln(Z, R, row, col)` returns the coordinates in a single two-column matrix of the form `[latitude longitude]`.

Examples

Load elevation raster data and a geographic cells reference object. Then, find the coordinates of row 45 and column 65.

```
load topo60c
[lat,lon,indxPointOutsideGrid] = setltln(topo60c,topo60cR,45,65)
```

```
lat =
```

```
        -45.5000


lon =

        64.5000


indxPointOutsideGrid =

        []
```

The third output argument is empty because the point is valid.

Compatibility Considerations

`setltln` will be removed
Not recommended starting in R2021a

Some functions that accept referencing vectors or referencing matrices as input will be removed, including the `setltln` function. Use a geographic raster reference object and the `intrinsicToGeographic` function instead. Reference objects have several advantages over referencing vectors and matrices.

- Unlike referencing vectors and matrices, reference objects have properties that document the size of the associated raster, its geographic limits, and the direction of its rows and columns. For more information about reference object properties, see the `GeographicCellsReference` and `GeographicPostingsReference` objects.
- You can manipulate the limits of rasters associated with reference objects using the `geocrop` function.
- You can manipulate the size and resolution of rasters associated with reference objects using the `georesize` function.
- Most functions that accept referencing vectors or matrices as inputs also accept reference objects.

To update your code, first create a reference object for either a raster of cells using the `georefcells` function or a raster of regularly posted samples using the `georefpostings` function. Alternatively, convert from a referencing vector or a referencing matrix to a reference object using the `refvecToGeoRasterReference` or `refmatToGeoRasterReference` function, respectively.

Then, replace uses of the `setltln` function with the `intrinsicToGeographic` function according to these patterns. Note that the `intrinsicToGeographic` function does not round the row and column coordinates. Additionally, the `intrinsicToGeographic` function extrapolates points outside the raster limits instead of discarding them.

Will Be Removed	Recommended

`[lat,lon] = setltln(A,R,row,col);`	`[lat,lon] = intrinsicToGeographic(R,round(col),round(`
	`lat(~contains(R,lat,lon)) = [];`
	`lon(~contains(R,lat,lon)) = [];`

`[lat,lon,indxPointOutsideGrid] = setltln(A,R,row,col);`	`[lat,lon] = intrinsicToGeographic(R,round(col),round(`
	`lat(~contains(R,lat,lon)) = [];`
	`lon(~contains(R,lat,lon)) = [];`
	`indxPointOutsideGrid = find(~contains(R,lat,lon));`

Will Be Removed	Recommended

`latlon = setltln(A,R,row,col);`	`[lat,lon] = intrinsicToGeographic(R,round(col),round(`
	`lat(~contains(R,lat,lon)) = [];`
	`lon(~contains(R,lat,lon)) = [];`
	`latlon = [lat(:) lon(:)];`

See Also

geointerp | geographicToDiscrete | intrinsicToGeographic

Introduced before R2006a

setm

Set properties of map axes and graphics objects

Syntax

```
setm(h,MapAxesPropertyName,PropertyValue,...)
setm(texthndl,'MapPosition',position)
setm(surfhndl,'Graticule',lat,lon,alt)
setm(surfhndl,'MeshGrat',npts,alt)
```

Description

`setm(h,MapAxesPropertyName,PropertyValue,...)`, where `h` is a valid map axes handle, sets the map axes properties specified in the input list. The map axes properties must be recognized by `axesm`. For a list of map axes properties, see Map Axes Properties.

`setm(texthndl,'MapPosition',position)` alters the position of the projected text object specified by its handle to the `[latitude longitude]` or the `[latitude longitude altitude]` specified by the position vector.

`setm(surfhndl,'Graticule',lat,lon,alt)` alters the graticule of the projected surface object specified by its handle. The graticule is specified by the latitude and longitude matrices, specifying locations of the graticule vertices. The altitude can be specified by a scalar, or by a matrix providing a value for each vertex.

`setm(surfhndl,'MeshGrat',npts,alt)` alters the mesh graticule of projected surface objects displayed using the `meshm` function. In this case, the two-element vector `npts` specifies the graticule size in the manner described under `meshm`. The altitude can be a scalar or a matrix with a size corresponding to `npts`.

Examples

Display a map axes and alter it:

```
axesm('bonne','Frame','on','Grid','on')
```

The standard Bonne projection has a standard parallel at 30ºN.
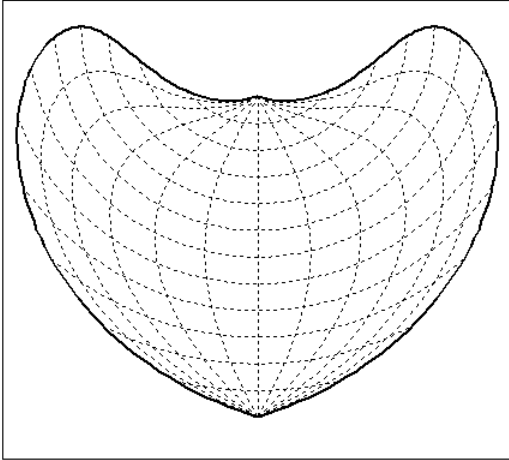
Setting this standard parallel to 0º results in a Sinusoidal projection:

```
setm(gca,'MapParallels',0)
```
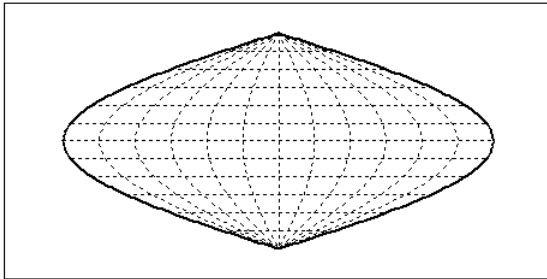

See Also

axesm | getm

Introduced before R2006a

setpostn

(To be removed) Convert latitude-longitude to data grid rows and columns

---

Note The setpostn function will be removed in a future release. Use the geographicToDiscrete function instead. For more information, see "Compatibility Considerations".

Syntax

```
[row, col] = setpostn(Z, R, lat, lon)
indx = setpostn(...)
[row, col, indxPointOutsideGrid] = setpostn(...)
```

Description

[row, col] = setpostn(Z, R, lat, lon) returns the row and column indices of the regular data grid Z for the points specified by the vectors lat and lon. R can be a geographic raster reference object, a referencing vector, or a referencing matrix.

If R is a geographic raster reference object, its RasterSize property must be consistent with size(Z).

If R is a referencing vector, it must be 1-by-3 with elements:

[cells/degree northern_latitude_limit western_longitude_limit]

If R is a referencing matrix, it must be 3-by-2 and transform raster row and column indices to/from geographic coordinates according to:

[lon lat] = [row col 1] * R

If R is a referencing matrix, it must define a (non-rotational, non-skewed) relationship in which each column of the data grid falls along a meridian and each row falls along a parallel. Points falling outside the grid are ignored in row and col. All input angles are in degrees.

indx = setpostn(...) returns the indices of Z corresponding to the points in lat and lon. Points falling outside the grid are ignored in indx.

[row, col, indxPointOutsideGrid] = setpostn(...) returns the indices of lat and lon corresponding to points outside the grid. These points are ignored in row and col.

Examples

Load elevation raster data and a geographic cells reference object. Then, find the matrix coordinates of Denver, Colorado.

```
load topo60c
[row,col] = setpostn(topo60c,topo60cR,39.7,105)

row =
```

```
    130


col =

    105
```

Compatibility Considerations

setpostn will be removed
Not recommended starting in R2021a

Some functions that accept referencing vectors or referencing matrices as input will be removed, including the `setpostn` function. Use a geographic raster reference object and the `geographicToDiscrete` function instead. Reference objects have several advantages over referencing vectors and matrices.

- Unlike referencing vectors and matrices, reference objects have properties that document the size of the associated raster, its geographic limits, and the direction of its rows and columns. For more information about reference object properties, see the `GeographicCellsReference` and `GeographicPostingsReference` objects.
- You can manipulate the limits of rasters associated with reference objects using the `geocrop` function.
- You can manipulate the size and resolution of rasters associated with reference objects using the `georesize` function.
- Most functions that accept referencing vectors or matrices as inputs also accept reference objects.

To update your code, first create a reference object for either a raster of cells using the `georefcells` function or a raster of regularly posted samples using the `georefpostings` function. Alternatively, convert from a referencing vector or a referencing matrix to a reference object using the `refvecToGeoRasterReference` or `refmatToGeoRasterReference` function, respectively.

Then, replace uses of the `setpostn` function with the `geographicToDiscrete` function according to these patterns. Note that the `geographicToDiscrete` function includes NaN values in `row` and `col` for points outside the raster limits instead of discarding them. Additionally, the `geographicToDiscrete` function does not issue a warning for points outside the raster limits.

Will Be Removed	Recommended

`[row,col] = setpostn(A,R,lat,lon);`	`[row,col] = geographicToDiscrete(R,lat,lon);`
	`n = isnan(row);`
	`row(n) = [];`
	`col(n) = [];`

`indx = setpostn(A,R,lat,lon);`	`[row,col] = geographicToDiscrete(R,lat,lon);` `indx = sub2ind(R.RasterSize,row,col);`

`[row,col,indexPointOutsideGrid] = setpostn(A,R,lat,lon)`	`[row,col] = geographicToDiscrete(R,lat,lon);`
	`n = isnan(row);`
	`row(n) = [];`
	`col(n) = [];`
	`indexPointOutsideGrid = find(n);`

See Also

geographicToDiscrete | geographicToIntrinsic | intrinsicToGeographic

Introduced before R2006a

servers

Return URLs of unique WMS servers

Syntax

```
serverURLs = servers(layers)
```

Description

`serverURLs = servers(layers)` returns the URLs of unique servers associated with Web map service layers, `layers`.

Examples

Find Unique URLs of Servers

Search the WMS Database for government servers. Set the unique URLs to a variable.

```
layers = wmsfind('*.gov*','SearchField','serverurl');
serverURLs = servers(layers);
```

List Server URLs and Number of Layers

Search the WMS Database for servers that contain information about clouds. Set the unique server URLs to a variable.

```
temperature = wmsfind('clouds');
serverURLs = servers(temperature);
```

Display the server URLs and the number of layers that reference the URL.

```
for k=1:numel(serverURLs)
   querystr = serverURLs{k};
   layers = refine(temperature,querystr, ...
      'SearchFields', 'serverurl');
   fprintf('Server URL\n%s\n', layers(1).ServerURL);
   fprintf('Number of layers: %d\n\n', numel(layers));
end

Server URL
https://disc1.gsfc.nasa.gov/daac-bin/wms_omi?

Number of layers: 2

Server URL
https://svs.gsfc.nasa.gov/cgi-bin/wms?

Number of layers: 8
```

Your results might be different because the WMS Database changes each release.

Input Arguments

`layers` — Layers to provide server URLs
array of `WMSLayer` objects

Layers to provide server URLs, specified as an array of `WMSLayer` objects.

Output Arguments

`serverURLs` — URLs of unique servers
cell array of character vectors

URLs of unique servers, returned as a cell array of character vectors.

See Also
`wmsfind` | `refine` | `serverTitles`

Introduced in R2009b

serverTitles

Return titles of unique WMS servers

Syntax

```
titles = serverTitles(layers)
```

Description

`titles = serverTitles(layers)` returns the titles of unique servers associated with Web map service layers, `layers`.

Examples

List Titles of Unique Servers

Search the WMS Database for government servers. Set the titles of unique servers to a variable.

```
layers = wmsfind('*.gov*', 'SearchField', 'serverurl');
titles = serverTitles(layers);
```

Display the first ten servers titles as a column vector.

```
titles(1:10)'
```

```
ans = 10x1 cell
    {'LRS-SDI WMS'                        }
    {'My GeoServer WMS'                   }
    {'Geologic Map of Arizona'            }
    {'WMS'                                }
    {'Geologic Map of Arizona'            }
    {'AZGS HIGS Geology'                  }
    {'AZGS IDGS Geology'                  }
    {'aasggeothermal_AKBoreholeTemperatures'}
    {'aasggeothermal_AKaqSpringChemistry'  }
    {'aasggeothermal_AZActiveFaults'       }
```

Your results might be different because the WMS Database changes each release.

Input Arguments

`layers` — Layers to provide server URLs
array of `WMSLayer` objects

Layers to provide server URLs, specified as an array of `WMSLayer` objects.

Output Arguments

`titles` — Titles of unique WMS servers
cell array of character vectors

Titles of unique WMS servers, returned as a cell array of character vectors.

See Also
`wmsfind` | `servers`

Introduced in R2009b

shaderel

Construct `cdata` and colormap for shaded relief

Syntax

```
[cindx,cimap,clim] = shaderel(X,Y,Z,cmap)
[cindx,cimap,clim] = shaderel(X,Y,Z,cmap,[azim elev])
[cindx,cimap,clim] = shaderel(X,Y,Z,cmap,[azim elev],cmapl)
[cindx,cimap,clim] = shaderel(X,Y,Z,cmap,[azim elev],cmapl,clim)
```

Description

`[cindx,cimap,clim] = shaderel(X,Y,Z,cmap)` constructs the colormap and color indices to allow a surface to be displayed in colored shaded relief. The colors are proportional to the magnitude of Z, but modified by shades of gray based on the surface normals to simulate surface lighting. This representation allows both large and small-scale differences to be seen. X, Y, and Z define the surface. `cmap` is the colormap used to create the new shaded colormap `cimap`. `cindx` is a matrix of color indices to `cimap`, based on the elevation and surface normal of the Z matrix element. `clim` contains the color axis limits.

`[cindx,cimap,clim] = shaderel(X,Y,Z,cmap,[azim elev])` places the light at the specified azimuth and elevation. By default, the direction of the light is East (90º azimuth) at an elevation of 45º.

`[cindx,cimap,clim] = shaderel(X,Y,Z,cmap,[azim elev],cmapl)` chooses the number of grays to give a `cimap` of length `cmapl`. By default, the number of grayscales is chosen to keep the shaded colormap below 256. If the vector of azimuth and elevation is empty, the default locations are used.

`[cindx,cimap,clim] = shaderel(X,Y,Z,cmap,[azim elev],cmapl,clim)` uses the color limits to index Z into `cmap`.

Examples

Display the `peaks` surface with a shaded colormap:

```
[X,Y,Z] = peaks(100);
cmap = hot(16);
[cindx,cimap,clim] = shaderel(X,Y,Z,cmap);
surf(X,Y,Z,cindx); colormap(cimap); caxis(clim)
shading flat
```
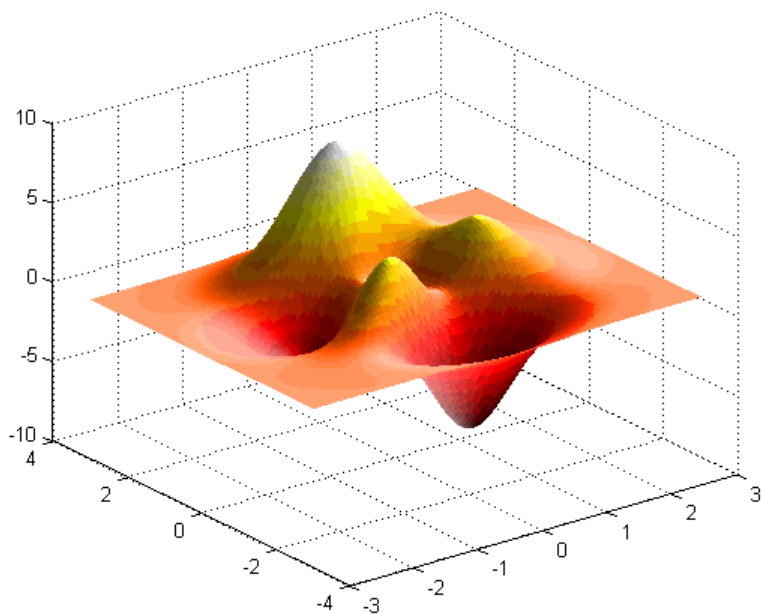
Tips

This function effectively multiplies two colormaps, one with color based on elevation, the other with a grayscale based on the slope of the surface, to create a new colormap. This produces an effect similar to using a light on a surface, but with all of the visible colors actually in the colormap. Lighting calculations are performed on the unprojected data.

See Also

caxis | colormap | light | meshlsrm | surf | surflsrm

Introduced before R2006a

shapeinfo

Information about shapefile

Syntax

```
info = shapeinfo(filename)
```

Description

`info = shapeinfo(filename)` returns a structure, `info`, whose fields contain information about the contents of a shapefile. `filename` can be the base name or the full name of any one of the component files. `shapeinfo` reads all three files as long as they exist in the same folder and the unit of length or angle is not specified. If the main file (with extension `.SHP`) is missing, `shapeinfo` returns an error. If either of the other files is missing, `shapeinfo` returns a warning.

Examples

Get CRS Information from Shapefile

Get projected or geographic CRS information from a shapefile by using the `shapeinfo` function.

First, return information about a shapefile as a structure. For this example, specify a shapefile that uses projected coordinates. Then, get information about the coordinate reference system by querying the `CoordinateReferenceSystem` field of the structure.

```
S = shapeinfo('boston_placenames.shp');
S.CoordinateReferenceSystem

ans =
  projcrs with properties:

                   Name: "NAD83 / Massachusetts Mainland"
          GeographicCRS: [1x1 geocrs]
       ProjectionMethod: "Lambert Conic Conformal (2SP)"
             LengthUnit: "meter"
   ProjectionParameters: [1x1 map.crs.ProjectionParameters]
```

Note that the value of the `CoordinateReferenceSystem` field is a `projcrs` object because the shapefile uses projected coordinates.

Input Arguments

`filename` — File name
string scalar | character vector

File name of the shapefile, specified as a string scalar or character vector. `filename` can be the base name or the full name of any one of the component files.

Output Arguments

`info` — Information about shapefile contents
structure

Information about shapefile contents, returned as a structure contains the following fields.

`Filename`	Char array containing the names of the files that were read
`ShapeType`	Character vector containing the shape type
`BoundingBox`	Numerical array of size 2-by-*N* that specifies the minimum (row 1) and maximum (row 2) values for each dimension of the spatial data in the shapefile
`Attributes`	Structure array of size 1-by-`numAttributes` that describes the attributes of the data. The structure contains these fields: • `Name` — Character vector containing the attribute name as given in the xBASE table • `Type` — Character vector specifying the MATLAB class of the attribute data returned by `shaperead`. The following attribute (xBASE) types are supported: Numeric, Floating, Character, and Date
`NumFeatures`	The number of spatial features in the shapefile
`CoordinateReferenceSystem`	Coordinate reference system (CRS), returned as a `geocrs` or `projcrs` object. The value of `CoordinateReferenceSystem` depends on the data contained in the file: • If the data is referenced to a geographic coordinate system, then `CoordinateReferenceSystem` is a `geocrs` object. • If the data is referenced to a projected coordinate system, then `CoordinateReferenceSystem` is a `projcrs` object. • If the file does not contain valid coordinate reference system information, then `CoordinateReferenceSystem` is empty.

See Also
shaperead | shapewrite

Topics
"Geographic Data Structures"

Introduced before R2006a

shaperead

Read vector features and attributes from shapefile

Syntax

```
S = shaperead(filename)
S = shaperead(filename,Name,Value)
[S,A] = shaperead( ___ )
```

Description

`S = shaperead(filename)` reads the shapefile, `filename`, and returns an *N*-by-1 geographic data structure array in projected map coordinates (a mapstruct). The geographic data structure combines geometric and feature attribute information. `shaperead` supports the ordinary 2-D shape types: `'Point'`, `'Multipoint'`, `'PolyLine'`, and `'Polygon'`.

`S = shaperead(filename,Name,Value)` returns a subset of the shapefile contents in `S`, as determined by the name-value pair arguments. Use `RecordNumbers`, `BoundingBox`, and `Selector` to select which features to read. If you use more than one of these parameters in the same call, you receive the intersection of the records that match the individual specifications. For instance, if you specify values for both `RecordNumbers` and `BoundingBox`, you import only those features with record numbers that appear in your list and that also have bounding boxes intersecting the specified bounding box.

If you do not specify any parameters, `shaperead` returns an entry for every non-null feature and creates a field for every attribute.

`[S,A] = shaperead( ___ )` returns an *N*-by-1 geographic data structure array, `S`, containing geometric information, and a parallel *N*-by-1 attribute structure array, `A`, containing feature attribute information.

Examples

Read Entire Shapefile

Read the entire shapefile called `concord_hydro_line.shp`, including the attributes in `concord_hydro_line.dbf`. The `shaperead` function returns a `mapstruct` with *x* and *y* coordinate vectors.

```
S = shaperead('concord_hydro_line.shp')

S=237×1 struct array with fields:
    Geometry
    BoundingBox
    X
    Y
    LENGTH
```

Read Portion of Shapefile Based on Bounding Box

Specify a bounding box to limit the data returned by shaperead . In addition, specify the names of the attributes you want to read.

```
bbox = [2.08 9.11; 2.09 9.12] * 1e5;
S = shaperead('concord_roads.shp','BoundingBox',bbox,...
              'Attributes',{'STREETNAME','CLASS'})
```

```
S=87×1 struct array with fields:
    Geometry
    BoundingBox
    X
    Y
    STREETNAME
    CLASS
```

Read road data by class from shapefile

Read road data only for class 4 and higher road segments that are at least 200 meters in length. Note the use of an anonymous function in the selector.

```
S = shaperead('concord_roads.shp','Selector',...
        {@(v1,v2) (v1 >= 4) && (v2 >= 200),'CLASS','LENGTH'})
```

```
S=115×1 struct array with fields:
    Geometry
    BoundingBox
    X
    Y
    STREETNAME
    RT_NUMBER
    CLASS
    ADMIN_TYPE
    LENGTH
```

Determine the number of roads of each class.

```
n = histcounts([S.CLASS],'BinLimits',[1 7],'BinMethod','integer')
```

```
n = 1×7

    0    0    0    7    93    15    0
```

Display a histogram of the number of roads that fall in each category of length.

```
figure
histogram([S.LENGTH])
```
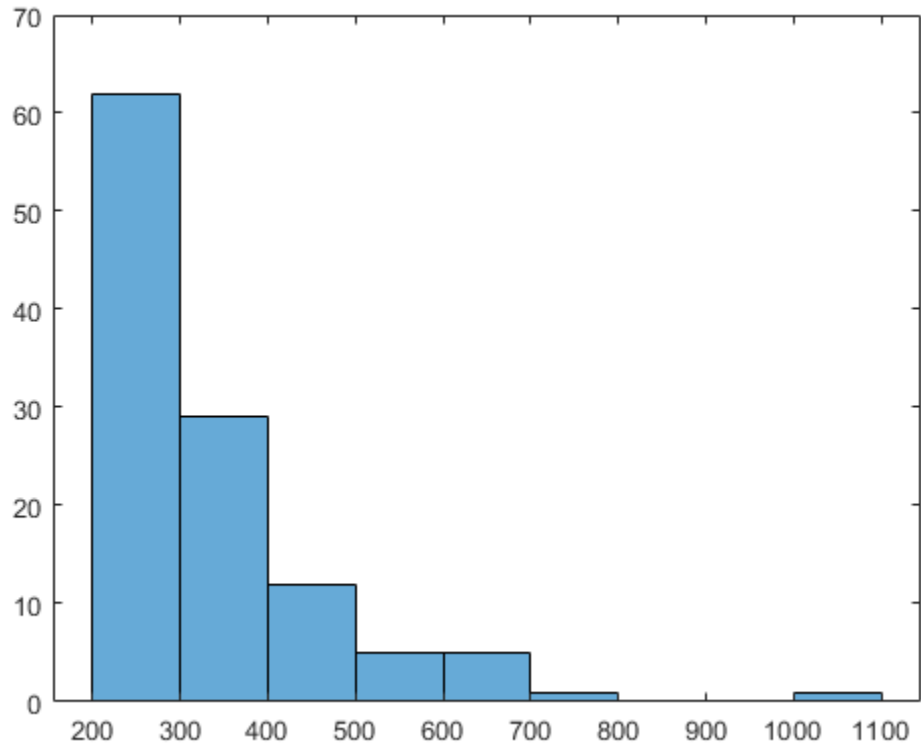
Read Shapefile with Geographic Coordinates

Specify that a shapefile uses latitude and longitude coordinates using the `'UseGeoCoords'` name-value pair.

For instance, return information about a shapefile as a structure. Verify the shapefile uses latitude and longitude coordinates by querying the `CoordinateReferenceSystem` field. The shapefile uses latitude and longitude coordinates if the field contains a `geocrs` object.

```
info = shapeinfo('landareas.shp');
crs = info.CoordinateReferenceSystem

crs =
  geocrs with properties:

           Name: "WGS 84"
          Datum: "World Geodetic System 1984"
       Spheroid: [1x1 referenceEllipsoid]
  PrimeMeridian: 0
      AngleUnit: "degree"
```

Read the shapefile by using the `shaperead` function. Indicate that the shapefile uses latitude and longitude coordinates using the `'UseGeoCoords'` name-value pair.

```
S = shaperead('landareas.shp','UseGeoCoords',true)

S=537×1 struct array with fields:
    Geometry
    BoundingBox
    Lon
    Lat
    Name
```

Note that the `shaperead` function returns a geographic data structure with latitude and longitude fields (a `geostruct`).

Input Arguments

`filename` — File name
character vector | string scalar

File name, specified as a string scalar or character vector. `filename` refers to the base name or full name of one of the component files in a shapefile. If the main file (with extension `.shp`) is missing, `shaperead` throws an error. If either of the other files is missing, `shaperead` issues a warning.

Make sure that your machine is set to the same character encoding scheme as the data you are importing. For example, if you are trying to import a shapefile that contains Japanese characters, configure your machine to support the `Shift-JIS` encoding scheme.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Attributes',{'STREETNAME','LENGTH'}`

`RecordNumbers` — Record numbers
vector of integers

Record numbers, specified as the comma-separated pair consisting of `'RecordNumbers'` and a vector of integers. Use the parameter `RecordNumbers` to import only features with listed record numbers.

Data Types: `double`

`BoundingBox` — Bounding box
2-by-2 matrix

Bounding box, specified as the comma-separated pair consisting of `'BoundingBox'` and a 2-by-2 matrix. `BoundingBox` has the form `[xmin,ymin;xmax,ymax]`, for map coordinates, or `[lonmin,latmin;lonmax,latmax]` for geographic coordinates. Use the parameter `BoundingBox` to import only features whose bounding boxes intersect the specified box. The `shaperead` function does not trim features that partially intersect the box.

Data Types: `double`

`Selector` — Selector
cell array

Selector, specified as the comma-separated pair consisting of `'Selector'` and a cell array containing a function handle and one or more attribute names. The function must return a `logical` scalar. Use the `Selector` parameter to import only features for which the function, when applied to the corresponding attribute values, returns `true`. For more information about function handles, see "Create Function Handle".

Attributes — Attribute names
cell array

Attribute names, specified as the comma-separated pair consisting of `'Attributes'` and a cell array of attribute names. Use the parameter `Attributes` to include listed attributes and set the order of attributes in the structure array. Use `{}` to omit all attributes.

UseGeoCoords — Flag to return shapefile contents in a geostruct
`false` (default) | `true`

Flag to return shapefile contents in a geostruct, specified as the comma-separated pair consisting of `'UseGeoCoords'` and `false` or `true`.

When `UseGeoCoords` is set to `true`, the shapefile contents are returned in a geostruct. Use this parameter when you know that the *x*- and *y*- coordinates in the shapefile actually represent longitude and latitude data.

To determine whether a shapefile uses latitude and longitude data, first use the `shapeinfo` function to return information about the shapefile as a structure. Then query the `CoordinateReferenceSystem` field of the structure. The shapefile uses geographic coordinates if `CoordinateReferenceSystem` is a `geocrs` object.

This code shows how to query the `CoordinateReferenceSystem` field of a structure associated with the shapefile `landareas.shp`.

```
info = shapeinfo('landareas.shp');
info.CoordinateReferenceSystem
```

Output Arguments

S — Vector geographic features
N-by-1 geographic data structure array

Vector geographic features, returned as an *N*-by-1 map geographic data structure array. Unless `UseGeoCoords` is `true`, `S` is a `mapstruct` and contains an element for each non-null, spatial feature in the shapefile.

A — Feature attribute information
N-by-1 attribute structure array

Feature attribute information, returned as an *N*-by-1 attribute structure array corresponding to array `S`.

The fields in the output structure arrays `S` and `A` depend on the type of shape contained in the file and the names and types of attributes included in the file. The `shaperead` function supports the following four attribute types: `numeric` and `floating` (stored as type `double` in MATLAB) and `character` and `date` (stored as `char array`).

Tips

To get information about the coordinate reference system (CRS) associated with a shapefile, use the `shapeinfo` function.

See Also

`shapeinfo` | `shapewrite` | `readgeotable`

Topics
"Geographic Data Structures"
"Find Geospatial Data Online"

Introduced before R2006a

shapewrite

Write geographic vector data structure to shapefile

Syntax

```
shapewrite(S,filename)
shapewrite(S,filename,'DbfSpec',dbfspec)
```

Description

`shapewrite(S,filename)` writes the vector geographic features stored in `S` to the file specified by `filename` in shapefile format.

`shapewrite(S,filename,'DbfSpec',dbfspec)` writes a shapefile in which the content and layout of the DBF file is controlled by `dbfspec`, a DBF specification.

Examples

Write Feature Data to Shapefile

Import a shapefile, remove a subset of the data, and write the modified data to a new shapefile.

Get information about a shapefile as a structure. The shapefile contains a network of road segments in Concord, MA.

```
shapeinfo("concord_roads.shp")
```

```
ans = struct with fields:
                    Filename: [3x84 char]
                   ShapeType: 'PolyLine'
                 BoundingBox: [2x2 double]
                 NumFeatures: 609
                  Attributes: [5x1 struct]
    CoordinateReferenceSystem: [1x1 projcrs]
```

The `NumFeatures` field indicates that the shapefile contains `609` features.

Import the shapefile as a geospatial table.

```
GT = readgeotable("concord_roads.shp");
```

The `CLASS` attribute of the shapefile includes information about road types. For example, a value of 2 indicates a multilane highway and a value of 6 indicates a minor road. Create a subtable containing major roads such that the value of `CLASS` is less than 4.

```
rows = GT.CLASS < 4;
majorGT = GT(rows,:);
```

Export the geospatial table as a shapefile.

```
shapewrite(majorGT,"main_concord_roads.shp")
```

Get information about the new shapefile as a structure.

```
shapeinfo("main_concord_roads.shp")

ans = struct with fields:
                       Filename: [3x89 char]
                      ShapeType: 'PolyLine'
                    BoundingBox: [2x2 double]
                    NumFeatures: 107
                     Attributes: [5x1 struct]
    CoordinateReferenceSystem: []
```

The new shapefile has 107 features instead of 609 features.

Write Data Stored in Planar Point Vector to Shapefile

Read a shapefile containing a vector of world cities. Store the data as a planar point vector using the `mappoint` function.

```
S = shaperead("worldcities.shp");
p = mappoint(S)

p =
 318x1 mappoint vector with properties:

 Collection properties:
    Geometry: 'point'
    Metadata: [1x1 struct]
 Feature properties:
           X: [-3.9509 54.7589 -0.2121 35.3894 38.7575 138.8528 44.5408 72.2474 ... ]
           Y: [5.2985 24.6525 5.6106 37.0613 9.0235 -34.6645 12.8767 22.7778 ... ]
        Name: {1x318 cell}
```

Append Paderborn, Germany to the point vector.

```
x = 51.715254;
y = 8.75213;
p = append(p,x,y,"Name","Paderborn")

p =
 319x1 mappoint vector with properties:

 Collection properties:
    Geometry: 'point'
    Metadata: [1x1 struct]
 Feature properties:
           X: [-3.9509 54.7589 -0.2121 35.3894 38.7575 138.8528 44.5408 72.2474 ... ]
           Y: [5.2985 24.6525 5.6106 37.0613 9.0235 -34.6645 12.8767 22.7778 ... ]
        Name: {1x319 cell}
```

The size of p has increased by 1.

Write the updated point vector to a shapefile.

```
shapewrite(p,"worldcities_updated.shp")
```

Input Arguments

S — Vector geographic features
geospatial table | `mappoint` vector | `mapshape` vector | mapstruct vector | `geopoint` vector | `geoshape` vector | geostruct vector

Vector geographic features, specified as one of the following:

- A geospatial table. The `shapewrite` function does not support geospatial tables containing more than one type of shape object. For more information about geospatial tables, see "Create Geospatial Tables".
- A `mappoint` vector.
- A `mapshape` vector.
- A map structure array (`mapstruct`) with X and Y coordinate fields.
- A `geopoint` vector.
- A `geoshape` vector.
- A geographic structure array (`geostruct`) with `Lat` and `Lon` fields.

S has the following restrictions on its attribute fields:

- Each attribute field value must be either a real, finite, scalar `double` or a character vector.
- The type of a given attribute must be consistent across all features.
- If S is a geospatial table containing `geopointshape`, `geolineshape`, or `geopolyshape` objects, a `geopoint` vector, a `geoshape` vector, or a geographic structure array, the `shapewrite` function writes the latitude and longitude values as *Y* and *X* coordinates, respectively.
- If a given attribute is integer-valued for all features, `shapewrite` writes it to the `[basename '.dbf']` file as an integer. If an attribute is non-integer-valued for any feature, `shapewrite` writes it as a fixed point decimal value with six digits to the right of the decimal place.

filename — File name
string scalar | character vector

File name and location of the shapefile to create, specified as a string scalar or character vector. If the file name includes a file extension, it must be `'.shp'` or `'.SHP'`. `shapewrite` creates three output files: `[basename '.shp']`, `[basename '.shx']`, and `[basename '.dbf']`, where `basename` is `filename` without its extension.

dbfspec — Feature attributes to include in shapefile
scalar structure

Feature attributes to include in the shapefile, specified as a scalar MATLAB structure containing one field for each feature attribute. Assign to that field a scalar structure with the following four fields:

- `FieldName` — The field name to be used in the file
- `FieldType` — The field type to be used in the file: `'N'` (numeric) or `'C'` (character)

- `FieldLength` — The field length in the file, in bytes
- `FieldDecimalCount` — For numeric fields, the number of digits to the right of the decimal place

To create a DBF spec, call `makedbfspec` and then modify the output to remove attributes or change the `FieldName`, `FieldLength`, or `FieldDecimalCount` for one or more attributes.

To include an attribute in the output file, specify a field in `dbfspec` with the same name as the attribute is specified in `S`.

Tips

- The xBASE (`.dbf`) file specifications require that geostruct and mapstruct attribute names are truncated to 11 characters when copied as DBF field names. Consider shortening long field names before calling `shapewrite`. By doing this, you make field names in the DBF file more readable and avoid introducing duplicate names as a result of truncation.
- Remember to set your character encoding scheme to match that of the geographic data structure you are exporting. For instance, if you are exporting a map that displays Japanese text, configure your machine to support `Shift-JIS` character encoding.

See Also
makedbfspec | shapeinfo | shaperead

Topics
"Geographic Data Structures"

Introduced before R2006a

showaxes

Toggle display of map coordinate axes

Syntax

```
showaxes(action)
showaxes
```

Description

showaxes(*action*) modifies the Cartesian axes based on the value of *action*, as defined in the Inputs section below.

showaxes toggles between displaying the default axes ticks on the MATLAB Cartesian axes and removing the axes ticks.

Input Arguments

action

A character vector or RGB triple that specifies how to modify the Cartesian axes

Default:

Value	Action
'on'	Displays the MATLAB Cartesian axes and default axes ticks
'off'	Removes the axes ticks from the MATLAB Cartesian axes
'hide'	Hides the Cartesian axes
'show'	Shows the Cartesian axes
'reset'	Sets the Cartesian axes to the default settings
'boxoff'	Removes axes ticks, color, and box from the Cartesian axes
colorstr	Sets the Cartesian axes to the color specified by *colorstr*
colorvec	Uses *colorvec* to set the Cartesian axes color

See Also
axesm

Introduced before R2006a

showm

Specify graphic objects to display on map axes

Syntax

```
showm
showm(handle)
showm(object)
```

Description

showm brings up a dialog box for selecting the objects to show (set their `Visible` property to `'on'`).

showm(handle) shows the objects specified by a vector of handles.

showm(*object*) shows those objects specified by the *object* character vector, which can be any character vector recognized by the handlem function.

See Also

clma | clmo | handlem | hidem | namem | tagm

Introduced before R2006a

size

Return size of geographic or planar vector

Syntax

```
sz = size(v)
sz = size(v,dim)
[m, n] = size(v)
```

Description

`sz = size(v)` returns the vector `[length(v), 1]`.

`sz = size(v,dim)` returns the length of geographic or planar vector `v` in the dimension specified by `dim`.

`[m, n] = size(v)` returns the length of `v` for `m` and 1 for `n`.

Examples

Find the Size of a Mapshape Vector

Create a mapshape vector from a `structArray`.

```
structArray = shaperead('worldrivers.shp');
ms = mapshape(structArray)

ms =
 128x1 mapshape vector with properties:

 Collection properties:
    Geometry: 'line'
    Metadata: [1x1 struct]
 Vertex properties:
  (128 features concatenated with 127 delimiters)
          X: [126.7796 126.5321 126.3121 126.2383 126.0362 NaN NaN 126.0362 ... ]
          Y: [73.4571 73.0669 72.8343 72.6010 72.2894 NaN NaN 72.2894 72.3784 ... ]
 Feature properties:
        Name: {1x128 cell}
```

Get the size of the mapshape vector.

```
sz = size(ms)

sz = 1×2

   128      1
```

Input Arguments

v — Geographic or planar vector
geopoint, geoshape, mappoint, or mapshape object

Geographic or planar vector, specified as a geopoint, geoshape, mappoint, or mapshape object.

dim — Dimension to measure length
positive integer scalar

Dimension to measure length of vector v, specified as a positive integer scalar.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | logical

Output Arguments

sz — Size of geographic or planar vector
nonnegative integer scalar or two-element vector

Size of vector v, returned as a nonnegative integer scalar or a two-element vector.

- When dim is not specified, sz is the two-element vector [length(v), 1].
- When dim is 1, sz is a scalar equal to the length of v.
- When dim ≥ 2, sz is 1.

Data Types: double

m — Size of vector in first dimension
nonnegative integer scalar

Size of vector v in the first dimension, returned as a nonnegative integer scalar. m is length(v).

Data Types: double

n — Size of vector in second dimension
1

Size of vector v in the second dimension, returned as the value 1.

Data Types: double

See Also
length | isprop

Introduced in R2012a

sizem

(To be removed) Row and column dimensions needed for regular data grid

---

Note `sizem` will be removed in a future release. Instead, create a geographic raster reference object, and then query its `RasterSize` property. For more information, see "Compatibility Considerations".

Syntax

```
[r,c] = sizem(latlim,lonlim,scale)
rc = sizem(latlim,lonlim,scale)
[r,c,refvec] = sizem(latlim,lonlim,scale)
```

Description

`[r,c] = sizem(latlim,lonlim,scale)` returns the required size for a regular data grid lying between the latitude and longitude limits specified by the two-element input vectors `latlim` and `lonlim`, which are of the form `[south-limit north-limit]` and `[west-limit and east-limit]`, respectively. The `scale` is the desired cells-per-degree measure of the desired data grid.

`rc = sizem(latlim,lonlim,scale)` returns the size of the matrix in one two-element vector.

`[r,c,refvec] = sizem(latlim,lonlim,scale)` also returns the three-element referencing vector geolocating the desired regular data grid.

Examples

How large a matrix would be required for a map of the world at a scale of 25 matrix cells per degree? (That's 25x25=625 cells per "square" degree.)

```
[r,c] = sizem([90,-90],[-180,180],25)

r =
        4500
c =
        9000
```

Bear in mind for memory purposes — 9000 x 4500 = 4.05 x 10^7 entries!

Compatibility Considerations

sizem will be removed
Not recommended starting in R2016b

Some functions that return referencing vectors will be removed, including the `sizem` function. Instead, create a geographic raster reference object using the `georefcells` function, and then query its `RasterSize` property. Reference objects have several advantages over referencing vectors.

- Unlike referencing vectors, reference objects have properties that document the size of the associated raster, its geographic limits, and the direction of its rows and columns. For examples of reference object properties, see the `GeographicPostingsReference` object.

- You can manipulate the limits of rasters associated with geographic reference objects using the `geocrop` function.

- You can manipulate the size and resolution of rasters associated with geographic reference objects using the `georesize` function.

- Most functions that accept referencing vectors as inputs also accept reference objects.

This table shows how to update your code to use the `georefcells` function instead of the `sizem` function.

Will Be Removed	Recommended

`[nrows,ncols] = sizem(latlim,lonlim,scale);`	`R = georefcells(latlim,lonlim,1/scale,1/scale);`
	`nrows = R.RasterSize(1);`
	`ncols = R.RasterSize(2);`

See Also
findm | NaN | ones | sparse | zeros | georefcells

Introduced before R2006a

sizesMatch

Package: `map.rasterref`

Determine if geographic or map raster object and image or raster are size-compatible

Syntax

```
tf = sizesMatch(R,A)
```

Description

`tf = sizesMatch(R,A)` determines whether geographic or map raster `R` is size-compatible with image or raster `A`.

Examples

Check If Image and Geographic Raster Are Size-Compatible

Create a `GeographicPostingsReference` raster reference object.

```
latlim = [0 90];
lonlim = [-180 180];
rasterSize = [91 361];
R = georefpostings(latlim,lonlim,rasterSize,'ColumnsStartFrom','north')

R =
  GeographicPostingsReference with properties:

              LatitudeLimits: [0 90]
             LongitudeLimits: [-180 180]
                  RasterSize: [91 361]
       RasterInterpretation: 'postings'
            ColumnsStartFrom: 'north'
              RowsStartFrom: 'west'
     SampleSpacingInLatitude: 1
    SampleSpacingInLongitude: 1
      RasterExtentInLatitude: 90
     RasterExtentInLongitude: 360
            XIntrinsicLimits: [1 361]
            YIntrinsicLimits: [1 91]
       CoordinateSystemType: 'geographic'
                GeographicCRS: []
                   AngleUnit: 'degree'
```

Create an arbitrary image (raster) with dimensions 91-by-361. Confirm that the image size is compatible with the geographic raster reference object.

```
A = ones(91,361);
tf_A = sizesMatch(R,A)
```

```
tf_A = logical
   1
```

Create an image of a different size. Confirm that this new image is not size-compatible with the geographic raster reference object.

```
B = ones(90,361);
tf_B = sizesMatch(R,B)

tf_B = logical
   0
```

Input Arguments

R — Geographic or map raster
GeographicCellsReference object | GeographicPostingsReference object | MapCellsReference object | MapPostingsReference object

Geographic or map raster, specified as a `GeographicCellsReference`, `GeographicPostingsReference`, `MapCellsReference`, or `MapPostingsReference` object.

A — Image or raster
numeric array or raster object

Image or raster, specified as a numeric array or raster object.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | logical | geographic raster object | map raster object

Output Arguments

tf — Flag indicating geographic or map raster is size-compatible with image or raster
logical scalar

Flag indicating geographic or map raster is size-compatible with image or raster, returned as a logical scalar. `tf` is `True` when `R.RasterSize` is equal to `[size(A,1) size(A,2)]` or `A.RasterSize`.

Data Types: logical

See Also

Introduced in R2013b

sm2deg

Convert spherical distance from statute miles to degrees

Syntax

```
deg = sm2deg(sm)
deg = sm2deg(sm,radius)
deg = sm2deg(sm,sphere)
```

Description

`deg = sm2deg(sm)` converts distances from statute miles to degrees, as measured along a great circle on a sphere with a radius of 3958.748 sm, the mean radius of the Earth.

`deg = sm2deg(sm,radius)` converts distances from statute miles to degrees, as measured along a great circle on a sphere having the specified radius.

`deg = sm2deg(sm,sphere)` converts distances from statute miles to degrees, as measured along a great circle on a sphere approximating an object in the Solar System.

Examples

Convert Statute Miles to Degrees on Earth

Convert 500 statute miles to degrees, as measured along a great circle on the Earth. By default, the `sm2deg` function assumes your input is a measurement of the Earth, which is represented as a sphere with a radius of 3958.748 statute miles.

```
deg = sm2deg(500)
```

```
deg = 7.2366
```

Convert Statute Miles to Degrees on Mars

Convert 500 statute miles to degrees, as measured along a great circle on Mars.

```
deg = sm2deg(500,'mars')
```

```
deg = 13.6001
```

Input Arguments

sm — Distance in statute miles
numeric array

Distance in statute miles, specified as a numeric array.

Data Types: `single` | `double`

radius — Radius
3958.748 (default) | numeric scalar

Radius of sphere in units of statute miles, specified as a numeric scalar.

sphere — Sphere
`'sun'` | `'moon'` | `'mercury'` | `'venus'` | `'earth'` | …

Sphere approximating an object in the Solar System, specified as one of the following values: `'sun'`, `'moon'`, `'mercury'`, `'venus'`, `'earth'`, `'mars'`, `'jupiter'`, `'saturn'`, `'uranus'`, `'neptune'`, or `'pluto'`. The value of `sphere` is case-insensitive.

Output Arguments

deg — Distance in degrees
numeric array

Distance in degrees, returned as a numeric array.

Data Types: `single` | `double`

See Also

deg2rad | rad2deg | deg2sm | sm2rad | km2deg | nm2deg

Introduced in R2007a

sm2km

Convert statute miles to kilometers

Syntax

```
km = sm2km(sm)
```

Description

`km = sm2km(sm)` converts distances from statute miles to kilometers.

See Also

deg2km | km2deg | km2rad | rad2km | deg2nm | nm2deg | nm2rad | rad2nm | deg2sm | sm2deg | deg2sm | sm2rad | rad2sm

Introduced in R2007a

sm2nm

Convert statute to nautical miles

Syntax

```
nm = sm2nm(sm)
```

Description

`nm = sm2nm(sm)` converts distances from statute to nautical miles.

See Also

deg2km | km2deg | km2rad | rad2km | deg2nm | nm2deg | nm2rad | rad2nm | deg2sm | sm2deg | deg2sm | sm2rad | rad2sm

Introduced in R2007a

sm2rad

Convert spherical distance from statute miles to radians

Syntax

```
rad = sm2rad(sm)
rad = sm2rad(sm,radius)
rad = sm2rad(sm,sphere)
```

Description

`rad = sm2rad(sm)` converts distances from statute miles to radians, as measured along a great circle on a sphere with a radius of 3958.748 sm, the mean radius of the Earth.

`rad = sm2rad(sm,radius)` converts distances from statute miles to radians, as measured along a great circle on a sphere having the specified radius.

`rad = sm2rad(sm,sphere)` converts distances from statute miles to radians, as measured along a great circle on a sphere approximating an object in the Solar System.

Examples

Convert Statute Miles to Radians on Earth

Convert 500 statute miles to radians, as measured along a great circle on the Earth. By default, the `sm2rad` function assumes your input is a measurement of the Earth, which is represented as a sphere with a radius of 3958.748 statute miles.

```
rad = sm2rad(500)
```

```
rad = 0.1263
```

Convert Statute Miles to Radians on Mars

Convert 500 statute miles to radians, as measured along a great circle on Mars.

```
rad = sm2rad(500,'mars')
```

```
rad = 0.2374
```

Input Arguments

sm — Distance in statute miles
numeric array

Distance in statute miles, specified as a numeric array.

Data Types: `single` | `double`

radius — Radius
3958.748 (default) | numeric scalar

Radius of sphere in units of statute miles, specified as a numeric scalar.

sphere — Sphere
`'sun'` | `'moon'` | `'mercury'` | `'venus'` | `'earth'` | ...

Sphere approximating an object in the Solar System, specified as one of the following values: `'sun'`, `'moon'`, `'mercury'`, `'venus'`, `'earth'`, `'mars'`, `'jupiter'`, `'saturn'`, `'uranus'`, `'neptune'`, or `'pluto'`. The value of `sphere` is case-insensitive.

Output Arguments

rad — Distance in radians
numeric array

Distance in radians, returned as a numeric array.

Data Types: `single` | `double`

See Also
deg2rad | rad2deg | sm2deg | nm2rad | km2rad | rad2sm

Introduced in R2007a

smoothlong

Remove discontinuities in longitude data

Compatibility

Note The `smoothlong` function is obsolete and has been replaced by `unwrapMultipart`, which requires input to be in radians. When working in degrees, use `rad2deg(unwrapMultipart(deg2rad(lon)))`.

Syntax

```
ang = smoothlong(angin)
ang = smoothlong(angin,angleunits)
```

Description

`ang = smoothlong(angin)` removes discontinuities in longitude data. The resulting angles can cover more than one revolution.

`ang = smoothlong(angin,angleunits)` uses the units defined by *angleunits*. If omitted, default units of `'degrees'` are assumed. Valid *angleunits* are:

- `'degrees'` — decimal degrees
- `'radians'`

See Also
unwrapMultipart

Introduced before R2006a

spcread

Read columns of data from ASCII text file

Syntax

```
mat = spcread
mat = spcread(filename)
mat = spcread(cols)
```

Description

`mat = spcread` reads an ASCII file of space-delimited data in two columns and returns the data in a matrix, `mat`. The file is selected by dialog box.

`mat = spcread(filename)` specifies the file from which to read by its name, given as the character vector `filename`.

`mat = spcread(cols)` specifies the number of columns of space-delimited data in the file with the integer `cols`. The default value of `cols` is 2.

Tips

The `spcread` function is similar to the standard MATLAB function `dlmread`. `spcread`, however, is much faster at reading large data sets of the type common for geographic purposes.

See Also
`nanclip`

Introduced before R2006a

spzerom

(To be removed) Construct sparse regular data grid of 0s

> **Note** spzerom will be removed in a future release. Use the `georefcells` and `sparse` functions instead. For more information, see "Compatibility Considerations".

Syntax

```
[Z,refvec] = spzerom(latlim,lonlim,scale)
```

Description

`[Z,refvec] = spzerom(latlim,lonlim,scale)` returns a sparse regular data grid consisting entirely of 0s and a three-element referencing vector for the returned Z. The two-element vectors `latlim` and `lonlim` define the latitude and longitude limits of the geographic region. They should be of the form `[south north]` and `[west east]`, respectively. The scalar `scale` specifies the number of rows and columns per degree of latitude and longitude.

Examples

```
[Z,refvec] = spzerom([46,51],[-79,-75],1)

Z =
   All zero sparse: 5-by-4
refvec =
     1    51   -79
```

Compatibility Considerations

spzerom will be removed
Not recommended starting in R2015b

Some functions that return referencing vectors will be removed, including the `spzerom` function. Instead, create a geographic raster reference object using the `georefcells` function and a sparse matrix of all zeros using the `sparse` function. Reference objects have several advantages over referencing vectors.

- Unlike referencing vectors, reference objects have properties that document the size of the associated raster, its geographic limits, and the direction of its rows and columns. For examples of reference object properties, see the `GeographicPostingsReference` object.

- You can manipulate the limits of rasters associated with geographic reference objects using the `geocrop` function.

- You can manipulate the size and resolution of rasters associated with geographic reference objects using the `georesize` function.

- Most functions that accept referencing vectors as inputs also accept reference objects.

This table shows how to update your code to use the `georefcells` and `sparse` functions instead of the `spzerom` function.

Will Be Removed	Recommended
`[Z,refvec] = spzerom(latlim,lonlim,scale);`	`R = georefcells(latlim,lonlim,1/scale,1/scale);` `Z = sparse(R.RasterSize(1),R.RasterSize(2));`

See Also
NaN | ones | zeros | sparse | georefcells

Introduced before R2006a

stdist

Standard distance for geographic points

Syntax

```
dist = stdist(lat,lon)
dist = stdist(lat,lon,units)
dist = stdist(lat,lon,ellipsoid)
dist = stdist(lat,lon,ellipsoid,units,method)
```

Description

`dist = stdist(lat,lon)` computes the average standard distance for geographic data. This function assumes that the data is distributed on a sphere. In contrast, `std` assumes that the data is distributed on a Cartesian plane. The result is a single value based on the great-circle distance of the data points from their geographic mean point. When `lat` and `lon` are vectors, a single distance is returned. When `lat` and `lon` are matrices, a row vector of distances is given, providing the distances for each column of `lat` and `lon`. N-dimensional arrays are not allowed. Distances are returned in degrees of angle units.

`dist = stdist(lat,lon,units)` indicates the angular units of the data. When the standard angle units is omitted, `'degrees'` is assumed. Output measurements are in terms of these `units` (as arc length distance).

`dist = stdist(lat,lon,ellipsoid)` specifies the shape of the Earth to be used with `ellipsoid`, which can be a `referenceSphere`, `referenceEllipsoid`, or `oblateSpheroid` object, or a vector of the form `[semimajor_axis eccentricity]`. The default is a unit sphere. Output measurements are in terms of the distance units of the semimajor axis of the `ellipsoid`.

`dist = stdist(lat,lon,ellipsoid,units,method)` specifies the method of calculating the standard distance of the data. The default, `'linear'`, is simply the average great circle distance of the data points from the centroid. Using `'quadratic'` results in the square root of the average of the squared distances, and `'cubic'` results in the cube root of the average of the cubed distances.

Background

The function `stdm` provides independent standard deviations in latitude and longitude of data points. `stdist` provides a means of examining data scatter that does not separate these components. The result is a *standard distance*, which can be interpreted as a measure of the scatter in the great circle distance of the data points from the centroid as returned by `meanm`.

The output distance can be thought of as the radius of a circle centered on the geographic mean position, which gives a measure of the spread of the data.

Examples

Create latitude and longitude lists using the `worldcities` data set and obtain standard distance deviation for group (compare with the example for `stdm`):

```
cities = shaperead('worldcities.shp', 'UseGeoCoords', true);
Paris = strcmp('Paris',{cities(:).Name});
London = strcmp('London',{cities(:).Name});
Rome = strcmp('Rome',{cities(:).Name});
Madrid = strcmp('Madrid',{cities(:).Name});
Berlin = strcmp('Berlin',{cities(:).Name});
Athens = strcmp('Athens',{cities(:).Name});
lat = [cities(Paris).Lat cities(London).Lat...
       cities(Rome).Lat cities(Madrid).Lat...
       cities(Berlin).Lat cities(Athens).Lat]
lon = [cities(Paris).Lon cities(London).Lon...
       cities(Rome).Lon cities(Madrid).Lon...
       cities(Berlin).Lon cities(Athens).Lon]

dist = stdist(lat,lon)

lat =
    48.8708    51.5188    41.9260    40.4312    52.4257    38.0164
lon =
     2.4131    -0.1300    12.4951    -3.6788    13.0802    23.5183
dist =
     8.1827
```

See Also

meanm | stdm

Introduced before R2006a

stdm

Standard deviation for geographic points

Syntax

```
[latdev,londev] = stdm(lat,lon)
[latdev,londev] = stdm(lat,lon,ellipsoid)
[latdev,londev] = stdm(lat,lon,units)
```

Description

`[latdev,londev] = stdm(lat,lon)` returns row vectors of the latitude and longitude geographic standard deviations for the data points specified by the columns of `lat` and `lon`.

`[latdev,londev] = stdm(lat,lon,ellipsoid)` specifies the shape of the Earth to be used by the `ellipsoid`, which can be a `referenceSphere`, `referenceEllipsoid`, or `oblateSpheroid` object, or a vector of the form `[semimajor_axis eccentricity]`. The default ellipsoid is a unit sphere. Output measurements are in terms of the distance units of the `ellipsoid` vector.

`[latdev,londev] = stdm(lat,lon,units)` indicates the angular units of the data. When you omit units, `'degrees'` is assumed. Output measurements are in terms of these units (as arc length distance).

If a single output argument is used, then `geodevs = [latdev longdev]`. This is particularly useful if the original `lat` and `lon` inputs are column vectors.

Background

Determining the deviations of geographic data in latitude and longitude is more complicated than simple sum-of-squares deviations from the data averages. For latitude deviation, a straightforward angular standard deviation calculation is performed from the *geographic mean* as calculated by `meanm`. For longitudes, a similar calculation is performed based on data *departure* rather than on angular deviation. See "Geographic Statistics for Point Locations on a Sphere" in the *Mapping Toolbox User's Guide*.

Examples

Create latitude and longitude lists using the `worldcities` data set and obtain standard distance deviation for group (compare with the example for `stdist`):

```
cities = shaperead('worldcities.shp', 'UseGeoCoords', true);
Paris = strcmp('Paris',{cities(:).Name});
London = strcmp('London',{cities(:).Name});
Rome = strcmp('Rome',{cities(:).Name});
Madrid = strcmp('Madrid',{cities(:).Name});
Berlin = strcmp('Berlin',{cities(:).Name});
Athens = strcmp('Athens',{cities(:).Name});
lat = [cities(Paris).Lat cities(London).Lat...
       cities(Rome).Lat cities(Madrid).Lat...
       cities(Berlin).Lat cities(Athens).Lat]
```

```
lon = [cities(Paris).Lon cities(London).Lon...
       cities(Rome).Lon cities(Madrid).Lon...
       cities(Berlin).Lon cities(Athens).Lon]
[latstd,lonstd]=stdm(lat,lon)

lat =
    48.8708    51.5188    41.9260    40.4312    52.4257    38.0164
lon =
     2.4131    -0.1300    12.4951    -3.6788    13.0802    23.5183
latstd =
     2.7640
lonstd =
    68.7772
```

See Also
departure | filterm | hista | histr | meanm | stdist

Introduced before R2006a

stem3m

Project stem plot on map axes

Syntax

```
h = stem3m(lat,lon,z)
h = stem3m(lat,lon,z,LineSpec)
h = stem3m(lat,lon,z,PropertyName,PropertyValue,...)
```

Description

h = stem3m(lat,lon,z) displays a stem plot on the current map axes. Stems are located at the points (lat,lon) and extend from an altitude of 0 to the values of z. The coordinate inputs should be in the same AngleUnits as the map axes. It is important to note that the selection of *z*-values will greatly affect the 3-D look of the plot. Regardless of AngleUnits, the *x* and *y* limits of the map axes are at most -π to +π and -π/2 to +π/2, respectively. This means that for most purposes, appropriate z values would be on the order of 1 to 3, not 10 to 30. The axes DataAspectRatio property can be used to adjust the appearance of the graphic. The handles of the displayed stem lines can be returned in h.

h = stem3m(lat,lon,z,LineSpec) specifies the line style, marker, and color on page 1-1348.

h = stem3m(lat,lon,z,PropertyName,PropertyValue,...) allows any property/value pair recognized by the MATLAB line function to be specified for the stems.

A stem plot displays data as lines extending normal to the *xy*-plane, in this case, on a map.

Examples

Project Stem Plot on Map Axes

Project a stem plot on a map axes.

```
load coastlines
axesm sinusoid;
view(3)
h = framem;
set(h,'zdata',zeros(size(coastlat)))
plotm(coastlat,coastlon)
ptlat = [0 30 30 -50 -78]';
ptlon = [0 30 -70 65 -35]';
ptz = [1 1.5 2 .5 1]';
stem3m(ptlat,ptlon,ptz,'r-')
```

More About

Line Style, Marker, and Color

Specify the line style, marker, and color as a character vector or string containing symbols. The symbols can appear in any order. You do not need to specify all three characteristics (line style, marker, and color). For example, specify a red dashed line using `'--r'`.

Line Style	Description	Resulting Line
`'-'`	Solid line	————————
`'--'`	Dashed line	– – – – –
`':'`	Dotted line	················
`'-.'`	Dash-dotted line	—·—·—·—·

Marker	Description	Resulting Marker
`'o'`	Circle	○
`'+'`	Plus sign	+

Marker	Description	Resulting Marker
'*'	Asterisk	✳
'.'	Point	•
'x'	Cross	✕
'_'	Horizontal line	—
'|'	Vertical line	|
's'	Square	□
'd'	Diamond	◇
'^'	Upward-pointing triangle	△
'v'	Downward-pointing triangle	▽
'>'	Right-pointing triangle	▷
'<'	Left-pointing triangle	◁
'p'	Pentagram	☆
'h'	Hexagram	✡

Color Name	Short Name	RGB Triplet	Appearance
'red'	'r'	[1 0 0]	
'green'	'g'	[0 1 0]	
'blue'	'b'	[0 0 1]	
'cyan'	'c'	[0 1 1]	
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

See Also

scatterm

Introduced before R2006a

str2angle

Convert text to angles in degrees

Syntax

```
angles = str2angle(strings)
```

Description

`angles = str2angle(strings)` converts string scalars or character vectors containing latitudes or longitudes, expressed in one of four different formats of degree-minutes-seconds, to numeric angles in units of degrees.

Format Description	Example
Degree Symbol, Single/Double Quotes	`'123°30''00"W'`
'd', 'm', 's' Separators	`'123d30m00sW'`
Minus Signs as Separators	`'123-30-00W'`
"Packed DMS"	`'1233000W'`

Input must conform closely to the examples provided; in particular, the seconds field must be included, even if it is not significant. Except in Packed DMS format, the seconds field can contain a fractional component. Sign characters are not supported. Instead, terminate each value with `'N'` for positive latitude, `'S'` for negative latitude, `'E'` for positive longitude, or `'W'` for negative longitude. `strings` is a string scalar, character vector, or a cell array of character vectors. For backward compatibility, `strings` can also be a character matrix. If more than one angle is represented, `strings` can either contain homogeneous or heterogeneous formatting (see example). `angles` is a column vector of class `double`.

Examples

```
strs = {'23°30''00"N', '23-30-00S', '123d30m00sE', '1233000W'}

strs =
    '23°30'00"N'    '23-30-00S'    '123d30m00sE'    '1233000W'

str2angle(strs)

ans =
        23.5
       -23.5
       123.5
      -123.5
```

See Also
angl2str

Introduced before R2006a

struct

Convert geographic or planar vector to scalar structure

Syntax

```
s = struct(v)
```

Description

`s = struct(v)` converts the geographic or planar vector `v` to a scalar structure, `s`.

Examples

Convert a Mappoint Vector into a Structure

Create a mappoint vector.

```
mp = mappoint(shaperead('tsunamis.shp'))

mp =
 162x1 mappoint vector with properties:

 Collection properties:
       Geometry: 'point'
       Metadata: [1x1 struct]
 Feature properties:
              X: [128.3000 -156 157.9500 143.8500 -155 -82.4000 159.5000 156.5000 ... ]
              Y: [-3.8000 19.5000 -9.0200 42.1500 19.1000 43.1000 52.7500 50 ... ]
           Year: [1950 1951 1951 1952 1952 1952 1952 1953 1953 1953 1953 1954 ... ]
          Month: [10 8 12 3 3 5 11 3 6 9 11 6 10 4 11 3 6 7 1 4 7 11 11 5 8 8 ... ]
            Day: [8 21 22 4 17 6 4 18 27 14 25 26 NaN 19 2 9 22 28 19 22 10 6 12 ... ]
           Hour: [3 10 NaN 1 3 NaN 16 NaN NaN 0 17 NaN NaN 20 NaN 14 23 8 14 ... ]
         Minute: [23 57 NaN 22 58 NaN 58 NaN NaN 26 48 NaN NaN 24 NaN 22 50 ... ]
         Second: [NaN NaN NaN 41 NaN NaN NaN NaN NaN 36 NaN NaN NaN NaN NaN ... ]
       Val_Code: [2 4 2 4 4 1 4 3 3 4 4 1 2 4 3 4 3 4 4 2 4 4 2 4 4 4 4 4 4 ... ]
       Validity: {1x162 cell}
     Cause_Code: [1 1 6 1 1 9 1 1 6 3 1 9 0 1 1 1 1 1 1 1 3 1 1 1 0 3 1 1 ... ]
          Cause: {1x162 cell}
         Eq_Mag: [7.6000 6.9000 NaN 8.1000 4.5000 NaN 9 5.8000 NaN 6.8000 ... ]
        Country: {1x162 cell}
       Location: {1x162 cell}
     Max_Height: [2.8000 3.6000 6 6.5000 1 1.5200 18 1.5000 1.4000 3 3 3 ... ]
       Iida_Mag: [1.5000 1.8000 2.6000 2.7000 NaN NaN 4.2000 0.6000 0.5000 ... ]
      Intensity: [1.5000 NaN NaN 2 NaN NaN 4 NaN 1 1.5000 1.5000 NaN NaN ... ]
     Num_Deaths: [NaN NaN NaN 33 NaN NaN 2236 NaN NaN NaN NaN 7 NaN 3 NaN ... ]
    Desc_Deaths: [NaN NaN NaN 1 NaN NaN 3 NaN NaN NaN NaN 1 NaN 1 NaN ... ]
```

Convert the mappoint vector into a structure.

```
s = struct(mp)
```

```
s = struct with fields:
       Geometry: 'point'
       Metadata: [1x1 struct]
              X: [128.3000 -156 157.9500 143.8500 -155 -82.4000 159.5000 ... ]
              Y: [-3.8000 19.5000 -9.0200 42.1500 19.1000 43.1000 ... ]
           Year: [1950 1951 1951 1952 1952 1952 1952 1953 1953 1953 1953 ... ]
          Month: [10 8 12 3 3 5 11 3 6 9 11 6 10 4 11 3 6 7 1 4 7 11 11 ... ]
            Day: [8 21 22 4 17 6 4 18 27 14 25 26 NaN 19 2 9 22 28 19 22 ... ]
           Hour: [3 10 NaN 1 3 NaN 16 NaN NaN 0 17 NaN NaN 20 NaN 14 23 ... ]
         Minute: [23 57 NaN 22 58 NaN 58 NaN NaN 26 48 NaN NaN 24 NaN 22 ... ]
         Second: [NaN NaN NaN 41 NaN NaN NaN NaN NaN 36 NaN NaN NaN NaN ... ]
       Val_Code: [2 4 2 4 4 1 4 3 3 4 4 1 2 4 3 4 3 4 4 2 4 4 2 4 4 4 4 ... ]
       Validity: {1x162 cell}
     Cause_Code: [1 1 6 1 1 9 1 1 6 3 1 9 0 1 1 1 1 1 1 1 3 1 1 1 0 3 1 ... ]
          Cause: {1x162 cell}
         Eq_Mag: [7.6000 6.9000 NaN 8.1000 4.5000 NaN 9 5.8000 NaN ... ]
        Country: {1x162 cell}
       Location: {1x162 cell}
     Max_Height: [2.8000 3.6000 6 6.5000 1 1.5200 18 1.5000 1.4000 3 3 3 ... ]
       Iida_Mag: [1.5000 1.8000 2.6000 2.7000 NaN NaN 4.2000 0.6000 ... ]
      Intensity: [1.5000 NaN NaN 2 NaN NaN 4 NaN 1 1.5000 1.5000 NaN NaN ... ]
     Num_Deaths: [NaN NaN NaN 33 NaN NaN 2236 NaN NaN NaN NaN 7 NaN 3 ... ]
    Desc_Deaths: [NaN NaN NaN 1 NaN NaN 3 NaN NaN NaN NaN 1 NaN 1 NaN ... ]
```

Input Arguments

v — Geographic or planar vector
geopoint, geoshape, mappoint, or mapshape object

Geographic or planar vector, specified as a geopoint, geoshape, mappoint, or mapshape object.

Output Arguments

s — Structure representing a geographic or planar vector
scalar structure

Structure representing a geographic or planar vector, returned as a scalar structure.

See Also
length | properties

Introduced in R2012a

struct2geotable

Convert structure array to geospatial table

Syntax

```
GT = struct2geotable(S)
GT = struct2geotable(S,coordinateSystemType,fnames)
GT = struct2geotable(S,Name=Value)
```

Description

GT = struct2geotable(S) converts the structure array S to a geospatial table GT. The function creates the Shape variable of GT by using the latitude-longitude or *x*-*y* coordinates in S. Each field of S is a variable of GT.

GT = struct2geotable(S,coordinateSystemType,fnames) creates the Shape variable of GT by using the coordinate system type coordinateSystemType and structure array fields fnames.

GT = struct2geotable(S,Name=Value) specifies options using name-value arguments. For example, specify that coordinates represent points, lines, or polygons by using the GeometryType name-value argument.

Examples

Convert Structure Array to Geospatial Table

Read a shapefile containing hydrographic data for Concord, MA, as a structure array. Import only the polygons with an area greater than 100 square kilometers. The *x*- and *y*-coordinates are in the X and Y fields of the structure.

```
areafilterfcn = @(areafilter) (areafilter > 100000);
S = shaperead("concord_hydro_area.shp", ...
    Selector={areafilterfcn,'AREA'});
```

Get the coordinate reference system for the data.

```
info = shapeinfo("concord_hydro_area.shp");
crs = info.CoordinateReferenceSystem;
```

Convert the structure array to a geospatial table. The struct2geotable function detects the X and Y fields and uses them to create the Shape variable. Specify the coordinate reference system for the Shape variable.

```
GT = struct2geotable(S, ...
    CoordinateReferenceSystem=crs);
```

View the shape object in the second row of the table. Shape object properties contain information such as the number of holes.

```
GT.Shape(2)
```

```
ans =
  mappolyshape with properties:

              NumRegions: 1
                NumHoles: 3
                Geometry: "polygon"
    CoordinateSystemType: "planar"
            ProjectedCRS: [1x1 projcrs]
```

Display the polygons.

```
mapshow(GT,FaceColor="c")
```
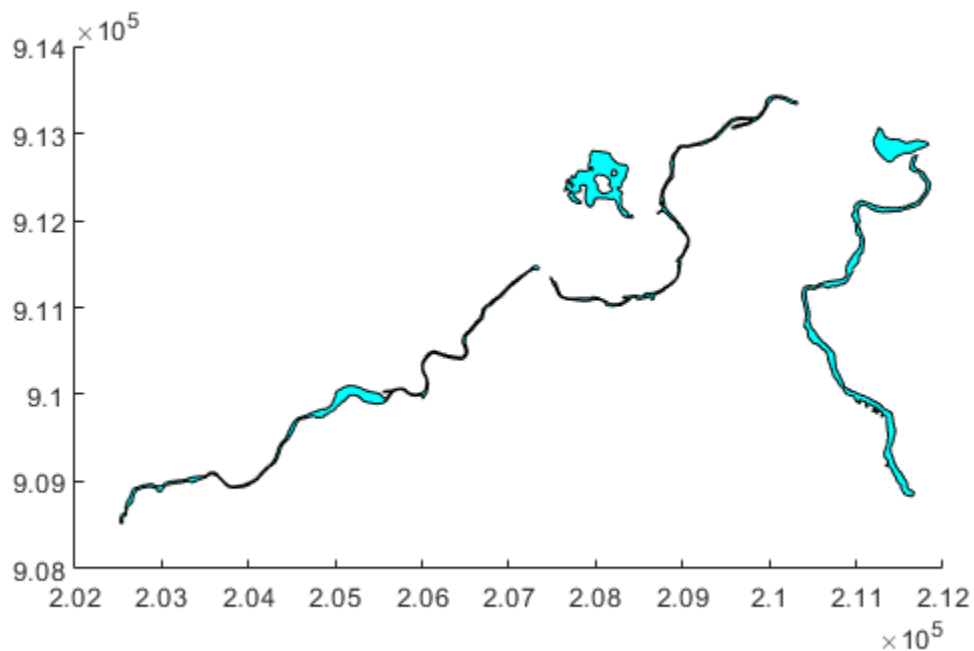

Specify Fields Containing Coordinate Data

Create a sample structure array containing the locations of cities. The `Lats` field contains latitudes, the `Lons` field contains longitudes, and the `Names` field contains names.

```
lats = {35.7082,-22.8842,51.5074,39.9042,37.9838};
lons = {139.6401,-43.3882,-0.1278,116.4074,23.7275};
n = {'Tokyo','Rio de Janeiro','London','Beijing','Athens'};
S = cell2struct([lats;lons;n],{'Lats','Lons','Names'},1)
```

```
S=5×1 struct array with fields:
    Lats
```

```
    Lons
    Names
```

Convert the structure array to a geospatial table. The `struct2geotable` function does not automatically detect the field names `Lats` and `Lons`. Specify the coordinate system type as `"geographic"` and the field names containing the coordinates.

`GT = struct2geotable(S,"geographic",["Lats" "Lons"])`

```
GT=5×4 table
            Shape              Lats       Lons          Names
    _____   _____   _____   _____

    (35.7082°N, 139.6401°E)     35.708    139.64    {'Tokyo'         }
    (22.8842°S,  43.3882°W)    -22.884   -43.388    {'Rio de Janeiro'}
    (51.5074°N,   0.1278°W)     51.507    -0.1278   {'London'        }
    (39.9042°N, 116.4074°E)     39.904    116.41    {'Beijing'       }
    (37.9838°N,  23.7275°E)     37.984    23.727    {'Athens'        }
```

View the `Shape` variable of the geospatial table.

`GT.Shape`

```
ans =
  5x1 geopointshape array with properties:

              NumPoints: [5x1 double]
               Latitude: [5x1 double]
              Longitude: [5x1 double]
               Geometry: "point"
     CoordinateSystemType: "geographic"
           GeographicCRS: []
```

Input Arguments

S — Input structure array
structure array

Input structure array.

The `struct2geotable` function detects the coordinates when the structure array fields have these names, ignoring case.

- Latitude coordinates — `Latitude` or `Lat`
- Longitude coordinates — `Longitude`, `Lon`, or `Long`
- *x*-coordinates — X
- *y*-coordinates — Y

If you specify an input structure array with a field called `Shape`, then the `struct2geotable` function overwrites it.

Data Types: `struct`

coordinateSystemType — Coordinate system type
`"planar"` | `"geographic"`

Coordinate system type, specified as one of these options:

- `"planar"` — Coordinates are in a planar coordinate system.
- `"geographic"` — Coordinates are in a geographic coordinate system.

Data Types: `string`

fnames — Names of structure array fields
two-element string vector

Names of the structure array fields used to create the `Shape` variable of the geospatial table, specified as a two-element string vector, such as `["Lats" "Lons"]`. The fields contain latitude-longitude or *x-y* coordinates.

Data Types: `string`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `struct2geotable(S,CoordinateReferenceSystem=geocrs(4326))` specifies the geographic CRS as the World Geodetic System of 1984, which has EPSG code `4326`.

CoordinateReferenceSystem — Coordinate reference system
`projcrs` object | `geocrs` object

Coordinate reference system (CRS) to use when creating the `Shape` variable of the geospatial table, specified as a `projcrs` or `geocrs` object.

Specify a `projcrs` object when `coordinateSystemType` is `"planar"` and specify a `geocrs` object when `coordinateSystemType` is `"geographic"`.

GeometryType — Geometry type
`"point"` (default) | `"line"` | `"polygon"`

Geometry type of the coordinate fields, specified as one of these options:

- `"point"` — The coordinate fields represent points.
- `"line"` — The coordinate fields represent lines.
- `"polygon"` — The coordinate fields represent polygons.

When you specify the geometry type as `"polygon"`, the `struct2geotable` function assumes that the coordinates define polygons with a valid topology. A polygon has a valid topology when:

- Region interiors are to the right as you trace boundaries from vertex to vertex.
- The boundaries have no self-intersections.

In general, the outer boundaries of polygons with a valid topology have vertices in a clockwise order and the interior holes have vertices in a counterclockwise order.

The `struct2geotable` function detects the geometry type when the input structure array has a Geometry field with a value of `'Point'`, `'Multipoint'`, `'Line'`, or `'Polygon'`.

Output Arguments

GT — Output geospatial table
geospatial table

Output geospatial table. A geospatial table is a `table` or `timetable` object with a Shape variable that contains `geopointshape`, `geolineshape`, `geopolyshape`, `mappointshape`, `maplineshape`, or `mappolyshape` objects.

For an input structure array with M elements and N fields, the size of GT is M-by-($N + 1$).

Tips

- When the input structure has latitude-longitude fields and x-y fields, the `struct2geotable` function creates the geospatial table using the latitude-longitude fields. Create the geospatial table using the x-y fields by specifying the `coordinateSystemType` and `fnames` arguments.

- When you know that a shapefile contains latitude and longitude coordinates and read the shapefile using the `shaperead` function, you must specify the UseGeoCoords argument as `true`. Otherwise, the structure array contains X and Y fields instead of Lat and Lon fields. As a result, the `struct2geotable` function creates the Shape variable of the geospatial table using planar shape objects such as `mappointshape` instead of geographic shape objects such as `geopointshape`.

See Also

Functions
readgeotable | table2geotable

Topics
"Create Geospatial Tables"

Introduced in R2021b

surfacem

Project and add geolocated data grid to current map axes

Syntax

```
surfacem(lat,lon,Z)
surfacem(latlim,lonlim,Z)
surfacem(lat,lon,Z,alt)
surfacem(...,prop1,val1,prop2,val2,...)
h = surfacem(...)
```

Description

`surfacem(lat,lon,Z)` constructs a surface to represent the data grid `Z` in the current map axes. The surface lies flat in the horizontal plane with its `CData` property set to Z. The vectors or 2-D arrays `lat` and `lon` define the latitude-longitude graticule mesh on which `Z` is displayed. For a complete description of the various forms that `lat` and `lon` can take, see `surfm`.

`surfacem(latlim,lonlim,Z)` defines the graticule using the latitude and longitude limits `latlim` and `lonlim`. These limits should match the geographic extent of the data grid Z. The two-element vector `latlim` has the form:

```
[southern_limit northern_limit]
```

Likewise, `lonlim` has the form:

```
[western_limit eastern_limit]
```

A latitude-longitude graticule of size 50-by-100 is constructed. The surface `FaceColor` property is `'texturemap'`, except when Z is precisely 50-by-100, in which case it is `'flat'`.

`surfacem(lat,lon,Z,alt)` sets the `ZData` property of the surface to `'alt'`, resulting in a 3-D surface. `Lat` and `lon` must result in a graticule mesh that matches `alt` in size. `CData` is set to Z. `Facecolor` is `'texturemap'`, unless Z matches `alt` in size, in which case it is `'flat'`.

`surfacem(...,prop1,val1,prop2,val2,...)` applies additional MATLAB graphics properties to the surface via property/value pairs. You can specify any property accepted by the `surface` function, except `XData`, `YData`, and `ZData`.

`h = surfacem(...)` returns a handle to the surface object.

Note Unlike `meshm` and `surfm`, `surfacem` always adds a surface to the current axes, regardless of hold state.

Examples

Load elevation raster data and a geographic cells reference object. Then, display the data as a surface.

```
load topo60c
[topo60c,topo60cR] = geocrop(topo60c,topo60cR,[-90 90],[-180,180]);
[lat,lon] = geographicGrid(topo60cR);
worldmap world
surfacem(lat,lon,topo60c)
demcmap(topo60c)
```
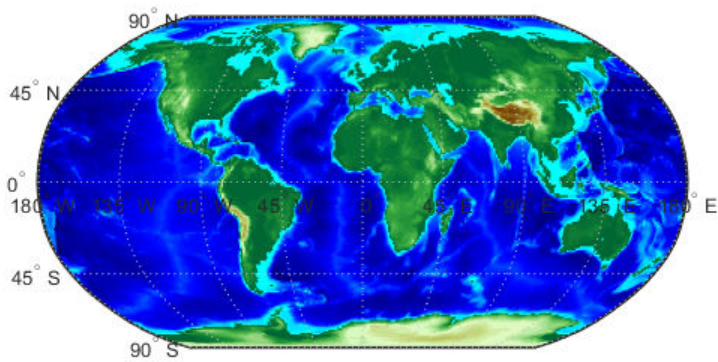

See Also

geoshow | meshm | pcolorm | surfm

Introduced before R2006a

surflm

3-D shaded surface with lighting on map axes

Syntax

```
surflm(lat,lon,Z)
surflm(latlim,lonlim,Z)
surflm(...,s)
surflm(...,s,k)
h = surflm(...)
```

Description

surflm(lat,lon,Z) and surflm(latlim,lonlim,Z) are the same as surfm(...) except that they highlight the surface with a light source. The default light source (45 degrees counterclockwise from the current view) and reflectance constants are the same as in surfl.

surflm(...,s) and surflm(...,s,k) use a light source vector, s, and a vector of reflectance constants, k. For more information on s and k, see the help for surfl.

h = surflm(...) returns a handle to the surface object.

Examples

Display 3-D Shaded Surface with Lighting

Display a 3-D shaded surface with lighting on a map. To do this, first load elevation raster data and a geographic cells reference object. Get the coordinates of coastlines.

```
load topo60c
[topo60c,topo60cR] = geocrop(topo60c,topo60cR,[-90 90],[-180 180]);
load coastlines
```

Create a map axes object using a Miller projection. Remove the axes background by calling axis off.

```
axesm miller
axis off
```

Create a latitude-longitude mesh from the raster using the geographicGrid function. Then, display the elevation data as a shaded surface with lighting. Apply a grayscale colormap.

```
[lat,lon] = geographicGrid(topo60cR);
surflm(lat,lon,topo60c)
colormap(gray)
```

Display the coastlines over the surface.

```
plotm(coastlat,coastlon,max(topo60c(:)),...
        'LineWidth',1.5,'Color','y')
```
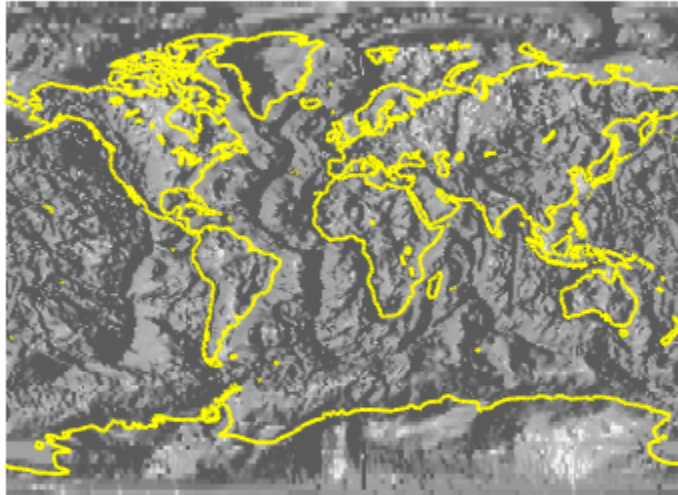
Tips

`surflm` is like `surfm`, except that it shades the monochrome map surface with a light source, and the only allowed graticule is the size of the data matrix.

See Also

`surfm`

Introduced before R2006a

surflsrm

3-D lighted shaded relief of geolocated data grid

Syntax

```
surflsrm(lat,long,Z)
surflsrm(lat,long,Z,[azim elev])
surflsrm(lat,long,Z,[azim elev],cmap)
surflsrm(lat,long,Z,[azim elev],cmap,clim)
h = surflsrm(...)
```

Description

`surflsrm(lat,long,Z)` displays the geolocated data grid, colored according to elevation and surface slopes. The current axes must have a valid map projection definition.

`surflsrm(lat,long,Z,[azim elev])` displays the geolocated data grid with the light coming from the specified azimuth and elevation. Lighting is applied before the data is projected. Angles are in degrees, with the azimuth measured clockwise from North, and elevation up from the zero plane of the surface. By default, the direction of the light source is east (90º azimuth) at an elevation of 45º.

`surflsrm(lat,long,Z,[azim elev],cmap)` displays the geolocated data grid using the provided colormap. The number of grayscales is chosen to keep the size of the shaded colormap below 256. By default, the colormap is constructed from 16 colors and 16 grays. If the vector of azimuth and elevation is empty, the default locations are used.

`surflsrm(lat,long,Z,[azim elev],cmap,clim)` uses the provided color axis limits, which are, by default, automatically computed from the data.

`h = surflsrm(...)` returns the handle to the surface drawn.

Examples

Create a new colormap using `demcmap` with white colors for the sea and default colors for land. Use this colormap for the lighted shaded relief map of the Middle East region:

```
load mapmtx
[cmap,clim] = demcmap(map1,[],[1 1 1],[]);
axesm loximuth
surflsrm(lt1,lg1,map1,[],cmap,clim)
```
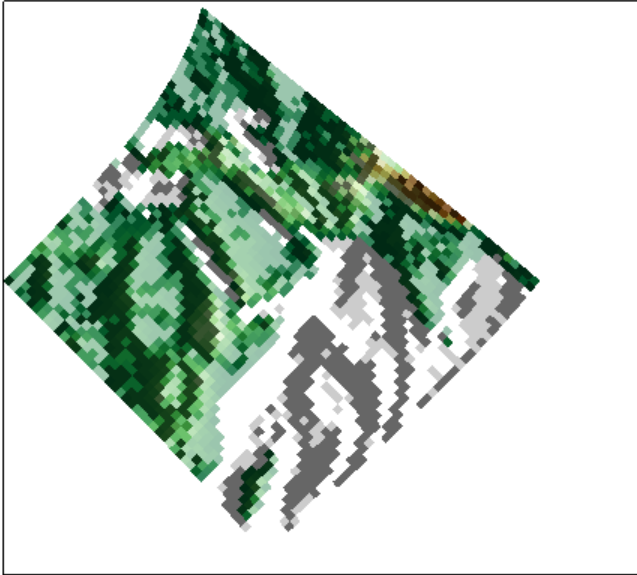
Tips

This function effectively multiplies two colormaps, one with color based on elevation, the other with a grayscale based on the slope of the surface, to create a new colormap. This produces an effect similar to using a light on a surface, but with all of the visible colors actually in the colormap. Lighting calculations are performed on the unprojected data.

See Also

meshlsrm | meshm | pcolorm | shaderel | surfacem | surflm | surfm

Introduced before R2006a

surfm

Project geolocated data grid on map axes

Syntax

```
surfm(lat,lon,Z)
surfm(latlim,lonlim,Z)
surfm(lat,lon,Z,alt)
surfm(...,prop1,val1,prop2,val2,...)
h = surfm(...)
```

Description

`surfm(lat,lon,Z)` constructs a surface to represent the data grid `Z` in the current map axes. The surface lies flat in the horizontal plane with its `CData` property set to `Z`. The 2-D arrays or vectors `lat` and `lon` define the latitude-longitude graticule mesh on which Z is displayed. The sizes and shapes of `lat` and `lon` affect their interpretation, and also determine whether the default `FaceColor` property of the surface is `'flat'` or `'texturemap'`. There are three options:

- 2-D arrays (matrices) having the same size as `Z`. Lat and `lon` are treated as geolocation arrays specifying the precise location of each vertex. `FaceColor` is `'flat'`.

- 2-D arrays having a different size than `Z`. The arrays `lat` and `lon` define a graticule mesh that might be either larger or smaller than `Z`. Lat and `lon` must match each other in size. `FaceColor` is `'texturemap'`.

- Vectors having more than two elements. The elements of `lat` and `lon` are repeated to form a graticule mesh with size equal to `numel(lat)-by-numel(lon)`. `FaceColor` is `'flat'` if the graticule mesh matches `Z` in size. Otherwise, `FaceColor` is `'texturemap'`.

`surfm` clears the current map if the hold state is `'off'`.

`surfm(latlim,lonlim,Z)` defines the graticule using the latitude and longitude limits `latlim` and `lonlim`, which should match the geographic extent of the data grid `Z`. Latlim is a two-element vector of the form:

`[southern_limit northern_limit]`

Likewise `lonlim` has the form:

`[western_limit eastern_limit]`

A latitude-longitude graticule is constructed to match `Z` in size. The surface `FaceColor` property is `'flat'` by default.

`surfm(lat,lon,Z,alt)` sets the `ZData` property of the surface to `'alt'`, resulting in a 3-D surface. `lat` and `lon` must result in a graticule mesh that matches `alt` in size. `CData` is set to `Z`. The `FaceColor` property is `'texturemap'`, unless `Z` matches `alt` in size, in which case it is `'flat'`.

`surfm(...,prop1,val1,prop2,val2,...)` applies additional MATLAB graphics properties to the surface via property/value pairs. You can specify any property accepted by the `surface` function except `XData`, `YData`, and `ZData`.

`h = surfm(...)` returns a handle to the surface object.

Examples

Load elevation raster data and a geographic cells reference object. Then, display the data as a surface.

```
load topo60c
axesm miller
axis off
framem on
gridm on
[topo60c,topo60cR] = geocrop(topo60c,topo60cR,[-90 90],[-180,180]);
[lat,lon] = geographicGrid(topo60cR);
surfm(lat,lon,topo60c)
demcmap(topo60c)
```
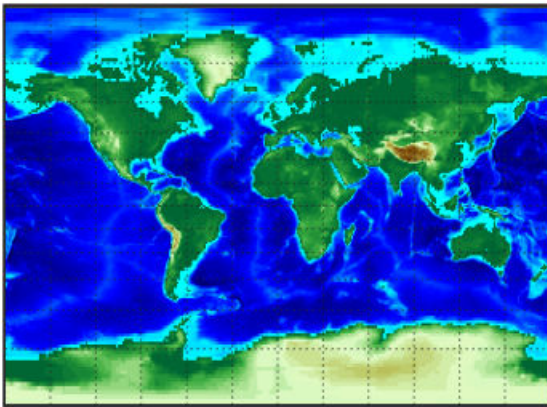

Tips

This function warps a data grid to a graticule mesh, which is projected according to the map axes property `MapProjection`. The fineness, or resolution, of this grid determines the quality of the projection and the speed of plotting it. There is no hard and fast rule for sufficient graticule resolution, but in general, cylindrical projections need very few graticule points in the longitudinal direction, while complex curve-generating projections require more.

See Also
geoshow | geographicGrid | meshm | pcolorm | surfacem

Introduced before R2006a

symbolm

Project point markers with variable size

Syntax

```
symbolm(lat,lon,z,'MarkerType')
symbolm(lat,lon,z,'MarkerType','PropertyName',PropertyValue,...)
h = symbolm(...)
```

Description

`symbolm(lat,lon,z,'MarkerType')` constructs a thematic map where the symbol size of each data point (`lat, lon`) is proportional to it weighting factor (`z`). The point corresponding to `min(z)` is drawn at the default marker size, and all other points are plotted with proportionally larger markers. `MarkerType` specifies the marker and optionally a color. For more information, see "Markers and Colors" on page 1-1366.

`symbolm(lat,lon,z,'MarkerType','PropertyName',PropertyValue,...)` applies the line properties to all the symbols drawn.

`h = symbolm(...)` returns a vector of handles to the projected symbols. Each symbol is projected as an individual line object.

See also

`stem3m`, `plotm`, `plot`
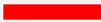
More About

Markers and Colors

Specify the marker and color as a character vector or string containing symbols. The symbols can appear in any order. You do not need to specify a color. For example, specify a red plus sign using `'r +'`.

Marker	Description	Resulting Marker
`'o'`	Circle	○
`'+'`	Plus sign	+
`'*'`	Asterisk	✳
`'.'`	Point	•
`'x'`	Cross	✕

Marker	Description	Resulting Marker
'_'	Horizontal line	—
'|'	Vertical line	|
's'	Square	□
'd'	Diamond	◇
'^'	Upward-pointing triangle	△
'v'	Downward-pointing triangle	▽
'>'	Right-pointing triangle	▷
'<'	Left-pointing triangle	◁
'p'	Pentagram	☆
'h'	Hexagram	✡

Color Name	Short Name	Appearance
'red'	'r'	
'green'	'g'	
'blue'	'b'	
'cyan'	'c'	
'magenta'	'm'	
'yellow'	'y'	
'black'	'k'	
'white'	'w'	

Introduced before R2006a

table2geotable

Convert table to geospatial table

Syntax

```
GT = table2geotable(T)
GT = table2geotable(T,coordinateSystemType,varnames)
GT = table2geotable( ___ ,Name=Value)
```

Description

`GT = table2geotable(T)` converts the table or timetable `T` to a geospatial table `GT`. The function creates the `Shape` variable of `GT` by using the latitude-longitude or *x-y* coordinates in `T`. Each variable of `T` is a variable of `GT`.

`GT = table2geotable(T,coordinateSystemType,varnames)` creates the `Shape` variable of `GT` by using the coordinate system type `coordinateSystemType` and table variables `varnames`. Use this syntax to create the `Shape` variable from well-known text (WKT) string representations of geometry or from coordinate data.

`GT = table2geotable( ___ ,Name=Value)` specifies options using name-value arguments. For example, specify that coordinates represent points, lines, or polygons by using the `GeometryType` name-value argument. Use this syntax with any of the input arguments in the previous syntaxes.

Examples

Convert Table to Geospatial Table

Import data about tsunami events as a table. The latitude and longitude coordinates of the origin points are in the `Latitude` and `Longitude` table variables.

```
T = readtable("tsunamis.xlsx");
```

Convert the table to a geospatial table. The `table2geotable` function detects the `Latitude` and `Longitude` variables and uses them to create the `Shape` variable.

```
GT = table2geotable(T);
```

View the `Shape` variable.

```
GT.Shape
```

```
ans =
  162x1 geopointshape array with properties:

               NumPoints: [162x1 double]
                Latitude: [162x1 double]
               Longitude: [162x1 double]
                Geometry: "point"
    CoordinateSystemType: "geographic"
```

```
        GeographicCRS: []
```

Convert Table with WKT Strings to Geospatial Table

Create a sample table. The first table variable (`WKTString`) contains the WKT string representation of geometry for lines in planar coordinates. The second variable (`Name`) contains names.

```
wkt1 = "LINESTRING (210320.1875 913333.8125,210320.140625 913366.6875)";
wkt2 = "LINESTRING (210320.4375 913170.9375,210320.34375 913231.1875)";
name1 = "Line 1";
name2 = "Line 2";
T = table([wkt1;wkt2],[name1;name2],VariableNames={'WKTString','Name'})
```

T=*2×2 table*

WKTString	Name
"LINESTRING (210320.1875 913333.8125,210320.140625 913366.6875)"	"Line 1"
"LINESTRING (210320.4375 913170.9375,210320.34375 913231.1875)"	"Line 2"

Convert the table to a geospatial table. Create the `Shape` variable of the table by using the WKT strings. For this example, specify the projected CRS as World Equidistant Cylindrical, which has the EPSG authority code `4087`.

```
GT = table2geotable(T,"planar","WKTString", ...
    CoordinateReferenceSystem=projcrs(4087))
```

GT=*2×3 table*

Shape	WKTString	Name
maplineshape	"LINESTRING (210320.1875 913333.8125,210320.140625 913366.6875)"	"Line 1"
maplineshape	"LINESTRING (210320.4375 913170.9375,210320.34375 913231.1875)"	"Line 2"

View the `Shape` variable of the geospatial table.

```
GT.Shape
```

ans=*2×1 object*
```
  2x1 maplineshape array with properties:

              NumParts: [2x1 double]
              Geometry: "line"
    CoordinateSystemType: "planar"
          ProjectedCRS: [1x1 projcrs]
```

Input Arguments

T — Input table
`table` object | `timetable` object

Input table, specified as a `table` or `timetable` object.

The `table2geotable` function detects the coordinates when the table variables have these names, ignoring case.

- Latitude coordinates — `Latitude` or `Lat`
- Longitude coordinates — `Longitude`, `Lon`, or `Long`
- *x*-coordinates — `X`
- *y*-coordinates — `Y`

If you specify an input table with a variable called `Shape`, then the `table2geotable` function overwrites it.

Data Types: `table`

coordinateSystemType — Coordinate system type
`"planar"` | `"geographic"`

Coordinate system type, specified as one of these options:

- `"planar"` — Coordinates are in a planar coordinate system.
- `"geographic"` — Coordinates are in a geographic coordinate system.

Data Types: `string`

varnames — Names of table variables
string scalar | two-element string vector

Names of the table variables used to create the `Shape` variable of the geospatial table, specified as a string scalar or two-element string vector.

- Create the `Shape` variable from one table variable that contains WKT string representations of geometry by specifying `varnames` as a string scalar, such as `"WKT"`
- Create the `Shape` variable from two table variables that contain latitude-longitude or *x-y* coordinates by specifying `varnames` as a two-element string vector, such as `["Lats" "Lons"]`.

Data Types: `string`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `table2geotable(T,CoordinateReferenceSystem=geocrs(4326))` specifies the geographic CRS as the World Geodetic System of 1984, which has EPSG code `4326`.

CoordinateReferenceSystem — Coordinate reference system
`projcrs` object | `geocrs` object

Coordinate reference system (CRS) to use when creating the `Shape` variable of the geospatial table, specified as a `projcrs` or `geocrs` object.

Specify a `projcrs` object when `coordinateSystemType` is `"planar"` and specify a `geocrs` object when `coordinateSystemType` is `"geographic"`.

GeometryType — Geometry type
`"point"` (default) | `"line"` | `"polygon"`

Geometry type of the coordinate variables, specified as one of these options:

- `"point"` — The coordinate fields represent points.
- `"line"` — The coordinate fields represent lines.
- `"polygon"` — The coordinate fields represent polygons.

When you specify the geometry type as `"polygon"`, the `table2geotable` function assumes that the coordinates define polygons with a valid topology. A polygon has a valid topology when:

- Region interiors are to the right as you trace boundaries from vertex to vertex.
- The boundaries have no self-intersections.

In general, the outer boundaries of polygons with a valid topology have vertices in a clockwise order and the interior holes have vertices in a counterclockwise order.

If you specify `varnames` as a table variable containing WKT string representations of geometry, then the `table2geotable` function ignores this argument.

Output Arguments

GT — Output geospatial table
geospatial table

Output geospatial table. A geospatial table is a `table` or `timetable` object with a `Shape` variable that contains `geopointshape`, `geolineshape`, `geopolyshape`, `mappointshape`, `maplineshape`, or `mappolyshape` objects.

For an input table of size M-by-N, the size of GT is M-by-$(N + 1)$.

Tips

- When the input table has latitude-longitude fields and *x-y* variables, the `table2geotable` function creates the geospatial table using the latitude-longitude variables. Create the geospatial table using the *x-y* variables by specifying the `coordinateSystemType` and `varnames` arguments.

See Also

Functions
geotable2table | readgeotable | readtable | table2timetable

Topics
"Create Geospatial Tables"

Introduced in R2021b

tagm

Set Tag property of map graphics object

Syntax

```
tagm(hndl,tagstr)
```

Description

tagm(hndl,tagstr) sets the Tag property of each object designated in the vector of handles hndl to the associated row of the character matrix tagstr.

This property is recognized by the namem and handlem functions.

Examples

Normally, a plotted line has a name of 'line':

```
axesm miller
lats = [3 2 1 1 2 3]; longs = [7 8 9 7 8 9];
h=plotm(lats,longs);

untagged = namem(h)
untagged =
line
```

The tagm function can rename it:

```
tagm(h,'testpath');
tagged = namem(h)
tagged =
testpath
```

See Also

clma | clmo | handlem | hidem | namem | showm

Introduced before R2006a

tbase

(To be removed) Read 5-minute global terrain elevations from TerrainBase

Note `tbase` will be removed in a future release. Use `readgeoraster` instead. For more information, see "Compatibility Considerations".

Syntax

```
[Z,refvec] = tbase(scalefactor)
[Z,refvec] = tbase(scalefactor,latlim,lonlim)
```

Description

`[Z,refvec] = tbase(scalefactor)` reads the data for the entire world, reducing the resolution of the data by the specified scale factor. The result is returned as a regular data grid and an associated three-element referencing vector.

`[Z,refvec] = tbase(scalefactor,latlim,lonlim)` reads the data for the part of the world within the latitude and longitude limits. The limits must be two-element vectors in units of degrees.

Background

TerrainBase is a global model of terrain and bathymetry on a regular 5-minute grid (approximately 10 km resolution). It is a compilation of the public domain data from almost 20 different sources, including the DCW-DEM and ETOPO5. The data set was created by the National Geophysical Data Center and World Data Center-A for Solid Earth Geophysics in Boulder, Colorado.

Examples

Read Data for a Region at Full Resolution

Set the scale factor to 1 and set the longitude and latitude limits. This example reads the data for Korea and Japan.

```
scalefactor = 1;
latlim = [30 45];
lonlim = [115 145];
```

Read the data and view the size of the returned variables.

```
[Z,refvec] = tbase(scalefactor,latlim,lonlim);
whos
```

```
Name        Size          Bytes  Class
```

```
Z        180x360        518400  double array
refvec     1x3              24  double array
```

Compatibility Considerations

tbase will be removed
Warns starting in R2021b

Support for reading the TerrainBase Global Terrain Model will be removed. In addition, raster reading functions that return referencing vectors will be removed, including `tbase`. Use a supported data set and return a raster reference object using `readgeoraster` instead.

Reference objects have several advantages over referencing vectors.

- Unlike referencing vectors, reference objects have properties that document the size of the associated raster, its geographic limits, and the direction of its rows and columns. For examples of reference object properties, see `GeographicCellsReference` and `MapPostingsReference`.
- You can manipulate the limits of rasters associated with reference objects using the `geocrop` or `mapcrop` functions.
- You can manipulate the size and resolution of rasters associated with reference objects using the `georesize` or `mapresize` functions.
- Most functions that accept referencing vectors as input also accept reference objects.

For information about finding data sets, see "Find Geospatial Raster Data".

See Also
`readgeoraster` | `georasterinfo`

Introduced before R2006a

textm

Project text annotation on map axes

Syntax

```
textm(lat,lon,string)
textm(lat,lon,z,string)
textm(lat,lon,z,string,PropertyName,PropertyValue,...)
h = textm(...)
```

Description

`textm(lat,lon,string)` projects the text in `string` onto the current map axes at the locations specified by the `lat` and `lon`. The units of `lat` and `lon` must match the `'angleunits'` property of the map axes. If `lat` and `lon` contain multiple elements, `textm` places a text object at each location. In this case `string` may be a cell array of character vectors with the same number of elements as `lat` and `lon`. (For backward compatibility, `string` may also be a 2-D character array such that `size(string,1)` matches `numel(lat)`).

`textm(lat,lon,z,string)` draws the text at the altitude(s) specified in `z`, which must be the same size as `lat` and `lon`. The default altitude is 0.

`textm(lat,lon,z,string,PropertyName,PropertyValue,...)` sets the text object properties. All properties supported by the MATLAB `text` function are supported by `textm`.

`h = textm(...)` returns the handles to the text objects drawn.

Examples

The feature of `textm` that distinguishes it from the standard MATLAB `text` function is that the text object is projected appropriately. Type the following:

```
axesm sinusoid
framem('FEdgeColor','red')
textm(60,90,'hello')
```
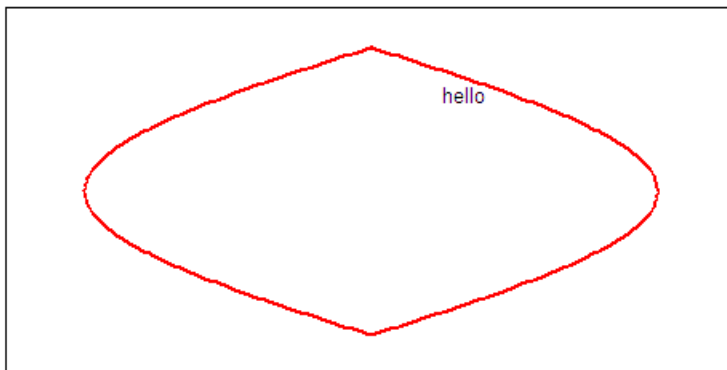
```
figure; axesm miller
framem('FEdgeColor','red')
textm(60,90,'hello')
```
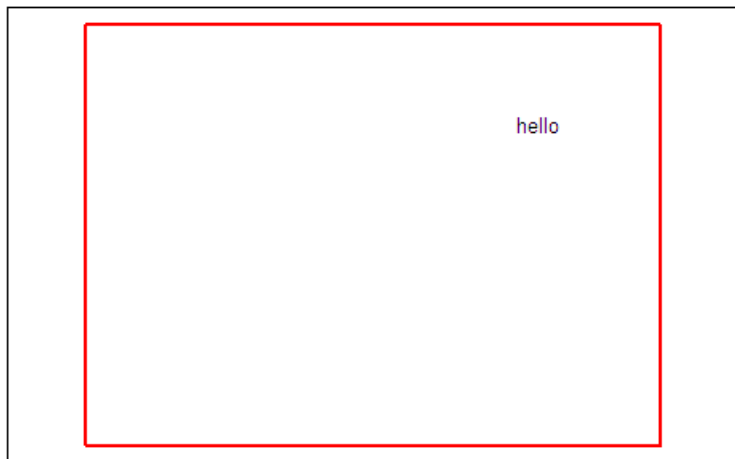

The text `'hello'` is placed at the same geographic point, but it appears to have moved relative to the axes because of the different projections. If you change the projection using the `setm` function, the text moves as necessary. Use `text` to fix text objects in the axes independent of projection.

Tips

You may be working with scalar `lat` and `lon` data or vector `lat` and `lon` data. If you are in scalar mode and you enter a cell array of character vectors, you will get a text object containing multiple lines. Also note that vertical slash characters, rather than producing multiple lines of text, yield a single line of text containing vertical slashes. On the other hand, if `lat` and `lon` are nonscalar, then the size of the cell array input must match their size exactly.

See Also

`axesm` | `text`

Introduced before R2006a

tgrline

Read TIGER/Line data

Compatibility

Note `tgrline` will be removed in a future version. More recent TIGER/Line data sets are available in shapefile format and can be imported using `shaperead`.

Syntax

```
[CL,PR,SR,RR,H,AL,PL] = tgrline(filename)
[CL,PR,SR,RR,H,AL,PL] = tgrline(filename,year)
[CL,PR,SR,RR,H,AL,PL] = tgrline(filename,year,countyname)
```

Description

`[CL,PR,SR,RR,H,AL,PL] = tgrline(filename)` reads a set of 1994 TIGER/Line files which share the same file name, but different extensions. The results are returned in a set of Mapping Toolbox display structures tagged with feature names and containing:

- county boundaries (CL)
- primary roads (PR)
- secondary roads (SR)
- railroads (RR)
- hydrography (H)
- area landmarks (AL)
- point landmarks (PL)

`[CL,PR,SR,RR,H,AL,PL] = tgrline(filename,year)` reads the TIGER line files in the format from that year. The layout of TIGER/Line files is updated periodically and file name extensions may change from year to year. Valid years are 1990, 1992, 1994, 1995, 1999, 2000, 2002, 2003, and 2004.

`[CL,PR,SR,RR,H,AL,PL] = tgrline(filename,year,countyname)` uses the character vector `countyname` to tag the county data.

Background

The United States Census Bureau distributes TIGER/Line data over the Internet and via CD-ROM or DVD.

TIGER/Line files contain vector map data used to support mapping for the U.S. Census Bureau. TIGER is an acronym for Topographically Integrated Geographic Encoding and Referencing. These files contain data for political boundaries, including states, counties, Indian reservations, and census tracts, as well as roads, railroads, hydrography, and landmarks. In addition to the geographically referenced information, the files also contain data to determine the address of an object. The data

covers the United States of America and its territories or administrative units: Puerto Rico, the Virgin Islands of the United States, American Samoa, Guam, the Commonwealth of the Northern Marianna Islands, the Republic of Palau, the other Pacific entities that were part of the Trust Territory of the Pacific Islands (the Republic of the Marshall Islands and the Federated States of Micronesia), and the Midway Islands. The most common application of this data is to commercial CD-ROM road atlases.

TIGER/Line is a registered trademark of the United States Census Bureau.

Examples

Read from the data for Washington, D.C.:

```
[CL,PR,SR,RR,H,AL,PL] = tgrline('TGR11001',1994,'Wash,DC');
```

Tips

This function reads only a subset of the data in the TIGER/Line files. For example, the function does not return local roads, zip codes, or census tract numbers.

Data are returned as Mapping Toolbox display structures, which you can then update to geographic data structures. For information about display structure format, see "Version 1 Display Structures" on page 1-286 in the reference page for `displaym`. The `updategeostruct` function performs such conversions.

See Also
shaperead | updategeostruct

Introduced before R2006a

tightmap

Remove white space around map

Syntax

```
tightmap
```

Description

`tightmap` sets the axis limits to be tight around the map in the current axes. This eliminates or reduces the white border between the map frame and the axes box. Use `axis auto` to undo `tightmap`.

Examples

Display Map of Africa With and Without Surrounding White Space

Display a map of Africa. Notice the white space between the map frame and the edge of the axes box.

```
axesm('miller','maplatlim',[-40 40],'maplonlim',[-20 60])
framem;
gridm;
mlabel;
plabel
load coastlines
plotm(coastlat,coastlon)
```
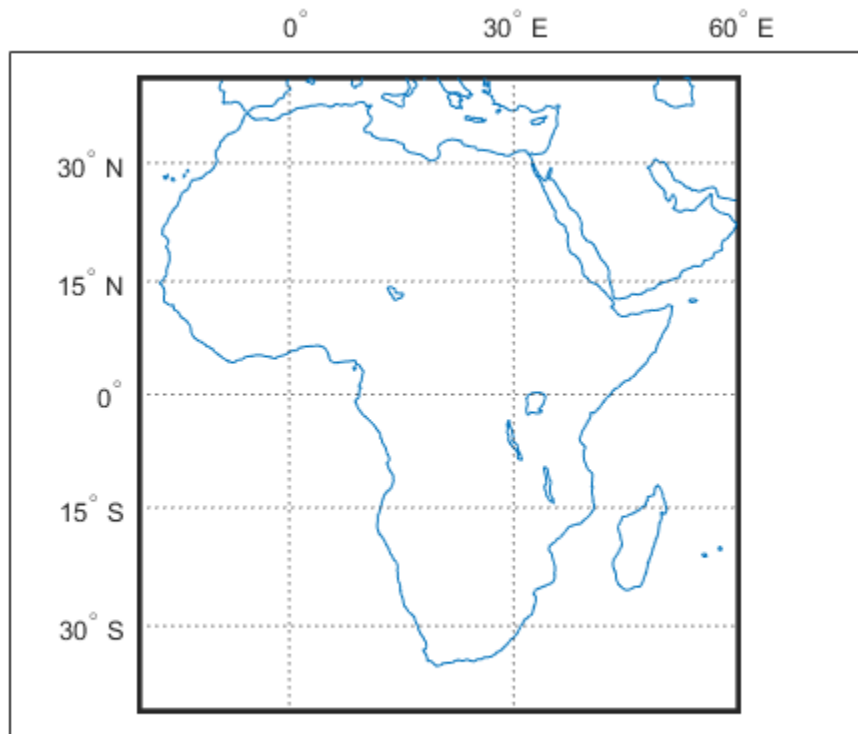
Now remove white space around map axes.

```
tightmap
```
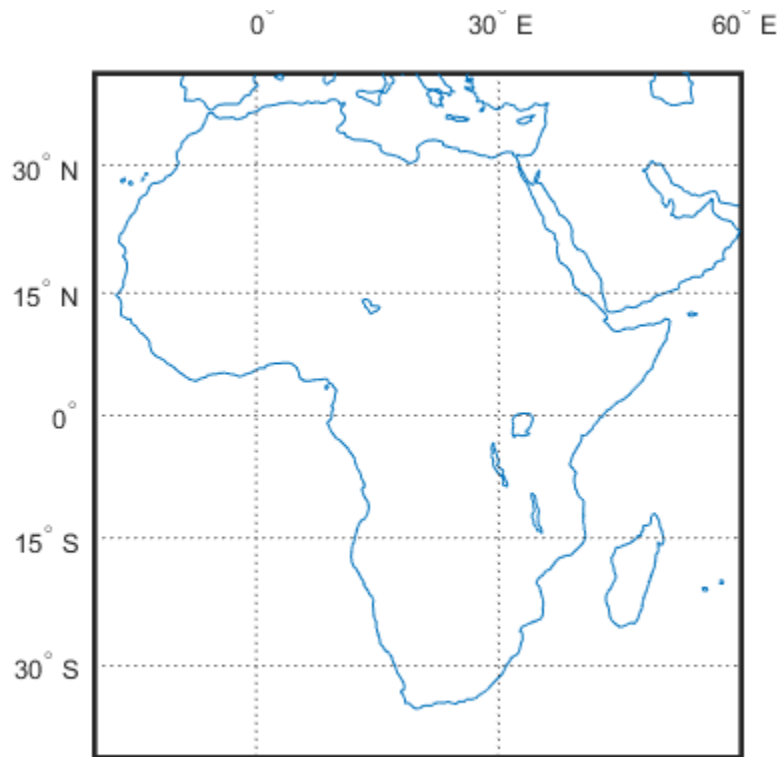
Tips

The axis limits are fixed. If a change in the projection parameters changes the size or position of the map display within the projected coordinate system, execute `tightmap` again. Also note that `tightmap` needs to be re-applied following any call to `setm` that causes projected map objects to be re-projected.

The `tightmap` function performs no action on a `'globe'` map axes.

See Also

panzoom | zoom | paperscale | axesscale | previewmap

Introduced before R2006a

timezone

Time zone based on longitude

Syntax

```
[zd,zltr,zone] = timezone(long)
[zd,zltr,zone] = timezone(long,units)
```

Description

`[zd,zltr,zone] = timezone(long)` returns an integer zone description, `zd`, an alphabetical zone indicator, `zltr`, and a character vector, `zone`, with the complete zone description and alphabetical zone indicator corresponding to the input longitude `long`.

`[zd,zltr,zone] = timezone(long,units)` specifies the angular units with a standard angle units. The default value is `'degrees'`. Valid units are:

- `'degrees'` — decimal degrees
- `'radians'`

Examples

Given that is it locally 1330 (1:30 p.m.) at a longitude of 75ºW, determine Coordinated Universal Time (UTC):

```
[zd,zltr,zone] = timezone(-75,'degrees')

zd =

     5


zltr =

    'R'


zone =

    '+5 R'
```

UTC is 1330 plus five hours, or 1830 (6:30 p.m.).

Background

Time is determined by the position of the sun relative to the prime meridian, the zero longitude line running through Greenwich, England. When this meridian lies directly below the sun, it is noon Coordinated Universal Time (UTC). For local times elsewhere, the Earth is divided into 15º longitude bands, each centered on a central meridian. When a central meridian lies directly below the sun, Local Mean Time (LMT) in that zone is noon. The zone description is an integer that when added to

LMT gives UTC. For notational convenience, each zone is also given an alphabetical indicator. The indicator at Greenwich is *Z*, so UTC is sometimes called *ZULU time*.
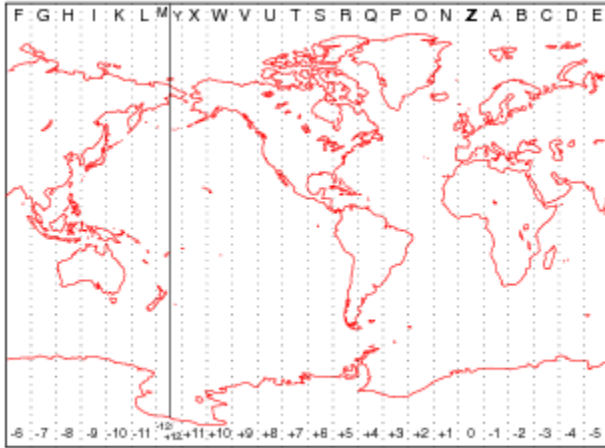
Note that there are actually 25 time zones, because the zone centered on the International Date Line (180º E/W) is split into two: "+12 Y" and "-12 M."

Limitations

National and local governments set their own time zone boundaries for political or geographic convenience. The `timezone` function does not account for statutory deviations from the meridian-based system.

See Also
`timezones` | `tzoffset`

Introduced before R2006a

tissot

Project Tissot indicatrices on map axes

Syntax

```
h = tissot
h = tissot(spec)
h = tissot(spec,linetype)
h = tissot(linestyle)
h = tissot(spec,PropertyName,PropertyValue,...)
h = tissot(linetype,PropertyName,PropertyValue,...)
```

Description

`h = tissot` plots the default Tissot diagram, as described above, on the current map axes and returns handles for the displayed indicatrices.

`h = tissot(spec)` allows you to specify plotting parameters of the displayed Tissot diagram as described above.

`h = tissot(spec,linetype)` and `h = tissot(linestyle)` specifies the line style and color on page 1-1385.

`h = tissot(spec,PropertyName,PropertyValue,...)` and `h = tissot(linetype,PropertyName,PropertyValue,...)` allow the specification of any property and value recognized by the `line` function.

Background

Tissot indicatrices are plotting symbols that are useful for understanding the various distortions of a given map projection. The indicatrices are circles of identical true radius on the Earth's surface. When plotted on a map projection, they indicate whether the projection has certain features. If the plotted indicatrices all enclose the same area, the projection is equal area (for example, a Sinusoidal projection would have this feature). If they all remain circular, then conformality is indicated (a Mercator projection has this property). Distortions in meridional or parallel distance are exhibited by flattened or stretched indicatrices. Many projections will show very even, circular indicatrices in some regions, often near the center, and wildly distorted indicatrices in others, such as near the edges. The Tissot diagram is therefore very useful in analyzing the appropriateness of a projection to a given purpose or region.

The general layout of the Tissot diagram is defined by the specification vector `spec`.

```
spec = [Radius]
spec = [Latint,Longint]
spec = [Latint,Longint,Radius]
spec = [Latint,Longint,Radius,Points]
```

`Radius` is the small circle radius of each indicatrix circle. If entered, it should be in the same units as the map axes `Geoid`. The default radius is 1/10th the radius of the sphere.

Latint is the latitude interval between indicatrix circle centers. If entered it should be in the map axes AngleUnits. The default value is one circle every 30º of latitude (that is, 0º, +/-30º, etc.).

Longint is the longitude interval between indicatrix circle centers. If entered it should be in the map axes AngleUnits. The default value is one circle every 30º of latitude (that is, 0º, +/-30º, etc.).

Points is the number of plotting points per circle. The default is 100 points.

Examples

```
axesm sinusoid; framem
tissot
```
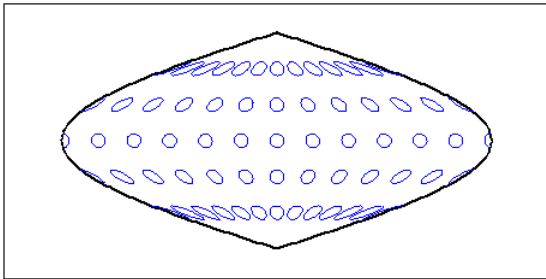

The Sinusoidal projection is equal area.

```
setm(gca,'MapProjection','Mercator')
```
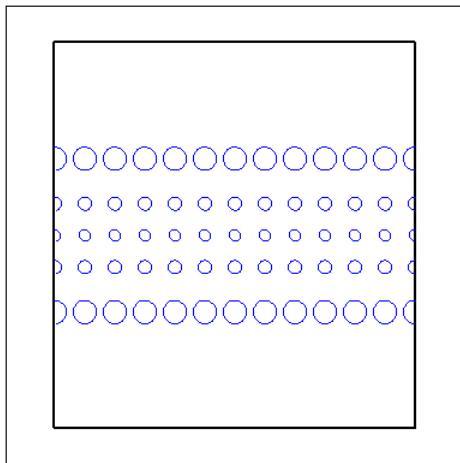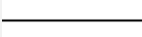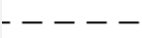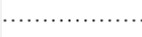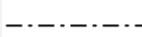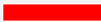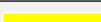

The Mercator projection is conformal.

More About

Line Style and Color

Specify the line style and color as a character vector or string containing symbols. The symbols can appear in any order. You do not need to specify a color. For example, specify a red dashed line using 'r--'.

Line Style	Description	Resulting Line
'-'	Solid line	——————
'--'	Dashed line	- — — — —
':'	Dotted line	··················
'-.'	Dash-dotted line	—·—·—·—·

Color Name	Short Name	Appearance
'red'	'r'	████
'green'	'g'	████
'blue'	'b'	████
'cyan'	'c'	████
'magenta'	'm'	████
'yellow'	'y'	████
'black'	'k'	████
'white'	'w'	████

See Also

mdistort | distortcalc

Introduced before R2006a

toDegrees

Convert angles to degrees

Syntax

```
[D1,...,Dn] = toDegrees(fromUnits,A1,...,An)
```

Description

`[D1,...,Dn] = toDegrees(fromUnits,A1,...,An)` converts the angles specified by `A1,...,An` to degrees from the angle units specified by `fromUnits`. The value of `fromUnits` can be either `'degrees'` or `'radians'` and may be abbreviated. The inputs `A1,...,An` and their corresponding outputs are numeric arrays of various sizes, with `size(Dn)` matching `size(An)`.

See Also

`fromDegrees`, `fromRadians`, `rad2deg`, `toRadians`

Introduced in R2007b

toRadians

Convert angles to radians

Syntax

[R1,...,Rn] = toRadians(*fromUnits*,A1,...,An)

Description

[R1,...,Rn] = toRadians(*fromUnits*,A1,...,An) converts the angles specified by A1,...,An to radians from the angle units specified by `fromUnits`. The value of `fromUnits` can be either `'degrees'` or `'radians'` and may be abbreviated. The inputs A1,...,An and their corresponding outputs are numeric arrays of various sizes, with `size(Rn)` matching `size(An)`.

See Also

deg2rad, fromDegrees, fromRadians, toDegrees

Introduced in R2007b

track

Track segments to connect navigational waypoints

Syntax

```
[lattrk,lontrk] = track(waypts)
[lattrk,lontrk] = track(waypts,units)
[lattrk,lontrk] = track(lat,lon)
[lattrk,lontrk] = track(lat,lon,ellipsoid)
[lattrk,lontrk] = track(lat,lon,ellipsoid,units,npts)
[lattrk,lontrk] = track(method,lat,...)
trkpts = track(lat,lon...)
```

Description

`[lattrk,lontrk] = track(waypts)` returns points in `lattrk` and `lontrk` along a track between the waypoints provided in navigational track format in the two-column matrix `waypts`. The outputs are column vectors in which successive segments are delineated with `NaN`s.

`[lattrk,lontrk] = track(waypts,units)` specifies the units of the inputs and outputs, where *units* is any valid angle unit string. The default is `'degrees'`.

`[lattrk,lontrk] = track(lat,lon)` allows the user to input the waypoints in two vectors, `lat` and `lon`.

`[lattrk,lontrk] = track(lat,lon,ellipsoid)` specifies the shape of the Earth using `ellipsoid`, which can be a `referenceSphere`, `referenceEllipsoid`, or `oblateSpheroid` object, or a vector of the form `[semimajor_axis eccentricity]`. The default ellipsoid is a unit sphere

`[lattrk,lontrk] = track(lat,lon,ellipsoid,units,npts)` establishes how many intermediate points are to be calculated for every track segment. By default, `npts` is 30.

`[lattrk,lontrk] = track(method,lat,...)` establishes the logic to be used to determine the intermediate points along the track between waypoints. Because this is a navigationally motivated function, the default method is `'rh'`, which results in rhumb line logic. Great circle logic can be specified with `'gc'`.

`trkpts = track(lat,lon...)` compresses the output into one two-column matrix, `trkpts`, in which the first column represents latitudes and the second column, longitudes.

Examples

Calculate Track and Display It on Map

The track function is useful for generating data in order to display tracks. Lieutenant Sextant is the navigator of the USS Neversail. He is charged with plotting a track to take Neversail from the Straits of Gibraltar to Port Said, Egypt, the northern end of the Suez Canal. He has picked appropriate waypoints and now would like to display the track for his captain's approval.

First, display a chart of the Mediterranean Sea.

```
load coastlines
axesm('mercator','MapLatLimit',[28 47],'MapLonLimit',[-10 37],...
    'Grid','on','Frame','on','MeridianLabel','on','ParallelLabel','on')
geoshow(coastlat,coastlon,'DisplayType','line','color','b')
```
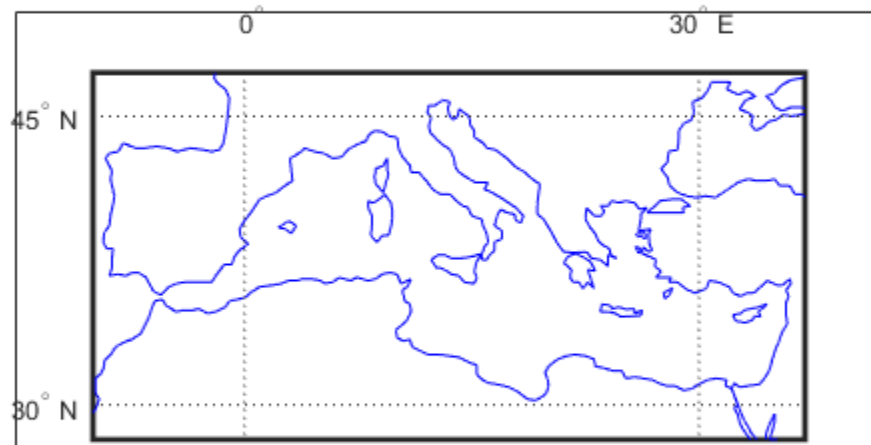

These are the waypoints Lt. Sextant has selected.

```
waypoints = [36,-5; 36,-2; 38,5; 38,11; 35,13; 33,30; 31.5,32];
```

Now display the track. With a display this clear, the captain gladly approves the plan.

```
[lttrk,lntrk] = track('rh',waypoints,'degrees');
geoshow(lttrk,lntrk,'DisplayType','line','color','r')
```
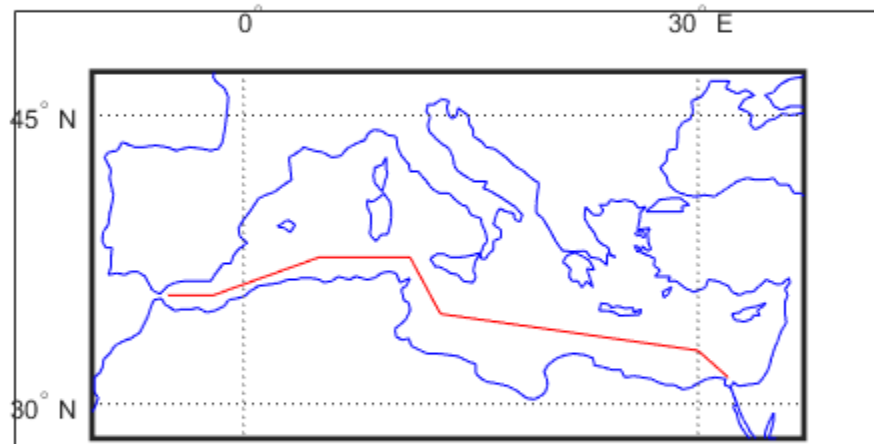
See Also

dreckon | gcwaypts | legs | navfix

Introduced before R2006a

track1

Geographic tracks from starting point, azimuth, and range

Syntax

```
[lat,lon] = track1(lat0,lon0,az)
[lat,lon] = track1(lat0,lon0,az,arclen)
[lat,lon] = track1(lat0,lon0,az,arclen,ellipsoid)
[lat,lon] = track1(lat0,lon0,az,angleunits)
[lat,lon] = track1(lat0,lon0,az,arclen,angleunits)
[lat,lon] = track1(lat0,lon0,az,arclen,ellipsoid,angleunits)
[lat,lon] = track1(lat0,lon0,az,arclen,ellipsoid,angleunits,npts)
[lat,lon] = track1(trackstr,...)
mat = track1(...)
```

Description

`[lat,lon] = track1(lat0,lon0,az)` computes complete great circle tracks on a sphere starting at the point `lat0,lon0` and proceeding along the input azimuth, `az`. The inputs can be scalar or column vectors.

`[lat,lon] = track1(lat0,lon0,az,arclen)` uses the input `arclen` to specify the arc length of the great circle track. `arclen` is specified in units of degrees of arc. If `arclen` is a column vector, then the track is computed from the starting point, with positive distance measured easterly. If `arclen` is a two column matrix, then the track is computed starting at the range in the first column and ending at the range in the second column. If `arclen = []`, then the complete track is computed.

`[lat,lon] = track1(lat0,lon0,az,arclen,ellipsoid)` computes the track along a geodesic arc on the ellipsoid defined by the input `ellipsoid`, which can be a `referenceSphere`, `referenceEllipsoid`, or `oblateSpheroid` object, or a vector of the form `[semimajor_axis eccentricity]`. `arclen` must be expressed in length units that match the units of the semimajor axis — unless `ellipsoid` is `[]` or the semimajor axis length is zero. In these special cases, `arclen` is assumed to be in degrees of arc and the tracks are computed on a sphere, as in the preceding syntax.

`[lat,lon] = track1(lat0,lon0,az,angleunits)`,
`[lat,lon] = track1(lat0,lon0,az,arclen,angleunits)`, and
`[lat,lon] = track1(lat0,lon0,az,arclen,ellipsoid,angleunits)` where `angleunits` defines the units of the input and output angles as `'degrees'` or `'radians'`.

`[lat,lon] = track1(lat0,lon0,az,arclen,ellipsoid,angleunits,npts)` uses the scalar input `npts` to specify the number of points per track. The default value of `npts` is 100.

`[lat,lon] = track1(trackstr,...)` where `trackstr` is a string scalar or a character vector that defines either a great circle (`'gc'`) or rhumb line track (`'rh'`). If `trackstr` is `'gc'`, then either great circle (given a sphere) or geodesic (given an ellipsoid) tracks are computed. If `trackstr` is `'rh'`, then the rhumb line tracks are computed.

`mat = track1(...)` returns a single output argument `mat` such that `mat = [lat lon]`. This is useful if only a single track is computed.

Multiple tracks can be defined from a single starting point by providing scalar `lat0` and `lon0` and column vectors for `az` and `arclen`.

Examples

```
% Set up the axes.
axesm('mercator','MapLatLimit',[-60 60],'MapLonLimit',[-60 60])
gridm on; plabel on; mlabel on;

% Plot the great circle track in green.
[lattrkgc,lontrkgc] = track1(0,0,45,[-55 55]);
plotm(lattrkgc,lontrkgc,'g')

% Plot the rhumb line track in red.
[lattrkrh,lontrkrh] = track1('rh',0,0,45,[-55 55]);
plotm(lattrkrh,lontrkrh,'r')
```
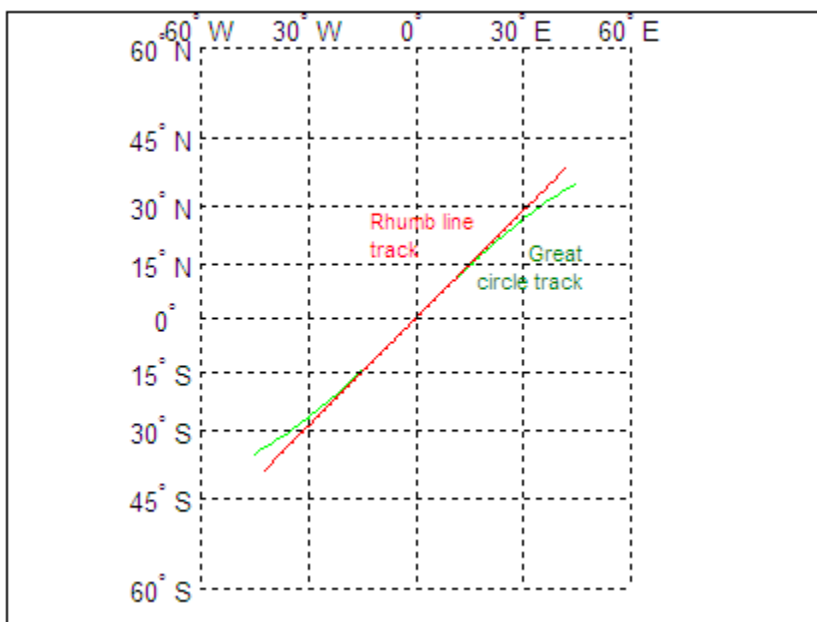

More About

Track Lines

A path along the surface of the Earth connecting two points is a *track*. Two types of track lines are of interest geographically, *great circles* and *rhumb lines*. Great circles represent the shortest possible path between two points. Rhumb lines are paths with constant angular headings. They are not, in general, the shortest path between two points.

Full great circles bisect the Earth; the ends of the track meet to form a complete circle. Rhumb lines with true east or west azimuths are parallels; the ends also meet to form a complete circle. All other rhumb lines terminate at the poles; their ends do not meet.

See Also
azimuth | distance | reckon | scircle1 | scircle2 | track | track2 | trackg

Introduced before R2006a

track2

Geographic tracks from starting and ending points

Syntax

```
[lat,lon] = track2(lat1,lon1,lat2,lon2)
[lat,lon] = track2(lat1,lon1,lat2,lon2,ellipsoid)
[lat,lon] = track2(lat1,lon1,lat2,lon2,units)
[lat,lon] = track2(lat1,lon1,lat2,lon2,ellipsoid,units)
[lat,lon] = track2(lat1,lon1,lat2,lon2,ellipsoid,units,npts)
[lat,lon] = track2(track,...)
mat = track2(...)
```

Description

`[lat,lon] = track2(lat1,lon1,lat2,lon2)` computes great circle tracks on a sphere starting at the point `lat1,lon1` and ending at `lat2,lon2`. The inputs can be scalar or column vectors.

`[lat,lon] = track2(lat1,lon1,lat2,lon2,ellipsoid)` computes the great circle track on the ellipsoid defined by the input `ellipsoid`. `ellipsoid` is a `referenceSphere`, `referenceEllipsoid`, or `oblateSpheroid` object, or a vector of the form [`semimajor_axis eccentricity`]. If `ellipsoid = []`, a sphere is assumed.

`[lat,lon] = track2(lat1,lon1,lat2,lon2,units)` and
`[lat,lon] = track2(lat1,lon1,lat2,lon2,ellipsoid,units)` are both valid calling forms, which use the input units to define the angle units of the inputs and outputs. If you omit `units`, `'degrees'` is assumed.

`[lat,lon] = track2(lat1,lon1,lat2,lon2,ellipsoid,units,npts)` uses the scalar input `npts` to determine the number of points per track computed. The default value of `npts` is 100.

`[lat,lon] = track2(track,...)` uses the track to define either a great circle or a rhumb line track. If *track* = `'gc'`, then great circle tracks are computed. If `track` = `'rh'`, then rhumb line tracks are computed. If you omit track, `'gc'` is assumed.

`mat = track2(...)` returns a single output argument where `mat = [lat lon]`. This is useful if a single track is computed. Multiple tracks can be defined from a single starting point by providing scalar inputs for `lat1,lon1` and column vectors for `lat2,lon2`.

Examples

```
% Set up the axes.
axesm('mercator','MapLatLimit',[30 50],'MapLonLimit',[-40 40])

% Calculate the great circle track.
[lattrkgc,lontrkgc] = track2(40,-35,40,35);

% Calculate the rhumb line track.
[lattrkrh,lontrkrh] = track2('rh',40,-35,40,35);
```

```
% Plot both tracks.
plotm(lattrkgc,lontrkgc,'g')
plotm(lattrkrh,lontrkrh,'r')
```
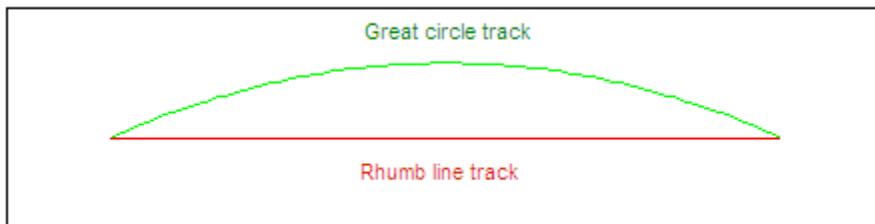

More About

Track Lines

A path along the surface of the Earth connecting two points is a *track*. Two types of track lines are of interest geographically, *great circles* and *rhumb lines*. Great circles represent the shortest possible path between two points. Rhumb lines are paths with constant angular headings. They are not, in general, the shortest path between two points.

See Also

azimuth | distance | reckon | scircle1 | scircle2 | track | track1 | trackg

Introduced before R2006a

trackg

Great circle or rhumb line defined via mouse input

Syntax

```
h = trackg(ntrax)
h = trackg(ntrax,npts)
h = trackg(ntrax,LineSpec)
h = trackg(ntrax,PropertyName,PropertyValue,...)
[lat,lon] = trackg(ntrax,npts,...)
h = trackg(track,ntrax,...)
```

Description

`h = trackg(ntrax)` brings forward the current map axes and waits for the user to make (2 x `ntrax`) mouse clicks. The output `h` is a vector of handles for the `ntrax` track segments, which are then displayed.

`h = trackg(ntrax,npts)` specifies the number of plotting points to be used for each track segment. `npts` is 100 by default.

`h = trackg(ntrax,LineSpec)` specifies the line style, marker, and color on page 1-1397.

`h = trackg(ntrax,PropertyName,PropertyValue,...)` allows property name/property value pairs to be set, where *PropertyName* and *PropertyValue* are recognized by the `line` function.

`[lat,lon] = trackg(ntrax,npts,...)` returns the coordinates of the plotted points rather than the handles of the track segments. Successive segments are stored in separate columns of `lat` and `lon`.
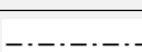
`h = trackg(track,ntrax,...)` specifies the logic with which tracks are calculated. If *track* is `'gc'` (the default), a great circle path is used. If *track* is `'rh'`, rhumb line logic is used.

This function is used to define great circles or rhumb lines for display using mouse clicks. For each track, two clicks are required, one for each endpoint of the desired track segment. You can modify the track after creation by **Shift**+clicking it. The track is then in edit mode, during which you can change the length and position by dragging control points, or by entering values into a control panel. **Shift**+clicking again exits edit mode.
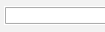
More About

Line Style, Marker, and Color

Specify the line style, marker, and color as a character vector or string containing symbols. The symbols can appear in any order. You do not need to specify all three characteristics (line style, marker, and color). For example, specify a red dashed line using `'--r'`.

Line Style	Description	Resulting Line
'-'	Solid line	_____
'--'	Dashed line	- — — — —
':'	Dotted line	....
'-.'	Dash-dotted line	—.—.—.—.—..

Marker	Description	Resulting Marker
'o'	Circle	○
'+'	Plus sign	+
'*'	Asterisk	✳
'.'	Point	•
'x'	Cross	×
'_'	Horizontal line	—
'|'	Vertical line	|
's'	Square	□
'd'	Diamond	◇
'^'	Upward-pointing triangle	△
'v'	Downward-pointing triangle	▽
'>'	Right-pointing triangle	▷
'<'	Left-pointing triangle	◁
'p'	Pentagram	☆
'h'	Hexagram	✡

Color Name	Short Name	RGB Triplet	Appearance
'red'	'r'	[1 0 0]	▬
'green'	'g'	[0 1 0]	▬
'blue'	'b'	[0 0 1]	▬
'cyan'	'c'	[0 1 1]	▬
'magenta'	'm'	[1 0 1]	▬

Color Name	Short Name	RGB Triplet	Appearance
`'yellow'`	`'y'`	[1 1 0]	
`'black'`	`'k'`	[0 0 0]	
`'white'`	`'w'`	[1 1 1]	

See Also

track1 | track2 | scircleg

Introduced before R2006a

trimcart

Trim graphic objects to map frame

Syntax

```
trimcart(h)
```

Description

`trimcart(h)` clips the graphic objects to the map frame. `h` can be a handle or a vector of handles to graphics objects. `h` can also be any object name recognized by `handlem`. `trimcart` clips lines, surfaces, and text objects.

Examples

Trim Graphic Objects to Map Frame

Trim graphic objects to map frame.

```
figure;
axesm('miller')
framem
[x, y] = humps(0:.05:1);
h = plot(x, y/25, 'r+-');
load coastlines
geoshow(coastlat,coastlon)
trimcart(h)
```
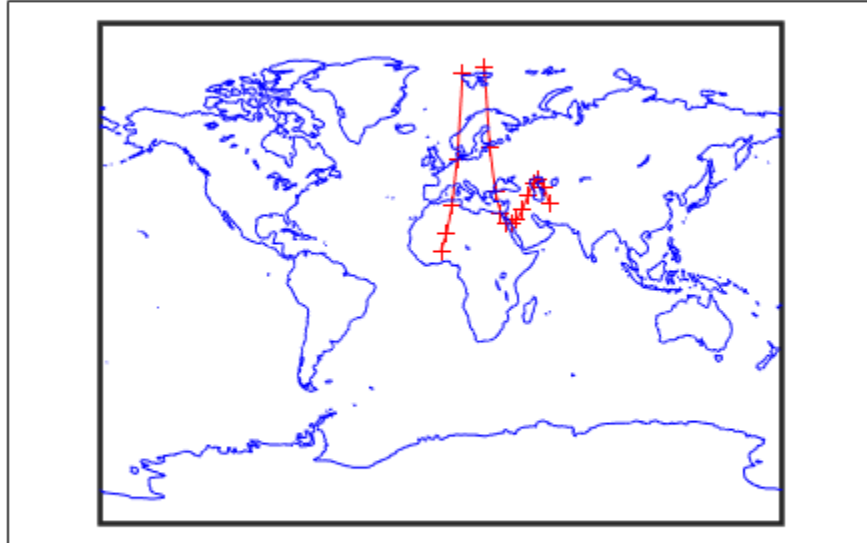
Limitations

`trimcart` does not trim patch objects.

See Also

`handlem` | `makemapped`

Introduced before R2006a

trimdata

Trim map data exceeding projection limits

Syntax

```
[ymat,xmat,trimpts] = trimdata(ymat,ylim,xmat,xlim,object)
```

Description

`[ymat,xmat,trimpts] = trimdata(ymat,ylim,xmat,xlim,object)` identifies points in map data that exceed projection limits. The projection limits are defined by the lower and upper inputs. The particular object to be trimmed is identified by the `object` input.

Allowable objects are

- `'surface'` for trimming graticules
- `'light'` for trimming lights,
- `'line'` for trimming lines
- `'patch'` for trimming patches
- `'text'` for trimming text object location points
- `'none'` to skip all trimming operations

See Also
`clipdata` | `undotrim` | `undoclip`

Introduced before R2006a

undoclip

Remove object clips introduced by `clipdata`

Syntax

```
[lat,long] = undoclip(lat,long,clippts,'object')
```

Description

`[lat,long] = undoclip(lat,long,clippts,'object')` removes the object clips introduced by `clipdata`. This function is necessary to properly invert projected data from the Cartesian space to the original latitude and longitude data points.

The input variable, `clippts`, must be constructed by the function `clipdata`.

Allowable objects are

- `'surface'` for trimming graticules
- `'light'` for trimming lights
- `'line'` for trimming lines
- `'patch'` for trimming patches
- `'text'` for trimming text object location points
- `'none'` to skip all trimming operations

See Also
clipdata | trimdata | undotrim

Introduced before R2006a

undotrim

Remove object trims introduced by `trimdata`

Syntax

```
[ymat,xmat] = undotrim(ymat,xmat,trimpts,object)
```

Description

`[ymat,xmat] = undotrim(ymat,xmat,trimpts,object)` removes the object trims introduced by `trimdata`. This function is necessary to properly invert projected data from the Cartesian space to the original latitude and longitude data points.

The input variable, `trimpts`, must be constructed by the function `trimdata`.

Allowable objects are

- `'surface'` for trimming graticules
- `'light'` for trimming lights
- `'line'` for trimming lines
- `'patch'` for trimming patches
- `'text'` for trimming text object location points
- `'none'` to skip all trimming operations

See Also
`clipdata` | `trimdata` | `undoclip`

Introduced before R2006a

unitsratio

Unit conversion factors

Syntax

```
ratio = unitsratio(to,from)
```

Description

`ratio = unitsratio(to,from)` returns the number of `to` units per one `from` unit. You can specify any of the measurement units supported by the `validateLengthUnit` function, such as kilometers, or the angle units `'radians'` and `'degrees'`.

For example, using measurement units, `unitsratio('cm','m')` returns the value `100` because there are 100 centimeters per meter.

The variables `to` and `from` are case insensitive and can be either singular or plural. For example, `unitsratio` accepts any of the values listed in the following table for angle units.

Unit Name	Acceptable Values
radian	`'rad'`, `'radian(s)'`
degree	`'deg'`, `'degree(s)'`

The `unitsratio` function makes it easy to convert values from one system of units to another. For example, if you want to convert the value `100` kilometers (`from` units) to meters (`to` units), you can use the following code:

```
y = unitsratio('meters','kilometers') * 100

y =

     100000
```

Examples

Compute Conversion Ratio Between Units of Length

Create a conversion factor for international feet to meters.

```
mPerFoot = unitsratio('meter','feet')

mPerFoot = 0.3048
```

Convert 100 international feet into meters.

```
m = mPerFoot * 100

m = 30.4800
```

Create the inverse conversion factor and multiply it by the `mPerFoot` conversion factor.

```
unitsratio('feet','meter') * mPerFoot

ans = 1.0000
```

See Also

validateLengthUnit

unitstr

Check spatiotemporal unit names and abbreviations

Compatibility

Note The unitstr function is obsolete and will be removed in a future release. The syntax str = unitstr(str,'times') has already been removed.

Syntax

```
unitstr
str = unitstr(str0,'angles')
str = unitstr(str0,'distances')
```

Description

unitstr, with no arguments, displays a list of names and abbreviations, recognized by certain Mapping Toolbox functions, for units of angle and length/distance.

str = unitstr(*str0*,'angles') checks for valid angle unit names or abbreviations. If a valid name or abbreviation is found, it is converted to a standardized, preset name. 'angles' can be abbreviated.

str = unitstr(*str0*,'distances') checks for valid length unit names or abbreviations. If a valid name or abbreviation is found, it is converted to a standardized, preset name. 'distances' can be abbreviated. Note that 'miles' and 'mi' are converted to 'statutemiles'; there is no way to specify international miles in the unitstr function.

Examples

This function recognizes and standardizes certain abbreviations:

```
str = unitstr('sm','distances')

str =
statutemiles
```

And any unique truncation:

```
str = unitstr('ra','angles')

str =
radians
```

See Also
unitsratio

Introduced before R2006a

unwrapMultipart

Unwrap vector of angles with NaN-delimited parts

Syntax

```
unwrapped = unwrapMultipart(p)
unwrapped = unwrapMultipart(p,angleUnit)
```

Description

`unwrapped = unwrapMultipart(p)` unwraps a row or column vector of azimuths, longitudes, or phase angles. Input and output units are both radians. If `p` is separated into multiple parts delimited by values of NaN, each part is unwrapped independently. If `p` has only one part, the result is equivalent to `unwrap(p)`. The output is the same size as the input and has NaNs in the same locations.

`unwrapped = unwrapMultipart(p,angleUnit)` unwraps a row or column vector of azimuths, longitudes, or phase angles, where `angleUnit` specifies the unit used for the input and output angles: `'degrees'` or `'radians'`.

Examples

Example 1

Compare the behavior `unwrapMultipart` to that of `unwrap`. The output of `unwrapMultipart` starts over again at 6.11 following the NaN, unlike the output of `unwrap`. The output of `unwrapMultipart` is equivalent to a concatenation (with NaN-separator) of separate calls to `unwrap`:

```
p1 = [0.17        5.67        4.89            4.10];
p2 = [6.11        1.05        2.27];
unwrap([p1 NaN p2])

ans =
    0.1700   -0.6132   -1.3932   -2.1832      NaN   -0.1732   1.0500   2.2700

unwrapMultipart([p1 NaN p2])

ans =
    0.1700   -0.6132   -1.3932   -2.1832      NaN   6.1100   7.3332   8.5532

[unwrap(p1) NaN unwrap(p2)]

ans =
    0.1700   -0.6132   -1.3932   -2.1832      NaN   6.1100   7.3332   8.5532
```

Example 2

Wrap two revolutions of a sphere to π with `wrapToPi`, and then unwrap it with `unwrapMultipart`:

```
lon = wrapToPi(deg2rad(0:10:720));
unwrappedlon = unwrapMultipart(lon);
figure; hold on
plot(lon,'--')
plot(unwrappedlon)
xlabel 'Point Number'
ylabel 'Longitude in radians'
```
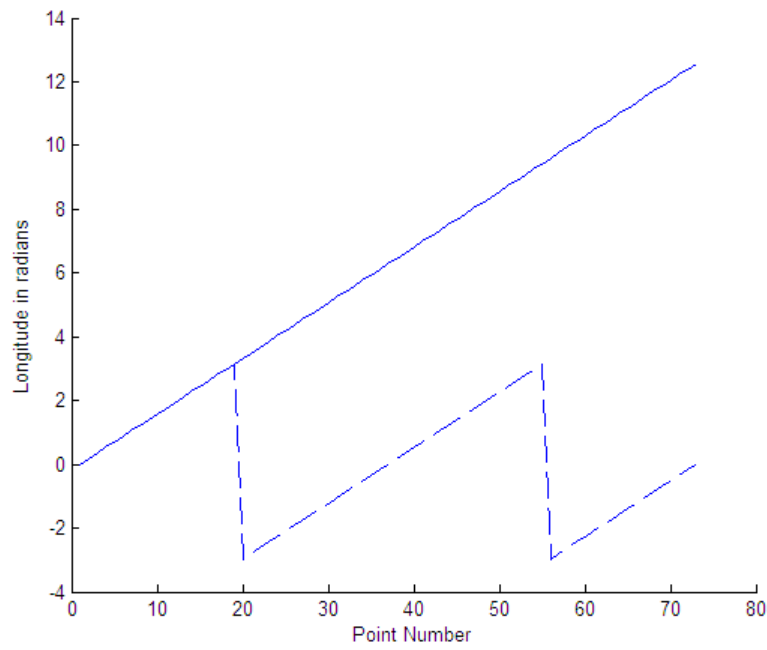
See Also

unwrap | wrapTo180 | wrapTo360 | wrapToPi | wrapTo2Pi

Introduced in R2007b

updategeostruct

Convert line or patch display structure to geostruct

Syntax

```
geostruct = updategeostruct(displaystruct)
geostruct = updategeostruct(displaystruct, str)
[geostruct,symbolspec] = updategeostruct(displaystruct, ...)
[geostruct,symbolspec] = updategeostruct(displaystruct, ..., cmap)
```

Description

`geostruct = updategeostruct(displaystruct)` accepts a Mapping Toolbox display structure `displaystruct`. If `displaystruct` is a vector display structure for which the `'type'` field has value `'line'` or `'patch'`, `updategeostruct` restructures its elements to create a geostruct, `geostruct`. If `displaystruct` is a already geographic data structure, it is copied unaltered to `geostruct`. `updategeostruct` does not update display structure arrays of type `'text'`, `'light'`, `'regular'`, or `'surface'`.

`geostruct = updategeostruct(displaystruct, str)` selects only elements whose `tag` field begins with the string scalar or character vector `str` (and whose `type` field is either `'line'` or `'patch'`). The selection is case insensitive.

`[geostruct,symbolspec] = updategeostruct(displaystruct, ...)` restructures a display structure and determines a symbolspec based on the graphic properties specified in the `otherproperty` field for each element of `displaystruct` and, if necessary, the `jet` colormap.

`[geostruct,symbolspec] = updategeostruct(displaystruct, ..., cmap)` specifies a colormap, `cmap`, to define the colors used in symbolspec.

Examples

Update and display the Great Lakes display structure to a geostruct:

```
load greatlakes
cmap = cool(3*numel(greatlakes));
[gtlakes, spec] = updategeostruct(greatlakes, cmap);
lat = extractfield(gtlakes,'Lat');
lon = extractfield(gtlakes,'Lon');
lonlim = [min(lon) max(lon)];
latlim = [min(lat) max(lat)];
figure
usamap(latlim, lonlim);
geoshow(gtlakes, 'SymbolSpec', spec)
```
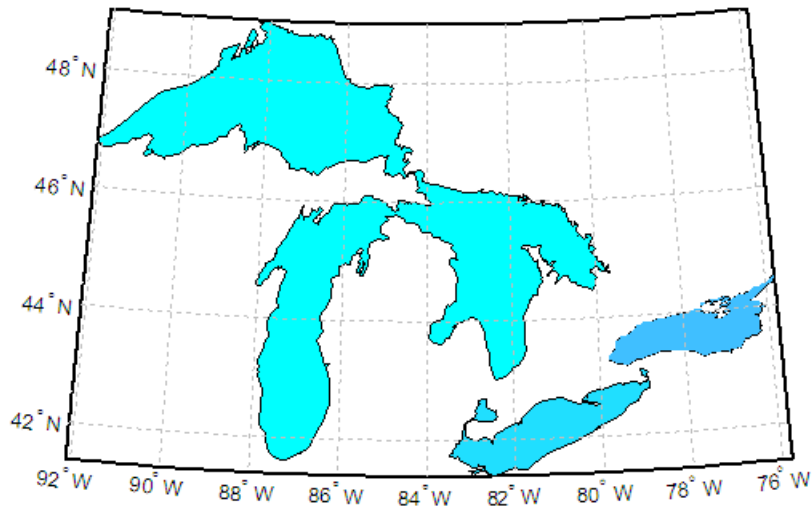
Tips

There are two Mapping Toolbox encodings for vector features that use MATLAB structure arrays. In both cases there is one feature per array element, and in both cases a given array's elements all held the same type of feature. Version 1.3.1 and earlier of the Mapping Toolbox software only supported Mapping Toolbox display structures. Version 2.0 introduced a data structure for vector geodata which was less rigidly defined and more open-ended. The new structures are called geostructs (if they contain geographic coordinate data) and mapstructs (if they contain projected coordinate data). Over time, display structures are being phased out of the toolbox; the `updategeostruct` function is provided to help users migrate from the old display structure format to the current geostruct/mapstruct format.

A Version 1 Mapping Toolbox display structure is a MATLAB structure that can contain line, patch, text, regular data grid, geolocated data grid, and light objects. The `displaym` function does not accept geostructs produced by Version 2 of the Mapping Toolbox software.

Display structures for lines and patches and Line and Polygon geostructs have the following things in common:

- A field that specifies the type of feature geometry:

    - A `type` field a display structure (value: `'line'` or `'patch'`)
    - A `Geometry` field for a geostruct (value: 'Line' or `'Polygon'`)
- A latitude field:

    - `lat` for a display structure
    - `Lat` for a geostruct
- A longitude field:

    - `long` for a display structure
    - `Lon` for a geostruct

In terms of their differences,

- A geostruct has a `BoundingBox` field; there is no display structure counterpart for this
- A geostruct typically has one or more "attribute" fields, whose values must be either scalar doubles or character vectors, with arbitrary field names. The presence or absence of a given attribute field—and its value—is dependent on the specific data set that the geostruct represents.
- A (line or patch) display structure has the following fields:
  - A `tag` field that names an individual feature or object
  - An `altitude` coordinate array that extends coordinates to 3-D
  - An `otherproperty` field in which MATLAB graphics can be specified explicitly, on a per-feature basis

Object properties used in the display are taken from the `otherproperty` field of the structure. If a line or patch object's `otherproperty` field is empty, `displaym` uses default colors. A patch is assigned an index into the current colormap based on the structure's `tag` field. Lines are assigned colors from the current color order according to their tags.

The newer geostruct representation has significant advantages:

- It can represent a much wider range of attributes (display structures essentially can represent only a feature name).
- The geostruct representation (in combination with `geoshow` and `makesymbolspec`) keeps graphics display properties separate from the intrinsic properties of the geographic features themselves.

For example, a road-class attribute can be used to display major highways with a distinctive color and greater line width than secondary roads. The same geographic data structure can be displayed in many different ways, without altering any of its contents, and shapefile data imported from external sources need not be altered to control its graphic display.

For information about the display structure format, see "Version 1 Display Structures" on page 1-286 in the reference page for `displaym`. For a discussion of the characteristics of geographic data structures, see "Geographic Data Structures".

See Also
`displaym` | `geoshow` | `makesymbolspec` | `mapshow` | `mapview` | `shaperead`

Introduced before R2006a

updateLayers

Update layer properties

Syntax

```
[updatedLayer,index] = updateLayers(server,layer)
```

Description

`[updatedLayer,index] = updateLayers(server,layer)` returns an array of `WMSLayer` objects and updates the layer properties with values from the web map server, `server`. The `WMSLayer` array `layer` must contain only one unique `ServerURL`. The `updateLayers` function removes layers no longer available on the server. The logical array `index` contains `true` for each available layer.

Examples

Update Properties of Layer

Update the properties of a MODIS global mosaic layer obtained from the NASA Earth Observations WMS server.

Search the WMS Database for the layer.

```
nasa = wmsfind('NASA Earth Observations','SearchField','any');
modis = refine(nasa,'land*day*month');
modis = modis(1);
```

Create a web map server object by specifying the server URL of the layer.

```
server = WebMapServer(modis.ServerURL);
```

Update the properties of the layer by specifying the server and the layer.

```
updatedLayer = updateLayers(server,modis);
```

Read and display the map.

```
mapRequest = WMSMapRequest(updatedLayer,server);
A = getMap(server,mapRequest.RequestURL);
R = mapRequest.RasterReference;

figure
ax = worldmap('world');
geoshow(A,R)
setm(ax,'MLabelParallel',-90,'MLabelLocation',90)
title({'MODIS Global Mosaic',modis.LayerTitle})
```
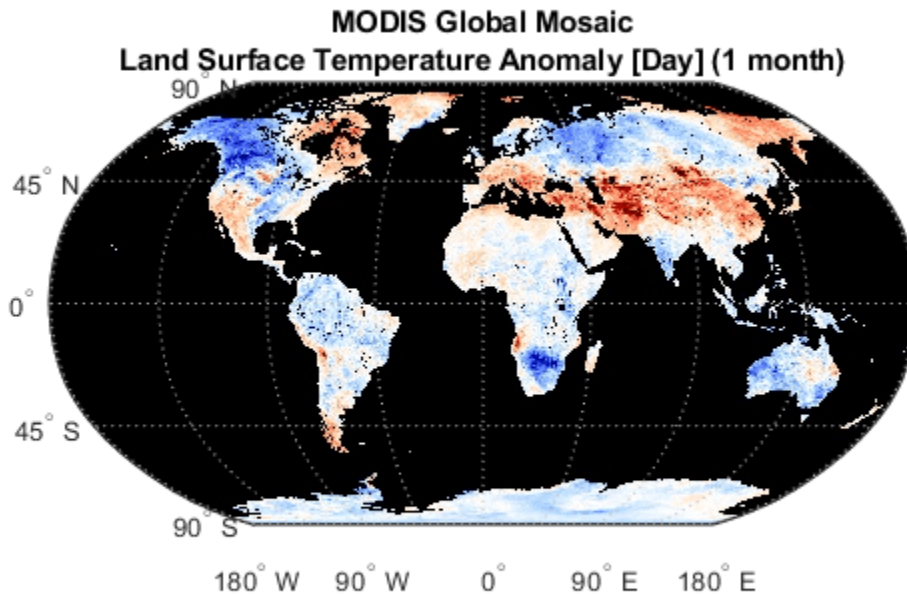
The layer used in this example is courtesy of the NASA Earth Observing System.

Update Properties of Layers from Multiple Servers

Search the WMS Database for layers from USGS servers that contain the word "imagery" in the server URL.

```
usgsLayers = wmsfind('usgs*imagery','SearchField','serverurl');
```

Create an array called `updatedLayers`. Use a loop to find the layers for individual servers, update the properties of the layers, and append the updated layers to the array. Display a message to the command line if a server does not respond.

```
updatedLayers = [];
serverURLs = usgsLayers.servers;

for k=1:numel(serverURLs)
   serverLayers = refine(usgsLayers, serverURLs{k}, ...
          'SearchField','serverurl','MatchType','exact');
   serverURL = serverLayers(1).ServerURL;
   server = WebMapServer(serverURL);
   try
      layers = updateLayers(server,serverLayers);
      % Grow using concatenation because layers can have any
      % length ranging from 0 to numel(serverLayers).
```

```
        updatedLayers = [updatedLayers; layers];
    catch e
        fprintf('Server %s is not responding.\n', ...
                serverURL);
        fprintf('Error message is %s\n', e.message)
    end
end
```

Input Arguments

server — Web map server
array of `WebMapServer` objects

Web map server, specified as an array of `WebMapServer` objects.

layer — Web map service layers
`WMSLayer` object

Web map service layer, specified as a `WMSLayer` object.

Output Arguments

updatedLayer — Updated web map service layers
array of `WMSLayer` objects.

Updated web map service layers, returned as an array of `WMSLayer` objects. `updatedLayers` has the same size as `layer(index)`.

index — Availability of layers
logical array

Availability of layers, returned as a logical array. `index` contains `true` for each available layer.

Tips

`updateLayers` accesses the Internet to update the properties. Occasionally, a WMS server is unavailable, or several minutes elapse before the properties are updated.

See Also

Introduced before R2006a

usamap

Construct map axes for United States of America

Syntax

```
usamap state
usamap(state)
usamap 'conus'
usamap('conus')
usamap
usamap(latlim,lonlim)
usamap(Z,R)
h = usamap( ___ )
h = usamap('all')
```

Description

`usamap state` and

`usamap(state)` create an empty map axes with a Lambert Conformal Conic projection and map limits covering a U.S. state or group of states specified by `state`. The map axes is created in the current axes and the axis limits are set tight around the map frame.

`usamap 'conus'` and

`usamap('conus')` create an empty map axes for the conterminous 48 states (that is, all states excluding Alaska and Hawaii).

`usamap` with no arguments presents a menu from which you can select a single state, the District of Columbia, the conterminous 48 states, or all states.

`usamap(latlim,lonlim)` creates an empty Lambert Conformal map axes for a region of the U.S. defined by its latitude and longitude limits in degrees.

`usamap(Z,R)` derives the map limits from the extent of a regular data grid, Z, georeferenced by R.

`h = usamap( ___ )` returns the handle of the map axes.

`h = usamap('all')` constructs three empty map axes, inset within a single figure, for the conterminous states, Alaska, and Hawaii, with a spherical Earth model and other projection parameters suggested by the U.S. Geological Survey. The maps in the three axes are shown at approximately the same scale. The handles for the three map axes are returned in h.

`usamap('allequal')` is the same as `usamap('all')`, but usage of `'allequal'` will be removed in a future release.

Examples

Make a Map of Alabama

Make a map of the state of Alabama only.

```
figure
usamap("Alabama")

states = readgeotable("usastatehi.shp");
row = states.Name == "Alabama";
alabama = states(row,:);

geoshow(alabama,"FaceColor",[0.3 1.0, 0.675])
```

Label the state by adding text.

```
textm(alabama.LabelLat,alabama.LabelLon,alabama.Name, ...
   "HorizontalAlignment","center")
```
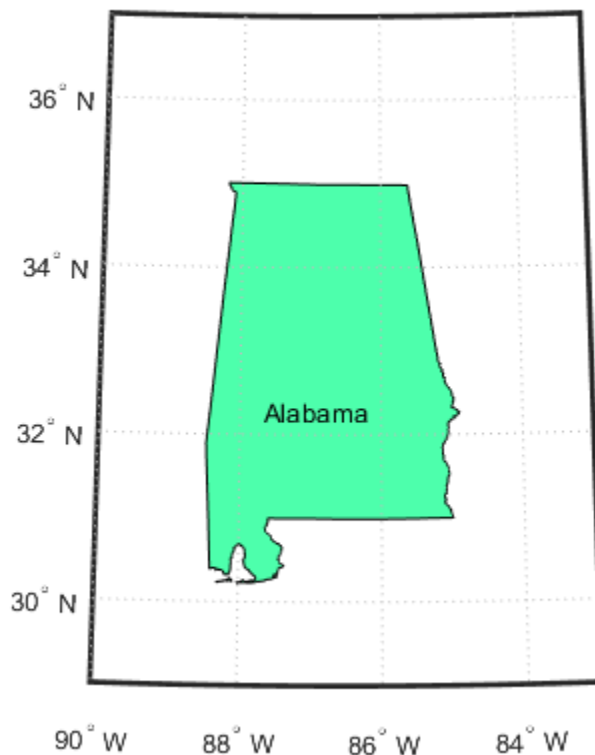

Map a Region Extending From California to Montana

Create a map of a contiguous landmass that contains California and Montana.

```
figure
ax = usamap({'CA','MT'});
set(ax,'Visible','off')
```

```
states = readgeotable('usastatehi.shp');
geoshow(states, 'FaceColor', [0.5 0.5 1])
```

Add labels to the states that are within the map limits.

```
latlim = getm(ax,'MapLatLimit');
lonlim = getm(ax,'MapLonLimit');

lat = states.LabelLat;
lon = states.LabelLon;
tf = ingeoquad(lat,lon,latlim,lonlim);
textm(lat(tf),lon(tf),states.Name(tf), ...
    'HorizontalAlignment','center')
```
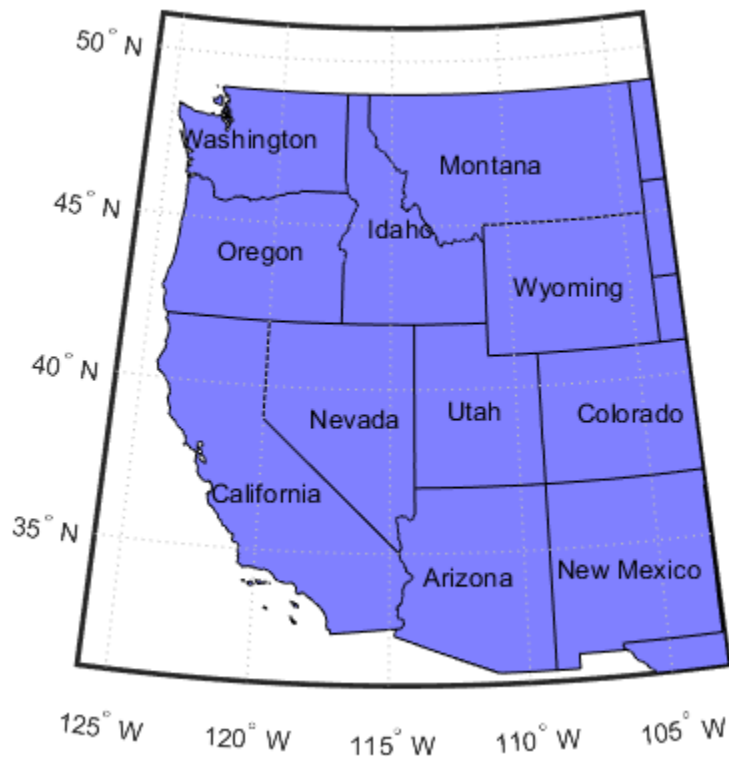

Map the Conterminous United States

Map the conterminous United States. Color each state using a random, muted color.

```
figure
usamap("conus");

states = readgeotable("usastatelo.shp");
rows = states.Name ~= "Alaska" & states.Name ~= "Hawaii";
states = states(rows,:);
```

```
h = height(states);
faceColors = makesymbolspec("Polygon",...
    {'INDEX',[1 h],'FaceColor',polcmap(h)});

geoshow(states,"DisplayType","polygon","SymbolSpec",faceColors)
```

Set optional display settings.

```
framem off
gridm off
mlabel off
plabel off
```
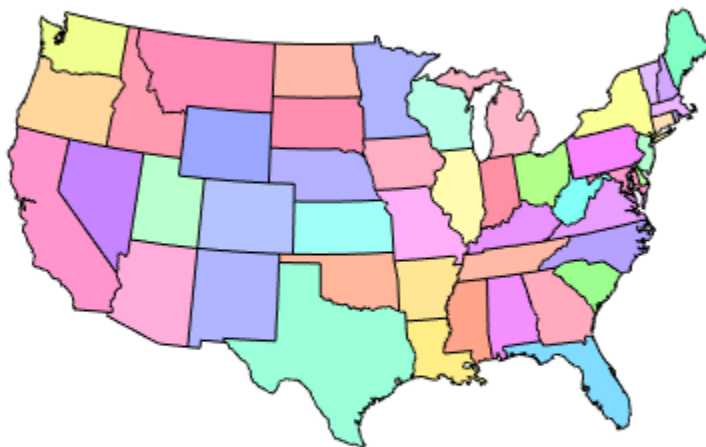

Map the USA Including Alaska and Hawaii

Map the USA with separate axes for Alaska and Hawaii.

```
figure
ax = usamap('all');
set(ax, 'Visible', 'off')
states = shaperead('usastatelo.shp', 'UseGeoCoords', true);
names = {states.Name};
indexHawaii = strcmp('Hawaii',names);
indexAlaska = strcmp('Alaska',names);
```

```
indexConus = 1:numel(states);
indexConus(indexHawaii|indexAlaska) = [];
stateColor = [0.5 1 0.5];
```

Display the three regions.

```
geoshow(ax(1), states(indexConus),  'FaceColor', stateColor)
geoshow(ax(2), states(indexAlaska), 'FaceColor', stateColor)
geoshow(ax(3), states(indexHawaii), 'FaceColor', stateColor)
```

Hide the frame.

```
for k = 1:3
    setm(ax(k), 'Frame', 'off', 'Grid', 'off',...
        'ParallelLabel', 'off', 'MeridianLabel', 'off')
end
```
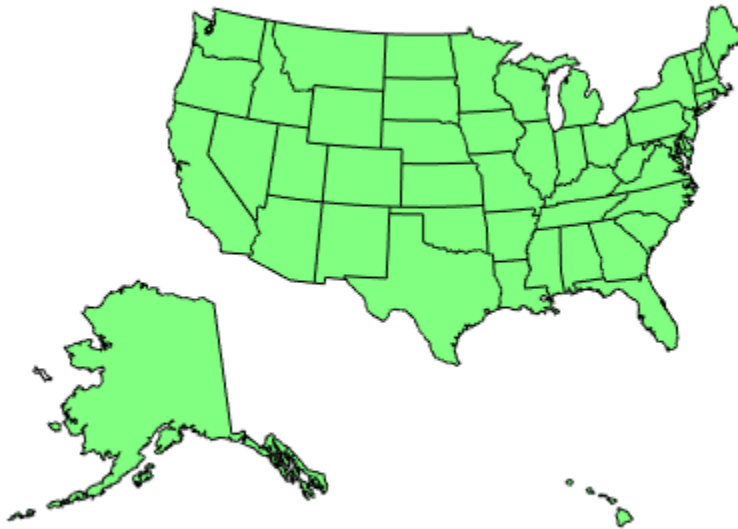

Input Arguments

state — State to display

character vector | string scalar | string array | cell array of character vectors | `'District of Columbia'` | `'Alabama'` | `'AL'` | `'Alaska'` | `'AK'` | ...

State to display, specified as a string scalar, string array, character vector or cell array of character vectors. Permissible values include names of states, standard two-letter U.S. Postal Service abbreviations for states, and `'District of Columbia'`.

Example: usamap({'Maine','Florida'}) sets the map limits to cover the region spanning from Maine to Florida.

latlim — Latitude limits
two-element vector

Latitude limits, specified as a two-element vector of the form [southern_limit northern_limit].

lonlim — Longitude limits
two-element vector

Longitude limits, specified as a two-element vector of the form [western_limit eastern_limit].

Z — Data grid
M-by-*N* array

Data grid, specified as an *M*-by-*N* array. Z is a regular data grid associated with a geographic reference R.

R — Geographic reference
geographic raster reference object | vector | matrix

Geographic reference, specified as one of the following.

Type	Description
Geographic raster reference object	GeographicCellsReference or GeographicPostingsReference geographic raster reference object. The RasterSize property must be consistent with the size of the data grid, size(Z).
Vector	1-by-3 numeric vector with elements: [cells/degree northern_latitude_limit western_longitude_limit]
Matrix	3-by-2 numeric matrix that transforms raster row and column indices to or from geographic coordinates according to: [lon lat] = [row col 1] * R R defines a (non-rotational, non-skewed) relationship in which each column of the data grid falls along a meridian and each row falls along a parallel.

For more information about referencing vectors and matrices, see "Georeferenced Raster Data".

Output Arguments

h — Handle of the map axes
handle object | array of handle objects

Handle of the map axes, returned as a handle object.

If you use the syntax h = usamap('all'), then h is array of handle objects. h(1) is for the conterminous states, h(2) is for Alaska, and h(3) is for Hawaii.

Tips

- All axes created with `usamap` are initialized with a spherical Earth model having a radius of 6,371,000 meters.
- In some cases, `usamap` uses `tightmap` to adjust the axis limits tight around the map. If you change the projection, or just want more white space around the map frame, use `tightmap` again or `axis auto`.
- `axes(h(n))`, where n = 1, 2, or 3, makes the desired axes current.
- `set(h,'Visible','on')` makes the axes visible.
- `axesscale(h(1))` resizes the axes containing Alaska and Hawaii to the same scale as the conterminous states.

See Also

axesm | axesscale | geoshow | paperscale | plotedit | tightmap | worldmap

Introduced before R2006a

usgs24kdem

(To be removed) Read USGS 7.5 minute (30 meter or 10 meter) Digital Elevation Models

---

Note `usgs24kdem` will be removed in a future release. Use `readgeoraster` instead. For more information, see "Compatibility Considerations".

Syntax

```
[lat,lon,Z] = usgs24kdem
[lat,lon,Z] = usgs24kdem(filename)
[lat,lon,Z] = usgs24kdem(filename,samplefactor)
[lat,lon,Z] = usgs24kdem(filename,samplefactor,latlim,lonlim)
[lat,lon,Z] = usgs24kdem(filename,samplefactor,latlim,lonlim,gsize)
[lat,lon,Z,header,profile] = usgs24kdem( ___ )
```

Description

`[lat,lon,Z] = usgs24kdem` reads a USGS 1:24,000 digital elevation map (DEM) file in standard format. You select the file interactively. `usgs24kdem` reads the entire file, subsampled by a factor of 5, returning a geolocated data grid with a latitude array, `lat`, longitude array, `lon`, and an elevation array, Z. Horizontal units are in degrees, vertical units might vary. The 1:24,000 series of DEMs store data as a grid of elevations spaced either at 10 or 30 meters apart. The number of points in a file varies with the geographic location.

`[lat,lon,Z] = usgs24kdem(filename)` reads the USGS DEM specified by `filename` and returns the result as a geolocated data grid.

`[lat,lon,Z] = usgs24kdem(filename,samplefactor)` reads a subset of the DEM data from `filename`, where `samplefactor` is a scalar integer that specifies the sample frequency.

`[lat,lon,Z] = usgs24kdem(filename,samplefactor,latlim,lonlim)` reads the subset of the elevation data from `filename` specified by the two-element vectors `latlim` and `lonlim`. You specify the latitude and longitude limits in degrees. Elements in the vectors must be in ascending order. The data might extend outside the requested area.

`[lat,lon,Z] = usgs24kdem(filename,samplefactor,latlim,lonlim,gsize)` specifies the graticule size in `gsize`. `gsize` is a two-element vector specifying the number of rows and columns in the latitude and longitude coordinated grid.

`[lat,lon,Z,header,profile] = usgs24kdem( ___ )` also returns the contents of the header and raw profiles of the DEM file. The `header` structure contains descriptions of the data from the file header. The `profile` structure is the raw profile data from which the geolocated data grid is constructed.

Examples

Read USGS 24K DEM File

This example shows how to read a USGS 24K Digital Elevation Model file.

Unzip a USGS 24K DEM file. The toolbox includes a DEM file `sanfranciscos.dem.gz` .

```
filenames = gunzip('sanfranciscos.dem.gz',tempdir);
demFilename = filenames{1};
```

Read every other point of the 1:24,000 DEM file.

```
[lat,lon,Z,header,profile] = usgs24kdem(demFilename,2);
```

Delete the temporary gunzipped file.

```
delete(demFilename)
```

As no negative elevations exist, move all points at sea level to -1 to color them blue.

```
Z(Z==0) = -1;
```

Compute the latitude and longitude limits for the DEM.

```
latlim = [min(lat(:)) max(lat(:))]
lonlim = [min(lon(:)) max(lon(:))]

latlim =

   37.6249   37.7504


lonlim =

 -122.5008 -122.3740
```

Display the DEM values.

```
figure
usamap(latlim,lonlim)
geoshow(lat,lon,Z,'DisplayType','surface')
demcmap(Z)
daspectm('m',1)
```
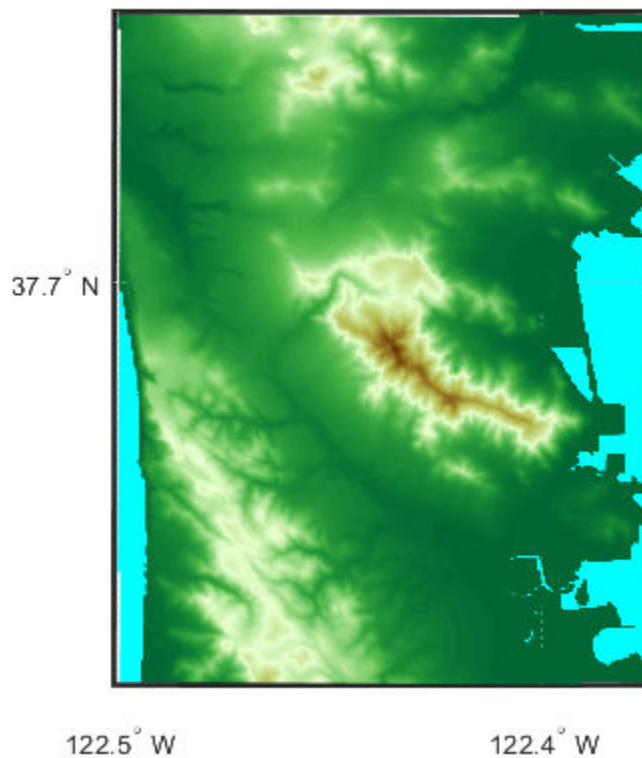
Input Arguments

`filename` — Name of file containing the digital elevation map
string scalar | character array

Name of file containing the digital elevation map, specified as a string scalar or character array.

Data Types: `char` | `string`

`samplefactor` — Data sampling factor
5 (default) | scalar integer

Data sampling factor, specified as a scalar integer. For example, if `samplefactor` is equal to 1, `usgs24kdem` reads the data at its full resolution, that is, every pixel. If you specify a `samplefactor` value n that is greater than 1, `usgs24kdem` reads every nth point.

Data Types: `double`

`latlim` — Limits of the desired data
two-element vector

Limits of the desired data, specified as a two-element vector, in degrees. The limits must be in ascending order. The data might extend outside the requested area.

Data Types: `double`

lonlim — Limits of desired data
two-element vector

Limits of desired data, specified as a two-element vector, in degrees.

Data Types: `double`

gsize — Graticule size
same size as geolocated data grid (default) | two-element vector

Graticule size, specified as a two-element vector. `gsize` specifies the number of rows and columns in the latitude and longitude coordinated grid. If omitted, `usgs24kdem` returns a graticule the same size as the geolocated data grid. To specify the coordinated grid size without specifying the geographic limits, use empty matrices for `latlim` and `lonlim`.

Data Types: `double`

Output Arguments

lat — Latitude array
matrix of class `double`

Latitude array, returned as a matrix of class `double`.

lon — Longitude array
matrix of class `double`

Longitude array, returned as a matrix of class `double`.

Z — Elevation array
matrix of class `double`

Elevation array, returned as a matrix of class `double`.

header — Descriptions of the data from the file header
struct

Descriptions of the data from the file header, returned as a `struct`.

profile — Raw profile data from which the geolocated data grid is constructed
struct

Raw profile data from which the geolocated data grid is constructed, returned as a `struct`.

Tips

- The U.S. Geological Survey has created a series of digital elevation models based on their paper 1:24,000 scale maps. The grid spacing for these elevations models is either 10 meters or 30 meters on a Universal Transverse Mercator grid. Each file covers a 7.5-minute quadrangle. The map and data series are available for much of the conterminous United States, Hawaii, and Puerto Rico. The data has been released in several formats. This function reads the data in the "standard" file format.

- This function reads USGS DEM files stored in the UTM projection. The function unprojects the grid back to latitude and longitude. Use `usgsdem` for data stored in geographic grids.

- The number of points in a file varies with the geographic location. Unlike the USGS DEM products, which use an equal-angle grid, the UTM projection grid DEMs cannot simply be concatenated to cover larger areas. There can be data gaps between DEMs.

- You can obtain the data files from the U.S. Geological Survey and from commercial vendors . Other agencies have made some local area data available online. See "Find Geospatial Data Online". The DEM files are ASCII files, and can be transferred as text. Line-ending conversion is not necessarily required.

Compatibility Considerations

`usgs24kdem` will be removed
Warns starting in R2021b

Some raster reading functions that return latitude-longitude grids will be removed, including `usgs24kdem`. Instead, use `readgeoraster`, which returns a map raster reference object. Reference objects have several advantages over latitude-longitude grids.

- Unlike latitude-longitude grids, reference objects have properties that document the size of the associated raster, its limits, and the direction of its rows and columns. For more information about reference object properties, see `MapCellsReference` and `MapPostingsReference`.

- You can manipulate the limits of rasters associated with reference objects using the `mapcrop` function.

- You can manipulate the size and resolution of rasters associated with reference objects using the `mapresize` function.

Get metadata about files using the `georasterinfo` function.

This table shows some typical usages of `usgs24kdem` and how to update your code to use `readgeoraster`.

Will Be Removed	Recommended
`[latgrat,longrat,z] = usgs24kdem(filename);`	`[Z,R] = readgeoraster(filename);`
`[latgrat,longrat,z] = usgs24kdem(filename,samplefactor);`	`[Z,R] = readgeoraster(filename);` `[Z,R] = georesize(Z,R,1/samplefactor);`

`[latgrat,longrat,z] = usgs24kdem(filename,samplefactor,latlim,lonlim);`	`[Z,R] = readgeoraster(filename);`
	`[Z,R] = geocrop(Z,R,latlim,lonlim);`
	`[Z,R] = georesize(Z,R,1/samplefactor);`

`[latgrat,longrat,z,header,profile] = usgs24kdem(filename);`	`[Z,R] = readgeoraster(filename);` `info = georasterinfo(filename);`

The `readgeoraster` function returns data using the native data type embedded in the file. Return a different data type by specifying the `'OutputType'` name-value pair. For example, use `[Z,R] = readgeoraster(filename,'OutputType','double')`.

The `readgeoraster` function does not automatically replace missing data with `NaN` values. If your data set uses large negative numbers to indicate missing data, replace them with `NaN` values using the `standardizeMissing` function.

```
[Z,R] = readgeoraster('MtWashington-ft.grd');
info = georasterinfo('MtWashington-ft.grd');
m = info.MissingDataIndicator;
Z = standardizeMissing(Z,m);
```

See Also
demdataui | usgsdems | readgeoraster | georasterinfo

Introduced before R2006a

usgsdem

(To be removed) Read USGS 1-degree (3-arc-second) Digital Elevation Model

---

Note `usgsdem` will be removed in a future release. Use `readgeoraster` instead. For more information, see "Compatibility Considerations".

Syntax

```
[Z,refvec] = usgsdem(filename,scalefactor)
[Z,refvec] = usgsdem(filename,scalefactor,latlim,lonlim)
```

Description

`[Z,refvec] = usgsdem(filename,scalefactor)` reads the specified file and returns the data in a regular data grid along with referencing vector `refvec`, a 1-by-3 vector having elements `[cells/degree north-latitude west-longitude]` with latitude and longitude limits specified in degrees. The data can be read at full resolution (`scalefactor` = 1), or can be downsampled by the `scalefactor`. A `scalefactor` of 3 returns every third point, giving 1/3 of the full resolution.

`[Z,refvec] = usgsdem(filename,scalefactor,latlim,lonlim)` reads data within the latitude and longitude limits. These limits are two-element vectors with the minimum and maximum values specified in units of degrees.

Background

The U.S. Geological Survey has made available a set of digital elevation maps of 1-degree quadrangles covering the contiguous United States, Hawaii, and limited portions of Alaska. The data is on a regular grid with a spacing of 30 arc-seconds (or about 100-meter resolution). 1-degree DEMs are also referred to as *3-arc-second* or *1:250,000 scale* DEM data.

The data is derived from the U.S. Defense Mapping Agency's DTED-1 digital elevation model, which itself was derived from cartographic and photographic sources. The cartographic sources were maps from the 7.5-minute through 1-degree series (1:24,000 scale through 1:250,000 scale).

Examples

Read every fifth point in the file containing part of Rhode Island and Cape Cod.

```
[Z,refvec] = usgsdem('providence-e',5);
```

Tips

The grid for the digital elevation maps is based on the 1984 World Geodetic System (WGS84). Older DEMs were based on WGS72. Elevations are in meters relative to National Geodetic Vertical Datum of 1929 (NGVD 29) in the continental U.S. and local mean sea level in Hawaii.

The absolute horizontal accuracy of the DEMs is 130 meters, while the absolute vertical accuracy is ±30 meters. The relative horizontal and vertical accuracy is not specified, but is probably much better than the absolute accuracy.

These DEMs have a grid spacing of 3 arc-seconds in both the latitude and longitude directions. The exception is DEM data in Alaska, where latitudes between 50 and 70 degrees North have grid spacings of 6 arc-seconds, and latitudes greater than 70 degrees North have grid spacings of 9 arc-seconds.

Statistical data in the files is not returned.

You can obtain the data files from the U.S. Geological Survey and from commercial vendors. Other agencies have made some local area data available online.

Note For details on locating map data for download over the Internet, see the following documentation at the MathWorks Web site: "Find Geospatial Data Online".

Compatibility Considerations

usgsdem will be removed
Warns starting in R2021b

Raster reading functions that return referencing vectors will be removed, including `usgsdem`. Instead, use `readgeoraster`, which returns a geographic raster reference object. Reference objects have several advantages over referencing vectors.

- Unlike referencing vectors, reference objects have properties that document the size of the associated raster, its geographic limits, and the direction of its rows and columns. For examples of reference object properties, see `GeographicCellsReference` and `GeographicPostingsReference`.
- You can manipulate the limits of rasters associated with reference objects using the `geocrop` function.
- You can manipulate the size and resolution of rasters associated with reference objects using the `georesize` function.
- Most functions that accept referencing vectors as input also accept reference objects.

This table shows some typical usages of `usgsdem` and how to update your code to use `readgeoraster` instead. The `readgeoraster` function requires you to specify a file extension.

Will Be Removed	Recommended
`[Z,refvec] = usgsdem(filename,scalefactor);`	`[Z,R] = readgeoraster(filename);` `[Z,R] = georesize(Z,R,1/scalefactor);`

`[Z,refvec] = usgsdem(filename,scalefactor,latlim,lonlim);`	`[Z,R] = readgeoraster(filename);`
	`[Z,R] = geocrop(Z,R,latlim,lonlim);`
	`[Z,R] = georesize(Z,R,1/scalefactor);`

The `readgeoraster` function returns data using the native data type embedded in the file. Return a different data type by specifying the `'OutputType'` name-value pair. For example, use `[Z,R] = readgeoraster(filename,'OutputType','double')`.

The `readgeoraster` function does not automatically replace missing data with `NaN` values. If your data set uses large negative numbers to indicate missing data, you can replace them with `NaN` values using the `standardizeMissing` function.

```
[Z,R] = readgeoraster('MtWashington-ft.grd');
info = georasterinfo('MtWashington-ft.grd');
m = info.MissingDataIndicator;
Z = standardizeMissing(Z,m);
```

See Also
usgsdems | readgeoraster | georasterinfo

Introduced before R2006a

usgsdems

USGS 1-degree (3-arc-sec) DEM file names for latitude-longitude quadrangle

Syntax

```
[fname,qname] = usgsdems(latlim,lonlim)
```

Description

`[fname,qname] = usgsdems(latlim,lonlim)` returns cell arrays of the DEM file names and quadrangle names covering the geographic region. The region is specified by scalar latitude and longitude points or two-element vectors of latitude and longitude limits in units of degrees.

Background

The U.S. Geological Survey has made available a set of digital elevation maps of 1-degree quadrangles covering the contiguous United States, Hawaii, and limited portions of Alaska. These are referred to as *1-degree*, *3-arc second* or *1:250,000 scale* DEMs. Because the file names of these 1 degree data sets are taken from the names of cities or features in the quadrangle, determining the files needed to cover a particular region generally requires consulting an index map or other reference. This function takes the place of such a reference by returning the file names for a given geographic region.

Examples

Which files are needed to map part of New England?

```
usgsdems([41 44], [-72 -69])

ans =
    'providence-w'
    'providence-e'
    'chatham-w'
    'boston-w'
    'boston-e'
    'portland-w'
    'portland-e'
    'bath-w'
```

Tips

This function only returns file names for the contiguous United States.

See Also
```
readgeoraster
```

Introduced before R2006a

utmgeoid

Select ellipsoids for given UTM zone

Syntax

```
ellipsoid = utmgeoid
ellipsoid = utmgeoid(zone)
[ellipsoid,ellipsoidstr] = utmgeoid(...)
```

Description

The purpose of this function is to recommend a local ellipsoid for use with a given UTM zone, depending on the geographic location of that zone. Use it only if you are not using a global reference ellipsoid, such as the World Geodetic System (WGS) 1984 ellipsoid. In many cases, depending on your application, you should just use the output of `wgs84Ellipsoid`, or one of the other options available through `referenceEllipsoid`.

`ellipsoid = utmgeoid`, without any arguments, opens the `utmzoneui` interface for selecting a UTM zone. This zone is then used to return the recommended ellipsoid definitions for that particular zone.

`ellipsoid = utmgeoid(zone)` uses the input *zone* to return the recommended ellipsoid definitions.

`[ellipsoid,ellipsoidstr] = utmgeoid(...)` returns the short name(s) for the reference ellipsoid(s), as used by `referenceEllipsoid`, in a `char` array with one name in each row.

Background

The Universal Transverse Mercator (UTM) system of projections tiles the world into quadrangles called zones. Each zone has different projection parameters and commonly used ellipsoidal models of the Earth. This function returns a list of ellipsoid models commonly used in a zone.

Examples

```
zone = utmzone(0,100) % degrees

zone =
47N

[ellipsoid,names] = utmgeoid(zone)

ellipsoid =
      6377.3      0.081473
      6377.4      0.081697
names =
everest
bessel
```

See Also

referenceEllipsoid | wgs84Ellipsoid

Introduced before R2006a

utmzone

Select UTM zone given latitude and longitude

Syntax

```
zone = utmzone
zone = utmzone(lat,long)
zone = utmzone(mat),
[latlim,lonlim] = utmzone(zone)
lim = utmzone(zone)
```

Description

`zone = utmzone` selects a Universal Transverse Mercator (UTM) zone with a graphical user interface. Returns the UTM zone designation as a character vector.

`zone = utmzone(lat,long)` returns the UTM zone containing the geographic coordinates. If `lat` and `long` are vectors, the zone containing the geographic mean of the data set is returned. The geographic coordinates must be in units of degrees.

`zone = utmzone(mat),` where `mat` is of the form `[lat long]`.

`[latlim,lonlim] = utmzone(zone)` returns the geographic limits of the zone, where `zone` is a valid UTM zone designation. Valid UTM zones designations are numbers, or numbers followed by a single letter. For example, `'31'` or `'31N'`. The returned limits are in units of degrees.

`lim = utmzone(zone)` returns the limits in a single vector output.

Background

The Universal Transverse Mercator (UTM) system of projections tiles the world into quadrangles called zones. This function can be used to identify which zone is used for a geographic area and, conversely, what geographic limits apply to a UTM zone.

Examples

```
[latlim,lonlim] = utmzone('12F')

latlim =
    -56    -48
lonlim =
  -114   -108

utmzone(latlim,lonlim)

ans =
12F
```

Limitations

The UTM zone system is based on a regular division of the globe, with the exception of a few zones in northern Europe. `utmzone` does not account for these deviations.

See Also

`utmgeoid`

Introduced before R2006a

validateLengthUnit

Validate and standardize length unit

Syntax

```
standardName = validateLengthUnit(unit)
standardName = validateLengthUnit(unit,funcName,varName,argIndex)
```

Description

`standardName = validateLengthUnit(unit)` checks that `unit` is a valid length unit and converts it to a standard unit name. The function is case-insensitive with respect to its input. Spaces, periods, and apostrophes are ignored. Plural forms are accepted in most cases, but the result, `standardName` is always singular.

`standardName = validateLengthUnit(unit,funcName,varName,argIndex)` checks that `unit` is a valid length unit and, if it isn't, creates an error message using the optional inputs `funcName`, `varName`, and `argIndex` in error message formatting, with behavior identical to that provided by the `validateattributes` inputs of the same names.

Examples

Find Valid Length Unit Name

Find the valid length unit name for `'foot'` when other valid strings for `'foot'` are input.

```
validateLengthUnit('foot')
```

```
ans =
'foot'
```

```
validateLengthUnit('feet')
```

```
ans =
'foot'
```

```
validateLengthUnit('international feet')
```

```
ans =
'foot'
```

Find the valid length unit name for `'kilometer'` when other valid strings for `'kilometer'` are input.

```
validateLengthUnit('kilometer')
```

```
ans =
'kilometer'
```

```
validateLengthUnit('km')
```

```
ans =
'kilometer'
```

```
validateLengthUnit('kilometre')
```

```
ans =
'kilometer'
```

```
validateLengthUnit('kilometers')
```

```
ans =
'kilometer'
```

```
validateLengthUnit('kilometres')
```

```
ans =
'kilometer'
```

Find the valid length unit name when the name contains an apostrophe.

```
validateLengthUnit('Clarke''s foot')
```

```
ans =
'Clarke's foot'
```

Create Custom Error Message

Create custom error messages using the `validateLengthUnit` function. An invalid input to `validateLengthUnit` results in an error message referencing a function name, `'FOO'`, a variable name, `'UNIT'` and an argument number, 5.

```
validateLengthUnit(17,'FOO','UNIT',5)
```

```
Error using FOO
Expected input number 5, UNIT, to be one of these types:

char, string

Instead its type was double.

Error in validateLengthUnit (line 87)
validateattributes(unit,{'char','string'},{'nonempty','scalartext'},varargin{:})
```

Input Arguments

unit — Length unit
character vector | string scalar

Length unit, specified as any of the following:

Unit Name	Value(s)
meter	`'m'`, `'meter(s)'`, `'metre(s)'`
centimeter	`'cm'`, `'centimeter(s)'`, `'centimetre(s)'`

Unit Name	Value(s)
millimeter	'mm', 'millimeter(s)', 'millimetre(s)'
micron	'micron(s)'
kilometer	'km', 'kilometer(s)', 'kilometre(s)'
nautical mile	'nm', 'naut mi', 'nautical mile(s)'
foot	'ft','international ft', 'foot','international foot', 'feet','international feet'
inch	'in', 'inch', 'inches'
yard	'yd', 'yds', 'yard(s)'
mile	'mi', 'mile(s)', 'international mile(s)'
U.S. survey foot	'sf', 'survey ft', 'US survey ft', 'U.S. survey ft', 'survey foot', 'US survey foot', 'U.S. survey foot', 'survey feet', 'US survey feet', 'U.S. survey feet'
U.S. survey mile (statute mile)	'sm', 'survey mile(s)', 'statute mile(s)', 'US survey mile(s)', 'U.S. survey mile(s)'
Clarke's foot	'Clarke's foot', 'Clarkes foot'
German legal metre	'German legal metre', 'German legal meter'
Indian foot	'Indian foot'

Example: validateLengthUnit('feet')

Data Types: char | string

funcName — Name of the function whose input you are validating
character vector | string scalar

Name of the function whose input you are validating, specified as a string scalar or character vector. If you specify an empty string or character vector (''), the validateLengthUnit function ignores the funcName value.

Example: validateLengthUnit(17,'FOO','UNIT',5)

Data Types: char | string

varName — Name of input variable
character vector | string scalar

Name of input variable, specified as a string scalar or character vector. If you specify an empty string or character vector (''), the validateLengthUnit function ignores the varName value.

Example: validateLengthUnit(17,'FOO','UNIT',5)

Data Types: `char` | `string`

`argIndex` — Position of the input argument
positive integer

Position of the input argument, specified as a positive integer.

Example: `validateLengthUnit(17,'FOO','UNIT',5)`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

`standardName` — Standard length unit name
character vector

Standard length unit name, returned as a character vector.

See Also
`unitsratio`

Introduced before R2006a

vec2mtx

Convert latitude-longitude vectors to regular data grid

Syntax

```
[Z,R] = vec2mtx(lat,lon,density)
[Z,R] = vec2mtx(lat,lon,density,latlim,lonlim)
[Z,R] = vec2mtx(lat,lon,Z1,R1)
[Z,R] = vec2mtx(...,'filled')
```

Description

`[Z,R] = vec2mtx(lat,lon,density)` creates a regular data grid `Z` from vector data, placing ones in grid cells intersected by a vector and zeroes elsewhere. `R` is the raster reference object for the computed grid. `lat` and `lon` are vectors of equal length containing geographic locations in units of degrees. `density` indicates the number of grid cells per unit of latitude and longitude (a value of 10 indicates 10 cells per degree, for example), and must be scalar-valued. Whenever there is space, a buffer of two grid cells is included on each of the four sides of the grid. The buffer is reduced as needed to keep the latitudinal limits within [-90 90] and to keep the difference in longitude limits from exceeding 360 degrees.

`[Z,R] = vec2mtx(lat,lon,density,latlim,lonlim)` uses the two-element vectors `latlim` and `lonlim` to define the latitude and longitude limits of the grid.

`[Z,R] = vec2mtx(lat,lon,Z1,R1)` uses a pre-existing data grid `Z1`, georeferenced by `R1`, to define the limits and density of the output grid. `R1` can be a referencing vector, a referencing matrix, or a geographic raster reference object.

If `R1` is a geographic raster reference object, its `RasterSize` property must be consistent with `size(Z1)` and its `RasterInterpretation` must be `'cells'`.

If `R1` is a referencing vector, it must be a 1-by-3 vector containing these elements:

`[cells/degree northern_latitude_limit western_longitude_limit]`

If `R1` is a referencing matrix, it must be 3-by-2 and transform raster row and column indices to or from geographic coordinates according to this equation:

`[lon lat] = [row col 1] * R1`

The matrix must define a (non-rotational, non-skewed) relationship in which each column of the data grid falls along a meridian and each row falls along a parallel.

With this syntax, the output `R` is equal to `R1`, and may be a referencing object, vector, or matrix.

`[Z,R] = vec2mtx(...,'filled')`, where `lat` and `lon` form one or more closed polygons (with NaN-separators), fills the area outside the polygons with the value two instead of the value zero.

Notes

Empty `lat,lon` vertex arrays will result in an error unless the grid limits are explicitly provided (via `latlim,lonlim` or `Z1,R1`). In the case of explicit limits, `Z` will be filled entirely with 0s if the `'filled'` parameter is omitted, and 2s if it is included.

It's possible to apply `vec2mtx` to sets of polygons that tile without overlap to cover an area, as in Example 1 below, but using `'filled'` with polygons that actually overlap may lead to confusion as to which areas are inside and which are outside.

Examples

Example 1

Convert latitude-longitude polygons to a regular data grid and display as a map.

```
states = shaperead('usastatelo', 'UseGeoCoords', true);
lat = [states.Lat];
lon = [states.Lon];
[Z,R] = vec2mtx(lat,lon,5,'filled');
figure
worldmap(Z,R)
geoshow(Z,R,'DisplayType','texturemap')
colormap(flag(3))
```
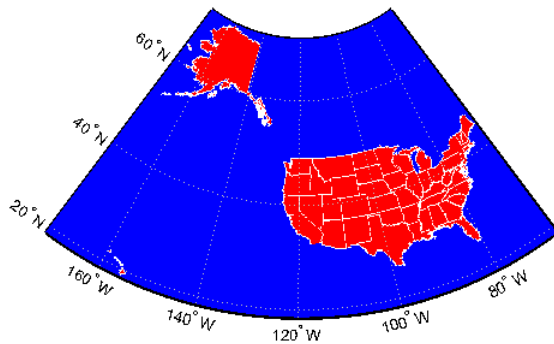

Example 2

Combine two separate calls to `vec2mtx` to create a 4-color raster map showing interior land areas, coastlines, oceans, and world rivers.

```
load coastlines
[Z,R] = vec2mtx(coastlat,coastlon,1, ...
                [-90 90],[-90 270],'filled');

rivers = shaperead('worldrivers.shp','UseGeoCoords',true);
A = vec2mtx([rivers.Lat],[rivers.Lon],Z,R);
Z(A == 1) = 3;
figure
worldmap(Z,R)
```

```
geoshow(Z,R,'DisplayType','texturemap')
colormap([.45 .60 .30; 0 0 0; 0 0.5 1; 0 0 1])
```
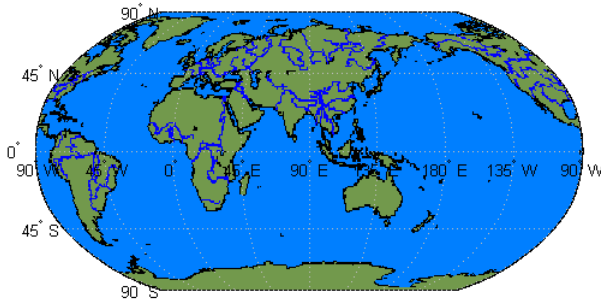

Define Limits and Density of Output Grid Using Spatial Referencing Object

Import US state polygons as a geospatial table. Extract the coordinates of the polygons by converting the geospatial table to a table.

```
states = readgeotable("usastatelo.shp");
T = geotable2table(states,["Latitude" "Longitude"]);
[lat,lon] = polyjoin(T.Latitude',T.Longitude');
```

Choose geographic limits.

```
latlim = [  15  75];
lonlim = [-190 -65];
```

Specify a grid with 5 cells per degree.

```
density = 5;
```

Compute raster size. (M and N both work out to be integers.)

```
M = density * diff(latlim);
N = density * diff(lonlim);
```

Create a geographic raster reference object.

```
R = georasterref("RasterSize", [M N], ...
    "ColumnsStartFrom","north","Latlim",latlim, ...
        "Lonlim",lonlim);
```

Create a blank grid that is consistent with R in size. `vec2mtx` requires a data grid as input.

```
Z = zeros(R.RasterSize);
```

Overwrite Z with a new grid including state outlines and interiors.

```
Z = vec2mtx(lat,lon,Z,R,"filled");
```

Plot the georeferenced grid.

```
figure; worldmap(Z,R);
geoshow(Z,R,"DisplayType", "texturemap")
colormap(flag(3))
```
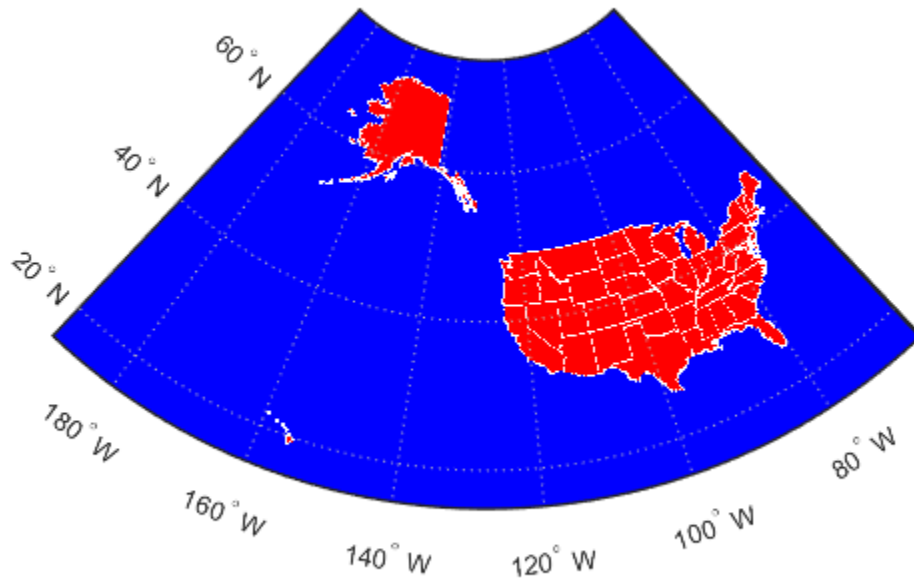

Compatibility Considerations

vec2mtx returns reference objects instead of referencing vectors
Behavior changed in R2021a

Starting in R2021a, most syntaxes of the vec2mtx function return a raster reference object instead of a referencing vector. This change is unlikely to affect your existing code because most Mapping Toolbox functions that accept referencing vectors as input also accept raster reference objects.

If you specify a referencing vector or matrix as input, then the vec2mtx function still returns a referencing vector or matrix.

See Also

Functions
imbedm

Objects
GeographicCellsReference | GeographicPostingsReference

Introduced before R2006a

vertcat

Vertically concatenate geographic or planar vectors

Syntax

```
v = vertcat(v1,v2,...)
```

Description

`v = vertcat(v1,v2,...)` vertically concatenates the geographic or planar vectors `v1`, `v2`, and so on. If the class type of any property is a cell array, then the resultant field in the output `v` is also a cell array.

Examples

Vertically Concatenate Mapshape Vectors

Create three mapshape vectors. The second vector has two features and is size 2x1.

```
ms1 = mapshape(42, -110, 'Temperature', 65, 'Name', 'point1');
ms2 = mapshape({50, 50.1}, {-101 -101.4}, 'Temperature', {73.2 77}, 'Name', {'point2','point3'})
ms3 = mapshape(42.1, -110.4, 'Temperature', 65.5, 'Name', 'point4');
```

Vertically concatenate the vectors into a single mapshape vector.

```
ms = vertcat(ms1, ms2, ms3)

ms =
 4x1 mapshape vector with properties:

 Collection properties:
       Geometry: 'line'
       Metadata: [1x1 struct]
 Vertex properties:
  (4 features concatenated with 3 delimiters)
              X: [42 NaN 50 NaN 50.1000 NaN 42.1000]
              Y: [-110 NaN -101 NaN -101.4000 NaN -110.4000]
    Temperature: [65 NaN 73.2000 NaN 77 NaN 65.5000]
 Feature properties:
           Name: {'point1'  'point2'  'point3'  'point4'}
```

The concatenated mapshape vector `ms` is size 4x1 and has four features. Note that the property `'Temperature'` is a Vertex property in `ms` because it is a Vertex property in `ms2`, even though it is a Feature property in `ms1` and `ms3`.

Input Arguments

`v1,v2,...` — Geographic or planar vectors to be concatenated
`geopoint`, `geoshape`, `mappoint`, or `mapshape` objects

Geographic or planar vectors to be concatenated, specified as one or many `geopoint`, `geoshape`, `mappoint`, or `mapshape` objects. All of `v1`, `v2`,... are the same type of object.

Output Arguments

v — Concatenated geographic or planar vector
geopoint, geoshape, mappoint, or mapshape object

Concatenated geographic or planar vector, returned as a `geopoint`, `geoshape`, `mappoint`, or `mapshape` object. The object type of `v` matches the object type of `v1,v2,...`.

See Also
`cat`

Introduced in R2012a

vfwdtran

Transform azimuth on ellipsoid to direction on map

Syntax

```
th = vfwdtran(lat,lon,az)
th = vfwdtran(mstruct,lat,lon,az)
[th,len] = vfwdtran(...)
```

Description

`th = vfwdtran(lat,lon,az)` transforms the azimuth angle at specified latitude and longitude points on the sphere into the projection space. The map projection currently displayed is used to define the projection space. The input angles must be in the same units as specified by the current map projection. The inputs can be scalars or matrices of the equal size. The angle in the projection space is defined as positive counterclockwise from the *x*-axis.

`th = vfwdtran(mstruct,lat,lon,az)` uses the map projection defined by the input `mstruct` to compute the map projection.

`[th,len] = vfwdtran(...)` also returns the vector length in the projected coordinate system. A value of 1 indicates no scale distortion.

Background

The direction of north is easy to define on the three-dimensional sphere, but more difficult on a two-dimensional map. For cylindrical projections in the normal aspect, north is always in the positive *y*-direction. For conic projections, north can be to the left or right of the *y*-axis. This function transforms any azimuth angle on the sphere to the corresponding angle in the projected paper coordinates.

Examples

Sample calculations:

```
axesm('eqdconic','maplatlim',[-10 45],'maplonlim',[-55 55])
gridm; framem; mlabel; plabel
quiverm([0 0 0],[-45 0 45],[0 0 0],[10 10 10],0)
quiverm([0 0 0],[-45 0 45],[10 10 10],[0 0 0],0)
```
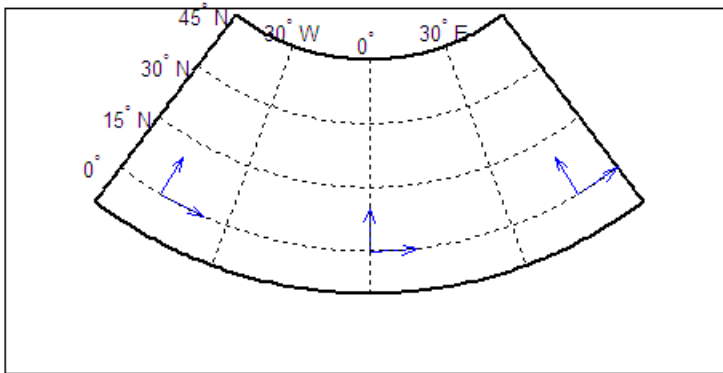
```
vfwdtran([0 0 0],[-45 0 45],[0 0 0])

ans =
      59.614              90        120.39

vfwdtran([0 0 0],[-45 0 45],[90 90 90])

ans =
     -30.385     0.0001931        30.386
```

Limitations

This transformation is limited to the region specified by the frame limits in the current map definition.

Tips

The geographic azimuth angle is measured clockwise from north. The projection space angle is measured counterclockwise from the *x*-axis.

This function uses a finite difference technique. The geographic coordinates are perturbed slightly in different directions and projected. A small amount of error is introduced by numerical computation of derivatives and the variation of map distortion parameters.

See Also
`vinvtran` | `projinv` | `projfwd` | `defaultm`

Introduced before R2006a

viewshed

Areas visible from point on terrain elevation grid

Syntax

```
[vis,R] = viewshed(Z,R,lat1,lon1)
viewshed(Z,R,lat1,lon1,observerAltitude)
viewshed(Z,R,lat1,lon1,observerAltitude,targetAltitude)
viewshed(Z,R,lat1,lon1,observerAltitude,targetAltitude, ...
  observerAltitudeOption)
viewshed(Z,R,lat1,lon1,observerAltitude,targetAltitude, ...
  observerAltidueOption,targetAltitudeOption)
viewshed(Z,R,lat1,lon1,observerAltitude,targetAltitude, ...
  observerAltitudeOption,targetAltitudeOption,actualRadius)
viewshed(Z,R,lat1,lon1,observerAltitude,targetAltitude, ...
  observerAltitudeOption,targetAltitudeOption, ...
  actualRadius,effectiveRadius)
```

Description

`[vis,R] = viewshed(Z,R,lat1,lon1)` computes areas visible from a point on a digital elevation grid. `Z` is a regular data grid containing elevations in units of meters. The observer location is provided as scalar latitude and longitude in units of degrees. The visibility grid `vis` contains `1`s at the surface locations visible from the observer location, and `0`s where the line of sight is obscured by terrain. `R` can be a geographic raster reference object, a referencing vector, or a referencing matrix.

If `R` is a geographic raster reference object, its `RasterSize` property must be consistent with `size(Z)`.

If `R` is a referencing vector, it must be a 1-by-3 with elements:

`[cells/degree northern_latitude_limit western_longitude_limit]`

If `R` is a referencing matrix, it must be 3-by-2 and transform raster row and column indices to or from geographic coordinates according to:

`[lon lat] = [row col 1] * R`

If `R` is a referencing matrix, it must define a (non-rotational, non-skewed) relationship in which each column of the data grid falls along a meridian and each row falls along a parallel. Nearest-neighbor interpolation is used by default. NaN is returned for points outside the grid limits or for which `lat` or `lon` contain NaN. All angles are in units of degrees.

`viewshed(Z,R,lat1,lon1,observerAltitude)` places the observer at the specified altitude in meters above the surface. This is equivalent to putting the observer on a tower. If omitted, the observer is assumed to be on the surface.

`viewshed(Z,R,lat1,lon1,observerAltitude,targetAltitude)` checks for visibility of target points a specified distance above the terrain. This is equivalent to putting the target points on towers that do not obstruct the view. If omitted, the target points are assumed to be on the surface.

viewshed(Z,R,lat1,lon1,observerAltitude,targetAltitude, ...
    observerAltitudeOption) controls whether the observer is at a relative or absolute altitude. If the observerAltitudeOption is 'AGL', then observerAltitude is in meters above ground level. If observerAltitudeOption is 'MSL', observerAltitude is interpreted as altitude above zero, or mean sea level. If omitted, 'AGL' is assumed.

viewshed(Z,R,lat1,lon1,observerAltitude,targetAltitude, ...
    observerAltidueOption,targetAltitudeOption) controls whether the target points are at a relative or absolute altitude. If the target altitude option is 'AGL', the targetAltitude is in meters above ground level. If targetAltitudeOption is 'MSL', targetAltitude is interpreted as altitude above zero, or mean sea level. If omitted, 'AGL' is assumed.

viewshed(Z,R,lat1,lon1,observerAltitude,targetAltitude, ...
    observerAltitudeOption,targetAltitudeOption,actualRadius) does the visibility calculation on a sphere with the specified radius. If omitted, the radius of the earth in meters is assumed. The altitudes, the elevations, and the radius should be in the same units. This calling form is most useful for computations on bodies other than the Earth.

viewshed(Z,R,lat1,lon1,observerAltitude,targetAltitude, ...
    observerAltitudeOption,targetAltitudeOption, ...
    actualRadius,effectiveRadius) assumes a larger radius for propagation of the line of sight. This can account for the curvature of the signal path due to refraction in the atmosphere. For example, radio propagation in the atmosphere is commonly treated as straight line propagation on a sphere with 4/3 the radius of the Earth. In that case the last two arguments would be R_e and 4/3*R_e, where R_e is the radius of the earth. Use Inf for flat Earth viewshed calculations. The altitudes, the elevations, and the radii should be in the same units.

Examples

Compute Visibility for Point on Peaks Map

Compute visibility for a point on the peaks map. Add the detailed information for the line of sight calculation between two points from los2 .

Create the peak map and plot the visibility.

```
Z = 500*peaks(100);
refvec = [1000 0 0];
[lat1,lon1,lat2,lon2] = deal(-0.027,0.05,-0.093,0.042);

[visgrid,visleg] = viewshed(Z,refvec,lat1,lon1,100);
[vis,visprofile,dist,zi,lattrk,lontrk] ...
   = los2(Z,refvec,lat1,lon1,lat2,lon2,100);

axesm('globe','geoid',earthRadius('meters'))
meshm(visgrid,visleg,size(Z),Z);
axis tight
camposm(-10,-10,1e6)
camupm(0,0)
colormap(flipud(summer(2)))
brighten(0.75)
shading interp
camlight
h = lcolorbar({'obscured','visible'});
```

```
h.Position = [.875 .45 .02 .1];

plot3m(lattrk([1;end]),lontrk([1; end]), ...
    zi([1; end])+[100; 0],'r','linewidth',2)
plotm(lattrk(~visprofile),lontrk(~visprofile), ...
    zi(~visprofile),'r.','markersize',10)
plotm(lattrk(visprofile),lontrk(visprofile), ...
    zi(visprofile),'g.','markersize',10)
```
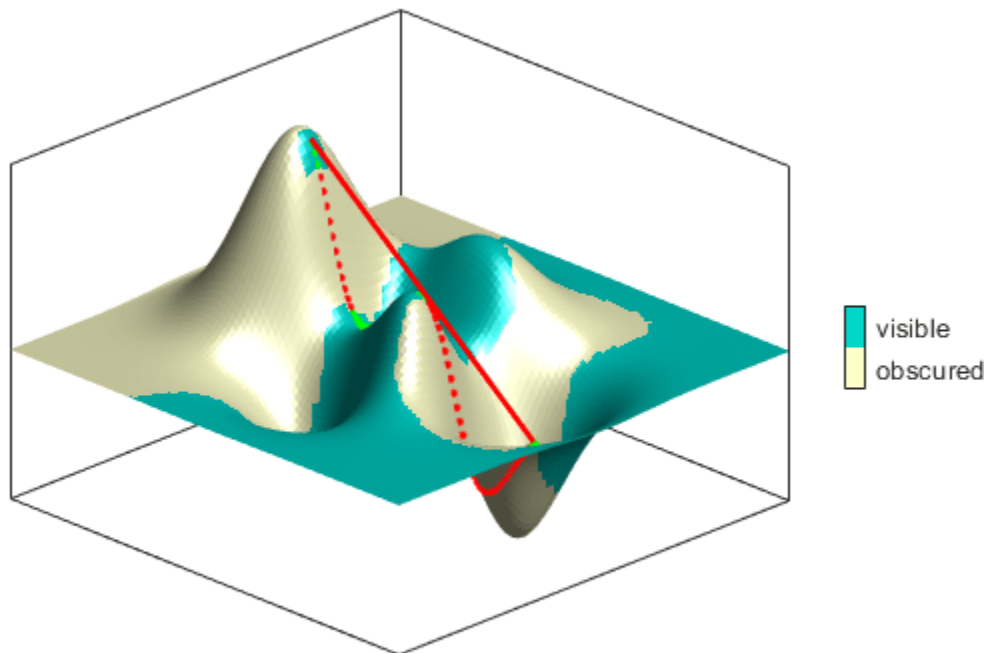

Compute Surface Area Visible By Radar

Compute the surface area visible by radar from an aircraft flying 3000 meters above the Yellow Sea. Assume that radio wave propagation in the atmosphere can be modeled as straight lines on a 4/3 radius Earth. This example also calculates the area visible to the plane above it at 5000 meters.

Load elevation data and a geographic cells reference object for the Korean peninsula, calculate the viewshed, and display it.

```
load korea5c
korea5c(korea5c<0) = -1;
figure
worldmap(korea5c,korea5cR)
setm(gca,'geoid',[1 0])
da = daspect;
pba = pbaspect;
```

```
da(3) = 7.5*pba(3)/da(3);
daspect(da);
demcmap(korea5c)
camlight(90,5);
camlight(0,5);
lighting gouraud
material([0.25 0.8 0])
lat = 34.0931; lon = 125.6578;
altobs = 3000; alttarg = 0;
plotm(lat,lon,'wo')
Re = earthRadius('meters');
[vmap,vmapl] = viewshed( ...
    korea5c,korea5cR,lat,lon,altobs,alttarg, ...
    'MSL','AGL',Re,4/3*Re);
meshm(vmap,vmapl,size(korea5c),korea5c)
```
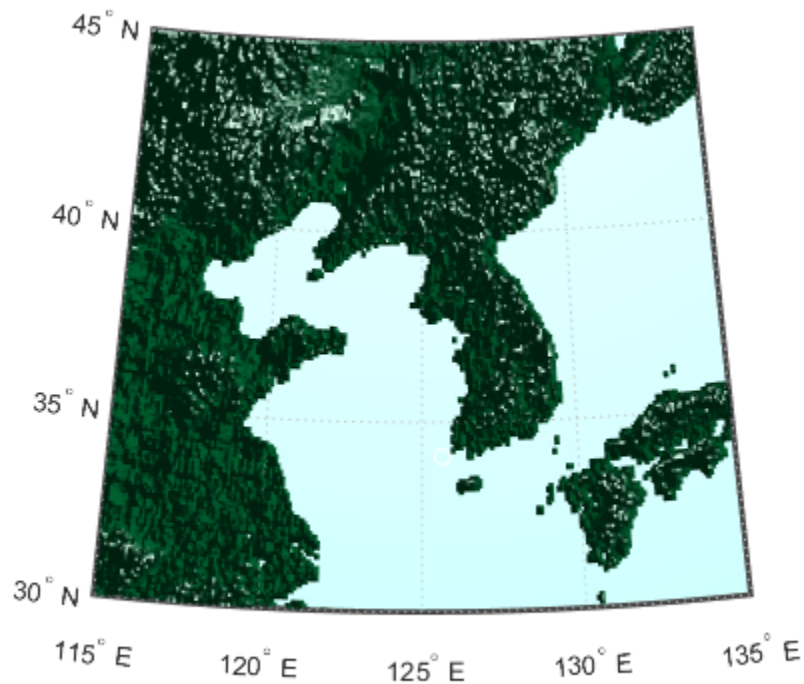

Display the visible areas as blue and the obscured areas as red and drape the visibility colors on an elevation map, using lighting to bring out the surface topography. The aircraft's radar can see out a certain radius on the surface of the ocean, but note how some ocean areas are shadowed by the island of Jeju-Do. Also some mountain valleys closer than the ocean horizon are obscured, while some mountain tops further away are visible.

```
caxis auto; colormap([1 0 0; 0 0 1])
lighting gouraud;
axis off
```
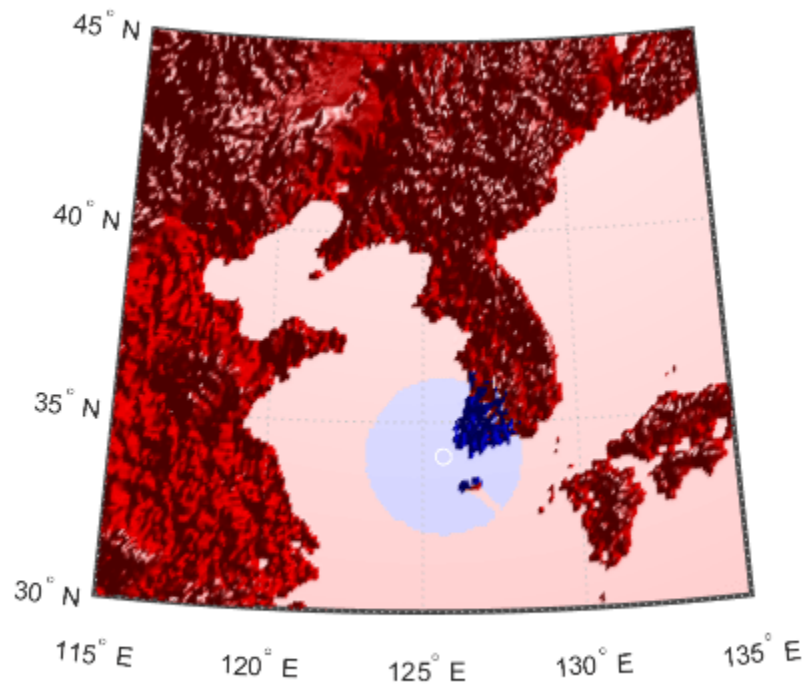
Now calculate the area that the radar plane flying at an altitude of 3000 meters can have line-of-sight to other aircraft flying above it at 5000 meters. Note how the area is much larger but that some edges of the area are reduced by shadowing from Jeju-Do and the mountains on the Korean peninsula.

```
[vmap,vmapl] = viewshed(korea5c,korea5cR,lat,lon,3000,5000, ...
               'MSL','MSL',Re,4/3*Re);
clmo surface
meshm(vmap,vmapl,size(korea5c),korea5c)
lighting gouraud
```
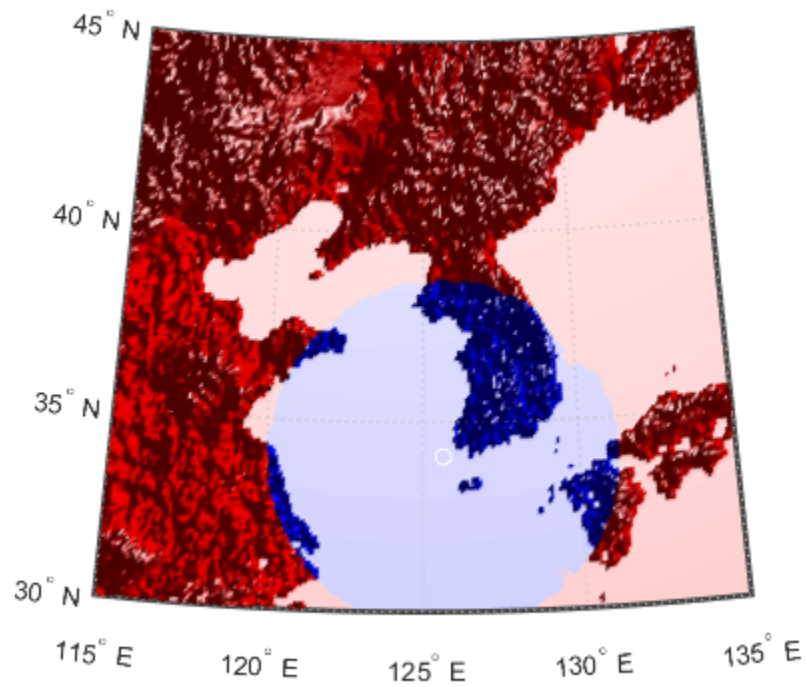
Tips

The observer should be located within the latitude-longitude limits of the elevation grid. If the observer is located outside the grid, there is insufficient information to calculate a viewshed. In this case `viewshed` issues a warning and sets all elements of `vis` to zero.

See Also
`los2`

Introduced before R2006a

vinvtran

Transform direction on map to azimuth on ellipsoid

Syntax

```
az = vinvtran(x,y,th)
az = vinvtran(mstruct,x,y,th)
[az,len] = vinvtran(...)
```

Description

`az = vinvtran(x,y,th)` transforms an angle in the projection space at the point specified by `x` and `y` into an azimuth angle in geographic coordinates. The map projection currently displayed is used to define the projection space. The input angles must be in the same units as specified by the current map projection. The inputs can be scalars or matrices of equal size. The angle in the projection space angle `th` is defined as positive counterclockwise from the *x*-axis.

`az = vinvtran(mstruct,x,y,th)` uses the map projection defined by the input `struct` to compute the map projection.

`[az,len] = vinvtran(...)` also returns the vector length in the geographic coordinate system. A value of 1 indicates no scale distortion for that angle.

Background

While vectors along the *y*-axis always point to north in a cylindrical projection in the normal aspect, they can point east or west of north on conic, azimuthal, and other projections. This function computes the geographic azimuth for angles in the projected space.

Examples

Sample calculations:

```
axesm('eqdconicstd','maplatlim',[-10 45],'maplonlim',[-55 55])
gridm; framem; mlabel; plabel; axis off
mstruct = gcm;
[x,y] = projfwd(mstruct,[0 0 0],[-45 0 45]);
quiver(x,y,[0.2 0.2 0.2],[0 0 0],0)
quiver(x,y,[0 0 0],[0.2 0.2 0.2],0)
```
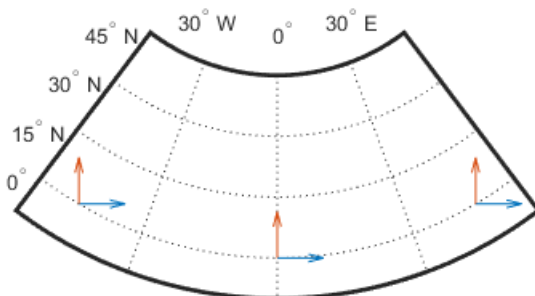
```
vinvtran(x,y,[0 0 0])

ans =

    56.1765    90.0000   123.8235

vinvtran(x,y,[90 90 90])

ans =

   332.8360     0.0000    27.1640
```

Limitations

This transformation is limited to the region specified by the frame limits in the current map definition.

Tips

The geographic azimuth angle is measured clockwise from north. The projection space angle is measured counterclockwise from the *x*-axis.

This function uses a finite difference technique. The geographic coordinates are perturbed slightly in different directions and projected. A small amount of error is introduced by numerical computation of derivatives and the variation of map distortion parameters.

See Also
`vfwdtran` | `projinv` | `projfwd` | `defaultm`

Introduced before R2006a

vmap0data

Read selected data from Vector Map Level 0

Syntax

```
struct = vmap0data(library,latlim,lonlim,theme,topolevel)
struct = vmap0data(devicename,library, ...)
[struct1,struct2,...] = vmap0data(...,{topolevel1,topolevel2,...})
```

Description

`struct = vmap0data(library,latlim,lonlim,theme,topolevel)` reads the data for the specified theme and topology level directly from the VMAP0 CD-ROM. There are four CDs, one for each of the libraries: `'NOAMER'` (North America), `'SASAUS'` (Southern Asia and Australia), `'EURNASIA'` (Europe and Northern Asia), and `'SOAMAFR'` (South America and Africa). The desired `theme` is specified by a two-letter code. A list of valid codes is displayed when an invalid code, such as `'?'`, is entered. `topolevel` defines the type of data returned: `'patch'`, `'line'`, `'point'`, or `'text'`. The region of interest can be given as a point latitude and longitude or as a region with two-element vectors of latitude and longitude limits. The units of latitude and longitude are degrees. The data covering the requested region is returned, but will include data extending to the edges of the tiles. The result is returned as a Mapping Toolbox Version 1 display structure.

`struct = vmap0data(devicename,library, ...)` specifies the logical device name of the CD-ROM for computers that do not automatically name the mounted disk.

`[struct1,struct2,...] = vmap0data(...,{topolevel1,topolevel2,...})` reads several topology levels. The levels must be specified as a cell array with the entries `'patch'`, `'line'`, `'point'`, or `'text'`. Entering `{'all'}` for the topology level argument is equivalent to `{'patch'`, `'line'`, `'point'`, `'text'}`. Upon output, the data structures are returned in the output arguments by topology level in the same order as they were requested.

Background

The Vector Map (VMAP) Level 0 database represents the third edition of the *Digital Chart of the World*. The second edition was a limited release item published in 1995. The product is dual named to show its lineage to the original DCW, published in 1992, while positioning the revised product within a broader emerging family of VMAP products. VMAP Level 0 is a comprehensive 1:1,000,000 scale vector base map of the world. It consists of cartographic, attribute, and textual data stored on compact disc read-only memory (CD-ROM). The primary source for the database is the Operational Navigation Chart (ONC) series of the U. S. National Geospatial Intelligence Agency (NGA), formerly the National Imagery and Mapping Agency (NIMA), and before that, the Defense Mapping Agency (DMA). This is the largest scale unclassified map series in existence that provides consistent, continuous global coverage of essential base map features. The database contains more than 1,900 MB of vector data and is organized into 10 thematic layers. The data includes major road and rail networks, major hydrologic drainage systems, major utility networks (cross-country pipelines and communication lines), all major airports, elevation contours (1000 foot (ft), with 500 ft and 250 ft supplemental contours), coastlines, international boundaries, and populated places. The database can be accessed directly from the four optical CD-ROMs that store the database or can be transferred to magnetic media.

Examples

The *devicename* is platform dependent. On an MS-DOS based operating system it would be something like `'d:'`, depending on the logical device code assigned to the CD-ROM drive. On a UNIX operating system, the CD-ROM might be mounted as `'\cdrom'`, `'\CDROM'`, `'\cdrom1'`, or something similar. Check your computer's documentation for the right *devicename*.

To view a list of valid themes, call `vmap0data` and specify an invalid theme, for example `'?'`. MATLAB errors and displays a list of valid themes.

```
s = vmap0data(devicename,'NOAMER',41,-69,'?','patch');

??? Error using ==> vmap0data
Theme not present in library NOAMER

Valid theme identifiers are:
libref : Library Reference
tileref: Tile Reference
bnd    : Boundaries
dq     : Data Quality
elev   : Elevation
hydro  : Hydrography
ind    : Industry
phys   : Physiography
pop    : Population
trans  : Transportation
util   : Utilities
veg    : Vegetation

BNDpatch = vmap0data(devicename,'NOAMER',...
                    [41 44],[-72 -69],'bnd','patch')
BNDpatch =
1x169 struct array with fields:
    type
    otherproperty
    altitude
    lat
    long
    tag
```

Here are some examples that specify valid themes:

```
[TRtext,TRline] = vmap0data(devicename,'SASAUS',...
    [-48 -34],[164 180],'trans',{'text','line'});

[BNDpatch,BNDline,BNDpoint,BNDtext] = vmap0data(devicename,...
    'EURNASIA',-48 ,164,'bnd',{'all'});
```

Tips

Data are returned as Mapping Toolbox display structures, which you can then update to geographic data structures. For information about display structure format, see "Version 1 Display Structures" on page 1-286 in the reference page for `displaym`. The `updategeostruct` function performs such conversions.

Latitudes and longitudes use WGS84 as a horizontal datum. Elevations and depths are in meters above mean sea level.

Some VMAP0 themes do not contain all topology levels. In those cases, empty matrices are returned.

Patches are broken at the tile boundaries. Setting the `EdgeColor` to `'none'` and plotting the lines gives the map a normal appearance.

The major differences between VMAP0 and the DCW are the elimination of the gazette layer, addition of bathymetric data, and updated political boundaries.

Vector Map Level 0, created in the 1990s, is still probably the most detailed global database of vector map data available to the public. VMAP0 CD-ROMs are available from through the U.S. Geological Survey (USGS):

USGS Information Services (Map and Book Sales)
Box 25286
Denver Federal Center
Denver, CO 80225
Telephone: (303) 202-4700
Fax: (303) 202-4693

Note For details on locating map data for download over the Internet, see the following documentation at the MathWorks Web site: "Find Geospatial Data Online".

See Also

vmap0read | vmap0rhead | geoshow | extractm | mlayers | updategeostruct

Introduced before R2006a

vmap0read

Read Vector Map Level 0 file

Syntax

```
vmap0read
vmap0read(filepath,filename)
vmap0read(filepath,filename,recordIDs)
vmap0read(filepath,filename,recordIDs,field,varlen)
struc = vmap0read(...)
[struc,field] = vmap0read(...)
[struc,field,varlen] = vmap0read(...)
[struc,field,varlen,description] = vmap0read(...)
[struc,field,varlen,description,narrativefield] = vmap0read(...)
```

Description

`vmap0read` reads a VMAP0 file. The user selects the file interactively.

`vmap0read(filepath,filename)` reads the specified file. The combination `[filepath filename]` must form a valid complete file name.

`vmap0read(filepath,filename,recordIDs)` reads selected records or fields from the file. If `recordIDs` is a scalar or a vector of integers, the function returns the selected records. If `recordIDs` is a cell array of integers, all records of the associated fields are returned.

`vmap0read(filepath,filename,recordIDs,field,varlen)` uses previously read field and variable-length record information to skip parsing the file header (see below).

`struc = vmap0read(...)` returns the file contents in a structure.

`[struc,field] = vmap0read(...)` returns the file contents and a structure describing the format of the file.

`[struc,field,varlen] = vmap0read(...)` also returns a vector describing which fields have variable-length records.

`[struc,field,varlen,description] = vmap0read(...)` also returns `description`, a character vector that describes the contents of the file.

`[struc,field,varlen,description,narrativefield] = vmap0read(...)` also returns the name of the narrative file for the current file.

Background

The Vector Map Level 0 (VMAP0) uses binary files in a variety of formats. This function determines the format of the file and returns the contents in a structure. The field names of this structure are the same as the field names in the VMAP0 file.

Examples

The following examples use the UNIX directory system and file separators for the path name:

```
s = vmap0read('VMAP/VMAPLV0/NOAMER/','GRT')

s =
                    id: 1
             data_type: 'GEO'
                 units: 'M'
         ellipsoid_name: 'WGS 84'
       ellipsoid_detail: 'A=6378137 B=6356752 Meters'
        vert_datum_name: 'MEAN SEA LEVEL'
        vert_datum_code: '015'
       sound_datum_name: 'N/A'
       sound_datum_code: 'N/A'
         geo_datum_name: 'WGS 84'
         geo_datum_code: 'WGE'
        projection_name: 'Dec. Deg. (unproj.)'

s = vmap0read('VMAP/VMAPLV0/NOAMER/TRANS/','INT.VDT')

s =
34x1 struct array with fields:
    id
    table
    attribute
    value
    description

s(1)

ans =
               id: 1
            table: 'aerofacp.pft'
        attribute: 'use'
            value: 8
      description: 'Military'
s = vmap0read('VMAP/VMAPLV0/NOAMER/TRANS/','AEROFACP.PFT',1)

s =
          id: 1
       f_code: 'GB005'
          iko: 'BGTL'
          nam: 'THULE AIR BASE'
          na3: 'GL52085'
          use: 8
          zv3: 77
      tile_id: 10
       end_id: 1

s = vmap0read('VMAP/VMAPLV0/NOAMER/TRANS/','AEROFACP.PFT',{1,2})

s =
1x4424 struct array with fields:
    id
    f_code
```

Tips

This function reads all VMAP0 files except index files (files with names ending in `'X'`), thematic index files (files with names ending in `'TI'`), and spatial index files (files with names ending in `'SI'`).

File separators are platform dependent. The *filepath* input must use appropriate file separators, which you can determine using the MATLAB `filesep` function.

See Also
`vmap0data` | `vmap0rhead`

Introduced before R2006a

vmap0rhead

Read Vector Map Level 0 file headers

Syntax

```
vmap0rhead
vmap0rhead(filepath,filename)
vmap0rhead(filepath,filename,fid)
vmap0rhead(...),
str = vmap0rhead(...)
```

Description

vmap0rhead allows the user to select the header file interactively.

vmap0rhead(filepath,filename) reads from the specified file. The combination [filepath filename] must form a valid complete file name.

vmap0rhead(filepath,filename,fid) reads from the already open file associated with fid.

vmap0rhead(...), with no output arguments, displays the formatted header information on the screen.

str = vmap0rhead(...) returns a character vector containing the VMAP0 header.

Background

The Vector Map Level 0 (VMAP0) uses the header in most files to document the contents and format of that file. This function reads the header and displays a formatted version in the Command Window, or returns it as a character vector.

Examples

The following example uses UNIX file separators and path name:

```
s = vmap0rhead('VMAP/VMAPLV0/NOAMER/','GRT')
s =
L;Geographic Reference Table;-;id=I,1,P,Row
Identifier,-,-,-,:data_type=T,3,N,Data
Type,-,-,-,:units=T,3,N,Units of Measure Code for
Library,-,-,-,:ellipsoid_name=T,15,N,Ellipsoid,-,-,-,:ellipsoid
_detail=T,50,N,Ellipsoid
Details,-,-,-,:vert_datum_name=T,15,N,Datum Vertical
Reference,-,-,-,:vert_datum_code=T,3,N,Vertical Datum
Code,-,-,-,:sound_datum_name=T,15,N,Sounding
Datum,-,-,-,:sound_datum_code=T,3,N,Sounding Datum
Code,-,-,-,:geo_datum_name=T,15,N,Datum Geodetic
Name,-,-,-,:geo_datum_code=T,3,N,Datum Geodetic
Code,-,-,-,:projection_name=T,20,N,Projection Name,-,-,-,:;
```

```
vmap0rhead('VMAP/VMAPLV0/NOAMER/TRANS/','AEROFACP.PFT')
L
Airport Point Feature Table
aerofacp.doc
id=I,1,P,Row Identifier,-,-,-,
f_code=T,5,N,FACC Feature Code,char.vdt,-,-,
iko=T,4,N,ICAO Designator,char.vdt,-,-,
nam=T,*,N,Name,char.vdt,-,-,
na3=T,*,N,Name,char.vdt,-,-,
use=S,1,N,Usage,int.vdt,-,-,
zv3=S,1,N,Airfield/Aerodrome Elevation (meters),int.vdt,-,-,
tile_id=S,1,N,Tile Reference ID,-,tile1_id.pti,-,
end_id=I,1,N,Entity Node Primitive ID,-,end1_id.pti,-,
```

Tips

This function reads all VMAP0 files except index files (files with names ending in `'X'`), thematic index files (files with names ending in `'TI'`) and spatial index files (files with names ending in `'SI'`).

File separators are platform dependent. The *filepath* input must use appropriate file separators, which you can determine using the MATLAB `filesep` function.

See Also

vmap0data | vmap0read

Introduced before R2006a

webmap

Open web map

Syntax

```
webmap
webmap(baseLayer)
webmap(wmsLayer)
webmap(customBasemap)
webmap( ___ ,'WrapAround',tf)
wm = webmap( ___ )
webmap(wm)
```

Description

webmap opens a new web map, with its center at the latitude, longitude point `[0 0]`. By default, webmap sets the base layer to World Street Map at the maximum spatial extent available. After the web map opens, you can select a different base layer using the Layer Manager available on the right side of the web map. Named base layers are tiled at discrete zoom resolutions.

Note The webmap function requires an Internet connection. MathWorks cannot guarantee the stability and accuracy of web maps, as the servers are on the Internet and are independent from MathWorks. Occasionally, maps may be slow to display, display partially, or fail to display, because web map servers can become unavailable for short periods of time.

---

webmap(baseLayer) opens a new web map with baseLayer as the default base layer. See baseLayer for a list of the available maps.

webmap(wmsLayer) Opens a new web map with wmsLayer as the default base layer. wmsLayer is a WMSLayer array. The LayerTitle property of each wmsLayer is set as an item in the Layer Manager.

webmap displays WMS layers in the "Web Mercator" map coordinate reference system, if that projection is available for all layers in the array. Otherwise, webmap displays the layers in the EPSG:4326 geographic coordinate reference system. When using EPSG:4326, webmap does not include the default base layers in the Layer Manager since they are in a different coordinate reference system. These projections include a geographic quadrangle bounded north/south by parallels and east/west by meridians. Parallels map to horizontal lines. Meridians map to vertical lines.

webmap(customBasemap) opens a new web map using the custom basemap specified by the addCustomBasemap function. customBasemap is a string scalar or character vector specifying the display name of the custom basemap, if provided, or the basemap name.

webmap(___ ,'WrapAround',tf), where tf is specified as the logical value false or 0, opens a new web map with the display clipped to the west at -180 degrees and to the east at +180 degrees. The default for tf is true or 1, which opens a map that supports continuous pan and zoom across the

180-degree meridian. The `webmap` function constrains zoom to show less than 180 degrees of longitude at a time.

`wm = webmap( ___ )` returns a handle to a web map, `wm`.

`webmap(wm)` makes the web map specified by `wm` the current web map.

Examples

Display Web Map

Open a web map centered at `[0 0]`.

`webmap`

View the list of available base layers (basemaps) by clicking the expander arrow ◀ on the right side of the map.
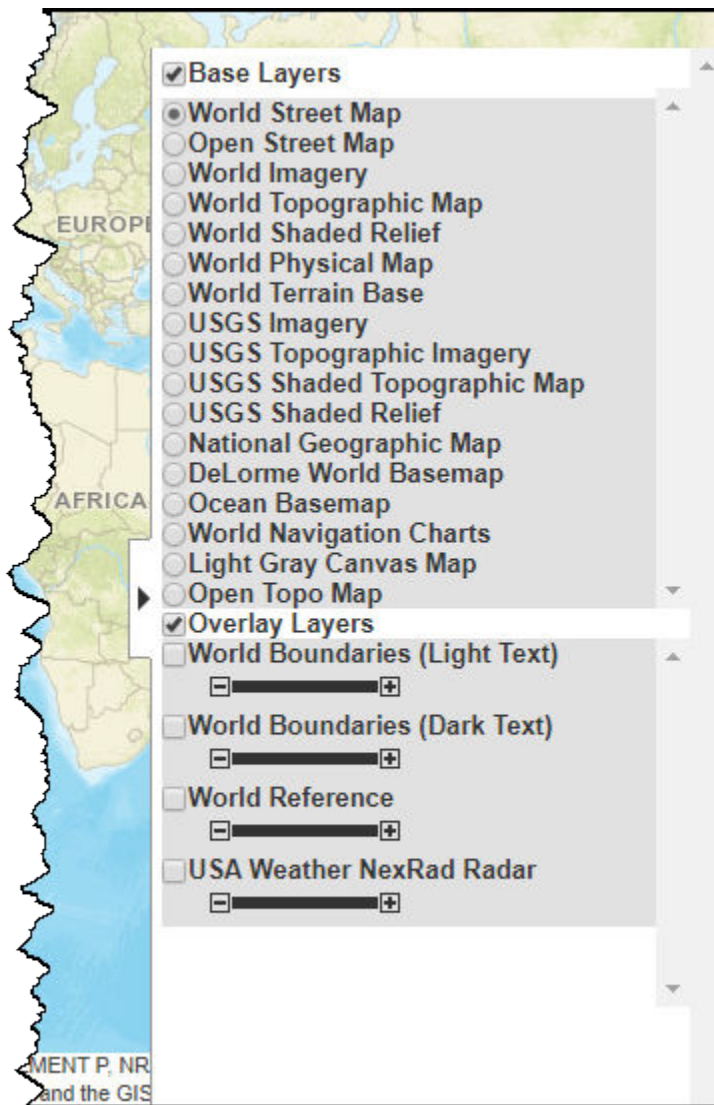
Display Web Map Specifying Base Layer

Open a web map specifying the base layer and show the full extent of the world.

```
webmap('Light Gray Canvas Map','WrapAround',false)
```

Display Web Map with WMS Layer as Base Layer

Read the Blue Marble WMS layer.

```
info = wmsinfo('https://neo.sci.gsfc.nasa.gov/wms/wms?');
nasa = info.Layer;
baselayer = refine(nasa,'bluemarbleng',  ...
                   'SearchField','layername',...
                   'MatchType','exact');
baselayer = wmsupdate(baselayer);
```

Display a web map with the Blue Marble WMS layer as the base layer.

```
webmap(baselayer)
```
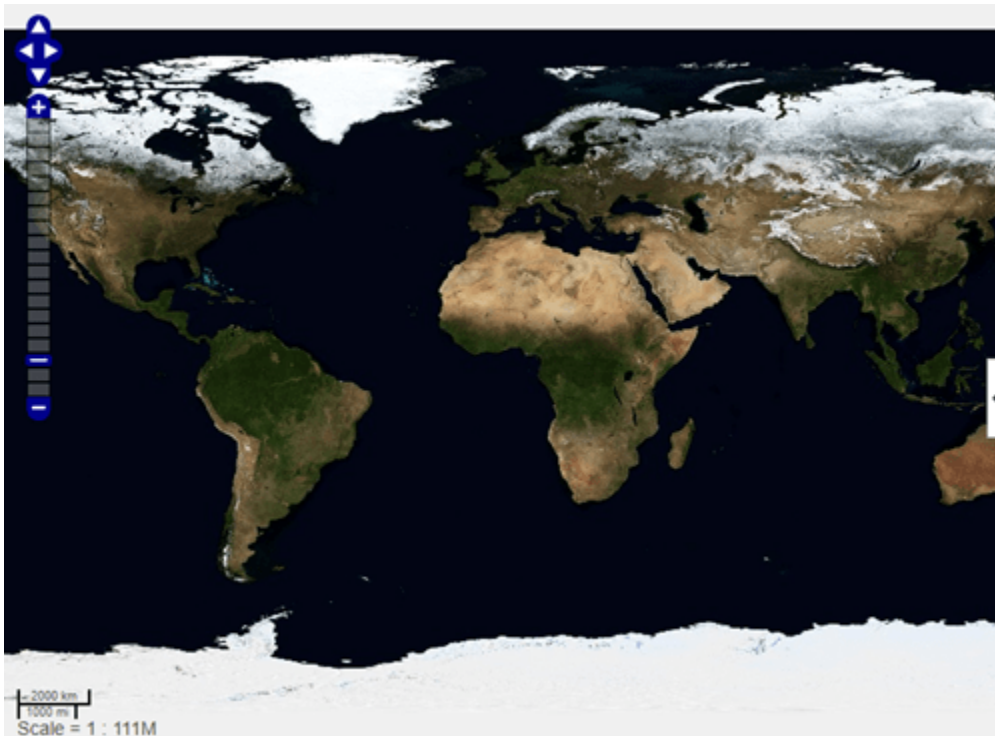
Display Web Map Using Custom Base Layer

Add a custom base layer. To do this, specify its name, URL, attribution, and display name.

```
name = 'opentopomap';
url = 'a.tile.opentopomap.org';
attribution = '©OpenStreetMap contributors';
displayName = 'Open Topo Map';
addCustomBasemap(name,url,'Attribution',attribution, ...
      'DisplayName',displayName)
```

Display a web map. To do this, call webmap and specify the base layer using the name you gave it when you created it.

```
webmap opentopomap
```
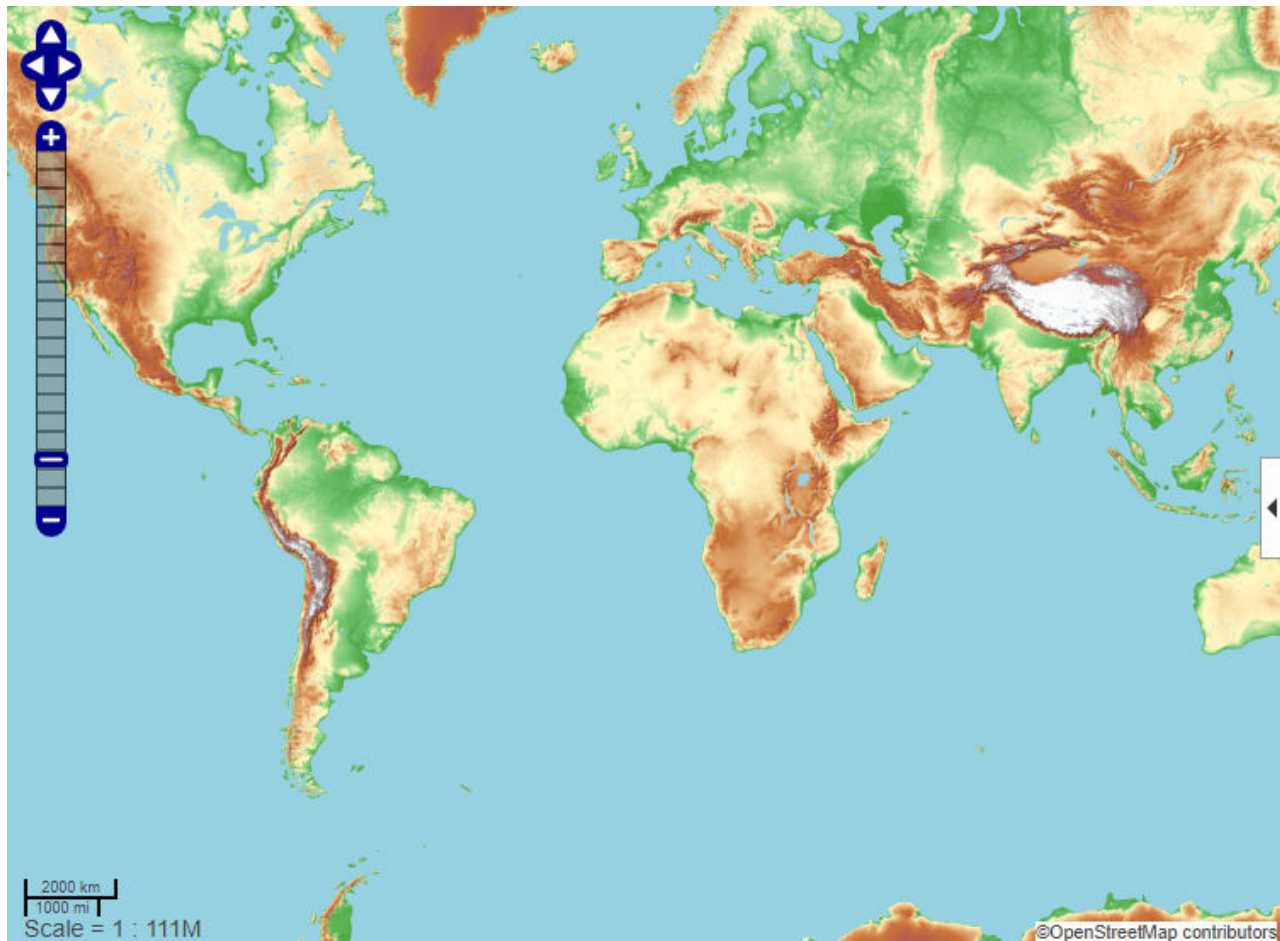
Input Arguments

baseLayer — Map displayed in web map
character vector | string scalar

Map displayed in the web map, specified as a string scalar or character vector, listed in the following table. If specified as string scalar or character vector, the value is case insensitive and spaces are optional.

Name	Description
'World Street Map'	Worldwide street map provided by Esri. For information about the Esri ArcGIS Online layers, visit https://www.arcgis.com/home/gallery.html#c=esri&f=basemaps&t=maps.
'Open Street Map'	Street map from openstreetmap.org. For more information, visit https://www.openstreetmap.org.
'World Imagery'	Worldwide imagery provided by Esri.
'World Topographic Map'	Topographic map for the world from Esri.
'World Shaded Relief'	Surface elevation as shaded relief provided by Esri

Name	Description
'World Physical Map'	Natural Earth map of the world provided by Esri
'World Terrain Base'	Shaded relief and bathymetry provided by Esri
'USGS Imagery'	Composite of Blue Marble, NAIP, and Landsat provided by the USGS.
'USGS Topographic Imagery'	Topographic map with imagery provided by the USGS.
'USGS Shaded Topographic Map'	Composite of contours, shaded relief, and vector layers provided by the USGS.
'USGS Shaded Relief'	Shaded relief from National Elevation Dataset provided by the USGS.
'National Geographic Map'	General reference map provided by Esri
'DeLorme World Basemap'	Topographic map provided by Esri
'Ocean Basemap'	Bathymetry, marine features, depth in meter provided by Esri
'World Navigation Charts'	Topographic data with nautical information provided by Esri
'Light Gray Canvas Map'	Neutral background map with minimal colors provided by Esri

Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

Data Types: char | string

wmsLayer — Web map service layer
WMSLayer array

Web map service layer, specified as a WMSLayer array.

customBasemap — Name of custom basemap
character vector | string

Name of custom basemap, specified as a string scalar or character vector. The value is case-insensitive and spaces are optional.

Data Types: char | string

wm — Web map
web map handle

Web map, specified as a web map handle, returned by the webmap function.

Output Arguments

wm — Web map
web map handle

Web map, returned as a web map handle.

Limitations

In MATLAB Online, if you refresh the URL, then the window containing the web map closes. In addition, if you returned a handle to the web map, then it is deleted.

More About

Web Map

An interactive map accessed through a web page. In a web map, you can select different map layers to view and navigate around the map using interactive tools, such as zooming. The web map is a window that displays map base layers obtained from web servers on the Internet. You can also display overlay layers that contain custom point and line vector data.

Tips

Particular maps may not support every available zoom level. If your map displays as white, try another zoom level. The map you are displaying might not support the zoom level you have currently selected. You can also select another base layer, which might support the specified zoom level.

Compatibility Considerations

Web maps appear in multiple windows and have a default size
Behavior changed in R2021a

Starting in R2021a, web maps created using the webmap function have these differences from previous releases:

- Each web map appears in a separate window. In previous releases, each web map appeared as a tab in a single window.
- Web maps have a default size of 750-by-550 pixels. In previous releases, each new web map matched the size of the previous web map.
- You cannot dock a web map.

Compiling web maps using Linux requires files in directory of application
Behavior changed in R2020a

Starting in R2020a, to compile web maps using MATLAB Compiler on Linux, you must copy these files to the application directory and distribute them with the application.

- icudtl.dat
- natives_blob.bin
- snapshot_blob.bin

You can find the path to these files using the command
fullfile(matlabroot,'bin','glnxa64').

See Also
wmclose | wmcenter | wmline | wmlimits | wmprint | wmmarker | wmremove | wmzoom | addCustomBasemap

Introduced in R2013b

WebMapServer

Web map server

Description

A `WebMapServer` object represents a Web Map Service (WMS) and acts as a proxy to a WMS server.

The `WebMapServer` object resides physically on the client side. The object can access the capabilities document on the WMS server and perform requests to obtain maps. It supports multiple WMS versions and negotiates with the server automatically to use the highest known version that the server can support.

Creation

Syntax

`server = WebMapServer(serverURL)`

Description

`server = WebMapServer(serverURL)` creates a `WebMapServer` object, setting the `ServerURL` property.

Properties

Timeout — Number of milliseconds before a server times out
0 (default) | nonnegative integer

Number of milliseconds before a server times out, specified as a nonnegative integer. When `Timeout` has a value of 0, the `WebMapServer` object ignores the timeout mechanism.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

EnableCache — Flag enabling cache
1 (default) | logical scalar

Flag enabling cache, specified as a logical scalar. When `EnableCache` is `True`, the `WebMapServer` object caches the `WMSCapabilities` object, which is returned when you use the `getCapabilities` method. The cache expires if the `AccessDate` property of the cached `WMSCapabilities` object is not the current day.

Data Types: `logical`

ServerURL — URL of the server
character vector

URL of the server, specified as a character vector. The URL must include the protocol `'http://'` or `'https://'`. The URL can contain additional WMS keywords.

Data Types: `char`

RequestURL — URL of the last request to the server
character vector

URL of the last request to the server, specified as a character vector. `RequestURL` specifies a request for either the XML capabilities document or a map. You can insert the requested URL into a browser.

Data Types: `char`

Object Functions

getCapabilities   Get capabilities document from server
getMap   Get raster map from server
updateLayers   Update layer properties

Examples

Get Capabilities Document from Web Map Server

Construct a `WebMapServer` object that communicates with one of the Environmental Research Division's Data Access Program (ERDDAP) WMS servers hosted by NOAA. Obtain the capabilities document from the server.

```
layers = wmsfind('jplg1sst','SearchField','serverurl');
serverURL = layers(1).ServerURL;
server = WebMapServer(serverURL);
capabilities = getCapabilities(server);
layers = capabilities.Layer;
```

Search for a server that shows nation boundaries (`nations`). Search for a server that provides daily, global sea surface temperature (`sst`) data produced by the Jet Propulsion Laboratory's Regional Ocean Modeling System (JPL ROMS) group.

```
nations = refine(layers,'nations');
nations = nations(1);
sst = refine(layers,'sst');
sst = sst(1);
layer = [sst nations];
```

Obtain data from the servers.

```
request = WMSMapRequest(layer,server);
A = getMap(server,request.RequestURL);
R = request.RasterReference;
```

Display data from the servers.

```
figure
worldmap(A,R)
geoshow(A,R)
title({sst.LayerTitle(1:51),sst.LayerTitle(52:end), ...
      sst.Details.Dimension.Default})
```

GHRSST Global 1-km Sea Surface Temperature (G1SST),
Global, 0.01 Degree, 2010-2017, Daily - SST
2017-09-13T00:00:00Z
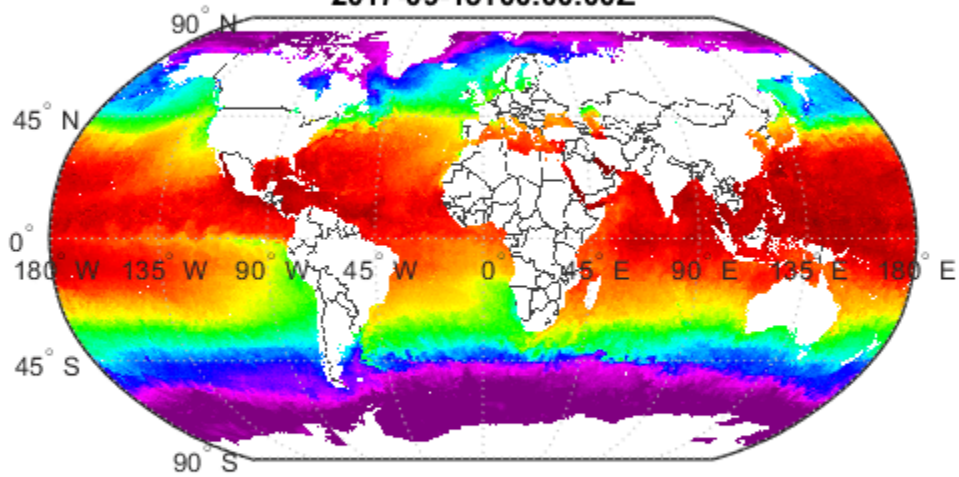
See Also

Functions
wmsfind | wmsinfo | wmsread | wmsupdate

Objects
WMSLayer | WMSCapabilities | WMSMapRequest

Introduced before R2006a

westof

Wrap longitudes to values west of specified meridian

Compatibility

Note The `westof` function is obsolete and will be removed in a future release of the toolbox. Replace it with the following calls, which are also more efficient:

```
westof(lon,meridian,'degrees') ==> meridian-mod(meridian-lon,360)
```

```
westof(lon,meridian,'radians') ==> meridian-mod(meridian-lon,2*pi)
```

Syntax

```
lonWrapped = westof(lon,meridian)
lonWrapped = westof(lon,meridian,angleunits)
```

Description

`lonWrapped = westof(lon,meridian)` wraps angles in `lon` to values in the interval `(meridian-360 meridian]`. `lon` is a scalar longitude or vector of longitude values. All inputs and outputs are in degrees.

`lonWrapped = westof(lon,meridian,angleunits)` where `angleunits` specifies the input and output units as either `'degrees'` or `'radians'`. It may be abbreviated and is case-insensitive. If *angleunits* is `'radians'`, the input is wrapped to the interval `(meridian-2*pi meridian]`.

Introduced before R2006a

wgs84Ellipsoid

Reference ellipsoid for World Geodetic System 1984

Syntax

```
E = wgs84Ellipsoid
E = wgs84Ellipsoid(lengthUnit)
```

Description

`E = wgs84Ellipsoid` returns a `referenceEllipsoid` object representing the World Geodetic System of 1984 (WGS 84) reference ellipsoid. The semimajor axis and semiminor axis are expressed in meters.

`E = wgs84Ellipsoid(lengthUnit)` returns a WGS 84 reference ellipsoid object in which the semimajor axis and semiminor axis are expressed in the specified unit, `lengthUnit`.

Input Arguments

`lengthUnit` — Unit of measure
`'meter'` (default) | string scalar | character vector

Unit of measure, specified as a string scalar or character vector. You can specify any length unit accepted by the `validateLengthUnit` function.

Data Types: `char` | `string`

Output Arguments

E — WGS84 reference ellipsoid
`referenceEllipsoid` object

WGS84 reference ellipsoid, returned as a `referenceEllipsoid` object.

Examples

Create wgs84 Reference Ellipsoid and View Derived Property

Create a World Geodetic System of 1984 (wgs84) reference ellipsoid and view one of its derived properties.

```
wgs84InMeters = wgs84Ellipsoid

wgs84InMeters =

referenceEllipsoid with defining properties:

            Code: 7030
            Name: 'World Geodetic System 1984'
```

```
        LengthUnit: 'meter'
     SemimajorAxis: 6378137
     SemiminorAxis: 6356752.31424518
  InverseFlattening: 298.257223563
      Eccentricity: 0.0818191908426215

and additional properties:

  Flattening
  ThirdFlattening
  MeanRadius
  SurfaceArea
  Volume
```

View value of `SurfaceArea` property.

`wgs84InMeters.SurfaceArea`

```
ans =

   5.1007e+14
```

Create wgs84 Reference Ellipsoid Specifying Units

Create a World Geodetic System of 1984 (wgs84) reference ellipsoid, specifying the unit of measure. In the summary information returned, note the value of the `LengthUnit` field.

`wgs84InKilometers = wgs84Ellipsoid('km')`

```
wgs84InKilometers =

referenceEllipsoid with defining properties:

              Code: 7030
              Name: 'World Geodetic System 1984'
        LengthUnit: 'kilometer'
     SemimajorAxis: 6378.137
     SemiminorAxis: 6356.75231424518
  InverseFlattening: 298.257223563
      Eccentricity: 0.0818191908426215

and additional properties:

  Flattening
  ThirdFlattening
  MeanRadius
  SurfaceArea
  Volume
```

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Setting the length unit of measure is not supported.

See Also
referenceEllipsoid | validateLengthUnit

Introduced in R2012a

wktstring

Well-known text string

Syntax

```
str = wktstring(crs)
str = wktstring(crs,Name,Value)
```

Description

`str = wktstring(crs)` returns the well-known text (WKT) string representation of the specified projected or geographic coordinate reference system. By default, `wktstring` uses the WKT 2 standard and does not apply formatting.

`str = wktstring(crs,Name,Value)` specifies version and formatting options using one or more `Name,Value` pair arguments. For example, `'Format','formatted'` includes line breaks and indentations in the WKT string.

Examples

Get WKT of Projected CRS

Return information about projected data as a `RasterInfo` object. Find the projected CRS for the data by accessing its `CoordinateReferenceSystem` property.

```
info = georasterinfo('MtWashington-ft.grd');
p = info.CoordinateReferenceSystem;
```

Return the WKT as a string.

```
str = wktstring(p)
```

```
str =
"PROJCRS["UTM Zone 19, Northern Hemisphere",BASEGEOGCRS["NAD27",DATUM["North American Datum 1927"
```

Get WKT of Geographic CRS

Return information about geographic data as a `RasterInfo` object. Find the geographic CRS for the data by accessing its `CoordinateReferenceSystem` property.

```
[Z,R] = readgeoraster('n39_w106_3arc_v2.dt1');
g = R.GeographicCRS;
```

Return the WKT as a string.

```
wkt = wktstring(g)
```

```
wkt =
"GEOGCRS["WGS 84",DATUM["World Geodetic System 1984",ELLIPSOID["WGS 84",6378137,298.257223563,LEI
```

Change Default Formatting

Return information about a data set as a `RasterInfo` object. Find the projected CRS for the data by accessing the `CoordinateReferenceSystem` property.

```
info = georasterinfo('MtWashington-ft.grd');
p = info.CoordinateReferenceSystem;
```

Return the WKT as a formatted string by using the `'Format'` name-value pair.

```
str = wktstring(p,'Format','formatted')

str =
    "PROJCRS["UTM Zone 19, Northern Hemisphere",
        BASEGEOGCRS["NAD27",
            DATUM["North American Datum 1927",
                ELLIPSOID["Clarke_1866",6378206.4,294.978698213898,
                    LENGTHUNIT["metre",1]],
                ID["EPSG",6267]],
            PRIMEM["Greenwich",0,
                ANGLEUNIT["Degree",0.0174532925199433]]],
        CONVERSION["UTM zone 19N",
            METHOD["Transverse Mercator",
                ID["EPSG",9807]],
            PARAMETER["Latitude of natural origin",0,
                ANGLEUNIT["Degree",0.0174532925199433],
                ID["EPSG",8801]],
            PARAMETER["Longitude of natural origin",-69,
                ANGLEUNIT["Degree",0.0174532925199433],
                ID["EPSG",8802]],
            PARAMETER["Scale factor at natural origin",0.9996,
                SCALEUNIT["unity",1],
                ID["EPSG",8805]],
            PARAMETER["False easting",500000,
                LENGTHUNIT["Meter",1],
                ID["EPSG",8806]],
            PARAMETER["False northing",0,
                LENGTHUNIT["Meter",1],
                ID["EPSG",8807]],
            ID["EPSG",16019]],
        CS[Cartesian,2],
            AXIS["easting",east,
                ORDER[1],
                LENGTHUNIT["Meter",1]],
            AXIS["northing",north,
                ORDER[2],
                LENGTHUNIT["Meter",1]]]"
```

Export WKT as Projection File

Return information about a data set as a `RasterInfo` object. Find the projected CRS for the data by accessing the `CoordinateReferenceSystem` property. Return the WKT as a string using the WKT 1 standard.

```
info = georasterinfo('MtWashington-ft.grd');
p = info.CoordinateReferenceSystem;
str = wktstring(p,'Version','wkt1');
```

Export the WKT as a projection file called `mtwash.prj`.

```
writematrix(str,'mtwash.prj','FileType','text','QuoteStrings',false)
```

Input Arguments

`crs` — Coordinate reference system
`projcrs` object | `geodcrs` object

Coordinate reference system, specified as a `projcrs` object or `geocrs` object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Version','wkt1'` specifies the WKT 1 standard for the output well-known text string.

`Format` — Format of WKT
`'compact'` (default) | `'formatted'`

Format of the WKT string, specified as the comma-separated pair consisting of `'Format'` and one of these values:

- `'compact'` – Do not include formatting.
- `'formatted'` – Include line breaks and indentations.

Example: `'Format','formatted'`

Data Types: `char` | `string`

`Version` — WKT version
`'wkt2'` (default) | `'wkt1'`

WKT version, specified as the comma-separated pair consisting of `'Version'` and one of these values:

- `'wkt2'` – Use the WKT 2 standard. For more information about this standard, see Geographic Information – Well-known text representation of coordinate reference systems.
- `'wkt1'` – Use the WKT 1 standard. For more information about this standard, see Well-Known Text format.

Example: `'Version','wkt1'`

Data Types: `char` | `string`

Tips

The WKT strings for two equivalent `projcrs` or `geocrs` objects may not be equivalent. Compare `projcrs` or `geocrs` objects by using the `isequal` function instead of by comparing WKT strings.

See Also

Objects
`projcrs` | `geocrs`

Introduced in R2020b

wmclose

Close web map

Syntax

```
wmclose
wmclose(wm)
wmclose all
```

Description

`wmclose` closes the current web map.

`wmclose(wm)` closes the web map specified by `wm`.

`wmclose all` closes all web maps.

Examples

Close Current Web Map

Open a web map, pause one second, and then close the web map.

```
webmap
pause(1)
wmclose
```

Close Specified Web Map

Open two web maps, pause for one second, and then close one of the web maps.

```
h1 = webmap;
h2 = webmap('ocean basemap');
pause(1)
wmclose(h1)
```

Close All Web Maps

Open two web maps, pause for one second, and then close all the web maps.

```
h1 = webmap;
h2 = webmap('ocean basemap');
```

```
pause(1)
wmclose all
```

Input Arguments

wm — Web map
web map handle

Web map, specified as a web map handle.[7] You use the `webmap` function to get a web map handle when you create a web map.

See Also

webmap | wmcenter | wmline | wmlimits | wmprint | wmmarker | wmremove | wmzoom

Introduced in R2013b

---

7. Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

wmprint

Print web map

Syntax

```
wmprint()
wmprint(wm)
```

Description

wmprint() prints the contents of the current web map to a printer.

wmprint(wm) prints the contents of the web map specified by wm.

Examples

Print Web Map

Create a web map, specifying a base layer.

```
webmap('OpenStreetMap')
```

Position the web map.

```
wmcenter(51.487, 0, 15)
```

Print the contents of the web map.

```
wmprint()
```

Input Arguments

wm — Web map
scalar web map handle

Web map, specified as a scalar web map handle.[8]

Limitations

MATLAB Online does not support the wmprint function.

See Also

wmclose | wmcenter | wmline | wmlimits | webmap | wmmarker | wmremove | wmzoom

---

8. Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

Introduced in R2013b

wmmarker

Display geographic marker on web map

Syntax

```
wmmarker(lat,lon)
wmmarker(P)
wmmarker(wm,___)
wmmarker(___,Name,Value)
h = wmmarker(___)
```

Description

wmmarker(lat,lon) displays a marker overlay at the points specified by lat and lon on the current web map. If there is no current web map, wmmarker creates one. wmmarker centers the map so that all vector overlays on the web map are visible. A marker is also called a map pin.

wmmarker(P) displays a marker overlay based on the vector geographic features stored in P. Each element of P defines one marker overlay.

wmmarker(wm, ___) displays the overlay in the web map specified by the web map handle, wm.

wmmarker(___,Name,Value) specifies name-value pairs that set additional display properties. Parameter names can be abbreviated and are case-insensitive.

h = wmmarker(___) returns a handle to the overlay.

Examples

Display Marker on Web Map

Display a marker at the location of London, England. There is no current web map, so the wmmarker function creates one.

```
lat = 51.5187666404504;
lon = -0.130003487285315;
wmmarker(lat,lon)
```
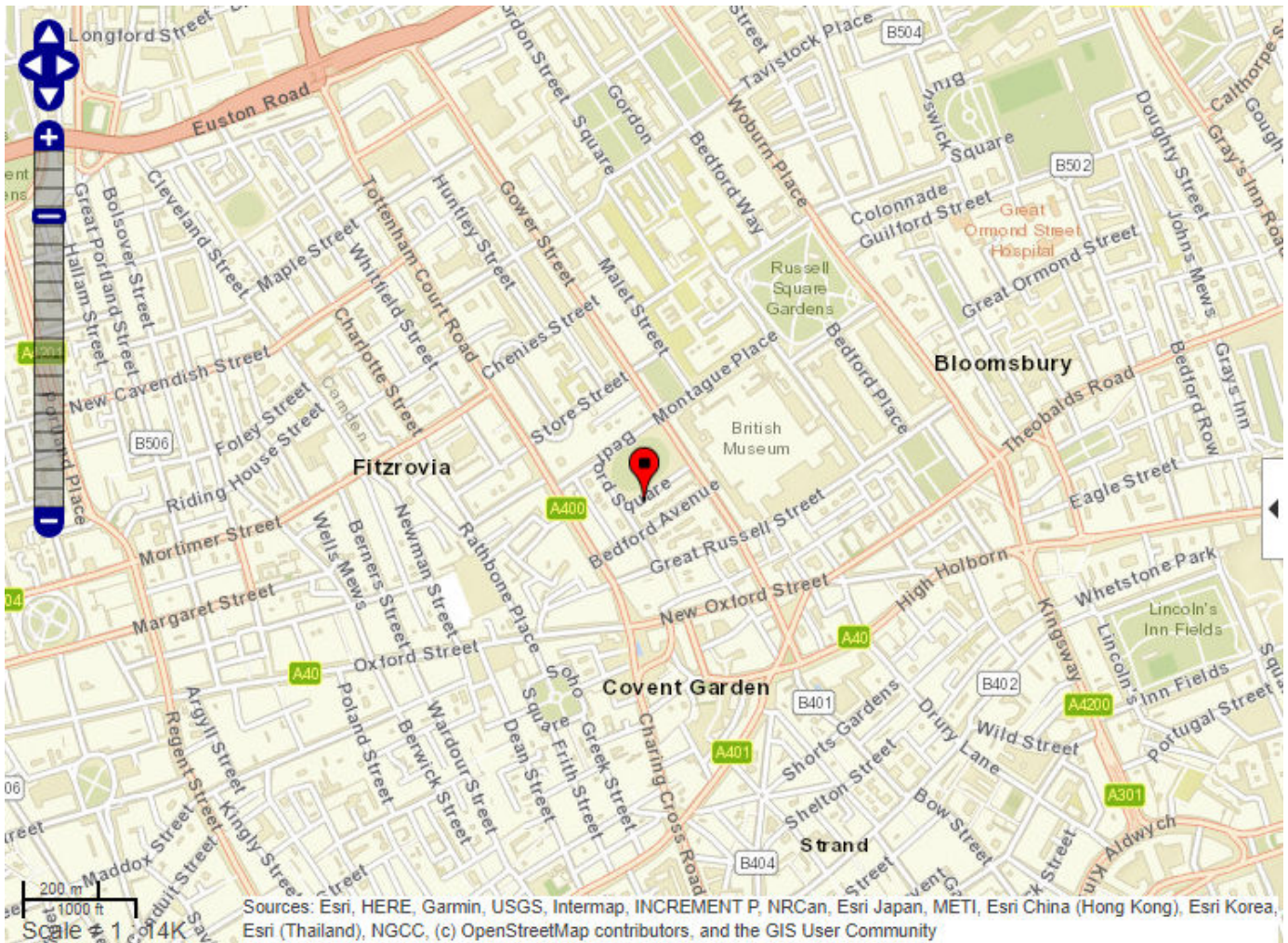
`wmmarker` adds the marker name to the list of overlays in the Layer Manager. The default name is
Marker Overlay 1.
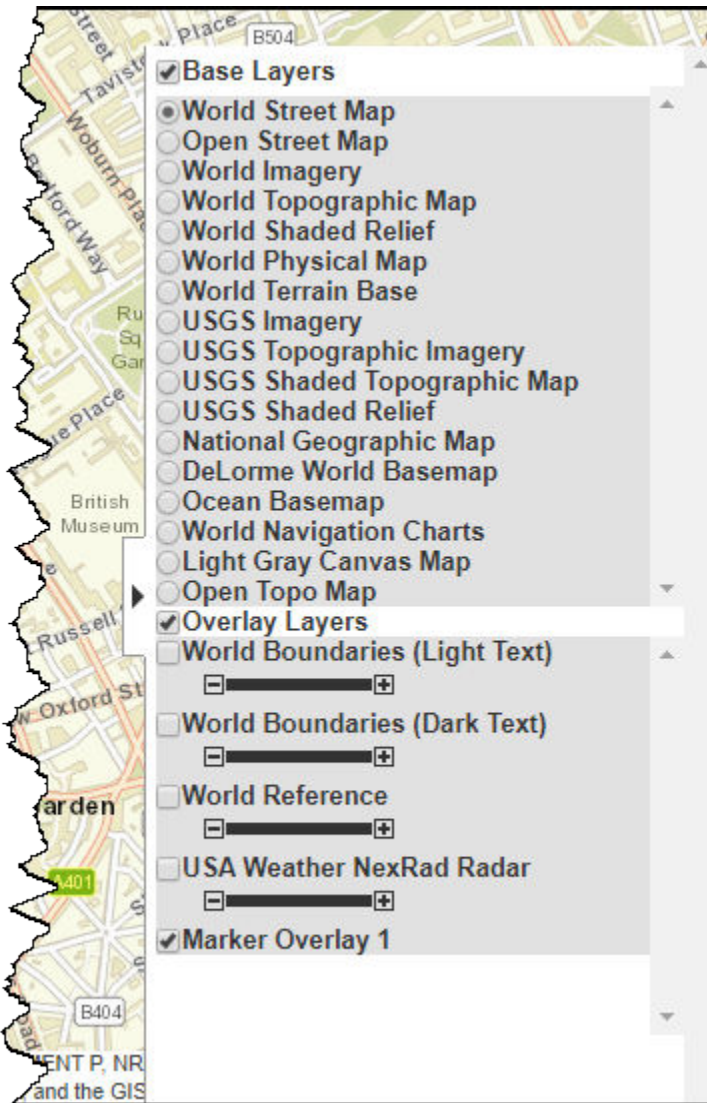
Display Information About Markers

Read the coordinates and names of locations in Boston, MA, as a geospatial table.

```
P = readgeotable("boston_placenames.gpx");
```

Display the data as markers on a web map. Specify the feature names and overlay names using name-value arguments. The `wmmarker` function creates a new web map when there is no current web map.

```
wmmarker(P,"FeatureName",P.Name,"OverlayName","Boston Placenames")
```

Click on a marker to see information about the feature, including its name.
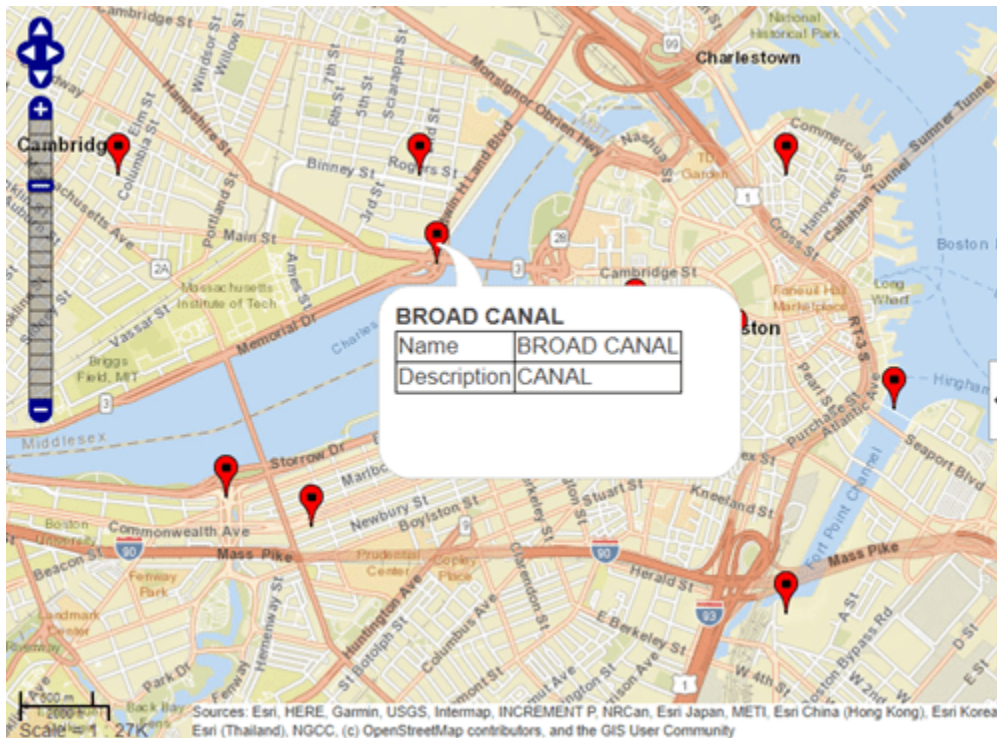
Display Marker Using Custom Icon and Description Data

Define a location. For this example, specify the coordinates of MathWorks.

```
lat =  42.299827;
lon = -71.350273;
```

Specify a name and text to display in the description balloon. This code makes the MathWorks URL a link.

```
name = 'MathWorks';
description = sprintf(...
  '%s<br>%s</br><br>%s</br>',...
  '3 Apple Hill Drive', 'Natick, MA. 01760', ...
  '<a href="https://www.mathworks.com" target="_blank">https://www.mathworks.com</a>');
```

Specify a custom icon for the marker.

```
iconDir = fullfile(matlabroot,'toolbox','matlab','icons');
iconFilename = fullfile(iconDir,'matlabicon.gif');
```

Display the marker on the web map by using wmmarker. Specify the Description, FeatureName, Icon, and OverlayName name-value pairs. Note the custom icon. Display the text you included by clicking on the marker. Note the HTML formatting in the description.

```
wmmarker(lat,lon,'Description',description, ...
                 'FeatureName',name, ...
                 'Icon',iconFilename, ...
                 'OverlayName',name)
```

Display Marker Overlay Using Attribute Specification

Import a shapefile representing tsunami (tidal wave) events reported over several decades, tagged geographically by source location.

```
GT = readgeotable("tsunamis.shp","CoordinateSystemType","geographic");
```

Create an attribute specification for use in the feature balloons. Modify the specification so that it defines a table of values, including year, cause, country, location, and maximum height. The attribute specification defines the format of the expected value for each field.

```
attribspec = makeattribspec(GT);

desiredAttributes = ...
        {'Max_Height','Cause','Year','Location','Country'};
allAttributes = fieldnames(attribspec);
attributes = setdiff(allAttributes,desiredAttributes);
attribspec = rmfield(attribspec,attributes);
attribspec.Max_Height.AttributeLabel = "<b>Maximum Height</b>";
attribspec.Max_Height.Format = "%.1f Meters";
```

```
attribspec.Cause.AttributeLabel = "<b>Cause</b>";
attribspec.Year.AttributeLabel = "<b>Year</b>";
attribspec.Year.Format = "%.0f";
attribspec.Location.AttributeLabel = "<b>Location</b>";
attribspec.Country.AttributeLabel = "<b>Country</b>";
```

Create a web map with a base layer containing ocean details. Add the marker overlay. Display the table data you specified in the attribute specification by selecting a marker.

```
webmap("ocean basemap");
wmmarker(GT,"Description",attribspec,...
        "OverlayName","Tsunami Events")
wmzoom(2)
```


Input Arguments

lat — Latitudes of points
matrix

Latitudes of points, specified as a matrix.

Data Types: `single` | `double`

lon — Longitudes of points
matrix

Longitudes of points, specified as a matrix.

Data Types: `single` | `double`

P — Geographic features
geospatial table | `geopoint` vector

Geographic features, specified as one of the following:

*   A geospatial table containing `geopointshape` objects. For more information about geospatial tables, see "Create Geospatial Tables".

*   A `geopoint` vector.

`wm` — Web map
web map handle

Web map, specified as a web map handle.[9]

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `wmmarker(lat,lon,'Autofit',true)`

`Autofit` — Overlay visibility
true (default) | false

Overlay visibility, specified as the comma-separated pair consisting of `'Autofit'` and the logical flag `true` or `false`, or the numeric value 1 or 0. If `true` or 1, `wmmarker` adjusts the spatial extent of the map to ensure that all the vector overlays on the map are visible. If `false`, `wmmarker` does not adjust the spatial extent when the overlay is added to the map.

Overlay visibility, specified as a scalar logical or numeric value `true` (1) or `false` (0).

*   If `true`, `wmmarker` adjusts the spatial extent of the map to ensure that all the vector overlays on the map are visible.

*   If `false`, `wmmarker` does not adjust the spatial extent when the overlay is added to the map.

Data Types: `double` | `logical`

`Description` — Description of feature
empty character vector (`'  '`) (default) | character vector | cell array of character vectors | scalar structure

Description of feature, specified as the comma-separated pair consisting of `'Description'` and a character vector, cell array of character vectors, or scalar structure. The description defines the content that `wmmarker` displays in the feature's description balloon which appears when a user clicks on the feature in the web map. Description elements can be either plain text or HTML markup. When you specify an attribute spec, the display in the balloon for the attribute fields of `P` are modified according to the specification. The default value is an empty character vector (`''`). If the value is a structure, the attribute spec is applied to the attributes of each feature of `P` and ignored with `lat` and `lon` input.

*   If the value is a cell array it is either scalar or the same length as `P`, or `lat` and `lon`, and specifies the description for each marker.

---

9.   Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

- If the value is a structure, the attribute spec is applied to the attributes of each feature of `P` and ignored with `lat` and `lon` input.

Data Types: `char` | `struct` | `cell`

OverlayName — Name of overlay layer
`'Marker Overlay N'`, (default) | character vector

Name of overlay layer, specified as the comma-separated pair consisting of `'OverlayName'` and a character vector. `wmmarker` inserts the name in the Layer Manager under the Overlays item. The Layer Manager is the tool that appears on the right side of the web map frame. The default name is `'Marker Overlay N'` where `N` is the number assigned to this overlay.

Data Types: `char`

FeatureName — Name of feature
`'OverlayName: Point K'` (default) | character vector | cell array of character vectors

Name of feature, specified as the comma-separated pair consisting of `'FeatureName'` and a character vector or cell array of character vectors. The name appears in the feature's balloon when a user clicks on the feature in the web map. The default value is `'OverlayName : Point K'`, where *OverlayName* is the name of the overlay and *K* is the number assigned to a particular point. If the value is a character vector, `wmmarker` applies it to all features. If the value is a cell array, it must be a scalar or an array with the same length as `P` or `lat` and `lon`.

Data Types: `char` | `cell`

Icon — File name of custom icon for a marker
character vector | cell array of character vectors

File name of custom icon for a marker, specified as the comma-separated pair consisting of `'Icon'` and a character vector or cell array of character vectors. If the icon file name is not in the current folder, or in a folder on the MATLAB path, specify a full or relative path name. If you specify an Internet URL it must include the protocol type. If the icon file name is not specified, the default icon is displayed. For best results when you want to view a non-default icon, specify a PNG file containing image data with an alpha mask.

- If the value is a character vector, `wmmarker` applies the value to all markers.
- If you specify a cell array, it must be the same length as `P`, or `lat` and `lon`, and specifies the icon for each marker.

Data Types: `char` | `cell`

IconScale — Scaling factor for icon
1 (default) | positive numeric scalar or vector.

Scaling factor for icon, specified as the comma-separated pair consisting of `'IconScale'` and a positive numeric scalar or vector.

- If the value is a scalar, the value is applied to all icons.
- If the value is a vector, it must specify a value for each icon, and it must be the same length as `lat` and `lon` or `P`.

Data Types: `double`

Color — Icon color
'red' (default) | color name | RGB triplet | cell array of color names | string vector of color names | matrix of RGB triplets

Icon color, specified as the comma-separated pair consisting of 'Color' and one of these options.

- A color name such as 'red' or a short name such as 'r'.
- An RGB triplet, which is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range [0,1]; for example, [0.4 0.6 0.7].
- A cell array of color names such as {'red','green','blue'} or {'r','g','b'}.
- A string vector of color names such as ["red" "green" "blue"] or ["r" "g" "b"].
- A matrix of RGB triplets, which is a three-column matrix in which each row is an RGB triplet.

The way you specify the color depends on the desired color scheme.

- To apply the same color to all icons, specify a single color name or RGB triplet.
- To apply a different color to each icon, specify a cell array of color names, a string vector of color names, or a matrix of RGB triplets. The number of colors and RGB triplets must match the length of lat and lon or P.

If you specify a custom icon file, then the wmmarker function ignores this argument.

This table contains the color names and equivalent RGB triplets for some common colors.

Color Name	Short Name	RGB Triplet	Appearance
'red'	'r'	[1 0 0]	
'green'	'g'	[0 1 0]	
'blue'	'b'	[0 0 1]	
'cyan'	'c'	[0 1 1]	
'magenta'	'm'	[1 0 1]	
'yellow'	'y'	[1 1 0]	
'black'	'k'	[0 0 0]	
'white'	'w'	[1 1 1]	

Data Types: char | string | cell | double

Alpha — Transparency of marker
1 (default) | numeric scalar or vector

Transparency of marker, specified as the comma-separated pair consisting of 'Alpha' and a numeric scalar or vector. If you specify a vector, it must include a value for each marker, that is, the vector must be the same length as P. The default value, 1, means that the marker is fully opaque.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64

Output Arguments

h — Marker overlay
handle to a marker overlay object

Marker overlay, returned as a handle to a marker overlay object.

See Also
wmclose | wmpolygon | wmcenter | wmline | wmlimits | wmprint | webmap | wmremove | wmzoom

Introduced in R2013b

wmline

Display geographic line on web map

Syntax

```
wmline(lat,lon)
wmline(P)
wmline(wm, ___ )
wmline( ___ ,Name,Value)
h = wmline( ___ )
```

Description

wmline(lat,lon) displays a line overlay defined by the vertices in lat,lon on the current web map. If there is no current web map, wmline creates one. wmline centers the map so that all vector overlays displayed on the web map are visible.

wmline(P) displays a line overlay based on the vector geographic features stored in P.

wmline(wm, ___) displays the line overlay on the web map specified by the web map handle, wm.

wmline(___ ,Name,Value) specifies name-value pairs that set additional display properties.

h = wmline(___) returns a handle to line overlay.

Examples

Display Line on Web Map

Import a shapefile containing the coordinates of world cities as a geospatial table. Create a subtable containing the rows for London and Paris.

```
cities = readgeotable("worldcities.shp");
rows = cities.Name == "London" | cities.Name == "Paris";
londonParis = cities(rows,:);
```

Display a line from London to Paris on a web map.

```
wmline(londonParis)
```

Display Reduced Line on Web Map

Large data sets can sometimes be slow to display, making the web map appear to hang. To work around this issue, reduce the size of the data set using the `reducem` function before calling `wmline`.

Load vector data representing the coordinates of coastlines.

```
load coastlines
```

Reduce the number of points in the latitude and longitude vectors using the `reducem` function.

```
[lat,lon] = reducem(coastlat,coastlon);
```

Create a web map that does not wrap because the data is of global extent.

```
webmap('ocean basemap','WrapAround',false)
```
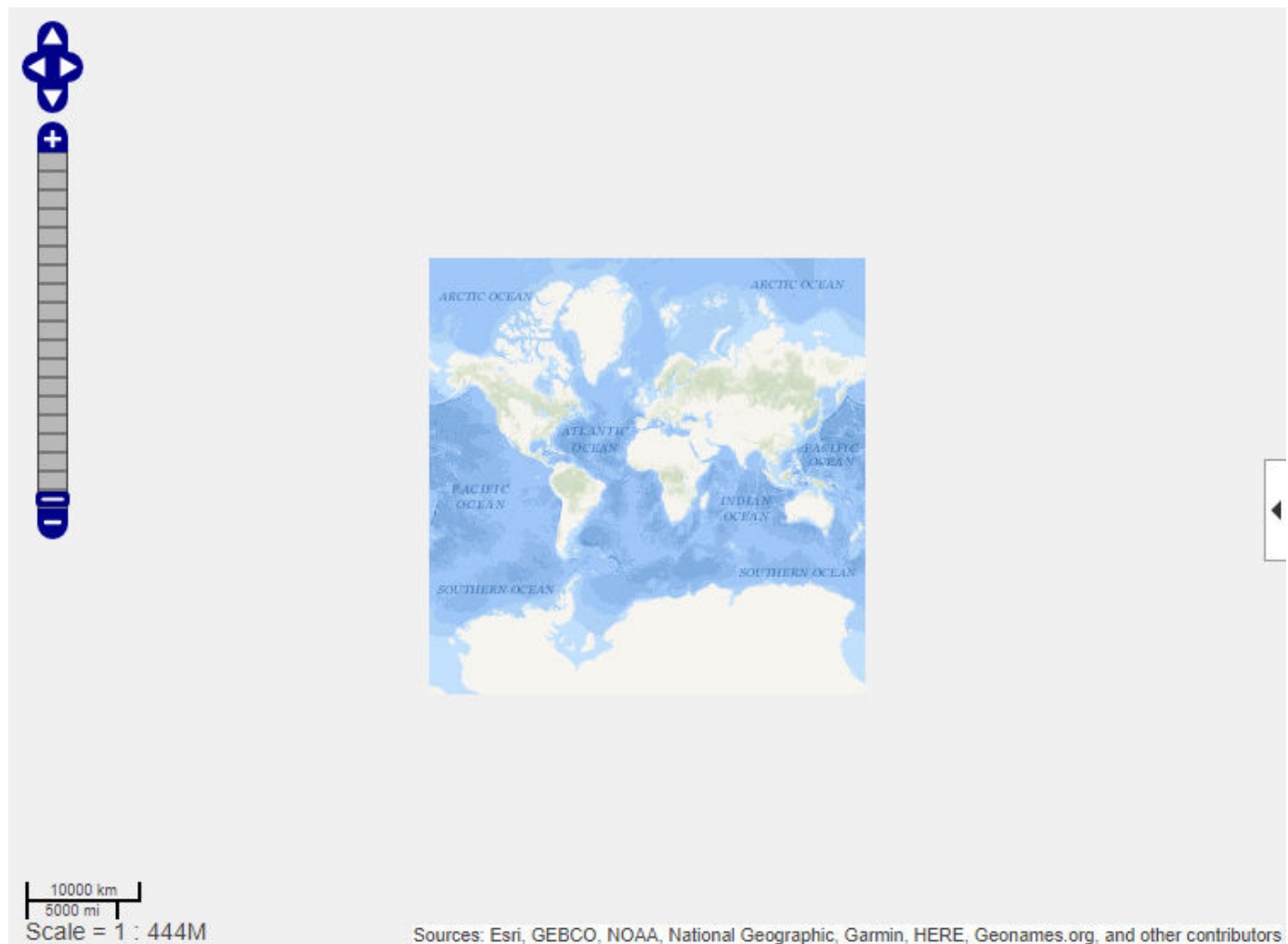
Display the coastlines on the web map. The figure shows the description balloon that appears when you click on the line. Name the feature using the `'FeatureName'` name-value pair.

```
wmline(lat,lon,'LineWidth',3,'FeatureName','coastline')
wmzoom(3)
```
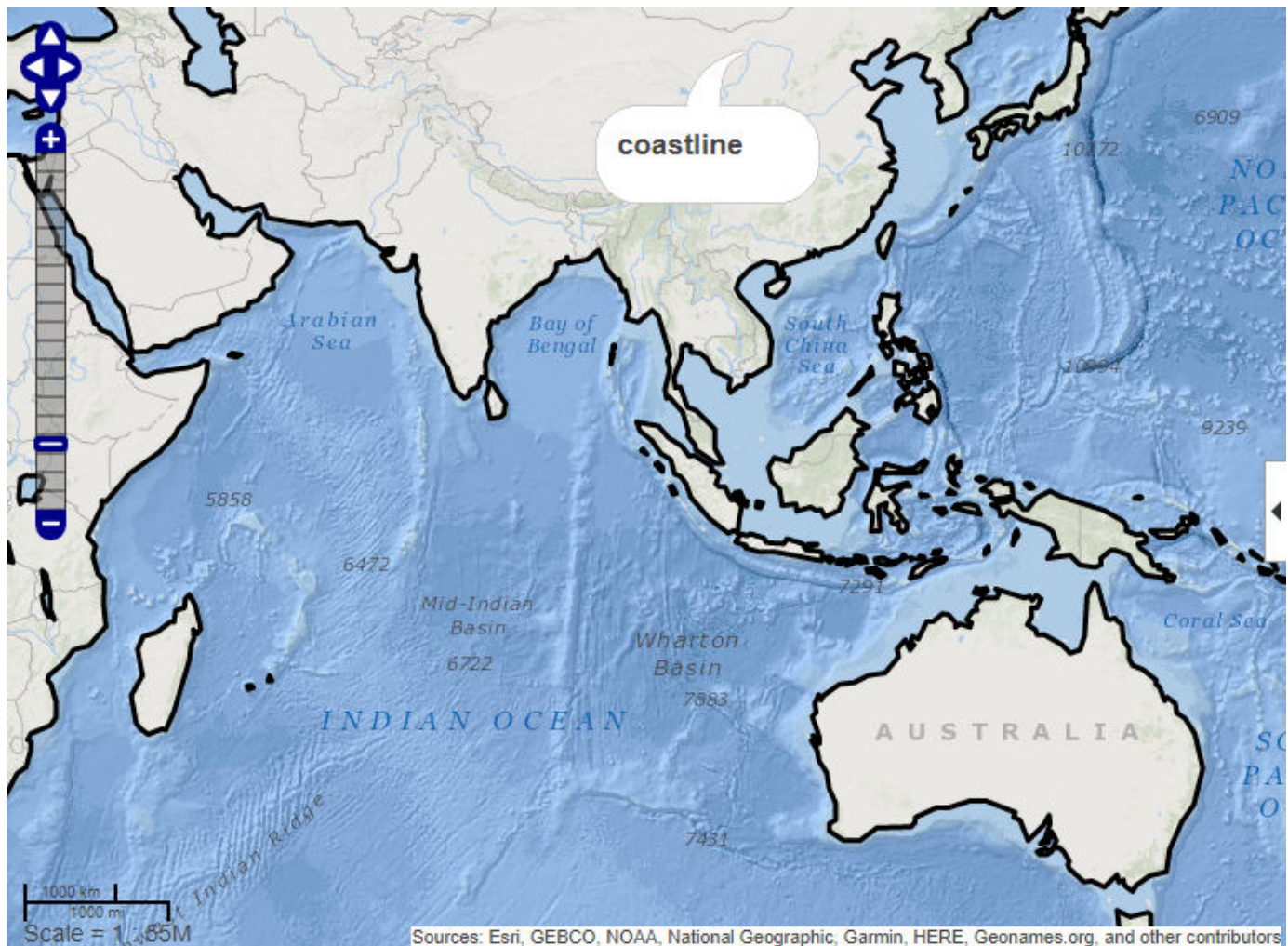
Display Circles on Web Map

Define the latitude and longitude of the feature. This example shows how to display lines on a web map that represent range data for an airport approach pattern.

```
lat0 = 51.50487;
lon0 = 0.05235;
```

Create a web map and specify a base layer.

```
webmap('OpenStreetMap')
```

Compute a small circle with a 1000 meter radius. Setting the `az` parameter to an empty matrix causes `scircle1` to compute a complete circle.

```
radius = 1000;
az = [];
e = wgs84Ellipsoid;
[lat,lon] = scircle1(lat0,lon0,radius,az,e);
```

Display a red circle with 1000 meter radius, using the latitude and longitude values returned by `scircle1` in the previous step.

```
wmline(lat,lon,'Color','red','OverlayName','1000 Meters')
```
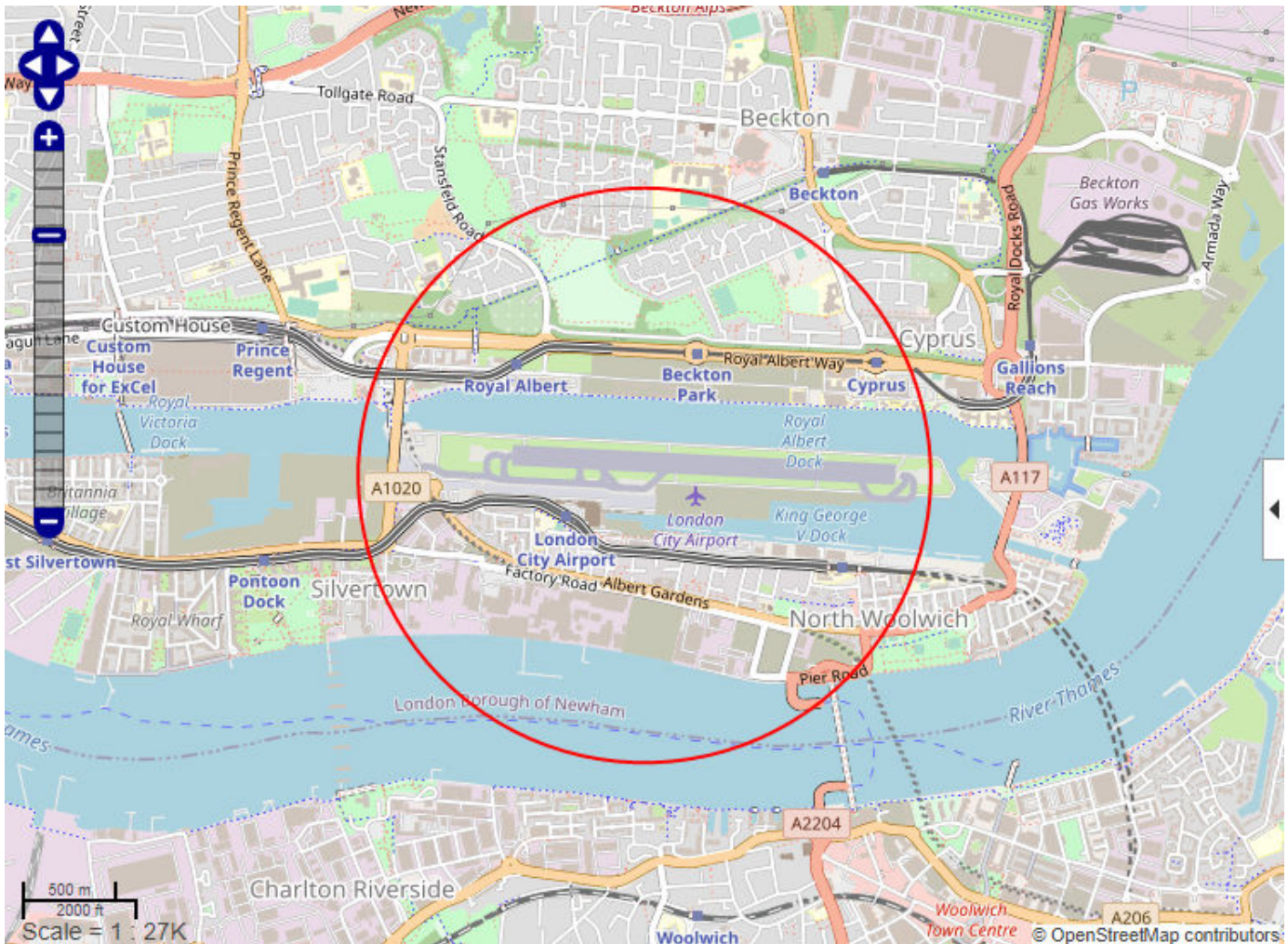
Compute another small circle, this time with a 2000 meter radius.

```
radius = 2000;
[lat,lon] = scircle1(lat0,lon0,radius,az,e);
```

Draw the 2000 meter radius circle on the web map, setting the color to black.

```
wmline(lat,lon,'Color','k','OverlayName','2000 Meters')
```
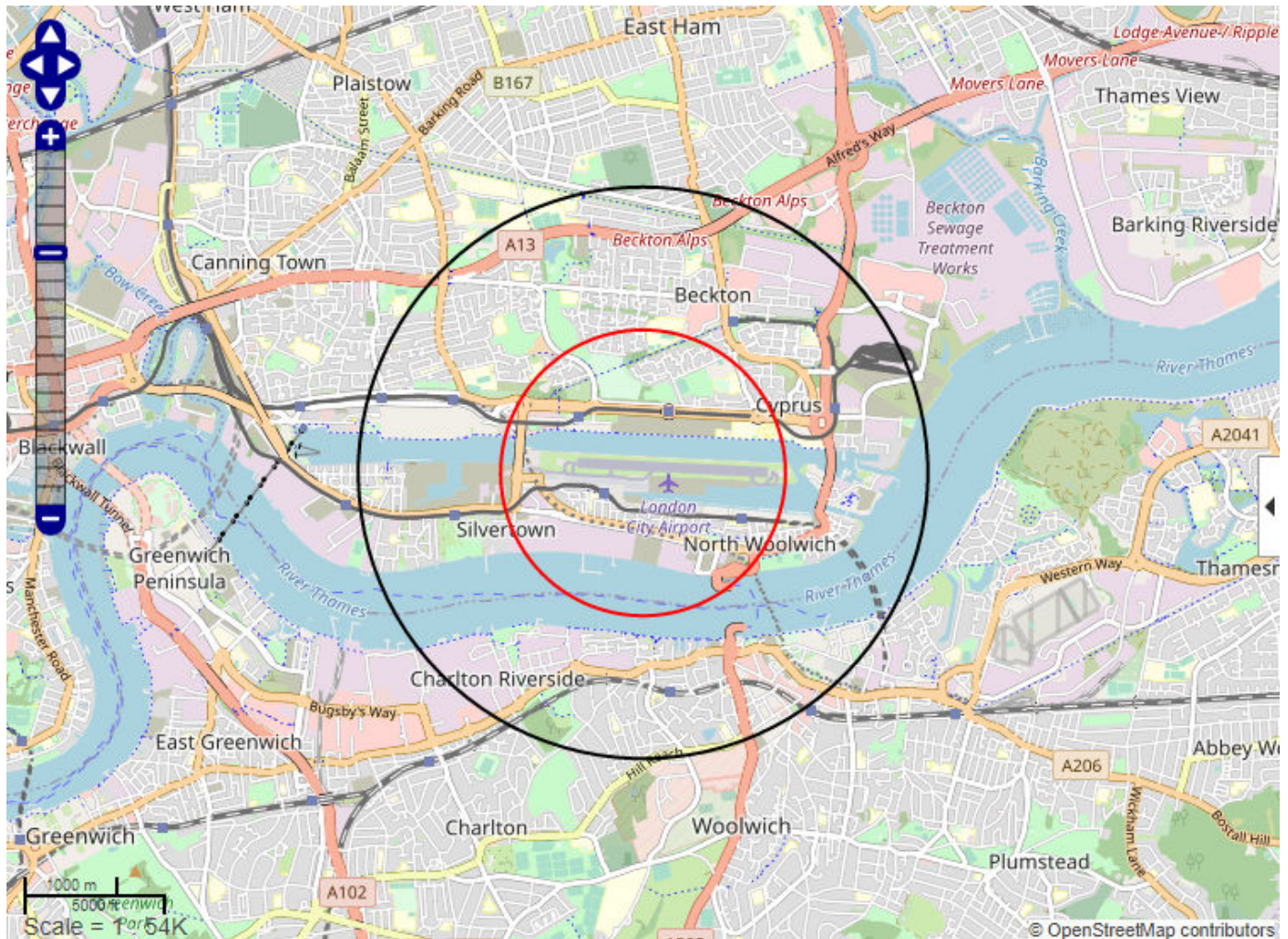
Input Arguments

`lat` — Latitudes of vertices
matrix

Latitudes of vertices, specified as a matrix.

Data Types: `single` | `double`

`lon` — Longitudes of vertices
matrix

Longitudes of vertices, specified as a matrix.

Data Types: `single` | `double`

`P` — Geographic features
geospatial table | `geopoint` vector | `geoshape` vector

Geographic features, specified as one of the following:

- A geospatial table containing `geopointshape`, `geolineshape`, or `geopolyshape` objects. The `wmline` function does not support geospatial tables containing more than one type of shape object. For more information about geospatial tables, see "Create Geospatial Tables".

- A `geopoint` vector.

- A `geoshape` vector.

If P is a geospatial table containing `geopointshape` objects or a `geopoint` vector, then the overlay contains a single line connecting the vertices.

If P is a geospatial table containing `geolineshape` or `geopolyshape` objects, or a `geoshape` vector, then the overlay contains one line feature for each feature of P.

`wm` — Web map
web map handle

Web map, specified as a web map handle.[10]

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `wmline(lat,lon,'OverlayName','Shortest Route');`

Autofit — Overlay visibility
true (default) | false

Overlay visibility, specified as the comma-separated pair consisting of `'Autofit'` and the scalar logical or numeric value `true` (1) or `false` (0).

- If `true`, `wmline` adjusts the spatial extent of the map to ensure that all the vector overlays on the map are visible.

- If `false`, `wmline` does not adjust the spatial extent when this vector layer is added to the map.

Data Types: `double` | `logical`

Description — Description of feature
empty character vector (' ') (default) | character vector | cell array of character vectors | scalar structure

Description of feature, specified as the comma-separated pair consisting of `'Description'` and a character vector, cell array of character vectors, or a scalar structure. The description defines the content of the description balloon displayed when you click the feature in a web map. Description elements can be either plain text or HTML markup. When an attribute spec is provided, the display in the balloon for the attribute fields of P are modified according to the specification.

- If you specify a scalar cell array, `wmline` applies the value to all line features.

- If you specify a nonscalar cell array, the cell array must contain a value for each feature, that is, the cell array must be the same length as P.

---

10. Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

- If the value is a structure, `wmline` applies the attribute specification to each line.

Data Types: `char` | `struct` | `cell`

OverlayName — Name of overlay layer
`'Line Overlay N'`, (default) | character vector

Name of overlay layer, specified as the comma-separated pair consisting of `'OverlayName'` and a character vector. `wmline` inserts the name in the Layer Manager under the "Overlays" item. The Layer Manager is the tool that appears on the right side of the web map. The default name is `'Line Overlay N'` where *N* is the number assigned to this overlay.

Data Types: `char`

FeatureName — Name of feature
`'OverlayName: Line K'` (default) | character vector | cell array of character vectors

Name of feature, specified as the comma-separated pair consisting of `'FeatureName'` and character vector or cell array of character vectors. The name appears in the balloon that displays when you click the feature in the web map. The default value is `'OverlayName : Line K'`, where *OverlayName* is the name of the overlay and *K* is the number assigned to the particular line.

- If the value is a character vector, it applies to all features.
- If the value is a cell array of character vectors, it must be either a scalar or the same length as P.

Data Types: `char` | `cell`

Color — Line color
`'black'` (default) | color name | RGB triplet | cell array of color names | string vector of color names | matrix of RGB triplets

Line color, specified as the comma-separated pair consisting of `'Color'` and one of these options.

- A color name such as `'red'` or a short name such as `'r'`.
- An RGB triplet, which is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A cell array of color names such as `{'red','green','blue'}` or `{'r','g','b'}`.
- A string vector of color names such as `["red" "green" "blue"]` or `["r" "g" "b"]`.
- A matrix of RGB triplets, which is a three-column matrix in which each row is an RGB triplet.

The way you specify the color depends on the desired color scheme.

- To apply the same color to all lines in P, specify a single color name or RGB triplet.
- To apply a different color to each line in P, specify a cell array of color names, a string vector of color names, or a matrix of RGB triplets. The number of colors and RGB triplets must match the length of P.

This table contains the color names and equivalent RGB triplets for some common colors.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	▬▬▬

Color Name	Short Name	RGB Triplet	Appearance
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Data Types: `char` | `string` | `cell` | `double`

LineWidth — Width of line in pixels
1 (default) | positive numeric scalar or vector

Width of line in pixels, specified as the comma-separated pair consisting of `LineWidth` and a positive numeric scalar or vector. If you specify a vector, it must include a value for each line, that is, the vector must be the same length as P.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Alpha — Transparency of line
1 (default) | numeric scalar or vector

Transparency of line, specified as the comma-separated pair consisting of `'Alpha'` and a numeric scalar or vector. If you specify a vector, it must include a value for each line, that is, the vector must be the same length as P. The default value, `1`, means that the line is fully opaque.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

h — Overlay layer
handle to line overlay

Overlay layer, returned as a handle to a line overlay.

Tips

- Under certain conditions, when you zoom in on a line overlay in a web map, parts of the line may become invisible. This can occur if the data is one long line segment that is composed of many parts. To workaround this issue, break the long line into a series of line segments by inserting NaNs in the line data.

See Also

`wmclose` | `wmpolygon` | `wmcenter` | `webmap` | `wmlimits` | `wmprint` | `wmmarker` | `wmremove` | `wmzoom`

Introduced in R2013b

wmpolygon

Display geographic polygon on web map

Syntax

```
wmpolygon(lat,lon)
wmpolygon(P)
wmpolygon(wm, ___ )
wmpolygon( ___ ,Name,Value)
h = wmpolygon( ___ )
```

Description

`wmpolygon(lat,lon)` displays the polygon overlay defined by the vertices in `lat` and `lon` on the current web map. If there is no current web map, `wmpolygon` creates one. `wmpolygon` centers and scales the map so that all the vector overlays displayed in the web map are visible.

`wmpolygon(P)` displays a polygon overlay based on the vector geographic features stored in `P`. The overlay contains one polygon feature for each element of `P`.

`wmpolygon(wm, ___ )` displays the overlay in the web map specified by the web map handle, `wm`.

`wmpolygon( ___ ,Name,Value)` specifies name-value pairs that set additional display properties.

`h = wmpolygon( ___ )` returns a handle to the overlay.

Examples

Display Coastlines as a Polygon

Load coastline data from a MAT-file.

```
load coastlines
```

Display the coast lines as a polygon overlay layer.

```
wmpolygon(coastlat,coastlon,'OverlayName','Polygon coastlines')
```
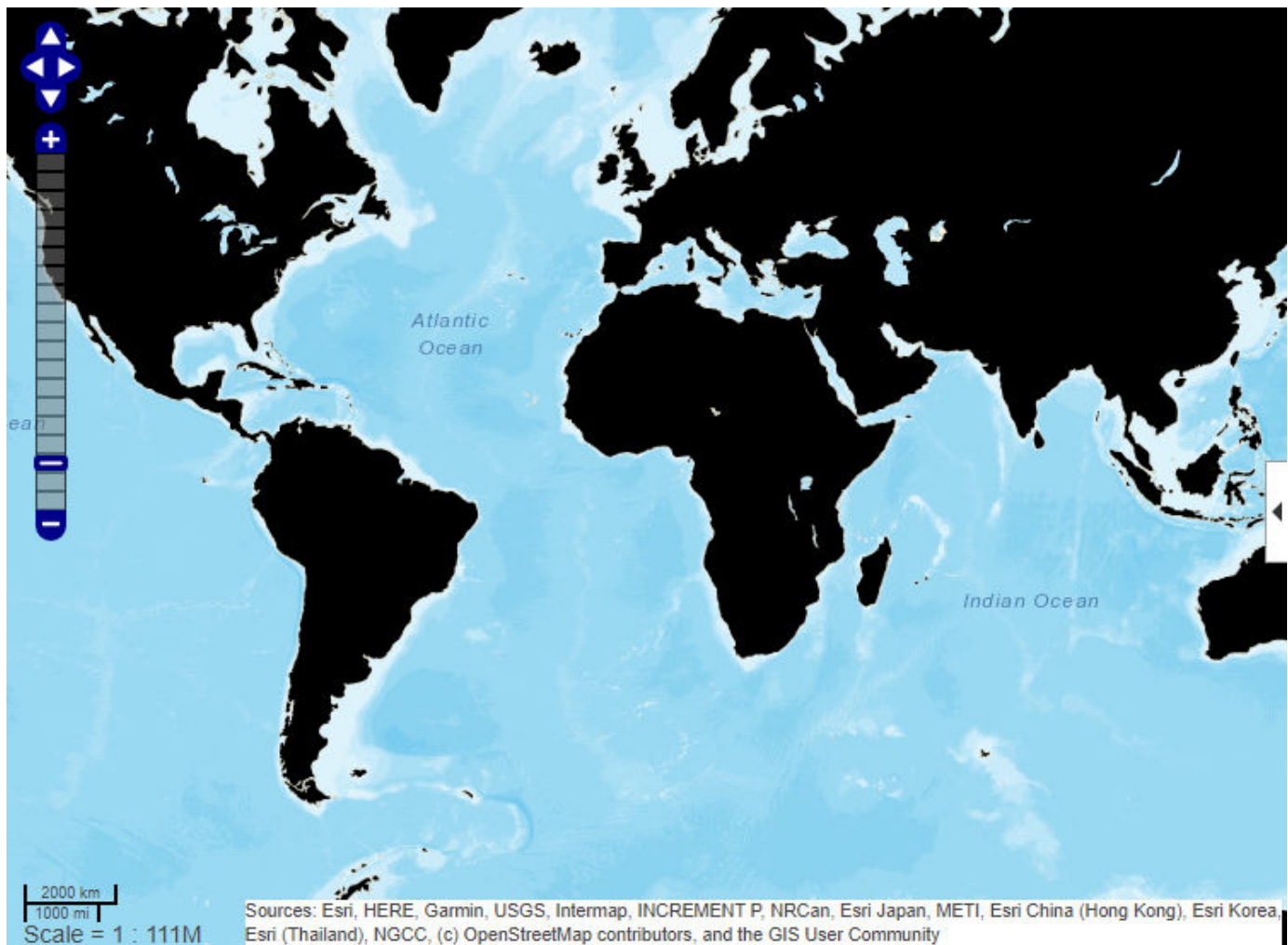
Display Polygon with Inner Ring

Define coordinates of rings. For this example, the coordinates define a location centered on the Eiffel Tower.

```
lat0 = 48.858288;
lon0 = 2.294548;
outerRadius = .01;
innerRadius = .005;
[lat1,lon1] = scircle1(lat0,lon0,outerRadius);
[lat2,lon2] = scircle1(lat0,lon0,innerRadius);
lat2 = flipud(lat2);
lon2 = flipud(lon2);
lat = [lat1; NaN; lat2];
lon = [lon1; NaN; lon2];
```

Display on web map.

```
webmap('worldimagery')
wmpolygon(lat,lon,'EdgeColor','g','FaceColor','c','FaceAlpha',.5)
```
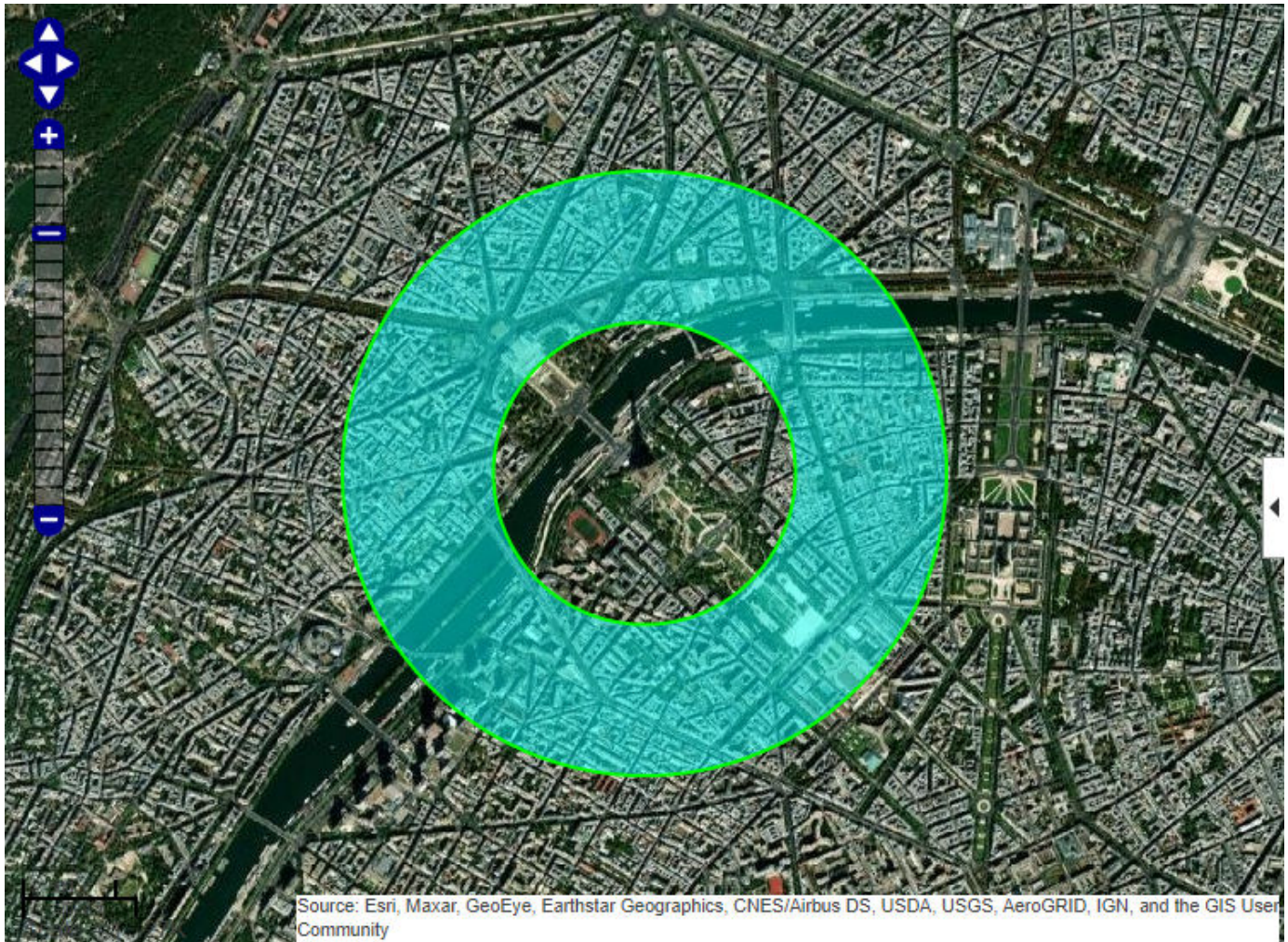
Display USA State Boundaries Using Political Colormap

Import a shapefile containing state boundary data as a geospatial table.

```
GT = readgeotable("usastatelo.shp");
```

Create a colormap from random, muted colors.

```
colors = polcmap(height(GT));
```

Display the polygons as an overlay on a web map. Make the polygons semitransparent by using the `FaceAlpha` name-value argument.

```
webmap("worldphysicalmap")
wmpolygon(GT,"FaceColor",colors,"FaceAlpha",0.5,"EdgeColor","k", ...
    "EdgeAlpha",0.5,"OverlayName","USA Boundary","FeatureName",GT.Name)
```
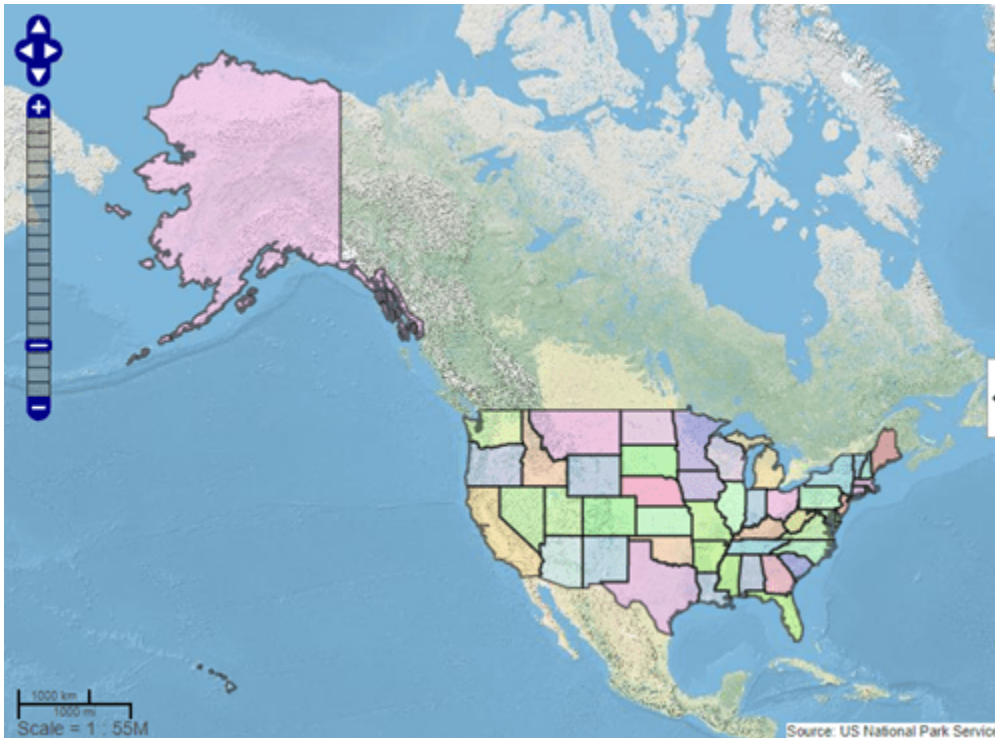
Display Reduced High Resolution Polygon Data on Web Map

Large data sets can sometimes be slow to display, making the web map appear to hang. This example shows how to reduce the size of a data set using `reducem` before calling `wmpolygon`.

First, load high-resolution vector data into the workspace.

```
states = shaperead('usastatehi.shp','UseGeoCoords', true);
states = geoshape(states);
```

Then, reduce the number of points in the latitude and longitude vectors using the `reducem` function.

```
for k = 1:length(states)
    [states(k).Latitude, states(k).Longitude] = reducem( ...
        states(k).Latitude', states(k).Longitude');
end
```

Display state boundaries on the web map. Note that the borders of the reduced polygons may not meet if you zoom in on them.

```
colors = polcmap(length(states));
webmap('worldphysicalmap')
wmpolygon(states,'FaceColor',colors,'FaceAlpha',.5,'EdgeColor','k', ...
    'EdgeAlpha',.5,'OverlayName','USA Boundary','FeatureName',states.Name)
```
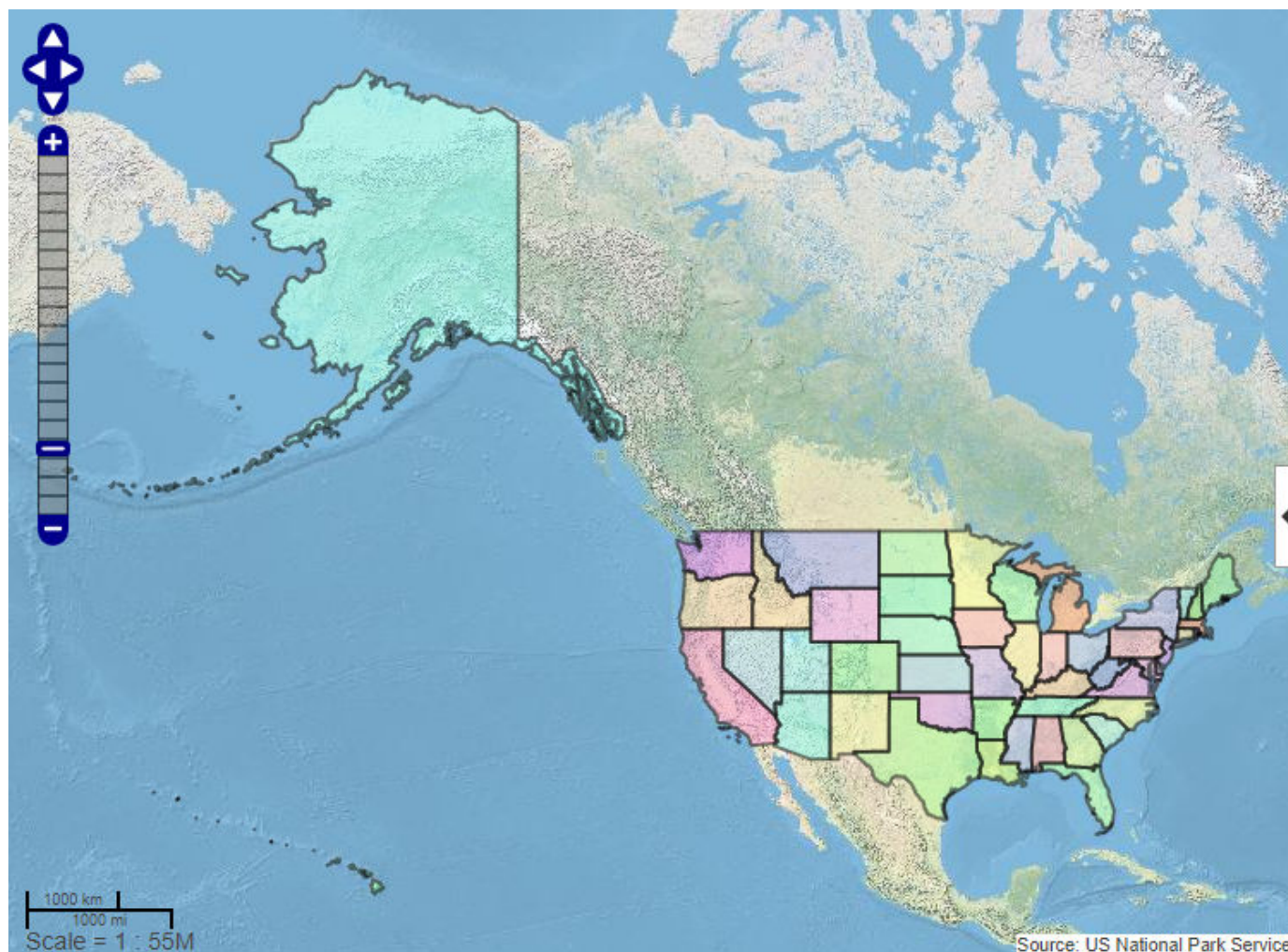
Input Arguments

`lat` — Latitude vertices
matrix in the range [-90, 90]

Latitude vertices, specified as matrix in the range [-90, 90].

Data Types: `single` | `double`

`lon` — Longitude vertices
matrix

Longitude of vertices, specified as a matrix.

Data Types: `single` | `double`

P — Geographic features
geospatial table | `geoshape` vector

Geographic features, specified as one of the following:

- A geospatial table containing `geopolyshape` objects. For more information about geospatial tables, see "Create Geospatial Tables".
- A `geoshape` vector with polygon geometry.

`wm` — Web map
handle to a web map

Web map, specified as a handle to a web map.[11]

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `wmpolygon(lat,lon,'Autofit',true)`

`Autofit` — Overlay visibility
true (default) | false

Overlay visibility, specified as the comma-separated pair consisting of `'Autofit'` and the scalar logical or numeric value `true` (1) or `false` (0).

- If `true`, `wmpolygon` adjusts the spatial extent of the map to ensure that all the vector overlays on the map are visible.
- If `false`, `wmpolygon` does not adjust the spatial extent of the map when this vector layer is added to the map.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

`Description` — Description of feature
empty character vector (default) | character vector | cell array of character vectors | scalar structure

Description of feature, specified as the comma-separated pair consisting of `'Description'` and a character vector, cell array of character vectors, or a scalar structure.

- If you specify a character vector, the text defines the content displayed in the description balloon, which appears when you click the feature in the web map. Description elements can be either plain text or marked up with HTML markup.
- If you specify a cell array, it must be either a scalar or the same length as `P`, and specifies the description for each polygon.
- If the value is a structure (attribute specification), `wmpolygon` displays the attribute fields of `P` in the balloon, modified according to the specification.

Data Types: `char` | `struct` | `cell`

`OverlayName` — Name of overlay layer
`'Polygon Overlay N'`, (default) | character vector

Name of overlay layer, specified as the comma-separated pair consisting of `'OverlayName'` and a character vector. `wmpolygon` inserts the name in the Layer Manager under the "Overlays" item. The

---

11. Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

Layer Manager is the tool that appears on the right side of the web map. The default name is `'Polygon Overlay N'` where *N* is the number assigned to this overlay.

Data Types: `char`

FeatureName — Name of feature
`'OverlayName: Line K'` (default) | character vector | cell array of character vectors

Name of feature, specified as the comma-separated pair consisting of `'FeatureName'` and character vector or cell array of character vectors. The name appears in the balloon when you click the feature in the web map. The default value is `'OverlayName: Polygon K'`, where *OverlayName* is the name of the overlay and *K* is the number assigned to the particular polygon.

- If the value is a character vector, it applies to all features.
- If the value is a cell array of character vectors, it must be either a scalar or the same length as P.

Data Types: `char` | `cell`

FaceColor — Color of polygon faces
`'black'` (default) | color name | RGB triplet | cell array of color names | string vector of color names | matrix of RGB triplets | `'none'`
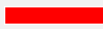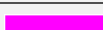
Color of polygon faces, specified as the comma-separated pair consisting of `'FaceColor'` and one of these options.

- A color name such as `'red'` or a short name such as `'r'`.
- An RGB triplet, which is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A cell array of color names such as `{'red','green','blue'}` or `{'r','g','b'}`.
- A string vector of color names such as `["red" "green" "blue"]` or `["r" "g" "b"]`.
- A matrix of RGB triplets, which is a three-column matrix in which each row is an RGB triplet.

The way you specify the color depends on the desired color scheme.

- To apply the same face color to all polygons in P, specify a single color name or RGB triplet.
- To apply a different color to each polygon in P, specify a cell array of color names, a string vector of color names, or a matrix of RGB triplets. The number of colors and RGB triplets must match the length of P.
- To create polygons with no fill, specify `'none'`.

This table contains the color names and equivalent RGB triplets for some common colors.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	
`'cyan'`	`'c'`	`[0 1 1]`	
`'magenta'`	`'m'`	`[1 0 1]`	

Color Name	Short Name	RGB Triplet	Appearance
`'yellow'`	`'y'`	`[1 1 0]`	
`'black'`	`'k'`	`[0 0 0]`	
`'white'`	`'w'`	`[1 1 1]`	

Data Types: `char` | `string` | `cell` | `double`

FaceAlpha — Transparency of polygon faces
1 (default) | numeric scalar or vector in the range [0, 1]

Transparency of polygon faces, specified as the comma-separated pair consisting of `'FaceAlpha'` and a numeric scalar or vector in the range [0, 1]. The default value, `1`, means that the polygon is fully opaque.

- If the value is a scalar, it applies to all polygon faces.

- If the value is a vector, it must be the same length as `P`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

EdgeColor — Color of polygon edges
`'none'` (default) | color name | RGB triplet | cell array of color names | string vector of color names | matrix of RGB triplets | `'none'`
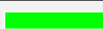
Color of polygon edges, specified as the comma-separated pair consisting of `'EdgeColor'` and one of these options.

- A color name such as `'red'` or a short name such as `'r'`.

- An RGB triplet, which is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.

- A cell array of color names such as `{'red','green','blue'}` or `{'r','g','b'}`.

- A string vector of color names such as `["red" "green" "blue"]` or `["r" "g" "b"]`.

- A matrix of RGB triplets, which is a three-column matrix in which each row is an RGB triplet.

The way you specify the color depends on the desired color scheme.

- To apply the same edge color to all polygons in `P`, specify a single color name or RGB triplet.

- To apply a different edge color to each polygon in `P`, specify a cell array of color names, a string vector of color names, or a matrix of RGB triplets. The number of colors and RGB triplets must match the length of `P`.

- To create polygons with no edge color, specify `'none'`.

This table contains the color names and equivalent RGB triplets for some common colors.

Color Name	Short Name	RGB Triplet	Appearance
`'red'`	`'r'`	`[1 0 0]`	
`'green'`	`'g'`	`[0 1 0]`	
`'blue'`	`'b'`	`[0 0 1]`	

Color Name	Short Name	RGB Triplet	Appearance
'cyan'	'c'	[0 1 1]	████
'magenta'	'm'	[1 0 1]	████
'yellow'	'y'	[1 1 0]	████
'black'	'k'	[0 0 0]	████
'white'	'w'	[1 1 1]	░░░░

Data Types: char | string | cell | double

EdgeAlpha — Transparency of polygon edges
1 (default) | numeric scalar or vector in the range [0, 1]

Transparency of polygon edges, specified the comma-separated pair consisting of `'EdgeAlpha'` and as a numeric scalar or vector in the range [0, 1].

- If the value is a scalar, it applies to all polygon faces.
- If the value is a vector, it must be the same length as P.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64

LineWidth — Width of polygon edges
1 (default) | positive numeric scalar or vector

Width of polygon edges, specified as the comma-separated pair consisting of `'LineWidth'` and a positive numeric scalar or vector.

- If the value is a scalar, it applies to all polygon faces.
- If the value is a vector, it must be the same length as P.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64

Output Arguments

h — Polygon overlay
handle to polygon overlay

Polygon overlay, returned as a handle to the polygon overlay.

Tips

- When you move the cursor over the polygons you define on an image and click, `wmpolygon` displays a description balloon and disables panning. If you move the cursor off the polygon, you can still click and pan the image. You can also use the arrow keys to pan the image.

See Also

Functions
wmclose | wmcenter | webmap | wmlimits | wmline | wmprint | wmmarker | wmremove | wmzoom

Objects
geoshape

Topics
"Create and Display Polygons"

Introduced in R2016a

wmremove

Remove overlay on web map

Syntax

```
wmremove()
wmremove(h)
```

Description

`wmremove()` removes the overlay most recently inserted into the current web map.[12]

`wmremove(h)` removes the overlay or overlays specified by the scalar overlay handle or vector of overlay handles, `h`.

Examples

Remove a Marker Overlay

Draw a marker on a web map. `wmmarker` creates the web map. Pause, and then remove the marker overlay.

```
wmmarker(42, -73);
pause(1);
wmremove()
```

Remove Multiple Overlays

Draw several marker overlays on a web map. `wmmarker` creates the web map. Pause, and then remove the marker overlays, specifying a vector of overlay handles.

```
h1 = wmmarker(42, -80);
h2 = wmmarker(42, -78);
pause(1);
wmremove([h1 h2])
```

Remove Line Overlay

Create a web map.

```
wm = webmap();
```

Load coastline data and display it as an overlay on the web map.

---

12. Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

```
load coastlines
h = wmline(coastlat, coastlon);
```

Remove the overlay specified by h.

```
wmremove(h)
```

Input Arguments

h — Web map overlay
scale overlay handle or vector of overlay handles

Web map overlay, specified as a scalar overlay handle or a vector of overlay handles.

See Also

wmclose | wmcenter | wmline | wmlimits | wmprint | wmmarker | webmap | wmzoom

Introduced in R2013b

wmcenter

Set or obtain web map center point

Syntax

```
wmcenter(centerLatitude,centerLongitude)
wmcenter(wm,centerLatitude,centerLongitude)
wmcenter( ___ ,zoomLevel)
[lat,lon] = wmcenter()
[lat,lon] = wmcenter(wm)
```

Description

`wmcenter(centerLatitude,centerLongitude)` centers the current web map at the specified latitude and longitude. If there is no current web map, `wmcenter` creates a new web map.

`wmcenter(wm,centerLatitude,centerLongitude)` centers the web map, specified by the handle `wm`, at the specified latitude and longitude.

`wmcenter( ___ ,zoomLevel)` centers and zooms the web map to the specified zoom level.

`[lat,lon] = wmcenter()` returns the latitude and longitude of the center point of the current web map.

`[lat,lon] = wmcenter(wm)` returns the latitude and longitude of the center point of the web map specified by the handle `wm`.

Examples

Center a Web Map

Display a web map and find its center point. There is no current web map, so `wmcenter` creates one.

```
[centerLatitude,centerLongitude] = wmcenter()

centerLatitude =

     0


centerLongitude =

     0
```

Center the map at a specified center point.

```
wmcenter(51.52,0)
```

**Specify Center and Zoom Level for Multiple Web Maps**

Create two web maps.

```
wm1 = webmap('OpenStreetMap');
wm2 = webmap('USGSImagery');
```

Specify the latitude and longitude for the web map center, as well as a zoom level. Then, center the web maps.

```
centerLatitude = 36.1;
centerLongitude = -113.2;
zoomLevel = 10;
wmcenter(wm1,centerLatitude,centerLongitude,zoomLevel)
wmcenter(wm2,centerLatitude,centerLongitude,zoomLevel)
```
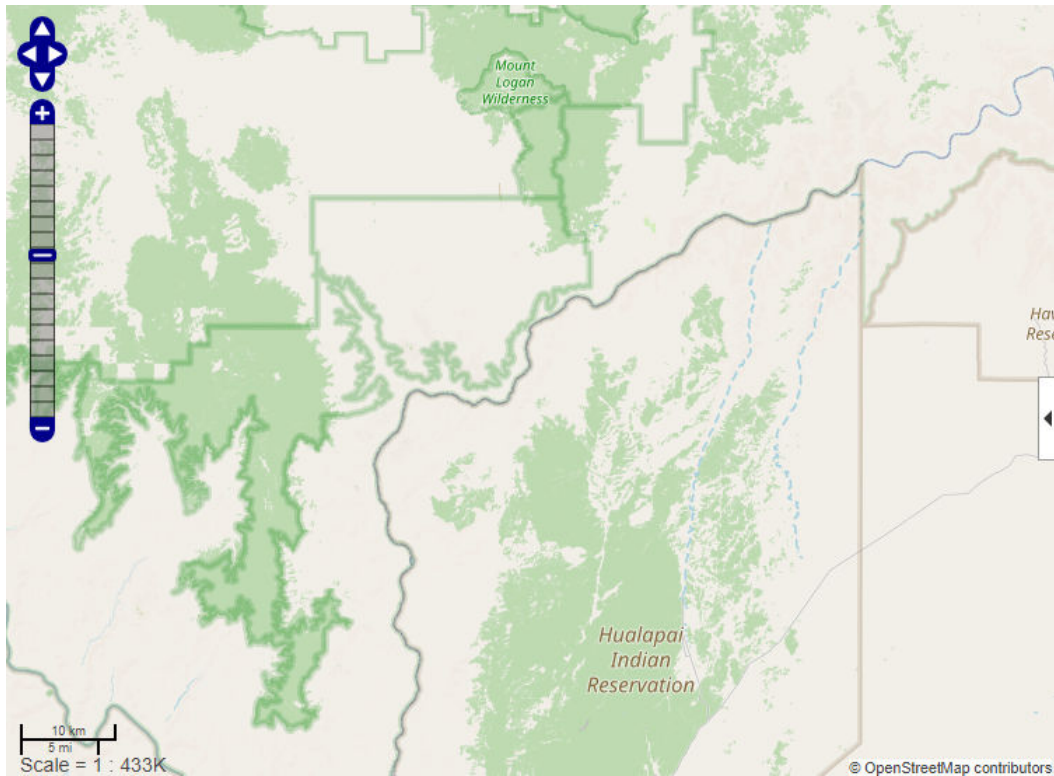
Input Arguments

`centerLatitude` — Latitude of center point
scalar in the range `[-90 90]` of type `single` or `double`

Latitude of center point, specified as a scalar in the range `[-90 90]` of type `single` or `double`.

Data Types: `single` | `double`

`centerLongitude` — Longitude of center point
scalar in the range `[-180 180]` of type `single` or `double`

Longitude of center point, specified as a scalar in the range `[-180 180]` of type `single` or `double`.

Data Types: `single` | `double`

`wm` — Web map
web map handle

Web map, specified as a web map handle.[13]

`zoomLevel` — Zoom level
scalar numeric integer in the range `[0 18]` of type `single` or `double`

Zoom level, specified as a scalar numeric integer in the range `[0 18]` of type `single` or `double`.

Data Types: `single` | `double`

---

13.   Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

Output Arguments

`lat` — Latitude of center point
scalar in the range `[-90 90]` of type `single` or `double`

Latitude of center point, returned as a scalar in the range `[-90 90]` of type `single` or `double`.

Data Types: `single` | `double`

`lon` — Longitude of center point
scalar in the range `[-180 180]` of type `single` or `double`

Longitude of center point, returned as a scalar in the range `[-180 180]` of type `single` or `double`.

Data Types: `single` | `double`

Tips

- Particular maps may not support every available zoom level. If your map displays as completely white, try another zoom level. The map you are displaying may not support the zoom level you have currently selected. You can also select another base layer, which might support the specified zoom level.

See Also
`wmclose` | `webmap` | `wmline` | `wmlimits` | `wmprint` | `wmmarker` | `wmremove` | `wmzoom`

Introduced in R2013b

wmzoom

Set or obtain zoom level of web map

Syntax

```
wmzoom(zoomLevel)
wmzoom(wm,zoomLevel)
zoomLevelOut = wmzoom()
zoomLevelOut = wmzoom(wm)
```

Description

wmzoom(zoomLevel) specifies the zoom level of the current web map, zoomLevel. If there is no current web map, wmzoom creates a new web map.

wmzoom(wm,zoomLevel) specifies the zoom level of the web map, specified by the handle wm.

zoomLevelOut = wmzoom() returns the zoom level of the current web map.

zoomLevelOut = wmzoom(wm) returns the zoom level of the web map specified by the handle wm.

Examples

Get Zoom Level of Current Web Map and Specify New Zoom Level

Create a web map at default zoom level.

```
zoomLevel = wmzoom()

zoomLevel =

     0
```

Zoom in and center the web map at the specified latitude and longitude (London).

```
wmzoom(10)
wmcenter(51.52, 0)
```

Specify Zoom Level of Several Web Maps

Create two web maps.

```
wm1 = webmap;
wm2 = webmap('worldtopographic');
```

Zoom in and center both maps at a specified latitude and longitude (Paris).

```
lat = 48.821;
lon = 1.9391;
```

```
zoomLevel = 10;

wmzoom(wm1, zoomLevel)
wmcenter(wm1, lat, lon)
wmzoom(wm2, zoomLevel)
wmcenter(wm2, lat, lon)
```

Input Arguments

`zoomLevel` — Zoom level
scalar numeric integer in the range `[0 18]`

Zoom level, specified as a scalar numeric integer in the range `[0 18]`.

Data Types: `single` | `double`

`wm` — Web map
web map handle

Web map, specified as a web map handle.[14]

Output Arguments

`zoomLevelOut` — Zoom level of the current web map
scalar numeric integer in the range `[0 18]`

Zoom level of the current web map, returned as a scalar numeric integer in the range `[0 18]`.

Data Types: `single` | `double`

Tips

- Particular maps may not support every available zoom level. If your map displays as completely white, try another zoom level. The map you are displaying may not support the zoom level you have currently selected. You can also select another base layer, which might support the specified zoom level.

See Also
`wmclose` | `wmcenter` | `wmline` | `wmlimits` | `wmprint` | `wmmarker` | `wmremove` | `webmap`

Introduced in R2013b

---

14. Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

wmlimits

Set or obtain web map limits

Syntax

```
wmlimits(latitudeLimits,longitudeLimits)
wmlimits(wm,latitudeLimits,longitudeLimits)
[latlim,lonlim] = wmlimits()
[latlim,lonlim] = wmlimits(wm)
```

Description

wmlimits(latitudeLimits,longitudeLimits) center the current web map within the specified latitude limits and the longitude limits. If there is no current web map, wmlimits creates one.

Note The resulting limits often do not match the specified limits because the zoom level is quantized to discrete integer values and the longitude limits may be constrained if the map was constructed with the WrapAround property equal to false.

---

wmlimits(wm,latitudeLimits,longitudeLimits) centers the web map specified by the web map handle wm within the specified latitude limits and longitude limits.

[latlim,lonlim] = wmlimits() returns the latitude and longitude limits of the current web map.

[latlim,lonlim] = wmlimits(wm) returns the latitude and longitude limits of the web map specified by wm.

Examples

Specify Limits for Web Map

Specify latitude and longitude limits. wmlimits creates the web map.

```
wmlimits([37 42],[-108.9 -100.7])
```

Get the latitude and longitude limits of the current web map. The limits returned by `wmlimits` depend on the size of the web map browser and the zoom level. Therefore, your results may differ from the results shown here.

```
[latitudeLimits,longitudeLimits] = wmlimits()
```

```
latitudeLimits =

    34.8844    43.9124
```

```
longitudeLimits =

  -112.9738   -96.6262
```

**Specify Limits for Multiple Web Maps**

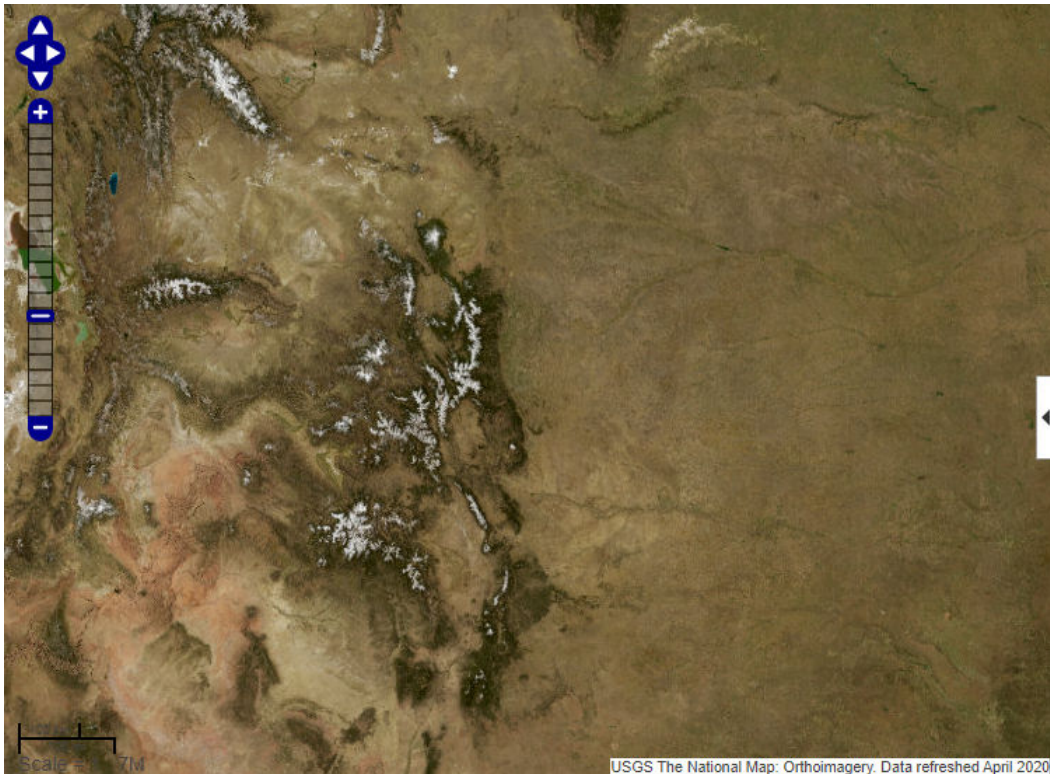Create two web maps, specifying different base layers.

```
wm1 = webmap('OpenStreetMap');
wm2 = webmap('USGSImagery');
```

Specify latitude and longitude limits and apply to both maps to get two different views of the same region.

```
latitudeLimits = [37, 42];
longitudeLimits = [-108.9, -100.7];
wmlimits(wm1, latitudeLimits, longitudeLimits)
wmlimits(wm2, latitudeLimits, longitudeLimits)
```
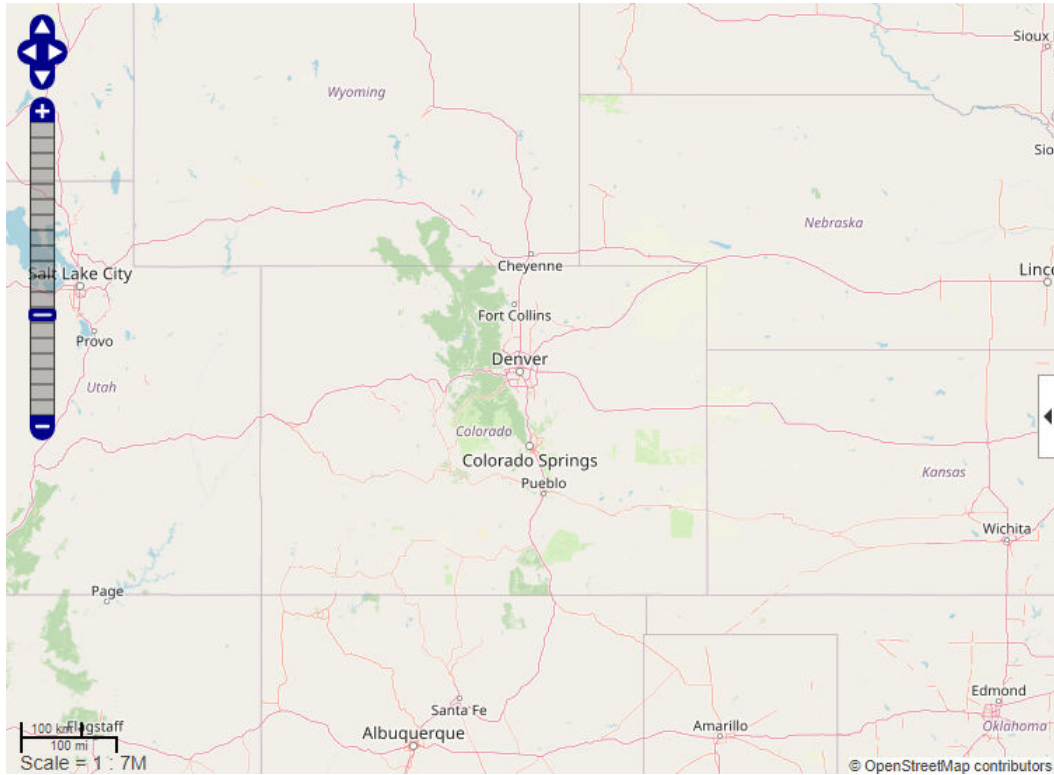
Input Arguments

`latitudeLimits` — Latitude limits in degrees
1-by-2 row vector of type `double`

Latitude limits in degrees, specified as a 1-by-2 row vector of type `double` of the form `[southern-limit northern-limit]`.

Example: `[37, 42]`

Data Types: `double`

`longitudeLimits` — Longitude limits in degrees
1-by-2 row vector of type `double`

Longitude limits in degrees, specified as a 1-by-2 row vector of type `double` of the form `[western-limit eastern-limit]`.

Example: `[-108.9, -100.7]`

Data Types: `double`

`wm` — Web map
scalar web map handle

Web map, specified as a scalar web map handle.[15]

---

15. Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

Output Arguments

`latlim` — Latitude limits in degrees
1-by-2 row vector of type `double`

Latitude limits in degrees, returned as a 1-by-2 row vector of type `double`.

Example: `[37, 42]`

Data Types: `double`

`lonlim` — Longitude limits in degrees
1-by-2 row vector of type `double`

Longitude limits in degrees, returned as a 1-by-2 row vector of type `double`.

Example: `[-108.9, -100.7]`

Data Types: `double`

See Also

`wmclose` | `wmcenter` | `wmline` | `webmap` | `wmprint` | `wmmarker` | `wmremove` | `wmzoom`

Introduced in R2013b

WMSCapabilities

Web Map Service capabilities document

Description

A `WMSCapabilities` object represents a Web Map Service (WMS) capabilities document obtained from a WMS server. A capabilities document is an XML document that contains metadata describing the geographic content offered by the server.

Creation

You can create a `WMSCapabilities` object using the `WMSCapabilities` function described here, or using the `wmsinfo` function to specify a timeout. A `WMSCapabilities` is also returned from the `getCapabilities` function when you have a `WebMapServer` object.

Syntax

`capabilities = WMSCapabilities(serverURL,capabilitiesResponse)`

Description

`capabilities = WMSCapabilities(serverURL,capabilitiesResponse)` creates a `WMSCapabilities` object, sets the `ServerURL` property, and defines the capabilities of the server using the XML elements in `capabilitiesResponse`.

Input Arguments

serverURL — URL of WMS server
character vector | string scalar

URL of WMS server, specified as a character vector or a string scalar. The server URL must include the protocol `'http://'` or `'https://'`.

Data Types: `char` | `string`

capabilitiesResponse — XML elements that describe the capabilities of the WMS server
character vector | string scalar

XML elements that describe the capabilities of the WMS server, specified as a character vector or a string scalar.

Data Types: `char` | `string`

Properties

ServerTitle — Title of WMS server
character vector

This property is read-only.

Title of WMS server, returned as a character vector.

Data Types: char

ServerURL — URL of WMS server
character vector

This property is read-only.

URL of WMS server, returned as a character vector. The server URL must include the protocol
'http://' or 'https://'.

Data Types: char

ServiceName — Name of Web map service
character vector

This property is read-only.

Name of Web map service, returned as a character vector.

Data Types: char

Version — WMS version specification
character vector

This property is read-only.

WMS version specification, returned as a character vector.

Data Types: char

Abstract — Information about server
character vector

This property is read-only.

Information about server, returned as a character vector.

Data Types: char

OnlineResource — Online information about server
character vector

This property is read-only.

Online information about server, returned as a character vector.

Data Types: char

ContactInformation — Contact information for an individual or an organization
structure

This property is read-only.

Contact information for an individual or an organization, returned as a ContactInformation
structure, containing the following fields:

Field Name	Data Type	Field Content
Person	Character vector	Name of individual
Organization	Character vector	Name of organization
Email	Character vector	Email address

Data Types: struct

AccessConstraints — Constraints inherent in accessing the server
character vector

This property is read-only.

Constraints inherent in accessing the server, such as server load limits, returned as a character vector.

Data Types: char

Fees — Types of fees associated with accessing server
character vector

This property is read-only.

Types of fees associated with accessing server, returned as a character vector.

Data Types: char

KeywordList — Descriptive keywords of the server
cell array of character vectors

This property is read-only.

Descriptive keywords of the server, returned as a cell array of character vectors.

Data Types: char

ImageFormats — Image formats supported by server
cell array of character vectors

This property is read-only.

Image formats supported by server, returned as a cell array of character vectors.

Data Types: char

LayerNames — Layer names provided by server
cell array of character vectors

This property is read-only.

Layer names provided by server, returned as a cell array of character vectors.

Data Types: cell

Layer — Information about layers on WMS server
WMSLayer array

This property is read-only.

Information about layers on WMS server, returned as an array of `WMSLayer` objects.

AccessDate — Date of request to server
character vector

This property is read-only.

Date of request to server, returned as a character vector.

Data Types: `char`

Object Functions

disp   Display properties of WMS layers or capabilities

Examples

Create WMS Capabilities Object

Create a `WMSCapabilities` object from the content of a downloaded capabilities file from the NASA SVS Image Server.

```
nasa = wmsfind('NASA SVS Image', 'SearchField', 'servertitle');
serverURL = nasa(1).ServerURL;
server = WebMapServer(serverURL);
capabilities = server.getCapabilities;
filename = 'capabilities.xml';
websave(filename, server.RequestURL);

fid = fopen(filename, 'r');
capabilitiesResponse = fread(fid,'uint8=>char');
fclose(fid);
capabilities = WMSCapabilities(serverURL,capabilitiesResponse);
```

See Also

WMSLayer | WebMapServer | wmsinfo | websave

Topics
"Explore Layers on Same Server"

Introduced in R2009b

wmsfind

Search local database for Web map servers and layers

Syntax

```
layers = wmsfind(querystr)
layers = wmsfind(querystr,Name,Value)
```

Description

`layers = wmsfind(querystr)` searches the fields of the installed "WMS Database" on page 1-1546 for partial matches of `querystr`, which is a string, string array, character vector, or cell array of character vectors. By default, `wmsfind` searches the `Layer` or `LayerName` properties but you can include other fields in the search using the `SearchFields` parameter.

`wmsfind` returns `layers`, an array of `WMSLayer` objects containing one object for each layer whose name or title partially matches `querystr`. WMS servers produce maps of spatially referenced raster data, such as temperature or elevation, that are known as layers.

`layers = wmsfind(querystr,Name,Value)` modifies the search of the WMS Database based on the values of the parameters. You can abbreviate parameter names and case does not matter.

Examples

Find Temperature Data in the WMS Database

Search the WMS Database for temperature data.

```
layers = wmsfind('temperature');
```

The `wmsfind` function searches both the `LayerName` and `LayerTitle` fields of the WMS Database for partial matches. The function returns an array of `WMSLayer` objects, which contains one object for each layer whose name or title partially matches `'temperature'`. Display information about the first layer.

```
layers(1)

ans =
  WMSLayer

  Properties:
          Index: 1
    ServerTitle: 'CubeSERV WMS'
      ServerURL: 'https://ca.nfis.org/cubewerx/cubeserv/cubeserv.cgi?'
     LayerTitle: 'Temperature / Température (SCRIBE forecast) - 2020-10-22'
      LayerName: 'CFSNet-CWFIS.temp_current'
         Latlim: [35.8409 83.3979]
         Lonlim: [-174.2769 -11.4519]

  Methods
```

A `WMSLayer` object contains three properties that do not appear in the WMS Database: `Abstract`, `CoordRefSysCodes`, and `Details`. Populate these properties by using the `wmsupdate` function.

```
updatedLayers = wmsupdate(layers(1));
updatedLayers(1)

ans =
  WMSLayer

  Properties:
            Index: 1
      ServerTitle: 'CubeSERV WMS'
        ServerURL: 'https://ca.nfis.org/cubewerx/cubeserv/cubeserv.cgi?'
       LayerTitle: 'Temperature / Température (SCRIBE forecast) - 2021-03-31'
        LayerName: 'CFSNet-CWFIS.temp_current'
           Latlim: [35.8409 83.3979]
           Lonlim: [-174.2769 -11.4519]
         Abstract: 'This map displays conditions at local noon. Because Canada covers several time

        Metadata / Métadonnées: http://cwfis.cfs.nrcan.gc.ca/downloads/cffdrs/wx_grids_metadata_N
CoordRefSysCodes: {5x1 cell}
          Details: [1x1 struct]

  Methods
```

Your results might be different because the WMS Database changes each release.

Search the WMS Database for Layers

Search the Entire WMS Database

Search the WMS database for layers that contain the word "temperature". The `wmsfind` function returns an array of `WMSLayer` objects.

```
layers = wmsfind('temperature');
```

Find layers that contain global temperature data. The query includes the asterisk wildcard character '*'.

```
layers = wmsfind('global*temperature');
```

Search Specific Fields in the WMS Database

Search the `LayerTitle` field for all layers that contain an exact match for the term `'Rivers'`. You must use the `MatchType` parameter to specify an exact match.

```
layers = wmsfind('Rivers','MatchType','exact', ...
                 'IgnoreCase',false,'SearchFields','layertitle');
```

Search the `LayerName` field for all layers that contain a partial match for 'elevation'. By default, `wmsfind` searches for partial matches.

```
layers = wmsfind('elevation','SearchFields','layername');
```

Search the `LayerName` field for all unique servers that contain `'BlueMarbleNG'`.

```
layers = wmsfind('BlueMarbleNG','SearchFields','layername', ...
                 'MatchType','exact');
urls = servers(layers);
```

Limit Your Search to Specific Geographic Regions

Find layers that contain elevation data for Colorado. Use the `Latlim` and `Lonlim` parameters to specify the location.

```
latlim = [35 43];
lonlim = [-111 -101];
layers = wmsfind('elevation','Latlim',latlim,'Lonlim',lonlim);
```

Find all layers that contain temperature data for a point in Perth, Australia. Use the `Latlim` and `Lonlim` parameters to specify the location.

```
lat = -31.9452;
lon = 115.8323;
layers = wmsfind('temperature','Latlim',lat,'Lonlim',lon);
```

Find all the unique URLs of all government servers.

```
layers = wmsfind('*.gov*','SearchFields','serverurl');
urls = servers(layers);
```

Search Multiple Fields at the Same Time and Refine Your Search

Search both the `LayerTitle` and the `LayerName` fields for all the layers that contain the word "temperature".

```
fields = [string('layertitle') string('layername')];
temperature = wmsfind('temperature','SearchFields',fields);
```

Refine the results of your temperature search to find only those layers that deal with sea surface temperatures. Use the `WMSLayer` object `refine` method.

```
sst = refine(temperature,'sea surface');
```

Refine your sea surface temperature search further to find only those layers that deal with global sea surface temperatures.

```
global_sst = refine(sst,'global');
```

Search the Entire WMS Database and Progressively Refine Your Search

Note that finding all the layers from the WMS database may take several seconds to execute and require a substantial amount of memory. The database contains more than 100,000 layers.

Find all the layers in the WMS database and sort them into a set that comprises only the unique layer titles.

```
layers = wmsfind('*');
layerTitles = sort(unique({layers.LayerTitle}))';
```

Refine your original search, `layers`, to include only those layers with global coverage. Use the `WMSLayer` object `refineLimits` method.

```
global_layers = refineLimits(layers, ...
                    'Latlim',[-90 90],'Lonlim',[-180 180]);
```

Refine the results of your global layers search to contain only layers with global extent that include the word "topography". Use the `WMSLayer` object `refine` method.

```
global_topography_layers = refine(global_layers,'topography');
```

Refine your original search, `layers`, to contain only layers that have some combination of the terms "oil" and "gas" in the `LayerTitle` field.

```
oil_gas_layers = refine(layers,'oil*gas','SearchFields','layertitle');
```

Search Online Version of WMS Database

Search the WMS database for layers containing the word "elevation". Search the online version of the database by specifying the `Version` name-value pair argument as `'online'`. If you do not specify the version, then `wmsfind` reads from the installed database.

```
elevation = wmsfind('elevation','Version','online');
```

Input Arguments

querystr — Characters to search for in WMS Database fields
string scalar | string array | character vector | cell array of character vectors

Characters to search for in WMS Database fields, specified as a string scalar, string array, character vector, or cell array of character vectors. `querystr` can contain the asterisk wildcard character (*).

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `[A,R] = wmsfind('elevation','SearchFields','layername');`

IgnoreCase — Ignore case when comparing field values to querystr
true (default) | false

Ignore case when comparing field values to `querystr`, specified as the logical value `true` or `false`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

Latlim — Latitudinal limits of the search
two-element numeric vector | numeric scalar

Latitudinal limits of the search, specified as a two-element vector of the form `[southern_limit northern_limit]` or a scalar value representing the latitude of a single point. Values are in the range `[-90, 90]`. All angles are in units of degrees. If provided and not empty, a given layer appears in the results only if its limits fully contain the specified `'Latlim'` limits. Partial overlap does not result in a match.

Data Types: `double` | `int16` | `int32` | `int64` | `int8` | `single` | `uint8` | `uint16` | `uint32` | `uint64`

Lonlim — Longitudinal limits of the search
two-element numeric vector | numeric scalar

Longitudinal limits of the search, specified as a two-element vector of the form `[western_limit eastern_limit]` or scalar value representing the longitude of a single point. All angles are in units of degrees. If provided and not empty, a given layer appears in the results only if its limits fully contain the specified `'Lonlim'` limits. Partial overlap does not result in a match.

Data Types: `double` | `int16` | `int32` | `int64` | `int8` | `single` | `uint8` | `uint16` | `uint32` | `uint64`

MatchType — Strictness of match
`'partial'` (default) | `'exact'`

Strictness of match, specified as the character vector or string `'partial'` or `'exact'`. If `'MatchType'` is `'exact'` and `querystr` is `'*'`, a match occurs when the search field matches the character `'*'`.

Data Types: `char` | `string`

SearchFields — Fields to search in the WMS Database
`'layer'` (default) | `'server'` | `'layertitle'` | `'layername'` | `'servertitle'` | `'serverurl'` | `'any'`

Fields to search in the WMS Database, specified as a character vector, cell array of character vectors, string, or array of strings. The function searches the values of the specified fields in the WMS Database for a partial or exact match with `querystr`. The following table lists valid values.

Field	Behavior
`'layername'`	Search the `LayerName` field in the WMS Database. The layer name is an abbreviated form of the `LayerTitle` field and is the keyword the server uses to retrieve the layer.
`'layertitle'`	Search the `LayerTitle` field in the WMS Database. The layer title includes descriptive information about a layer and facilitates understanding the meaning of the raster values of the layer.
`'layer'`	Search both the `LayerTitle` and the `LayerName` fields.
`'servertitle'`	Search the `ServerTitle` field in the WMS Database. A server title includes descriptive information about the server.
`'serverurl'`	Search the `ServerURL` field in the WMS Database. The server URL and layer information facilitate the reading of raster layers by the function `wmsread`.
`'server'`	Search both the `ServerTitle` and the `ServerURL` fields.
`'any'`	Search all fields.

Data Types: `char` | `cell` | `string`

Version — Version of the WMS Database to read
`'installed'` (default) | `'online'` | `'custom'`

Version of the WMS Database to read, specified as one of the following values.

Value	Description
`'installed'`	Read from the installed database. This is the default. The information found in the installed database is static and is not automatically updated—it was validated at the time of the software release.
`'online'`	Read from the version of the database hosted on the MathWorks website. This version of the database contains more up-to-date information about servers because it is updated regularly. Note, however, that searching this online database requires a connection to the Internet. If your network uses a firewall or another method of protection that restricts Internet access, you can provide information about your proxy server to MATLAB. For more information, see "Proxy Server Authentication".
`'custom'`	Read from the `wmsdatabase.mat` file on the MATLAB path, such as from a previous release.

Data Types: `char` | `string`

Output Arguments

`layers` — Layers that match search criteria
`WMSLayer` objects

Layers that match search criteria, returned as an array of `WMSLayer` objects, one for each layer found.

More About

WMS Database

Mapping Toolbox contains a database of over 1000 stored WMS servers and over 100,000 layers. MathWorks creates this database, called the WMS Database, by conducting a series of Internet searches and qualifying the search results.

Note MathWorks does not guarantee the stability or accuracy of WMS data. The servers listed in the WMS Database are located on the Internet and are independent from MathWorks. Occasionally, you might receive error messages from servers experiencing difficulties. Servers might go down or become unavailable.

The `wmsfind` function is the only function that accesses the stored WMS Database. By default, the `wmsfind` function searches the WMS Database that is installed with the product. You can search a version of the WMS Database hosted on the MathWorks website or a WMS Database from a previous release by using the `Version` name-value argument. The information in the WMS Database installed with the product is static. The web-hosted database is updated regularly.

The WMS Database contains these fields:

Field Name	Data Type	Field Content
ServerTitle	Character vector	Title of the WMS server, descriptive information about the server
ServerURL	Character vector	URL of the WMS server
LayerTitle	Character vector	Title of the layer, descriptive information about the layer
LayerName	Character vector	Name of the layer, keyword the server uses to retrieve the layer
Latlim	Two-element vector	Southern and northern latitude limits of the layer
Lonlim	Two-element vector	Western and eastern longitude limits of the layer

The `LayerTitle` and `LayerName` fields sometimes have the same values. The `LayerName` field indicates a code used by the servers, such as `'29:2'`, while the `LayerTitle` field provides more descriptive information. For example, `'Elevation and Rivers with Backdrop'` might be an entry in the `LayerTitle` field.

Tips

- The `WMSLayer` objects returned by `wmsfind` contain properties with the same names as the field names of the WMS Database, along with three additional properties: `'Abstract'`, `'CoordRefSysCodes'`, and `'Details'`. The WMS Database does not contain information about these properties. To get this information about a server, you must use the `wmsupdate` function which updates these properties of the `WMSLayer` object by downloading information from the server. To view these properties, use the `WMSLayer.disp` method, specifying the `'Properties'` parameter value `'all'`. If you want to know more about a WMS server, use the `wmsinfo` function with the specific server URL.

See Also

`wmsupdate` | `wmsinfo` | `wmsread` | `WebMapServer` | `WMSLayer`

Topics
"Basic Workflow for Creating WMS Maps"

Introduced in R2009b

wmsinfo

Information about WMS server from capabilities document

Syntax

```
[capabilities,infoRequestURL] = wmsinfo(serverURL)
[capabilities,infoRequestURL] = wmsinfo(infoRequestURL)
[capabilities,infoRequestURL] = wmsinfo( ___ ,'TimeoutInSeconds',sec)
```

Description

`[capabilities,infoRequestURL] = wmsinfo(serverURL)` accesses the Internet to read a capabilities document from a Web Map Service (WMS) server specified by string or character vector `serverURL`. A capabilities document is an XML document that contains metadata describing the geographic content offered by the server.

The `wmsinfo` function returns the contents of the capabilities document in `capabilities`, a `WMSCapabilities` object. The `wmsinfo` function also returns the character vector `infoRequestURL`, which is composed of the `serverURL` with additional WMS parameters. You can insert `infoRequestURL` into a browser, or the `urlread` function, to get the XML capabilities document.

The `wmsinfo` function requires an Internet connection. WMS servers can periodically be unavailable. Retrieving the map can take several minutes.

`[capabilities,infoRequestURL] = wmsinfo(infoRequestURL)` reads the capabilities document from a WMS `infoRequestURL` and returns the contents in `capabilities`.

`[capabilities,infoRequestURL] = wmsinfo( ___ ,'TimeoutInSeconds',sec)` specifies the number of seconds before a server times out. Specify `sec` as a non-negative integer. If you specify `sec` as 0, then `wmsinfo` ignores the time-out mechanism.

Examples

Read Capabilities Document and Display Abstract

Read the capabilities document from the NASA Goddard Space Flight Center WMS server.

```
serverURL = 'http://svs.gsfc.nasa.gov/cgi-bin/wms?';
capabilities = wmsinfo(serverURL);
```

Display information about the first layer.

```
capabilities.Layer(1)

ans =
  WMSLayer

  Properties:
          Index: 1
```

```
      ServerTitle: 'NASA SVS Image Server'
        ServerURL: 'http://svs.gsfc.nasa.gov/cgi-bin/wms?'
       LayerTitle: 'African Fires During 2002 (1024x1024 Animation)'
        LayerName: '2890_17402'
           Latlim: [-39.0000 41.0000]
           Lonlim: [-22.0000 58.0000]
         Abstract: 'This animation shows fire activity in Africa from January 1, 2002 to December

Additional Credit:
B>Please give credit for this item to:</b><br />'
CoordRefSysCodes: {'CRS:84'}
          Details: [1x1 struct]

  Methods
```

Refine the list to include only layers with the term "glacier retreat" in the layer title.

```
glaciers = capabilities.Layer.refine('glacier retreat', ...
    'SearchFields','LayerTitle');
```

Display the abstract of the first layer.

```
glaciers(1).Abstract
```

```
ans =
    'Since measurements of Jakobshavn Isbrae were first taken in 1850, the glacier has gradually

    Additional Credit:
    B>Please give credit for this item to:</b><br />'
```

Input Arguments

serverURL — WMS server URL
string scalar | character vector

WMS server URL, specified as a string scalar or character vector. The serverURL contains the
protocol 'http://' or 'https://' and additional WMS or access keywords.

Data Types: char | string

Output Arguments

capabilities — Capabilities document
WMSCapabilities object

Capabilities document, returned as a WMSCapabilities object.

infoRequestURL — URL composed of serverURL with additional WMS parameters
string | character vector

URL composed of the serverURL with additional WMS parameters, returned as a character vector.

Tips

- To specify a proxy server to connect to the Internet, select **File>Preferences>Web** and enter your proxy information. Use this feature if you have a firewall.

- `wmsinfo` communicates with the server using a `WebMapServer` object representing an implementation of a WMS specification. The object acts as a proxy to a WMS server and resides physically on the client side. The object accesses the server's capabilities document. The object supports multiple WMS versions and negotiates with the server to use the highest known version that the server can support. The object automatically times-out after 60 seconds if a connection is not made to the server.

See Also

`wmsfind` | `wmsupdate` | `wmsread` | `WebMapServer` | `WMSLayer`

Topics
"Basic Workflow for Creating WMS Maps"

Introduced in R2009b

WMSLayer

Web Map Service layer

Description

A `WMSLayer` object describes a Web Map Service (WMS) layer or layers.

Creation

You can create a `WMSLayer` object using any of the following methods:

- `wmsfind` — Returns a `WMSLayer` array.
- `wmsinfo` — Returns a `WMSCapabilities` object, which contains an array of `WMSLayer` objects in its `Layer` property.
- The `WMSLayer` object creation function, described here.

Syntax

```
layers = WMSLayer(Name,Value)
```

Description

`layers = WMSLayer(Name,Value)` constructs a `WMSLayer` object, where `Name` is the name of any property of the `WMSLayer` and `Value` is the value that you want to assign to the property. You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`. The size of the output `layers` is scalar unless all inputs are cell arrays, in which case, the size of `layers` matches the size of the cell arrays.

Properties

ServerTitle — Descriptive information about the server
empty character vector (`''`) (default) | character vector

Descriptive information about the server, specified as a character vector.

Data Types: `char`

ServerURL — URL of WMS Server
empty character vector (`''`) (default) | character vector

URL of WMS Server, specified as a character vector.

Data Types: `char`

LayerTitle — Descriptive information about the layer
empty character vector (`''`) (default) | character vector

Descriptive information about the layer, specified as a character vector. The `LayerTitle` clarifies the meaning of the raster values of the layer.

Data Types: `char`

LayerName — Keyword the server uses to retrieve the layer
empty character vector (`' '`) (default) | character vector

Keyword the server uses to retrieve the layer, specified as a character vector.

Data Types: `char`

Latlim — Latitude limits of the layer in units of degrees
`[]` (default) | two-element numeric vector

Latitude limits of the layer in units of degrees, specified as a two-element numeric vector. The limits specify the southern and northern latitude limits and must be in units of degrees and in the range [-90, 90].

Data Types: `double`

Lonlim — Longitude limits of the layer in units of degrees
`[]` (default) | two-element numeric vector

Longitude limits of the layer in units of degrees, specified as a two-element numeric vector. The limits specify the western and eastern longitude limits and must be ascending and in the range [-180, 180] or [0, 360].

Data Types: `double`

Abstract — Information about the layer
empty character vector (`' '`) (default) | character vector

Information about the layer, specified as a character vector.

Data Types: `char`

CoordRefSysCodes — Codes identifying available coordinate reference systems
empty cell array `{}` (default) | cell array of character vectors

Codes identifying available coordinate reference systems, specified as a cell array of character vectors.

Data Types: `cell`

Details — Detailed information about the layer
structure

Detailed information about the layer, specified as a structure containing: URL of metadata, attributes, scale, dimension, and style. See the `WMSLayer.Details` on page 1-1555 reference page for more information.

Data Types: `struct`

Object Functions

disp   Display properties of WMS layers or capabilities
refine   Refine search of WMS layers

refineLimits Refine search of WMS layers based on geographic limits
servers Return URLs of unique WMS servers
serverTitles Return titles of unique WMS servers

Examples

Construct WMSLayer Object from WMS GetMap Request URL

Specify the server URL. These values are typically found during an Internet search. The WMSLayer `ServerURL` value is obtained from the host and path of the request URL. The WMSLayer `LayerName` value is obtained from the `LAYERS` value in the query part of the URL.

```
host = 'www.mrlc.gov';
path = '/geoserver/NLCD_Land_Cover/wms?';
serverURL = ['https://' host path];
requestURL = [serverURL 'SERVICE=WMS&FORMAT=image/jpeg&REQUEST=GetMap&' ...
  'STYLES=&SRS=EPSG:4326&VERSION=1.1.1&LAYERS=mrlc_display:NLCD_2016_Land_Cover_L48&', ...
  'WIDTH=1024&HEIGHT=470&BBOX=-128,23,-65,51'];
layerName = 'mrlc_display:NLCD_2016_Land_Cover_L48';
```

Construct the `WMSLayer` object by using the `serverURL` variable and the value of the WMS `LAYERS` parameter.

```
layer = WMSLayer('ServerURL',serverURL,'LayerName',layerName);
```

Use the `wmsupdate` function to get the other properties of the WMSLayer array from the server.

```
layer = wmsupdate(layer);
layer.Lonlim = [-180 180];
```

Retrieve an image from the WMS server using layer and parameter values from the WMS `GetMap` request URL. Set latitude and longitude limits from the `BBOX` request value. Set image height and width values from the `WIDTH` and `HEIGHT` request values.

```
lonlim = [-128 -65];
latlim = [23 51];
height = 470;
width = 1024;
[A,R] = wmsread(layer,'Latlim',latlim,'Lonlim',lonlim, ...
        'ImageHeight',height,'ImageWidth',width);
```

Display the image from the server.

```
figure
usamap(A,R)
geoshow(A,R)
title('NLCD Land Cover')
```

NLCD Land Cover
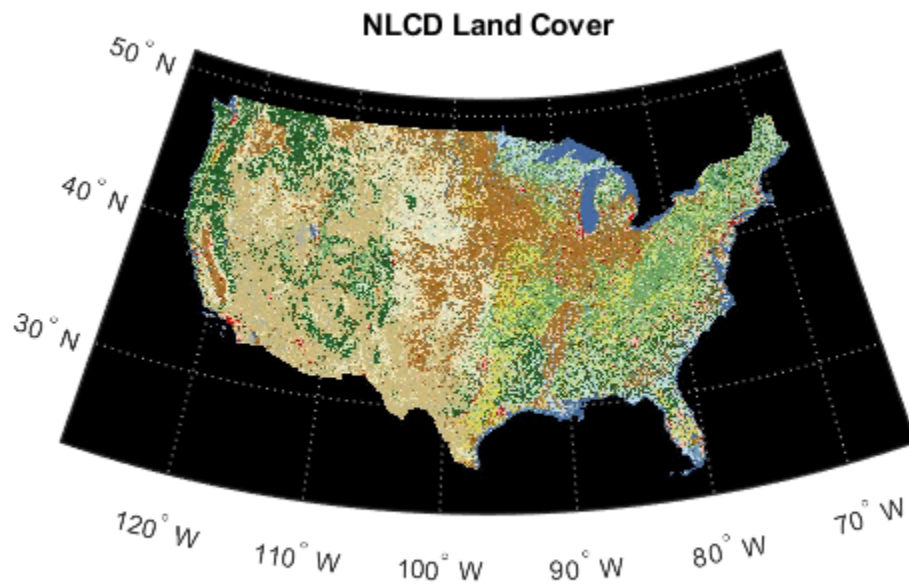
This image can also be retrieved using the WMS `requestURL`.

```
[A,R] = wmsread(requestURL);
```

See Also

Functions
wmsfind | wmsinfo | wmsread | wmsupdate

Objects
WebMapServer | WMSCapabilities | WMSMapRequest

Introduced in R2009b

Details property

Detailed information about WMS layer

Description

A structure containing detailed information about a layer

Details Structure

Field Name	Data Type	Field Content
MetadataURL	Character vector	URL containing metadata information about layer.
Attributes on page 1-1555	Structure	Attributes of layer.
BoundingBox on page 1-1556	Structure array	Bounding box of layer.
Dimension on page 1-1556	Structure array	Dimensional parameters of layer, such as time or elevation.
ImageFormats	Cell array	Image formats supported by server.
ScaleLimits on page 1-1557	Structure	Scale limits of layer.
Style on page 1-1557	Structure array	Style parameters that determine layer rendering.
Version	Character vector	WMS version specification.

Attributes Structure

Field Name	Data Type	Field Content
Queryable	Logical	True if you can query the layer for feature information.
Cascaded	Double	Number of times a Cascading Map server has retransmitted the layer.
Opaque	Logical	True if the map data are mostly or completely opaque.
NoSubsets	Logical	True if the map must contain the full bounding box. false if the map can be a subset of the full bounding box.
FixedWidth	Logical	True if the map has a fixed width that the server cannot change. false if the server can resize the map to an arbitrary width.
FixedHeight	Logical	True if the map has a fixed height that the server cannot change. false if the server can resize the map to an arbitrary height.

BoundingBox Structure

Field Name	Data Type	Field Content
CoordRefSysCode	character vector	Code number for coordinate reference system.
XLim	Double array	X limit of layer in units of coordinate reference system.
YLim	Double array	Y limit of layer in units of coordinate reference system.

Dimension Structure

Field Name	Data Type	Field Content
Name	Character vector	Name of the dimension; such as time, elevation, or temperature.
Units	Character vector	Measurement unit.
UnitSymbol	Character vector	Symbol for unit.
Default	Character vector	Default dimension setting. For example, if default is 'time' and dimension is not specified, server returns time holding.
MultipleValues	Logical	True if multiple values of the dimension may be requested. false if only single values may be requested.
NearestValue	Logical	True if nearest value of dimension is returned in response to request for nearby value. false if request value must correspond exactly to declared extent values.
Current	Logical	True if temporal data are kept current (valid only for temporal extents).
Extent	Character vector	Values for dimension. Expressed in any of the following ways: • Single value (value) • List of values (value1, value2, ...) • Interval defined by bounds and resolution (min1/max1/res1) • List of intervals (min1/max1/res1, min2/max2/res2, ...)

ScaleLimits Structure

Field Name	Data Type	Field Content
ScaleHint	Double array	Minimum and maximum scales for which it is appropriate to display layer. Expressed as scale of ground distance in meters represented by diagonal of central pixel in image.
MinScaleDenominator	Double	Minimum scale denominator of maps for which a layer is appropriate.
MaxScaleDenominator	Double	Maximum scale denominator of maps for which a layer is appropriate.

Style Structure Array

Field Name	Data Type	Field Content
Title	Character vector	Descriptive title of style.
Name	Character vector	Name of style.
Abstract	Character vector	Information about style.
LegendURL on page 1-1557	Structure	Information about legend graphics.

LegendURL Structure

Field Name	Data Type	Field Content
OnlineResource	Character vector	URL of legend graphics.
Format	Character vector	Format of legend graphics.
Height	Double	Height of legend graphics.
Width	Double	Width of legend graphics.

WMSMapRequest

Web Map Service map request

Description

A `WMSMapRequest` object contains a request to a WMS server to obtain a map, which represents geographic information. The WMS server renders the map as a color or grayscale image. The object contains properties that you can set to control the geographic extent, rendering, or size of the requested map.

Creation

Syntax

```
mapRequest = WMSMapRequest(layer)
mapRequest = WMSMapRequest(layer,server)
```

Description

`mapRequest = WMSMapRequest(layer)` creates a `WMSMapRequest` object, setting the `Layer` property. The `WMSMapRequest` object updates the properties of `Layer`, if necessary.

`mapRequest = WMSMapRequest(layer,server)` creates a `WMSMapRequest` object, setting the `Layer` and `Server` properties. The `ServerURL` property of `layer` must match the `ServerURL` property of `server`. The `Server` object updates `Layer` properties.

Properties

Server — Web map server
scalar `WebMapServer` object

Web map server, specified as a scalar `WebMapServer` object. If a server is not supplied as an argument when creating the `WMSMapRequest`, the value of `Server` is set to the `ServerURL` of `Layer`.

Layer — Web Map Service layers
array of `WMSLayer` objects

Web Map Service layers, specified as an array of `WMSLayer` objects.

`Layer` contains one unique `ServerURL`, which must match the `ServerURL` property of `Server`. The `Server` property updates the properties of `Layer` when the property is set.

CoordRefSysCode — Coordinate reference system code
`'CRS:84'` | `'EPSG:4326'`

Coordinate reference system code, specified as the character vector `'CRS:84'` for WMS version 1.3.x, and `'EPSG:4326'` for all other versions.

- If `'EPSG:4326'` is not found in `Layer.CoordRefSysCodes`, then the `CoordRefSysCode` value is set from the first `CoordRefSysCode` found in the `Layer.Details.BoundingBox` structure array.
- When `CoordRefSysCode` is set to `'EPSG:4326'` or `'CRS:84'`, the `XLim` and `YLim` properties are set to `[]` and the `Latlim` and `Lonlim` properties are set to the geographic extent defined by the `Layer` array.
- When `CoordRefSysCode` is set to a value other than `'EPSG:4326'` or `'CRS:84'`, then the `XLim` and `YLim` properties are set from the values found in the `Layer.Details.BoundingBox` structure and the `Latlim` and `Lonlim` properties are set to `[]`.
- Automatic projections are not supported. (Automatic projections begin with `'AUTO'`.)

Data Types: `char`

RasterReference — Map or geographic raster reference
`MapCellsReference` or `GeographicCellsReference`

Map or geographic raster reference, specified as a `MapCellsReference` or `GeographicCellsReference` object. `RasterReference` references the raster map to an intrinsic coordinate system

Latlim — Latitude limits
two-element vector

Latitude limits, specified as a two-element vector. `Latlim` contains the southern and northern latitudinal limits of the request in units of degrees. The limits must be ascending. By default, the latitude limits span all latitudinal limits found in the `Layer.Latlim` property.

Lonlim — Longitude limits
two-element vector

Longitude limits, specified as a two-element vector. `Lonlim` contains the western and eastern longitudinal limits of the request in units of degrees. The limits must be ascending and in the range [-180, 180] or [0, 360]. By default, the longitude limits span all longitudinal limits found in the `Layer.Lonlim` property.

XLim — Western and eastern limits in the units of the coordinate reference system
`[]` (default) | two-element vector

Western and eastern limits of the requested map in the units of the coordinate reference system, specified as a two-element vector. The limits must be ascending. You can set `XLim` only if you set `CoordRefSysCode` to a value other than `EPSG:4326`.

YLim — Southern and northern in the units of the coordinate reference system
`[]` (default) | two-element vector

Southern and northern limits of the requested map in the units of the coordinate reference system, specified as a two-element vector. The limits must be ascending. You can set `YLim` only if you set `CoordRefSysCode` to a value other than `EPSG:4326`.

ImageHeight — Height in pixels for the requested raster map
positive integer

Height in pixels for the requested raster map, specified as a positive integer. The property `MaximumHeight` defines the maximum value for `ImageHeight`. The `WMSMapRequest` object

initializes the `ImageHeight` property to either 512 or to an integer value that best preserves the aspect ratio of the coordinate limits, without changing the coordinate limits.

`ImageWidth` — Width in pixels for the requested raster map
positive integer

Width in pixels for the requested raster map, specified as a positive integer. The property `MaximumWidth` defines the maximum value for `ImageWidth`. The `WMSMapRequest` object initializes the `ImageWidth` property to either 512 or to an integer value that best preserves the aspect ratio of the coordinate limits, without changing the coordinate limits.

`MaximumHeight` — Maximum height in pixels of requested map
8192

This property is read-only.

Maximum height in pixels for the requested map, specified as the number 8192.

Data Types: `double`

`MaximumWidth` — Maximum width in pixels of requested map
8192

This property is read-only.

Maximum width in pixels for the requested map, specified as the number 8192.

Data Types: `double`

`Elevation` — Elevation extent of requested map
`''` (default) | character vector

Elevation extent of the requested map, specified as a character vector. When you set the property, `'elevation'` must be the value of the `Layer.Details.Dimension.Name` field.

`Time` — Time extent of requested map
character vector | numeric scalar

Time extent of the requested map, specified as a character vector or numeric scalar. See the `WMSMapRequest.Time` on page 1-1566 reference page for more information.

Data Types: `double` | `char`

`SampleDimension` — Name and value of a sample dimension
two-element cell array of character vectors

Name and value of a sample dimension, specified as a two-element cell array of character vectors. The name cannot be `'time'` or `'elevation'`. `SampleDimension{1}` must be the value of the `Layer.Details.Dimension.Name` field.

`Transparent` — Flag indicating transparency of map background
false (default) | logical scalar

Flag indicating transparency of map background, specified as a logical scalar. When you set `Transparent` to `true`, the server sets all pixels not representing features or data values in that layer to a transparent value, producing a composite map. When you set `Transparent` to `false`, the server sets all non-data pixels to the value of the background color.

Data Types: `logical`

BackgroundColor — Color of the background (non-data) pixels of the map
three-element numeric vector

Color of the background (non-data) pixels of the map, specified as a three-element numeric vector. The values range from 0 to 255. The default value, `[255,255,255]`, specifies the background color as white. You can set `BackgroundColor` using non-`uint8` numeric values, but they are cast and stored as `uint8`.

Data Types: `uint8`

StyleName — Style to use when rendering the image
`{}` (default) | character vector or cell array of character vectors

Style to use when rendering the image, specified as a character vector or cell array of character vectors. The `StyleName` must be a valid entry in the `Layer.Details.Style.Name` field. The cell array of character vectors contains the same number of elements as does `Layer`.

ImageFormat — Desired image format used to render the map as an image
character vector

Desired image format used to render the map as an image, specified as a character vector. If set, the format must match an entry in the `Layer.Details.ImageFormats` cell array and an entry in the `ImageRenderFormats` property. If not set, the format defaults to a value in the `ImageRenderFormats` property.

ImageRenderFormats — Preferred image rendering formats when `Transparent` is set to `false`
cell array

This property is read-only.

Preferred image rendering formats when `Transparent` is set to `false`, specified as a cell array. The first entry is the most preferred image format. If the preferred format is not stored in the `Layer` property, then the next format from the list is selected, until a format is found. The `ImageRenderFormats` array is not used if the `ImageFormat` property is set.

ImageTransparentFormats — Preferred image rendering formats when `Transparent` is set to `true`
cell array

This property is read-only.

Preferred image rendering formats when `Transparent` is set to `true`, specified as a cell array. The first entry is the most preferred image format. If the preferred format is not stored in the `Layer` property, then the next format from the list is selected, until a format is found. If a transparent image format is not found in the list, or if the `ImageFormat` property is set to a non-default value, then `ImageFormat` is unchanged.

ServerURL — Server URL for the WMS `GetMap` request
character vector

Server URL for the WMS `GetMap` request, specified as a character vector. In general, `ServerURL` matches the `ServerURL` of the `Layer`. However, some WMS servers, such as the Microsoft®

TerraServer, require a different URL for `GetMap` requests than for WMS `GetCapabilities` requests. By default, `ServerURL` is `Layer(1).ServerURL`.

Data Types: `char`

RequestURL — Full URL for the WMS GetMap request
character vector

This property is read-only.

Full URL for the WMS `GetMap` request, specified as a character vector. It is composed of the `ServerURL` with additional WMS parameter/value pairs.

Object Functions
boundImageSize   Bound size of raster map

Examples

Read and Display Map of Global Sea-Surface Temperature

Read a global, half-degree resolution, sea-surface temperature map for the month of November 2009. The map, from the AMSR-E sensor on NASA's Aqua satellite, uses data provided by NASA's Earth Observations (NEO) WMS server.

```
sst = wmsfind('AMSRE_SSTAn_M');
sst = sst(1);
server = WebMapServer(sst.ServerURL);
mapRequest = WMSMapRequest(sst, server);
timeRequest = '2009-11-01';
mapRequest.Time = timeRequest;
samplesPerInterval = .5;
mapRequest.ImageHeight = ...
   round(abs(diff(sst.Latlim))/samplesPerInterval);
mapRequest.ImageWidth = ...
   round(abs(diff(sst.Lonlim))/samplesPerInterval);
mapRequest.ImageFormat = 'image/png';
sstImage = server.getMap(mapRequest.RequestURL);
```

Read the legend for the layer using the `OnlineResource` URL field in the `LegendURL` structure. The legend shows that the temperature ranges from -2 to 35 degrees Celsius. The `WMSMapRequest` object updates the layer information from the server.

```
url = mapRequest.Layer.Details.Style(1).LegendURL.OnlineResource;
[legendImg,cmap] = imread(url);
if ~isempty(cmap)
   % Convert indexed image to RGB.
   legendRGB = ind2rgb(legendImg,cmap);
else
   % Already have an RGB image.
   legendRGB = legendImg;
end
```

Display the temperature map and legend.

```
fig = figure;
ax = worldmap('world');
```

```
setm(ax,'MlabelParallel',-90,'MlabelLocation',90)
    geoshow(sstImage,mapRequest.RasterReference);
    title({mapRequest.Layer.LayerTitle,timeRequest})

figurePosition = fig.Position;
centerWidth = figurePosition(3)/2;
axleft = centerWidth-size(legendImg,2)/2;
axbottom = 30;
axwidth  = size(legendRGB,2);
axheight = size(legendRGB,1);
axes('Units','pixels','Position',[axleft axbottom axwidth axheight])
image(legendRGB)
axis off
```
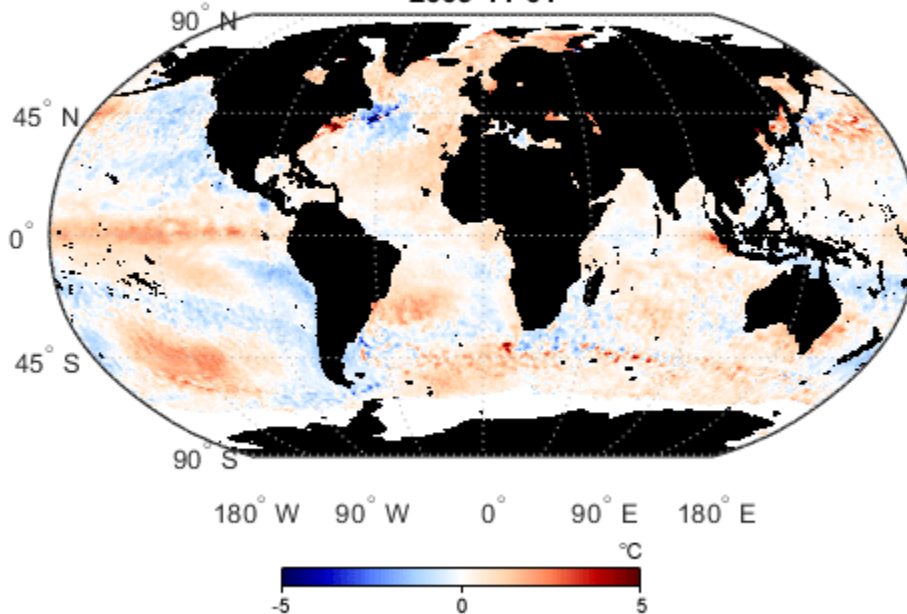

Read abstract information for this layer from the `MetadataURL` field.

```
readerFcn = @(filename)readstruct(filename,'FileType','xml');
options = weboptions('ContentReader',readerFcn,'Timeout',10);
xml = webread(mapRequest.Layer.Details.MetadataURL,options);
abstract = xml.idinfo.descript.abstract;
```

Read and Display Map of Global Elevation and Bathymetry

Read a global elevation and bathymetry layer for the Gulf of Maine at a 30 arc-second sampling interval. The values are in units of meters.

```
layers = wmsfind('worldwind26*elev','SearchField','serverurl');
layers = wmsupdate(layers);
layer = refine(layers,'SRTM','SearchField','layername');
server = WebMapServer(layer.ServerURL);
mapRequest = WMSMapRequest(layer,server);
mapRequest.Latlim = [40 46];
mapRequest.Lonlim = [-71 -65];
samplesPerInterval = 30/3600;
mapRequest.ImageHeight = ...
    round(abs(diff(mapRequest.Latlim))/samplesPerInterval);
mapRequest.ImageWidth = ...
    round(abs(diff(mapRequest.Lonlim))/samplesPerInterval);
mapRequest.ImageFormat = 'image/tiff';
Z = getMap(server, mapRequest.RequestURL);
R = mapRequest.RasterReference;
```

Display the map and plot contour lines at sea level. Apply a colormap that is appropriate for elevation data. Add a colorbar and title.

```
figure
worldmap(mapRequest.Latlim,mapRequest.Lonlim)
geoshow(double(Z),R,'DisplayType','texturemap')
contourm(double(Z),R,[0,0],'Color','k')
demcmap(double(Z))
colorbar
title({'Gulf of Maine',mapRequest.Layer.LayerTitle}, ...
       'Interpreter','none','FontWeight','bold')
```
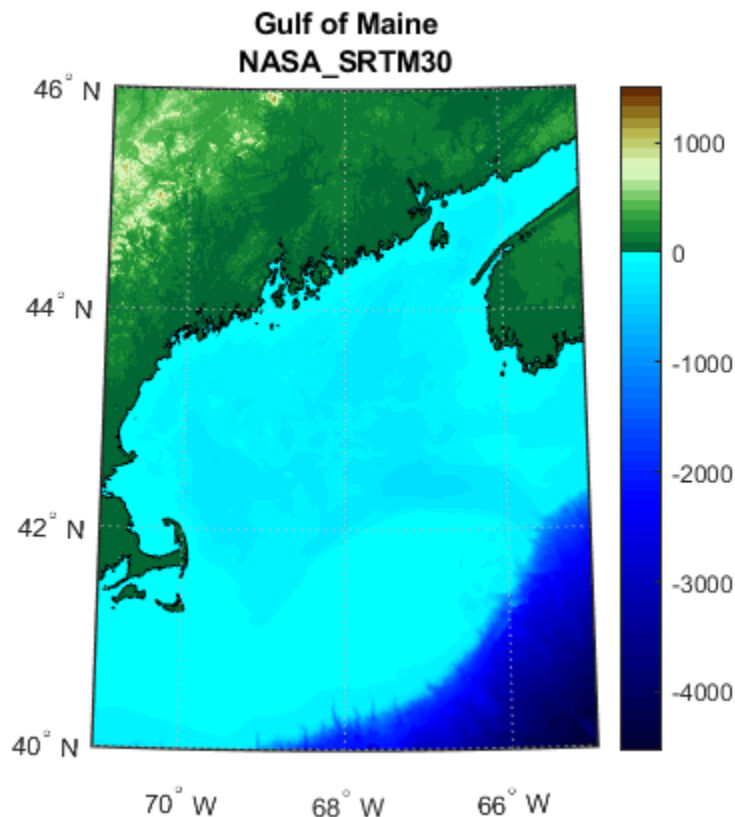
See Also

Functions
wmsread | wmsinfo | wmsfind

Objects
WebMapServer | WMSCapabilities | WMSLayer

Introduced in R2009b

Time property

Requested time extent

Description

The `WMSMapRequest.Time` property stores time as a character vector or a `double` indicating the desired time extent of the requested map. When you set the property, `'time'` must be the value of the `Layer.Details.Dimension.Name` field. The default value is `''`.

Time is stored in the ISO® 8601:1988(E) extended format. In general, the `Time` property is stored in a `yyyy-mm-dd` format or a `yyyy-mm-ddThh:mm:ssZ` format, if the precision requires hours, minutes, or seconds. You can use several different character vector and numeric formats to set the `Time` property, according to the following table (where dateform number is the number used by the Standard MATLAB Date Format Definitions). Express all hours, minutes, and seconds in Coordinated Universal Time (UTC).

Dateform (number)	Input (character vector)	Stored format
0	dd-mm-yyyy HH:MM:SS	yyyy-mm-ddTHH:MM:SSZ
1	dd-mm-yyyy	yyyy-mm-dd
2	mm/dd/yy	yyyy-mm-dd
6	mm/dd	yyyy-mm-dd (current year)
10	yyyy	yyyy
13	HH:MM:SS	yyyy-mm-ddTHH:MM:SSZ
14	HH:MM:SS PM	yyyy-mm-ddTHH:MM:SSZ
15	HH:MM	yyyy-mm-ddTHH:MM:00Z
16	HH:MM PM	yyyy-mm-ddTHH:MM:00Z
21	mmm.dd,yyyy HH:MM:SS	yyyy-mm-ddTHH:MM:SSZ
22	mmm.dd,yyyy	yyyy-mm-dd
23	mm/dd/yyyy	yyyy-mm-dd
26	yyyy/mm/dd	yyyy-mm-dd
29	yyyy-mm-dd	yyyy-mm-dd
30	yyyymmddTHHMMSS	yyyy-mm-ddTHH:MM:SSZ
31	yyyy-mm-dd HH:MM:SS	yyyy-mm-ddTHH:MM:SSZ

Inputs using the dateform numbers 13–16 return the date set to the current year, month, and day. Use of other dateform formats, especially 19, 20, 24, and 25, results in erroneous output.

In addition to these standard MATLAB dateform formats, the `WMSMapRequest.Time` property also accepts the following inputs.

Input (character vector)	Description
'current'	The current time holdings of the server

Input (character vector)	Description
`numeric datenum`	Numeric date value converted to yyyy-mm-dd (dateform 29 format)
`Byyyy`	B.C.E. year

Use the prefixes K, M, and G, followed by a character vector number (thousand, million, and billion years, respectively), for geologic data sets that refer to the distant past.

wmsread

Retrieve WMS map from server

Syntax

```
[A,R] = wmsread(layer)
[A,R] = wmsread(layer,Name,Value,...)
[A,R] = wmsread(mapRequestURL)
[A,R,mapRequestURL] = wmsread(...)
```

Description

`[A,R] = wmsread(layer)` accesses the Internet to render and retrieve a raster map from a Web Map Service (WMS) server. The `ServerURL` property of the `WMSLayer` object, `layer`, specifies the server. If `layer` has more than one element, then the server overlays each subsequent layer on top of the base (first) layer, forming a single image. The server renders multiple layers only if all layers share the same `ServerURL` value.

The WMS server returns a raster map, either a color or grayscale image, in the output A. The second output, R, is a raster reference object that ties A to the EPSG:4326 geographic coordinate system on page 1-1577. The rows of A are aligned with parallels, with even sampling in longitude. Likewise, the columns of A are aligned with meridians, with even sampling in latitude.

The geographic limits of A span the full latitude and longitude extent of `layer`. The `wmsread` function chooses the larger spatial size of A to match its larger geographic dimension. The larger spatial size is fixed at the value 512. In other words, assuming RGB output, A is `512-by-N-by-3` if the latitude extent exceeds longitude extent and `N-by-512-by-3` otherwise. In both cases N <= 512. The `wmsread` function sets N to the integer value that provides the closest possible approximation to equal cell sizes in latitude and longitude. The map spans the full extent supported for the `layer`.

`[A,R] = wmsread(layer,Name,Value,...)` specifies parameter-value pairs that modify the request to the server. You can abbreviate parameter names, which are case-insensitive.

`[A,R] = wmsread(mapRequestURL)` uses the input argument `mapRequestURL` to define the request to the server. The `mapRequestURL` contains a WMS `serverURL` with additional WMS parameters. These WMS parameters include BBOX, GetMap and the `EPSG:4326` or `CRS:84` keyword. Obtain a `mapRequestURL` from the output of `wmsread`, the `RequestURL` property of a `WMSMapRequest` object, or an Internet search.

`[A,R,mapRequestURL] = wmsread(...)` returns a WMS `GetMap` request URL in the character vector `mapRequestURL`. You can insert the `mapRequestURL` into a browser to make a request to a server, which then returns the raster map. The browser opens the returned map if its mime type is understood, or saves the raster map to disk.

Examples

Read and Display Layer from NASA

Search the WMS database for layers containing the string "NASA". Refine the search to find layers containing Blue Marble: Next Generation by specifying the search string as "bluemarbleng".

```
nasa = wmsfind('NASA','SearchField','serverurl');
layer = refine(nasa,'bluemarbleng', ...
    'SearchFields','layername', ...
    'MatchType', 'exact');
```

Read the first layer and display the map.

```
[A,R] = wmsread(layer(1));
```

```
axesm globe
axis off
geoshow(A,R)
title('Blue Marble')
```


Blue Marble
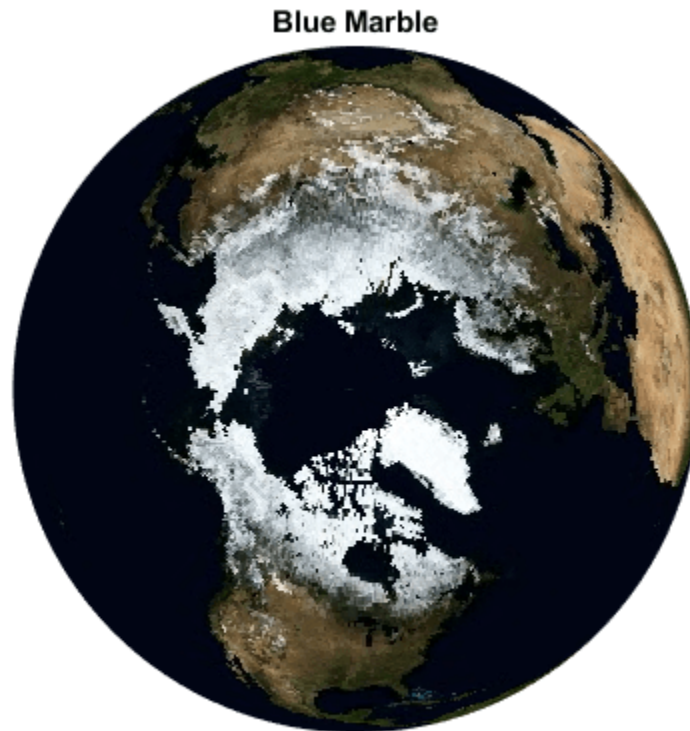
Read and Display Orthoimage

Read and display an orthoimage of the northern section of the Golden Gate Bridge in San Francisco, California, using the USGS National Map Seamless server.

First, specify the latitude and longitude limits of the Golden Gate Bridge.

```
latlim = [37.78 37.84];
lonlim = [-122.53 -122.40];
```

Then, find the USGS high-resolution orthoimagery layer by reading the capabilities document from the server. The server may be busy, so try to connect multiple times.

```
numberOfAttempts = 5;
attempt = 0;
info = [];
serverURL = 'http://basemap.nationalmap.gov/ArcGIS/services/USGSImageryOnly/MapServer/WMSServer?
while(isempty(info))
    try
        info = wmsinfo(serverURL);
        orthoLayer = info.Layer(1);
    catch e

        attempt = attempt + 1;
        if attempt > numberOfAttempts
            throw(e);
        else
            fprintf('Attempting to connect to server:\n"%s"\n', serverURL)
        end
    end
end
```

Retrieve the map from the server and display it in a UTM projection.

```
imageLength = 1024;
[A,R] = wmsread(orthoLayer,'Latlim',latlim, ...
                          'Lonlim',lonlim, ...
                          'ImageHeight',imageLength, ...
                          'ImageWidth',imageLength);


axesm('utm', ...
      'Zone',utmzone(latlim, lonlim), ...
      'MapLatlimit', latlim, ...
      'MapLonlimit', lonlim, ...
      'Geoid', wgs84Ellipsoid)
geoshow(A,R)
axis off
title({'San Francisco','Northern Section of Golden Gate Bridge'})
```

San Francisco
Northern Section of Golden Gate Bridge

Read and Display Composite of Sea Surface Temperature

Read and display a global monthly composite of sea surface temperature based on data from the AMSR-E sensor on board the Aqua satellite.

First, search the WMS database for layers containing the word "coastwatch". Refine the search to find layers from the AMSR-E sensor.

```
coastwatch = wmsfind('coastwatch','SearchField','serverurl');
layers = refine(coastwatch,'erdAAsstamday','Searchfield','serverurl');
```

Read and display the composite of sea surface temperature for April 16, 2010. Include the coastline, landmask, and nation layers.

```
time = '2010-04-16T00:00:00Z';
[A,R] = wmsread(layers(end:-1:1),'Time',time);

axesm('pcarree','Maplonlimit', [0, 360], ...
                'PLabelLocation',45, ...
                'MLabelLocation',90, ...
                'MLabelParallel', -90, ...
                'MeridianLabel', 'on', ...
                'ParallelLabel','on');
geoshow(A,R);
title({layers(end).LayerTitle,time})
```

Aqua AMSR-E, Near Real Time, Global, 2005-2011 (Monthly Composite), Lon+/-1
2010-04-16T00:00:00Z
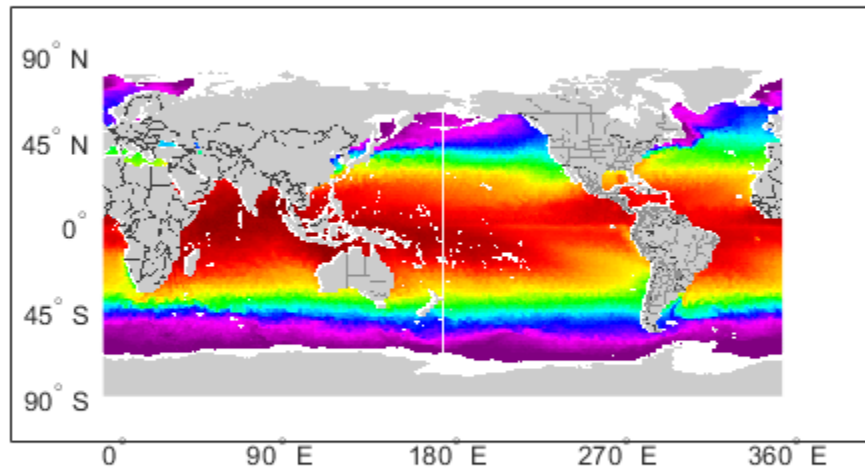
Customize Map Appearance

The `wmsread` function enables you to set optional parameters such as the image height, image width, and background color. For this example, customize the map by specifying an image resolution and background color.

Search the WMS Database for layers that contain `foundation.gtopo30` in the name. GTOPO30, a digital elevation model developed by the United States Geological Survey (USGS), has a horizontal grid spacing of 30 arc seconds.

```
gtopo30layer = wmsfind('foundation.gtopo30');
```

Define a background color by specifying the red, green, and blue levels. Specify the cell size as 0.5 degrees.

```
oceanColor = [0 170 255];
cellSize = 0.5;
```

Read the map. Specify the background color and resolution using name-value arguments.

```
[A,R] = wmsread(gtopo30layer,'BackgroundColor',oceanColor, ...
    'CellSize',cellSize);
```

Display the map and add a title.

```
worldmap world
geoshow(A,R)
title('GTOPO30 Elevation Model')
```


GTOPO30 Elevation Model
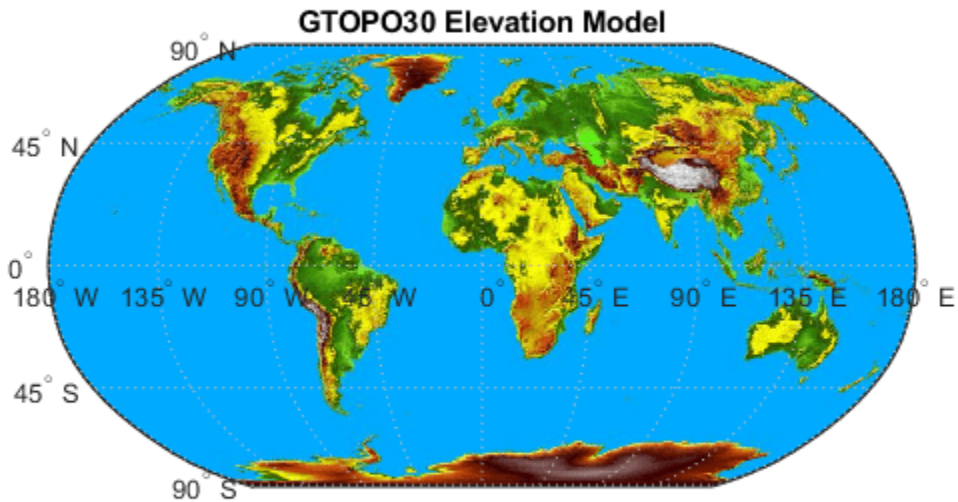
Input Arguments

layer — Information about the layer you are retrieving
WMSLayer object

Information about the layer you are retrieving, specified as a WMSLayer object.

Example: [A,R] = wmsread(layers(1));

mapRequestURL — WMS GetMap request URL
character vector

WMS GetMap request URL, specified as a character vector.

Example: [A,R] = wmsread(mapURL);

Data Types: char

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside quotes. You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: [A,R] = wmsread(layers(1),'latlim',[40 50]);

latlim — Latitude limits of the output image in degrees
[ ] (default) | two-element vector

Latitude limits of the output image in degrees, specified as a two-element vector of the form
`[southern_limit northern_limit]`. The limit values must be ascending. By default, `'Latlim'`
is empty, and `wmsread` uses the full extent in latitude of `layer`. If
`Layer.Details.Attributes.NoSubsets` is `true`, `'Latlim'` may not be modified.

Example: [A,R] = wmsread(layers(1),'latlim',[40 50]);

Data Types: double

lonlim — Longitude limits of the output image in degrees
[ ] (default) | two-element vector

Longitude limits of the output image in degrees, specified as a two-element vector in the form
`[western_limit eastern_limit]`. The limit values must be ascending. By default, `'Lonlim'` is
empty and the full extent in longitude of `layer` is used. If
`Layer.Details.Attributes.NoSubsets` is `true`, you cannot modify `'Lonlim'`

Example: [A,R] = wmsread(layers(1),'lonlim',[40 50]);

Data Types: double

ImageHeight — Desired height of the raster map in pixels
scalar, positive, integer-valued number

Desired height of the raster map in pixels, specified as a scalar, positive, integer-valued number.
`ImageHeight` cannot exceed 8192. If `layer.Details.Attributes.FixedHeight` contains a
positive number, you cannot modify `'ImageHeight'`.

Example: [A,R] = wmsread(layers(1),'ImageHeight',40);

Data Types: double

ImageWidth — Desired width of the raster map in pixels
scalar, positive, integer-valued number

Desired width of the raster map in pixels, specified as a scalar, positive, integer-valued number.
`ImageWidth` cannot exceed 8192. If `Layer.Details.Attributes.FixedWidth` contains a
positive number, you cannot modify `'ImageWidth'`.

Example: [A,R] = wmsread(layers(1),'ImageWidth',100);

Data Types: double

CellSize — Target size of the output pixels (raster cells) in degrees
scalar or two-element vector

Target size of the output pixels (raster cells) in degrees, specified as a scalar or two-element vector. If
you specify a scalar, the value applies to both height and width dimensions. If you specify a vector,
use the form `[height width]`. The `wmsread` function issues an error if you specify both `CellSize`
and `ImageHeight` or `ImageWidth`. The output raster map must not exceed a size of `[8192,8192]`.

Example: [A,R] = wmsread(layers(1),'Cellsize',5);

Data Types: double

RelTolCellSize — Relative tolerance for 'CellSize'
.001 (default) | scalar or two-element vector

Relative tolerance for 'CellSize', specified as a scalar or two-element vector. If you specify a scalar, the value applies to both height and width dimensions. If you specify a vector, the tolerances appear in the order [height width].

Example: [A,R] = wmsread(layers(1),'RelTolCellsize',[4 5]);

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64

ImageFormat — Desired format to use in rendering the map as an image
first available format in the Layer.Details.ImageFormats cell array (default) | character vector

Desired format to use in rendering the map as an image, specified as one of the following character vectors. If specified, the format must match an entry in the Layer.Details.ImageFormats cell array. If not specified, the format defaults to the first available format in the supported format list. .

Value	Description
'image/jpeg'	JPEG
'image/gif'	GIF
'image/png'	PNG
'image/tiff'	TIFF
'image/geotiff'	GeoTIFF
'image/geotiff8'	GeoTIFF8
'image/tiff8'	TIFF8
'image/png8'	PNG8
'image/bil'	Band Interleaved by Line (BIL) format. When you specify the 'image/bil' format, wmsread returns A as a two-dimensional array with a class type of int16 or int32.

Example: [A,R] = wmsread(layers(1),'ImageFormat','image/png');

Data Types: char

StyleName — Style to use when rendering the image
'' (default) | character vector | cell array of character vectors

Style to use when rendering the image, specified as a character vector or cell array of character vectors. The StyleName must be a valid entry in the Layer.Details.Style.Name field. If you request multiple layers, each with a different style, then StyleName must be a cell array of character vectors.

Example: [A,R] = wmsread(layer(1),'StyleName','style');

Data Types: char | cell

Transparent — Pixel transparency
false (default) | true

Pixel transparency, specified as a logical value, true or false. When you set Transparent to true, pixel transparency is enabled, meaning all pixels not representing features or data values are set to a

transparent value. When you set `Transparent` to `false`, non-data pixels are set to the value of the background color.

Example: `[A,R] = wmsread(layers(1),'Transparent',true);`

Data Types: `logical`

`BackgroundColor` — Color used for background (nondata) pixels of the map
`[255,255,255]` (default) | three-element vector

Color used for background (nondata) pixels of the map, specified as a three-element vector.

Example: `[A,R] = wmsread(layers(1),'BackgroundColor',[0,0,255]);`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`Elevation` — Desired elevation extent of the requested map
character vector

Desired elevation extent of the requested map, specified as a character vector. The layer must contain elevation data, which is indicated by the `'Name'` field of the `Layer.Details.Dimension` structure. The `'Name'` field must contain the value `'elevation'`. The `'Extent'` field of the `Layer.Details.Dimension` structure determines the permissible range of values for the parameter.

Example: `[A,R] = wmsread(layer(1),'Elevation','test');`

Data Types: `char`

`Time` — Desired time extent of the requested map
character vector | numeric date number

Desired time extent of the requested map, specified as a character vector or numeric date number. The layer must contain data with a time extent, which is indicated by the `'Name'` field of the `Layer.Details.Dimension` structure. The `'Name'` field must contain the value `'time'`. The `'Extent'` field of the `Layer.Details.Dimension` structure determines the permissible range of values for the parameter. For more information about setting this parameter, see the `WMSMapRequest.Time` property reference page.

Example: `[A,R] = wmsread(layer(1),'Time','June 15, 2015');`

Data Types: `double` | `char`

`SampleDimension` — Name of sample dimension
two-element cell array of character vectors

Name of dimension, specified as a two-element cell array of character vectors, other than `'time'` or `'elevation'` and its character vector value. The layer must contain data with a sample dimension extent, which is indicated by the `'Name'` field of the `Layer.Details.Dimension` structure. The `'Name'` field must contain the value of the first element of `'SampleDimension'`. The `'Extent'` field of the `Layer.Details.Dimension` structure determines the permissible range of values for the second element of `'SampleDimension'`.

Example: `[A,R] = wmsread(layer(1),'SampleDimension',{'sample','test'});`

Data Types: `cell`

`TimeoutInSeconds` — Number of seconds to elapse before issuing a server time-out
60 (default) | scalar integer

Number of seconds to elapse before issuing a server time-out, specified as a scalar integer. If you set the value to `0`, `wmsread` ignores the time-out mechanism.

Example: `[A,R] = wmsread(layers(1),'TimeoutInSeconds',80);`

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output Arguments

A — Color or grayscale image
real, nonsparse, 2-D matrix

Color or grayscale image, returned as a real, nonsparse, 2-D matrix.

R — Geographic raster reference object
`map.rasterref.GeographicCellsReference` object

Geographic raster reference object, returned as a `map.rasterref.GeographicCellsReference` object. A raster referencing object ties the image A to the EPSG:4326 geographic coordinate system.

mapRequestURL — WMS GetMap request URL
character vector

WMS `GetMap` request URL, returned as a character vector.

More About

EPSG: 4326 Coordinate Reference System

The EPSG:4326 coordinate reference system is based on the WGS84 (1984 World Geodetic System) datum. Latitude and longitude are in degrees and longitude is referenced to the Greenwich Meridian.

Tips

- Establish an Internet connection to use `wmsread`. Periodically, the WMS server is unavailable. Retrieving the map can take several minutes. `wmsread` communicates with the server using a `WebMapServer` object representing a WMS server. The object acts as a proxy to a WMS server and resides physically on the client side. The object retrieves the map from the server. The object automatically times-out after 60 seconds if a connection is not made to the server.

- To specify a proxy server to connect to the Internet, select **File > Preferences > Web** and enter your proxy information. Use this feature if you have a firewall.

- `wmsread` supports reading data in WMS versions 1.0.0, 1.1.1, and 1.3.0. For version 1.3.0 only, the WMS specification states, "EPSG:4326 refers to WGS 84 geographic latitude, then longitude. That is, in this CRS the *x*-axis corresponds to latitude, and the *y*-axis to longitude." Most servers provide data in this manner; however, some servers conform to version 1.1.1, where the *x*-axis corresponds to longitude and the *y*-axis to latitude.

  `wmsread` attempts to validate whether a server is confirming to the specification. It checks the EPSG:4326 bounding box, and if the `XLim` values exceeds the range of latitude, then the axes are swapped to conform to version 1.1.1 rather than 1.3.0. If `wmsread` does not detect that the `XLim` values exceed the range of latitude and you notice that the latitude and longitude limits are reversed, then you need to swap them. You can either modify the `bbox` parameters in the `mapRequestURL` or modify the `Latlim` and `Lonlim` parameter values, if permissible.

See Also

wmsfind | wmsinfo | wmsupdate | WebMapServer | WMSLayer

Topics
"Basic Workflow for Creating WMS Maps"

Introduced before R2006a

wmsupdate

Synchronize WMSLayer object with server

Syntax

```
[updatedLayers,index] = wmsupdate(layers)
[...] = wmsupdate(layers,Name,Value, ...)
```

Description

`[updatedLayers,index] = wmsupdate(layers)` returns a Web Map Service (WMS) layer array with its properties synchronized with values from the server, where `layers` contains only one, unique `ServerURL`. `wmsupdate` removes layers that are no longer available on the server.

`wmsupdate` returns the logical array `index` which contains `true` for each available layer. Thus, the return value `updatedLayers` has the same size as `layers(index)`. Except for deletion, `updatedLayers` preserves the same order of layers as `layers`.

`[...] = wmsupdate(layers,Name,Value, ...)` specifies parameter-value pairs that modify the request. Parameter names can be abbreviated and are case-insensitive.

The function accesses the Internet to update the properties. Periodically, the WMS server is unavailable. Updating the layer can take several minutes. The function times-out after 60 seconds if a connection is not made to the server.

Examples

Update Layers

Search the WMS database for layers from the NASA Goddard Space Flight Center. Then, synchronize the properties of the layers with values from the servers.

```
nasa = wmsfind('gsfc.nasa.gov','SearchField','serverurl');
nasa = wmsupdate(nasa,'AllowMultipleServers',true);
```

Update Layer and Refine Search

When you synchronize WMS layers with the corresponding WMS server, the `Abstract`, `CoordRefSysCodes`, and `Details` properties of the `WMSLayer` objects become available. You can use the additional information to refine your search.

Find all layers in the WMS Database with the title `'Global Sea Surface Temperature'`. Determine the number of unique servers by using the `servers` function.

```
global_sst = wmsfind ('Global Sea Surface Temperature', ...
    'SearchField','LayerTitle');
servers(global_sst)
```

```
ans = 1×1 cell array
    {'https://svs.gsfc.nasa.gov/cgi-bin/wms?'}
```

Synchronize the layers with the server. View the abstract, coordinate reference system code, and details of the first layer.

```
global_sst = wmsupdate(global_sst);
global_sst(1).Abstract

ans =
    'The temperature of the surface of the world's oceans provides a clear indication of the sta

     Additional Credit:
     B>Please give credit for this item to:</b><br />'
```

```
global_sst(1).CoordRefSysCodes

ans = 1×1 cell array
    {'CRS:84'}
```

```
global_sst(1).Details

ans = struct with fields:
     MetadataURL: 'http://svs.gsfc.nasa.gov/vis/a000000/a002900/a002905/a002905.fgdc'
      Attributes: [1×1 struct]
     BoundingBox: [1×1 struct]
       Dimension: [1×1 struct]
    ImageFormats: {'image/png'}
     ScaleLimits: [1×1 struct]
           Style: [1×2 struct]
         Version: '1.3.0'
```

Search for abstracts for layers that contain `'El Nino'`.

```
el_nino = refine(global_sst,'El Nino', ...
    'SearchFields','abstract');
```

Update and Display Layers

Search the WMS database for layers from the NASA Goddard Space Flight Center SVS Image Server. Synchronize the properties of the layers with values from the server. Then, refine the search to find layers containing the term "blue marble".

```
gsfc = wmsfind('svs.gsfc.nasa.gov','SearchField','serverurl');
gsfc = wmsupdate(gsfc);
blue_marble = refine(gsfc,'blue marble','SearchField','abstract');
```

Further refine the search to find the first layer with a title containing the terms "512" and "image".

```
queryStr = '*512*image';
layers =  refine(blue_marble,queryStr);
layer = layers(1);
```

Display the layer.

```
[A,R] = wmsread(layer);
worldmap world
plabel off
mlabel off
geoshow(A,R)
title(layer.LayerTitle)
```

Complete Earth (1024x512 Image)

Save Updated Layers

Updating layers from multiple servers might take a long time. You can avoid updating layers multiple times by saving updated layers to a MAT file.

Search the WMS Database for layers that contain information about clouds.

```
layers = wmsfind('cloud');
```

Update the layers. The code might take several minutes to run because it is synchronizing layers for multiple servers.

```
updatedLayers = wmsupdate(layers,'AllowMultipleServers',true);
```

Save the layers to a MAT file called `cloudLayers`.

```
save('cloudLayers','updatedLayers')
```

In the future, you can load the MAT file, refine your search, and read a map without updating the layers again.

Note that servers change, so the layers contained in the MAT file might become out-of-date.

Input Arguments

`layers` — Updated layers
`WMSLayer` object

Updated layers, specified as an array of `WMSLayer` objects.

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

`TimeoutInSeconds` — Number of seconds before server times out
numeric scalar

Number of seconds to elapse before a server times out, specified as an integer-valued, scalar double. If you specify the value `0`, `wmsinfo` ignores the time-out mechanism.

Data Types: `double` | `int16` | `int32` | `int64` | `int8` | `single` | `uint8` | `uint16` | `uint32` | `uint64`

`AllowMultipleServers` — Layer array may contain elements from multiple servers
`false` (default) | `true`

Layer array may contain elements from multiple servers, specified as `true` or `false`. The value `false` indicates the array must contain elements from the same server. Use caution when setting the value to `true`, since you are making a request to each unique server and each request can take several minutes to finish.

Data Types: `double` | `int16` | `int32` | `int64` | `int8` | `single` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

Output Arguments

`updatedLayers` — Synchronized layers
`WMSLayer objects`

Synchronized layers, returned as an array of `WMSLayer` objects with its properties synchronized with values from the server.

`index` — Available layers
`logical` array

Available layers, returned as a logical array where the value `true` indicates that the layer was available from the server.

Tips

- To specify a proxy server to connect to the Internet, click **Preferences** and, in the Preferences dialog box, select **Web**. Enter your proxy information. Use this feature if you have a firewall. For more information, see "Proxy Server Authentication".

- Depending on the number or reliability of servers, you might have to wait several minutes for information to download. Servers can become unavailable.

See Also

`wmsfind` | `wmsinfo` | `wmsread` | `WebMapServer` | `WMSLayer`

Topics
"Basic Workflow for Creating WMS Maps"

Introduced in R2009b

worldFileMatrix

Package: `map.rasterref`

Return world file parameters for transformation

Syntax

```
W = worldFileMatrix(R)
```

Description

`W = worldFileMatrix(R)` returns a 2-by-3 world file matrix from geographic or map raster R.

Examples

Create World File Matrix from a Planar Map Raster

Create a `MapCellsReference` raster reference object.

```
xWorldLimits = [207000 208000];
yWorldLimits = [912500 913000];
rasterSize = [10 20];
R = maprefcells(xWorldLimits,yWorldLimits,rasterSize,'ColumnsStartFrom','north')

R =
  MapCellsReference with properties:

            XWorldLimits: [207000 208000]
            YWorldLimits: [912500 913000]
              RasterSize: [10 20]
    RasterInterpretation: 'cells'
       ColumnsStartFrom: 'north'
          RowsStartFrom: 'west'
      CellExtentInWorldX: 50
      CellExtentInWorldY: 50
    RasterExtentInWorldX: 1000
    RasterExtentInWorldY: 500
        XIntrinsicLimits: [0.5 20.5]
        YIntrinsicLimits: [0.5 10.5]
      TransformationType: 'rectilinear'
    CoordinateSystemType: 'planar'
             ProjectedCRS: []
```

Compute the world file matrix.

```
W = worldFileMatrix(R)

W = 2×3

        50          0      207025
```

```
        0         -50      912975
```

Observe that `W(2,1)` and `W(1,2)` are 0. This value is expected since `R.TransformationType` is `'rectilinear'`.

Input Arguments

R — Geographic or map raster
GeographicCellsReference, GeographicPostingsReference, MapCellsReference, or MapPostingsReference object

Geographic or map raster, specified as a `GeographicCellsReference`, `GeographicPostingsReference`, `MapCellsReference`, or `MapPostingsReference` object.

Output Arguments

W — World file matrix
2-by-3 numeric array

World file matrix, returned as a 2-by-3 numeric array. Each of the six elements in `W` matches one of the lines in a world file corresponding to the transformation defined by raster referencing object R.

Data Types: `double`

More About

World File Matrix

A world file matrix maps points in intrinsic coordinates to points in geographic or planar world coordinates.

Given a world file matrix W of the form:

```
W = [A B C;
     D E F]
```

a point (x_i, y_i) maps to a point (xw, yw) in world coordinates according to:
$$xw = A \times (x_i - 1) + B \times (y_i - 1) + C$$
$$yw = D \times (x_i - 1) + E \times (y_i - 1) + F.$$

More compactly:
$$[xw\ yw]' = W \times [(x_i - 1)\ (y_i - 1)].$$

Note Similar equations hold true for points (*lat*, *lon*) in geographic coordinates. However, the geographic coordinate ordering is switched. That is, *xw* is substituted by *lon*, and *yw* is substituted by *lat*.

The −1s are needed to maintain the Mapping Toolbox convention for intrinsic coordinates, which is consistent with the 1-based indexing used throughout MATLAB.

W is stored in a world file with one term per line in column-major order: *A*, *D*, *B*, *E*, *C*, *F*. That is, a world file contains the elements of *W* in this order:

```
W(1,1)
W(2,1)
W(1,2)
W(2,2)
W(1,3)
W(2,3)
```

The previous expressions hold for both affine and rectilinear transformations of rasters. The values *B*, *D*, *W*(2,1) and *W*(1,2) are identically 0 whenever:

- R is a geographic raster, since longitude depends only on intrinsic *x* and latitude depends only on intrinsic *y*
- R is a map raster and R.TransformationType is 'rectilinear'

See Also

Introduced in R2013b

worldFileMatrixToRefmat

(To be removed) Convert world file matrix to referencing matrix

---

Note worldFileMatrixToRefmat will be removed in a future release. Use the georasterref or maprasterref function instead. For more information, see "Compatibility Considerations".

Syntax

refmat = worldFileMatrixToRefmat(W)

Description

refmat = worldFileMatrixToRefmat(W) converts the 2-by-3 world file matrix W to a 3-by-2 referencing matrix refmat.

For a definition of a world file matrix, see the worldFileMatrix method of the map raster reference and geographic raster reference classes.

Compatibility Considerations

worldFileMatrixToRefmat will be removed
Not recommended starting in R2020b

Some functions that return referencing matrices will be removed, including the worldFileMatrixToRefmat function. Instead, create a raster reference object using the georasterref or maprasterref function. Reference objects have several advantages over referencing matrices.

- Unlike referencing matrices, reference objects have properties that document the size of the associated raster, its limits, and the direction of its rows and columns. For more information about reference object properties, see the GeographicCellsReference and MapPostingsReference functions.
- You can manipulate the limits of rasters associated with reference objects using the geocrop or mapcrop function.
- You can manipulate the size and resolution of rasters associated with reference objects using the georesize or mapresize function.
- Most functions that accept referencing matrices as inputs also accept reference objects.

To update your code, change instances of the worldFileMatrixToRefmat function to the georasterref or maprasterref function and specify the size of the raster A. Use the georasterref function for geographic coordinates and the maprasterref function for planar map coordinates.

```
R = georasterref(W,size(A));
```

You can also specify the raster interpretation as 'cells' for a raster of cells or 'postings' for a raster of regularly posted samples. By default, the georasterref and maprasterref functions use 'cells'.

```
R = georasterref(W,size(A),'postings');
```

See Also
refmatToWorldFileMatrix | georasterref | maprasterref

Introduced in R2011a

worldfileread

Read world file and return referencing object or matrix

Note Syntaxes of the `worldfileread` function that return referencing matrices will be removed in a future release. Use a syntax that returns a reference object instead. For more information, see "Compatibility Considerations".

Syntax

```
R = worldfileread(worldFileName,coordinateSystemType,rasterSize)
refmat = worldfileread(worldFileName)
```

Description

`R = worldfileread(worldFileName,coordinateSystemType,rasterSize)` reads the world file, `worldFileName`, and constructs a spatial referencing object, R. The argument `coordinateSystemType` specifies the type of referencing object. `rasterSize` specifies the size of the image corresponding to the world file.

`refmat = worldfileread(worldFileName)` reads the world file, `worldFileName`, and constructs a 3-by-2 referencing matrix, `refmat`.

Examples

Read Image Referenced to Projected Coordinate System

Read an ortho image referenced to a projected coordinate system (Massachusetts State Plane Mainland).

```
filename = 'concord_ortho_w.tif';
[X, cmap] = imread(filename);
```

Derive world file name from image file name.

```
worldFileName = getworldfilename(filename);
```

Read the world file, returning a referencing object.

```
R = worldfileread(worldFileName, 'planar', size(X))

R = 
  MapCellsReference with properties:

            XWorldLimits: [207000 209000]
            YWorldLimits: [911000 913000]
              RasterSize: [2000 2000]
    RasterInterpretation: 'cells'
        ColumnsStartFrom: 'north'
           RowsStartFrom: 'west'
```

```
   CellExtentInWorldX: 1
   CellExtentInWorldY: 1
RasterExtentInWorldX: 2000
RasterExtentInWorldY: 2000
    XIntrinsicLimits: [0.5 2000.5]
    YIntrinsicLimits: [0.5 2000.5]
  TransformationType: 'rectilinear'
CoordinateSystemType: 'planar'
         ProjectedCRS: []
```

Read Image Referenced to Geographic Coordinate System

Read image reference to a geographic coordinate system.

```
filename = 'boston_ovr.jpg';
RGB = imread(filename);
```

Derive world file name from image file name,

```
worldFileName = getworldfilename(filename);
```

Read world file, returning a referencing object.

```
R = worldfileread(worldFileName, 'geographic', size(RGB))

R =
  GeographicCellsReference with properties:

            LatitudeLimits: [42.3052018188767 42.4165064733949]
           LongitudeLimits: [-71.1308390797572 -70.9898400731705]
                RasterSize: [769 722]
     RasterInterpretation: 'cells'
          ColumnsStartFrom: 'north'
             RowsStartFrom: 'west'
      CellExtentInLatitude: 0.000144739472715501
     CellExtentInLongitude: 0.000195289482807142
    RasterExtentInLatitude: 0.11130465451822
   RasterExtentInLongitude: 0.140999006586757
          XIntrinsicLimits: [0.5 722.5]
          YIntrinsicLimits: [0.5 769.5]
      CoordinateSystemType: 'geographic'
              GeographicCRS: []
                  AngleUnit: 'degree'
```

Input Arguments

worldFileName — Name of world file
character vector

Name of world file, specified as a character vector.

Example: `worldFileName = getworldfilename(filename);`

Data Types: `char`

coordinateSystemType — Type of referencing object
`'planar' | 'geographic'`

Type of referencing object, specified as one of the following:

Value	Description
`'geographic'`	Latitude-longitude systems
`'planar'`	Projected map coordinate systems

Example: `R = worldfileread(worldFileName, 'geographic', size(RGB));`

Data Types: `char`

rasterSize — Size of the image corresponding to the world file
two-element vector

Size of the image corresponding to the world file, specified as a two-element vector, [m n], where m is the number of rows in the image and n is the number of columns in the image.

Example: `size(I)`

Data Types: `double`

Output Arguments

R — Spatial referencing object
raster reference object

Spatial referencing object, returned as a raster reference object.

refmat — Referencing matrix
3-by-2 matrix

Referencing matrix, returned as a 3-by-2 `double` matrix.

Compatibility Considerations

`worldfileread` syntaxes that return referencing matrices will be removed
Not recommended starting in R2020b

Syntaxes of the `worldfileread` function that return referencing matrices will be removed. Use syntaxes that return reference objects instead. Reference objects have several advantages over referencing matrices.

- Unlike referencing matrices, reference objects have properties that document the size of the associated raster, its limits, and the direction of its rows and columns. For more information about reference object properties, see the `GeographicCellsReference` and `MapPostingsReference` objects.
- You can manipulate the limits of rasters associated with reference objects using the `geocrop` or `mapcrop` function.

- You can manipulate the size and resolution of rasters associated with reference objects using the `georesize` or `mapresize` function.
- Most functions that accept referencing matrices as inputs also accept reference objects.

To update your code, specify the coordinate system type as a second argument. Use `'planar'` for planar map coordinates or `'geographic'` for geographic coordinates. Specify the size of the associated raster, A, as a third argument.

```
R = worldfileread(worldFileName,coordinateSystemType,size(A));
```

See Also

getworldfilename | worldToIntrinsic | worldfilewrite | intrinsicToWorld | readgeoraster

Introduced before R2006a

worldfilewrite

Write world file from referencing object or matrix

Syntax

```
worldfilewrite(R, worldfilename)
```

Description

worldfilewrite(R, worldfilename) calculates the world file entries corresponding to referencing object or matrix R and writes them into the file worldfilename. The argument R can be a map raster reference object, a geographic raster reference object, or a 3-by-2 referencing matrix.

Examples

Write out the information from a referencing object for the image file concord_ortho_w.tif

```
info = imfinfo('concord_ortho_w.tif');
R = worldfileread('concord_ortho_w.tfw', ...
     'planar', [info.Height info.Width])
worldfilewrite(R, 'concord_ortho_w_test.tfw');
type concord_ortho_w_test.tfw
```

See Also
getworldfilename | intrinsicToWorld | worldToIntrinsic | worldfileread

Introduced before R2006a

worldGrid

Package: `map.rasterref`

World coordinates of raster elements

Syntax

```
[X,Y] = worldGrid(R)
[X,Y] = worldGrid(R,gridOption)
```

Description

`[X,Y] = worldGrid(R)` returns the world coordinates of raster elements as the 2-D arrays X and Y. The coordinates of raster element `(i,j)` are `(X(i,j),Y(i,j))`.

`[X,Y] = worldGrid(R,gridOption)`, where `gridOption` is `'gridvectors'`, returns X and Y as row vectors. The coordinates of raster element `(i,j)` are `(X(j),Y(i))`. The default for `gridOption` is `'fullgrid'`, which returns X and Y as 2-D arrays.

Examples

Find Coordinates of Raster Elements

Import elevation data, find the coordinates of the imported data, then display the data as a surface.

First import elevation data [1] on page 1-0￼ for an area around Mount Washington as an array and a map cells reference object. Prepare the data for plotting by replacing missing data values with `0`. Then, find the coordinates of each element in the array.

```
[Z,R] = readgeoraster('MtWashington-ft.grd','OutputType','double');
info = georasterinfo('MtWashington-ft.grd');
Z = standardizeMissing(Z,info.MissingDataIndicator);

[X,Y] = worldGrid(R);
```

Display the elevation data as a surface using an appropriate colormap. Set the `EdgeColor` property to `'none'` to remove the mesh from the surface.

```
surf(X,Y,Z,'EdgeColor','none')
demcmap(Z)
xlabel('x (meters)')
ylabel('y (meters)')
zlabel('Elevation (feet)')
```
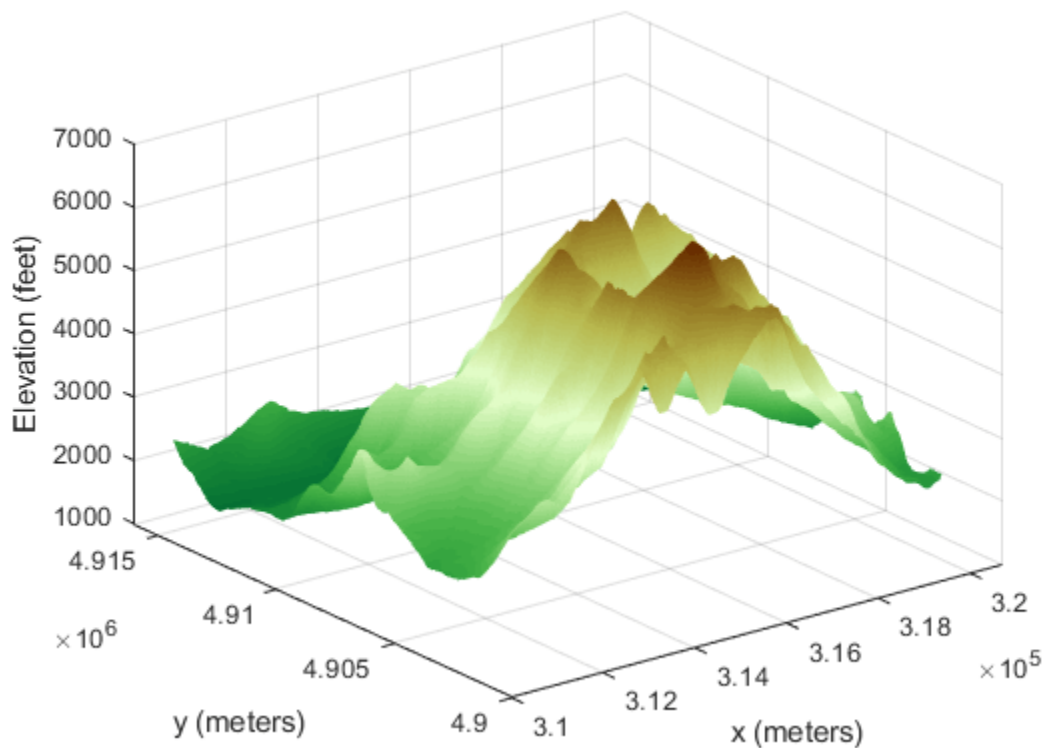
[1] The elevation data used in this example is courtesy of the U.S. Geological Survey.

Get Vectors Instead of Arrays

Create a map cells reference object for a 3-by-4 raster with x values in the range [7000, 7400] meters and y values in the range [2700, 3300] meters. Get the coordinates of the raster elements and return them as row vectors.

```
R = maprefcells([7000 7400],[2700 3300],[3 4]);
[X,Y] = worldGrid(R,'gridvectors')
```

X = 1×4

         7050   7150   7250   7350

Y = 1×3

         2800   3000   3200

If you do not specify the second argument as `'gridvectors'`, then the `worldGrid` function returns 2-D arrays by default.

```
[xFull,yFull] = worldGrid(R)
```

```
xFull = 3×4

        7050           7150           7250           7350
        7050           7150           7250           7350
        7050           7150           7250           7350


yFull = 3×4

        2800           2800           2800           2800
        3000           3000           3000           3000
        3200           3200           3200           3200
```

Input Arguments

R — Spatial reference
MapCellsReference object | MapPostingsReference object

Spatial reference, specified as a MapCellsReference or MapPostingsReference object.

If R is a MapCellsReference object, then X and Y are cell centers. If R is a MapPostingsReference object, then X and Y are posting points.

gridOption — Grid option
'fullgrid' (default) | 'gridvectors'

Grid option, specified as one of these values:

- 'fullgrid' — Return X and Y as 2-D arrays, where each row of X is identical and each column of Y is identical. This is the default behavior.
- 'gridvectors' — Return X and Y as row vectors. Use this option when you want to reduce memory usage and when 2-D arrays are unnecessary, such as when plotting large data sets with the surf function. You can specify gridOption as 'gridvectors' only when the TransformationType property of R has a value of 'rectilinear'.

This table shows the difference between 'fullgrid' and 'gridvectors'.

'fullgrid'	'gridvectors'

`R = maprefcells([7000 7400],[2700 3300],[3 4]);`
`[X,Y] = worldGrid(R)`
 	
`X =`
 	
`    7050       7150       7250       7350`
`    7050       7150       7250       7350`
`    7050       7150       7250       7350`
 	
`Y =`
 	
`    2800       2800       2800       2800`
`    3000       3000       3000       3000`
`    3200       3200       3200       3200`
	`R = maprefcells([7000 7400],[2700 3300],[3 4]);`
	`[X,Y] = worldGrid(R,'gridvectors')`
	
	`X =`
	
	`    7050       7150       7250       7350`
	
	`Y =`
	
	`    2800       3000       3200`

Data Types: char | string

Output Arguments

X — *x*-coordinates
2-D array (default) | row vector

x-coordinates, returned as a 2-D array or a row vector. By default, X is a 2-D array. To return X as a row vector, specify gridOption as 'gridvectors'.

By default, and when gridOption is 'fullgrid', the sizes of X and Y each equal the RasterSize property of R. When gridOption is 'gridvectors', the lengths of X and Y equal the second and first elements of the RasterSize property of R, respectively.

Y — *y*-coordinates
2-D array (default) | row vector

y-coordinates, returned as a 2-D array or a row vector. By default, Y is a 2-D array. To return Y as a row vector, specify gridOption as 'gridvectors'.

By default, and when gridOption is 'fullgrid', the sizes of X and Y each equal the RasterSize property of R. When gridOption is 'gridvectors', the lengths of X and Y equal the second and first elements of the RasterSize property of R, respectively.

See Also

Functions
ndgrid | meshgrid | intrinsicToWorld | geographicGrid

Objects
MapCellsReference | MapPostingsReference

Introduced in R2021a

worldmap

Construct map axes for given region of world

Syntax

```
worldmap region
worldmap(region)
worldmap
worldmap(latlim,lonlim)
worldmap(Z,R)
h = worldmap( ___ )
```

Description

`worldmap region` or `worldmap(region)` sets up an empty map axes with projection and limits suitable to the part of the world specified in `region`.

`worldmap` with no arguments presents a menu from which you can select the name of a single continent, country, island, or region.

`worldmap(latlim,lonlim)` allows you to define a custom geographic region in terms of its latitude and longitude limits in degrees.

`worldmap(Z,R)` derives the map limits from the extent of a regular data grid, Z, georeferenced by R.

`h = worldmap( ___ )` returns the handle of the map axes.

Examples

Set up World Map and Draw Coastlines

Set up a world map and draw coarse coastlines.

```
worldmap('World')
load coastlines
plotm(coastlat,coastlon)
```
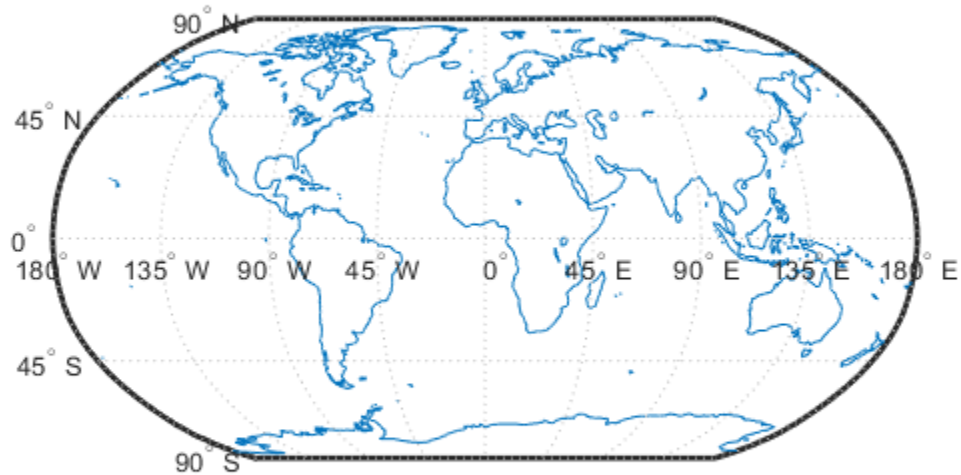
Set up World Map with Land Areas, Lakes, and Other Landmarks

Set up a world map with land areas, major lakes and rivers, and cities and populated places.

```
ax = worldmap('World');
setm(ax, 'Origin', [0 180 0])

land = shaperead('landareas.shp', 'UseGeoCoords', true);
geoshow(ax, land, 'FaceColor', [0.5 0.7 0.5])

lakes = shaperead('worldlakes.shp', 'UseGeoCoords', true);
geoshow(lakes, 'FaceColor', 'blue')

rivers = shaperead('worldrivers.shp', 'UseGeoCoords', true);
geoshow(rivers, 'Color', 'blue')

cities = shaperead('worldcities.shp', 'UseGeoCoords', true);
geoshow(cities, 'Marker', '.', 'Color', 'red')
```
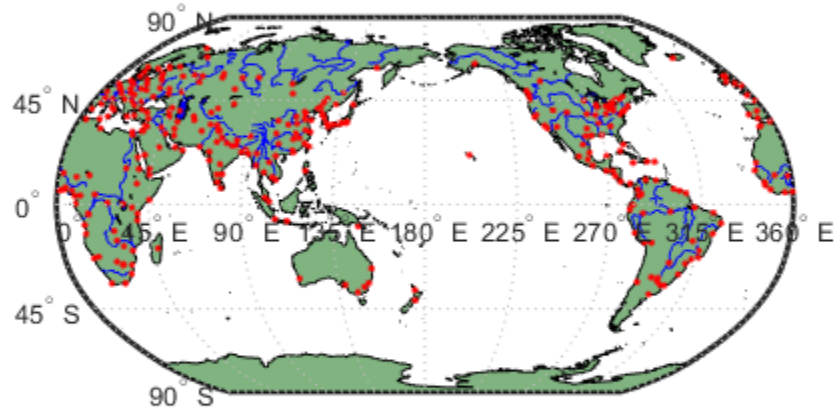
Draw Map of Antarctica

Draw a map of Antarctica, using the `worldmap` function.

```
worldmap('antarctica')
antarctica = shaperead('landareas.shp', 'UseGeoCoords', true,...
  'Selector',{@(name) strcmp(name,'Antarctica'), 'Name'});
patchm(antarctica.Lat, antarctica.Lon, [0.5 1 0.5])
```
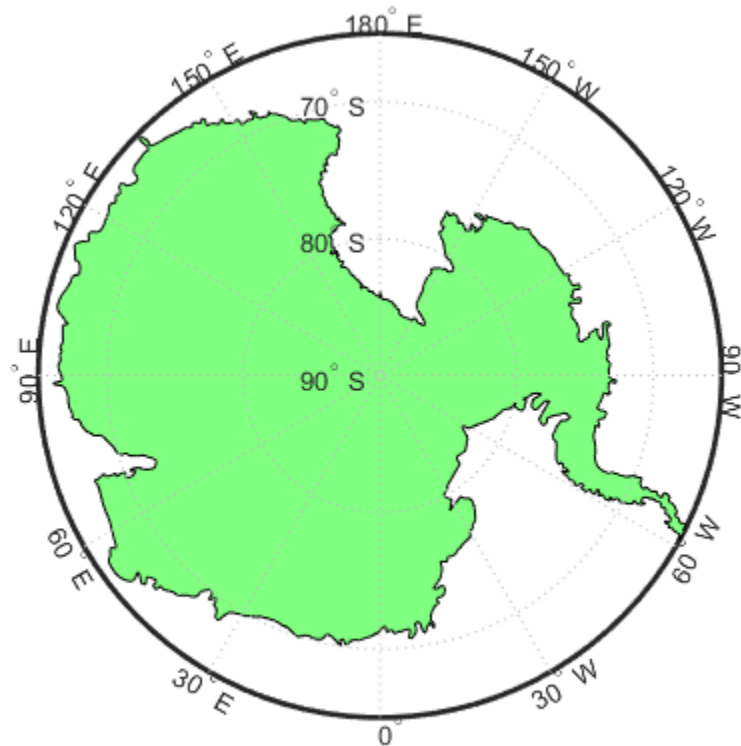
Draw a map of Africa and India with Major Cities

Draw a map of Africa and India with major cities and populated areas.

```
worldmap({'Africa','India'})
land = shaperead('landareas.shp', 'UseGeoCoords', true);
geoshow(land, 'FaceColor', [0.15 0.5 0.15])
cities = shaperead('worldcities', 'UseGeoCoords', true);
geoshow(cities, 'Marker', '.', 'Color', 'red')
```
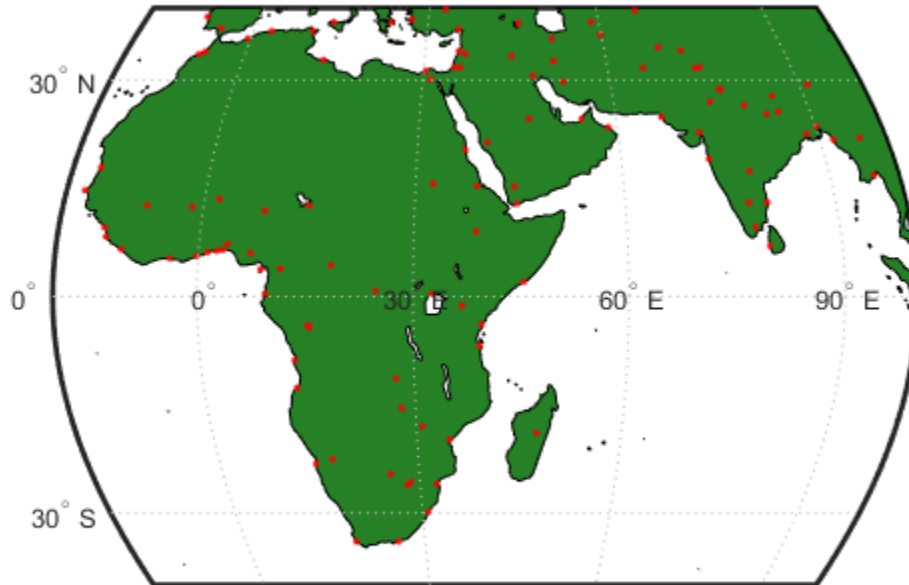
Display Geoid Heights Over South America and Central Pacific

Display geoid heights from the EGM96 geoid model over a map of South America and the central Pacific. First, get geoid heights and a geographic postings reference object. Load coastline latitude and longitude data.

```
[N,R] = egm96geoid;
load coastlines
```

Create a world map by specifying latitude and longitude limits. Then, display the geoid heights and coastline data.

```
latlim = [-50 50];
lonlim = [160 -30];
worldmap(latlim,lonlim)

geoshow(N,R,'DisplayType','surface')
geoshow(coastlat,coastlon,'Color','k')
```
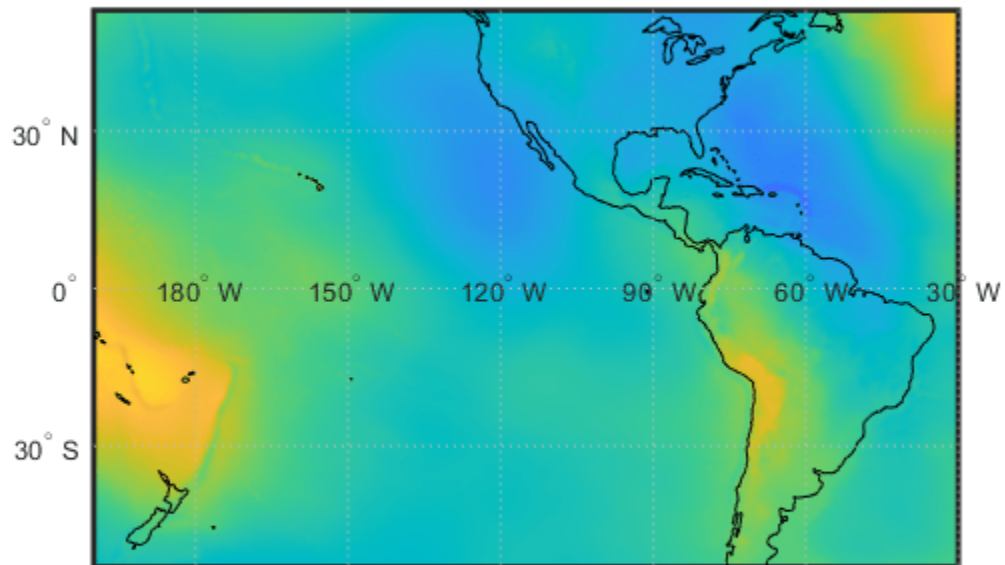
Display Map of Terrain Elevations in Korea

Load elevation data and a geographic cells reference object for the Korean peninsula. Create a world map with appropriate latitude and longitude limits. Then, display the data as a texture map using geoshow. Apply a colormap appropriate for elevation data using demcmap.

```
load korea5c
worldmap(korea5c,korea5cR);
geoshow(korea5c,korea5cR,'DisplayType','texturemap')
demcmap(korea5c)
```
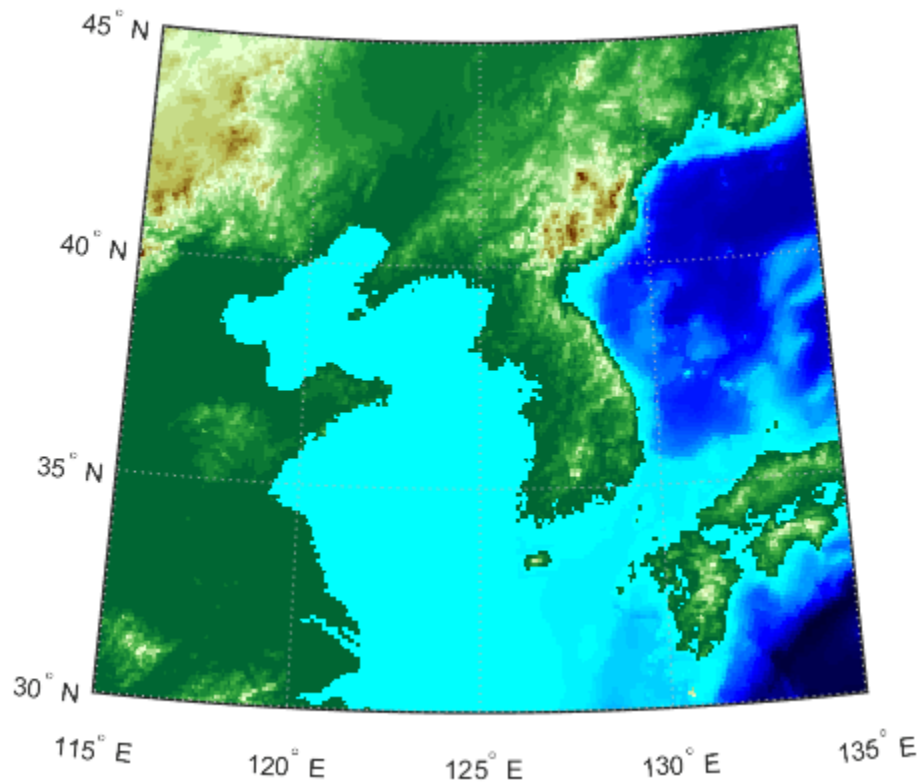
Map the United States of America Coloring State Polygons

Create a map of the United States of America. Display the states using a palette of random, muted colors.

```
worldmap("USA");

load coastlines
geoshow(coastlat,coastlon, ...
    "DisplayType","polygon","FaceColor",[0.45 0.60 0.30])

states = readgeotable("usastatelo.shp");

h = height(states);
faceColors = makesymbolspec("Polygon", ...
    {'INDEX',[1 h],'FaceColor',polcmap(h)});

geoshow(states,"DisplayType","polygon", ...
  "SymbolSpec",faceColors)
```
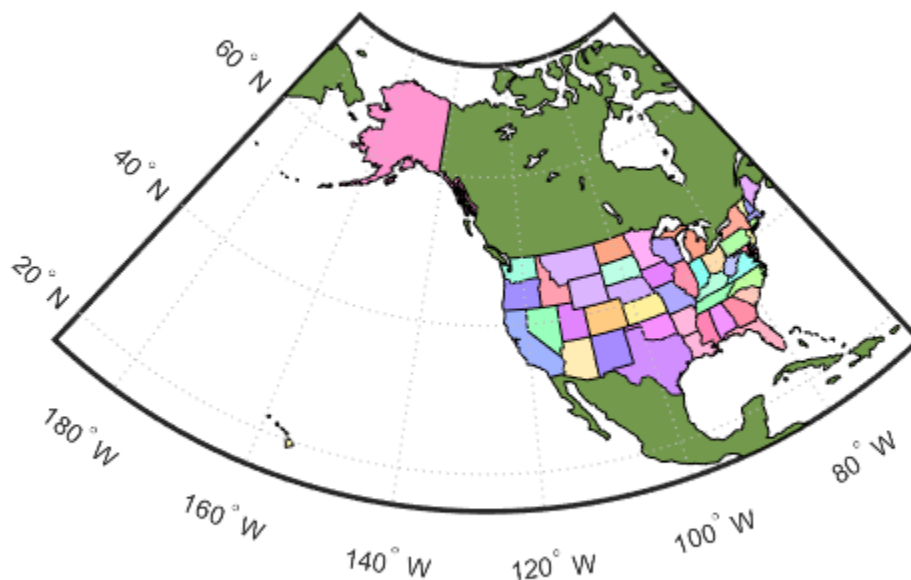
Input Arguments

`region` — Region to display
`'World'` | `'North Pole'` | `'South Pole'` | `'Pacific'` | character vector | string scalar | string array | cell array of character vectors

Region to display, specified as a string scalar, string array, character vector, or cell array of character vectors. Permissible values include names of continents, countries, and islands as well as `'World'`, `'North Pole'`, `'South Pole'`, and `'Pacific'`.

Example: `{'Africa','India'}`

`latlim` — Latitude limits
two-element vector

Latitude limits, specified as a two-element vector of the form `[southern_limit northern_limit]`.

`lonlim` — Longitude limits
two-element vector

Longitude limits, specified as a two-element vector of the form `[western_limit eastern_limit]`.

`Z` — Data grid
M-by-*N* array

Data grid, specified as an *M*-by-*N* array. Z is a regular data grid associated with a geographic reference R.

R — Geographic reference
geographic raster reference object | vector | matrix

Geographic reference, specified as one of the following.

Type	Description
Geographic raster reference object	`GeographicCellsReference` or `GeographicPostingsReference` geographic raster reference object. The `RasterSize` property must be consistent with the size of the data grid, `size(Z)`.
Vector	1-by-3 numeric vector with elements: `[cells/degree northern_latitude_limit western_longitude_limit]`
Matrix	3-by-2 numeric matrix that transforms raster row and column indices to or from geographic coordinates according to: `[lon lat] = [row col 1] * R` R defines a (non-rotational, non-skewed) relationship in which each column of the data grid falls along a meridian and each row falls along a parallel.

For more information about referencing vectors and matrices, see "Georeferenced Raster Data".

Output Arguments

h — Handle of the map axes
handle object

Handle of the map axes, returned as a handle object.

Tips

- All axes created with `worldmap` are initialized with a spherical Earth model having a radius of 6,371,000 meters.
- `worldmap` uses `tightmap` to adjust the axes limits around the map. If you change the projection, or just want more white space around the map frame, use `tightmap` again or `auto axis`.

See Also
`axesm` | `framem` | `geoshow` | `gridm` | `mlabel` | `plabel` | `tightmap` | `usamap`

Introduced before R2006a

worldToDiscrete

Package: `map.rasterref`

Transform planar world to discrete coordinates

Syntax

`[I,J] = worldToDiscrete(R,xWorld,yWorld)`

Description

`[I,J] = worldToDiscrete(R,xWorld,yWorld)` returns the indices corresponding to world coordinates `xWorld` and `yWorld` in map raster `R`. If `R.RasterInterpretation` is:

- `'cells'`, then `I` and `J` are the row and column subscripts of the raster cells (or image pixels)
- `'postings'`, then `I` and `J` refer to the nearest sample point (posting)

Examples

Find Indices of Cell from World Coordinates

Import a GeoTIFF image [1] on page 1-0 of Boston as an array and a `MapCellsReference` object. Find the indices of the cell containing data with world *x*- and *y*-coordinates of 776500 and 2956600 survey feet, respectively.

```
[Z,R] = readgeoraster('boston.tif');
[I,J] = worldToDiscrete(R,776500,2956600)
```

`I = 1212`

`J = 3714`

The result means that the cell center closest to the world coordinates is in row 1212 and column 3714 of the array.

[1] The data used in this example includes material copyrighted by GeoEye, all rights reserved.

Input Arguments

R — Map raster
`MapCellsReference` or `MapPostingsReference` object

Map raster, specified as a `MapCellsReference` or `MapPostingsReference` object.

xWorld — *x*-coordinates in the world coordinate system
numeric array

x-coordinates in the world coordinate system, specified as a numeric array.

Data Types: `single` | `double`

yWorld — *y*-coordinates in the world coordinate system
numeric array

y-coordinates in the world coordinate system, specified as a numeric array. `yWorld` is the same size as `xWorld`.

Data Types: `single` | `double`

Output Arguments

`I` — World *x*-coordinate indices
array of integers

World *x*-coordinate indices, returned as an array of integers. `I` is the same size as `xWorld`.

For an *m*-by-*n* raster, $1 \leq I \leq m$, except for points (`xWorld`(*k*), `yWorld`(*k*)) that fall outside the bounds of the raster as defined by the function `contains`. In this case `I`(*k*) and `J`(*k*) are `NaN`.

Data Types: `double`

`J` — World *y*-coordinate indices
array of integers

World *y*-coordinate indices, returned as an array of integers. `J` is the same size as `yWorld`.

For an *m*-by-*n* raster, $1 \leq I \leq m$, except for points (`xWorld`(*k*), `yWorld`(*k*)) that fall outside the bounds of the raster as defined by the function `contains`. In this case `I`(*k*) and `J`(*k*) are `NaN`.

Data Types: `double`

See Also
`contains` | `worldToIntrinsic` | `geographicToDiscrete`

Introduced in R2013b

worldToIntrinsic

Package: `map.rasterref`

Transform planar world to intrinsic coordinates

Syntax

```
[xIntrinsic,yIntrinsic] = worldToIntrinsic(R,xWorld,yWorld)
```

Description

`[xIntrinsic,yIntrinsic] = worldToIntrinsic(R,xWorld,yWorld)` returns the intrinsic coordinates corresponding to planar world coordinates (`xWorld`, `yWorld`) in map raster `R`. If a point is outside the bounds of `R`, then `worldToIntrinsic` extrapolates the `xIntrinsic` and `yIntrinsic` coordinates.

Examples

Find Intrinsic Coordinates from World Coordinates

Import a GeoTIFF image [1] on page 1-0 of Boston as an array and a `MapCellsReference` object. Find the intrinsic coordinates corresponding to the world *x*- and *y*-coordinates of 776500 and 2956600 survey feet, respectively.

```
[Z,R] = readgeoraster('boston.tif');
[xIntrinsic,yIntrinsic] = worldToIntrinsic(R,776500,2956600)
```

```
xIntrinsic = 3.7136e+03
```

```
yIntrinsic = 1.2119e+03
```

The result means that the world coordinates fall close to the center of the cell corresponding to column 3713 and row 1211 of the array.

You can reverse the operation by using the `intrinsicToWorld` function.

```
[xWorld,yWorld] = intrinsicToWorld(R,xIntrinsic,yIntrinsic)
```

```
xWorld = 776500
```

```
yWorld = 2956600
```

[1] The data used in this example includes material copyrighted by GeoEye, all rights reserved.

Input Arguments

R — Map raster
`MapCellsReference` or `MapPostingsReference` object

Map raster, specified as a `MapCellsReference` or `MapPostingsReference` object.

xWorld — *x*-coordinates in the world coordinate system
numeric array

x-coordinates in the world coordinate system, specified as a numeric array. xWorld coordinates can be outside the bounds of the raster R.

Data Types: single | double

yWorld — *y*-coordinates in the world coordinate system
numeric array

y-coordinates in the world coordinate system, specified as a numeric array. yWorld is the same size as xWorld. yWorld coordinates can be outside the bounds of the raster R.

Data Types: single | double

Output Arguments

xIntrinsic — *x*-coordinates in the intrinsic coordinate system
numeric array

x-coordinates in the intrinsic coordinate system, returned as a numeric array. xIntrinsic is the same size as xWorld.

When xWorld(*k*) is outside the bounds of raster R, xIntrinsic(*k*) is extrapolated in the intrinsic coordinate system.

Data Types: double

yIntrinsic — *y*-coordinates in the intrinsic coordinate system
numeric array

y-coordinates in the intrinsic coordinate system, returned as a numeric array. yIntrinsic is the same size as xWorld.

When yWorld(*k*) outside the bounds of raster R, yIntrinsic(*k*) is extrapolated in the intrinsic coordinate system.

Data Types: double

See Also
intrinsicToWorld | worldToDiscrete | geographicToIntrinsic

Introduced in R2013b

wrapTo180

Wrap angle in degrees to [-180 180]

Syntax

```
lonWrapped = wrapTo180(lon)
```

Description

`lonWrapped = wrapTo180(lon)` wraps angles in `lon`, in degrees, to the interval [–180, 180] such that 180 maps to 180 and –180 maps to –180. In general, odd, positive multiples of 180 map to 180 and odd, negative multiples of 180 map to –180.

Examples

Wrap Longitudes to 180 Degrees

Specify a short list of longitudes to wrap.

```
lon = [-400 -190 -180 -175 175 180 190 380];
```

Wrap the longitudes to the range [-180, 180] degrees.

```
lonWrapped = wrapTo180(lon)
```

```
lonWrapped = 1×8

   -40    170   -180   -175    175    180   -170     20
```

Specify a second list of longitudes that are sampled over a large range of angles. Wrap the longitudes.

```
lon2 = -720:720;
lon2Wrapped = wrapTo180(lon2);
```

Plot the wrapped longitudes. The wrapped longitudes stay in the range [-180, 180] degrees.

```
plot(lon2,lon2Wrapped)
xlabel("Longitude (degrees)")
ylabel("Wrapped Longitude (degrees)")
grid on
```
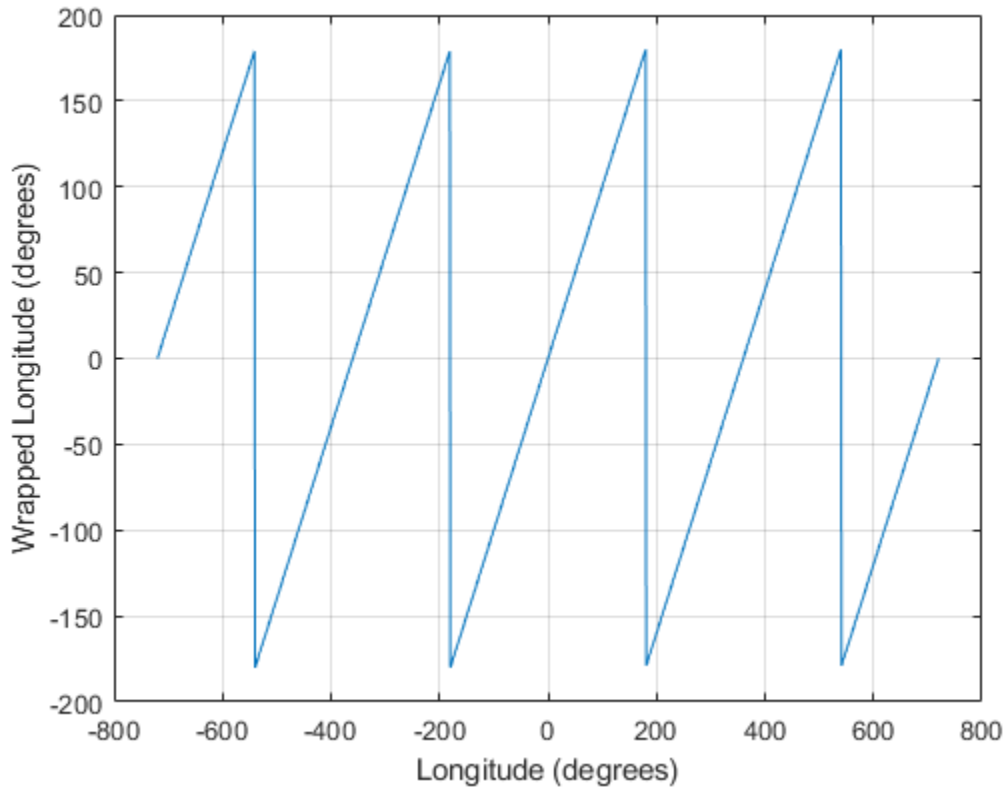
Input Arguments

lon — Angles
numeric vector

Angles, specified as a numeric vector.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

Output Arguments

lonWrapped — Wrapped angles
numeric vector

Wrapped angles, specified as a numeric vector with values in the range [–180, 180].

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

wrapTo360 | wrapTo2Pi | wrapToPi

Introduced in R2007b

wrapTo360

Wrap angle in degrees to [0 360]

Syntax

```
lonWrapped = wrapTo360(lon)
```

Description

`lonWrapped = wrapTo360(lon)` wraps angles in `lon`, in degrees, to the interval [0, 360] such that 0 maps to 0 and 360 maps to 360. In general, positive multiples of 360 map to 360 and negative multiples of 360 map to zero.

Examples

Wrap Longitudes to 360 Degrees

Specify a short list of longitudes to wrap.

```
lon = [-720 -400 -360 -355 350 360 370 720];
```

Wrap the longitudes to the range [0, 360] degrees.

```
lonWrapped = wrapTo360(lon)
```

```
lonWrapped = 1×8

     0   320     0     5   350   360    10   360
```

Specify a second list of longitudes that are sampled over a large range of angles. Wrap the longitudes.

```
lon2 = -1000:1000;
lon2Wrapped = wrapTo360(lon2);
```

Plot the wrapped longitudes. The wrapped longitudes stay in the range [0, 360] degrees.

```
plot(lon2,lon2Wrapped)
xlabel("Longitude (degrees)")
ylabel("Wrapped Longitude (degrees)")
grid on
```
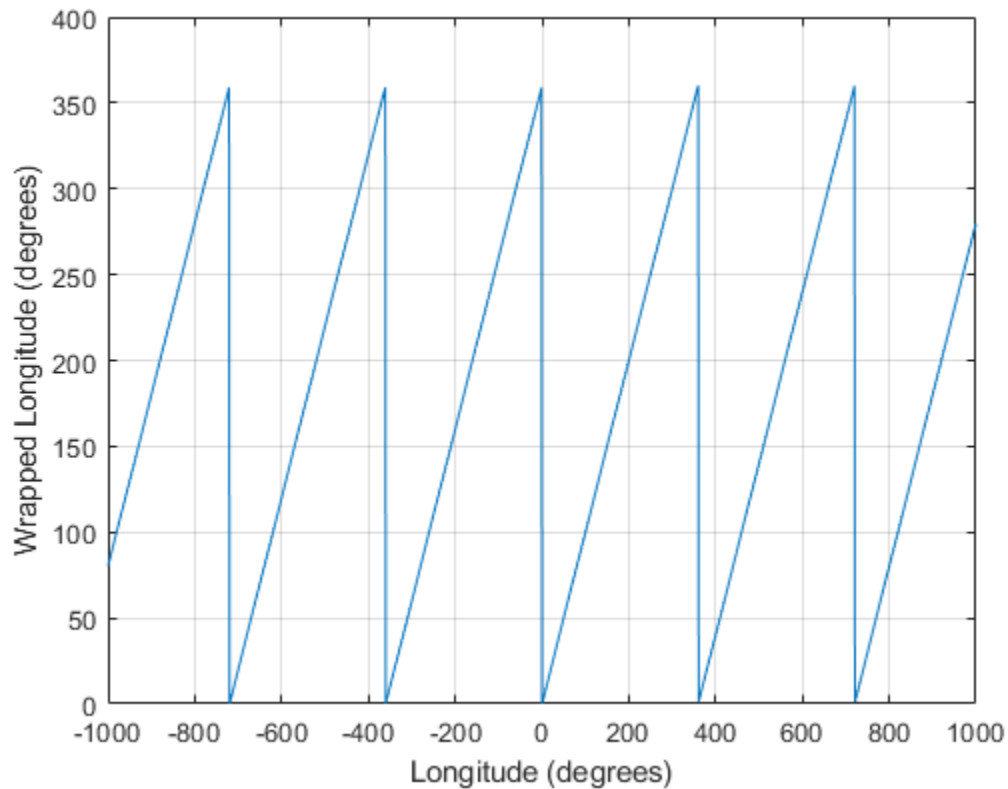
Input Arguments

lon — Angles
numeric vector

Angles, specified as a numeric vector.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

Output Arguments

lonWrapped — Wrapped angles
numeric vector

Wrapped angles, specified as a numeric vector with values in the range [0, 360].

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

wrapTo180 | wrapTo2Pi | wrapToPi

Introduced in R2007b

wrapTo2Pi

Wrap angle in radians to [0 2*pi]

Syntax

```
lambdaWrapped = wrapTo2Pi(lambda)
```

Description

`lambdaWrapped = wrapTo2Pi(lambda)` wraps angles in `lambda`, in radians, to the interval `[0, 2*pi]` such that 0 maps to 0 and `2*pi` maps to `2*pi`. In general, positive multiples of `2*pi` map to `2*pi` and negative multiples of `2*pi` map to 0.

Examples

Wrap Angles to 2Pi Radians

Specify a short list of angles to wrap.

```
lambda = [-2*pi -pi-0.1 -pi -2.8 3.1 pi pi+1 2*pi];
```

Wrap the angles to the range $[0, 2\pi]$ radians.

```
lambdaWrapped = wrapTo2Pi(lambda)
```

```
lambdaWrapped = 1×8

        0    3.0416    3.1416    3.4832    3.1000    3.1416    4.1416    6.2832
```

Specify a second list of angles, and wrap them.

```
lambda2 = -10:0.1:10;
lambda2Wrapped = wrapTo2Pi(lambda2);
```

Plot the wrapped angles. The wrapped angles stay in the range $[0, 2\pi]$ radians.

```
plot(lambda2,lambda2Wrapped)
xlabel("Angle (radians)")
ylabel("Wrapped Angle (radians)")
grid on
```
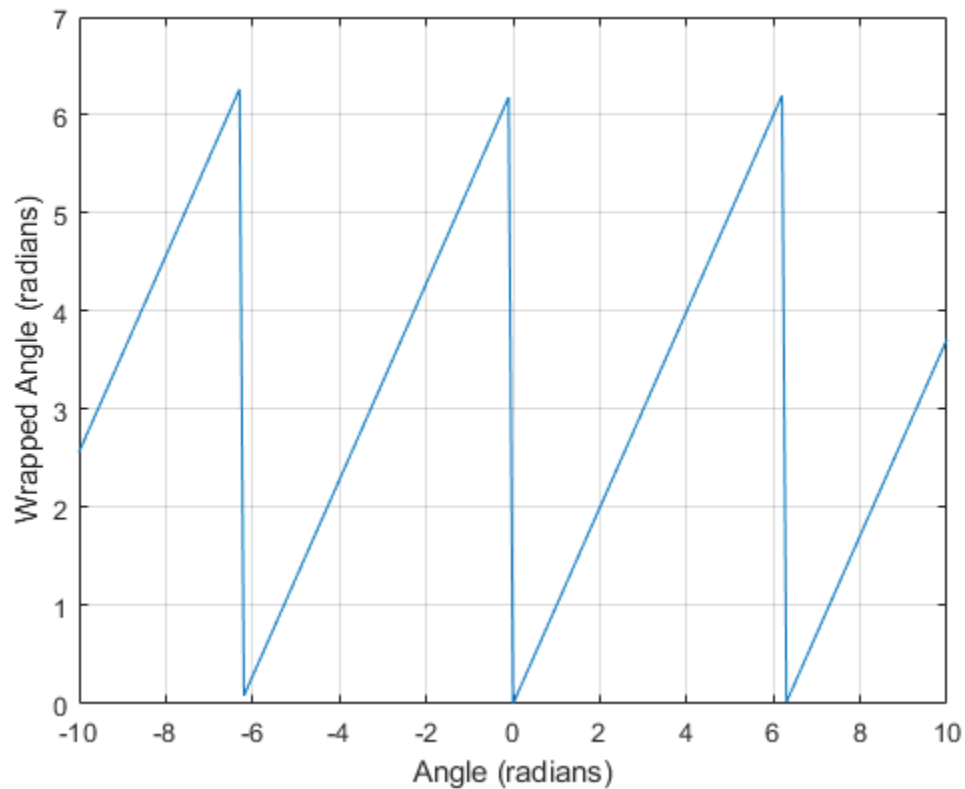
Input Arguments

lambda — Angles
numeric vector

Angles, specified as a numeric vector.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

Output Arguments

lambdaWrapped — Wrapped angles
numeric vector

Wrapped angles, specified as a numeric vector with values in the range `[0, 2*pi]`.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

wrapTo180 | wrapTo360 | wrapToPi

Introduced in R2007b

wrapToPi

Wrap angle in radians to [−pi pi]

Syntax

```
lambdaWrapped = wrapToPi(lambda)
```

Description

`lambdaWrapped = wrapToPi(lambda)` wraps angles in `lambda`, in radians, to the interval [−pi, pi] such that `pi` maps to `pi` and `−pi` maps to `−pi`. In general, odd, positive multiples of `pi` map to `pi` and odd, negative multiples of `pi` map to `−pi`.

Examples

Wrap Angles to Pi Radians

Specify a short list of angles to wrap.

```
lambda = [-2*pi -pi-0.1 -pi -2.8 3.1 pi pi+1 2*pi];
```

Wrap the angles to the range [-π, π] radians.

```
lambdaWrapped = wrapToPi(lambda)
```

```
lambdaWrapped = 1×8
```

```
      0    3.0416   -3.1416   -2.8000    3.1000    3.1416   -2.1416         0
```

Specify a second list of angles, and wrap them.

```
lambda2 = -10:0.1:10;
lambda2Wrapped = wrapToPi(lambda2);
```

Plot the wrapped angles. The wrapped angles stay in the range [-π, π] radians.

```
plot(lambda2,lambda2Wrapped)
xlabel("Angle (radians)")
ylabel("Wrapped Angle (radians)")
grid on
```

Input Arguments

`lambda` — Angles
numeric vector

Angles, specified as a numeric vector.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `logical`

Output Arguments

`lambdaWrapped` — Wrapped angles
numeric vector

Wrapped angles, specified as a numeric vector with values in the range `[-pi, pi]`.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

wrapTo180 | wrapTo360 | wrapTo2Pi

Introduced in R2007b

zdatam

Adjust *z*-plane of displayed map objects

Syntax

```
zdatam
zdatam(hndl)
zdatam('str')
zdatam(hndl,zdata)
zdatam('str',zdata)
```

Description

zdatam displays a GUI for selecting an object from the current axes and modifying its ZData property.

zdatam(hndl) and zdatam('str') display a GUI to modify the ZData of the object(s) specified by the input. str is any character vectors recognized by handlem.

zdatam(hndl,zdata) alters the *z*-plane position of displayed map objects designated by the MATLAB graphics handle hndl. The *z*-plane position may be the Z position in the case of text objects, or the ZData property in the case of other graphic objects. The function behaves as follows:

- If hndl is an hggroup handle, the ZData property of the children in the hggroup are altered.
- If the handle is scalar, then ZData can be either a scalar (*z*-plane definition), or a matrix of appropriate dimension for the displayed object.
- If hndl is a vector, then ZData can be a scalar or a vector of the same dimension as hndl.
- If ZData is a scalar, then all objects in hndl are drawn on the ZData *z*-plane.
- If ZData is a vector, then each object in hndl is drawn on the plane defined by the corresponding ZData element.
- If ZData is omitted, then a modal dialog box prompts for the ZData entry.

zdatam('str',zdata) identifies the objects by the input str, where str is any of the character vectors recognized by handlem, and uses zdata as described above to update their ZData property.

This function adjusts the *z*-plane position of selected graphics objects. It accomplishes this by setting the objects' ZData properties to the appropriate values.

See Also
handlem | setm

Introduced before R2006a

zero22pi

Wrap longitudes to [0 360] degree interval

Compatibility

Note The `zero22pi` function has been replaced by `wrapTo360` and `wrapTo2Pi`.

Syntax

```
newlon = zero22pi(lon)
newlon = zero22pi(lon,angleunits)
```

Description

`newlon = zero22pi(lon)` *wraps* the input angle `lon` in degrees to the 0 to 360 degree range.

`newlon = zero22pi(lon,angleunits)` works in the units defined by *angleunits,* which can be either `'degrees'` or `'radians'`. *angleunits* can be abbreviated and is case-insensitive.

Examples

```
zero22pi(567.5)

ans =
    207.5

zero22pi(-567.5)

ans =
    152.5

zero22pi(-7.5,'radian')

ans =
    5.0664
```

See Also
`wrapTo2Pi` | `wrapTo360`

Introduced before R2006a

zerom

(To be removed) Construct regular data grid of 0s

Note `zerom` will be removed in a future release. Use the `georefcells` and `zeros` functions instead. For more information, see "Compatibility Considerations".

Syntax

```
[Z,refvec] = zerom(latlim,lonlim,scale)
```

Description

`[Z,refvec] = zerom(latlim,lonlim,scale)` returns a full regular data grid consisting entirely of 0s and a three-element referencing vector for the returned Z. The two-element vectors `latlim` and `lonlim` define the latitude and longitude limits of the geographic region. They should be of the form `[south north]` and `[west east]`, respectively. The scalar `scale` specifies the number of rows and columns per degree of latitude and longitude.

Examples

```
[Z,refvec] = zerom([46,51],[-79,-75],1)

Z =
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
refvec =
     1    51   -79
```

Compatibility Considerations

zerom will be removed
Not recommended starting in R2015b

Some functions that return referencing vectors will be removed, including the `zerom` function. Instead, create a geographic raster reference object using the `georefcells` function and a matrix of zeros using the `zeros` function. Reference objects have several advantages over referencing vectors.

- Unlike referencing vectors, reference objects have properties that document the size of the associated raster, its geographic limits, and the direction of its rows and columns. For examples of reference object properties, see the `GeographicPostingsReference` object.

- You can manipulate the limits of rasters associated with geographic reference objects using the `geocrop` function.

- You can manipulate the size and resolution of rasters associated with geographic reference objects using the `georesize` function.

- Most functions that accept referencing vectors as inputs also accept reference objects.

This table shows how to update your code to use the `georefcells` and `zeros` functions instead of the `zerom` function.

Will Be Removed	Recommended
`[Z,refvec] = zerom(latlim,lonlim,scale);`	`R = georefcells(latlim,lonlim,1/scale,1/scale);` `Z = zeros(R.RasterSize);`

See Also
NaN | ones | sparse | georefcells | zeros

Introduced before R2006a

axesmui

Define map axes and modify map projection and display properties

Activation

Command Line	Maptool	Map Display
axesmui c = axesmui(...)	**Display** > **Projection**	extend-click map display

Description

`axesmui` activates a Projection Control dialog box for the current map axes. The dialog box allows map projection definition and property modification.

`c` is an optional output argument that indicates whether the Projection Control dialog box was closed by the cancel button. `c = 1` if the cancel button is pushed. Otherwise, `c = 0`.

Extend-clicking a map display brings up the Projection Control dialog box for that map axes.

Controls

The **Map Projection** pull-down menu is used to select a map projection. The projections are listed by type, and each is preceded by a four-letter type indicator:

```
Cyln = Cylindrical
Pcyl = Pseudocylindrical
Coni = Conic
Poly = Polyconic
Pcon = Pseudoconic
Azim = Azimuthal
Mazi = Modified Azimuthal
Pazi = Pseudoazimuthal
```

The **Zone** button and edit box are used to specify a zone for the Universal Transverse Mercator (UTM) and Universal Polar Stereographic (UPS) projections.

Note The **Zone** button and edit box are not supported in MATLAB Online. Programmatically create map axes instead. For example, create map axes using a UTM projection by specifying the `MapProjection` and `Zone` name-value pairs:

```
ax = axesm('MapProjection','UTM','Zone','54S');
```

The **Geoid** edit boxes and pull-down menu are used to specify the geoid. Units must be in meters for the UTM and UPS projections, since this is the standard unit for the two projections. For non-UTM and UPS projections, the geoid unit can be anything, bearing in mind that the resulting projected data will be in the same units as the geoid.

The **Angle Units** pull-down menu is used to specify the angle units used on the map projection. All angle entries corresponding to the current map projection must be entered in these units. Current angle entries are automatically updated when new angle units are selected.

The **Map Limits** edit boxes are used to specify the extent of the map data in geographic coordinates. The **Latitude** edit boxes contain the southern and northern limits of the map. The **Longitude** edit boxes contain the western and eastern limits of the map. The map limits establish the extent of the meridian and parallel grid lines, regardless of the display settings (see grid settings). Map limits are always in geographic coordinates, regardless of the map origin and orientation setting. In the normal aspect, the map display is trimmed to the minimum of the map and frame limits.

The **Frame Limits** edit boxes are used to specify the location of the map frame, measured from the center of the map projection in the base coordinate system. The **Latitude** edit boxes contain the southern and northern frame edge locations. The **Longitude** edit boxes contain the western and eastern frame edge locations. Displayed map data are trimmed at the frame limits. For azimuthal map projections, the latitude limits should be set to $-inf$ and the desired trim distance from the map origin. In the normal aspect, the map display is trimmed to the minimum of the map and frame limits.

The **Map Origin** edit boxes are used to specify the origin and aspect angle of the map projection. The **Lat** and **Long** boxes specify the map origin in geographic coordinates. This is the point that is placed in the center of the projection. If either box is left blank, 0 degrees is used. The **Orientation** box specifies the azimuth angle of the North Pole relative to the map origin. Azimuth is measured clockwise from the top of the projection. If the **Orientation** box is disabled, then the selected map projection requires a fixed orientation. See the *Mapping Toolbox User's Guide* for a complete description of the map origin.

The **Cartesian Origin** edit boxes are used to specify the *x-y* offset, along with a desired scale factor of the map projection. The **False E and N** boxes specify the false easting and northing in Cartesian

coordinates. These must be in the same units as the geoid. The **Scalefactor** box specifies the scale factor used in the map projection calculations.

The **Parallels** edit boxes specify the standard parallels of the selected map projection. A particular map projection may have one or two standard parallels. If the edit boxes are disabled, then the selected projection has no standard parallels or the standard parallels are fixed.

The **Aspect** pull-down menu is used to select a normal or transverse display aspect. When the aspect is normal, *north* (on the base projection) is up, and the map is displayed in a *portrait* setting. In a transverse aspect, north (in the base projection) is to the right, and the map is displayed in a *landscape* setting. This property does not control the map projection aspect. The projection aspect is determined by the map `Origin` property).

The **Frame** button brings up the Map Frame Properties dialog box, which allows the map frame settings to be modified.

The **Grid** button brings up the Map Grid Properties dialog box, which allows the map grid settings to be modified.

The **Labels** button brings up the Map Label Properties dialog box, which allows the parallel and meridian label settings to be modified.

The **Fill in** button is used to compute projection and display settings based on any currently specified map parameters. Only settings that are left blank are affected when this button is pushed.

The **Reset** button is used to reset the default projection properties and display settings of the current map. Default display settings include frame, grid, and label properties set to `'off'`.

The **Apply** button is used to apply the projection and display settings to the current map, which results in the map being reprojected.

The **Help** button is used to bring up online help text for each control on the Projection Control dialog box.

The **Cancel** button disregards any modified projection or display settings and closes the Projection Control dialog box.

Map Frame Properties Dialog Box

This dialog box allows modification of the map frame settings. It is accessed via the **Frame** button on the Projection Control dialog box.
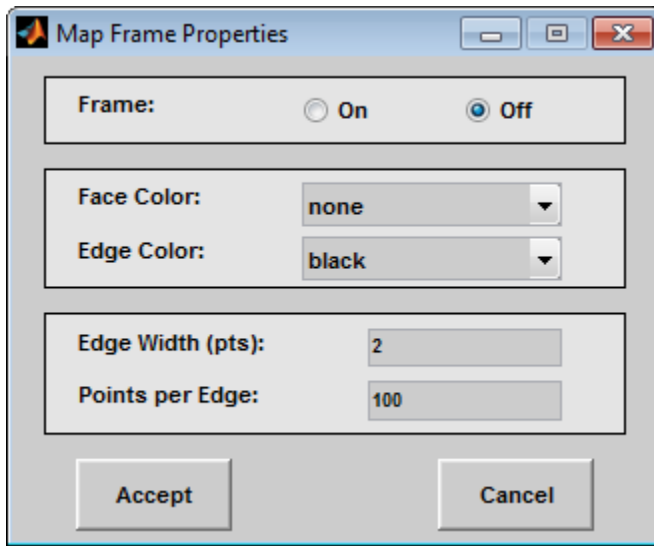
The **Frame** selection buttons determine whether the map frame is visible.

The **Face Color** pull-down menu is used to select the background color of the map frame. Selecting none results in a transparent frame background, i.e., the same as the axes color. Selecting custom allows a custom RGB triple to be defined for the background color.

The **Edge Color** pull-down menu is used to select the color of the frame edge. Selecting none hides the frame edge. Selecting custom allows a custom RGB triple to be defined for the edge color.

The **Edge Width** edit box is used to enter the line width of the frame edge, in points.

The **Points per Edge** edit box is used to enter the number of points used to display each edge of the map frame.

The **Accept** button accepts any modifications made to the map frame properties and returns to the Projection Control dialog box. Changes are applied to the current map only when the **Apply** button on the Projection Control dialog box is pushed.

The **Cancel** button disregards any modifications to the map frame properties and returns to the Projection Control dialog box.

Map Grid Properties Dialog Box

This dialog box allows modification of the map frame settings. It is accessed via the **Grid** button on the Projection Control dialog box.
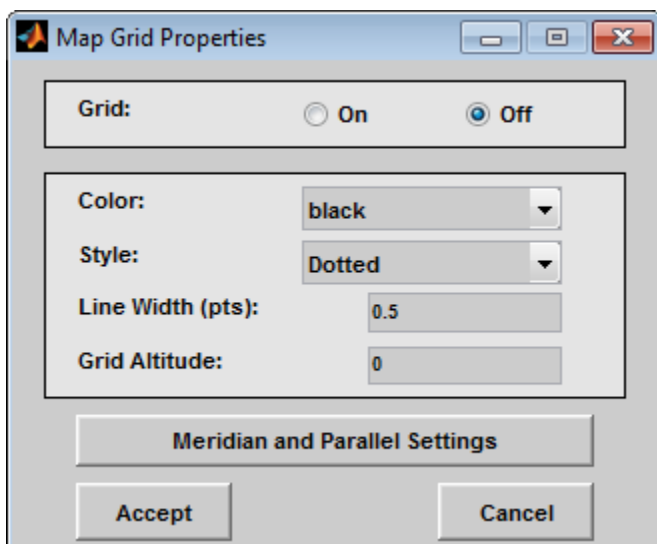
The Grid selection buttons determine whether the map grid is visible.

The **Color** pull-down menu is used to select the color of the map grid lines. Selecting `custom` allows a custom RGB triple to be defined for the grid line color.

The **Style** pull-down menu is used to select the line style of the map grid lines.

The **Line Width** edit box is used to enter the width of the map grid lines, in points.

The **Grid Altitude** edit box is used to enter *z*-axis location of the map grid. This property can be used to place some mapped objects above or below the map grid. The default map grid altitude is `inf`, which places the grid above all other mapped objects.

The **Meridian and Parallel Settings** button brings up the **Meridian and Parallel Properties** dialog box, which allows the properties of the meridian and parallel grid lines to be modified.

The **Accept** button accepts any modifications made to the map grid properties and returns to the Projection Control dialog box. Changes are applied to the current map only when the **Apply** button on the Projection Control dialog box is pushed.

The **Cancel** button disregards any modifications to the map grid properties and returns to the Projection Control dialog box.

Meridian and Parallel Properties Dialog Box

This dialog box is used to modify the settings for meridian and parallel grid lines. It is accessed via the **Meridian and Parallel Settings** button on the Map Grid Properties dialog box.
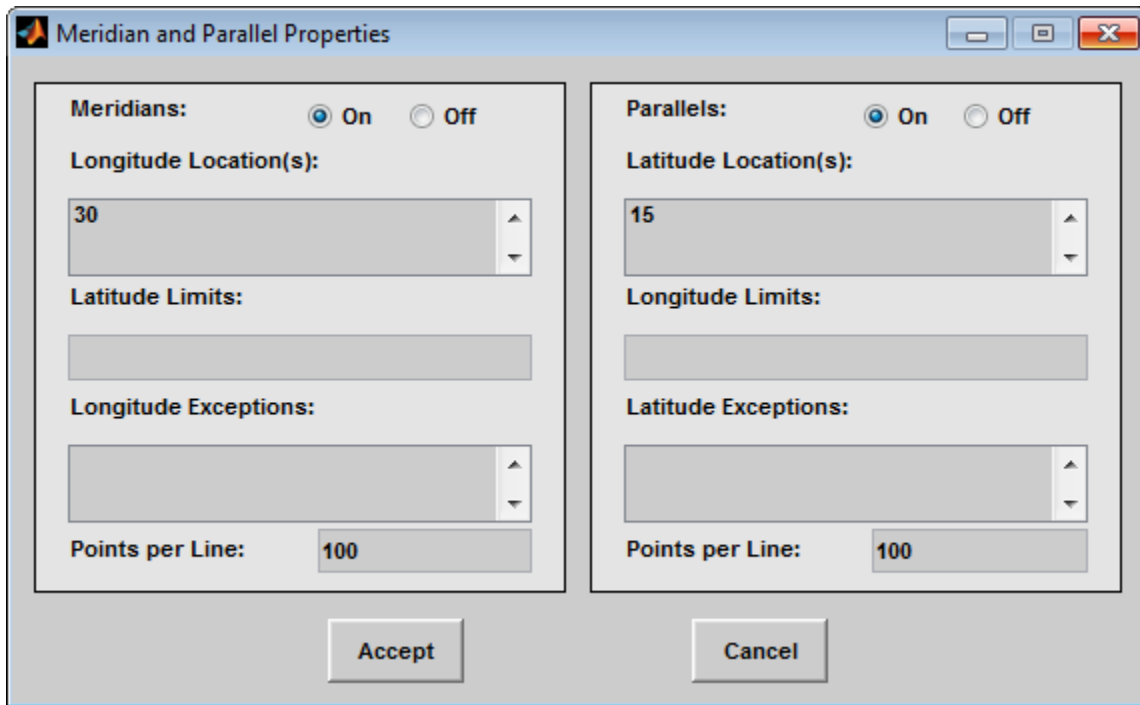
The **Meridians** selection buttons determine whether the meridian grid lines are visible when the map grid is turned on.

The **Longitude Location(s)** edit box is used to specify which meridians are to be displayed if the meridian lines are turned on. If a scalar interval value is entered, meridian lines are displayed at that interval, starting from the Prime Meridian and proceeding in east and west directions. If a vector of values is entered, meridian lines are displayed at locations given by each element of the vector.

The **Latitude Limits** edit box is used to specify the latitude limits beyond which meridian lines do not extend. If this property is left empty, all meridian lines extend to the map latitude limits (specified by the Latitude Map Limits entry on the **Projection Control** dialog box). This entry must be a two-element vector enclosed in brackets.

The **Longitude Exceptions** edit box is used to enter specific meridians of the displayed grid that are to extend beyond the latitude limits, to the map limits. This entry is a vector of longitude values.

The **Parallels** selection buttons determine whether the parallel grid lines are visible when the map grid is turned on.

The **Latitude Location(s)** edit box is used to specify which parallels are to be displayed if the parallel lines are turned on. If a scalar interval value is entered, parallel lines are displayed at that interval, starting from the Equator and proceeding in north and south directions. If a vector of values is entered, parallel lines are displayed at locations given by each element of the vector.

The **Longitude Limits** edit box is used to specify the longitude limits beyond which parallel lines do not extend. If this property is left empty, all parallel lines extend to the map longitude limits (specified by the Longitude Map Limits entry on the **Projection Control** dialog box). This entry must be a two-element vector enclosed in brackets.

The **Latitude Exceptions** edit box is used to enter specific parallels of the displayed grid that are to extend beyond the longitude limits, to the map limits. This entry is a vector of latitude values.

The **Points per Line** edit boxes are used to enter the number of points used to plot each meridian and each parallel grid line. The default value is 100 points.

The **Accept** button accepts any modifications that have been made to the meridian and parallel grid line properties and return to the Map Grid Properties dialog box. Changes are applied to the current map only when the **Apply** button on the Projection Control dialog box is pushed.

The **Cancel** button disregards any modifications to the meridian and parallel grid lines and returns to the Map Grid Properties dialog box.

Map Label Properties Dialog Box

This dialog box is used to modify the settings of the meridian and parallel labels. It is accessed via the **Label** button on the Projection Control dialog box.
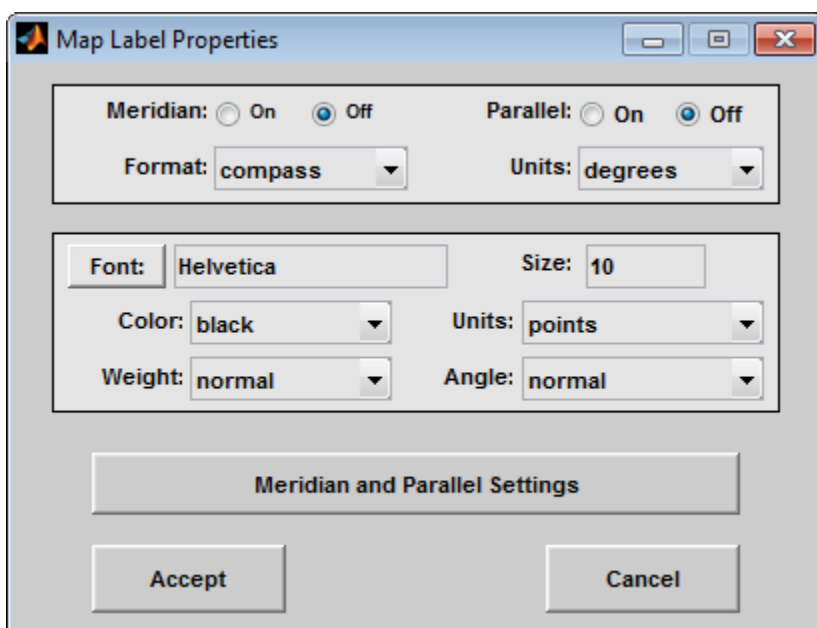
The **Meridian** and **Parallel** selection buttons determine whether the meridian and parallel labels are visible.

The **Format** pull-down menu is used to specify the format of the grid labels. If `compass` is selected, meridian labels are appended with `E` for east and `W` for west, and parallel labels are appended with `N` for north and `S` for south. If `signed` is chosen, meridian labels are prefixed with `+` for east and `-` for west, and parallel labels are prefixed with `+` for north and `-` for south. If `none` is selected, western meridian labels and southern parallel labels are prefixed by `-`, but no symbol precedes eastern meridian labels and northern parallel labels.

The label **Units** pull-down menu is used to specify the angle units used to display the parallel and meridian labels. These units, used for display purposes only, need not be the same as the angle units of the map projection.

The **Font** edit box is used to specify the character font used to display the parallel and meridian labels. If the font specified does not exist on the computer, the default of `Helvetica` is used. Pressing the **Font** button previews the selected font.

The font **Size** edit box is used to enter an integer value that specifies the font size of the parallel and meridian labels. This value must be in the units specified by the font **Units** pull-down menu.

The font **Color** pull-down menu is used to select the color of the parallel and meridian labels. Selecting `custom` allows a custom RGB triple to be defined for the labels.

The font **Weight** pull-down menu is used to specify the character weight of the parallel and meridian labels.

The font **Units** pull-down menu is used to specify the units used to interpret the font size entry. When set to `normalized`, the value entered in the **Size** edit box is interpreted as a fraction of the height of the axes. For example, a normalized font size of 0.1 sets the label text to a height of one tenth of the axes height.

The font **Angle** pull-down menu is used to select the character slant of the parallel and meridian labels. `normal` specifies nonitalic font. `italic` and `oblique` specify italic font.

The **Meridian and Parallel Settings** button brings up the Meridian and Parallel Label Properties dialog box, which allows modification of properties specific to the meridian and parallel grid labels.

The **Accept** button accepts any modifications that have been made to the map label properties and returns to the Projection Control dialog box. Changes are applied to the current map only when the **Apply** button on the Projection Control dialog box is pushed.

The **Cancel** button disregards any modifications to the map labels and returns to the **Projection Control** dialog box.

Meridian and Parallel Label Properties Dialog Box

This dialog box is used to modify properties specific to the meridian and parallel grid labels. It is accessed via the **Meridian and Parallel Settings** button on the Map Label Properties dialog box.
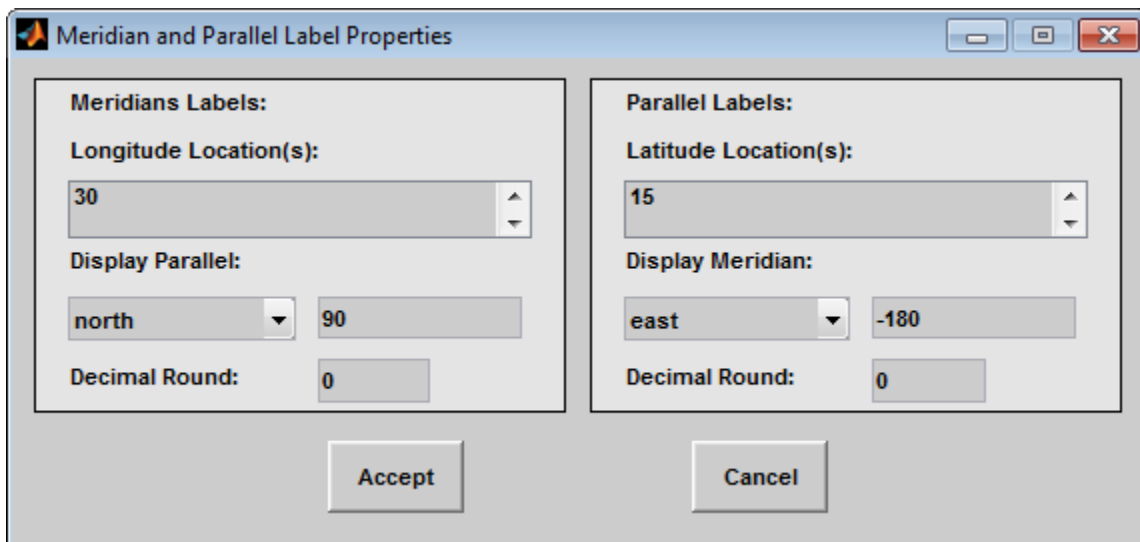
The **Longitude Location(s)** edit box is used to specify which meridians are to be labeled. Meridian labels need not coincide with displayed meridian grid lines. If a scalar interval value is entered, labels are displayed at that interval, starting from the Prime Meridian and proceeding in east and west

directions. If a vector of values is entered, labels are displayed at longitude locations given by each element of the vector.

The **Display Parallel** pull-down menu and edit box are used to specify the latitude location of the meridian labels. If a scalar latitude value is provided in the edit box, the meridian labels are placed at that parallel. Alternatively, the pull-down menu can be used to select a latitude location. If `north` is chosen, meridian labels are placed at the maximum map latitude limit. If `south` is chosen, meridian labels are placed at the minimum map latitude limit.

The **Latitude Location(s)** edit box is used to specify which parallels are to be labeled. Parallel labels need not coincide with displayed parallel grid lines. If a scalar interval value is entered, labels are displayed at that interval, starting from the Equator and proceeding in north and south directions. If a vector of values is entered, labels are displayed at latitude locations given by each element of the vector.

The **Display Meridian** pull-down menu and edit box are used to specify the longitude location of the parallel labels. If a scalar longitude value is provided in the edit box, the parallel labels are placed at that meridian. Alternatively, the pull-down menu can be used to specify a longitude location. If `east` is chosen, parallel labels are placed at the maximum map longitude limit. If `west` is chosen, parallel labels are placed at the minimum map longitude limit.

The **Decimal Round** edit boxes are used to specify the power of ten to which the meridian and parallel labels are rounded. For example, a value of -1 results in labels displayed to the tenths decimal place.

The **Accept** button accepts any modifications that have been made to the meridian and parallel label properties and return to the Map Label Properties dialog box. Changes are applied to the current map only when the **Apply** button on the **Projection Control** dialog box is pushed.

The **Cancel** button disregards any modifications to the meridian and parallel labels and returns to the Map Label Properties dialog box.

The **Map Geoid** edit box is used to specify the geoid (ellipsoid) definition for the current map axes. The geoid is defined by a two-element vector of the form `[semimajor-axis eccentricity]`. Eccentricity must be a value between 0 and 1, but not equal to 1. A nonzero eccentricity represents an ellipsoid. The default geoid is a sphere with radius 1, represented as `[1 0]`. If a scalar entry is provided, it is assumed to be the radius of a sphere.

The **Accept** button accepts any modifications that have been made to the map geoid and return to the Projection Control dialog box. Changes are applied to the current map only when the **Apply** button on the Projection Control dialog box is pushed.

The **Cancel** button disregards any modifications to the map geoid and returns to the Projection Control dialog box.

See Also
axesm

Introduced in R2007a

clmo-ui

GUI to clear mapped objects

Activation

Command Line	Maptool
clmo	**Tools** > **Delete** > **Object**

Description

clmo brings up a Select Object dialog box for selecting mapped objects to delete.

Controls

The scroll box is used to select the desired objects from the list of mapped objects.

Pushing the **Select all** button highlights all objects in the scroll box for selection. Pushing the **OK** button deletes the selected objects from the map. Pushing the **Cancel** button aborts the operation.

See Also
clmo

Introduced in R2007a

clrmenu

Add colormap menu to figure window

Activation

Command Line
`clrmenu`
`clrmenu(h)`

Description

`clrmenu` adds a colormap menu to the current figure.

`clrmenu(h)` adds a colormap menu to the figure specified by the handle `h`.

Controls

The following choices are included on the colormap menu:

Parula, **Gray**, **Hsv**, **Hot**, **Pink**, **Cool**, **Bone**, **Jet**, **Copper**, **Spring**, **Summer**, **Autumn**, **Winter**, **Flag**, and **Prism** generate colormaps.

Rand is a random colormap.

Brighten increases the brightness.

Darken decreases the brightness.

Flipud inverts the order of the colormap entries.

Fliplr interchanges the red and blue components.

Permute permutes the colormap: red > blue, blue > green, green > red.

Define allows a workspace variable to be specified for the colormap.

Remember stores the current colormap.

Restore reverts to the stored colormap (initially, the stored colormap is the colormap in use when `clrmenu` is invoked).

Refresh redraws the current figure window.

Digital Elevation activates the DEM Colormap Input dialog box. Use it to specify a colormap for a digital elevation map, and then apply the colormap to the current figure. The number of land and sea colors in the colormap is appropriate for the maximum elevations and depths of the data grid. The dialog box is shown and described below:
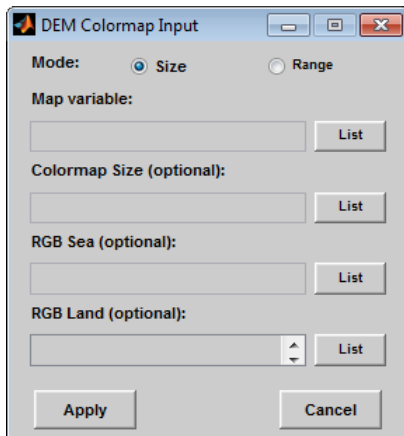
The **Mode** selection buttons are used to specify whether the length of the colormap is specified or whether the altitude range increment assigned to each color is specified.

The **Map variable** edit box is used to specify the data grid containing the elevation data.

The **Colormap Size** edit box is used in Size mode. This entry defines the length of the colormap. If omitted, a default length of 64 is used. This entry must be a scalar value.

The **Altitude Range** edit box is used in Range mode. This entry defines the altitude range increment assigned to each color. If omitted, a default increment of 100 is used. This entry must be a scalar value.

The **RGB Sea** edit box is used to define colors for data with negative values. The actual sea colors of the generated colormap are interpolated from this matrix. This entry can be a matrix of any length (n-by-3) The colormap matrix of the current figure can be used by entering `'window'` in this box. The `demcmap` function provides default sea colors, which are used if this entry is left blank.

The **RGB Land** edit box is used to define colors for data with positive values. The actual land colors of the generated colormap are interpolated from this matrix. This entry can be a matrix of any length (n-by-3). The colormap matrix of the current figure can be used by entering `'window'` in this box. The `demcmap` function provides default sea colors, which are used if this entry is left blank.

Pressing the **Apply** button accepts the input data, creates the colormap, and assigns it to the current figure.

Pressing the **Cancel** button disregards any input data and closes the DEM Colormap Input dialog box.

See Also
demcmap

Introduced before R2006a

colorm

Create index map colormaps

Note `colorm` will be removed in a future release.

Syntax

`colorm(datagrid,refvec)`

Description

`colorm(datagrid,refvec)` displays the data grid in a new figure window and allows a colormap to be edited and saved to a new variable. `datagrid` and `refvec` are the data grid and the referencing vector of the surface. `map` must have positive index values into the colormap.

Controls
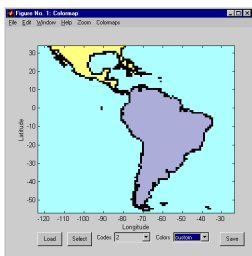
The `colorm` tool displays the surface map data in a new figure window with the current colormap. **Zoom** and **Colormaps** menus are activated for that figure.

The **Zoom On/Off** menu toggles interactive zoom on and off.

The **Colormaps** menu provided a variety of colormap options that can be applied to the map. See `clrmenu` in this guide for a description of the **Colormaps** menu options.

The **Load** button activates a dialog box, used to specify a colormap variable to be applied to the displayed surface map. This colormap can then be edited and saved.

The **Select** button activates the mouse cursor and allows a point on the map to be selected. The value of that point then appears in the **Codes** pull-down menu. The color of the selected point appears in the **Color** pull-down menu and can then be edited.

The **Codes** pull-down menu is used to select a particular value in the data grid. The color associated with that value then appears in the **Color** pull-down menu and can be edited.

The **Color** pull-down menu is used to select a particular color to assign to the value currently displayed in the Codes pull-down menu. A custom color can be defined by selecting the `custom` option. This brings up a custom color interface with which an RGB triple can be selected.

The **Save** button is used to save the modified colormap to the workspace. A dialog box appears in which the colormap variable name is entered.

See Also

encodem | getseeds | maptrim | seedm

Introduced before R2006a

demdataui

UI for selecting digital elevation data

Syntax

demdataui

Description

demdataui is a graphical user interface to extract digital elevation map data from a number of external data files. You can extract data to MAT-files or the base workspace as regular data grids with referencing vectors.

The demdataui panel lets you read data from a variety of high-resolution digital elevation maps (DEMs). These DEMs range in resolution from about 10 kilometers to 100 meters or less. The data files are available over the Internet at no cost, or (in some cases) on CD-ROMs for varying fees. demdataui reads ETOPO5, TerrainBase, GTOPO30, GLOBE, satellite bathymetry from Smith and Sandwell, and DTED data. demdataui looks for these geospatial data files on the MATLAB path and, for some operating systems, on CD-ROM disks.

Note When it opens, demdataui scans your Mapping Toolbox path for candidate data files. On PCs, it also checks the root directories of CD-ROMs and other drives, including mapped network drives. This can cause a delay before the GUI appears.

You use the list to select the source of data and the map to select the region of interest. When you click the **Get** button, data is extracted and displayed on the map. Use the **Save** button to save the data in a MAT-file or to the base workspace for later display. The **Close** button closes the window.

Controls
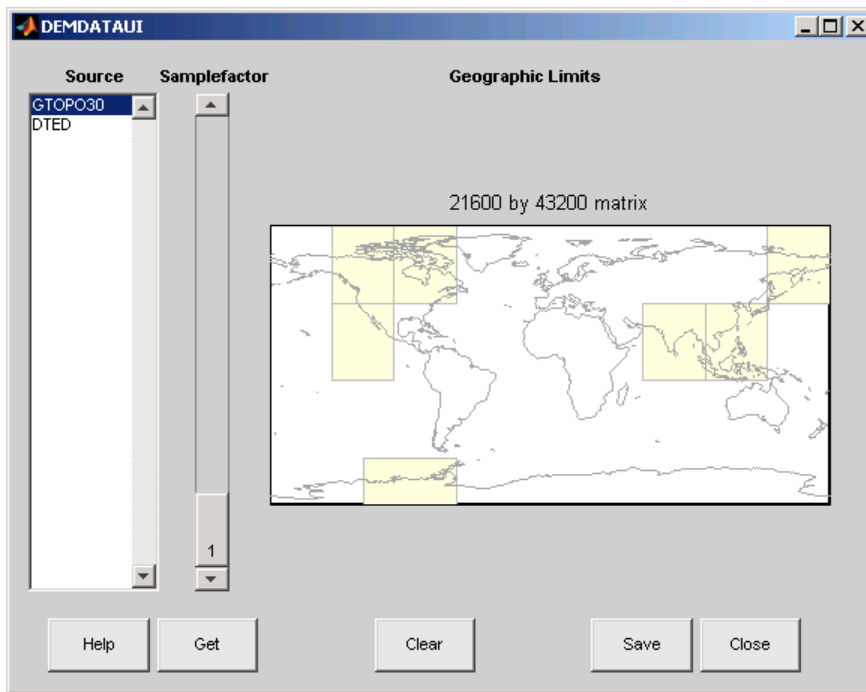
The Map

The map controls the geographic extent of the data to be extracted. `demdataui` extracts data for areas currently visible on the map. Use the mouse to zoom in or out to the area of interest. See `zoom` for more on zooming.

Some data sources divide the world up into tiles. When extracting, data is concatenated across all visible tiles. The map shows the tiles in light yellow with light gray edges. When data resolution is high, extracting data for large area can take much time and memory. An approximate count of the number of points is shown above the map. Use the **Samplefactor** slider to reduce the amount of data.

The List

The list controls the source of data to be extracted. Click a name to see the geographic coverage in light yellow. The sources list shows the data sources found when `demdataui` started.

`demdataui` searches for data files on the MATLAB path. On some computers, `demdataui` also checks for data files on the root level of letter drives. `demdataui` looks for the following data:

*   `etopo5`: The files `new_etopo5.bil` or `etopo5.northern.bat` and `etopo5.southern.bat`.
*   `tbase`: The file `tbase.bin`.
*   `satbath`: The file `topo_6.2.img`.
*   `gtopo30`: A folder that contains subfolders with the data files. For example, `demdataui` would detect GTOPO30 data if a folder on the path contained the folders E060S10 and E100S10, each of which holds the uncompressed data files.
*   `globedem`: A folder that contains data files and in the subfolder `/esri/hdr` and the `*.hdr` header files.

- `dted`: A folder that has a subfolder named DTED. The contents of the DTED folder are more subfolders organized by longitude and, below that, the DTED data files for each latitude tile.

The Samplefactor Slider

The **Sample Factor** slider allows you to reduce the density of the data. A sample factor of 2 returns every second point. The current sample factor is shown on the slider.

The Get Button

The **Get** button reads the currently selected data and displays it on the map. Press the standard interrupt key combination for your platform to interrupt the process.

The Clear Button

The **Clear** button removes any previously read data from the map.

The Save Button

The **Save** button saves the currently displayed data to a MAT-file or the base workspace. If you choose to save to a file, you will be prompted for a file name and location. If you choose to save to the base workspace, you can choose the variable name under which the data will be stored. The `demdataui` function returns one or more matrices as an array of display structures, having one element for each separate *get* you requested (assuming you did not subsequently **Clear**).

Data are returned as Mapping Toolbox Version 1 display structures. For information about display structure format, see "Version 1 Display Structures" on page 1-286 in the reference page for `displaym`.

Use `load` and `displaym` to redisplay the data from a file on a map axes. To display the data in the base workspace, use `displaym`. To gain access to the data matrices, subscript into the structure (for example, `datagrid = demdata(1).map; refvec = demdata(1).maplegend`. Use `worldmap` to create easy displays of the elevation data (for example, `worldmap(datagrid,refvec)`. Use `meshm` to add regular data grids to existing displays, or `surfm` or a similar function for geolocated data grids (for example, `meshm(datagrid,refvec)` or `surfm(latgrat,longrat,z)`.

The Close Button

The **Close** button closes the `demdataui` panel.

Examples

Read Elevation Data Interactively

Read from data sets that `demdataui` has located. You will not necessarily have all the DEM data sets shown in this example. For information about finding data sets over the Internet, see "Find Geospatial Raster Data".

Open the `demdataui` UI. It scans the path for data before it is displayed.

```
demdataui
```

The **Source** list shows the data sets that were found. Here, the source is selected to present all DTED files available to a user.
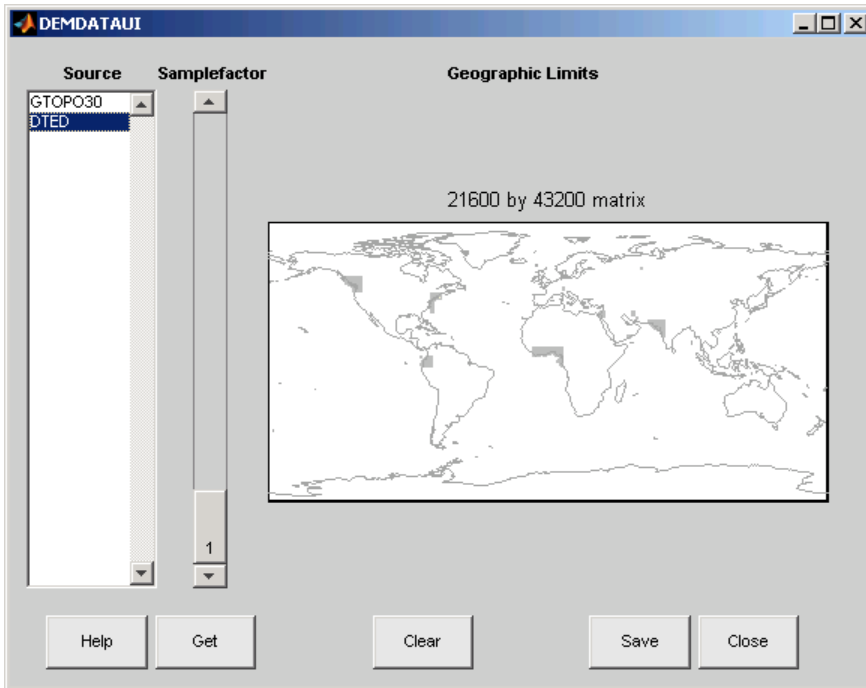
Click **GTOPO30**. In this example, there are available GTOPO30 tiles. The coverage of each data set is indicated by a yellow tint on the map with gray borders around each tile of data.
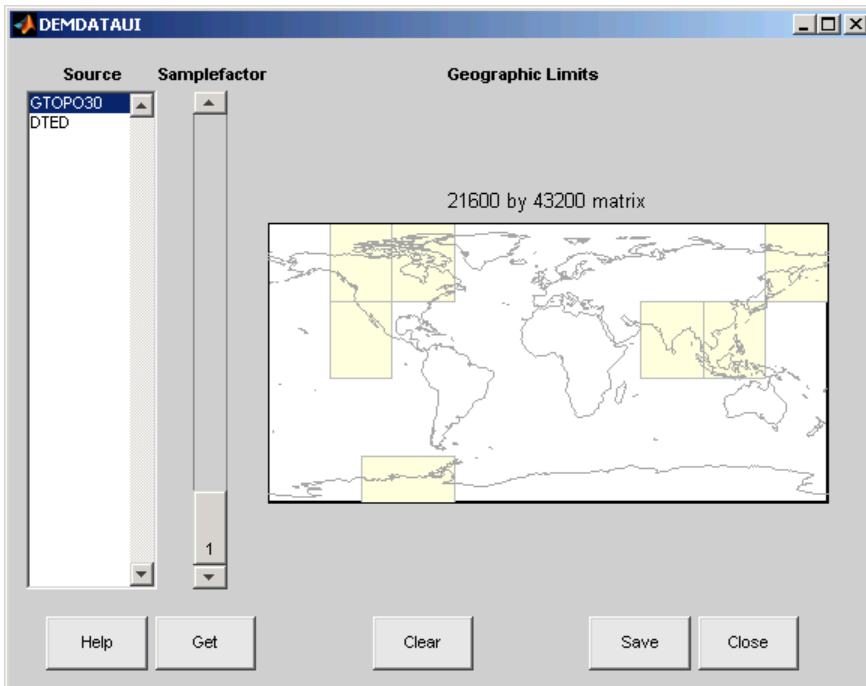
Use the map in the UI to specify the location and density of data to extract. To interactively set a region of interest, click in the map to zoom by a factor of two centered on the cursor, or click and drag across the map to define a rectangular region. The size of the matrix of the area currently displayed is printed above the map. To reduce the amount of data, you can continue to zoom in, or

you can raise the **Samplefactor** slider. A sample factor of 1 reads every point, 2 reads every other point, 3 reads every third point, etc. The matrix size is updated when you move the **Samplefactor** slider.

Here is the UI panel after selecting GTOPO30 data and zooming in on the Indian subcontinent.
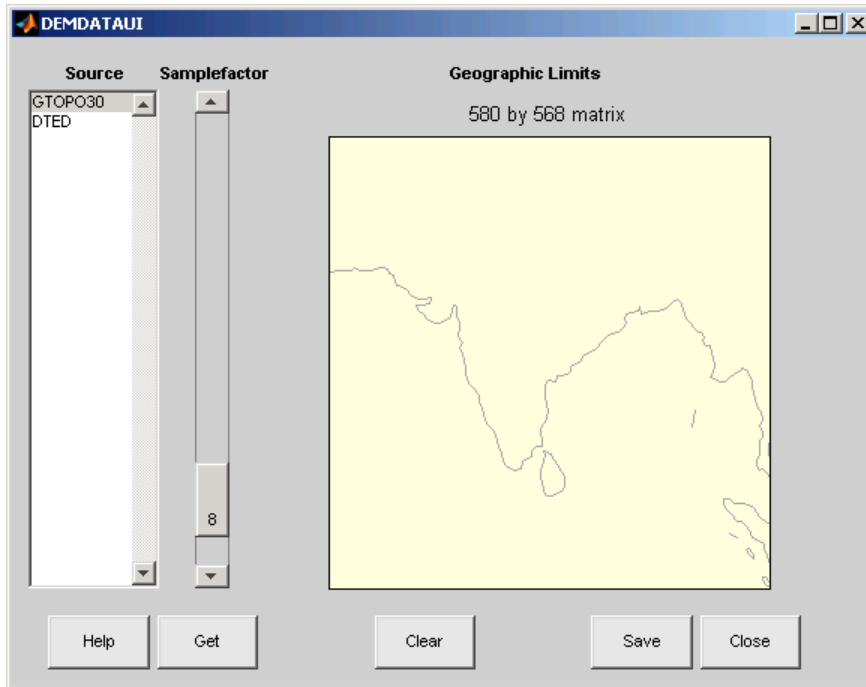
To see the terrain you have windowed at the sample factor you specified, click the **Get** button. This causes the GUI map pane to repaint to display the terrain grid with the `demcmap` colormap. In this example, the data grid contains 580-by-568 data values, as shown below.
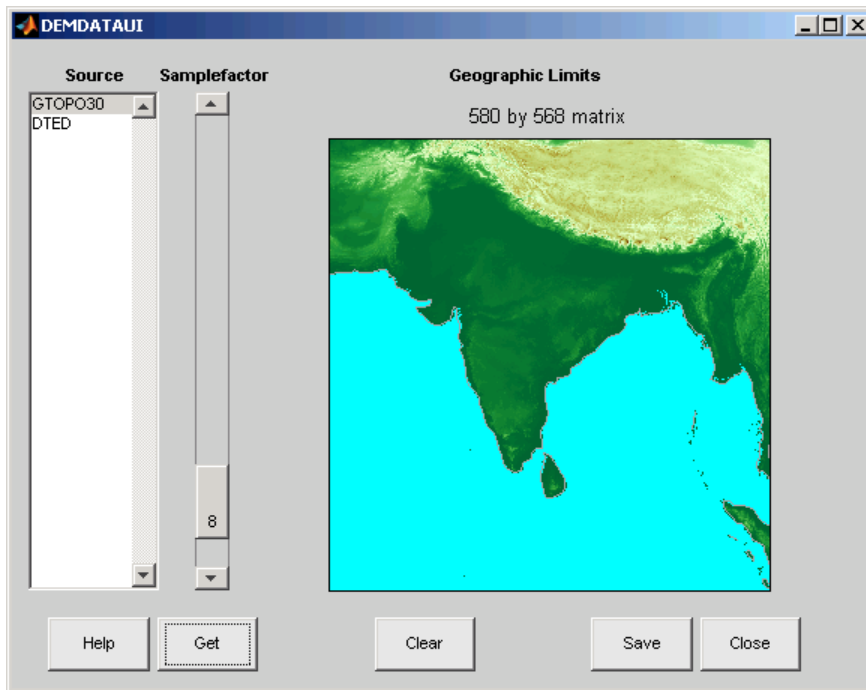
If you are not satisfied with the result, click the **Clear** button to remove all data previously read in via **Get** and make new selections. You might need to close and reopen `demdatui` in order to select a new region of interest.

When you are ready to import DEM data to the workspace or save it as a MAT-file, click the **Save** button.

Tips

- If `demdatui` does not recognize data you think it should find, check your path and click **Help** to read about how files are identified.
- You can add the data grids to map axes using the `geoshow` or `mlayers` functions.
- Updating the data returned by `demdatui` to geographic data structures (geostructs) using the `updategeostruct` function is not supported because they are of type `surface`.

See Also
`readgeoraster` | `vmap0ui`

Introduced before R2006a

handlem-ui

GUI for selecting mapped objects

Activation

Command Line
h = handlem
h = handlem('prompt')

Description

h = handlem brings up a Select Object dialog box, which lists all currently displayed objects. Returns the selected objects.

h = handlem('prompt') brings up a Specify Object dialog box, which allows greater control of object selection.

Controls

Select Object Dialog Box

The scroll box is used to select the desired objects from the list of mapped objects. Pushing the **Select all** button highlights all objects in the scroll box for selection. Pushing the **OK** button returns the object in the variable h. Pushing the **Cancel** button aborts the operation.

Specify Object Dialog Box
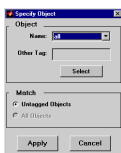
The **Object** Controls are used to select an object type or tag. The **Name** pull-down menu is used to select from a list of predefined objects. The **Other Tag** edit box is used to specify an object tag not listed in the **Name** pull-down menu. Pushing the **Select** button brings up the Select Object dialog box, which shows only the currently displayed objects for selection.

The **Match** Controls are used when a graphics object type (image, line, surface, patch, or text) is specified. The **Untagged Objects** selection button is used to return only those objects with empty tag properties. The **All Objects** selection button is used to return all objects of the specified type, regardless of whether they are tagged.

Pushing the **Apply** button returns the specified objects. Pushing the **Cancel** button aborts the operation.

See Also

`handlem`

Introduced in R2007a

hidem-ui

Hide specified mapped objects

Activation

Command Line	Maptool
`hidem`	**Tools** > **Hide** > **Object**

Description

`hidem` brings up a Select Object dialog box for selecting mapped objects to hide (`Visible` property set to `'off'`).

Controls

The scroll box is used to select the desired objects from the list of mapped objects. Pushing the **Select all** button highlights all objects in the scroll box for selection. Pushing the **OK** button changes the `Visible` property of the selected objects to `'off'`. Pushing the **Cancel** button aborts the operation without changing any properties of the selected objects.

See Also
`hidem`

Introduced in R2007a

lightmui

Control position of lights on globe or 3-D map

Compatibility

Note `lightmui` has been removed. Use `lightm` instead.

Syntax

`lightmui(hax)`

Description

`lightmui(hax)` creates a GUI to control the position of lights on a globe or 3-D map in map axes `hax`. You can control the position of lights by clicking and dragging the icon or by dialog boxes. Right-click the appropriate icon in the GUI to invoke the corresponding dialog box. You can change the light color by entering the RGB components manually or by clicking the pushbutton.

See Also
`lightm`

Introduced before R2006a

maptool

Add menu-activated tools to map figure

Activation

Command Line
maptool(*PropertyName*,PropertyValue)
maptool(*ProjectionFile*,...)
h = maptool(...)

Description

`maptool` adds several items to the menu in the current figure window with a map axes and opens the Projection Control dialog box for defining map projection and display properties. The figure window features a special menu bar that provides access to most of Mapping Toolbox capabilities.

`maptool(*PropertyName*,PropertyValue,...)` creates a figure window with a map axes defined by the supplied map properties. The `MapProjection` property must be the first input pair. `maptool` supports the same map properties as `axesm`.

`maptool(*ProjectionFile*,*PropertyName*, PropertyValue,...)` allows for the omission of the `MapProjection` property name. *ProjectionFile* must be the identifier of an available map projection.

`h = maptool(...)` returns a two-element vector containing the handle of the `maptool` figure window and the handle of the map axes.

Controls
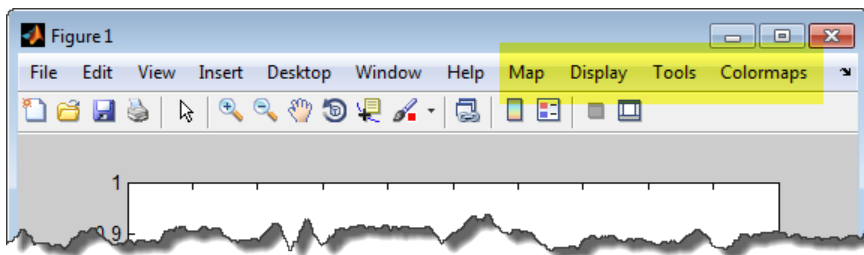
Map Menu

The **Lines** option activates the Line Map Input dialog box for projecting two- and three-dimensional line objects onto the map axes.

The **Patches** option activates the Patch Map Input dialog box for projecting patch objects onto the map axes.

The **Regular Surfaces** option activates the Mesh Map Input dialog box for projecting a regular data grid onto a graticule projected onto the map axes.

The **General Surfaces** option activates the Surface Map Input dialog box for projecting a geolocated data grid onto the map axes.

The **Contours** option activates the Contour Map Input dialog box for projecting a two- or three-dimensional contour plot onto the map axes.

The **Quiver 2D** option activates the Quiver Map Input dialog box for projecting a two-dimensional quiver plot onto the map axes.

The **Quiver 3D** option activates the Quiver3 Map Input dialog box for projecting a three-dimensional quiver plot onto the map axes.

The **Stem** option activates the Stem Map Input dialog box for projecting a stem plot onto the map axes.

The **Scatter** option activates the Scatter Map Input dialog box for projecting a scatter plot onto the map axes.

The **Text** option activates the Text Map Input dialog box for projecting text objects onto the map axes.

Display Menu

The **Projection** option activates the Projection Control dialog box for editing map projection properties and map display settings.

The **Graticule** option is used to view and edit the graticule size for surface maps.

The **Legend** option is used to display a contour map legend.

The **Frame** option is used to toggle the map frame on and off.

The **Grid** option is used to toggle the map grid on and off.

The **Meridian Labels** option is used to toggle the meridian grid labels on and off.

The **Parallel Labels** option is used to toggle the parallel grid labels on and off.

The **Tracks** option activates the Define Tracks input box for calculating and displaying Great Circle and Rhumb Line tracks on the map axes.

The **Small Circles** option activates the Define Small Circles input box for calculating and displaying small circles on the map axes.

The **Surface Distances** option activates the Surface Distance dialog box for distance, azimuth, and reckoning calculations.

Tools Menu

The **Hide** option is used to hide the mouse tool buttons.

The **Off** option is used to turn off the current mouse tool.

The **Set Limits** option is used to define the zoom out limits to the current settings on the axes.

The **Full View** option is used to zoom out to the current axes limit settings.

The **Origin** option is used to toggle Origin (`originui`) mode on and off. Origin mode is used to interactively modify the map origin.

The **2D View** option is used to set the default two-dimensional view (azimuth=0, elevation=90).

The **Objects** option activates the Object Sets dialog box, which allows for property manipulation of objects displayed on the map axes.

The **Edit** option activates the MATLAB Property Editor to manipulate properties of a plotted object. Choose the **Current Object** option to edit the currently selected object or choose the **Select Object** option to open the Select Object dialog box and choose the object you want to edit.

The **Show** option is used to set the `Visible` property of mapped objects to `'on'`. The **All** option shows all currently mapped objects. The **Object** option activates the Select Object dialog box.

The **Hide** option is used to set the `Visible` property of mapped objects to `'off'`. Choose from the **All** or **Map** options, or choose the **Object** option to activate the Select Object dialog box.

The **Delete** option is used to clear the selected objects. The **All** option clears the current map, frame, and grid lines. The map definition is left in the axes definition. The **Map** option clears the current map, deleting objects plotted on the map but leaving the frame and grid lines displayed. The **Object** option activates the Select Object dialog box.

The **Axes** option is used to manipulate the MATLAB Cartesian axes. The **Show** option shows this axes, the **Hide** option hides this axes, and the **Color** option allows for custom color selection for this axes.

Colormaps Menu

The **Colormaps** menu allows for manipulation of the colormap for the current figure. See the `clrmenu` reference page for details on the **Colormaps** menu options.

See Also

`axesm`

Introduced before R2006a

maptrim

Interactively trim and convert map data from vector to raster format

Syntax

```
maptrim(lat,lon,LineSpec)
maptrim(datagrid,refvec)
maptrim(datagrid,refvec,PropertyName,PropertyValue)
```

Description

`maptrim(lat,lon)` displays the supplied map data in a new figure window and allows a region of the map to be selected and saved in the workspace. `lat` and `lon` must be vector map data. The output can be line, patch, or regular surface (matrix) data. If patch map output is selected, the inputs `lat` and `lon` must originally be patch map data.

`maptrim(lat,lon,LineSpec)` displays the supplied map using the specified line style, marker, and color on page 1-1655.

`maptrim(datagrid,refvec)` displays data grid data in a new figure window and allows a subset of this map to be selected and saved. The output is regular surface data.

`maptrim(datagrid,refvec,PropertyName,PropertyValue)` displays the data grid using the surface properties provided. The object `Tag`, `EdgeColor`, and `UserData` properties cannot be set.

Controls
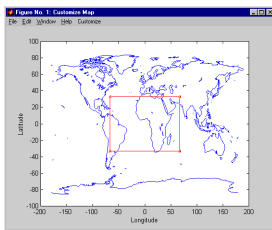
The `maptrim` tool displays the supplied map data in a new figure window and activates a **Customize** menu for that figure. The **Customize** menu has three menu options: **Zoom On/Off**, **Limits**, and **Save As**.

The **Zoom On/Off** menu toggles interactive zoom on and off.

The **Limits** menu option activates the Enter Map Limits dialog box, which is used to enter the latitude and longitude limits of the desired map subset. These entries are two-element vectors, enclosed in brackets. Pressing the **OK** button zooms in to the new limits. Pressing the **Cancel** button disregards the new limits and returns to the map display.
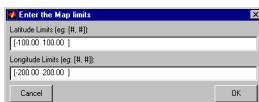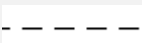
The **Save As** menu option is used to specify the variable names in which to save the map data subset. To save line and patch data, enter the new latitude and longitude variable names, along with the map resolution. For surface data, enter the new map and referencing vector variable names, along with the scale of the map. Latitude and longitude limits are optional.
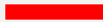
More About

Line Style, Marker, and Color

Specify the line style, marker, and color as a character vector or string containing symbols. The symbols can appear in any order. You do not need to specify all three characteristics (line style, marker, and color). For example, specify a red dashed line using `'--r'`.

Line Style	Description	Resulting Line
`'-'`	Solid line	──────────
`'--'`	Dashed line	─ ─ ─ ─ ─
`':'`	Dotted line	....
`'-.'`	Dash-dotted line	─·─·─·─··

Marker	Description	Resulting Marker
`'o'`	Circle	○
`'+'`	Plus sign	+
`'*'`	Asterisk	✳
`'.'`	Point	•
`'x'`	Cross	✕
`'_'`	Horizontal line	—
`'|'`	Vertical line		
`'s'`	Square	□
`'d'`	Diamond	◇
`'^'`	Upward-pointing triangle	△
`'v'`	Downward-pointing triangle	▽
`'>'`	Right-pointing triangle	▷
`'<'`	Left-pointing triangle	◁

Marker	Description	Resulting Marker
'p'	Pentagram	☆
'h'	Hexagram	✡

Color Name	Short Name	RGB Triplet	Appearance
'red'	'r'	[1 0 0]	▬
'green'	'g'	[0 1 0]	▬
'blue'	'b'	[0 0 1]	▬
'cyan'	'c'	[0 1 1]	▬
'magenta'	'm'	[1 0 1]	▬
'yellow'	'y'	[1 1 0]	▬
'black'	'k'	[0 0 0]	▬
'white'	'w'	[1 1 1]	▭

See Also

maptriml | maptrimp | geocrop

Introduced before R2006a

mlayers

GUI to control plotting of display structure elements

Note `mlayers` will be removed in a future release.

Activation

| Command Line	Maptool

`mlayers('filename')`
 	
`mlayers('filename',h)`
 	
`mlayers(cellarray)`
 	
`mlayers(cellarray,h)`
	Session > **Layers**

Description

`mlayers('filename')` associates all display structures, which in this context are also called map layers, in the MAT-file `filename` with the current map axes. The display structure variables are accessible only through the `mlayers` tool, and not through the base workspace. `filename` must be a character vector.

`mlayers('filename',h)` assigns the layers found in `filename` to the map axes indicated by the handle `h`.

`mlayers(cellarray)` associates the layers specified by `cellarray` with the current map axes. `cellarray` must be of size n by 2. Each row of `cellarray` represents a map layer. The first column of `cellarray` contains the layer structure, and the second column contains the name of the layer structure. Such a cell array can be generated from data in the current workspace with the function `rootlayr`. In this case, the calling sequence would be `rootlayr; mlayers(ans)`.

`mlayers(cellarray,h)` assigns the layers specified by `cellarray` to the map axes specified by the handle `h`.

Controls
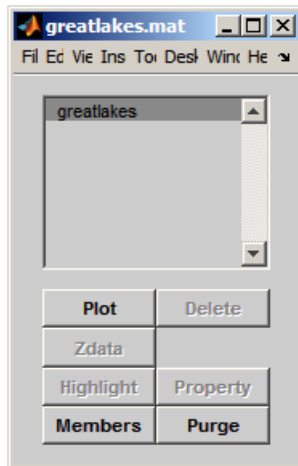
The scrollable list box displays all of the map layers currently associated with the map axes. An asterisk next to the layer name indicates that the layer is currently visible. An h next to the layer name indicates a layer that is plotted, but currently hidden.

The **Plot** button plots the selected map layer. Once the selected layer is plotted, the button toggles between **Hide** and **Show**, to turn the Visible property of the plotted objects to 'off' and 'on', respectively.

The **Zdata** button activates the Specify Zdata dialog box, which is used to enter the workspace variable containing the ZData for the selected map layer. Pressing the **List** button produces a list of all current workspace variables, from which the ZData variable can be selected. This entry can also be a scalar.
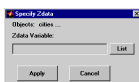

The **Highlight** button is used to toggle the selected map layer between highlighted and normal display.

The **Members** button brings up a list of members of the selected map layer. Members of a layer are defined by their Tag property.

The **Delete** button deletes the selected map layer from the map.

The **Property** button activates the Define Layer Properties dialog box, which is used to specify or change properties of all objects in the selected map layer. Character vector entries must be enclosed in single quotes.
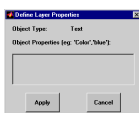
The **Purge** button deletes the selected map layer from the mlayers tool. Selecting **Yes** from the Confirm Purge dialog box deletes the map layer from both the mlayers tool and the map display.

Selecting **Data Only** from the Confirm Purge dialog box deletes the map layer from the `mlayers` tool, while retaining the plotted object on the map display.

See Also
`mobjects` | `rootlayr`

Introduced before R2006a

mobjects

Manipulate object sets displayed on map axes

Note `mobjects` will be removed in a future release.

Activation

Command Line	Maptool

`mobjects`
 	
`mobjects(h)`
	Tools > Objects

Description

An object set is defined as all objects with identical tags. If no tags are supplied, object sets are defined by object type.

`mobjects` allows manipulation of the object sets on the current map axes.

`mobjects(h)` allows manipulation of the objects set on the map axes specified by the handle `h`.

Controls

The scrollable list box displays all of the object sets associated with the map axes. An asterisk next to an object set name indicates that the object set is currently visible. An `h` next to an object set name indicates an object set that is plotted, but currently hidden. The order shown in the list indicates the stacking order of objects within the same plane.

The **Hide/Show** button toggles the `Visible` property of the selected object set to `'off'` and `'on'`, respectively, depending on the current `Visible` status.

The **Zdata** button activates the Specify Zdata dialog box, which is used to enter the workspace variable containing the ZData. The ZData property is used to specify the plane in which the selected object set is drawn. Pressing the **List** button produces a list of all current workspace variables, from which the ZData variable can be selected. Alternatively, a scalar value can be entered instead of a variable.


The **Highlight** button highlights all objects belonging to the selected object set.

The **Tag** button brings up an Edit Tag dialog box, which allows the tag of all members of the selected object set to be modified.

The **Delete** button clears all objects belonging to the selected object set from the map. The cleared object set remains associated with the map axes.

The **Property** button activates the Define Object Properties dialog box, which is used to specify additional properties of all objects in the selected object set. Character vector entries must be enclosed in single quotes.
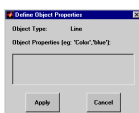
The **Update** button updates the list box display with current objects sets.

The **Stacking Order** buttons are used to modify the drawing order of the selected object set in relation to other plotted object sets in the same plane. Objects drawn first appear at the bottom of the stack, and objects drawn last appear at the top of the stack. The **Top** button places the selected object set above all other object sets in its plane. The **Up** and **Dwn** buttons move the selected object set up and down one place in the stacking order, respectively. The **Btm** button places the selected object set below all other object sets in its plane. Note that the ZData property overrides stacking order, i.e., if an object is at the top of the stacking order for its plane, it can still be covered by an object drawn in a higher plane.

See Also
mlayers

Introduced before R2006a

originui

Interactively modify map origin

Activation

Command Line	Maptool

`originui`
 	
`originui on`
 	
`originui off`
	Tools > **Origin (menu)** > **Origin**(button)

Description

`originui` provides a tool to modify the origin of a displayed map projection. A marker (dot) is displayed where the origin is currently located. This dot can be moved and the map reprojected with the identified point as the new origin.

`originui` automatically toggles the current axes into a mode where only actions recognized by `originui` are executed. Upon exit of this mode, all prior `ButtonDown` functions are restored to the current axes and its children.

`originui on` activates origin tool. `originui off` e-activates the tool. `originui` will toggle between these two states.

Controls

Keystrokes

`originui` recognizes the following keystrokes. **Enter** (or **Return**) will reproject the map with the identified origin and remain in the `originui` mode. **Delete** and **Escape** will exit the origin mode (same as `originui off`). **N**,**S**,**E**,**W** keys move the marker North, South, East or West by 10.0 degrees for each keystroke. **n**,**s**,**e**,**w** keys move the marker in the respective directions by 1 degree per keystroke.

Mouse Actions

`originui` recognizes the following mouse actions when the cursor is on the origin marker.

- Single-click and hold moves the origin marker. Double-click the marker reprojects the map with the specified map origin and remains in the origin mode (same as `originui` **Return**).
- Extended-click moves the marker along the Cartesian X or Y direction only (depending on the direction of greatest movement).
- Alternate-click exits the origin tool (same as `originui off`).

Macintosh Key Mapping

- Extend-click: **Shift**+click mouse button

- Alternate-click: **Option**+click mouse button

Microsoft Windows Key Mapping

- Extend-click: **Shift**+click left button or both buttons
- Alternate-click: **Ctrl**+click left button or right button

X-Windows Key Mapping

- Extend-click: **Shift**+click left button or middle button
- Alternate-click: **Ctrl**+click left button or right button

See Also
`axesm` | `setm`

Introduced before R2006a

panzoom

Zoom settings on 2-D map

Note panzoom will be removed in a future release. Use zoom instead.

Syntax

```
panzoom
panzoom on
panzoom off
panzoom setlimits
panzoom out
panzoom fullview
```

Description

panzoom toggles the pan and zoom tool on and off.

panzoom on is equivalent to zoom on.

panzoom off is equivalent to zoom off.

panzoom setlimits is equivalent to zoom reset.

panzoom out is equivalent to zoom out.

panzoom fullview sets the axes limit modes to 'auto' and resets zoom to the resulting limits.

See Also

zoom

Introduced before R2006a

parallelui

Interactively modify map parallels

Activation

Command Line	Maptool
parallelui parallelui on parallelui off	**Tools > Parallels (menu)**

Description

`parallelui` toggles the parallel tool on and off.

`parallelui on` activates the parallel tool

`parallelui off` deactivates the parallel tool

The `parallelui` GUI provides a tool to modify the standard parallels of a displayed map projection. One or two red lines are displayed where the standard parallels are currently located. The parallel lines can be dragged to new locations, and the map reprojected with the locations of the parallel lines as the new standard parallels.

Controls

Mouse Interaction

A single-click-and-drag moves the parallel lines. A double-click on one of the standard parallels reprojects the map using the new parallel locations.

See Also

axesm | setm

Introduced before R2006a

property editors

GUIs to edit properties of mapped objects

Activation

|	
map display:	Alternate-click mapped object (for Click-and-Drag Property Editor)
	In plot edit mode, double-click mapped object (to obtain MATLAB Property Editor; click the **More Properties...** button to open the Property Inspector)
maptool:	**Tools > Edit Plot** menu item (for MATLAB Property Editor)

Description

Alternate (e.g., **Ctrl**+clicking a mapped object activates a property editor, which allows modification of some basic properties of the object through simple mouse clicks and drags. The objects supported by this editor are map axes, lines, text, patches, and surfaces, and the properties supported for each object type are shown below.

In plot edit mode, double-clicking a mapped object activates the MATLAB Property Editor for that object. From the Property Editor you can launch the Property Inspector, a GUI that lists the properties and values of the selected object and allows you to modify them.

Controls

Click-and-Drag Property Editor

The Click-and-Drag editor lists object properties and values. The object tag appears at the top of the editor. Property names and values that appear in blue are toggles. For example, clicking **Frame** in the axes editor toggles the value of the `Frame` property between `'on'` and `'off'`.
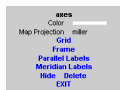
Click-and-Drag Editor for a map axes

Property values that appear on the right side of the editor box are modified by clicking and dragging. For example, to change the `MarkerColor` property of a line object, click and hold the dot next to **MarkerColor**, and drag the cursor until the dot appears in the desired color.

Click-and-Drag Editor for a line object

The **Drag** control in the text editor is used to reposition the text character vector. In drag mode, use the mouse to move the text to a new location, and click to reposition the text. The **Edit** control in the text editor activates a **Text Edit** window, which is used to modify text.

Click-and-Drag Editor for a text object

The **Marker** property name in the patch editor is used to toggle the marker on and off. The property value to the right of **Marker** can be modified by clicking and dragging until the desired marker symbol appears.


Click-and-Drag Editor for a patch object

The **Graticule** control on the surface editor activates a Graticule Mesh dialog box, which is used to alter the size of the graticule.

To move the property editor around the figure window, hold down the **Shift** key while dragging the editor box. Alternate-clicking the background of the property editor closes the **Click-and-Drag** editing session.

Guide Property Editor

The MATLAB **Property Inspector** allows you to view and modify property values for most properties of the selected object. Use it to expand and collapse the hierarchy of objects, showing an object's parents and children. A plus sign (+) before a property indicates that it can be expanded to show its components, for example the axes `AmbientLightColor` applied to the surface object displayed below. A minus sign (-) before an object indicates an object can be collapsed to hide its components. To activate the Object Browser, check the **Show Object Browser** check box. The **Property List** shows all the property names of the selected object and their current values. To activate the **Property List**, check the **Show Property List** check box. To change a property value, use the edit boxes above the Property List. Pressing the **Close** button closes the Guide Property Editor and applies the property modifications to the object.
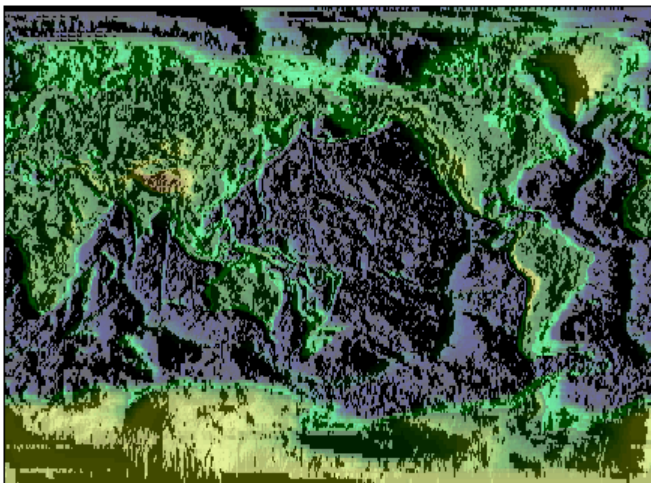
A lit surface object in a map axes
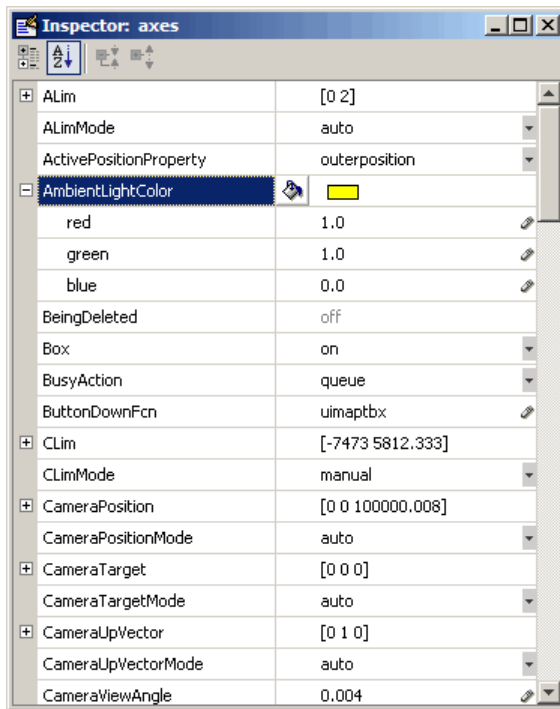
Property Inspector view of axes object

See Also
`propedit` | **Property Inspector** | `uimaptbx`

Introduced before R2006a

qrydata

GUI to interactively perform data queries

Note qrydata will be removed in a future release.

Activation

Command Line
qrydata(cellarray)
qrydata(*titlestr*,cellarray)
qrydata(h,cellarray)
qrydata(h,*titlestr*,cellarray)
qrydata(...,cellarray1,cellarray2,...)

Description

A data query is used to obtain the data corresponding to a particular (x,y) or (lat,lon) point on a standard or map axes.

qrydata(cellarray) activates a data query dialog box for interactive queries of the data set specified by cellarray (described below). qrydata can be used on a standard axes or a map axes. (x,y) or (lat,lon) coordinates are entered in the dialog box, and the data corresponding to these coordinates is then displayed.

qrydata(*titlestr*,cellarray) uses *titlestr* as the title of the query dialog box.

qrydata(h,cellarray) and qrydata(h,*titlestr*,cellarray) associate the data queries with the axes specified by the handle h, which in turn allows the input coordinates to be specified by clicking the axes.

The input cellarray is used to define the data set and the query. The first cell must contain a character vector that is the label of the data display line. The second cell must contain a character vector that specifies the type of query operation. The operation can be either a predefined operation or a valid user-defined function name. The predefined query operations are 'matrix', 'vector', 'mapmatrix', and 'mapvector'.

The 'matrix' query uses the MATLAB interp2 function to find the value of the matrix Z at the input (x,y) point. The format of the cellarray input for this query is: {'label','matrix',X,Y,Z,*method*}. X and Y are matrices specifying the points at which the data Z is given. The rows and columns of X and Y must be monotonic. *method* is an optional argument that specifies the interpolation method. Possible *method* values are 'nearest', 'linear', or 'cubic'. The default is 'nearest'.

The 'vector' query uses the interp2 function to find the value of the matrix Z at the input (x,y) point, then uses that value as an index to a data vector. The value of the data vector at that index is

returned by the query. The format of `cellarray` for this type of query is: `{'label','vector',X,Y,Z, vector}`. X and Y are matrices specifying the points at which the data Z is given. The rows and columns of X and Y must be monotonic. `vector` is the data vector.

The `'mapmatrix'` query interpolates to find the value of the map at the input (`lat,lon`) point. The format of `cellarray` for this query is: `{'label','mapmatrix',datagrid,refvec,method}`. `datagrid` and `refvec` are the data grid and the corresponding referencing vector. `method` is an optional argument that specifies the interpolation method. Possible `method` values are `'nearest'`, `'linear'`, or `'cubic'`. The default is `'nearest'`.

The `'mapvector'` query interpolates to find the value of the map at the input (`lat,lon`) point, then uses that value as an index to a data vector. The value of the vector at that index is returned by the query. The format of `cellarray` for this type of query is `{'label','mapvector',datagrid,refvec, vector}`. `datagrid` and `refvec` are the data grid and the corresponding referencing vector. `vector` is the data vector.

User-defined query operations allow for functional operations using the input (`x,y`) or (`lat,lon`) coordinates. The format of `cellarray` for this type of query is `{'label',function,other arguments...}` where the other arguments are the remaining elements of `cellarray` as in the four predefined operations above. `function` is a user-created function and must refer to a MATLAB function with the signature `z = fcn(x,y,other_arguments...)`.

`qrydata(...,cellarray1,cellarray2,...)` is used to input multiple cell arrays. This allows more than one data query to be performed on a given point.

Controls
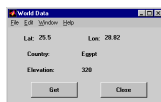
Sample data query dialog box

If an axes handle h is not provided, or if the axes specified by h is not a map axes, the currently selected point is labeled as **Xloc** and **Yloc** at the top of the query dialog box. If h is a map axes, the current point is labeled as **Lat** and **Lon**. Displayed below the current point are the results from the queries, each labeled as specified by the `'label'` input arguments.

The **Get** button appears if an axes handle h is provided. Pressing this button activates a mouse cursor, which is used to select the desired point by clicking the axes. Once a point is selected, the queries are performed and the results are displayed.

The **Process** button appears if the handle h is not provided. In this case, the (`x,y`) coordinates of the desired point are entered into the edit boxes. Pressing the **Process** button performs the data queries and displays the results.

Pressing the **Close** button closes the query dialog box.

Examples

This example illustrates use of a user-defined query to display city names for map points specified by a mouse click. The query is evaluated by a user-supplied file called `qrytest.m`, described below:

```
axesm miller
land = shaperead('landareas', 'UseGeoCoords', true);
geoshow(land, 'FaceColor', [0.5 0.7 0.5])
lakes = shaperead('worldlakes', 'UseGeoCoords', true);
geoshow(lakes, 'FaceColor', 'blue')
rivers = shaperead('worldrivers', 'UseGeoCoords', true);
geoshow(rivers, 'Color', 'blue')
cities = shaperead('worldcities', 'UseGeoCoords', true);
geoshow(cities, 'Marker', '.', 'Color', 'red')
tightmap
lat = [cities.Lat]';
lon = [cities.Lon]';
mat = char(cities.Name);
qrydata(gca,'City Data',{'City','qrytest',lat,lon,mat})
```

Create the file `qrytest.m` on your path, and in it put the following code:

```
function cityname = qrytest(lt, lg, lat, lon, mat)
% function QRYTEST returns city name for mouse click
% QRYTEST will find the closest city (min radius) from
% the mouse click, within an angle of 5 degrees.
%
latdiff = lt-lat;
londiff = lg-lon;
rad = sqrt(latdiff.^2+londiff.^2);
[minrad,index] = min(rad);
if minrad > 5
  index = [];
end
switch length(index)
  case 0, cityname = 'No city located near click';
  case 1, cityname = mat(index,:);
end
```
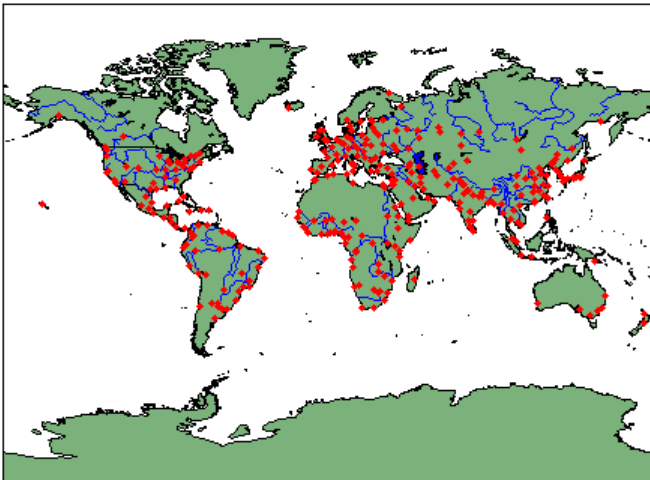

Clicking the mouse over a city marker displays the name of the selected city. Clicking the mouse in an area away from any city markers displays `'No city located near click'`.

See Also
`interp2`

Introduced before R2006a

scirclui

GUI to display small circles on map axes

Compatibility

Activation

Command Line	Maptool
scirclui scirclui(h)	**Display mall Circles**

Description

scirclui activates the Define Small Circles dialog box for adding small circles to the current map axes.

scirclui(h) activates the Define Small Circles dialog box for adding small circles to the map axes specified by the axes handle h.

Controls
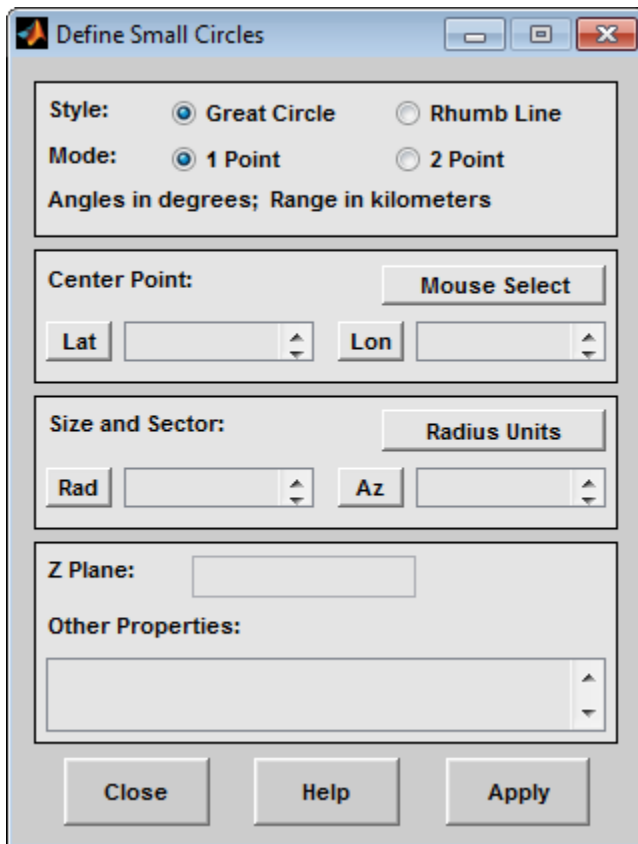
Define Small Circles dialog box for one-point mode

The **Style** selection buttons are used to specify whether the circle radius is a constant great circle distance or a constant rhumb line distance.

The **Mode** selection buttons are used to specify whether one point or two points are to be used in defining the small circle. If one-point mode is selected, a center point, radius, and azimuth are the required inputs. If two-point mode is selected, a center point, and perimeter point on the circle are the required inputs.

The **Center Point** controls are used in both one-point and two-point mode. The **Lat** and **Lon** edit boxes are used to enter the latitude and longitude of the center point of the small circle to be displayed. These values must be in degrees. To display more than one small circle, a vector of values can be entered, enclosed in brackets in each edit box. Pushing the **Lat** or **Lon** button brings up an expanded edit box for easier entry of long vectors. The **Mouse Select** button is used to select a center point by clicking the displayed map. The coordinates of the selected point then appear in the **Lat** and **Lon** edit boxes and can be modified. The coordinates appear in degrees, regardless of the angle units defined for the current map projection.

The **Circle Point** controls are used only in two-point mode. The **Lat** and **Lon** edit boxes are used to enter the latitude and longitude of a point on the perimeter of the small circle to be displayed. These values must be in degrees. To display more than one small circle, a vector of values can be entered, enclosed in brackets in each edit box. Pushing the **Lat** or **Lon** button brings up an expanded edit box

for easier entry of long vectors. The **Mouse Select** button is used to select a perimeter point by clicking the displayed map. The coordinates of the selected point then appear in the **Lat** and **Lon** edit boxes and can be modified. The coordinates appear in degrees, regardless of the angle units defined for the current map projection.

The **Size and Sector** controls are used only in one-point mode. The **Radius Units** button brings up a Define Range Units dialog box, which allows for modification of the small circle radius units and the normalizing geoid. The **Rad** edit box is used to enter the radius of the small circle in the proper units. The **Arc** edit box is used to specify the sector azimuth, measured in degrees, clockwise from due north. If the entry is omitted, a complete small circle is drawn. When entering radius and arc data for more than one small circle, vectors of values, enclosed in brackets, are entered in each edit box. Pushing the **Rad** or **Arc** button brings up an expanded edit box for that entry, which is useful for entering long vectors.

The **Z Plane** edit box is used to enter a scalar value that specifies the plane in which to display the small circles.

The **Other Properties** edit box is used to specify additional properties of the small circles to be projected, such as `'Color','b'`. Character vector entries must be enclosed in quotes.

Pressing the **Apply** button accepts the input data and displays the small circles on the current map axes.

Pressing the **Cancel** button disregards any input data and closes the Define Small Circles dialog box.
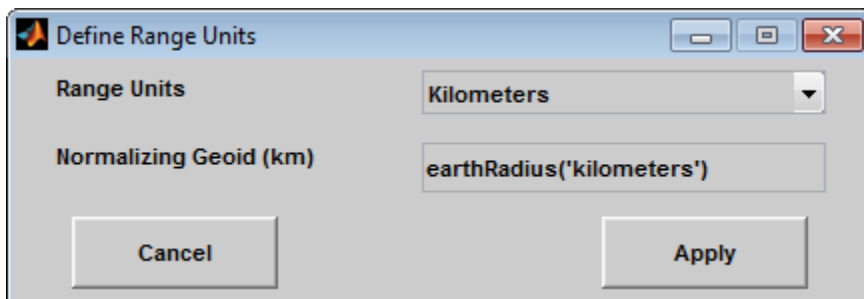
This dialog box, available only in one-point mode, allows for modification of the small circle radius units and the normalizing geoid.

The **Radius Units** pull-down menu is used to select the units of the small circle radius. The unit selected is displayed near the top of the Define Small Circles dialog box, and all latitude and longitude entries must be entered in these units. Users must also be sure to specify the normalizing geoid in the same units. If radians are selected, it is assumed the radius entry is a multiple of the radius used to display the current map, as defined by the map geoid property.

The **Normalizing Geoid** edit box is used modify the radius used to normalize the small circle radius to a radian value, which is necessary for proper calculations and map display. This entry must be in the same units as the small circle radius. If the small circle radius units are in radians, then the normalizing geoid must be the same as the geoid used for the current map axes.

Pressing the **Cancel** button disregards any modifications and closes the Define Radius Units dialog box.

Pressing the **Apply** button accepts any modifications and returns to the Define Small Circles dialog box.

See Also

scircle1 | scircle2

Introduced before R2006a

seedm

GUI to fill data grids with seeded values

Note `seedm` will be removed in a future release.

Activation

Command Line
`seedm(datagrid,refvec)`

Description

Encoding is the process of filling in specific values in regions of a data grid up to specified boundaries, which are indicated by entries of 1 in the variable `map`. Encoding entire regions at one time allows indexed maps to be created quickly.

`seedm(datagrid,refvec)` displays the surface map in a new figure window and allows for seeds to be specified and the encoded map generated. The encoded map can then be saved to the workspace. `map` is the data grid and must consist of positive integer index values. `refvec` is the referencing vector of the surface.

Controls
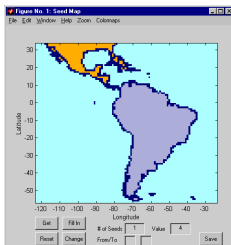
The **Zoom On/Off** menu toggles the zoom box on and off. The box can be moved by clicking the new location or by dragging the box to the new location. The box size can be increased or decreased by dragging a corner of the box. Pressing the **Return** key or double-clicking in the center of the box zooms in to the box limits.

The **Colormaps** menu provides a variety of colormap options that can be applied to the map. See `clrmenu` in this guide for a description of the **Colormaps** menu options.

The **Get** button allows mouse selection of points on the map to which seeds are assigned. The number of points to be selected is entered in the **# of Seeds** edit box. The value of the seed is entered in the **Value** edit box. This seed value is assigned to each point selected with the mouse. The **Get** button is pressed to begin mouse selection. After all the points have been selected, the **Fill In** button is pressed to perform the encoding operation. The region containing the seed point is filled in with the seed value. The **Reset** button is used to disregard all points selected with the mouse before the **Fill In** button is pressed.

Alternatively, specific map values can be globally replaced by using the **From/To** edit boxes. The value to be replaced is entered in the first edit box, and the new value is entered in the second edit box. Pressing the **Change** button replaces all instances of the **From** value to the **To** value in the map.

Note Values of 1 represent boundaries and should not be changed.

The **Save** button is used to save the encoded map to the workspace. A dialog box appears in which the map variable name is entered.

See Also
colorm | encodem | getseeds | maptrim

Introduced before R2006a

showm-ui

Show specified mapped objects

Activation

Command Line	Maptool
showm	**Tools > Show > Object**

Description

showm brings up a Select Object dialog box for selecting mapped objects to show (Visible property set to 'on').

Controls

The scroll box is used to select the desired objects from the list of mapped objects. Pushing the **Select all** button highlights all objects in the scroll box for selection. Pushing the **OK** button changes the Visible property of the selected objects to 'on'. Pushing the **Cancel** button aborts the operation without changing any properties of the selected objects.

See Also
showm

Introduced in R2007a

surfdist

Interactive distance, azimuth, and reckoning calculations

Activation

| Command Line	Maptool
surfdist surfdist(h) surfdist([])	**Display > Surface > Distances**

Description

surfdist activates the Surface Distance dialog box for the current axes only if the axes has a proper map definition. Otherwise, the Surface Distance dialog box is activated, but is not associated with any axes.

surfdist(h) activates the Surface Distance dialog box for the axes specified by the handle h. The axes must be a map axes.

surfdist([]) activates the Surface Distance dialog box and does not associate it with any axes, regardless of whether the current axes has a valid map definition.

Controls
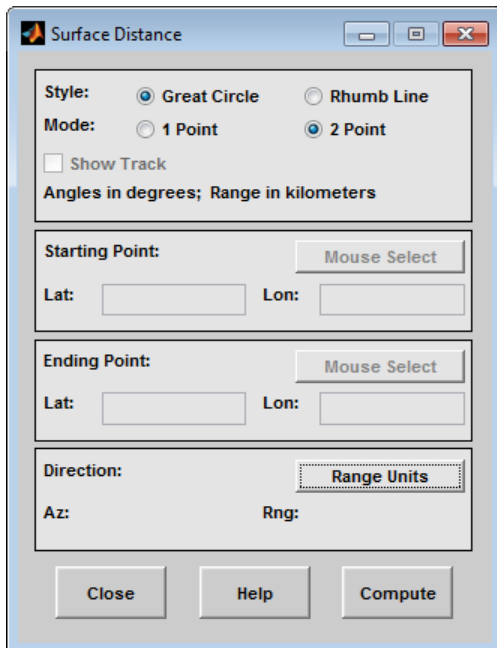
The **Style** selection buttons are used to specify whether a great circle or rhumb line is used to calculate the surface distance. When all other entries are provided, selecting a style updates the surface distance calculation.

The **Mode** selection buttons are used to specify whether one point or two points are to be used in defining the track distance. If one-point mode is selected, a starting point, azimuth, and range are the required inputs, and the ending point is computed. If two-point mode is selected, starting and ending points of the track are required, and the azimuth and distance along this track are then computed.

The **Show Track** check box is used to indicate whether the track is shown on the associated map display. The track is deleted when the Surface Distance dialog box is closed, or when the **Show Track** check box is unchecked and the surface distance calculations are recomputed.

The **Starting Point** controls are used for both one-point and two-point mode. The **Lat** and **Lon** edit boxes are used to enter the latitude and longitude of the starting point of the track. These values must be in degrees. Only one starting point can be entered. The **Mouse Select** button is used to select a starting point by clicking the displayed map. The coordinates of the selected point then appear in the **Lat** and **Lon** edit boxes and can be modified. The coordinates appear in degrees, regardless of the angle units defined for the current map projection.

The **Ending Point** controls are enabled only for two-point mode. The **Lat** and **Lon** edit boxes are used to enter the latitude and longitude of the ending point of the track. These values must be in degrees. Only one ending point can be entered. The **Mouse Select** button is used to select an ending point by clicking the displayed map. The coordinates of the selected point then appear in the **Lat** and **Lon** edit boxes and can be modified. The coordinates appear in degrees, regardless of the angle units defined for the current map projection. During one-point mode, the Ending Point controls are disabled, but the ending point that results from the surface distance calculation is displayed.

The **Direction** controls are enabled only for one-point mode. The **Range Units** button brings up a Define Range Units dialog box which allows for modification of the range units and the normalizing geoid. The **Az** edit box is used to enter the azimuth, which sets the initial direction of the track from the starting point. Azimuth is measured in degrees clockwise from due north. The **Rng** edit box is used to specify the reckoning range of the track, in the proper units. The azimuth and reckoning range, along with the starting point, are used to compute the ending point of the track in one-point mode. During two-point mode, the **Direction** controls are disabled, but the azimuth and range values resulting from the surface distance calculation are displayed.

Pressing the **Close** button disregards any input data, deletes any surface distance tracks that have been plotted, and closes the Surface Distance dialog box.

Pressing the **Compute** button accepts the input data and computes the specified distances.

Define Range Units Dialog Box

This dialog box, available only for one-point mode, allows for modification of the range units and the normalizing geoid.
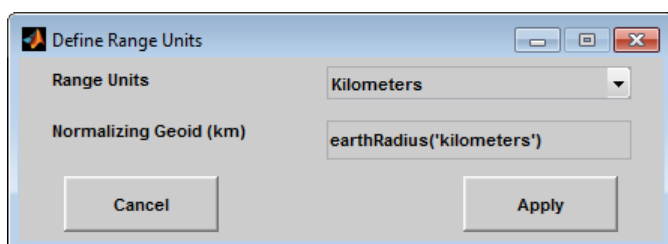
The **Range Units** pull-down menu is used to select the units of the reckoning range. The unit selected is displayed near the top of the Surface Distance dialog box, and all latitude and longitude entries must be entered in these units. Users must also be sure to specify the normalizing geoid in the same units. If radians are selected, it is assumed the range entry is a multiple of the radius of the normalizing geoid. In this case, the normalizing geoid must be the same as the geoid used to display the current map.

The **Normalizing Geoid** edit box is used modify the radius used to normalize range entries to radian values, which is necessary for proper calculations and map display. This entry must be in the same units as the range units. If the range units are in radians, then the normalizing geoid must be the same as the geoid used for the current map axes.

Pressing the **Cancel** button disregards any modifications and closes the Define Range Units dialog box.

Pressing the **Apply** button accepts any modifications and returns to the Surface Distance dialog box.

Introduced before R2006a

tagm-ui

GUI to edit tag property of mapped object

Activation

Command Line
tagm
tagm(h)

Description

tagm brings up a Select Object dialog box for selecting mapped objects and changing their Tag property. Upon selecting the objects, the Edit Tag dialog box is activated, in which the new tag is entered.

tagm(h) activates the Edit Tag dialog box for the objects specified by the handle h.

Controls

Select Object Dialog Box

The scroll box is used to select the desired objects from the list of mapped objects. Pushing the **Select all** button highlights all objects in the scroll box for selection. Pushing the Ok button activates the Edit Tag dialog box. Pushing the **Cancel** button aborts the operation without changing any properties of the selected objects.

Edit Tag Dialog Box

Enter the new tag character vector in the edit box. Pressing the **Apply** button changes the Tag property of all selected objected to the new tag character vector. Pressing the **Cancel** button closes the Edit Tag dialog box without changing the Tag property of the selected objects.

See Also
tagm

Introduced in R2007a

trackui

GUI to display great circles and rhumb lines on map axes

Compatibility

**Note** `trackui` is obsolete. Use `trackg` instead.

Activation

Command Line	Maptool

`trackui`
 	
`trackui(h)`
	Display > Tracks

Description

`trackui` activates the Define Tracks dialog box for adding great circle or rhumb line tracks to the current map axes.

`trackui(h)` activates the Define Tracks dialog box for adding great circle or rhumb line tracks to the map axes specified by the axes handle `h`.

Controls
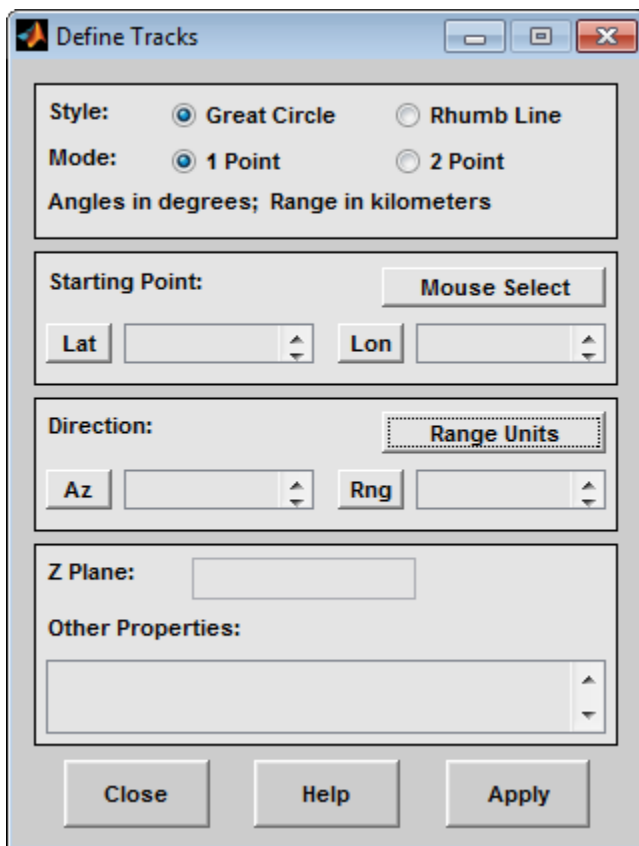
Define Tracks dialog box for two-point mode

The **Style** selection buttons are used to specify whether a great circle or rhumb line track is displayed.

The **Mode** selection buttons are used to specify whether one point or two points are to be used in defining the track. If one-point mode is selected, a starting point, azimuth, and range are the required inputs. If two-point mode is selected, starting and ending points are required.

The **Starting Point** controls are used for both one-point and two-point mode. The **Lat** and **Lon** edit boxes are used to enter the latitude and longitude of the starting point of the track to be displayed. These values must be in degrees. To display more than one track, a vector of values can be entered, enclosed in brackets in each edit box. Pushing the **Lat** or **Lon** button brings up an expanded edit box for easier entry of long vectors. The **Mouse Select** button is used to select a starting point by clicking the displayed map. The coordinates of the selected point then appear in the **Lat** and **Lon** edit boxes and can be modified. The coordinates appear in degrees, regardless of the angle units defined for the current map projection.

The **Ending Point** controls are used only for two-point mode. The **Lat** and **Lon** edit boxes are used to enter the latitude and longitude of the ending point of the track to be displayed. These values must be in degrees. To display more than one track, a vector of values can be entered, enclosed in brackets, in each edit box. Pushing the **Lat** or **Lon** button brings up an expanded edit box for easier entry of long vectors. The **Mouse Select** button is used to select an ending point by clicking the displayed map.

The coordinates of the selected point then appear in the **Lat** and **Lon** edit boxes and can be modified. The coordinates appear in degrees, regardless of the angle units defined for the current map projection.
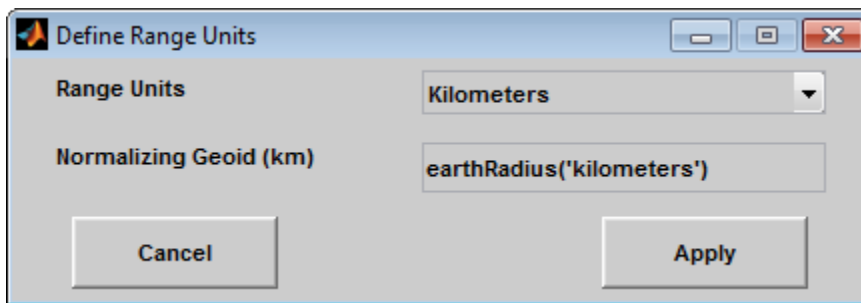
The **Direction** controls are used only for one-point mode. The **Range Units** button brings up a Define Range Units dialog box, which allows for modification of the range units and the normalizing geoid. The **Az** edit box is used to enter the azimuth, which sets the initial direction of the track from the starting point. Azimuth is measured in degrees clockwise from due north. The **Rng** edit box is used to specify the range of the track, in the proper units. If the range entry is omitted, a complete track is drawn. When inputting azimuth and range data for more than one track, vectors of values, enclosed in brackets, are entered in each edit box. Pushing the **Az** or **Rng** button brings up an expanded edit box for that entry, which is useful for entering long vectors.

The **Z Plane** edit box is used to enter a scalar value that specifies the plane in which to display the tracks.

The **Other Properties** edit box is used to specify additional properties of the tracks to be projected, such as `'Color','b'`. Character vector entries must be enclosed in quotes.

Pressing the **Apply** button accepts the input data and displays the tracks on the current map axes.

Pressing the **Cancel** button disregards any input data and closes the Define Tracks dialog box.

Define Range Units Dialog Box

This dialog box, available only for one-point mode, allows for modification of the range units and the normalizing geoid.

The **Range Units** pull-down menu is used to select the units of the track range. The unit selected is displayed near the top of the Define Tracks dialog box, and all latitude and longitude entries must be entered in these units. Users must also be sure to specify the normalizing geoid in the same units. If radians are selected, it is assumed the range entry is a multiple of the radius used to display the current map.

The **Normalizing Geoid** edit box is used to modify the radius used to normalize range entries to radian values, which is necessary for proper calculations and map display. This entry must be in the same units as the range units. If the range units are in radians, then the normalizing geoid must be the same as the geoid used for the current map axes.

Pressing the **Cancel** button disregards any modifications and closes the Define Range Units dialog box.

Pressing the **Apply** button accepts any modifications and returns to the Define Tracks dialog box.

See Also
track1 | track2

Introduced before R2006a

uimaptbx

Handle buttondown callbacks for mapped objects

Activation

set the `ButtonDownFcn` property to `'uimaptbx'`

Description

`uimaptbx` processes mouse events for mapped objects. `uimaptbx` can be assigned to an object by setting the `ButtonDownFcn` to `'uimaptbx'`. This is the default setting for all objects created with Mapping Toolbox functions.

If `uimaptbx` is assigned to an object, the following mouse events are recognized: A single-click and hold on an object displays the object tag. If no tag is assigned, the object type is displayed. A double-click on an object activates the MATLAB Property Editor. An extend-click on an object activates the Projection Control dialog box, which allows the map projection and display properties to be edited. An alternate-click on an object allows basic properties to be edited using simple mouse clicks and drags.

Definitions of extend-click and alternate-click on various platforms are as follows:

`For MS-Windows:`	Extend-click – **Shift**+click left button or both buttons
	Alternate-click – **Ctrl**+click left button or right button
`For X-Windows:`	Extend-click – **Shift**+click left button or middle button
	Alternate-click – **Ctrl**+ click left button or right button

See Also
axesm | axesmui | property editors

Introduced before R2006a

utmzoneui

Choose or identify UTM zone by clicking map

Activation

| Command Line |
| --- |
| utmzoneui |
| utmzoneui(InitZone) |

Description

`zone = utmzoneui` opens an graphical user interface for choosing a UTM zone on a world display map. You select a zone by clicking an area for its appropriate zone, or entering a valid zone to identify the zone on the map.

`zone = utmzoneui(InitZone)` initialize the map displayed to the zone character vector specified in `InitZone`.

To interactively pick a UTM zone, activate the interface, and then click any rectangular zone on the world map to display its UTM zone. The selected zone is highlighted in red and its designation is displayed in the **Zone** edit field. Alternatively, type a valid UTM designation in the **Zone** edit field to select and see the location of a zone. Valid zone designations consist of an integer from 1 to 60 followed by a letter from C to X.

Typing only the numeric portion of a zone designation will highlight a column of cells. Clicking **Accept** returns a that UTM column designation. You cannot return a letter (row designation) in such a manner, however.

Controls
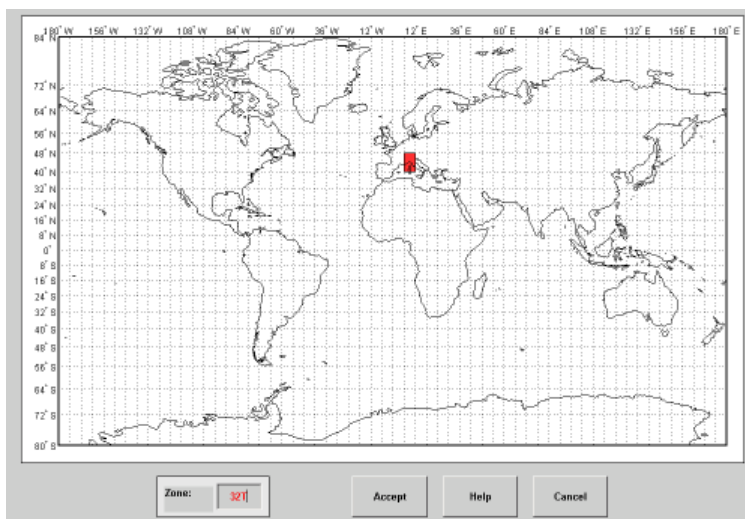
Tips

The syntax of `utmzoneui` is similar to that of `utmzone`. If `utmzone` is called with no arguments, the `utmzoneui` interface is displayed for you to select a zone. Note that `utmzone` can return latitude-longitude coordinates of a specified zone, but that `utmzoneui` only returns zone names.

See Also

utmgeoid | utmzone

Topics
ups
utm

Introduced before R2006a

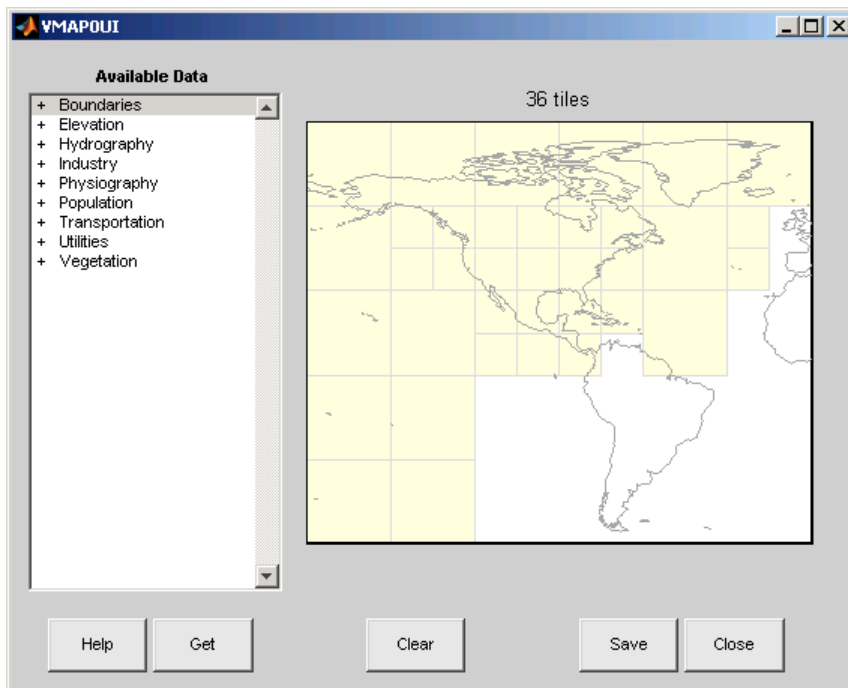# vmap0ui

UI for selecting data from Vector Map Level 0

Description

vmap0ui(dirname) launches a graphical user interface for interactively selecting and importing data from a Vector Map Level 0 (VMAP0) data base. dirname is a character vector that specifies the folder containing the data base. For more on using vmap0ui, click the **Help** button after the interface appears.

vmap0ui(devicename) or vmap0ui devicename uses the logical device (volume) name specified in the character vector devicename to locate CD-ROM drive containing the VMAP0 CD-ROM. Under the Windows operating system it could be 'F:', 'G:', or some other letter. Under Macintosh OS X it should be '/Volumes/VMAP'. Under other UNIX systems it could be '/cdrom/'.

vmap0ui can be used on Windows without any arguments. In this case it attempts to automatically detect a drive containing a VMAP0 CD-ROM. If vmap0ui fails to locate the CD-ROM device, then specify it explicitly.

Controls

The vmap0ui screen lets you read data from the Vector Map Level 0 (VMAP0). The VMAP0 is the most detailed world map database available to the public.

You use the list to select the type of data and the map to select the region of interest. When you click the **Get** button, data is extracted and displayed on the map. Use the **Save** button to save the data in a MAT-file or to the base workspace for later display. The **Close** button closes the window.

The Map

The **Map** controls the geographic extent of the data to be extracted. `vmap0ui` extracts data for areas currently visible on the map. Use the mouse to zoom in or out to the area of interest. Type `help zoom` for more on zooming.

The VMAP0 divides the world into tiles of about 5–by–5 degrees. When extracting, data is returned for all visible tiles, including those parts of the tile that are outside the current view. The map shows the VMAP0 tiles in light yellow with light gray edges. The data density is high, so extracting data for a large number of tiles can take much time and memory. A count of the number of visible tiles is above the map.

The List

The **List** controls the type of data to be extracted. The tree structure of the list reflects the structure of the VMAP0 database. Upon starting `vmap0ui`, the list shows the major categories of VMAP data, called themes. Themes are subdivided into features, which consist of data of common graphic types (patch, line, point, or text) or cultural types (airport, roads, railroads). Double-click a theme to see the associated features. Features can have properties and values, for example, a railroad tracks property, with values single or multiple. Double-click a feature to see the associated properties and values. Double-clicking an open theme or feature closes it. When a theme is selected, `vmap0ui` gets all the associated features. When a feature is selected, `vmap0ui` gets all of that feature's data. When properties and values are selected, `vmap0ui` gets the data for any of the properties and values that match (that is, the union operation).

The Get Button

The **Get** button reads the currently selected VMAP0 data and displays it on the map. Use the **Cancel** button on the progress bar to interrupt the process. For a quicker response, press the standard interrupt key combination for your platform.

The Clear Button

The **Clear** button removes any previously read data from the map.

The Save Button

The **Save** button saves the currently displayed VMAP0 data to a MAT-file or the base workspace. If you choose to save to a file, you are prompted for a file name and location. If you choose to save to the base workspace, you are notified of the variable names that will be overwritten.

Data are returned as Mapping Toolbox display structures with variable names based on theme and feature names. You can update vector display structures to geographic data structures. For information about display structure format, see "Version 1 Display Structures" on page 1-286 in the reference page for `displaym`. The `updategeostruct` function performs such conversions.

Use `load` and `displaym` to redisplay the data from a file on a map axes. You can also use the `mlayers` GUI to read and display the data from a file. To display the data in the base workspace, use `displaym`. To display all the display structures, use `rootlayr; displaym(ans)`. To display all of the display structures using the `mlayers` GUI, type `rootlayr; mlayers(ans)`.

The Close Button

The **Close** button closes the `vmap0ui` panel.

Examples

1 Launch `vmap0ui` and automatically detect a CD-ROM on Microsoft Windows:

```
vmap0ui
```

2 Launch `vmap0ui` on Macintosh OS X (need to specify volume name):

```
vmap0ui('Volumes/VMAP')
```

See also

`displaym`, `extractm`, `mlayers`, `vmap0data`

Introduced before R2006a

zdatam-ui

GUI to adjust *z*-plane of mapped objects

Activation

Command Line
zdatam
zdatam(h)
zdatam(*str*)

Description

zdatam brings up a Select Object dialog box for selecting mapped objects and adjusting their ZData property. Upon selecting the objects, the Specify Zdata dialog box is activated, in which the new ZData variable is entered. Note that not all mapped objects have the ZData property (for example text objects).

zdatam(h) activates the Specify Zdata dialog box for the objects specified by the handle h.

zdatam(*str*) activates the Specify Zdata dialog box for the objects identified by *str*, where *str* is any of the character vectors recognized by handlem.

Controls

Select Object Dialog Box

The scroll box is used to select the desired objects from the list of mapped objects. Pushing the **Select all** button highlights all objects in the scroll box for selection. Pushing the **OK** button activates another Specify Zdata dialog box. Pushing the **Cancel** button aborts the operation without changing any properties of the selected objects.

Specify ZData Dialog Box

The **Zdata Variable** edit box is used to specify the name of the ZData variable. Pressing the **List** button produces a list of all current workspace variables, from which the ZData variable can be selected. A scalar value or a valid MATLAB expression can also be entered. Pressing the **Apply** button changes the ZData property of all selected objected to the new values. Pressing the **Cancel** button closes the Specify ZData dialog box without changing the ZData property of the selected objects.

See Also
zdatam

Introduced in R2007a