Mapping Toolbox

For Use with MATLAB®

- Computation
- Visualization
- Programming

User's Guide
Version 2

How to Contact The MathWorks:

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup

`support@mathworks.com`	Technical support
`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information

508-647-7000 Phone

508-647-7001 Fax

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

For contact information about worldwide offices, see the MathWorks Web site.

Mapping Toolbox User's Guide

Printing History:	May 1997	First printing	New for Version 1
	October 1998	Second printing	Revised for Version 1.1
	November 2000	Third printing	Revised for Version 1.2 (Release 12)
	July 2002	Online only	Revised for Version 1.3 (Release 13)
	September 2003	Online only	Revised for Version 1.3.1 (Release 13SP1)
	January 2004	Online only	Revised for Version 2.0 (Release 13SP1+)
	April 2004	Online only	Revised for Version 2.0.1 (Release 13SP1+)
	June 2004	Online only	Revised for Version 2.0.2 (Release 14)

Contents

1 Getting Started

2 Understanding Map Data

3 Understanding Geospatial Geometry

4 Creating and Viewing Maps

5 Making Three-Dimensional Maps

6 Customizing and Printing Maps

7 Manipulating Geospatial Data

8 Using Map Projections and Coordinate Systems

9 Mapping Applications

10 Reference

11 Projections Reference

12 GUI Reference

A Atlas Data

1

Getting Started

Welcome to the Mapping Toolbox for MATLAB®. The Mapping Toolbox is a collection of MATLAB functions, user interfaces, sample data sets, and demos that read, write, display, and manipulate geospatial data. With it you can make maps of your own geospatial data or use data provided with the Mapping Toolbox, such as world coastlines, political boundaries, and topography. The following sections get you started using the Mapping Toolbox, and then describe what information this documentation covers and where to find it.

Note Some cross-references in this document refer to reference material that is included only in the complete, electronic version of this User's Guide, found in the MATLAB Help Browser, and also available in HTML and PDF formats on the MathWorks web site, at `http://www.mathworks.com/access/helpdesk/help/toolbox/map/map.html`.

What Is the Mapping Toolbox?

The Mapping Toolbox provides a comprehensive set of functions and graphical user interfaces for building map displays and performing geospatial data analysis in MATLAB. You can create map displays that combine data from multiple modalities and display them in their correct spatial relationships. The toolbox supports standard analyses, such as line-of-sight calculations on terrain data or geographic computations that account for the curvature of the Earth's surface. Most of the functions in the Mapping Toolbox are written in the open MATLAB language. This means that you can inspect the algorithms, modify the source code, create your own custom functions, and automate frequently performed tasks.

The toolbox supports key mapping and geospatial data analysis, manipulation, and visualization tasks that are useful in applications such as earth and planetary scientific research, oil and gas exploration, environmental monitoring, insurance risk management, aerospace, defense, and security.

Dedication and Acknowledgment

In memory of John P. Snyder, whose meticulous studies and systematic descriptions of map projections inspired and enabled the creation of the Mapping Toolbox.

The Mapping Toolbox was originally developed and maintained through version 1.3 by Systems Planning and Analysis, Inc. (SPA) of Alexandria, Virginia.

The information contained in atlas data files (MAT-files found in `toolbox/map/mapdisp`) was derived from publicly available digital map data sets. The MathWorks, Inc. provides these data files as a convenience to Mapping Toolbox users. The MathWorks makes no claims that the representation of geographic features, international boundaries, sovereign assignations, or feature names in these files are up-to-date, authoritative or free of defects and errors.

Your First Maps

Spatial data is a general term that refers to data describing the location, shape, and spatial relationships of anything, from engineering drawings to maps of galaxies. *Geospatial data* is spatial data that is in some way *georeferenced*, or tied to specific locations on, under, or above the surface of a planet.

Geospatial data can be voluminous, complex, and difficult to work with unless displayed graphically. The Mapping Toolbox handles many of the details of loading and displaying data for you. Nevertheless, the more you understand about your data and the capabilities of the toolbox, the more interesting applications you will be able to pursue, and the more useful their results will be to you and others.

This section helps you exercise graphic user interfaces (GUIs) to explore atlas data provided with the Mapping Toolbox. It explores `worldmap` and other commands, and then explores geodata with the Map Viewer (`mapview`). You can then use the "Documentation Summary" on page 1-24 to identify where to find the capabilities of the Mapping Toolbox you want to learn more about.

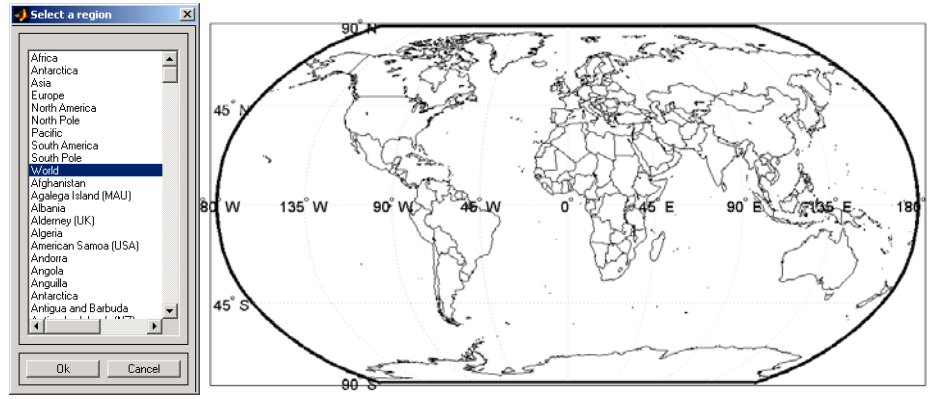
See the World

Getting started making world maps with the Mapping Toolbox is easy.

1 In the MATLAB Command Window, type

```
worldmap
```

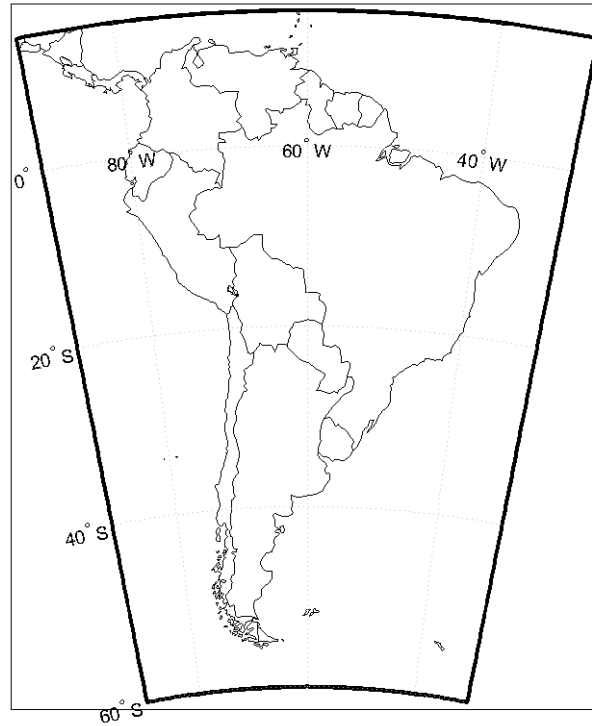
This brings up a scrolling list of countries and continents from which you can select a region of interest, or the entire world.



2 If you choose a country or region, `worldmap` displays it with an appropriate map projection and as much detail as can be clearly rendered. If you prefer, you can include the region or country name in your command:

```
h = worldmap('south america')
```

The map below appears in a new figure window.



When you specify a return argument for `worldmap` and certain other mapping functions, a handle (e.g., `h`) to the figure's axes is returned. The axes object on which map data are displayed it called a *map axes*. In addition to the graphics properties common to any MATLAB axes object, a map axes object contains additional properties covering map projection type, projection parameters, map limits, etc. Various functions in the mapping toolbox (most importantly `getm` and `setm`) allow you to define, access, and modify these properties.

3 To inspect this data, first dereference the handle with the `getm` command (which is similar to the MATLAB `get` command, but returns map-specific data):

```
mstruct = getm(h);
```

4 Now you can inspect the sa struct array by listing it, using the property editor, or by accessing any field directly. For instance, to see the map projection selected for the South America map, type

```
mstruct.mapprojection
ans =
eqdconic
```

5 Finally, place a label at 10°N latitude, 60°W longitude to identify the map:

```
textm(10,-60,'South America')
```

Explore Built-in Atlas Data

In mapping South America, `worldmap` used built-in atlas data, which includes

- Coastlines
- International boundaries
- Lakes
- Cities

The atlas data has two versions, low resolution and high resolution. In the above example, `worldmap` selected low-resolution atlas data because the continent occupies a large area, and at such a small scale the details of high-resolution data could not be discerned. In such cases, `worldmap` also omits details such as country names and cities. You can override `worldmap`'s choices using optional arguments.

A Patch Map of Pakistan

As you specify smaller regions, `worldmap` switches to higher resolution atlas data. Illustrate this by making a base map of Pakistan. This time, instead of drawing country boundaries as lines, use `worldmap` to render countries using patches (filled polygons). Countries are assigned colors at random. Also place another cartographic element, a graphic scale.

1 First, create a new figure window for the map:

```
figure
```

2 Use `worldmap` in patch (polygon) mode:

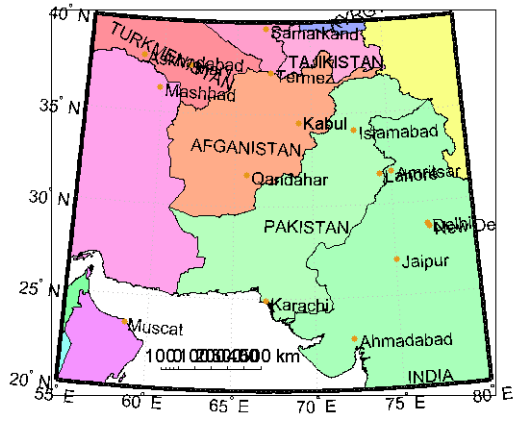
```
worldmap('pakistan','patch')
```

3 Add a graphic scale, using the default style and placement:

```
scaleruler
```

4 Use your mouse to move the scale ruler to a better position by clicking its baseline and dragging. See the reference documentation for `scaleruler` and `setm` for other properties of graphic scales that you can set.

5 Click different countries, place names, and axis labels. A description of each one appears in the lower left corner of the figure while the mouse button is down. To change the alignment or adjust the positions of city names, use the click and drag tools described in the Mapping Toolbox "GUI Reference" documentation. The map appears below.



6 You can also hide the axes border (a good idea when printing a map):

```
hidem(gca)
```

An Alternative View

The map of Pakistan was drawn using the high-resolution data. Zoom in on Karachi to see the details of the shoreline and islands. Then recreate the map using low-resolution data to see the difference:

```
figure
```

```
worldmap('lo','pakistan','patch')
```

Look at the reference documentation for `worldmap` and experiment with its options. To learn more about display properties for map axes and how to control them, see “Accessing and Manipulating Map Axes Properties” on page 4-6.

Tour Boston with the Map Viewer

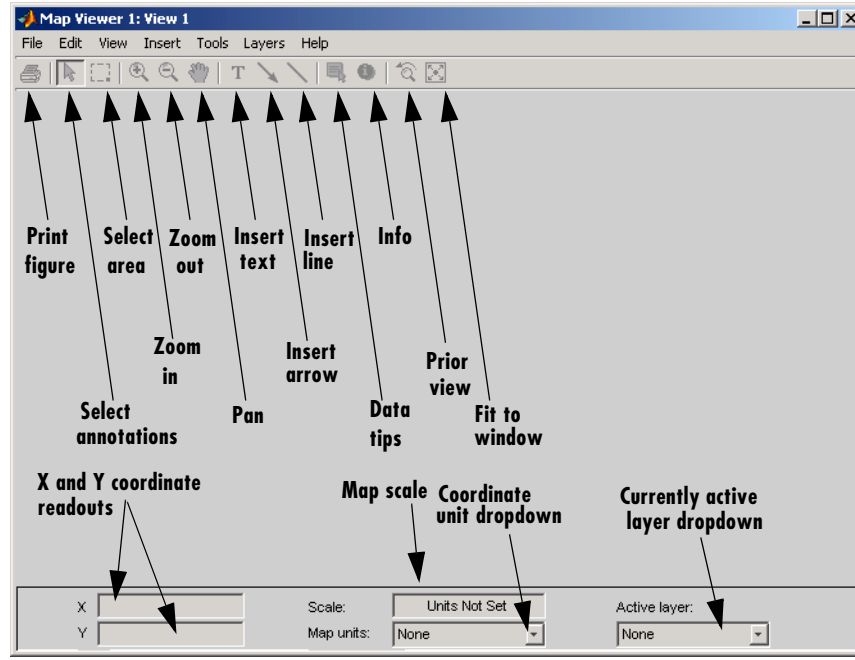
The Map Viewer is an interactive tool for browsing map data. With it you can assemble layers of vector and raster geodata and render them in 2-D. You can import, reorder, symbolize, hide, and delete data layers, identify coordinate locations, list data attributes, and display selected ones as “datatips” (signposts that identify attribute values, such as place names or route numbers). The following exercise shows how the Map Viewer works and what it can do.

A Map Viewer Session

1 You start a Map Viewer session by typing

```
mapview
```

at the MATLAB prompt. The Map Viewer opens with a blank canvas (no data is present). The viewer and its tools are shown below.



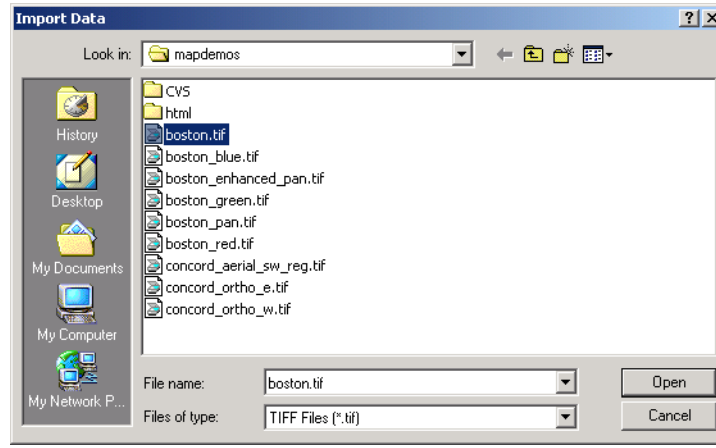
Most of the tool buttons can also be activated from the **Tools** menu.

2 For ease in importing data that is furnished with the Mapping Toolbox, set your working directory as follows:

```
cd(fullfile(matlabroot,'toolbox','map','mapdemos'))
```

However, you can also navigate to this directory with the Map Viewer **Import Data** dialog if you prefer.

3 Select **Import From File** from the **File** menu and open the geoTIFF file `boston.tif` in the Map Viewer, as shown below:



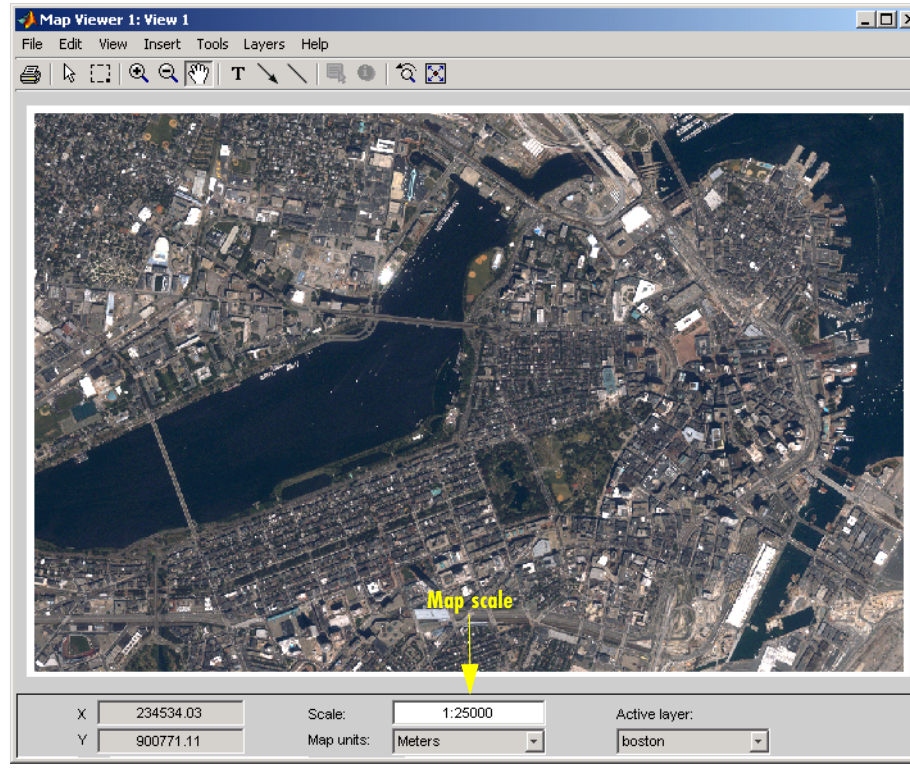
The file opens in the Map Viewer. This is an georeferenced RGB composite image at 4 m resolution covering part of Boston, Massachusetts, USA. The image is a subset of an IKONOS-2 panchromatic/multispectral product created by Space Imaging LLC. For further information, type

```
type boston.txt
```

at the MATLAB prompt.

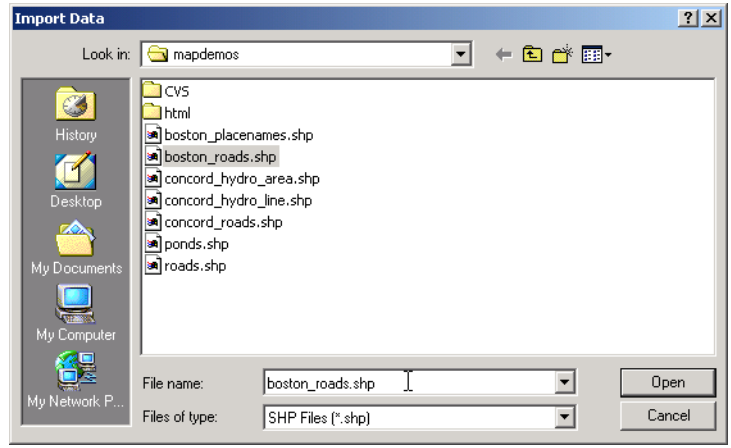
4. To see the map scale, set the map distance units. Use the drop-down **Map units** menu at the bottom center to select `Meters`.

5. Now set the scale to 1:25,000 by typing `1:25000` in the **Scale** box, which is above the **Map units** drop-down. The viewer now looks like this:

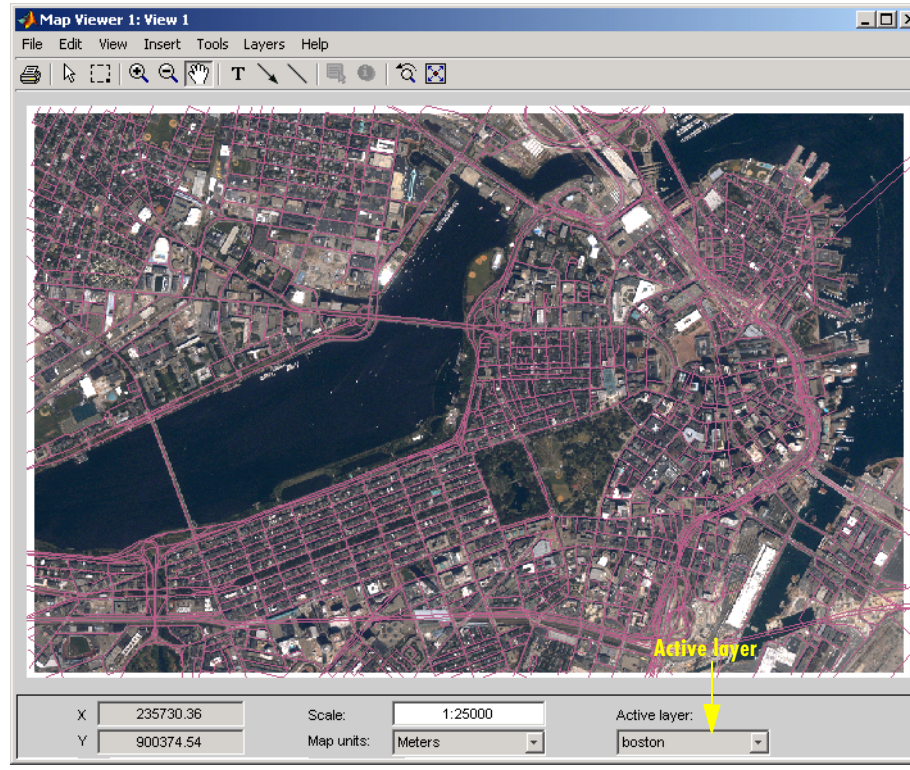


Note that the cursor is pointing at the front of the Massachusetts State House (capitol building). The map coordinates for this location are shown in the readout at the lower left as 235,935.25 meters easting (**X**), 900,923.13 meters northing (**Y**), in Massachusetts State Plane coordinates.

6 Next, import a vector data layer, the streets and highways in the central Boston area. For this you also use **Import From File** from the **File** menu, but this time you specify `SHP` as the type of file to import, and open the shapefile `boston-roads.shp`:



7 After the Map Viewer finishes importing the roads layer, it selects a random color and renders all the shapes with that color as solid lines. The view looks like this:

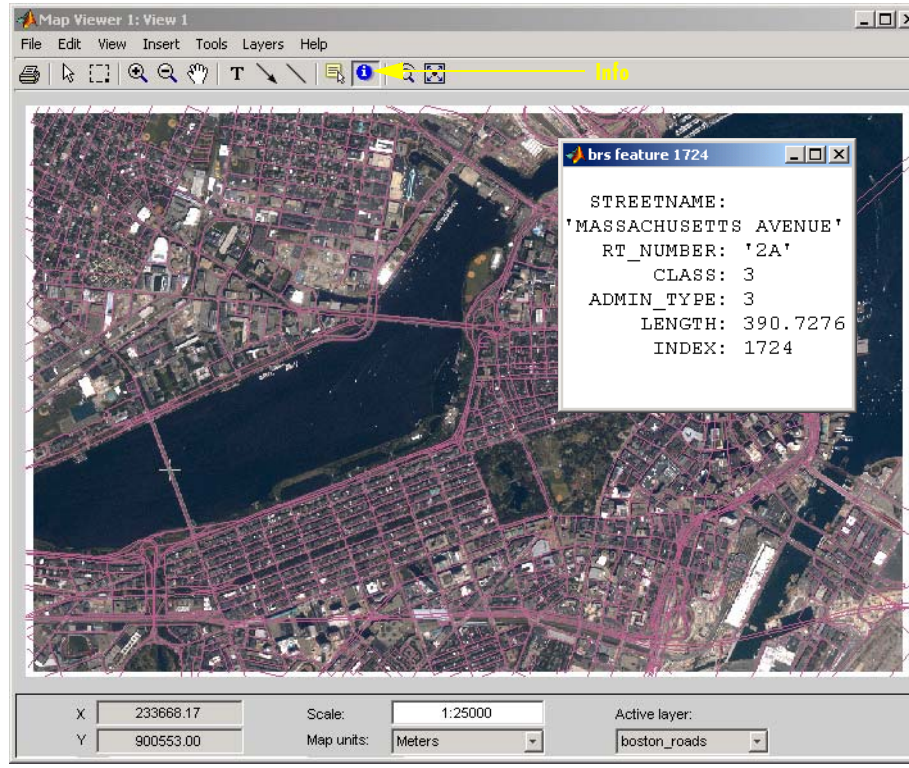


Being random, the color you see for the road layer can differ. How you can specify road colors is discussed below.

8 You can designate any layer to be the *active layer* (one that you can query); it does not need to be the topmost layer. By default no layer is active. Use the **Active layer** drop-down menu at the bottom left to select `boston_roads`.

Changing the active layer has no visual effect. Doing so allows you to query attributes of the layer you select.

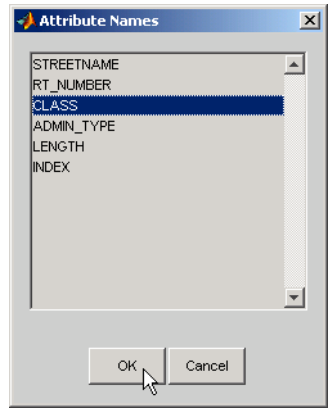
9 One way to see the attributes for a vector layer is to use the **Info** tool, the button on the right end of the toolbar. Select the **Info** tool and click on the bridge across the Charles River near the lower left of the map. This opens a text window displaying the attribute/values for the selected object:



The selected road is Massachusetts Avenue (Route 2A). As the above figure shows, the `boston_roads` vectors have six attributes.

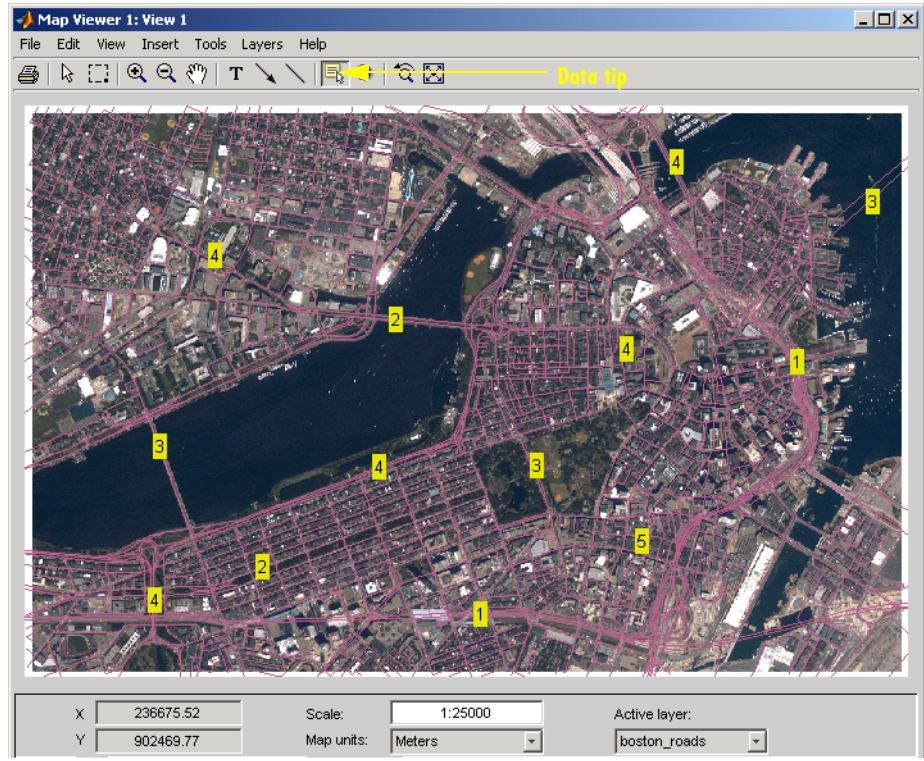
10 Get information about some other roads. Dismiss open Info windows by clicking their close boxes.

11 Choose an attribute for the **Datatip** tool to inspect. From the **Layers** menu, select **boston_roads -> Set Layer Attributes**. From the list in the list box of the **Attribute Names** dialog, select `CLASS` and click **OK** to dismiss it. The dialog looks like this:



12 Select the **Datatip** tool. The cursor assumes a crosshairs (+) shape.

13 Use the **Datatip** tool to identify the administrative class of any road displayed. When you click on a road segment, a datatip is left in that place to indicate the `CLASS` attribute of the active layer, as illustrated below.



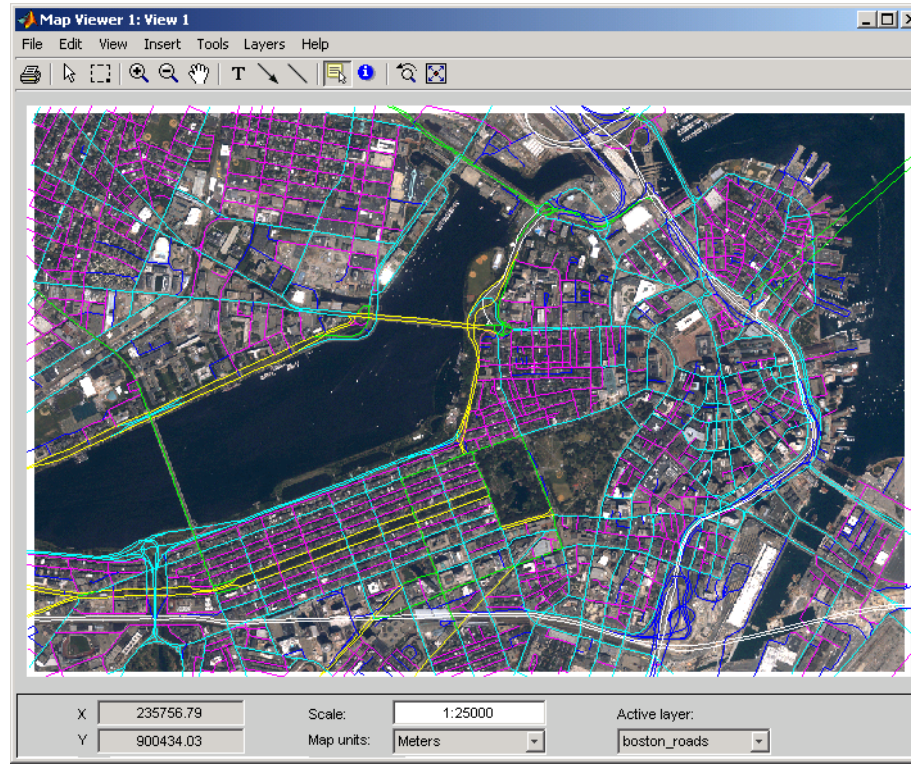
To dismiss datatips, right-click on any of them and select **Delete datatip** or **Delete all datatips** from the pop-up context menu that appears.

14 You can change how the roads are rendered by identifying an attribute to which to key line symbolism. Color roads according to their `CLASS` attribute, which takes on the values 1:6. Do this by creating a *symbolspec* in the workspace. A symbolspec is a cell array that associates attribute names and values to graphic properties for a specified geometric class (`'Point'`, `'MultiPoint'`, `'Line'`, `'Polygon'`, or `'Patch'`). To create a symbolspec for line objects (in this case roads) that have a `CLASS` attribute, type

```
roadcolors = makesymbolspec('Line', ...
{'CLASS',1,'Color',[1 1 1]}, {'CLASS',2,'Color',[1 1 0]}, ...
{'CLASS',3,'Color',[0 1 0]}, {'CLASS',4,'Color',[0 1 1]}, ...
{'CLASS',5,'Color',[1 0 1]}, {'CLASS',6,'Color',[0 0 1]})
```

```
roadcolors =
    ShapeType: 'Line'
        Color: {6x3 cell}
```

15 The Map Viewer recognizes and imports symbolspecs from the workspace. To apply the one you just created, select **boston_roads -> Set Symbol Spec** from the **Layers** menu. From the **Set Symbol Spec** dialog, select the `roadcolors` symbolspec you just created and click **OK**. After `mapview` has read and applied the symbolspec, the map looks like this:



16 Add another layer, a set of points that identify thirteen Boston landmarks. From the **File** menu choose **Import From File** and select `SHP Files` as the file type. Then pick the file `boston_placenames.shp` and Click **Open**.

   The points of interest are symbolized as small `x` markers.

17 As the `boston_placenames` markers are difficult to see over the orthophoto, hide the other map layers temporarily. To do this, go to the **Layers** menu, select **boston_roads**, and then slide right and deselect **Visible**. Do the same to hide the **boston** image layer.

   You can now see the thirteen markers showing points of interest.

18 To make the markers more visually prominent, create a symbolspec for them to represent them as red filled circles. At the MATLAB command line, type

```
places = makesymbolspec('Point',{'Default','Marker','o', ...
'MarkerEdgeColor','r','MarkerFaceColor','r'})
```

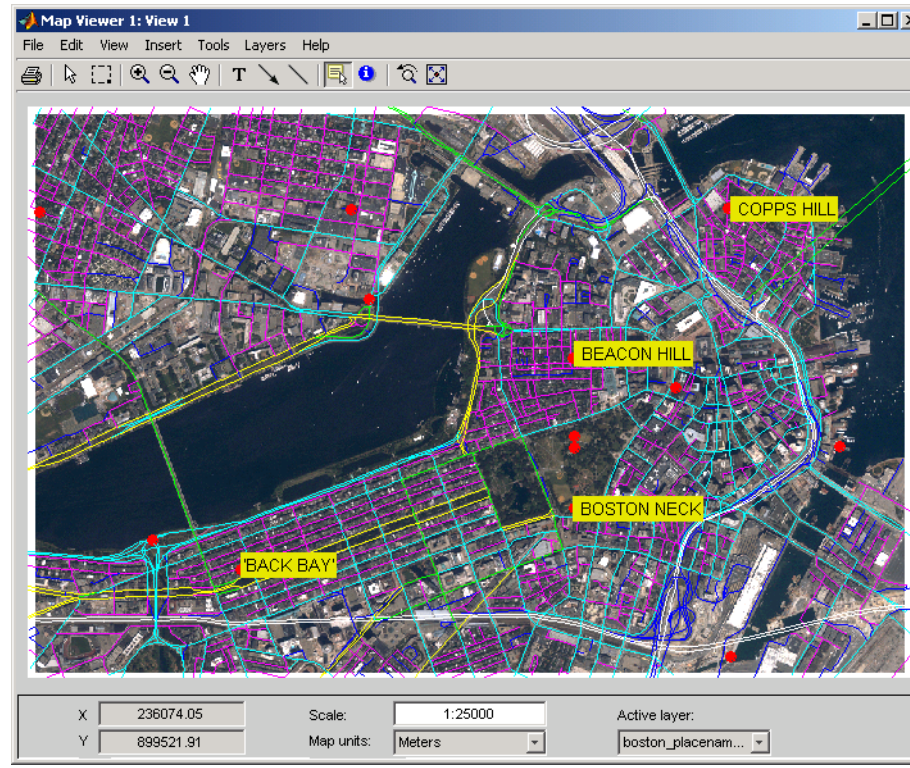
The `Default` keyword causes the specified symbol to be applied to all point objects in a given layer unless specifically overridden by an attribute-coded symbol in the same or a different symbolspec.

19 To activate this symbolspec, pull down the **Layers** menu, select **boston_placenames**, slide right, and select **Set Symbol Spec**. In the **Layer Symbols** dialog that appears, highlight `places` and click **OK**.

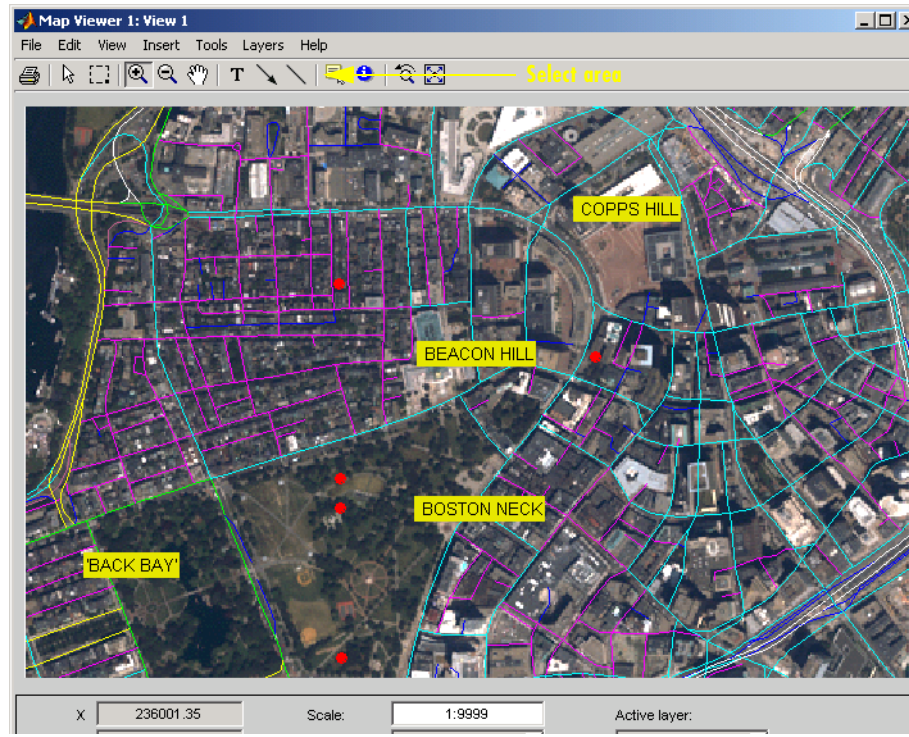
   The Map Viewer reads the workspace variable `places`; the cross marks turn into red circles. Note that a layer need not be active in order for you to apply a symbolspec to it.

20 Now restore the other layers' visibility. In the **Layers** menu, select **boston_roads**, and then slide right and select **Visible**. Do the same to show the **boston** image layer. The `boston_placenames` marker layer, because it was read in most recently, is on top.

21 Use the **Active layer** drop-down menu to make `boston_placenames` the currently active layer, and then select the **Datatip** tool. Click on any red circle to see the name of the feature it marks. The map looks like this (depending on which data tips you show):



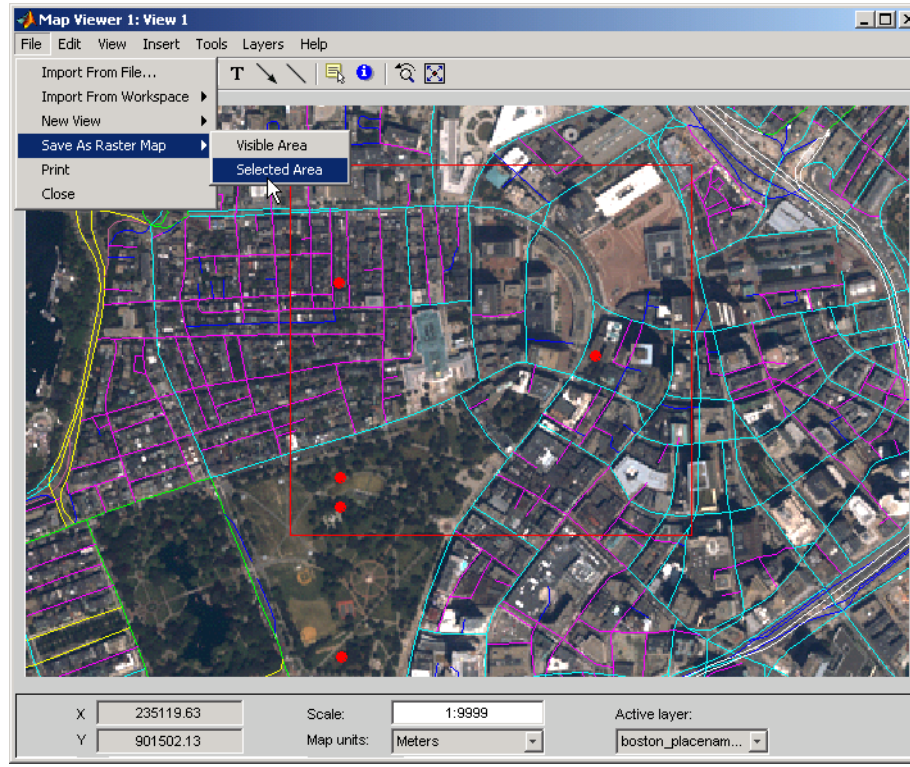
22 Zoom in on Beacon Hill, for a closer view of the Massachusetts State House and Boston Common. Select the **Zoom in** tool, move the (magnifier) cursor until the **X** readout is approximately 236,000 M and the **Y** readout is roughly 900,900 M, then click once to enlarge the view. The scale changes to about 1:10,000 and the map appears as below:



23 Right-click any of the data tips and select **Delete all datatips** from the pop-up context menu. This clears the place names you added to the maps.

24 Select an area of interest to save as an image file. Click on the **Select area** tool, then hold the mouse button down as you draw a selection rectangle. If you do not like the selection, repeat the operation until you are satisfied. If you know what ground coordinates you want, you can use the coordinate readouts to make a precise selection. The selected area appears as a red rectangle.

25 Save your selection as an image file. From the **File** menu, select **Save As Raster Map -> Selected Area** to open a **Save As** dialog, as shown below:



In the **Export to File** dialog, navigate to a directory where you want to save the map image, and give it a name, such as `boston_common`. You can format the image as a TIFF, a PNG, or a JPG file. The result from the selection shown is shown in the following figure:

26 Experiment with other tools and menu items. For example, you can annotate the map with lines, arrows, and text, fit the map to the window, draw a bounding box for any layer, and print the current view. You can also spawn a new Map Viewer using **New View** from the **File** menu. A new view can duplicate the current view, cover the active layer's extent, cover all layer extents, or include only the selected area, if any.

When you are through with a viewing session, close the Map Viewer using the window's close box or select **Close** from the **File** menu.

Documentation Summary

Chapter 1: Getting Started

Begin here to explore the world with the Mapping Toolbox, using `worldmap` and `mapview`. Read this high-level summary of the topics, tools, data, and functions covered in the documentation.

Chapter 3, "Understanding Geospatial Geometry"

Explains, at a high level, the principal concepts that underlie geometric computations on spherical surfaces; for example, spherical and spheroidal coordinates; the concept of a datum; computing distances, directions, and azimuths.

Chapter 2, "Understanding Map Data"

Summary of capabilities; types and formats of geospatial data; base maps, attributes; map coordinate representations and transformations; functions and user interfaces for importing geospatial data files

Chapter 4, "Creating and Viewing Maps"

Functions for displaying map data; using built-in atlas data; setting up map axes; map frames and map grids; symbolizing line data, patch data, and raster data; combining vector and raster data

Chapter 5, "Making Three-Dimensional Maps"

Making perspective views of projected and unprojected data; manipulating digital elevation models; draping data on elevatino maps; shading and lighting terrain

Chapter 6, "Customizing and Printing Maps"

Creating inset maps, north arrows, and graphic scales; types of thematic maps you can make; working with colormaps and colorbars; printing maps to scale

Chapter 7, "Manipulating Geospatial Data"

Useful operations for selecting, thinning, resampling, and combining data sets

Chapter 8, "Using Map Projections and Coordinate Systems"

Mapping 3-D worlds onto 2-D spaces; types, aspects, properties, and parameters of map projections; guidelines for selecting projections and parameters; forward and inverse projection

Chapter 9, "Mapping Applications" (online only)

Using the Mapping Toolbox to compute spatial statistics on the plane and on the sphere; navigational functions and their applications

Chapter 10, "Reference" (online only)

Descriptions of all Mapping Toolbox functions ordered alphabetically, also accessible by category; many descriptions include worked examples.

Chapter 11, "Projections Reference" (online only)

Detailed descriptions of map projections that you can use with functions such as `axesm`, `mfwdproj`, `minvproj`, `projfwd`, and `projinv`

Chapter 12, "GUI Reference" (online only)

Illustrated descriptions of the graphical user interfaces available in the Mapping Toolbox. Some of these appear by default when certain functions are typed without arguments, others are special commands, and a few are sub-dialogs of major GUIs.

"Atlas Data" (online only)

An appendix documenting geospatial data sets that are packaged with the Mapping Toolbox, describing their source, coverage, and examples of use; these are intended to help you generate base maps and thematic maps. The Mapping Toolbox also contains other geodata for specific regions to illustrate aspects of its functionality.

"Bibliography"

Literature you can consult to learn more about mapping

"Glossary"

Definitions of common geographic, geodetic, and cartographic terms

Getting More Help

The Mapping Toolbox documentation is available in electronic form as PDF and HTML files through the `helpdesk` command. You might want to print the reference chapters to browse through them. This is best done from the PDF version, available at the MathWorks Web site, `http://www.mathworks.com/access/helpdesk/help/pdf_doc/map/map_ug.pdf`.

You can find a classified list of functions in the "Guide to Reference Pages" (online only). Help is available for individual commands and classes of Mapping Toolbox commands:

- `help map` for computational functions
- `help mapdemos` for a list of Mapping Toolbox demos
- `maps` lists all Mapping Toolbox map projections by class, name and ID string
- `maplist` returns a structure describing all Mapping Toolbox map projections
- `projlist` to list map projections supported by `projfwd` and `projinv`
- `help` *`functionname`* for help on a specific function, often including examples
- `helpwin` *`functioname`* to see the output of help displayed in the help browser window instead of the command window
- `doc` *`functionname`* to read a function's reference page in the help browser, including examples and illustrations

Mapping Toolbox Demos

You can run demonstrations of Mapping Toolbox functions to further acquaint you with their use. Most of the demos highlight and explain features added in the current version. To see the full list of demos in the Help browser, click on the **Demos** tab in the **Help Navigator** pane, and select **Mapping** under **Toolboxes**. Another way to obtain this list is to type

```
mapdemos
```

at the MATLAB prompt. This will bring the Help browser to the fore.

You also can execute any of the demos listed below by clicking its name:

- `mapexenhance` — Enhancing Multispectral GeoTIFF Images
- `mapexfindcity` — Interactive Global City Finder
- `mapexgeo` — Creating Maps using `geoshow` (for latitude, longitude data)
- `mapexmap` — Creating Maps using `mapshow` (for x, y data)
- `mapexrefmat` — Creating and Using Referencing Matrices
- `mapexreg` — Georeferencing an Image to an Orthotile Base Layer
- `viewmaps` — GUI Demonstrating Map Projections

Note that the above commands run the demo scripts to produce figures, whereas `mapdemos` describes and illustrates the demos in the Help Browser.

You can type

```
help mapdemos
```

to see this list of functions as well as detailed descriptions of the sample data provided.

2

Understanding Map Data

This chapter describes how maps are digitally represented, and the range of data that the Mapping Toolbox can handle. Geodata is coded for computer storage and applications in two principal ways: *vector* and *raster* representations. It has been said that "raster is faster but vector is corrector." There is truth to this, but the situation is more complex. Sections that follow explore these two representations: how they differ, what data structures support them, why you would choose one over the other, and how they can work together in the Mapping Toolbox. It also summarizes the functions available for importing and exporting geodata formats.

Maps and Map Data

The Mapping Toolbox manipulates electronic representations of geographic data. It lets you create, use, and present geographic data in a variety of forms and to a variety of ends. In the digital network era, it is easy to think of geospatial data as maps and maps as data, but you should take care to note the differences between these concepts.

What Is a Map?

The simplest (although perhaps not the most general) definition of a *map* is *a representation of geographic data*. Most people today generally think of maps as two-dimensional; to the ancient Egyptians, however, maps first took the form of lists of place names in the order they would be encountered when following a given road. Today such a list would be considered as *map data* rather than as a map. When most people hear the word "map" they tend to visualize two-dimensional renditions such as printed road, political, and topographic maps, but even classroom globes and computer graphic flight simulation scenes are maps under this definition.

In this toolbox, map data is any variable or set of variables representing a set of geographic locations, properties of a region, or features on a planet's surface, regardless of how large or complex the data is, or how it is formatted. Such data can be rendered as maps in a variety of ways using the functions and user interfaces provided.

What Is Geospatial Data?

Geospatial data comes in many forms and formats, and its structure is more complicated than tabular or even nongeographic geometric data. It is, in fact, a subset of spatial data, which is simply data that indicates where things are within a given *coordinate system*. Mileposts on a highway, an engineering drawing of an automobile part, and a rendering of a building elevation all have coordinate systems, and can be represented as spatial data when properly quantified (digitized). Such coordinate systems, however, are local and not explicitly tied or oriented to the Earth's surface; thus, most digital representations of mileposts, machine parts, and buildings do not qualify as geospatial data (also called *geodata*).

What sets geospatial data apart from other spatial data is that it is absolutely or relatively positioned on a planet, or *georeferenced*. That is, it has a *terrestrial*

coordinate system that can be shared by other geospatial data. There are many ways to define a terrestrial coordinate system and also to transform it to any number of local coordinate systems, for example, to create a map projection. However, most are based on a framework that represents a planet as a sphere or spheroid that spins on a north-south axis, and which is girded by an *equator* (an imaginary plane midway between the poles and perpendicular to the rotational axis).

Types of Map Data Handled by the Mapping Toolbox

Vector data and *raster* data are different concepts and have been generally regarded as being incompatible representations for geospatial data for cartographic purposes. This section explains some of their differences and how the Mapping Toolbox bridges them.

"Vector Geodata" on page 2-4	Map data that codes shapes as points, lines, and polygons
"Raster Geodata" on page 2-7	Map data that dissects space into cells with values, including georeferenced imagery
"Combining Vector and Raster Geodata" on page 2-11	Registering vector data on raster data for display

Vector Geodata

Vector data (in the computer graphics sense rather than the physics sense) can represent a map. Such vectors take the form of sequences of latitude-longitude or projected coordinate pairs representing a point set, a linear map feature, or an areal map feature. For example, points delineating the boundary of the United States, the interstate highway system, the centers of major U.S. cities, or even all three sets taken together, can be used to make a map. In such representations, the geographic data is in *vector* format and displays of it are referred to as *vector maps*. Such data consists of lists of specific coordinate locations (which, if describing linear or areal features, are normally points of inflection where line direction changes), along with some indication of whether each is connected to the points adjacent to it in the list.

In the Mapping Toolbox, vector data consists of sequentially ordered pairs of geographic (latitude, longitude) or projected (x,y) coordinate pairs (also called *tuples*). Successive pairs are assumed to be connected in sequence; breaks in connectivity must be delineated by the creation of separate vector variables or by inserting separators (such as `NaN`s) into the sets at each break point. For vector map data, the connectivity (topological structure) of the data is often only a concern during display, but it also affects the computation of statistics such as length and area.

A Look at Vector Data

1 To inspect an example of vector map data, enter the following commands to MATLAB:

```
load coast
whos
```

```
  Name       Size       Bytes     Class

  lat         9589x1    76712     double array
  long        9589x1    76712     double array
```

The variables `lat` and `long` are vectors in the `coast` MAT-file, which together form a vector map of the coastlines of the world.

2 To view a map of this data, enter these commands:

```
axesm mercator
framem
plotm(lat,long)
```

Inspect the first 20 coordinates of the coastline vector data:

```
[lat(1:20) long(1:20)]
```

```
ans =
      -83.83      -180
      -84.33      -178
      -84.5       -174
      -84.67      -170
      -84.92      -166
      -85.42      -163
      -85.42      -158
      -85.58      -152
      -85.33      -146
      -84.83      -147
      -84.5       -151
      -84         -153.5
      -83.5       -153
      -83         -154
      -82.5       -154
      -82         -154
      -81.5       -154.5
      -81.17      -153
      -81         -150
      -80.92      -146.5
```

Does this give you any clue as to which continent's coastline these locations represent?

3 To see the coastline these vector points represent, type this command to display them in red:

```
plotm(lat(1:20), long(1:20),'r')
```

As you may have deduced by looking at the first column of the data, there is only one continent that lies below -80° latitude, Antarctica.

The above example presents the map in a Mercator projection. A map projection displays the surface of a sphere (or a spheroid) in a two-dimensional plane. As the word "plane" indicates, points on the sphere are geometrically projected to a plane surface. There are many possible ways to project a map, all of which introduce various type of distortions.

For further information on how the Mapping Toolbox handles map projections, see "Using Map Projections and Coordinate Systems" on page 8-1. For details on data structures that the Mapping Toolbox uses to represent vector geodata, see "Vector Geodata" on page 2-4.

Raster Geodata

You can also map data represented as a *matrix* (a 2-D MATLAB array) in which each row-and-column element corresponds to a rectangular patch of a specific geographic area, with implied topological connectivity to adjacent patches. This is commonly referred to as *raster data*. *Raster* is actually a hardware term meaning a systematic scan of an image that encodes it into a regular grid of pixel values arrayed in rows and columns.

When data in raster format represents the surface of a planet, it is called a *data grid*, and the data is stored as an array or matrix. The Mapping Toolbox uses the powerful matrix manipulation capabilities of MATLAB to fully exploit this type of map data. This documentation uses the terms *raster data* and *data grid* interchangeably to talk about geodata stored in two-dimensional array form.

A raster can encode either an average value across a cell or a value sampled (posted) at the center of that cell. While geolocated data grids explicitly indicate which type of values are present (see "Geolocated Data Grids" on page 2-38), external metadata/user knowledge is required to be able to specify whether a regular data grid encodes averages or samples of values.

Digital Elevation Data

When raster geodata consists of surface elevations, the map can also be referred to as a *digital elevation model/matrix* (DEM), and its display is a *topographical map*. The DEM is one of the most common forms of *digital*

terrain model (DTM), which can also be represented as contour lines, triangulated elevation points, quadtrees, octtrees, or otherwise.

The `topo` global terrain data is an example of a DEM. In this 180-by-360 matrix, each row represents one degree of latitude, and each column represents one degree of longitude. Each element of this matrix is the average elevation, in meters, for the one-degree-by-one-degree region of the Earth to which its row and column correspond.

Remotely Sensed Image Data

Raster geodata also encompasses georeferenced imagery. Like data grids, images are organized into rows and columns. There are subtle distinctions, however, which are important in certain contexts. One distinction is that an image may contain RGB or multispectral channels in a single array, so that it has a third (color or spectral) dimension. In this case a 3-D MATLAB array is used rather than a 2-D (matrix) array. Another distinction is that while data grids are stored as class double in the Mapping Toolbox, images may use a range of MATLAB storage classes, with the most common being `uint8`, `uint16`, `double`, and `logical`. Finally, for grayscale and RGB image of class double, the values of individual array elements are constrained to the interval `[0 1]`.

In terms of georeferencing---converting between column/row subscripts and 2-D map or geographic coordinates — images and data grids behave the same way (which is why we consider both to be a form of raster geodata). However, when performing operations that process the values raster elements themselves, including most display functions, it is important to be aware of whether you are working with an image or a data grid, and for images, how spectral data is encoded.

For further details concerning the structure of raster map data, see “Understanding Raster Data” on page 2-27.

A Look at Raster Data

1 To view one possible display of the `topo` data grid, type the following:

```
clear all;
load topo
```

```
whos
  Name             Size                    Bytes  Class

  topo           180x360                518400  double array
  topolegend       1x3                      24  double array
  topomap1        64x3                    1536  double array
  topomap2       128x3                    3072  double array

Grand total is 65379 elements using 523032 bytes
```

2 The raster elevation data is in the variable `topo`. Inspect it with the MATLAB Array Editor by double-clicking `topo` in the Workspace pane or by typing in the Command Window

```
openvar topo
```

You will see that `topo` is a 2-D array, and that its values near its upper left corner range from 2,500 to 3,000 meters of elevation. The first row represents land elevations near the South Pole. When georeferenced with a three-element referencing vector (the variable topolegend in this case), Mapping Toolbox raster data is stored from the bottom up.

3 Create an equal-area map projection to view the topographic data:

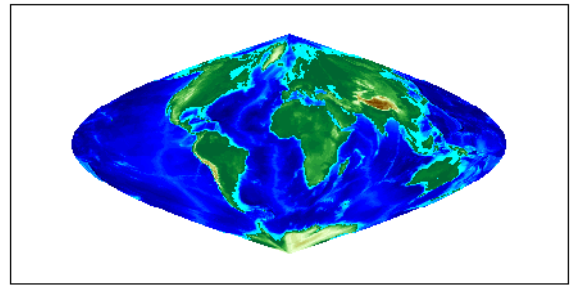
```
axesm sinusoid
```

A MATLAB figure window is created with map axes set to display a sinusoidal projection.

4 Generate a shaded relief map. You can do this in several ways. First use `geoshow` and apply a topographic colormap using `demcmap`:

```
geoshow(topo,topolegend,'DisplayType','texturemap')
demcmap(topo)
```

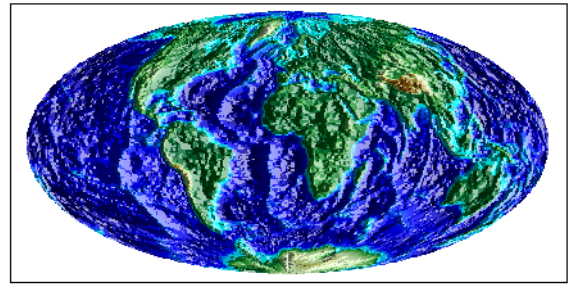
The `geoshow` function displays geodata in geographic (unprojected) coordinates. The `geoshow` output is shown below:



5 Now create a new figure using a Hammer projection (which, like the sinusoidal, is also equal-area), and display `topo` using `meshlsrm`, which enables control of lighting effects:

```
figure; axesm hammer
meshlsrm(topo,topolegend)
```

A colored relief map of the `topo` data set, illuminated from the east, is rendered in the second figure window:



For additional details on controlling the illumination of maps, see “Shading and Lighting Terrain Maps” on page 5-19.

Note that the content, symbolization, and the projection of the map are completely independent. The structure and content of the `topo` variable are the same no matter how you display it, although how it is projected and symbolized can affect its interpretation. The following example illustrates this.

Combining Vector and Raster Geodata

Vector map variables and data grid variables are often used or displayed together. For example, continental coastlines in vector form might be displayed with a grid of temperature data to make the latter more useful. When several map variables are used together, regardless of type, they can be referred to as a single map. To do this, of course, the different data sets must use the same coordinate system (i.e., geographic coordinates on the same ellipsoid or an identical map projection). See "Understanding Geospatial Geometry" on page 3-1 for an introduction to these concepts.

Viewing Raster and Vector Data on the Same Map

Using the `coast` and `topo` data from the previous examples, you can combine them in a single map and see how well the two types of data work together:

1. Clear the current map:

   ```
   clma
   ```

2. Reload the coastline data:

   ```
   load coast
   ```

3. If the topo data is not already in the workspace, load it as well:

   ```
   load topo
   ```

4. Set up a Robinson projection:

   ```
   axesm robinson
   ```

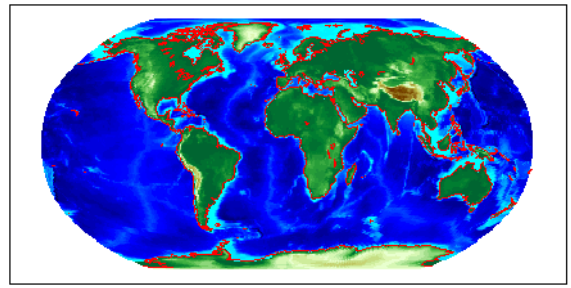
5. Plot the raster topographic data with an appropriate colormap:

   ```
   geoshow(topo,topolegend,'DisplayType','texturemap')
   demcmap(topo)
   ```

6. Plot the coastline data in white on top of the terrain map:

   ```
   geoshow(lat,long,'Color','r')
   ```

   Note that you can use `geoshow` to display both raster and vector data. Here is the resulting map:



For additional details on how the Mapping Toolbox handles raster geodata, see “Understanding Raster Data” on page 2-27.

The remainder of this chapter focuses on the fundamental principles of geographic measurement and data manipulation that are a prerequisite for creating map displays. “Reading and Writing Geospatial Data” on page 2-47 summarizes input functions for importing many formats of geospatial data into the toolbox. “Understanding Geospatial Geometry” on page 3-1 introduces geodetic concepts that underlie all geospatial data and its handling.

Understanding Vector Data

Vector geospatial data is used to represent linear features such as rivers, coastlines, boundaries and highways. Vector data can also represent areal features such as water bodies, political units, and enumeration districts. This section familiarizes you with how vector data structures digitally encode geographic entities and how to use this form of data.

Points, Lines, Polygons

In the context of geodata, *vector data* means "geometric descriptions of geographic objects" rather than its more general mathematical definition, "a quantity specified by a magnitude and a direction." In fact, some vector geodata is specified as points having neither magnitude nor direction. Other geodata — such as postcodes, highway mileposts, or census statistics — only implies an underlying geometry, which vector 2-D coordinate data is required to map or spatially analyze.

In the MATLAB workspace, vector data is expressed as pairs of variables that represent the geographic or plane coordinates for a set of points of interest. For example, the following two variables can be mapped as a vector:

```
lat = [45.6 -23.47 78];
long = [13 -97.45 165];
```

Note that either row or column vectors can be used, but both variables should have the same shape. For example, `lat` and `long` could be defined as columns:

```
lat = [45.6 -23.47 78]';
long = [13 -97.45 165]';
```

These values could mean anything. They could represent three locations over which geosynchronous satellites are stationed, and can be communicated by

plotting a symbol for each point on a map of the Earth. Alternatively, they might represent a starting point, a mid course marker, and a finish point of an sailboat race, in which case they can be rendered by plotting two line segments. Or perhaps the values represent the vertices of a triangle bounding a region of interest, and thus constitute a simple polygon.

Note When polygons become graphic objects, they are called patches. In this documentation, the words *patch* and *polygon* are often used interchangeably.

The Mapping Toolbox provides functionality for each of these interpretations. For many purposes, the distinction is irrelevant; for others, the choice of a function implies one interpretation over the others. For example, the function `plotm` displays the data as a line, while `fillm` displays it as a filled polygon. While you can draw an unfilled polygon with `fillm` that looks like the output from `plotm`, the resulting object has a different graphic data type (*patch* versus *line*), hence different properties you can set.

A line must contain at least two coordinate elements for each coordinate dimension, and a polygon at least three (note that it is not necessary to duplicate the first point as the last point to define or render a polygon). The Mapping Toolbox places no limit (beyond available memory) on how large or how complex the shape of a line and polygon can be, other than the restriction that it should not cross itself.

Objects in the real world that vector geodata represents can have many parts, for example, the islands that make up the state of Hawaii. When encoding as vector variables the shapes of such compound entities, you must separate successive entities. To indicate that such a discontinuity exists, the Mapping Toolbox uses the convention of placing `NaN`s in identical positions in both vector variables. For example, if a second segment is to be added to the preceding map, the two objects can reside in the same pair of variables:

```
lat = [45.6 -23.47 78 NaN 43.9 -67.14 90 -89];
lon = [13 -97.45 165 NaN 0 -114.2 -18 0];
```

Notice that the `NaN`s must appear in the same locations in both variables. Here is a segment of three points separated from a segment of four points. The `NaN`s perform two functions: they provide a means of identifying break points in the data, and they serve as *pen-up* commands when the Mapping Toolbox plots

vector maps. The NaNs are used to separate both distinct (but possibly connecting) lines and disconnected patch faces.

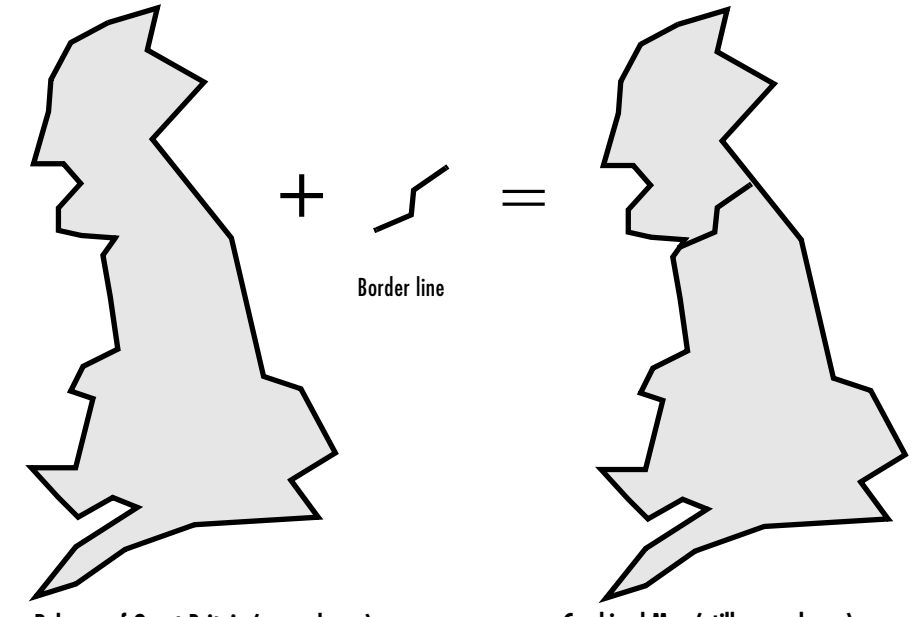
Note This convention departs from regular MATLAB graphics, in which NaN-separated polygons cannot be interpreted or displayed as patches.

Segments Versus Polygons

Geographic objects represented by vector data might or might not be formatted as polygons. Imagine two variables, `latcoast` and `loncoast`, that correspond to a sequence of points that caricature the coast of the island of Great Britain. If this data returns to its starting point, then a polygon describing Great Britain exists. This data might be plotted as a patch or as a line, and it might be logically employed in calculations as either.

Now suppose you want to represent the Anglo-Scottish border, proceeding from the west coast at Solway Firth to the east coast at Berwick-upon-Tweed. This data can only be properly defined as a line, defined by two or more points, which you can represent with two more variables, `latborder` and `lonborder`. When plotted together, the two pairs of variables can form a map. The patch of Great Britain plus the line showing the Scottish border might look like two patches or regions, but there is no object that represents England and no object that represents Scotland, either in the workspace or on the map axes.

In order to represent both regions properly, the Great Britain polygon needs to be split at the two points where the border meets it, and a copy of `latborder` and `lonborder` concatenated to both lines (placing one in reverse order). The resulting two polygons can be represented separately (e.g., in four variables named `latengland`, `lonengland`, `latscotland`, and `lonscotland`) or in two variables that define two polygons each, delineated by NaNs (e.g., `latuk`, `lonuk`).



The distinction between line and polygon data might not appear to be important, but it can make a difference when you are performing geographic analysis and thematic mapping. For example, polygon data can be treated as line data, and displayed with functions such as `linem`, but line data cannot be handled as polygons unless it is restructured to make all objects close on themselves, as described in "Matching Line Segments" on page 7-12.

Mapping Toolbox Geographic Data Structures

In examples provided in prior chapters, geodata was in the form of individual variables and had to be displayed using mapping functions specific to the type of available data (i.e., line, patch, matrix, text, etc.). The Mapping Toolbox also provides an easy means of displaying, extracting, and manipulating collections of all types of map objects that have been organized in a family of specially defined and formatted *geographic data structures* (in general, referred to as a *geostruct*). Note that these structures are different from the *map projection structure* (also called an *mstruct*), which defines a map projection along with its mapping properties (within the `UserData` element of a map axes structure).

The contents of mstructs are described in "Accessing and Manipulating Map Axes Properties" on page 4-6.

The following subsections describe two versions of Mapping Toolbox geographic data structures; the current version of the toolbox uses a form of geographic data structure which is more general than the type found in version 1.x of the toolbox. You can use the older type as well, in appropriate circumstances, and convert it to the newer type when the latter is called for. You should be cognizant of the differences between the two types of structures, because some functions that originate in different versions of the toolbox (for example, `extractm` from version 1 and `extractfield` from version 2) can handle only the type of geostruct introduced in that version of the toolbox.

Version 2 Geographic Data Structures

Certain functions introduced in version 2 of the Mapping Toolbox read, create, or manipulate vector geodata using a geographic data structure format which this document notates as *geostruct2*. This data structure has the flexibility to store any kind and number of attributes, and handles either geographic (latitude and longitude) or plane (x and y) coordinates. In contrast, the version 1 geographic data structure is limited to a fixed set of fields and can contain geographic coordinates only.

The typical way to create a version 2 geographic data structure is to input vector geodata to the workspace from a shapefile. The function `shaperead` returns a geostruct2 that encapsulates some or all the data stored in a group of shapefiles (which store attributes and coordinates in separate files). To determine what kinds of data a group of shapefiles contain, you can use the `shapeinfo` function to query them. `shapeinfo` returns a structure similar to that which `shaperead` returns, but it cannot be used as a geostruct.

You can also transform a geostruct1 into a geostruct2. Use the function `updategeostruct` for this purpose. See "Version 1 Geographic Data Structures" on page 2-19 for a description of that format.

The fields in a geostruct2 depend on the type of geometry and the names and types of the attributes that have been read in. There will always be a text field called `'Geometry'` which identifies the shape type. If the shape type is not `'Point'` there will also be a field called `'BoundingBox'` which contains `[minX minY; maxX maxY]`.

Coordinate data are stored in fields called `'X'` or `'Lon'` and `'Y'` or `'Lat'`, depending on what type of coordinates were read in. The names of these fields

are used by functions to determine if coordinates are projected or unprojected. However, the geostruct does not itself identify what map projection may be used or what its parameters are.

The remainder of the fields store attribute data. The fields are given appropriately mangled names by `shaperead` if the original attribute name could not be directly used as a field name. Unwanted attributes can be filtered out by `shaperead`.

Here is an example of an unfiltered geostruct returned by `shaperead`:

```
S = shaperead('concord_roads.shp')
S =
609x1 struct array with fields:
    Geometry
    BoundingBox
    X
    Y
    STREETNAME
    RT_NUMBER
    CLASS
    ADMIN_TYPE
    LENGTH
```

This indicates that the shapefile contains 609 features. Each one can contain any number of shape points, but will possess the same attribute fields (any of which can be empty). For example, the tenth element has nine coordinates:

```
S(10)
ans =
       Geometry: 'Line'
    BoundingBox: [2x2 double]
              X: [1x9 double]
              Y: [1x9 double]
     STREETNAME: 'WRIGHT FARM'
      RT_NUMBER: ''
          CLASS: 5
     ADMIN_TYPE: 0
         LENGTH: 79.0347
```

For additional information about geographic data structures, see the reference page for `updategeostruct`.

Version 1 Geographic Data Structures

Map ping Toolbox version 1 geographic data structures, which are more fixed in their content, contain information required for the display of graphic objects within map axes. This document notates the older format as a *geostruct1*. The objects that a geostruct1 describes are for the most part MATLAB figure graphic objects. Coordinate data is always given in latitude and longitude. The following table lists the six object types a geostruct1 may contain, and indicates which fields of information are required for each:

Field	Light	Line	Patch	Regular	Surface	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 object 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 a string 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, refvec is its corresponding data grid legend, 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 string, 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 Most of the Mapping Toolbox built-in atlas data, as well as data sets from public sources that are read by Mapping Toolbox external interface functions, enter the MATLAB workspace as geographic data structures. You can also format your own data as geographic data structures.

You can find additional details about version 1 geographic data structures in the references pages for displaym, extractm, and mlayers.

Understanding the Version 1 Geographic Data Structure

Try the following exercise to help you understand what is in a geostruct1:

1. The usalo data set contains several variables in the geographic data structure format:

   ```
   clear all; close all; load usalo
   ```

2. Type whos to identify what was loaded into the workspace:

   ```
   whos
     Name              Size                    Bytes  Class

     conus              1x1                     70224  struct array
     greatlakes         1x3                     21612  struct array
     gtlakelat       1229x1                      9832  double array
     gtlakelon       1229x1                      9832  double array
   ```

```
  state             1x51                  246970  struct array
  stateborder       1x1                    38358  struct array
  statelat       2345x1                    18760  double array
  statelon       2345x1                    18760  double array
  uslat          4339x1                    34712  double array
  uslon          4339x1                    34712  double array

Grand total is 61521 elements using 503772 bytes
```

3 Examine the `conus` variable:

```
conus
conus =
              lat: [4392x1  double]
             long: [4392x1  double]
             type: 'patch'
              tag: 'ContinentalUnitedStates'
    otherproperty: []
         altitude: []
```

This indicates that `conus` consists of one or more patches representing the continental U.S.

4 Examine the `stateborder` variable:

```
stateborder
stateborder =
              lat: [2345x1 double]
             long: [2345x1 double]
             type: 'line'
              tag: 'StateBorder'
    otherproperty: {   1x1 cell  }
         altitude: []
```

The `stateborder` structure contains line data for the state boundaries. It contains one property in its `otherproperty` field.

5 Examine the `greatlakes` variable:

```
greatlakes
greatlakes =
1x3 struct array with fields:
```

```
    type
    tag
    lat
    long
    altitude
    otherproperty
```

The `greatlakes` structure consists of three patches representing the Great Lakes. Identify them:

```
greatlakes(:).tag
ans =
   SuperiorMichiganHuron
ans =
   Erie
ans =
   Ontario
```

The three larger lakes are represented as one polygon.

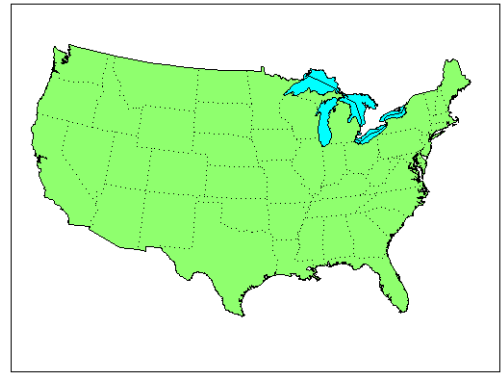
6 These vector objects can be displayed with the `geoshow` function. Define a map axes with a Lambert conic projection with limits appropriate for the U.S.:

```
axesm('MapProjection','lambert','MapParallels',[],...
     'MapLatLimit',[23 52],'MapLonLimit',[-130 -62])
```

7 Draw the U.S., then the Great Lakes, then the state boundaries:

```
geoshow(conus)
geoshow(greatlakes)
geoshow(stateborder)
```

The resulting map is virtually identical to that of the example given in “Displaying Vector Maps as Patches” on page 4-29, which used the vector variables `uslat`, `uslon`, `gtlakelat`, `gtlakelon`, `statelat`, and `statelon`:



8 Notice that the colors displayed for polygons in this map are specified by the `otherproperty` field of the patch data:

```
greatlakes.otherproperty
ans =
    'FaceColor'    'cyan'
ans =
    'FaceColor'    'cyan'
ans =
    'FaceColor'    'cyan'

conus.otherproperty
ans =
     []
```

9 The `otherproperty` field of `conus` does not specify a `FaceColor`, and `displaym` sets it to green. You can easily change it to red, but you will need to display the map again to see the change:

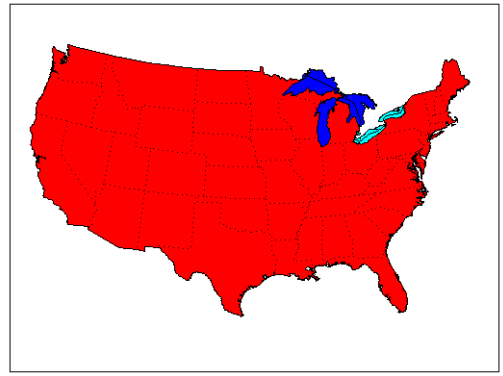
```
conus.otherproperty = {'FaceColor','red'}
displaym(conus)
```

To use `otherproperty` for display, you must use `displaym`. While you can also display this data with `geoshow`, you will need to provide `geoshow` with a symbolspec to set `FaceColor` (or other graphic attributes).

10 Now set the `greatlakes` polygons representing lakes Superior, Huron and Michigan to `blue` and display again:

```
greatlakes(1).otherproperty={'FaceColor','blue'}
displaym(greatlakes)
```

The map now looks like this:



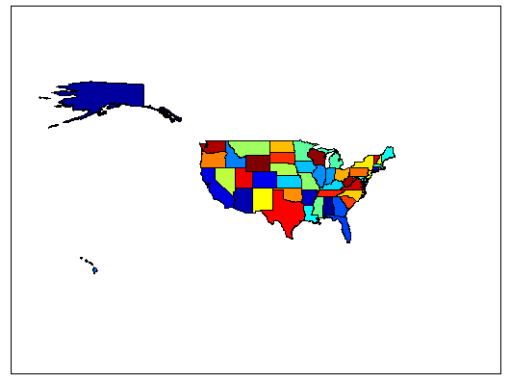
Displaying Specific Vector Elements

You can access and display individual elements of a geostruct1 as well as entire arrays of elements. For example, the `usalo` data set contains a polygonal outline for each state in its `state` structure. Do the following:

1 Create a new map axes and display polygons for all 50 states and the District of Columbia represented in the workspace variable `state`:

```
figure
axesm trystan
displaym(state)
```

This creates a U.S. map in the Trystan Edwards Cylindrical projection. Patch colors are sequentially assigned by state index number, as the geostruct1 `otherproperty` elements that might have specified them are empty for `state` polygons:



2 Now create a new map figure using the same projection:

```
figure
axesm trystan
```

3 View the state of Connecticut by name (its `tag` attribute):

```
displaym(state,'conn')
```

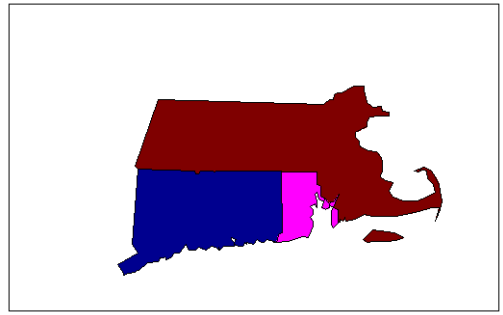
4 Display the state of Massachusetts by its index number:

```
displaym(state(21))
```

5 Use the `extractm` function to copy coordinate data for Rhode Island from the `state` structure, and display it directly with `patchm`:

```
[rilat,rilong] = extractm(state,'rhode island');
patchm(rilat,rilong,'m')
```

Notice that at each of the last three steps, which perform the same function, the map frame automatically readjusted to contain the polygons being displayed. Here is the final result.



You can also use the `mlayers` tool to query members of a geostruct1 and plot them in a map axes.

Understanding Raster Data

As the section "Raster Geodata" on page 2-7 explains, raster geodata consists of georeferenced data grids and images that MATLAB stores internally as matrices. While raster geodata looks like any other matrix of real numbers, what sets it apart is that it is georeferenced, either to the globe or to a specified map projection, so that each pixel of data occupies a known patch of territory on the planet.

Georeferencing Raster Data

Whether a raster geodata set covers the entire planet or not, its placement and resolution must be specified. Raster geodata is georeferenced in the Mapping Toolbox through a companion data structure called a *referencing matrix*. This 3-by-2 matrix of doubles describes the scaling, orientation, and placement of the data grid on the globe. For a given referencing matrix, R, one of the following relations holds between rows and columns and coordinates (depending on whether the grid is based on map coordinates or geographic coordinates, respectively):

```
[x y] = [row col 1] * R, or
[long lat] = [row col 1] * R
```

For additional details about and examples of using referencing matrices, see the reference page for `makerefmat`.

Referencing Vectors

In many instances (when the data grid or image is based on latitude and longitude and is aligned with the geographic graticule), a referencing matrix has more degrees of freedom than the data requires. In such cases, you can use a more compact representation, a three-element r*eferencing vector*. A

referencing vector defines the pixel size and northwest origin for a regular, rectangular data grid:

```
refvec = [cells-per-degree north-lat west-lon]
```

In MAT-files, this variable is often called `refvec` or `maplegend`. The first element, `cells-per-degree`, describes the angular extent of each grid cell (e.g., if each cell covers five degrees of latitude and longitude, `cells-per-degree` would be specified as `0.2`). Note that if the latitude extent of cells differs from their longitude extent you cannot use a referencing vector, and instead must specify a referencing matrix. The second element, `north-lat`, specifies the northern limit of the data grid (as a latitude), and the third element, `west-lon`, specifies the western extent of the data grid (as a longitude). In other words, `north-lat, west-lon` is the northwest corner of the data grid. Note, however, that cell (1,1) is always in the southwest corner of the grid. This need not be the case for grids or images described by referencing matrices, as opposed to referencing vectors.

Note Versions of the Mapping Toolbox prior to 2.0 did not use referencing matrices, and called referencing vectors *map legend vectors* or sometimes just *map legend*s. The current version of the toolbox uses the term *legend* only to refer to keys to symbolism.

An example of such a grid is the `geoid` data set (a MAT-file), which represents the shape of the geoid. In the `geoid` matrix, each cells represents one degree, the entire northern edge occupies the north pole, the southern edge occupies the south pole, and the western edge runs down the prime meridian. Thus, the referencing vector for `geoid` is

```
geoidrefvec = [1 90 0]
```

This structure is stored in the `geoid` MAT-file (note that it is duplicated by the `geoidlegend` referencing vector for backward compatibility). Interpret this referencing vector as follows:

- Each data grid entry represents one degree of latitude and one degree of longitude.
- The northern edge of the map is at 90°N (the North Pole).
- The western edge of the map is at 0° (the prime meridian).

All regular data grids require a a referencing matrix or vector, even if they cover the entire planet. Geolocated data grids do not, as they explicitly identify the geographic coordinates of all rows and columns. For details on geolocated grids, see “Geolocated Data Grids” on page 2-38. For additional information on referencing matrices and vectors, see the reference pages for `makerefmat`, `limitm`, and `sizem`.

Regular Data Grids

Regular data grids are rectangular, not sparse, matrices that contain double values. MATLAB stores them in column order, with their southern edge as the first row and their northern edge as their last row.

Constructing a Global Data Grid

Imagine an extremely coarse map of the world in which each cell represents 60°. Such a map matrix would be 3-by-6, and its referencing vector would be defined as

```
refvec = [1/60 90 -180] = [0.0167 90 -180]
```

1. First create data for this, starting with the data grid itself:

   ```
   minigrid=[1 2 3 4 5 6; 7 8 9 10 11 12; 13 14 15 16 17 18];
   ```

2. Now make a referencing vector, as described above:

   ```
   minivec= [1/60 90 -180]
   minivec =
       0.0167   90.0000 -180.0000
   ```

   As is often the case for global grids, the western edge is the international date line, at 180°W:

3. Set up an equidistant cylindrical map projection:

   ```
   axesm('MapProjection', 'eqdcylin')
   setm(gca, 'MapLatLimit',[-90 90],'MapLonLimit',[-180 180],...
   'GLineStyle','-', 'Grid','on','Frame','on')
   ```

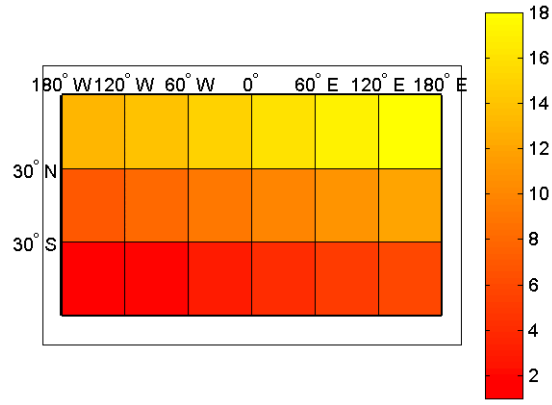
4. Draw a graticule with parallel and meridian labels at 60° intervals:

   ```
   setm(gca, 'MlabelLocation', 60, 'PlabelLocation',[-30 30],...
   'MLabelParallel','north', 'MeridianLabel','on',...
   ```

```
'ParallelLabel','on',...
'MlineLocation',60, 'PlineLocation',[-30 30])
```

5 Map the data using `meshm` and display with a color ramp and legend:

```
meshm(minigrid, minivec); colormap('autumn'); colorbar
```



Note that the first row of the matrix is displayed as the bottom of the map, while the last row is displayed as the top. All regular data grids in the Mapping Toolbox, as well as regular surfaces in MATLAB, are displayed in this fashion.

Computing Map Limits from Reference Vectors

Given a regular data grid and its reference vector, the full extent of the grid can be computed using the `limitm` function. To understand how this works for a data grid that does not encompass the entire world, do the following exercise:

1 Load the Cape Cod 30-arc-second elevation matrix using the `loadcape` script and inspect the reference vector, `maplegend`:

```
loadcape
maplegend
maplegend =
```

```
   120    44   -72
```

The `maplegend` referencing vector indicates that there are 120 cells per angular degree. This horizontal resolution is 120 times finer than that of the `topo` data grid, which was one cell per degree.

2 Use `limitm` to determine that the `cape` region extends from 41°N to 44°N and from 72°W to 69°W:

```
[latlimits,longlimits] = limitm(map,maplegend)
latlimits =
   41    44
longlimits =
   -72   -69
```

3 Verify this computation manually by getting the dimensions of the elevation matrix and computing the eastern and southern map limits from the reference vector:

```
[rows cols] = size(map)
rows =
   360
cols =
   360
southlat = maplegend(2) - rows/maplegend(1)
southlat =
    41
eastlon = maplegend(3) + cols/maplegend(1)
eastlon =
   -69
```

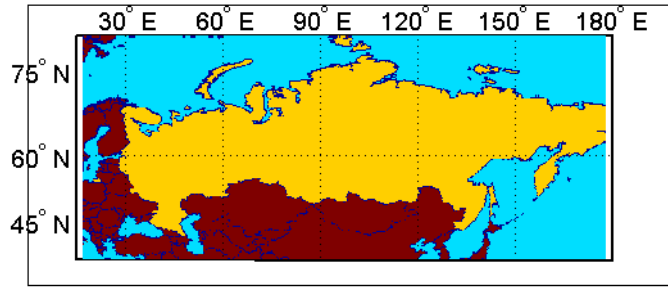
The results match `latlimits(1)` and `longlimits(2)`. The two formulas use different signs because latitudes decrease southwards and longitudes increase eastward.

Geographic Interpretation of Matrix Elements

You can access and manipulate gridded geodata and its associated referencing vector by either geographic or matrix coordinates. Use the `russia` data set to explore this. As was demonstrated above, the north, south, east, and west limits of the mapped area can be determined as follows:

```
clear; load russia
[latlim,longlim] = limitm(map,refvec)
latlim =
   35    80
longlim =
   15   190
```

The data grid in the `russia` MAT-file extends over the international date line (180° longitude). You could use the previously described function `npi2pi` to rename the eastern limit to be -170, or 170°W.



The function `setltln` retrieves the geographic coordinates of a particular matrix element. The returned coordinates actually show the center of the geographic area represented by the matrix entry:

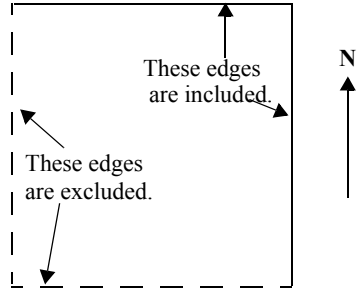
```
row = 23; col = 79;
[lat,long] = setltln(map,refvec,row,col)
lat =
   39.5
long =
   30.7
```

`setpostn` does the reverse of this, determining the row and column of the data grid element containing a given geographic point location:

```
[r,c] = setpostn(map,maplegend,lat,long)
r =
   23
c =
   79
```

The Geography of Gridded Geodata

Each matrix element (analogous to a pixel) can be thought of as a spheroidal *quadrangle*, which includes its northern and eastern edges, but not its western edge or southern edge.



An Element in a Data Grid

The exceptions to this are that the southernmost row (row 1) also contains its southern edge, and the westernmost column (column 1) contains its western edge, except when the map encompasses the entire 360° of longitude. In that case, the westernmost edge of the first column is not included, because it is identical to the easternmost edge of the last column. These exceptions ensure that all points on the globe can be represented exactly once in a regular data grid.

Although each data grid element represents an area, not a point, it is often useful to assign singular coordinates to provide a point of reference. The `setltln` function does this. It geolocates an element by the point in the center of the area represented by the element. The following code references the center cell coordinate for the row 3, column 17 of the Russia map:

```
clear; load russia
row = 3; col = 17;
[lat,long] = setltln(map,refvec,row,col)
lat =
   35.5
long =
   18.3
```

Since the cells in the `russia` matrix represent 0.2° squares (5 cells per degree), the cell in question extends from north of 35.4°S to exactly 35.6°S, and from east of 18.2°E to exactly 18.4°E.

Accessing Data Grid Elements

The actual values contained within the map matrix entries are important as well. The Mapping Toolbox provides several functions for accessing and altering the values of data grid elements.

If the actual row and column of a desired entry are known, then a simple matrix index can return the appropriate value:

1. Use the row and column from the previous example (row 3, column 17) to determine the value of that cell simply by querying the matrix:

```
value = map(row,col)
value =
   2
```

2. More often, the geographic coordinates are known, and the value can be retrieved with `ltln2val`:

```
value = ltln2val(map,maplegend,lat,long)
value =
   2
```

3. The latitude-longitude coordinates associated with particular values in a data grid can be found with `findm`, analogous to the MATLAB function `find`. Here the coordinates of elements in the `topo` matrix have values greater than 5,500 meters:

```
load topo
[lats,longs] = findm(topo>5500,topolegend);
[lats longs]
ans =
   34.5000   79.5000
   34.5000   80.5000
   30.5000   84.5000
   28.5000   86.5000
```

4. To get the row and column indices instead, simply use the MATLAB `find` function:

```
[i,j]=find(topo>5500)
i =
   125
   125
```

```
   121
   119
j =
    80
    81
    85
    87
```

5 To recode a specific matrix value to some other value, use `changem`. Load or reload the `russia` MAT-file, and then change all instances of a given value in a data grid to a new value in one step:

```
oldcode = ltln2val(map,maplegend,37,79)
oldcode =
   4
newmap = changem(map,5,oldcode);
newcode = ltln2val(newmap,maplegend,37,79)
newcode =
   5
```

All entries in `newmap` corresponding to 4's in `map` now have the value 5.

Using a Mask to Recode a Data Grid

You can also define a logical mask to identify the map entries to change. A mask is a matrix the same size as the map matrix, with 1's everywhere that values are to change. A mask is often generated by a logical operation on a map variable, a topic that is described in greater detail below:

1 The `russia` data grid contains `3` for each cell covering Russia. To set every non-Russia matrix entry to zero, use the following MATLAB commands:

```
clear; load russia
nonrussia = map;
nonrussia(map~=3) = 0;
```

2 Verify the data that results from these operations:

```
whos
Name             Size                    Bytes  Class
```

```
  clrmap            4x3                      96  double array
  description       5x69                    690  char array
  map             225x875                 1575000  double array
  maplegend         1x3                      24  double array
  nonrussia       225x875                 1575000  double array
  refvec            1x3                      24  double array
  source            1x68                    136  char array

Grand total is 394181 elements using 3150970 bytes

newcode = ltln2val(nonrussia,refvec,37,79)
newcode =
   0
```

Precomputing the Size of a Data Grid

Finally, if you know the latitude and longitude limits of a region, you can calculate the required matrix size and an appropriate referencing vector for any desired map resolution and scale. However, before making a large, memory-taxing data grid, you should first determine what its size will be. For a map of the continental U.S. at a scale of 10 cells per degree, do the following:

1. Compute the matrix dimensions using `sizem`, specifying latitude limits of 25°N to 50°N and longitudes from 60°W to 130°W:

   ```
   cellsperdeg = 10;
   [r,c,maplegend] = sizem([25 50],[-130 -60],cellsperdeg)
   r =
      250
   c =
      700
   maplegend =
       10    50  -130
   msize = r * c * 8
   msize =
        1400000
   ```

   This data grid would be 250-by-700, and consume 1,400,000 bytes.

2. Now determine what the storage requirements would be if the scale were reduced to 5 rows/columns per degree:

```
cellsperdeg2 = 5;
[r,c,maplegend] = sizem([25 50],[-130 -60],cellsperdeg2)
r =
   125
c =
   350
maplegend =
     5    50  -130

msize = r * c * 8
msize =
      350000
```

A 125-by-300 matrix that used 350,000 bytes might be more manageable, if it had sufficient resolution at its intended publication scale.

Geolocated Data Grids

In addition to regular data grids, the Mapping Toolbox provides another format for geodata: *geolocated data grids*. These multivariate data sets can be displayed, and their values and coordinates can be queried, but unfortunately much of the functionality supporting regular data grids is not available for geolocated data grids.

The examples thus far have shown maps that covered simple, regular quadrangles, that is, geographically rectangular and aligned with parallels and meridians. Geolocated data grids, in addition to these rectangular orientations, can have other shapes as well.

Geolocated Grid Format

To define a geolocated data grid, you must define three variables.

- A matrix of indices or values associated with the mapped region
- A matrix giving cell-by-cell latitude coordinates
- A matrix giving cell-by-cell longitude coordinates.

The following exercise demonstrates this data representation:

1 Load the MAT-file example of an irregularly shaped geolocated data grid called `mapmtx`:

```
load mapmtx
whos
   Name       Size       Bytes     Class

   lg1        50x50      20000     double array
   lg2        50x50      20000     double array
   lt1        50x50      20000     double array
   lt2        50x50      20000     double array
   map1       50x50      20000     double array
   map2       50x50      20000     double array
```

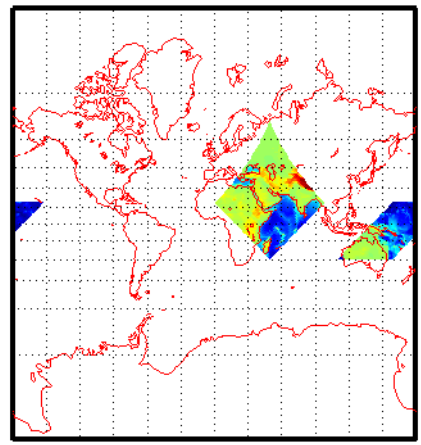
Two geolocated data grids are in this data set, each requiring three variables. The values contained in `map1` correspond to the latitude and longitude coordinates, respectively, in `lt1` and `lg1`. Notice that all three matrices are the same size. Similarly, `map2`, `lt2`, and `lg2` together form a second geolocated data grid. These data sets were extracted from the `topo` data grid shown in previous examples. Neither of these maps is regular, because their columns do not run north to south.

2 To see their geography, display the grids one after another:

```
close all
axesm mercator
gridm on
framem on
h1=surfm(lt1,lg1,map1);
h2=surfm(lt2,lg2,map2);
```

3 Showing coastlines will help to orient you to these skewed grids:

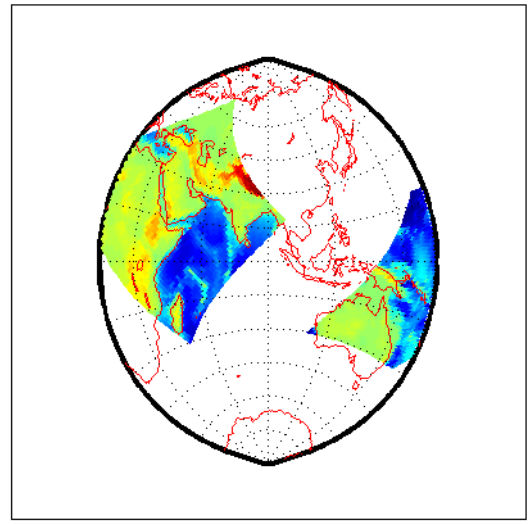
```
load coast
plotm(lat,long,'r')
```



Notice that neither `topo` matrix is a regular rectangle. One looks like a diamond geographically, the other like a trapezoid. The trapezoid is displayed in two pieces because it crosses the edge of the map. These shapes can be thought of as the geographic organization of the data, just as rectangles are for regular data grids. But, just as for regular data grids, this organizational logic does not mean that displays of these maps are necessarily a specific shape.

4 Now change the view to a polyconic projection with an origin at 0°N, 90°E:

```
setm(gca,'MapProjection','polyc', 'Origin',[0 90 0])
```



As the polyconic projection is limited to a 150° range in longitude, those portions of the maps outside this region are automatically trimmed.

Geographic Interpretations of Geolocated Grids

The Mapping Toolbox supports three different interpretations of geolocated data grids:

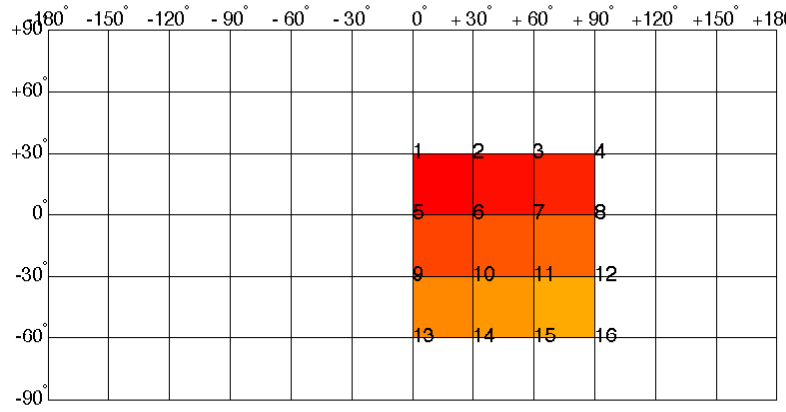
- First, a map matrix having the same number of rows and columns as the latitude and longitude coordinate matrices represents the values of the map data at the corresponding geographic points (centers of data cells).
- Next, a map matrix having one fewer row and one fewer column than the geographic coordinate matrices represents the values of the map data within the area formed by the four adjacent latitudes and longitudes.
- Finally, if the latitude and longitude matrices have smaller dimensions than the map matrix, you can interpret them as describing a coarser *graticule*, or mesh of latitude and longitude cells, into which the blocks of map data are warped.

This section discusses the first two interpretations of geolocated data grids. For more information on the use of graticules, see“The Map Grid” on page 4-23.

Type 1: Values associated with upper left grid coordinate. As an example of the first interpretation, consider a 4-by-4 map matrix whose cell size is 30-by-30 degrees, along with its corresponding 4-by-4 latitude and longitude matrices:

```
map = [ 1  2  3  4;...
   5  6  7  8;...
   9 10 11 12;...
   3 14 15 16];

lat = [ 30  30  30  30;...
   0   0   0   0;...
   -30 -30 -30 -30;...
   -60 -60 -60 -60];

long = [0 30 60 90;...
   0 30 60 90;...
   0 30 60 90;...
   0 30 60 90];
```

This geolocated data grid is displayed with the values of `map` shown at the associated latitudes and longitudes.

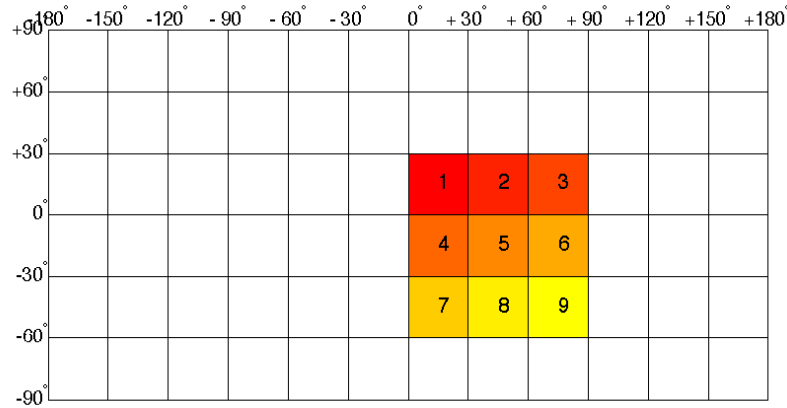


Notice that only 9 of the 16 total cells are displayed. The value displayed for each cell is the value at the upper left corner of that cell, whose coordinates are given by the corresponding `lat` and `long` elements. By MATLAB convention, the last row and column of the map matrix are not displayed, although they exist in the `CData` property of the surface object.

Type 2: Values centered within four adjacent coordinates. For the second interpretation, consider a 3-by-3 map matrix with the same `lat` and `long` variables:

```
map = [1 2 3;...
       4 5 6;...
       7 8 9];
```

Here is a surface plot of the map matrix, with the values of `map` shown at the center of the associated cells:



All the map data is displayed for this geolocated data grid. The value of each cell is the value at the center of the cell, and the latitudes and longitudes in the coordinate matrices are the boundaries for the cells.

Ordering of Cells. You may have noticed that the first row of the matrix is displayed as the top of the map, whereas for a regular data grid, the opposite was true: the first row corresponded to the bottom of the map. This difference is entirely due to how the `lat` and `long` matrices are ordered. In a geolocated data grid, the order of values in the two coordinate matrices determines the arrangement of the displayed values.

Transforming Regular to Geolocated Grids. When required, a regular data grid can be transformed into a geolocated data grid. This simply requires that a pair of coordinates matrices be computed at the desired spatial resolution from the regular grid. Do this with the `meshgrat` function, as follows:

```
load topo
[lat,lon] = meshgrat(topo,topolegend);
whos
NameSizeBytesClass

lat 180x360518400double array
lon 180x360518400double array
topo 180x360518400double array
```

```
topolegend1x3 24double array
topomap164x31536double array
topomap2128x33072double array
```

Transforming Geolocated to Regular Grids. Conversely, a regular data grid can also be constructed from a geolocated data grid. The coordinates and values can be embedded in a new regular data grid. The resolution of the regular grid to be output should be somewhat coarser than that of the geolocated data grid to avoid missing values. The function that performs this conversion is `imbedm`; it takes a geolocated data grid, a referencing vector, and a logical grid as inputs. To see how this works, convert the diamond-shaped geolocated data grid from the `mapmtx` data set back to a regular data grid at a resolution of one row and column for every two degrees of latitude and longitude:

1. Clear the workspace and load the geolocated data grid:

   ```
   close all; clear all; load mapmtx
   ```

2. Determine the extremes of latitude and longitude of the diamond-shaped section:

   ```
   latlim = [min(lt1(:)) max(lt1(:))];
   lonlim = [min(lg1(:)) max(lg1(:))];
   ```

3. Create a matrix of `NaN`s at the desired resolution (about twice as coarse as the original):

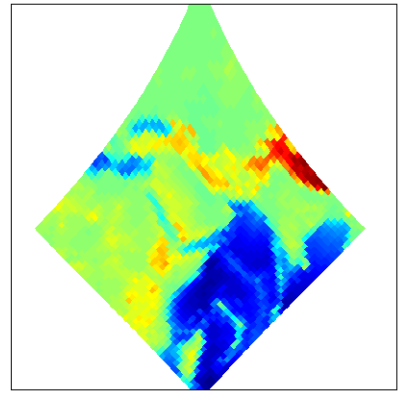
   ```
   [regrid,regridref] = nanm(latlim,lonlim,1/2);
   ```

4. Embed the values of the geolocated data grid in the grid of `NaN`s:

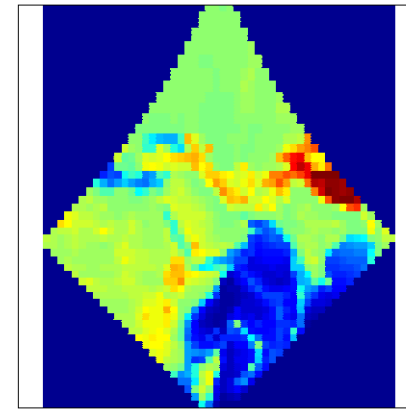
   ```
   regrid = imbedm(lt1,lg1,map1,regrid,regridref);
   ```

5. Map the resulting regular data grid and compare it to the plot of the geolocated data grid from the section "Geolocated Grid Format" on page 2-38:

   ```
   axesm mercator
   gridm on
   framem on
   meshm(map, maplegend)
   ```



Original geolocated data grid, one degree resolution



Transformed to a regular data grid, two degree resolution

Notice that the grid cells are now rectangular rather than diamond-shaped and that the display is more generalized due to having lower resolution.

Finally, note that should more than one value from the geolocated data grid be assigned to the same regular grid element, the last one to be assigned "wins." In other words, no control is provided over aliasing of data other than the choice of output resolution, and if it is chosen as too fine, many elements might end up being unassigned.

Reading and Writing Geospatial Data

Many vector and raster data formats have been developed for storing geospatial data in computer files. Some are widely used, others are obscure. Some are simple, while others are elaborate. Some formats are government or international standards, others are simply popular. A format can be general-purpose, specific to a narrow class of data, or may be used just to publish a certain data set.

The Mapping Toolbox includes both general-purpose data files for the Earth and its major regions as well as some demo data files covering small areas. Some geodata files are provided in internal formats (e.g., MAT-files) usable only with MATLAB, while others are in external formats commonly used for geospatial data exchange (e.g., shapefiles and GeoTIF files).

The following table lists data formats that the Mapping Toolbox reads and writes, along with the functions that handle them. For further information, see the Mapping Toolbox reference pages.

Function	Read	Write	Description
`arcgridread`	X		Read a gridded data set in Arc ASCII Grid Format.
`avhrrgoode`	X		Read Very High Resolution Radiometer (AVHRR) data stored in the Goode projection.
`avhrrlambert`	X		Read AVHRR data stored in the Lambert projection.
`dcwdata`	X		Read selected data from the Digital Chart of the World.
`dcwgaz`	X		Search for entries in the Digital Chart of the World gazette.
`dcwrdx`	X		Read the Digital Chart of the World index file.
`dcwread`	X		Read a Digital Chart of the World file.

Function	Read	Write	Description
dcwrhead	X		Read a Digital Chart of the World file header.
demdataui	X		Activate Digital Elevation Map Data User Interface.
dted	X		Read U. S. Department of Defense Digital Terrain Elevation Data (DTED) data.
dteds			List DTED data grid filenames.
egm96geoid	X		Read 15-minute gridded geoid heights from the EGM96 geoid model.
etopo5	X		Read data from the ETOPO5 data set.
fipsname	X		Read Topographically Integrated Geographic Encoding and Referencing (TIGER) thinned boundary file FIPS names.
geotiffinfo	X		Information about a GeoTIFF file.
geotiffread	X		Read a georeferenced image from GeoTIFF file
getworldfilename			Derive a worldfile name from an image filename.
globedem	X		Read Global Land One-km Base Elevation (GLOBE) 30-arc-second (1 km) Digital Elevation Map.
globedems			Read GLOBE 30-arc-second (1 km) Digital Elevation Map filenames.
gshhs	X		Read Global Self-consistent Hierarchical High-resolution Shoreline data.

Function	Read	Write	Description
gtopo30	X		Read GTOPO30 30-arc-second (1 km) global elevation data.
gtopo30s			List GTOPO30 30-arc-second (1 km) global elevation data filenames.
readfk5	X		Read data from the Fifth Fundamental Catalog of Stars.
satbath	X		Read Global 2-minute (4 km) topography from satellite bathymetry.
sdtsdemread	X		Read data from a SDTS DEM data set.
sdtsinfo	X		Information about a SDTS data set.
shapeinfo	X		Information about shapefile.
shaperead	X		Read geospatial data and associated attributes from a shapefile.
tbase	X		Read data from the TerrainBase dataset.
tgrline	X		Read data from TIGER/Line files.
tigermif	X		Read TIGER MapInfo Interchange Format thinned boundary files.
tigerp	X		Read TIGER ArcInfo Format thinned boundary files.
usgs24kdem	X		Read USGS 1:24,000 (30 m) digital elevation maps.
usgsdem	X		Read USGS 1:250,000 (100 m) digital elevation maps.
usgsdems	X		Read USGS digital elevation map filenames.
vmap0data	X		Extract selected data from the Vector Map Level 0 CD-ROMs.

Function	Read	Write	Description
vmapphead			Parse Vector Map Level 0 header string.
vmap0rdx	X		Read the Vector Map Level 0 index file.
vmap0read	X		Read Vector Map Level 0 file.
vmap0rhead	X		Read Vector Map Level 0 file headers.
vmap0ui			Activate Vector Map Level 0 User Interface.
worldfileread	X		Read a worldfile and return a referencing matrix.
worldfilewrite		X	Construct a worldfile from a referencing matrix.
imread	X		Read nongeoreferenced image from a graphics file (MATLAB function).
imwrite		X	Write nongeoreferenced image to a graphics file (MATLAB function).

3

Understanding Geospatial Geometry

Working with geospatial data involves geographic concepts (e.g., geographic and plane coordinates, spherical geometry) and geodetic concepts (such as ellipsoids and datums). This section explains, at a high level, some of the concepts that underlie geometric computations on spherical surfaces.

Spheres, Spheroids, and Geoids (p. 3-2)	Geodetic approaches to modeling the shapes of planets
Latitude and Longitude (p. 3-8)	Locating positions on spheres and spheroids
Datums (p. 3-10)	Establishing a reference system for coordinate data
Map Projections (p. 3-11)	Flattening the Earth for display and analysis
Great Circles, Rhumb Lines, and Small Circles (p. 3-13)	Three important types of curves on the surface of the sphere or spheroid
Angles and Directions on the Sphere and Spheroid (p. 3-18)	What an azimuth is, and how its meaning can vary
Planetary Almanac Data (p. 3-24)	Using the almanac function to set up spherical parameters for mapping calculations
Measuring Area of Spherical Quadrangles (p. 3-26)	Computing the intersection of a zone and a lune

See Chapter 2, “Understanding Map Data,” for information on how geographic phenomena are encoded and represented numerically, and how geodata is structured.

Spheres, Spheroids, and Geoids

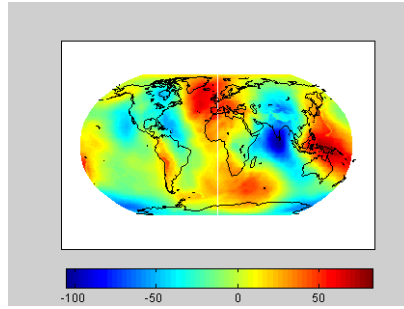
Although the Earth is very round, it is an oblate *spheroid* rather than a perfect sphere. This difference is so small (only one part in 300) that modeling the Earth as spherical is sufficient for making small-scale (world or continental) maps. However, making accurate maps at larger scale demands that a spheroidal model be used. Such models are essential, for example, when you are mapping high-resolution satellite or aerial imagery, or when you are working with coordinates from the Global Positioning System (GPS). This section addresses how the Mapping Toolbox accurately models the shape, or figure, of the Earth and other planets.

Geoid and Ellipsoid

Literally, *geoid* means *Earth-shaped*. The geoid is an empirical approximation of the figure of the Earth (minus topographic relief). Specifically, it is an equipotential surface with respect to gravity, more or less corresponding to mean sea level. It is approximately an oblate ellipsoid, but not exactly so because local variations in gravity create minor hills and dales (which range from -100 M to +60 M across the Earth).

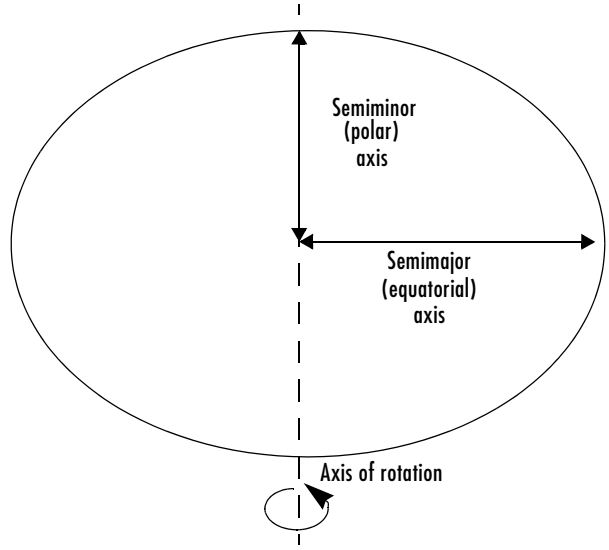
Mapping the Geoid. The following figure, made using the `geoid` data set, maps the figure of the Earth. To execute these commands, select them all by dragging over the list in the Help Browser, then click the right mouse button and choose `Evaluate Selection`:

```
clear;
load geoid; load coast
figure; axesm robinson
meshm(geoid,geoidlegend)
colorbar('horiz')
plotm(lat,long,'k')
```



The shape of the geoid is important for some purposes, such as calculating satellite orbits, but need not be taken into account for every mapping application. However, knowledge of the geoid is sometimes necessary, for example when you compare elevations given as height above mean sea level (a geoidal concept) to elevations derived from GPS measurements. Geoid representations are also inherent in datum definitions. See “Datums” on page 3-10.

When you are computing geospatial coordinates (e.g., for map projection), the geoid is generally treated as an *ellipsoid* (an ellipse rotated around one axis). You can deine ellipsoids in several ways. They are usually specified by a *semimajor* and a *semiminor axis*, but are often expressed in terms of a semimajor axis and either *inverse flattening* (which for the Earth, as mentioned above, is one part in 300) or *eccentricity*. Whatever parameters are used, the ellipsoid is fully constrained and the other parameters are derivable. The components of an ellipsoid are shown in the following diagram:



The Mapping Toolbox is equipped with ellipsoidal models that represent the figures of the Sun, Moon, and planets, as well as a set of the most common ellipsoid models of the Earth.

The Ellipsoid Vector

Ellipsoids in the Mapping Toolbox are most often represented as two-element vectors, called *ellipsoid vectors* in this guide. The ellipsoid vector has the form `[semimajor_axis eccentricity]`. The semimajor axis can be in any unit of distance; the choice of units typically drives the units used for distance outputs in the toolbox functions. Meters or kilometers are most frequently used. Bear in mind that some toolbox functions will calculate output based upon the semimajor axis units.

Eccentricity can range from 0 to 1. When only one element is provided, a spherical (0) eccentricity is assumed. The lack of an eccentricity value results in a spherical Earth assumption.

The default ellipsoid for the Earth is the 1980 Geodetic Reference System ellipsoid:

```
format long g
almanac('earth','ellipsoid','kilometers')
```

```
ans =
   1.0e+03 *
   6.37813700000000   0.00008181919104
```

Compare this is to a spherical ellipsoid definition:

```
almanac('earth','sphere','kilometers')
ans =
        6371           0
```

Note that you should set `format` to `long g`, as above, if you want MATLAB to list eccentricty values at full precision.

The `almanac` function treats the keyword `'geoid'` the same as `'ellipsoid'`.

Standard values for the ellipsoid vector, along with several other kinds of planetary data for each of the planets and the Earth's moon, are provided by the `almanac` function in the Mapping Toolbox (see "Planetary Almanac Data" on page 3-24). For example, examine the parameters of the `wgs72` (the 1972 World Geodetic System) ellipsoid, using the `almanac` function:

```
wgs72 = almanac('earth','wgs72')
wgs72 =
6378.135        0.0818188106627487
```

Compare this with Bessel's 1841 ellipsoid:

```
format long g
>> bessel = almanac('earth','bessel')
bessel =
               6377.397155        0.0816968312225275
```

The ellipsoid vector's values are the semimajor axis, in kilometers, and eccentricity. Both eccentricity and flattening are dimensionless ratios. The toolbox has functions to convert elliptical definitions from these forms to ellipsoid vector form. For example, the function `axes2ecc` returns an eccentricity when given semimajor and semiminor axes as arguments.

The ellipse in the previous diagram is highly exaggerated. For the Earth, the semimajor axis is about 21 kilometers longer than the semiminor axis. Use the `almanac` function to verify this:

```
grs80 = almanac('earth','ellipsoid','kilometers')
grs80 =
```

```
                  6378.137        0.0818191910428158
semiminor = minaxis(grs80)
semiminor =
           6356.75231414036
semidiff = grs80(1) - semiminor
semidiff =
          21.3846858596444
```

When compared to the semimajor axis, which is almost 6400 kilometers, this difference seems insignificant and can be neglected for world and other small-scale maps. For example, the scale at which 21.38 km would be smaller than a 0.5 mm line on a map (which is a typical line weight in cartography) is

```
nodiff = semidiff * 1e6 / 0.5
nodiff =
  4.2769e+007
```

The factor `1e6` simply converts the distance `semidiff` from kilometers into millimeters. This indicates that the earth's eccentricity cannot be portrayed at scales of less than 1:40,000,000, which is roughly the scale of a world map shown on a page of this document. For this reason, most functions in the Mapping Toolbox default to a spherical model of the Earth. However, you are free to specify any ellipsoid instead.

What Is the "Correct" Ellipsoid Vector?

A variety of reference ellipsoids have been proposed through the years. They differ because of the surveying information upon which they are based, or because they are intended to approximate the ellipsoid only within a specific geographic region. The Mapping Toolbox default ellipsoid vector (after the sphere) is based on the 1980 Geodetic Reference System ellipsoid. This ellipsoid vector is returned by the statement `almanac('earth','ellipsoid')`. It is also the reference ellipsoid for the 1984 World Geodetic System (WGS84).

In mapping a given spot on the Earth's surface, the choice of ellipsoid will affect the latitude assigned to that spot. Thus measurements from maps compiled using different ellipsoids cannot be accurately compared without converting to a common frame of reference. This also requires knowledge of the datum being used for the maps, as explained in "Datums" on page 3-10.

The Mapping Toolbox supports several other ellipsoid vectors, for models ranging from Everest's 1830 ellipsoid (used for India) to the International

Astronomical Union ellipsoid of 1965 (used for Australia). These can be referenced by the following statements:

```
ellipsoid1 = almanac('earth','ellipsoid',[],'everest');
ellipsoid2 = almanac('earth','ellipsoid',[],'iau65');
```

See the reference page for the `almanac` function for information on the ellipsoids that are built into the Mapping Toolbox. If you cannot find the ellipsoid vector you need, you can create it in the following form:

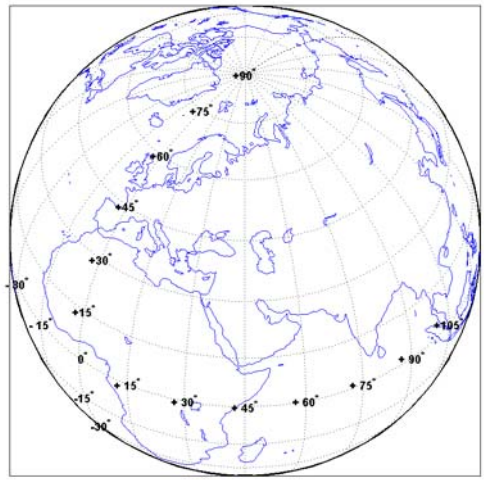
```
ellipsoidvec = [semimajor_axis eccentricity]
```

Note that the default units for the ellipsoid semimajor axis in the `almanac` function are kilometers, which you can use by simply passing in an empty matrix in place of the input units string (the third argument in the previous examples).

Latitude and Longitude

Two angles, *latitude* and *longitude*, specify the position of a point on the surface of a planet. These angles can be in degrees or radians; however, degrees are far more common in geographic notation.

Latitude is the angle between the plane of the equator and a line connecting the point in question to the planet's rotational axis. There are different ways to construct such lines, corresponding to different types of and resulting values for latitudes. Latitude is positive in the northern hemisphere, reaching a limit of +90° at the north pole, and negative in the southern hemisphere, reaching a limit of -90° at the south pole. Lines of constant latitude are called *parallels*. This system is depicted in the following figure.



Longitude is the angle at the center of the planet between two planes that align with and intersect along the axis of rotation, perpendicular to the plane of the equator. One plane passes through the surface point in question, and the other plane is the *prime meridian* (0° longitude), which is defined by the location of the Royal Observatory in Greenwich, England. Lines of constant longitude are called *meridians*. All meridians converge at the north and south poles (90°N and -90°S), and consequently longitude is under-specified in those two places.

Longitudes typically range from -180° to +180°, but can be represented otherwise, such as ranging from 0° to +360°. Longitudes can be specified in

other ways as well, such as from 0° to 180° east and 0° to 180° west. Adding or subtracting 360° from its longitude does not alter the position of a point.

Datums

A vertical *datum* (plural *datums*) is a base reference level for establishing the vertical dimension of elevation for the earth's surface. A datum can depend on the ellipsoid, the Earth model, or the definition of sea level. A coordinate system can be referenced to a datum or to an ellipsoid. A datum, however, always implies a specific ellipsoid.

As with ellipsoids, a datum can be defined globally or locally (e.g., particular to one country). While empirically determining a datum is a complex geodetic and surveying task, the result simply enables a map producer to know what the Earth's radius is at any given point. This is what ellipsoids also enable.

The datum used for a map (e.g, NAD27 or NAD83 for U.S. topographic sheets) must be known when you merge geospatial coordinate data compiled using different datums. This is because horizontal coordinates (both geographic and projected) shift when a new datum is applied. For example, locations of survey monuments in the U.S. can differ by 50 to 100 meters or more, depending on whether they were determined using the 1927 or 1983 North American Datum.

Map Projections

While all geospatial data needs to be georeferenced (pinned to locations on the Earth's surface) in some way, a given data set might or might not explicitly describe locations with geographic coordinates (latitudes and longitudes). When it does, many applications — particularly map display — cannot make direct use of geographic coordinates, and must transform them in some way to plane coordinates. This transformation process, called *map projection*, is both algorithmic and the core of the cartographer's art.

The Mapping Toolbox includes dozens of map projection functions. Some are ancient and well-known (such as Mercator), others are ancient and obscure (such as Bonne), while some are modern inventions (such as Robinson). Some are suitable for showing the entire world, others for half of it, and some are only useful over small areas. When geospatial data has geographic coordinates, any projection can be applied, although some are not good choices. The Mapping Toolbox can project both vector data and raster data.

Forward and Inverse Projection

When geospatial data has plane coordinates (i.e., it comes preprojected, as do many satellite images and municipal map data sets), it is usually possible to recover geographic coordinates if the projection parameters and datum are known. Using this information, you can perform an *inverse projection*, running the projection backward to solve for latitude and longitude. The Mapping Toolbox can perform accurate inverse projections for any of its projection functions as long as the original projection parameters and reference ellipsoid (or spherical radius) are provided to it.

Projection Distortions

All map projections introduce distortions compared to maps on globes. Distortions are inherent in flattening the sphere, and can take several forms:

- Areas — Relative size of objects (such as continents)
- Distances — Relative separations of points (such as a set of cities)
- Directions — Azimuths (angles between points and the poles)
- Shapes — Relative lengths and angles of intersection

Some classes of map projections maintain areas, and others preserve local shapes, distances, and/or directions. No projection, however, can preserve all these characteristics. Choosing a projection thus always requires compromising accuracy in some way, and that is one reason why so many different map projections have been developed. For any given projection, however, the smaller the area being mapped, the less distortion it introduces if properly centered. The Mapping Toolbox provides tools to help quantify and visualize projection distortions.

See "Using Map Projections and Coordinate Systems" on page 8-1 for a full discussion of map projections and how the Mapping Toolbox implements them. The "Summary and Guide to Projections" on page 8-53 lists all the available map projections and their intrinsic properties.

Great Circles, Rhumb Lines, and Small Circles

In plane geometry, lines have two important characteristics. A line represents the shortest path between two points, and the slope of such a line is constant. When describing lines on the surface of a spheroid, however, only one of these characteristics can be guaranteed at a time.

Great Circles

A *great circle* is the shortest path between two points along the surface of a sphere. The precise definition of a great circle is the intersection of the surface with a plane passing through the center of the planet. Thus, great circles always bisect the sphere. The equator and all meridians are great circles. All great circles other than these do not have a constant azimuth, the spherical analog of slope; they cross successive meridians at different angles. That great circles are the shortest path between points is not always apparent from maps, because very few map projections (the Gnomonic is one of them) represent arbitrary great circles as straight lines.

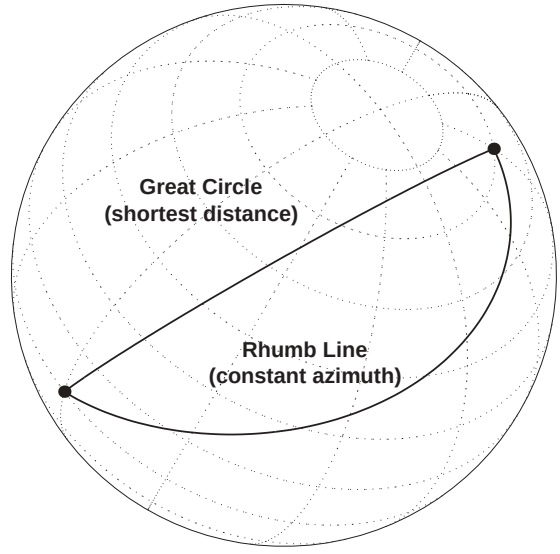
Because they define paths that minimize distance between two (or three) points, great circles are examples of *geodesics*. In general, a geodesic is the straightest possible path constrained to lie on a curved surface, independent of the choice of a coordinate system. The term comes from the Greek *geo-*, earth, plus *daiesthai*, to divide, which is also the root word of *geodesy*, the science of describing the size and shape of the Earth mathematically.

Rhumb Lines

A *rhumb line* is a curve that crosses each meridian at the same angle. This curve is also referred to as a *loxodrome* (from the Greek *loxos*, slanted, and *drome*, path). Although a great circle is a shortest path, it is difficult to navigate because your bearing (or *azimuth*) continuously changes as you proceed. Following a rhumb line covers more distance than following a geodesic, but it is easier to navigate.

All parallels, including the equator, are rhumb lines, since they cross all meridians at 90°. Additionally, all meridians are rhumb lines, in addition to being great circles. A rhumb line always spirals toward one of the poles, unless its azimuth is true east, west, north, or south, in which case the rhumb line closes on itself to form a parallel of latitude (small circle) or a pair of antipodal meridians.

The following figure depicts a great circle and one possible rhumb line connecting two distant locations. Descriptions and examples of how to calculate points along great circles and rhumb lines appear below.



Small Circles

In addition to rhumb lines and great circles, one other smooth curve is significant in geography and the Mapping Toolbox: the *small circle*. Parallels of latitude are all small circles (which also happen to be rhumb lines). The general definition of a small circle is the intersection of a plane with the surface of a sphere. On ellipsoids, this only yields true small circles when the defining plane is parallel to the equator. In the Mapping Toolbox, this definition includes planes passing through the center of the planet, so the set of all small circles includes all great circles as limiting cases. This usage is not universal.

Small circles are most easily defined by distance from a point. *All points 45 nm (nautical miles) distant from (45°N,60°E)* would be the description of one small circle. If degrees of arc length are used as a distance measurement, then (on a sphere) a great circle is the set of all points 90° distant from a particular *center* point.

For true small circles, the distance must be defined in a great circle sense, the shortest distance between two points on the surface of a sphere. However, the Mapping Toolbox also can calculate *loxodromic small circles*, for which distances are measured in a rhumb line sense (along lines of constant azimuth). Do not confuse such figures with true small circles.

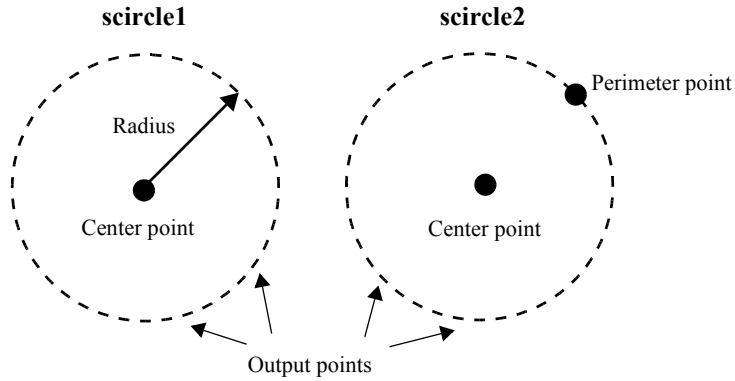
Computing Small Circles

You can calculate vector data for points along a small circle in two ways. If you have a center point and a known radius, use `scircle1`; if you have a center point and a single point along the circumference of the small circle, use `scircle2`. For example, to get data points describing the small circle at 10° distance from (67°N, 135°W), use the following:

```
[latc,lonc] = scircle1(67,-135,10);
```

To get the small circle centered at the same point that passes through the point (55°N,135°W), use `scircle2`:

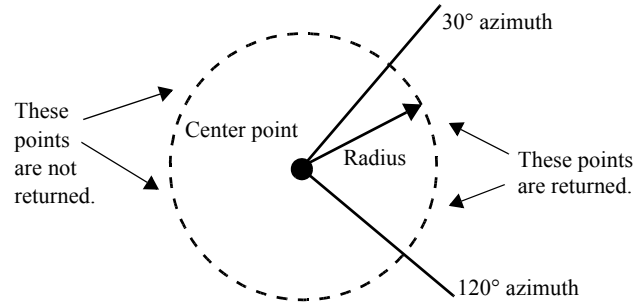
```
[latc,lonc] = scircle2(67,-135,55,-135);
```



The `scircle1` function also allows you to calculate points along a specific arc of the small circle. For example, if you want to know the points 10° in distance and between 30° and 120° in azimuth from (67°N,135°W), simply provide arc limits:

```
[latc,lonc] = scircle1(67,-154,10,[30 120]);
```

scircle1 with arc limits



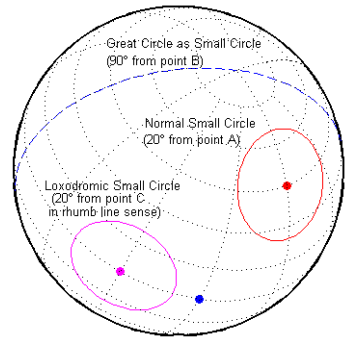
When an entire small circle is calculated, the data is in polygon format. For all calculated small circles, 100 points are returned unless otherwise specified. You can calculate several small circles at once by providing vector inputs. For more information, see the `scircle1` and `scircle2` functions in the online Mapping Toolbox reference documentation.

An Annotated Map Illustrating Small Circles. The following Mapping Toolbox commands illustrate generating small circles of the types described above, including the limiting case of a large circle. To execute these commands, select them all by dragging over the list in the Help Browser, then click the right mouse button and choose `Evaluate Selection`:

```
figure;
axesm ortho; gridm on; framem on
setm(gca,'Origin', [45 30 30], 'MLineLimit', [75 -75],...
'MLineException',[0 90 180 270])
A = [45 90];
B = [0 60];
C = [0 30];
sca = scircle1(A(1), A(2), 20);
scb = scircle2(B(1), B(2), 0, 150);
scc = scircle1('rh',C(1), C(2), 20);
plotm(A(1), A(2),'ro','MarkerFaceColor','r')
plotm(B(1), B(2),'bo','MarkerFaceColor','b')
plotm(C(1), C(2),'mo','MarkerFaceColor','m')
plotm(sca(:,1), sca(:,2),'r')
plotm(scb(:,1), scb(:,2),'b--')
```

```
plotm(scc(:,1), scc(:,2),'m')
textm(50,0,'Normal Small Circle')
textm(46,6,'(20\circ from point A)')
textm(4.5,-10,'Loxodromic Small Circle')
textm(4,-6,'(20\circ from point C')
textm(-2,-4,'in rhumb line sense)')
textm(40,-60,'Great Circle as Small Circle')
textm(45,-50,'(90\circ from point B)')
```

The result is the following display:



Angles and Directions on the Sphere and Spheroid

Azimuth is the angle a line makes with a meridian, measured clockwise from north. Thus the azimuth of due north is 0°, due east is 90°, due south is 180°, and due west is 270°. You can instruct the Mapping Toolbox to compute azimuths for any pair of point locations, either along rhumb lines or along great circles. These will have different results except along cardinal directions. For great circles, the result is the azimuth at the initial point of the pair defining a great circle path. This is because great circle azimuths other than 0°, 90°, 180°, and 270° do not remain constant. Azimuths for rhumb lines are constant along their entire path (by definition).

For rhumb lines, computing an azimuth backward (from the second point to the first) yields the complement of the forward azimuth ((Az + 180°) mod 360°). For great circles, the back azimuth is generally not the complement, and the difference depends on the distance between the two points.

In addition to forward and back azimuths, the Mapping Toolbox can compute locations of points a given distance and azimuth from a reference point, and can calculate tracks to connect waypoints, along either great circles or rhumb lines on a sphere or ellipsoid.

Reckoning — the Forward Problem

A common problem in geographic applications is the determination of a destination given a starting point, an initial azimuth, and a distance. In the Mapping Toolbox, this process is called *reckoning*. A new position can be reckoned in a great circle or a rhumb line sense (great circle or rhumb line track).

As an example, an airplane takes off from La Guardia Airport in New York (40.75°N, 73.9°W) and follows a northwestern rhumb line flight path at 200 knots (nautical miles per hour). Where would it be after 1 hour?

```
[rhlat,rhlong] = reckon('rh',40.75,-73.9,nm2deg(200),315)
rhlat =
   43.1054
rhlong =
   -77.0665
```

Notice that the distance, 200 nautical miles, must be converted to degrees of arc length with the `nm2deg` conversion function to match the latitude and

longitude inputs. If the airplane had a flight computer that allowed it to follow an exact great circle path, what would the aircraft's new location be?

```
[gclat,gclong] = reckon('gc',40.75,-73.9,nm2deg(200),315)
gclat =
   43.0615
gclong =
   -77.1238
```

Notice also that for short distances at these latitudes, the result hardly differs between great circle and rhumb line. The two destination points are less than 4 nautical miles apart. Incidentally, after 1 hour, the airplane would be just north of New York's Finger Lakes.

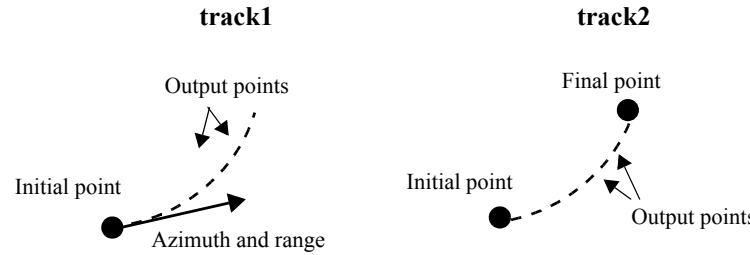
Calculating Tracks – Great Circles and Rhumb Lines

You can generate vector data corresponding to points along great circle or rhumb line tracks using `track1` and `track2`. If you have a point on the track and an azimuth at that point, use `track1`. If you have two points on the track, use `track2`. For example, to get the great circle path starting at (31°S, 90°E) with an azimuth of 45° with a length of 12°, use `track1`:

```
[latgc,longc] = track1('gc',-31,90,45,12);
```

For the great circle from (31°S, 90°E) to (23°S, 110°E), use `track2`:

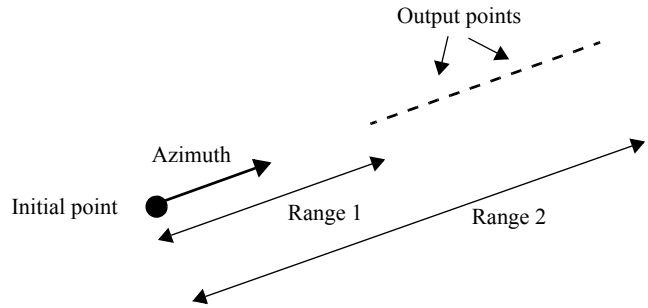
```
[latgc,longc] = track2('gc',-31,90,-23,110);
```



The `track1` function also allows you to specify range endpoints. For example, if you want points along a rhumb line starting 5° away from the initial point and ending 13° away, at an azimuth of 55°, simply specify the range limits:

```
[latrh,lonrh] = track1('rh',-31,90,55,[5 13]);
```

track1 with range limits



When no range is provided for `track1`, the returned points represent a *complete track*. For great circles, a complete track is 360°, encircling the planet and returning to the initial point. For rhumb lines, the complete track terminates at the poles, unless the azimuth is 90° or 270°, in which case the complete track is a parallel that returns to the initial point.

For calculated tracks, 100 points are returned unless otherwise specified. You can calculate several tracks at one time by providing vector inputs. For more information, see the `track1` and `track2` functions in the online Mapping Toolbox reference documentation. More vector path calculations are described later in "Navigation" on page 9-11.

Distance, Azimuth, and Back-azimuth (the Inverse Problem)

When you calculate the distance between two points with the Mapping Toolbox, the result depends upon whether you want a great circle or a rhumb line distance. The `distance` function returns the appropriate distance between two points as an angular arc length, employing the same angular units as the input latitudes and longitudes. The default path type is the shorter great circle, and the default angular units are degrees. The previous figure shows two points at (15°S, 0°) and (60°N, 150°E). The great circle distance between them, in degrees of arc, is as follows:

```
distgc = distance(-15,0,60,150)
distgc =
   129.9712
```

The rhumb line distance is greater:

```
distrh = distance('rh',-15,0,60,150)
distrh =
   145.0288
```

To determine how much longer the rhumb line path is in, say, kilometers, you can use a distance conversion function on the difference:

```
kmdifference = deg2km(distrh—distgc)
kmdifference =
   1.6744e+03
```

Several distance conversion functions are available in the toolbox, supporting degrees, radians, kilometers, meters, statute miles, nautical miles, and feet. Converting distances between angular arc length units and surface length units requires the radius of a planet or spheroid. By default, the radius of the Earth is used.

Calculating Azimuth and Elevation

Azimuth is the angle a line makes with a meridian, taken clockwise from north. When the azimuth is calculated from one point to another using the Mapping Toolbox, the result depends upon whether you want a great circle or a rhumb line azimuth. For great circles, the result is the azimuth at the starting point of the connecting great circle path. In general, the azimuth along a great circle is not constant. For rhumb lines, the resulting azimuth is constant along the entire path.

Azimuths, or bearings, are returned in the same angular units as the input latitudes and longitudes. The default path type is the shorter great circle, and the default angular units are degrees. In the example, the great circle azimuth from the first point to the second is

```
azgc = azimuth(-15,0,60,150)
azgc =
   19.0391
```

For the rhumb line, the constant azimuth is

```
azrh = azimuth('rh',-15,0,60,150)
azrh =
   58.8595
```

One feature of rhumb lines is that the inverse azimuth, from the second point to the first, is the complement of the forward azimuth and can be calculated by simply adding 180° to the forward value:

```
inverserh = azimuth('rh',60,150,-15,0)
inverserh =
   238.8595

difference = inverserh-azrh
difference =
   180
```

This is not true, in general, of great circles:

```
inversegc = azimuth('gc',60,150,-15,0)
inversegc =
   320.9353

difference = inversegc-azgc
difference =
   301.8962
```

The azimuths associated with cardinal and inter-cardinal compass directions are the following:

North	0° or 360°
Northeast	45°
East	90°
Southeast	135°
South	180°
Southwest	225°
West	270°
Northwest	315°

Elevation is the angle above the local horizontal of one point relative to the other. To compute the elevation angle of a second point as viewed from the first,

provide the position and altitude of the points. The default units are degrees for latitudes and longitudes and meters for altitudes, but you can specify other units for each. What are the elevation and slant range of a point 10 kilometers east and 10 kilometers above a surface point?

```
[elevang,slantrange] = elevation(0,0,0, 0,km2deg(10),10000)

elevang =

       44.901


slantrange =

        14156
```

The answer is slightly different from that expected from plane geometry because of the curvature of the Earth.

Planetary Almanac Data

The Mapping Toolbox contains a function that provides almanac data on the major bodies of our solar system. Basic geometric parameters, such as ellipsoid vectors, radii, surface areas, and volumes, can be accessed for the Sun, the Earth's moon, and all of the planets, in any of the supported units of distance measurement.

Many planets have ellipsoid vectors available. Some planets return spherical ellipsoid vectors only:

```
almanac('earth','ellipsoid','nauticalmiles')
ans =
   3443.92          0.08

almanac('mars','ellipsoid','kilometers')
ans =
   3396.90          0.11

almanac('moon','ellipsoid','statutemiles')
ans =
   1079.97             0
```

When you specify `'radius'` a scalar is returned representing the radius of the best spherical model of the planet. Notice that for a spherical model, the radius in radians is 1:

```
almanac('mercury','radius','kilometers')
ans =
   2439

almanac('neptune','radius','radians')
ans =
   1
```

Surface areas and volumes are calculated based on a spherical model by default. In most cases, you can use the ellipsoid model instead, and for the Earth you can specify any of the supported ellipsoid models. You can also request the actual tabulated values of the Earth:

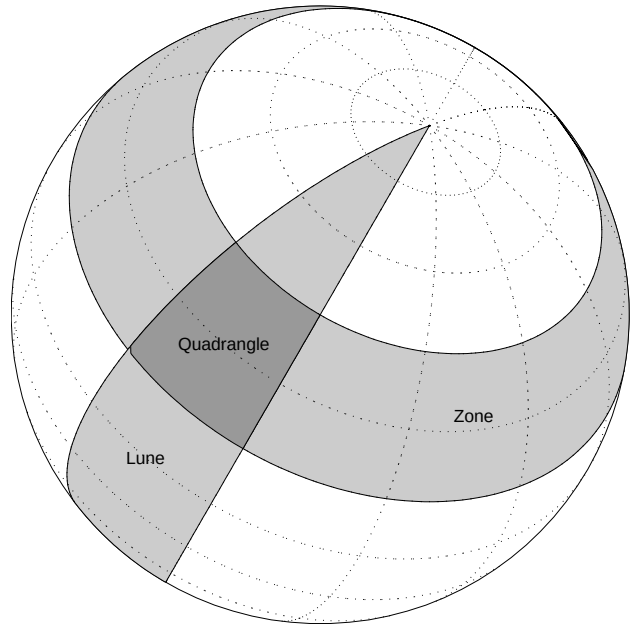
```
almanac('mars','surfarea','kilometers','ellipsoid')
ans =
   1.4441e+08
```

```
almanac('earth','volume','kilometers','international')
ans =
   1.0833e+12

almanac('earth','volume','kilometers','actual')
ans =
   1.0832e+12
```

For a complete description of available data, see the `almanac` function in the online Mapping Toolbox reference documentation.

Measuring Area of Spherical Quadrangles

In solid geometry, the area of a spherical quadrangle can be exactly calculated. A spherical quadrangle is the intersection of a *lune* and a *zone*. In geographic terms, a *quadrangle* is defined as a region bounded by parallels north and south, and meridians east and west.



In the pictured example, a quadrangle is formed by the intersection of a zone, which is the region bounded by 15°N and 45°N latitudes, and a lune, which is the region bounded by 0° and 30°E longitude. Under the spherical planet assumption, the fraction of the entire spherical surface area inscribed in the quadrangle can be calculated:

```
area = areaquad(15,0,45,30)
area =
   0.0187
```

That is, less than 2% of the planet's surface area is in this quadrangle. To get an absolute figure in, for example, square miles, you must provide the appropriate spherical radius. The radius of the Earth is about 3958.9 miles:

```
area = areaquad(15,0,45,30,3958.9)
area =
       3.6788e+06
```

The surface area within this quadrangle is over 3.6 million square miles for a spherical Earth.

4

Creating and Viewing Maps

The Mapping Toolbox provides many ways to control displays of geospatial data. This chapter provides an overview of the most important functions and associated interfaces for displaying and interacting with vector and raster geodata.

Introduction to Mapping Graphics

Even though geospatial data often is manipulated and analyzed without being displayed, high-quality interactive cartographic displays can play valuable roles in exploratory data analysis, application development, and presentation of results.

With the Mapping Toolbox, you can display geographic information almost as easily as you can plot tabular or time-series data in MATLAB. Most mapping functions are similar to MATLAB plotting functions, except they accept data with geographic/geodetic coordinates (latitudes and longitudes), instead of Cartesian and polar coordinates. Mapping functions typically have the same names as their MATLAB counterparts, with the addition of an `'m'` suffix (for maps). For example, the Mapping Toolbox analog to the MATLAB `plot` function is `plotm`.

The Mapping Toolbox manages most of the details in displaying a map. It projects your data, cuts and trims it to specified limits, and displays the resulting map at various scales. With the toolbox you can also add customary cartographic elements, such as a frame, gridlines, coordinate labels, and text labels, to your displayed map. If you change your projection properties, or even the projection itself, the Mapping Toolbox redraws the map with the new settings, undoing any cuts or trims if necessary. See “Accessing, Computing, and Inverting Map Projection Data” on page 8-31, for information on how to project data without displaying it.

The toolbox also makes it easy to modify and manipulate maps. You can modify the map display and mapped objects either from the command line or through graphical user interfaces and property editing tools you can invoke by clicking on the display. Most mapping display functions have graphical user interfaces. See the “GUI Reference ” chapter for more on these capabilities.

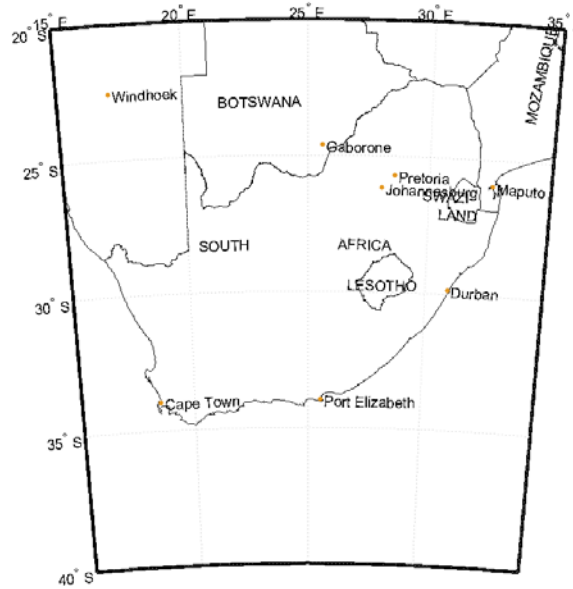
Note The Mapping Toolbox manages the map display with the `UserData` property field in the Axes structure. The Toolbox also uses the `UserData` property of mapped objects. This can cause conflicts with other functions that use the `UserData` property field, restricting their use.

Simple Map Displays Using worldmap and usamap

The Mapping Toolbox enables you to control many aspects of a map display. Some functions accept many parameters, but nearly all of them have useful defaults. Some functions, for example, the `worldmap` and `usamap` functions, trade control for simplicity. These functions create maps of regions of the world or the United States by automatically selecting appropriate map projections and settings. You can then modify or add to the map displays using the methods described later in this chapter. You can also use these functions to prepare a map display using your own data rather than the built-in atlas data.

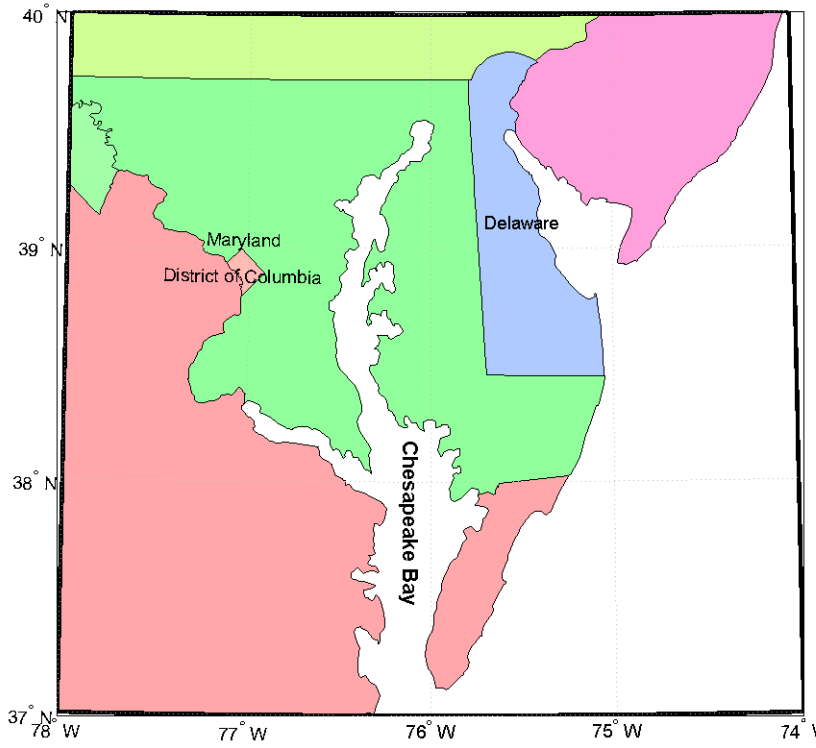
Here are two examples that create political maps. The first one creates a base map of South Africa.

```
figure
worldmap 'south africa'
hidem(gca)
```



The second example creates a map of the Chesapeake Bay region by specifying geographic limits. It also places text annotation:

```
latlim = [37 40]; lonlim = [-78 -74];
figure
usamap(latlim,lonlim)
textm(38.2,-76.1,' Chesapeake Bay ',...
'fontweight','bold','Rotation', 270)
```



For further information on options for these functions see the reference pages for `worldmap` and `usamap`.

Axes for Drawing Maps

When you create a map, you can use one of the Mapping Toolbox's built-in user interfaces (UIs), or you can build the graphic with MATLAB and Mapping Toolbox functions. Many MATLAB graphics are built using the `axes` function:

```
axes
axes('PropertyName',PropertyValue,...)
axes(h)
h = axes(...)
```

The Mapping Toolbox provides an extended version of `axes`, called `axesm`, that includes information about the current coordinate system (map projection). Its syntax is similar:

```
axesm
axesm(handle)
axesm(PropertyName,PropertyValue,...)
axesm(ProjectionFile,PropertyName,PropertyValue,...)
```

The `axesm` function without arguments brings up a UI that lists all supported projections and assists in defining their parameters. You can also summon this UI with the `axesmui` function.

You can also list all the names, classes, and ID strings of Mapping Toolbox map projections with the `maps` function.

Axes created with `axesm` share all properties associated with regular axes, and have additional properties related to projections, scale, and positioning in geographic coordinates. See the `axes` and `axesm` reference pages for lists of properties.

Map axes created by `axesm` contain projection information in a structure accessed by their `UserData` property. For an example of what these properties are, type

```
h = axesm('MapProjection','mercator')
```

and then use the `getm` function to retrieve all the map axes properties:

```
p = getm(h)
```

As the projection data is stored in the `UserData` fields of the axes structure, you can also access it via the general axes properties:

```
q = get(h, 'UserData')
```

Using axesm

The figure window created using `axesm` contains the same set of tools and menus as any MATLAB figure, and is by default blank, even if there is map data in your workspace. You can toggle certain properties, such as grids, frames, and axis labels by right-clicking in the figure window to obtain a pop-up menu.

You can define multiple independent map axes figures, but only one can be active at any one time. Return handles for them when you create them to allow them to be referenced when they are no longer current. Use the `axesm(handle)` syntax to activate an existing map axes figure.

Accessing and Manipulating Map Axes Properties

Just as the properties of the underlying standard axes can be accessed and manipulated using the MATLAB functions `set` and `get`, map axes properties can also be accessed and manipulated using the functions `setm` and `getm`.

Note Use the `axesm` function only to *create* a map axes. Use the `setm` function to *modify* existing map axes.

---

1. As an example, create a map axes containing no map data:

   ```
   axesm('MapProjection','miller','Frame','on')
   ```

   Note that you specify `MapProjection` string values in lowercase. 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`.

2. Access the current `FLineWidth` property value by typing

   ```
   getm(gca,'FLineWidth')
   ans =
      2
   ```

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

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

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,'MapProjection','eqdcylin')

getm(gca,'FLineWidth')
ans =
   3

getm(gca,'MapProjection')
ans =
eqdcylin
```

5. To inspect the entire set of map axes properties at their current settings, use the following command:

```
getm(gca)
ans =
     mapprojection: 'eqdcylin'
              zone: []
        angleunits: 'degrees'
            aspect: 'normal'
       falseeasting: []
      falsenorthing: []
        fixedorient: []
              geoid: [1 0]
       maplatlimit: [-90 90]
       maplonlimit: [-180 180]
      mapparallels: 30
        nparallels: 1
            origin: [0 0 0]
       scalefactor: []
           trimlat: [-90 90]
           trimlon: [-180 180]
             frame: 'on'
```

```
          ffill: 100
      fedgecolor: [0 0 0]
      ffacecolor: 'none'
       flatlimit: [-90 90]
      flinewidth: 3
       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: 9
      fontunits: 'points'
     fontweight: 'normal'
    labelformat: 'compass'
     labelunits: 'degrees'
  meridianlabel: 'off'
 mlabellocation: 30
 mlabelparallel: 90
    mlabelround: 0
  parallellabel: 'off'
 plabellocation: 15
 plabelmeridian: -180
    plabelround: 0
```

Note that the list of properties includes both those particular to map axes and general ones that apply to all MATLAB axes.

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

```
setm(gca)
AngleUnits               [ {degrees} | radians | dms | dm ]
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 | dms | dm ]
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,'AngleUnits')
AngleUnits                   [ {degrees} | radians | dms | dm ]
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
```

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

```
getm(gca,'AngleUnits')
ans =
degrees
```

```
getm(gca,'MapProjection')
ans =
eqdconic

getm(gca,'Frame')
ans =
on

getm(gca,'FixedOrient')
ans =
     []
```

For a complete listing and descriptions of map axes properties, see the reference page for `axesm`. To identify what properties apply to a given map projection, see the reference page for that projection.

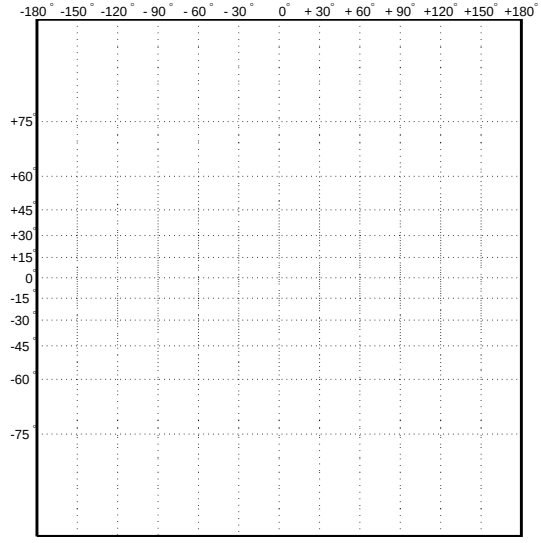
Switching Between Projections

Once map axes have been created with `axesm`, whether map data is displayed or not, it is possible to change the current projection as well as many of its parameters. You can use `setm` or the `maptool` UI to redefine the projection. If you do so, you might need to change some of the map axes properties to achieve proper appearance. Settings that are suitable for one projection might not be appropriate for another. Some projections have default properties that define *that* particular projection and cannot be altered; for example, the Balthasart cylindrical projection is defined to have standard parallels (`MapParallels`) at 50°. Other projections have default properties that are initially set for proper world display; for example, the Mercator projection limits the latitude range to ±86° to avoid “blowing up” at the poles.

Although similar projections can share the same set of properties (Miller cylindrical and plate carree cylindrical), others can be drastically different (polyconic and stereographic azimuthal). The classification of map projections is often a good indicator of whether changes need to be made. For instance, switching from a cylindrical to an azimuthal projection requires a few modifications, as the following examples indicate:

1 Create a Mercator projection with meridian and parallel labels:

```
axesm mercator
framem on; gridm on; mlabel on; plabel on
setm(gca,'LabelFormat','signed')
```



2 Get the default map and frame latitude limits for the Mercator projection:

```
[getm(gca,'MapLatLimit'); getm(gca,'FLatLimit')]
ans =
-86    86
-86    86
```

Both the frame and map latitude limits are set to 86° north and south for the Mercator projection to maintain a safe distance from the singularity at the poles.

3 Now switch the projection to an orthographic azimuthal:

```
setm(gca,'MapProjection','ortho')
```

What happened to the map frame and labels? If you recall, the frame latitude limits have not been changed and still correspond to the default values for a Mercator projection, as do all the other properties.

4 Only those properties that are required to have values are updated for the current projection. Among those that need not be are the latitude and longitude limits. Use `getm` to see their settings:

```
getm(gca,'FLatLimit')
ans =
-86    86
```

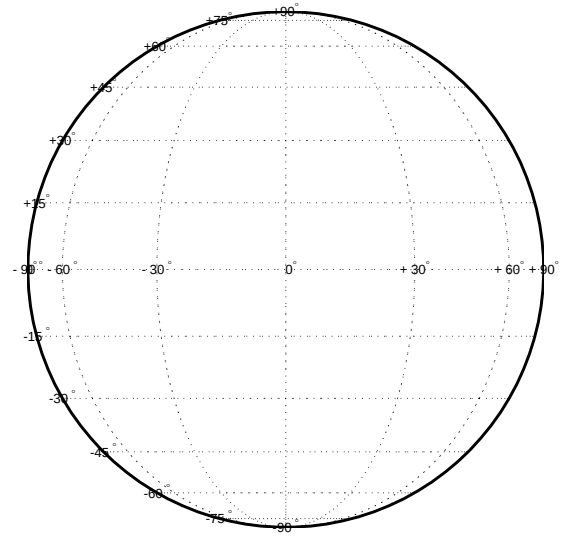
5 You must manually reset the frame and map limits to appropriate values for an orthographic projection so that the circular frame is displayed. If you don't know the default or appropriate numeric values, provide an empty matrix for any of the property values:

```
setm(gca,'FLatLimit',[],'MapLatLimit',[])
[getm(gca,'MapLatLimit'); getm(gca,'FLatLimit')]
ans =
   -90    90
   -Inf 89
```

6 You also need to manually specify the locations of the meridian and parallel labels (see "Labeling Grids" on page 4-25):

```
setm(gca,'MLabelParallel',0,'PLabelMeridian',-90)
```

Now the map is displayed correctly, with the frame:



You can reset default property values to new values by specifying empty matrices, as shown in the last example. You can reset the entire set of properties to default values by using the **Reset** button on the `axesmui` GUI.

For complete descriptions of all map axes properties, see the `axesm` reference page. For more information on the use of `axesmui`, refer to the `axesm, axesmui` reference page.

Projected and Unprojected Graphic Objects

Many graphic functions in the Mapping Toolbox functions project objects on a map axes based on their designated latitude-longitude positions. The latitudes and longitudes are mathematically transformed to x and y positions using the formulas for the current map projection. If the projection changes for a map axes (for example, if you use the `setm` function to alter the `MapProjection` property), these objects are reprojected into new positions. Mapping Toolbox functions with this property include the following:

- `contourm`
- `contour3m`
- `fillm`
- `fill3m`
- `gridm`
- `linem`
- `patchm`
- `plotm`
- `plot3m`
- `surfm`
- `surfacem`
- `textm`

Each of these functions is analogous to a standard MATLAB graphics function, which has the same name minus the trailing `m`. Both types of functions can be used on a map axes, as long as you are aware that the standard MATLAB graphics functions do not apply map projection transformations, and therefore require positions to be specified in Cartesian axes space.

If you have pre-projected vector or raster map data, you can display it with standard MATLAB graphics functions. If its projection is known and is one built into the Mapping Toolbox, you can use its parameters to project geodata in geographic coordinates to display it in the same axes. For additional

information, see “Using Cartesian MATLAB Display Functions” on page 6-18. You can also display projected geodata using `mapview` and `mapshow`.

Placing Geographic and Non-geographic Objects in a Map Axes

Here is an example of how the two types of functions can interact when you place text objects:

1. Make a Miller map axes and grid:

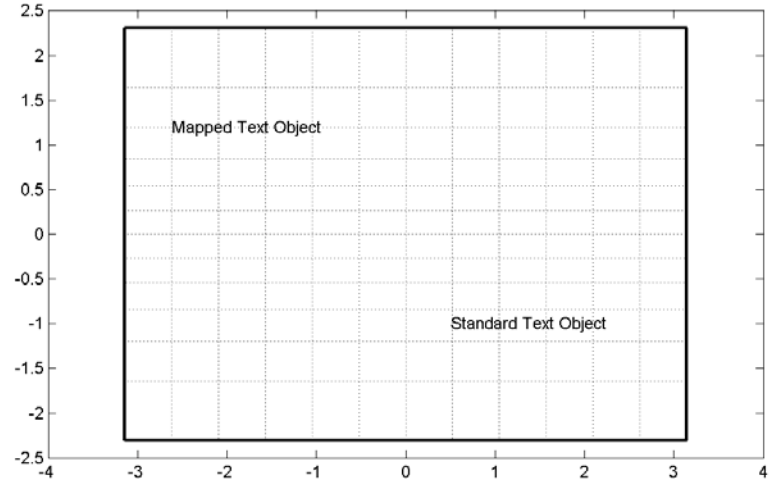
   ```
   axesm miller; framem on; gridm on;
   showaxes; grid off;
   ```

   These two functions create a map axes, a map frame enclosing the region of interest, and geographic grid lines. The *x-y* axes, which are normally hidden, are displayed, and the MATLAB `x-y` grid is turned off. Note that the Mapping Toolbox function `gridm` behaves differently than the MATLAB *x-y* `grid` function.

2. Now place a standard MATLAB text object and a mapped text object, using the two separate coordinate systems:

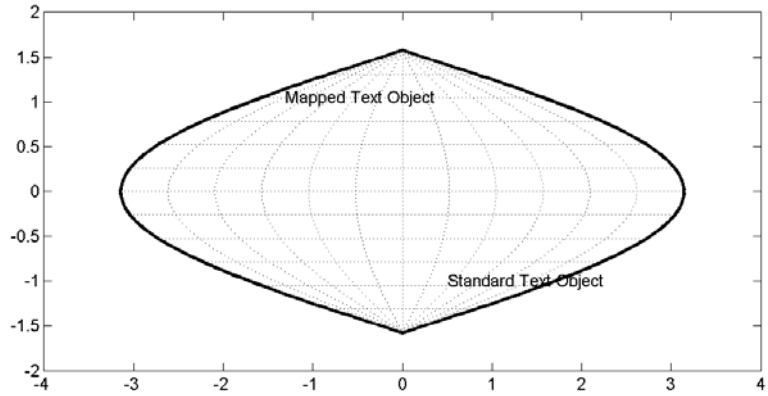
   ```
   text(.5,-1,'Standard Text Object')
   textm(60,-150,'Mapped Text Object')
   ```

   In the figure, shown below, a standard text object is placed at x=0.5 and y=-1, while a mapped text object has been placed at (60°N,150°W) in the Miller projection.



3 Now change the projection to sinusoidal. The MATLAB text object remains at the same Cartesian position, which alters its latitude-longitude position. The mapped text object remains at the same geographic location, so its *x*-*y* position is altered. Also, the frame and gridlines reflect the new map projection:

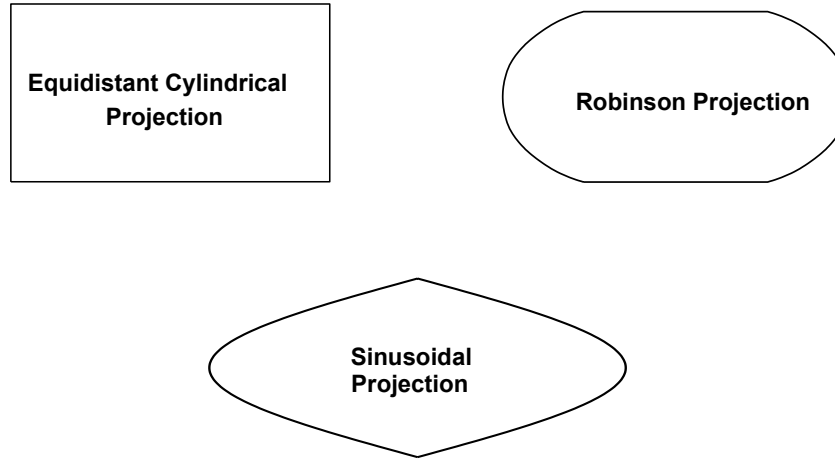
```
setm(gca,'MapProjection','sinusoid')
showaxes; grid off;
```



Similarly, vector and matrix data can be displayed using either mapping or standard functions (e.g., `plot`/`plotm`, `surf`/`surfm`). See “Displaying Vector Data with Mapping Toolbox Functions” on page 4-27 for information on plotting vector geodata, and “Displaying Data Grids” on page 4-34 for information on plotting raster geodata.

The Map Frame

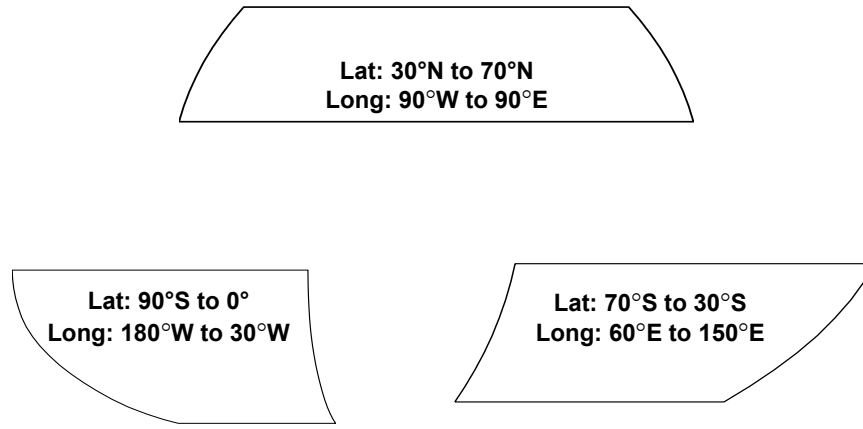
In the Mapping Toolbox, the *map frame* is the outline of the limits of a map, often in the form of a *box*, the "edge of the world," so to speak. The frame is displayed if the map axes property `Frame` is set to `'on'`. This can be accomplished upon map axes creation with `axesm`, or later with `setm`, or with the direct command `framem on`. The frame is geographically defined as a latitude-longitude quadrangle that is projected appropriately. For example, on a map of the world, the frame might extend from pole to pole and a full 360° range of longitude. In appearance, the frame would take on the characteristic shape of the projection. The examples below are full-world frames shown in three very different projections:



Full-World Map Frames

As a map object, each of the previously displayed frames is identical; however, the selection of a display projection has varied their appearance. Since each of the examples shows the entire world, `FLatLimit` is `[-90 90]`, and `FLonLimit` is `[-180 180]` for each case. The frame quadrangle can encompass smaller

regions, as well, in which case the shape is a section of a full-world outline or simply a quadrilateral with straight or curving sides:



Frame Quadrangles Shown in the Robinson Projection (symmetric about Prime Meridian)

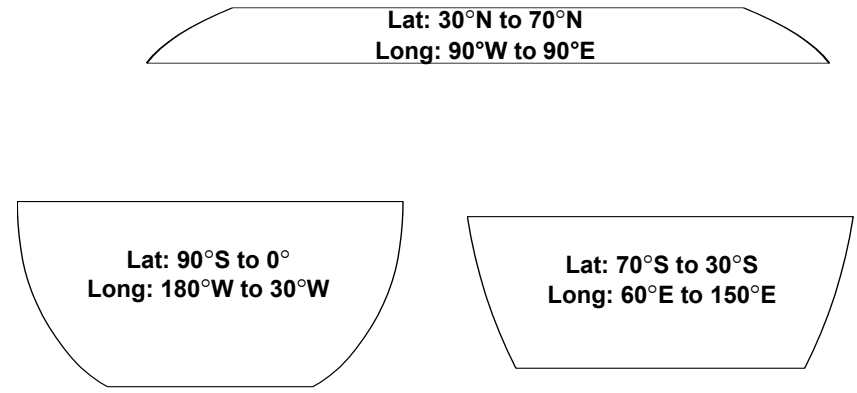
For the frames shown above, the projection is centered on the prime meridian, or 0 longitude. Such a frame would be the result of creating a map axes with the defaults for the Robinson projection and then resetting the frame limits to cover just part of the world.

For example, to view the asymmetric frame in the lower right of the previous figure, type

```
axesm robinson
setm(gca,'FLatLimit',[-70 -30],...
         'FLonLimit',[60 150],...
         'Frame','on')
```

Note that map axes properties that concern frames begin with `'F'`.

When you want your frame to be symmetric about the region of interest, let `axesm` determine the proper settings for you. If you specify the map limits without specifying the map origin and frame limits, `axesm` will automatically set the appropriate values for a proper symmetric frame.



Frame Quadrangles Shown in the Robinson Projection (Symmetric about Map Limits)

For example, to view the symmetric frame in the lower right of the above figure, set the map limits with `axesm`:

```
axesm('MapProjection','robinson',...
      'MapLatLimit',[-70 -30],...
      'MapLonLimit',[60 150],...
      'Frame','on')
```

You can manipulate properties beyond the latitude and longitude limits of the frame. Frame properties are established upon map axes creation; you subsequently can modify them with the `setm` and the `framem` functions. The command `framem` alone is a toggle for the `Frame` property, which controls the visibility of the frame. You can also call `framem` with property names and values to alter the appearance of the frame:

```
framem('FlineWidth',4,'FEdgeColor','red')
```

The frame is actually a patch with a default face color set to `'none'` and a default edge color of black. You can alter these map axes properties by manipulating the `FFaceColor` and `FEdgeColor` properties. For example, the command

```
setm(gca,'FFaceColor','cyan')
```

makes the background region of your display resemble water. Since the frame patch is always the lowest layer of a map display, other patches, perhaps representing land, will appear above the "water." If an object is subsequently plotted "below" the frame patch, the frame altitude can be recalculated to lie below this object with the command `framem reset`. The frame is replaced and not reprojected.

Set the line width of the edge, which is 2 points by default, using the `FLineWidth` property.

The primary advantage of displaying the map frame is that it can provide positional context for other displayed map objects. For example, when vector data of the coasts is displayed, the frame provides the "edge" of the world.

See the `framem` reference page for more details.

Map and Frame Limits

In the Mapping Toolbox, the map and frame limits are two related map axes properties that limit the map display to a defined region. The map latitude and longitude limits define the extents of geodata to be displayed, while the frame limits control how the frame fits around the displayed data. Any object that extends outside the frame limits is automatically trimmed.

The frame limits are also specified differently from the map limits. The map limits are in absolute geographic coordinates referenced to an origin at the intersection of the prime meridian and the equator, while the frame limits are referenced to the rotated coordinate system defined by the map axes origin.

For all non-azimuthal projections, frame limits are specified as quadrangles (`[`*`latmin latmax`*`]` and `[`*`longmin longmax`*`]`) in the frame coordinate system. In the case of azimuthal projections, the frames are circular and are described by a polar coordinate system. One of the frame latitude limits must be a negative infinity (`-Inf`) to indicate an azimuthal frame (think of this as the center of the circle), while the other limit determines the radius of the circular frame (*rlatmax*). The longitude limits of azimuthal frames are inconsequential, since a full circle is always displayed.

If you are uncertain about the correct format for a particular projection frame limit, you can reset the formats to the default values using empty matrices.

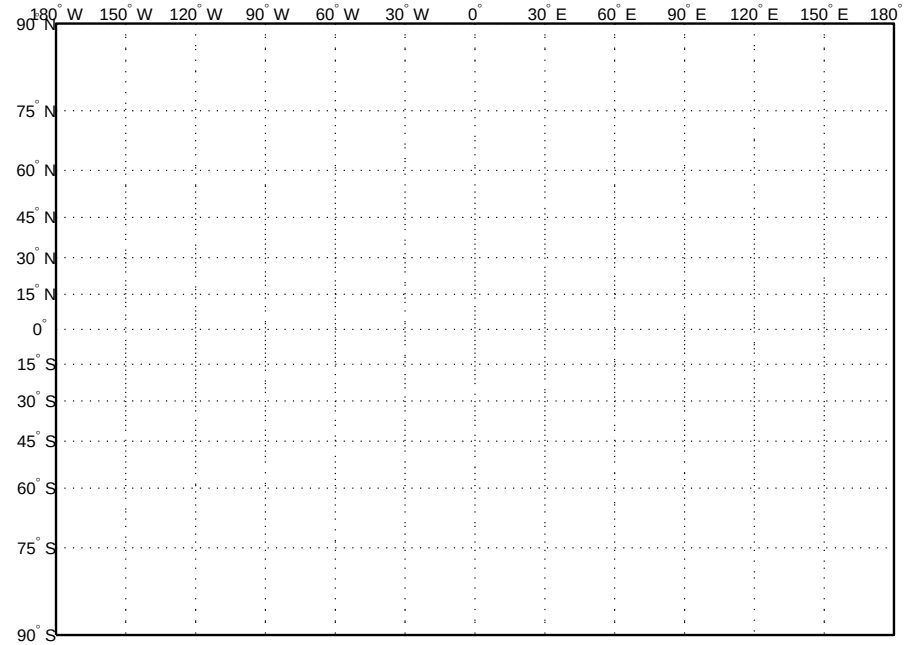
Note For nonazimuthal projections in the normal aspect, the map extent is limited by the minimum of the map limits and the frame limits; hence, the two limits will coincide after evaluation. Therefore if you manually change one set of limits, you might want to clear the other set to get consistent limits.

The Map Grid

The *map grid* is the set of displayed meridians and parallels, also known as a *graticule*. Display the grid by setting the map axes property `Grid` to `'on'`. You can do this when you create map axes with `axesm`, with `setm`, or with the direct command `gridm on`.

Grid Spacing

To control display of meridians and parallels, set a scalar meridian spacing or a vector of desired meridians in the `MLineLocation` property. The property `PLineLocation` serves a corresponding purpose for parallels. The default values place grid lines every 30° for meridians and every 15° for parallels.



Default Grid on a Miller Projection

Grid Layering

By default, the grid is placed as the top layer of any display. You can alter this by changing the `GAltitude` property, so that other map objects can be placed

"above" the grid. The new grid is drawn at its new altitude. The units used for `GAltitude` are specified with the `daspectm` function.

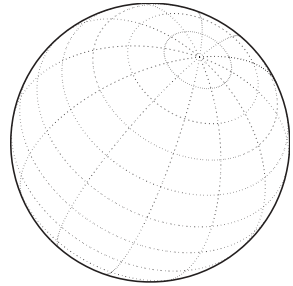
To reposition the grid back to the top of the display, use the command `gridm reset`. You can also control the appearance of grid lines with the `GLineStyle` and `GLineWidth` properties, which are `':'` and `0.5`, respectively, by default.

Limiting Grid Lines

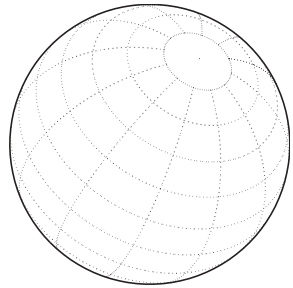
The Miller projection is an example in which all the meridians can extend to the poles without appearing to be cluttered. In other projections, such as the orthographic (below), the map grid can obscure the surface where they converge. Two map axes properties, `MLineLimit` and `MLineException`, enable you to control such clutter:

- Use the `MLineLimit` property to specify a pair of latitudes at which to terminate the meridians. For example, setting `MLineLimit` to `[-75 75]` completely clears the region above and below this latitude range of meridian lines.
- If you want some lines to reach the poles but not others, you can specify them with the `MLineException` property. For example, if `MLineException` is set to `[-90 0 90 180]`, then the meridians corresponding to the four cardinal longitudes will continue past the limit on to the pole.

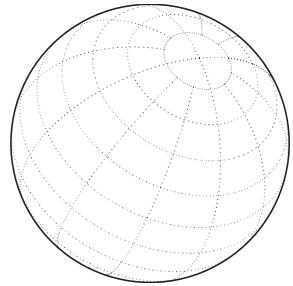
The use of these properties is illustrated in the figure below. Note that there are two corresponding map axes properties, `PLineLimit` and `PLineException`, for controlling the extent of displayed parallels.



Default grid allows all displayed meridians to extend to the poles.



The property `MLineLimit` can truncate meridians at a given latitude, here at 75×N and S.



The property MLineException allows certain meridians to extend to the poles despite the `MLineLimit`. Here, four meridians, at 90°W, 0°, 90°E, and 180°, are excepted.

Labeling Grids

You can label displayed parallels and meridians. `MeridianLabel` and `ParallelLabel` are on-off properties for displaying labels on the meridians and parallels, respectively. They are both `'off'` by default. Initially, the label locations coincide with the default displayed grid lines, but you can alter this by using the `PlabelLocation` and `MlabelLocation` properties. These grid lines are labeled across the north edge of the map for meridians and along the west edge of the map for parallels. However, the property `MlabelParallel` allows

you to specify `'north'`, `'south'`, `'equator'`, or a specific latitude at which to display the meridian labels, and the `PlabelMeridian` allows the choice of `'west'`, `'east'`, `'prime'`, or a specific longitude for the parallel labels. By default, parallel labels are displayed in the range of 0° to 90° north and south of the equator while meridian labels are displayed in the range of 0° to 180° east and west of the prime meridian. You can use the `mlabelzero22pi` function to redisplay the meridian labels in the range of 0° to 360° east of the prime meridian.

Properties affecting grid labeling are listed below:

Property	Effect
`MeridianLabel`	Toggles display of meridian labels
`ParallelLabel`	Toggles display of parallel labels
`MlabelLocation`	Alternate interval for labeling meridians
`PlabelLocation`	Alternate interval for labeling parallels
`MlabelParallel`	Keyword or latitude for placing meridian labels
`PlabelMeridian`	Keyword or longitude for placing parallel labels
`mlabelzero22pi` (function)	Relabel meridians with positive angle from 0° to 360°

For complete descriptions of all map axes properties, refer to the `axesm` reference page.

Displaying Vector Data with Mapping Toolbox Functions

In addition to `mapview`, `maptool`, and other Mapping Toolbox GUIs, you can create maps interactively by entering commands or via scripts. This section describes how to use the principal mapping functions for displaying vector geospatial data. The following section describes displaying raster map data.

Displaying Vector Maps as Lines

The Mapping Toolbox lets you display vector map data as line objects much like the line display functions in MATLAB. The Mapping Toolbox line graphics functions have MATLAB analogs, the names of which can usually be determined by appending an `m` to the MATLAB function name. For instance, the Mapping Toolbox version of `plot` is `plotm`. The main difference between the two classes of functions comes from the need for Mapping Toolbox functions to work with geographic coordinates and map projections.

The following table lists the available Mapping Toolbox line display functions:

Function	Used For
`contourm`	Contour plot of map data
`contour3m`	Contour plot of map data in 3-D space
`linem`	Draws line objects projected on map axes
`plotm`	Clears figure and draws line objects projected on map axes
`plot3m`	Projects lines on map axes in 3-D space

The following exercise shows how some of these functions work:

1. Set up a map axes and frame:

```
load coast
axesm mollweid
framem('FEdgeColor','blue','FLineWidth',0.5)
```

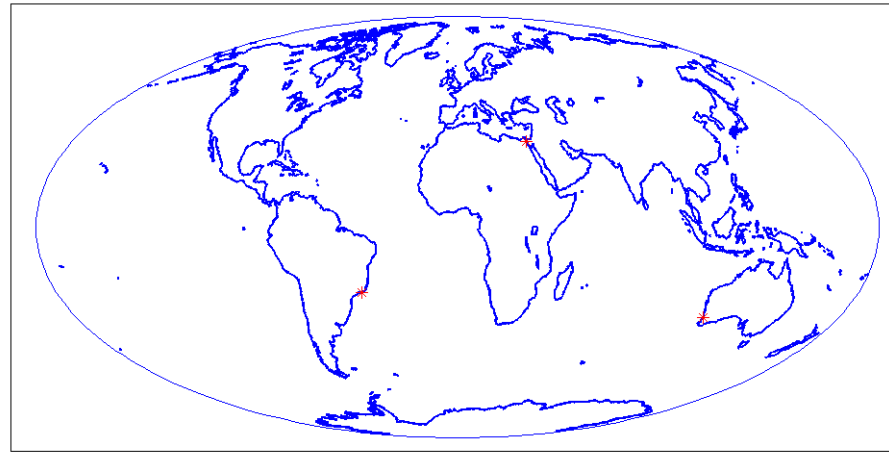
2. Plot the `coast` vector data using `plotm`. Just as with `plot`, you can specify line property names and values in the command:

```
plotm(lat,long,'LineWidth',1,'Color','blue')
```

Sometimes vector data represents specific points. Suppose you have variables representing the locations of Cairo (30°N,32°E), Rio de Janeiro (23°S,43°W), and Perth (32°S,116°E), and you want to plot them as markers only, without connecting line segments.

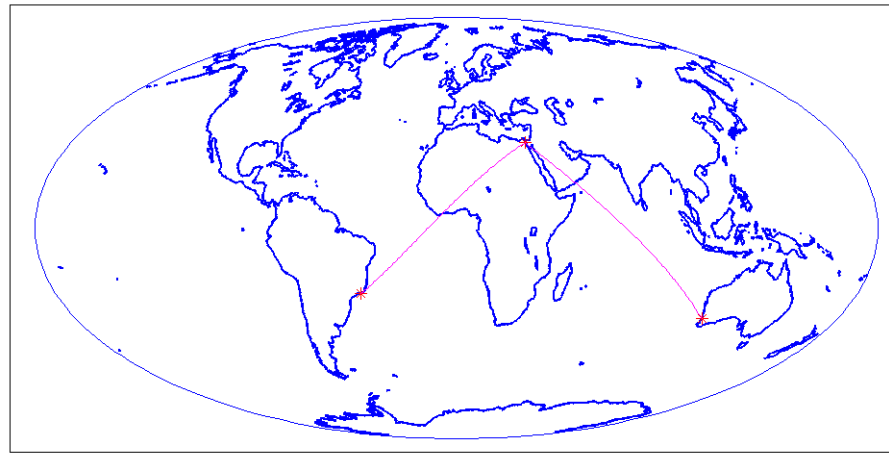
3 Define the three city geographic locations and plot symbols at them:

```
citylats = [30 -23 -32]; citylongs = [32 -43 116];
plotm(citylats,citylongs,'r*')
```



4 In addition to these sorts of "permanent" geographic data, you can also display calculated vector data. Calculate and plot a great circle track from Cairo to Rio de Janeiro, and a rhumb line track from Cairo to Perth:

```
[gclat,gclong] = track2('gc',citylats(1),citylongs(1),...
                              citylats(2),citylongs(2));
[rhlat,rhlong] = track2('rh',citylats(1),citylongs(1),...
                              citylats(3),citylongs(3));
plotm(gclat,gclong,'m-'); plotm(rhlat,rhlong,'m-')
```



Displaying Vector Maps as Patches

Vector map data that is properly formatted (i.e., as closed polygons) can be displayed as patches, or filled-in polygons. The `patchm` function is the Mapping Toolbox equivalent to the MATLAB `patch` function.

Note The Mapping Toolbox patch display functions differ from their MATLAB equivalents by allowing you to display patch vector data that use `NAN`s to separate closed regions.

The following table lists the available Mapping Toolbox patch display functions:

Function	Used For
`fillm`	Filled 2-D map polygons
`fill3m`	Filled 3-D map polygons in 3-D space

Function	Used For
`patchm`	Patch objects projected on map axes
`patchesm`	Patches projected as individual objects on map axes

1. The `usalo` MAT-file is useful for illustrating patches. Load it and observe what it contains:

```
load usalo
who
Your variables are:

conus          gtlakelon      statelat       uslon
greatlakes     state          statelon
gtlakelat      stateborder    uslat
```

   The variables `uslat` and `uslon` together describe three polygons (separated by `NaN`s), the largest of which represents the outline of the conterminous United States. The two smaller polygons represent Long Island and Martha's Vineyard. The variables `gtlakelat` and `gtlakelon` describe three polygons (separated by `NaN`s) for the Great Lakes. The variables `statelat` and `statelon` contain line-segment data (separated by `NaN`s) for the borders between states, which is not formatted for patch display.

2. Verify that line and patch data contains `NaN`s (hence multiple objects) by typing a command similar to `find(isnan(`*`vector`*`))`:

```
find(isnan(gtlakelon)) %or greatlakelat
ans =
         883
        1058
        1229
```

3. Thus you can tell that `greatlakes` contains three geographic objects. How do you know they are patches? If you type

```
greatlakes
```

   you see it is a structure having six fields:

```
greatlakes =
```

```
1x3 struct array with fields:
    type
    tag
    lat
    long
    altitude
    otherproperty
```

4 The `type` field specifies whether the data is stored as lines or patches:

```
greatlakes.type
ans =
patch
```

For further details on how the Mapping Toolbox structures geographic data, see "Understanding Vector Data" on page 2-13. and "Understanding Raster Data" on page 2-27.

Now you will display the state polygons. As conic projections are appropriate for displaying the entire United States, you will create a map axes using an Albers equal-area conic projection. Specifying map limits that contain the region of interest automatically centers the projection on an appropriate longitude; the frame encloses just the mapping area, not the entire globe.

Note Conic projections need two standard parallels (latitudes at which scale distortion is zero). A good rule is to set the standard parallels at one-sixth of the way from both latitude extremes.

---

5 Obtain default latitudes by providing an empty matrix as the standard parallels:

```
axesm('MapProjection','eqaconic', 'MapParallels',[],...
      'MapLatLimit',[23 52], 'MapLonLimit',[-130 -62])
```

6 If you do not know what latitude and longitude limits are appropriate for your map, as a starting point you could use the exact ones. Obtain them for the `usalo` data as follows:

```
uslatlim = [min(uslat) max(uslat)]
uslatlim =
```

```
   25.1200   49.3800

uslonlim = [min(uslon) max(uslon)]
uslonlim =
 -124.7200  -66.9700
```

7 Then you can use these variables to set the frame limits exactly, which will eliminate any border around the map:

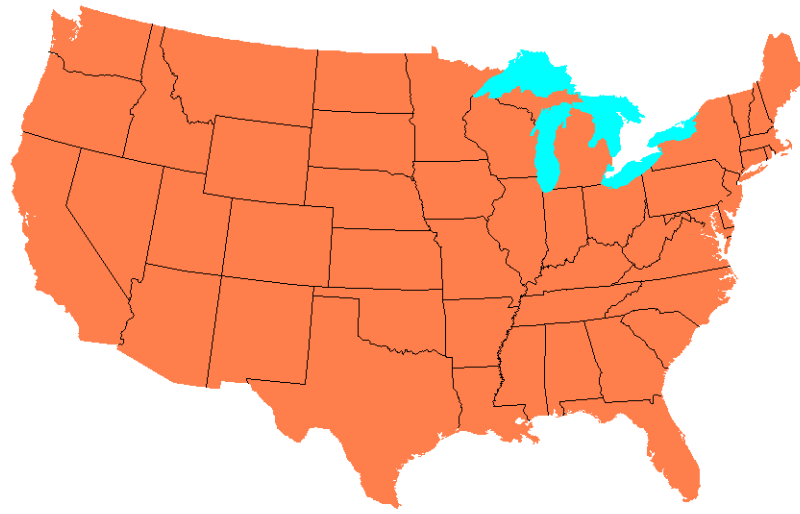
```
axesm('MapProjection','eqaconic', 'MapParallels',[],...
      'MapLatLimit',uslatlim, 'MapLonLimit',uslonlim)
```

When patch data is displayed, layering can become important as objects above can hide objects below. You can control the visibility of objects by the order of their display and by their altitude. For this reason, the primary patch display function in the Mapping Toolbox, `fillm`, allows you to control the z-axis level of displayed patches. Here the land data is displayed at the default level, $z = 0$. The Great Lakes data is assigned the altitude of $z = 1$.

8 To plot the line segment data between these layers, use the `plot3m` function and specify an altitude of `0.5`:

```
fillm(uslat,uslon, 'FaceColor',[1 .5 .3], 'EdgeColor','none')
fillm(gtlakelat,gtlakelon,1,...
      'FaceColor','cyan', 'EdgeColor','none')
plot3m(statelat,statelon,0.5,'k')
```

Here is the resulting map:



The `fillm` function makes use of the low-level function `patchm`. The Mapping Toolbox provides another patch drawing function called `patchesm`. The optimal use of either depends on the application and user preferences. The `patchm` function creates one displayed object and returns one handle for a patch, which can contain multiple faces that do not necessarily connect. The Mapping Toolbox uses `NaN`s to separate unconnected patch faces, unlike MATLAB, which does not handle `NaN` clipped data for patches. The `patchesm` function, on the other hand, treats each face as a separate object and returns an array containing a handle for each patch. In general, `patchm` requires more memory but is faster than `patchesm`. The `patchesm` function is useful if you need to manipulate the appearance of individual patches (as thematic maps often require).

Displaying Data Grids

The Mapping Toolbox provides functions for the display and enhancement of both regular and geolocated data grids originating in a variety of formats. Recall that regular data grids require a *referencing vector or matrix* that describes the sampling and location of the data points, while geolocated data grids require matrices of latitude and longitude coordinates.

The data grid display functions are geographic analogies to the MATLAB surface drawing functions, but operate specifically on map axes. Like the line plotting functions discussed in the previous chapter, Mapping Toolbox grid function names are mostly identical to their MATLAB counterparts, with an `m` appended.

Note In the Mapping Toolbox, functions beginning with `mesh` are used for regular data grids, while those with `surf` are reserved for geolocated data grids. This usage differs from the MATLAB definition; that is, `mesh` plots are used for colored, wire-frame views of the surface, while `surf` displays colored, faceted surfaces.

Surface map objects can be displayed in a variety of different ways. You can assign colors from the figure colormap to surfaces according to the values of their data. You can also display images where the matrix data consists of indices into a colormap or display the matrix as a three-dimensional surface, with the *z*-coordinates given by the map matrix. You can use monochrome surfaces that reflect a pseudo-light source, thereby producing a three-dimensional, shaded relief model of the surface. Finally, you can use a combination of color and light shading to create a lighted shaded relief map.

The following table lists the available Mapping Toolbox surface map display functions:

Function	Used For
`meshm`	Regular data grid warped to projected graticule mesh
`surfm`	Geolocated data grid projected on map axes
`pcolorm`	Projected data grid in z = 0 plane

Function	Used For
`surfacem`	Data grid warped to projected graticule mesh
`surflm`	3-D shaded surface with lighting projected on map axes
`meshlsrm`	3-D lighted shaded relief of regular data grid
`surflsrm`	3-D lighted shaded relief of geolocated data grid

Fitting Gridded Data to the Graticule

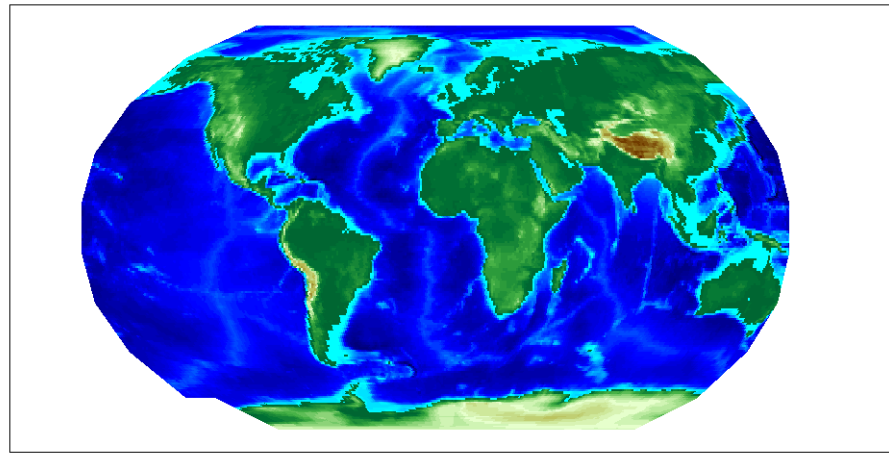
The Mapping Toolbox projects surface objects in a manner similar to the traditional methods of map making. A cartographer first lays out a grid of meridians and parallels called the *graticule*. Each graticule cell is a geographic quadrangle. The cartographer calculates or interpolates the appropriate *x-y* locations for every vertex in the graticule grid and draws the projected graticule by connecting the dots. Finally, the cartographer draws the map data freehand, attempting to account for the shape of the graticule cells, which usually change shape across the map. Similarly, the Mapping Toolbox calculates the *x-y* locations of the four vertices of each graticule cell and warps or samples the matrix data to fit the resulting quadrilateral.

In mapping data grids using the toolbox, as in traditional cartography, the finer the mesh (analogous to using a graticule with more meridians and parallels), the greater precision the projected map display will have, at the cost of greater effort and time. The graticule in a printed map is analogous to the spacing of grid elements in a regular data grid, which the Mapping Toolbox represents as two-element vectors, of the form `[`*`number-of-parallels`*`,` *`number-of-meridians`*`]`. The graticule for geolocated data grids is similar; it is the size of the latitude and longitude coordinate matrices, where *`number-of-parallels`*`=mrows-1` and *`number-of-meridians`*`=ncols-1`. However, because geolocated data grids have arbitrary cell corner locations, spacing can vary and thus their graticule is not a regular equiangular mesh.

In other words, while the structure of cells for regular data grids is restricted to equal-angle quadrangles (i.e., length of cell in latitude must equal length of cell in longitude), geolocated data grids have no such constraints. Their cells can be of any size.

The topo regular data grid can be displayed quickly using a coarse graticule, at a cost in precision of representation. Observe the map that results from the following commands:

```
close all; clear all;
load topo                                %Get data grid and ref vec
figure; axesm robinson                   %Set up Robinson proj
spacing = [10 20];                       %Spec a 10x20 cell grid
h = meshm(topo,topolegend,spacing);      %Draw data into grid
demcmap(topo)                            %Set DEM color map
```



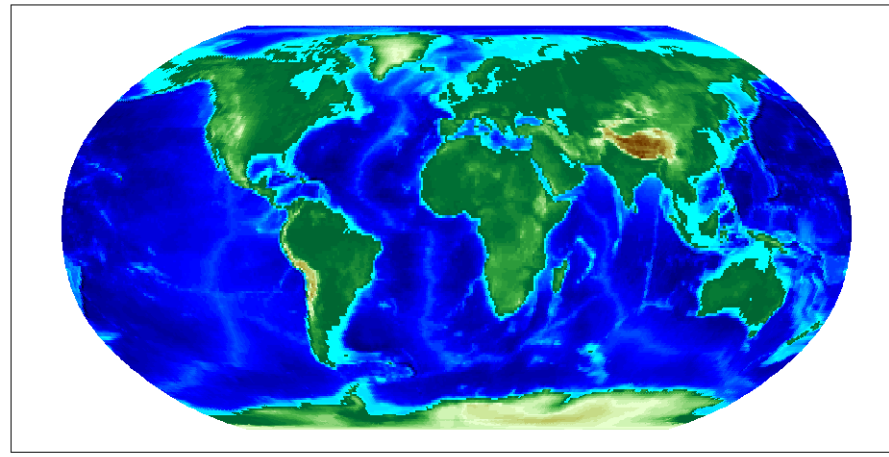
Notice that for this coarse graticule, the edges of the map do not appear as smooth curves. What might not be as obvious is that the easternmost column of graticule cells and the southwesternmost cell are sometimes invisible on displayed data grids. This is necessary for the proper projection of the surface object and is not a concern except with the coarsest graticules. Previous displays used the default [50 100] graticule, for which this effect is negligible.

Regardless of the graticule resolution, the grid data is unchanged. In this case, the data grid is the 180-by-360 topo matrix, and regardless of projection fidelity, the resolution of its value data is unchanged.

Map objects displayed as surfaces have all the properties of any MATLAB surface, which can be set at object creation or by using the MATLAB set function. The mapping setm function allows the MeshGrat graticule property to

be manipulated for regular matrix surfaces. Since you saved the handle of the last displayed map, reset its graticule to a very fine grid. As making the mesh more precise is a tradeoff between resolution and time, doing this will take longer to display the map:

```
setm(h,'MeshGrat',[200 400])
```



Another way you could have done this is with the `meshgrat` function:

```
[latgrat,longrat] = meshgrat(topo,topolegend,[200 400])
setm(h,'Graticule',latgrat,longrat)
```

The vectors `latgrat` and `longrat` produced by `meshgrat` are vectors containing parallel and meridian values in each mesh direction.

You'll probably notice that the result does not appear to be any better than the original display with the default [50 100] graticule, but it took much longer to produce. There is no point to specifying a mesh finer than the data resolution (in this case, 180-by-360 grid cells). In practice, you will probably use coarse graticules for development tasks and fine graticules for final graphics production.

Using Raster Data to Create 3-D Displays

The simplest way to display raster data is to assign colors to matrix elements according to their data values and view them in two dimensions. Raster data maps also can be displayed as 3-D surfaces using the matrix values as the *z* data. Here you explore some basic concepts and operations for setting up surface views, which requires explicit horizontal coordinates.

Note The difference between regular raster data and a geolocated data grid is that each grid intersection for a geolocated grid is explicitly defined with (x,y) or (latitude, longitude) matrices or is interpolated from a graticule, while a regular matrix only implies these locations (which is why it needs a georeferencing vector or matrix).

You will use the raster elevation data in the `korea` MAT-file, which also includes bathymetry data for the region around the Korean peninsula, along with a referencing vector variable, which indicates the data set is a regular data grid and locates it on the Earth:

1. Load the MAT-file and transform this representation to a fully geolocated data grid by calculating a mesh via the `meshgrat` function.

   ```
   load korea
   [lat,lon] = meshgrat(map,maplegend);
   ```

2. Next use the `km2deg` function to convert the units of elevation from meters to degrees, so they are commensurate with the latitude and longitude coordinate matrices.

   ```
   map = km2deg(map/1000);
   ```

3 Observe the results by typing the `whos` command.

```
whos
  Name            Size                    Bytes  Class

  ans             0x0                         0  double array
  lat           180x240                   345600  double array
  lon           180x240                   345600  double array
  map           180x240                   345600  double array
  maplegend       1x3                         24  double array
```

Notice that the `lat` and `lon` coordinate matrices form a mesh the same size as the `map` matrix. This is a requirement for constructing 3-D surfaces, unlike the example given above using the `topo` raster data set, which was displayed in 2-D using the `meshm` function. If you inspect `lat` and `lon` in the MATLAB array editor, you find that in `lon` all columns contain the same number for a given row, and in `lat`, all rows contain the same number for a given column. This is because the mesh produced by `meshgrat` in this case is regular, but such data grids need not have equal spacing.

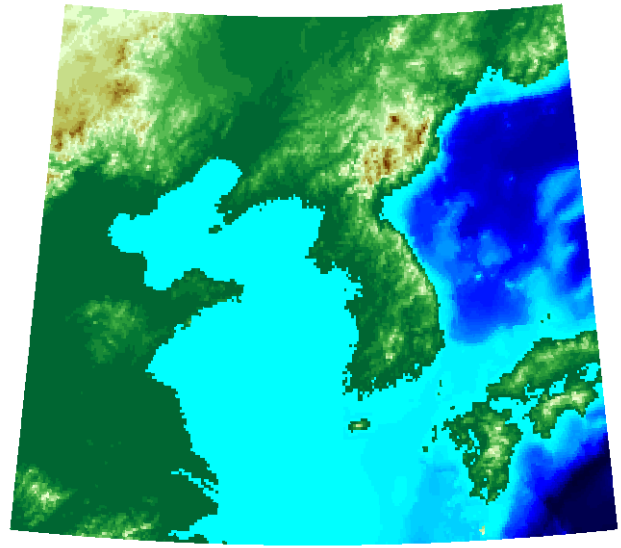
4 Now set up map axes with the equal area conic projection:

```
axesm('MapProjection','eqaconic','MapParallels',[],...
     'MapLatLimit',[30 45],'MapLonLimit',[115 135])
```

5 Instead of using the `meshm` function to make this map, display the `korea` geolocated data grid using the `surfm` function, and set an appropriate colormap:

```
surfm(lat,lon,map,map); demcmap(map)
```

Here is the result, which is no different than what `meshm` would produce:

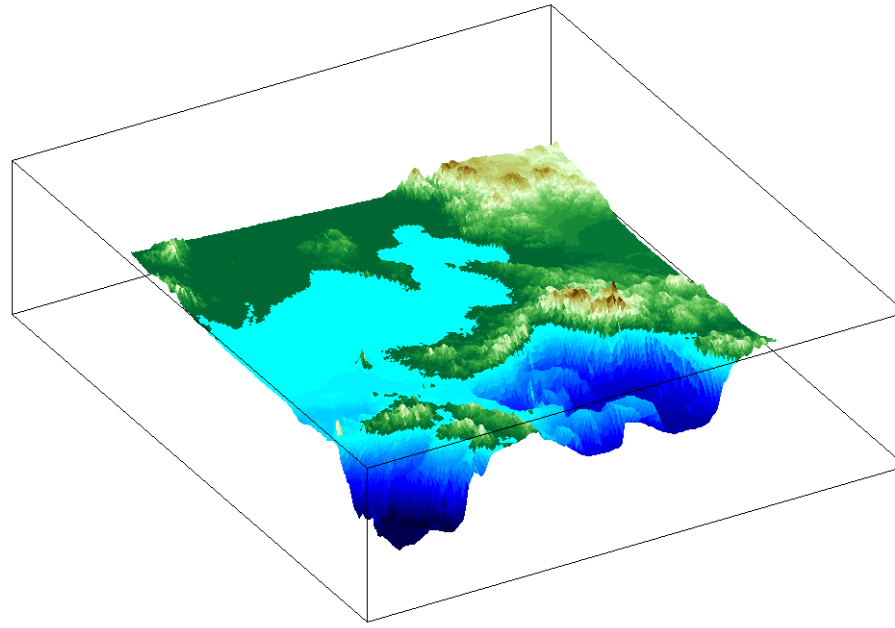


Be aware, however, that this map is really a 3-D view seen from directly overhead (the default perspective). To appreciate that, all you need to do is to change your viewpoint.

6 Use the `view` function to specify a viewing azimuth of 60 degrees (from the East Southeast) and a viewing elevation of 30 degrees above the horizon:

```
view(60,30)
```

The figure immediately rotates to the specified perspective:



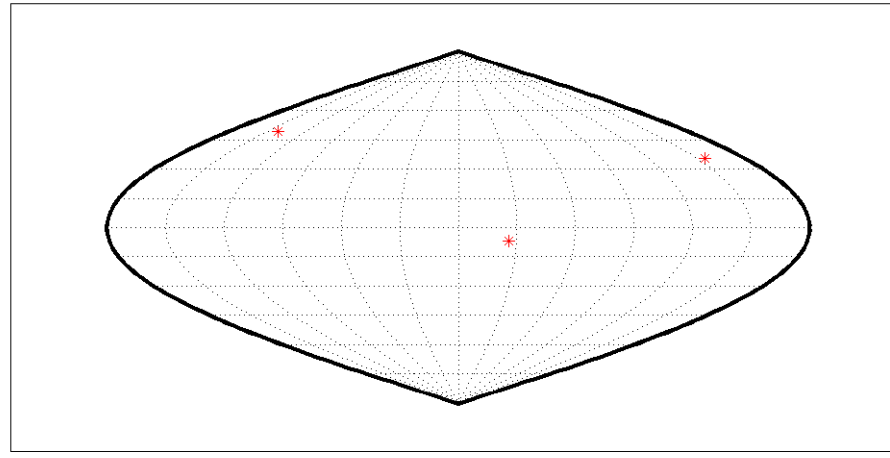
The Mapping Toolbox provides many other controls over perspective map representations. See Chapter 5, “Making Three-Dimensional Maps,” for additional help on constructing 3-D views.

Interacting with Displayed Maps

You can use the Mapping Toolbox to interact with maps, both in `mapview` and in figures created with `axesm`. This section describes two useful graphic input functions, `inputm` and `gcpmap`. The `inputm` function (analogous to the MATLAB `ginput` function) allows you to get the latitude-longitude position of a mouse click. The `gcpmap` function (analogous to the MATLAB function `get(gca,'CurrentPoint'))` returns the current mouse position, also in latitude and longitude.

Explore `inputm` with the following commands, which display a map axes with its grid, and then requests three mouse clicks, the locations of which are stored as geographic coordinates in the variable `points`. Then the `plotm` function plots the points you clicked on as red markers. The display you see depends on the points you select:

```
axesm sinusoid
framem on; gridm on
points=inputm(3)
points =
  -41.7177 -145.0293
    7.9211   -0.5332
   38.5492  149.2237
plotm(points,'r*')
```



Note If you click outside the map frame, `inputm` returns a valid but incorrect latitude and longitude, even though the point you indicated is off the map.

One reason you might want to manually identify points on a map is to interactively explore how much distortion a map projection has at given locations. For example, you can feed the data acquired with `inputm` to the `distortcalc` function, which computes area and angular distortions at any location on a displayed map axes. If you do so using the `points` variable, the results of the previous three mouse clicks are as follows:

```
[areascale,angledef] = distortcalc(points(1,1),points(1,2))
areascale =
    1.0000
angledef =
    85.9284
>> [areascale,angledef] = distortcalc(points(2,1),points(2,2))
areascale =
    1.0000
angledef =
    3.1143
[areascale,angledef] = distortcalc(points(3,1),points(3,2))
areascale =
    1.0000
angledef =
    76.0623
```

This indicates that the current projection (sinusoidal) has the equal-area property, but exhibits variable angular distortion across the map, less near the equator and more near the poles.

Defining Small Circles and Tracks Interactively

Geographic line annotations such as navigational tracks and small circles can be generated interactively. Great circle tracks are the shortest distance between points, and when closed partition the Earth into equal halves; a small circle is the locus of points at a constant distance from a reference point. Use `trackg` and `scircleg` to create them by clicking on the map. Double-click on the tracks or circles to modify the lines. Shift-click to type specific parameters

into a control panel. The control panels also allow you to retrieve or set properties of tracks and circles (for instance, great circle distances and small circle radii).

The following example illustrates how to interactively create a great circle track from Los Angeles, California, to Tokyo, Japan, and a 1000 km radius small circle centered on the Hawaiian Islands. The track is made via the `trackg` function, which prompts you to select end points for a track with the mouse. The `scircleg` function prompts for two points also, a center and any point on the circumference of the small circle. The specifics of the track and the circle are then adjusted more precisely with dialog controls:

1. Set up an orthographic view centered over the Pacific Ocean. Use the `coast` MAT-file:

   ```
   axesm('ortho','origin',[30 180])
   framem;gridm
   load coast
   plotm(lat,long,'k')
   ```

2. Create a track with the `trackg` function, which prompts for two endpoints. The default track type is a great circle:

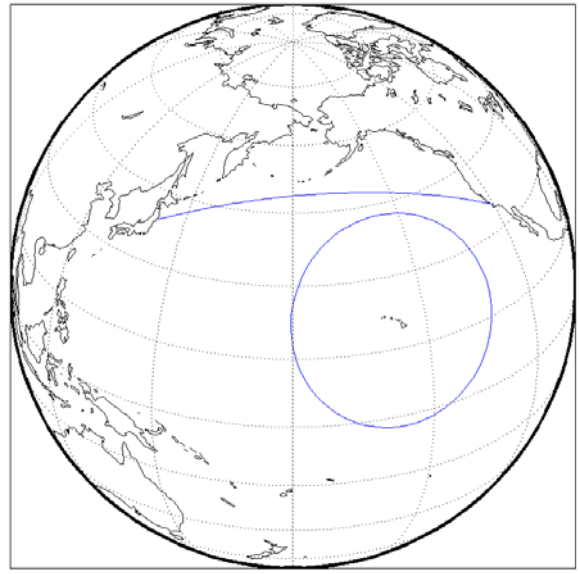
   ```
   trackg
   Track1:  Click on starting and ending points
   ```

   Click near Los Angeles and Tokyo and the track is drawn.

3. Now create a small circle around Hawaii with the `scircleg` function, which prompts for a center point and a point on the perimeter. Make the circle's radius about 2000 km, but don't worry about getting the size exact:

   ```
   scircleg
   Circle 1:  Click on center and perimeter
   ```

   The map should look approximately like this:



4 Adjust the size of the small circle to be 2000 km by shift-clicking anywhere on its perimeter. This brings up the **Small Circles** dialog box.

5 Type `2000` into the R**adius** field.

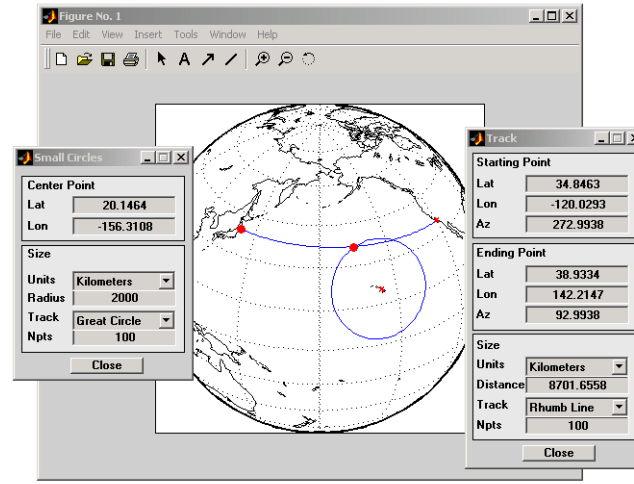
6 Click **Close**. The small circle readjusts to be 2000 km around Hawaii.

7 To adjust the track between Los Angeles and Tokyo, shift-click on it. This brings up the **Track** dialog, with which you specify a position and initial azimuth for either endpoint, as well as the length and type of the track.

8 Change the track type from `Great Circle` to `Rhumb Line` with the **Track** pop-up menu. The track immediately changes shape.

9 Experiment with the other **Track** dialog controls. Also note that you can move the endpoints of the track with the mouse by dragging the red circles, and obtain the arc's length in various units of distance.

The following figure shows the **Small Circles** and **Track** dialogs.



For further information, see the reference page descriptions of the related `scirclui` and `trackui` GUIs, which enable you to define these annotations with or without mouse input.

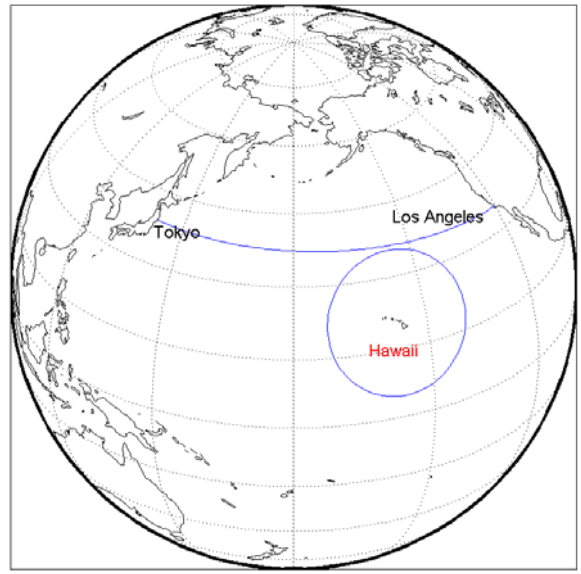
Interactive Text Annotation

You can also interactively place text annotations by clicking on a map display. The `textm` function, which requires numerical arguments for locating a specified text string, was illustrated in Chapter 4, “Placing Geographic and Non-geographic Objects in a Map Axes.” The `gtextm` function, which takes a text string and optional properties as arguments, interactively defines the location for the specified text object based on where you click on the map.

Try these `gtextm` commands to label the locations you have just annotated:

```
gtextm('Hawaii','color','r')
gtextm('Tokyo')
gtextm('Los Angeles')
```

The following figure displays the results of these `gtextm` commands. After you place text, you can move it interactively using the selection tool in the map figure window.



Working with Objects by Name

The Mapping Toolbox allows you to manipulate displayed objects by name. Many mapping functions assign descriptive names to the `Tag` property of the objects they create. The `namem` and related functions allow you to control the display of groups of similarly named objects, determine the names and change them if so desired, and use the name in the Handle Graphics® `set` and `get` functions. There is also a Mapping Toolbox graphical user interface, `mobjects`, to help you manage the display and control of objects.

Some mapping display functions like `framem`, `gridm`, and `contourm` assign object tags by default. You can also set the name upon display by assigning a string to the `Tag` property in mapping display functions that use property name / property value pairs. If the `Tag` does not contain a string, the name defaults to an object's `Type` property, such as `'line'` or `'text'`.

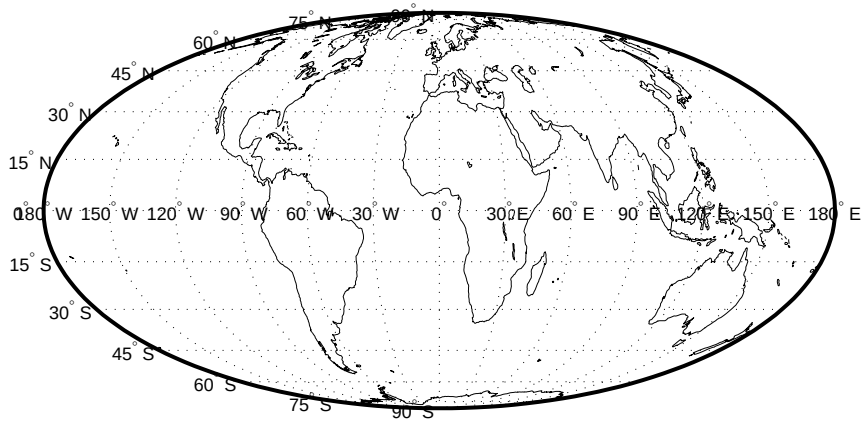
Determining and Manipulating Object Names

1 Display a vector map of the world:

```
f = axesm('fournier')
```

```
framem on; gridm on;
plabel on; mlabel('MLabelParallel',0)
load coast
plotm(lat,long,'k','Tag','Coastline')
```

Below is the resulting map.



2 List the names of the objects in the current axes using `namem`:

```
namem
ans =
Coastline
PLabel
MLabel
Meridian
Parallel
Frame
```

3 The `handlem` function allows you to dereference graphic objects and to get or set their properties. Change the line width of the coastline with `set`:

```
set(handlem('Coastline'),'LineWidth',2)
```

4 Change the colors of the meridian and parallel labels separately:

```
set(handlem('Mlabel'),'Color',[.5 .2 0])
```

```
set(handlem('Plabel'),'Color',[.2 .5 0])
```

You can also change these labels to be the same color using `setm`:

```
setm(f,'fontcolor', [.4 .5 .6])
```

5. The `handlem` command with no arguments summons a UI control with a list of map axes objects. This is useful for selecting objects interactively. Try

   ```
   handlem
   ```

   or

   ```
   h = handlem
   ```

6. Combined with `set`, this makes it simple to change properties such as color. Remember, however, to use the right property name. Patches, for example, have a `'FaceColor'` and `'EdgeColor'`, while most other objects simply have `'Color'`, as is the case with the `Coastline` object. Now use `handlem` to call a color picker to set the coastline to any color you like:

   ```
   set(handlem,'Color',uisetcolor)
   ```

   The reference page for `handlem` lists the object names that it recognizes. Note that most of these names can be prefixed with `'all'`, which returns an array of all handles for that class of object.

7. Now try `handlem` using the `all` modifier:

   ```
   t = handlem('alltext')
   l = handlem('allline')
   ```

   Note that you can also use `all` with the `hidem` and `showm` functions:

   ```
   hidem('alltext')
   showm('alltext')
   ```

For more information on the use of functions and tools for manipulating objects, consult the `setm`, `getm`, `handlem`, `hidem`, `showm`, `clmo`, `namem`, `tagm`, and `mobjects` entries in the Mapping Toolbox reference documentation.

5

Making Three-Dimensional Maps

The Mapping Toolbox constructs three-dimensional as well as two-dimensional map displays. Any map can be constructed and viewed in three dimensions. Some thematic mapping functions plot 3-D symbolism. The most common 3-D application is terrain visualization, for which terrain data grids supply the altitude data. This chapter describes how to obtain and work with terrain data, techniques for making 3-D surface representations, and continues on to describe ways to drape other data over terrain, how to shade, light, and view both planimetric and spherical 3-D relief displays.

Sources of Terrain Data

Nearly all published terrain elevation data is in the form of data grids. "Displaying Data Grids" on page 4-34 described basic approaches to rendering surface data grids with Mapping Toolbox functions, including viewing surfaces in 3-D axes. The following sections describe some common data formats for terrain data, and how to access and prepare data sets for particular areas of interest.

Digital Terrain Elevation Data from NIMA

The Digital Terrain Elevation Data (DTED) Model is a series of gridded elevation models with global coverage at resolutions of 1 kilometer or finer. DTEDs are products of the U. S. National Imagery and Mapping Agency (NIMA), formerly the Defense Mapping Agency (DMA). The data is provided as 1-by-1 degree tiles of elevations on geographic grids with product-dependent grid spacing.

DTED Level 0

The lowest resolution data is the DTED Level 0, with a grid spacing of 30 arc-seconds, or about 1 kilometer. NIMA has published specifications for DTED at `http://www.nima.mil/ast/fm/acq/89020.pdf`. Level 0 DTED files are available for public download from NIMA over the Internet; see the NIMA public data browser at `http://earth-info.nima.mil/`. The intended coverage is global, but there are currently large gaps in coverage for South America and Africa. There is no public FTP site; you must use the NIMA Web interface to select data, specify its packaging, and download it as a compressed archive.

The DTED files are binary. The files have filenames with the extension `dtN`, where `N` is the level of the DTED product.

Higher-Resolution DTED Files

NIMA also provides higher-resolution terrain data files. DTED Level 1 has a resolution of 3 arc-seconds, or about 100 meters, and was the primary source for the USGS 1:250,000 (1 degree) DEMs. DTED Level 2 and above are at even higher resolutions. DTED Level 1 and above are available to the U. S. Department of Defense and its contractors from NIMA.

Digital Elevation Model Files from USGS

The United States Geological Survey (USGS) has prepared terrain data grids for the U.S. suitable for use at scales between 1:24,000 and 1:250,000 and beyond. Some of this data originated from Defense Mapping Agency DTEDs. Specifications and data quality information are available for these digital elevation models (DEMs) and other U.S. National Mapping Program geodata from the USGS at `http://mapping.usgs.gov/standards/index.html`.

The largest-scale USGS DEMs are partitioned to match the USGS 1:24,000 scale map series. The grid spacing for these elevations models is 30 meters on a Universal Transverse Mercator grid. Each file covers a 7.5 minute quadrangle (note, however, that only a subset of paper quadrangle maps are projected with UTM, and that USGS vector geodata products may not use this coordinate system). The map and data series is available for much of the conterminous United States, Hawaii, and Puerto Rico.

Note USGS no longer directly distributes 1:24,000 DEMs and other large-scale geodata. U.S. DEM files in SDTS format are available from private vendors, either for a fee or at no charge, depending on the data sets involved. The USGS is creating a seamless National Elevation Database (NED) at 1/3 and 1 arc-second resolutions (see `http://seamless.usgs.gov`). USGS describes the available NED data distribution formats as ArcGRID, TIFF, GridFloat, and BIL-meters. For information on locating USGS DEM and other geodata, see `http://edc.usgs.gov/geodata/`.

Determining What Elevation Data Exists for a Region

The Mapping Toolbox provides several functions and a GUI to assist you in deriving file names for and managing digital elevation model data for areas of interest. These tools do not retrieve data from the Internet; however, they do locate files that lie on the MATLAB path and indicate the names of data sets that you can download or order on magnetic media or CD-ROM.

The Mapping Toolbox has utility functions for describing and importing elevation data. The following table describes functions that read in data,

determine what file names might exist for a given area, or return metadata for elevation grid files:

File Type	Description	Function to Read Files	Function to Identify Files
DTED	U.S. Department of Defense Digital Terrain Elevation Data	`dted`	`dteds`
DEM	USGS 1-degree (3-arc-second resolution) digital elevation models	`usgsdem`	`usgsdems`
DEM24K	USGS 1:24K (30-meter resolution) digital elevation models	`usgs24kdem`	n.a.
GTOPO30	Tiles of 30-arc-second global elevation models	`gtopo30`	`gtopo30s`
SDTS DEM	Digital elevation models in U.S. SDTS format	`sdtsdemread`	`sdtsinfo` (reads metadata from catalog file)

Note that the names of functions that identify file names are those of their respective file-reading functions appended with `s`. These functions determine file names for areas of interest, and have calling arguments of the form (`latlim`, `longlim`), with which you indicate the latitude and longitude limits for an area of interest, and all return a list of filenames that provide the elevations required. The southernmost latitude and the westernmost longitude must be the first numbers in `latlim` and `longlim`, respectively.

Using dteds, usgsdems, and gtopo30s to Identify DEM Files

Suppose you want to obtain elevation data for the area around Cape Cod, Massachusetts. You defined your area of interest to extend from 41.1°N to 43.9°N latitude and from 71.9°W to 69.1°W longitude.

1. To determine which DTED files you need, use the `dteds` function, which returns a cell array of strings:

```
dteds([41.1 43.9],[-71.9 -69.1])
ans =
    '\DTED\W072\N41.dt0'
    '\DTED\W071\N41.dt0'
    '\DTED\W070\N41.dt0'
    '\DTED\W072\N42.dt0'
    '\DTED\W071\N42.dt0'
    '\DTED\W070\N42.dt0'
    '\DTED\W072\N43.dt0'
    '\DTED\W071\N43.dt0'
    '\DTED\W070\N43.dt0'
```

   Note three important points about these files:

   a. DTED filenames reflect latitudes only and thus do not uniquely specify a data set; they must be organized within directories that specify longitudes. When you download level 0 DTEDs from the NIMA Web site, The `DTED` directory and its subdirectories are transferred as a compressed archive that you must decompress and place on the MATLAB path.

   b. Some files that the `dteds` function identifies do not exist, either because they completely cover water bodies or have never been created or released by NIMA. In the preceding case, the file `\DTED\W070\N42.dt0` has no land areas lying within its bounds, and thus is not available from NIMA. The `dted` function that reads the downloaded DTEDs handles missing cells appropriately.

   c. NIMA might or might not continue to make DTED and other digital data sets available to the general public online. At this time, it provides a Web interface (“Geospatial Engine”) for downloading DTED level 0 files, which is accessible through `http://www.nima.mil` > **`Products and Services`** > **`Topographical`** > **`Geospatial Engine: Overview`**.

2. To determine the USGS DEM files you need, use the `usgsdems` function:

```
usgsdems([41.1 43.9],[-71.9 -69.1])
ans =
    'portland-w'
    'portland-e'
    'bath-w'
```

```
    'boston-w'
    'boston-e'
    'providence-w'
    'providence-e'
    'chatham-w'
```

Note that, in contrast to the `dteds` command you executed above, there are eight rather than nine files listed to cover the 3-by-3-degree region of interest. The cell that consists entirely of ocean has no name and is thus omitted from the output cell array.

3 To determine the GTOPO30 files you need, use the `gtopo30s` function:

```
gtopo30s([41.1 43.9],[-71.9 -69.1])
ans =
    'w100n90'
```

Note The GTOPO30 and DTED grids are in latitude and longitude, and USGS DEMs grids are in UTM coordinates. The `usgs24kdem` function automatically unprojects the grids from UTM to latitude and longitude.

For additional information, see the reference pages for `dteds`, `usgsdems`, `usgs24kdem`, and `gtopo30s`.

Mapping a Single DTED File with the DTED Function

In this exercise, you render DTED level 0 data for a portion of Cape Cod. The 1° -by-1° file can be downloaded from NIMA or purchased on CD-ROM. You read and display the elevation data at full resolution as a lighted surface to show both large- and small-scale variations in the data.

1 Define the area of interest and determine the file to be obtained:

```
mylats = [41.2 41.95]
mylons = [-70.95 -70.1]
dteds(mylats, mylons)
ans =
    'dted\w071\n41.dt0'
```

2. Download the directories containing this file from NIMA or from a CD-ROM onto your system (for information see "Using dteds, usgsdems, and gtopo30s to Identify DEM Files" on page 5-4). The original data comes as a compressed `tar` or `zip` archive that you must expand before using.

3. Use the `dted` function to create a terrain grid in the workspace at full resolution. If more than one DTED file named `n41.dt0` exists on the path, your working directory must be `/dted/w071` in order to be sure that `dted` finds the correct file. If the file is not on the path, you are prompted to navigate to the `n41.dt0` file by the `dted` function:

   ```
   [cape1,caperef1] = dted('n41.dt0',1,mylats,mylons);
   ```

4. Because DTED files contain no bathymetric depths, decrease elevations of zero slightly to render them with blue when the colormap is reset:

   ```
   cape1(cape1==0)=-1;
   ```

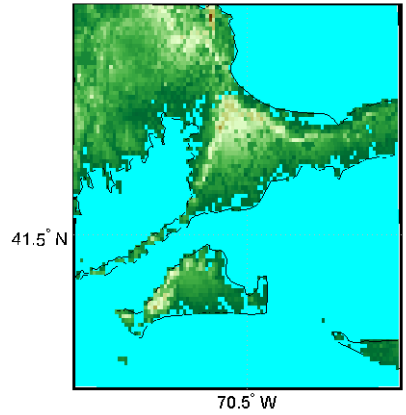
5. Use `usamap` to produce a map of coastlines within your specified limits:

   ```
   usamap(mylats,mylons,'line')
   ```

6. Use `meshm` to render the elevations, and set the colormap accordingly:

   ```
   meshm(cape1,caperef1,size(cape1),cape1);
   demcmap(cape1)
   ```

   The resulting map, shown below, is a window on Cape Cod, and illustrates the relative coarseness of DTED level 0 data.



Mapping Multiple DTED Files with the DTED Function

When your region of interest extends across more that one DTED tile, the `dted` function concatenates the tiles into a single matrix, which can be at full resolution or a sample of every *n*th row and column. You can specify a single DTED file, a directory containing several files (for different latitudes along a constant longitude), or a higher-level directory containing subdirectories with files for several longitude bands.

1 To follow this exercise, you need to acquire the necessary DTED files from `http://www.nima.mil`, as described in the prior exercise, or from a CD-ROM. This yields a set of directories that contain the following files:

```
/dted
    /w070
        n41.avg
        n41.dt0
        n41.max
        n41.min
        n43.avg
        n43.dt0
        n43.max
        n43.min
    /w071
        n41.avg
        n41.dt0
```

```
        n41.max
        n41.min
        n42.avg
        n42.dt0
        n42.max
        n42.min
        n43.avg
        n43.dt0
        n43.max
        n43.min
    /w072
        n41.avg
        n41.dt0
        n41.max
        n41.min
        n42.avg
        n42.dt0
        n42.max
        n42.min
        n43.avg
        n43.dt0
        n43.max
        n43.min
```

2 Change your working directory to the directory that includes the top-level DTED directory (which is always named `dted`):

3 Use the `dted` function, specifying that directory as the first argument. You must also specify a sampling interval as the second, latitude limits as the third, and longitude limits as the fourth arguments:

```
[capetopo,caperef] = dted(pwd, 5, [41.1 43.9],[-71.9 -69.1]);
```

The greater the sampling interval parameter is, the smaller your output grid file will be (by its square).

Note You can specify a DTED filename rather than a directory name if you are accessing only one DTED file. If the file cannot be found, a file dialog is presented for you to navigate to the file you want. See the example "Mapping a Single DTED File with the DTED Function" on page 5-6.

4 As DTEDs contain no bathymetric depths, recode all zero elevations to -1, to enable water areas to be rendered properly:

```
capetopo(capetopo==0)=-1;
```

5 Obtain the elevation grid's latitude and longitude limits; use them to draw an outline map of the area to orient the viewer:

```
[latlim,lonlim] = limitm(capetopo,caperef);
usamap(latlim,lonlim,'line')
```

6 Render the elevation grid with `meshm`, and then recolor the map with `demcmap` to display hypsometric colors (elevation tints):

```
meshm(capetopo,caperef,size(capetopo),capetopo);
demcmap(capetopo)
```

Reading Elevation Data Interactively

You can browse many formats of digital elevation map data using the `demdataui` graphical user interface. The `demdataui` GUI depicts coverage of data sets and reads data ETOPO5, TerrainBase, the satellite bathymetry model (SATBATH), GTOPO30, GLOBE, and DTED files into the workspace.

Note When it opens, `demdataui` scans your MATLAB path for candidate data files. On PCs, it also checks the root directory of CD-ROMs and other drives. This can cause a delay before the GUI appears.

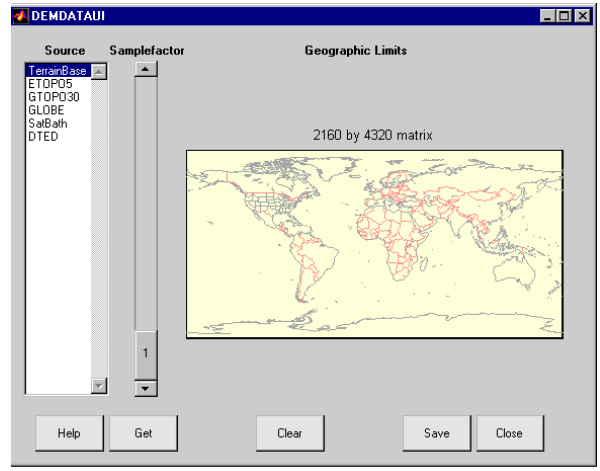
You can choose to read from any of the data sets `demdataui` has located. If `demdataui` does not recognize data that you think it should find, check your path and use the **Help** button to read about how files are identified.

Extracting DEM Data with demdataui

This exercise illustrates how to use the `demdataui` interface. You will not necessarily have all the DEM data sets shown in this example. Even if you have only one (the DTED used in the previous exercise, for example), you can still follow the steps to obtain your own results.

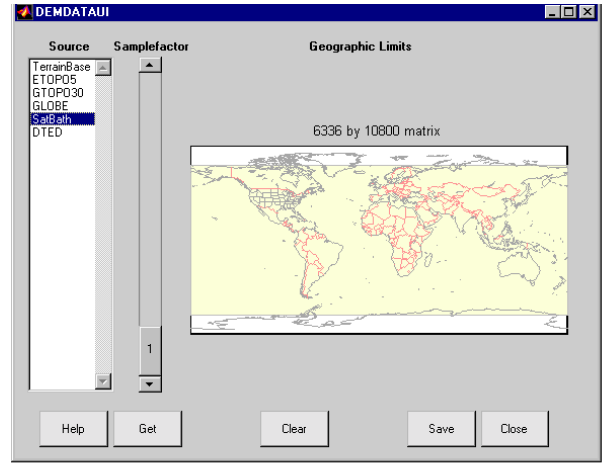
1. Open the `demdataui` UI. It will scan the path for data before it is displayed:

   ```
   demdataui
   ```



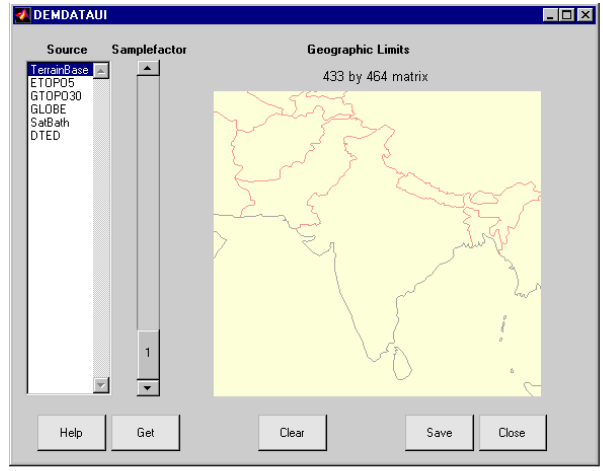
The **Source** list in the left pane shows the data sets that were found. The coverage of each data set is indicated by a yellow tint on the map.

2 Clicking on a different source in the left column updates the coverage display. Here is the coverage area for the satellite bathymetry data set:

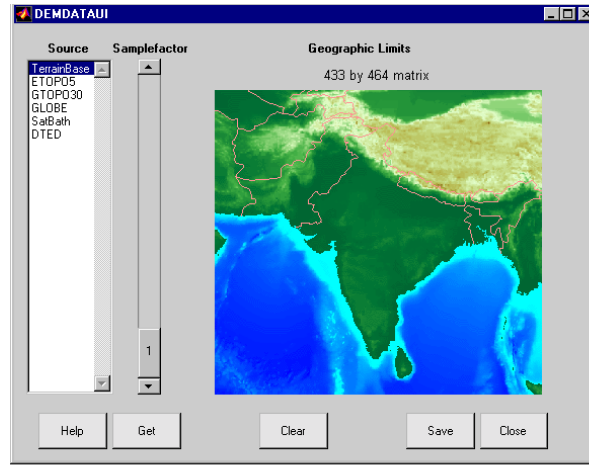


3 The map is used to determine how much data to extract. The storage required for the matrix of the area currently displayed is printed above the map. To reduce the amount of data, click or click and drag on the map to zoom in, or raise the **Samplefactor** slider. A sample factor of 1 reads every point, 2 reads every other point, 3 reads every third point, etc. The size estimate is only updated when you move the **Samplefactor** slider.

Here is the panel after selection of the TerrainBase data and zooming in on the Indian subcontinent



4 Click the **Get** button to show terrain data from the selected data set at the slider's sample factor within the geographic limits shown. It will appear in the GUI map pane. In this instance, the result of this operation is a matrix of about 400 by 500 cells, as shown on the map below:

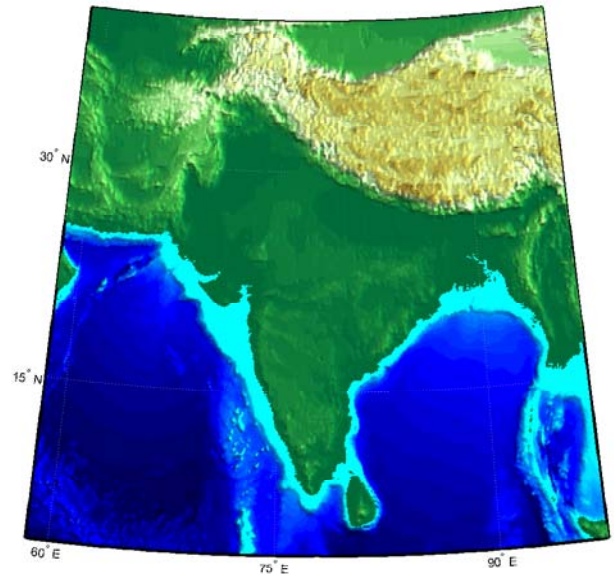


5 If you are not satisfied with the result, click the **Clear** button to remove all data previously read in via **Get** and make new selections.

6 When you are ready to import DEM data to the workspace or save it as a MAT-file, click the **Save** button. You are then asked to select a destination and name the output variable or file. You can save to a MAT-file or to a workspace variable. The `demdataui` function returns one or more matrices as an array of geographic data structures, having one element for each separate *get* you requested (assuming you did not subsequently *clear*). You can then use `displaym` or `mlayers` to add the data grids to a map axes.

7 To access the contents of the geographic data structure, use its field names. Here `map` and `maplegend` are copied from the structure and used to create a lighted three-dimensional elevation map display using `worldmap` (`demdata` is the default name for the structure, which you can override when you save it).

```
map = demdata.map;
maplegend = demdata.maplegend;
figure
worldmap(map,maplegend,'ldem3d')
hidem(gca)
```


30° N
15° N
60° E
75° E
90° E

Determining and Visualizing Visibility Across Terrain

You can use regular data grids of elevation data to answer questions about the mutual visibility of locations on a surface (intervisibility). For example,

- Is the line of sight from one point to another obscured by terrain?
- What area can be seen from a location?
- What area can see a given location?

The first question, on the line of sight between two points, can be answered with the `los2` function. In its simplest form, `los2` determines the visibility between two points on the surface of a digital elevation map. You can also specify the altitudes of the observer and target points, as well as the datum with respect to which the altitudes are measured. For specialized applications, you can even control the actual and effective radius of the Earth. This allows you to assume, for example, that the Earth has a radius 1/3 larger than its actual value, which is a model frequently used in predicting radio wave propagation.

Computing Line-of-Sight with los2

The following example shows a line-of-sight calculation between two points on a regular data grid generated by the `peaks` function. The calculation is performed by the `los2` function, which returns a logical result: `1` (points are intervisible), or `0` (points are not intervisible).

1 Create an elevation grid using `peaks` with a maximum elevation of 500, and set its origin at (0°N, 0°W), with a spacing of 1000 cells per degree):

```
map = 500*peaks(100);
maplegend = [ 1000 0 0];
```

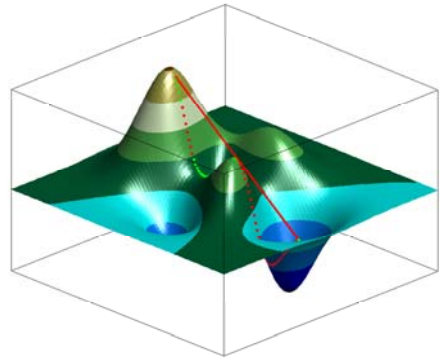
2 Define two locations on this grid to test intervisibility:

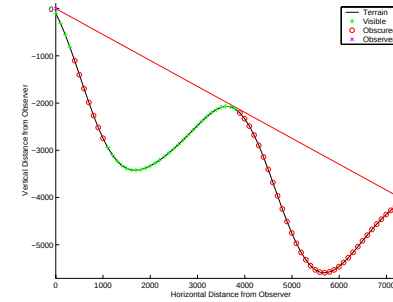
```
lat1 = -0.027; lon1 = 0.05; lat2 = -0.093; lon2 = 0.042;
```

3 Calculate intervisibility. The final argument specifies the altitude (in meters) above the surface of the first location (`lat1`, `lon1`) where the observer is located (the viewpoint):

```
los2(map,maplegend,lat1,lon1,lat2,lon2,100)
ans =
1
```

The `los2` function also produces a profile diagram in a figure window showing visibility at each grid cell along the line of sight that can be used to interpret the Boolean result. In this example, the diagram shows that the line between the two locations just barely clears an intervening peak.

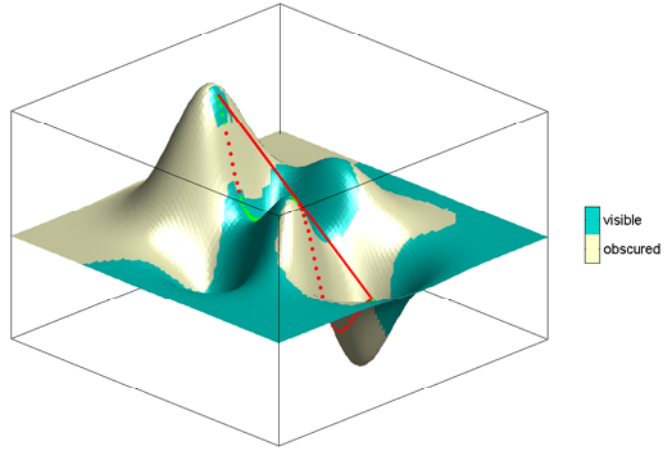




You can also compute the *viewshed*, a name derived from watershed, which is all of the areas that are visible from a particular location. The `viewshed` function can be thought of as performing the `los2` line-of-sight calculation from one point on a digital elevation map to every other entry in the matrix. The `viewshed` function supports the same options as `los2`.

The following shows which parts of the peaks elevation map in the previous example are visible from the first point:

```
[vismap,vismapleg] = viewshed(map,maplegend,lat1,lon1,100);
```


visible
obscured

Shading and Lighting Terrain Maps

The `lightm` function creates light objects in the current map. To modify the positions and colors of lights created on world maps or large regions you can use the interactive *`lightmui`* GUI. For finer control over light position (for example in small areas lit by several lights) you have to specify light positions using projected coordinates. This is because lights are children of axes and share their coordinate space. See "Lighting a Global Terrain Map with lightm and lightmui" on page 5-22 for an example of using `lightmui`.

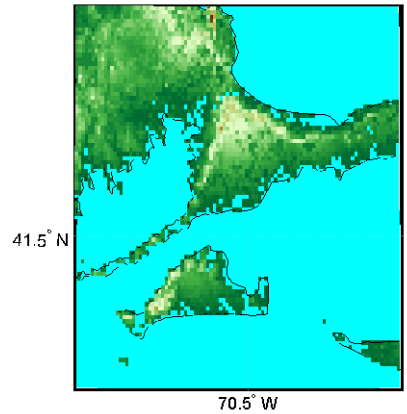
Lighting a Terrain Map Constructed from a DTED File

In this exercise, you manually specify the position of a single light in the northwest corner of a DTED DEM for Cape Cod.

1 To illustrate lighting terrain maps, begin by following the exercise in "Mapping a Single DTED File with the DTED Function" on page 5-6, or execute the steps as reproduced below:

```
mylats = [41.2 41.95];
mylons = [-70.95 -70.1];
cd dted\w071       %Note: Your absolute path may vary
[cape1,caperef1] = dted('n41.dt0',1,mylats,mylons);
cape1(cape1==0)=-1;
usamap(mylats,mylons,'line')
meshm(cape1,caperef1,size(cape1),cape1);
demcmap(cape1)
```

The map looks like this:



2 Set the vertical exaggeration. Use `daspectm` to specify that elevations are in meters and should be multiplied by 20:

```
daspectm('m',20)
```

3 Make sure that the line data is visible. To ensure that it is not obscured by terrain, use `zdatam` to set it to the highest elevation of the `cape1` terrain data:

```
zdatam('allline',max(cape1(:)))
```

4 Specify a location for a light source with `lightm`:

```
h = lightm(42,-71)
```

If you omit arguments, a GUI for setting positional properties for the new light will open.

5 To see the properties of light objects, inspect the handle returned by `lightm`:

```
get(h)
    Position = [-0.00616097 0.796039 1]
    Color = [1 1 1]
    Style = infinite

    BeingDeleted = off
    ButtonDownFcn =
```

```
Children = []
Clipping = on
CreateFcn =
DeleteFcn =
BusyAction = queue
HandleVisibility = on
HitTest = on
Interruptible = on
Parent = [138.001]
Selected = off
SelectionHighlight = on
Tag =
Type = light
UIContextMenu = []
UserData = [ (1 by 1) struct array]
Visible = on
```

Had you used the MATLAB `light` function in place of `lightm`, you would have needed to specify the position in Cartesian 3-space.

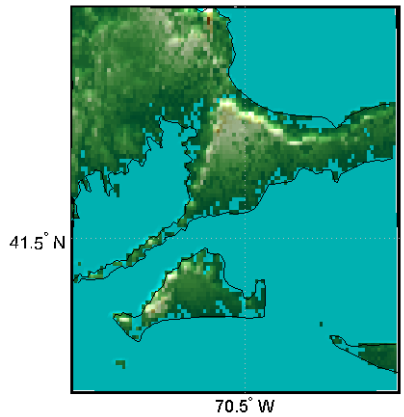
6 The lighting computations caused the map to become quite dark with specular highlights. Now restore its luminance by specifying three surface reflectivity properties in the range of 0 to 1:

```
ambient = 0.7; diffuse = 1; specular = 0.6;
material([ambient diffuse specular])
```

The surface looks blotchy because there is no interpolation of the lighting component (flat facets are being modeled). Correct this by specifying Phong shading:

```
lighting phong
```

The map now looks like this:



7 If you want to compare the lit map with the unlit version, you can toggle the lighting off:

```
lighting none
```

For additional information, see the reference pages for `daspectm`, `lightm`, `light`, `lighting`, and `material`.

Lighting a Global Terrain Map with lightm and lightmui

In this example you create a global topographic map and add a local light at a distance of 250 km above New York City, (40.75 °N, 73.9 °W). You then change the material and lighting properties, add a second light source, and then activate the `lightmui` tool to change light position, altitude, and colors.

The `lightmui` display plots lights as circular markers whose `facecolor` indicates the light color. To change the position of a light, click and drag the circular marker. Alternatively, right-clicking on the circular marker summons a dialog for changing the position or color of the light object. Clicking on the color bar in that dialog invokes the `uisetcolor` dialog box that can be used to specify or pick a color for the light.

1 Load the `topo` DTM files, and set up an orthographic projection:

```
close all; clear;
load topo
axesm ('mapprojection','ortho', 'origin',[10 -20 0])
```

2 Plot the topography and assign a topographic colormap:

```
meshm(topo,topolegend);
demcmap(topo)
```

3 Set up a yellow light source over New York City:

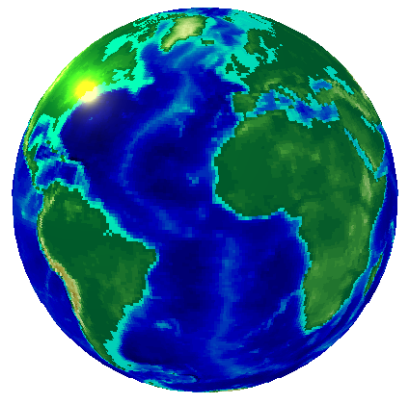
```
hl = lightm(40.75,-73.9, 500/almanac('earth','radius'),...
'color','yellow', 'style', 'local');
```

The first two arguments to `lightm` are the latitude and longitude of the light source. The third argument is its altitude, in units of Earth radii (in this case they are in kilometers, the default units of `almanac`).

4 The surface is quite dark, so give it more reflectivity by specifying

```
material([0.7270  1.5353  1.9860  4.0000  0.9925]);
lighting phong; hidem(gca)
```

The lighted orthographic map looks like this:



5 If you want, you can add more lights, as follows:

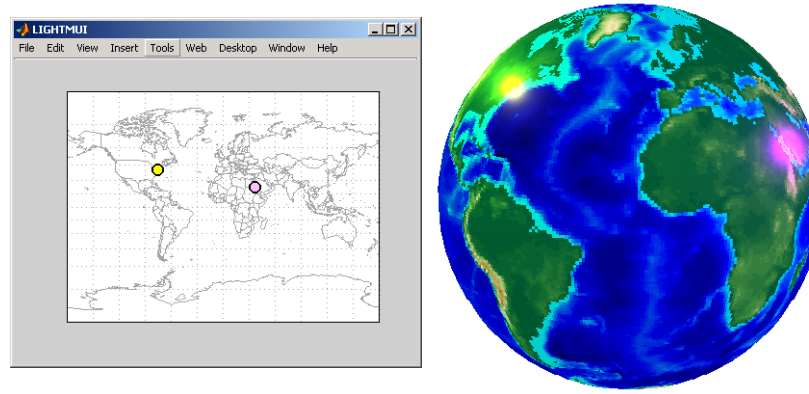
```
h2 = lightm(20,40, .1,'color','magenta', 'style', 'local')
```

The second light is magenta, and positioned over the Gulf of Arabia:

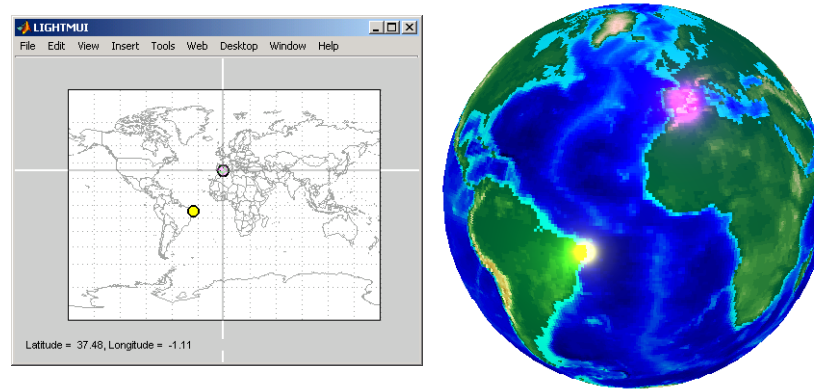
6 To modify the lights, you can use the `lightmui` GUI, which lets you drag lights across a world map and specify their color and altitudes:

```
lightmui(gca)
```

The lights are shown as appropriately colored circles, which you can drag to new positions. You can also **ctrl+**click on a circle to bring up a dialog for directly specifying that light's position, altitude and color. The GUI and the map look like this at this point:

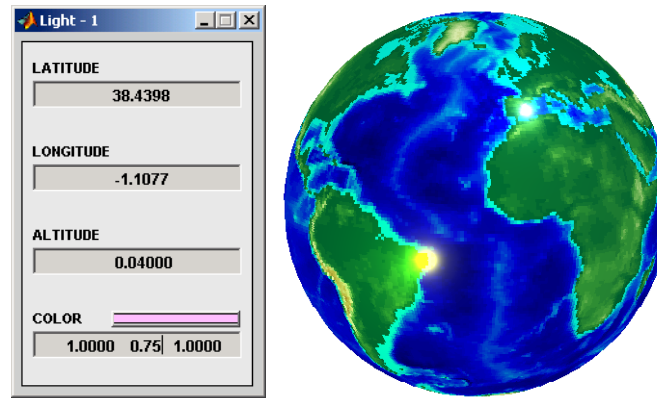


7 In the `lightmui` window, drag the yellow light to the eastern tip of Brazil, and drag the magenta light to the Straits of Gibraltar:



8 **ctrl+**click or Shift-click on the magenta circle in the `lightmui` window. A second UI, for setting light position and color, opens. Set the **altitude** to

0.04 (Earth radii). Set the light **color** components to 1.0 (red), 0.75 (green), and 1.0 (blue). Press **Return** after each action. The color bar on the UI changes to indicate the color you set. If you prefer to pick a color, click on the colorbar to bring up a color-choosing UI. The map now looks like this:



For additional information, see the reference pages for `lightm` and `lightmui`.

Surface Relief Shading

You can make dimensional monochrome shaded-relief maps with the function `surflm`, which is analogous to the MATLAB `surfl` function. The effect of `surflm` is similar to using lights, but the function models illumination itself (with one “light source” that you specify when you invoke it, but cannot reposition) by weighting surface normals rather than using light objects.

Shaded relief maps of this type are usually portrayed two-dimensionally rather than as perspective displays. The `surfl` function works with any projection except `globe`.

The `surflm` function accepts geolocated data grids only. Recall, however, that regular data grids are a subset of geolocated data grids, to which they can be converted using `meshgrat` (see “Fitting Gridded Data to the Graticule” on page 4-35). The following example illustrates this procedure.

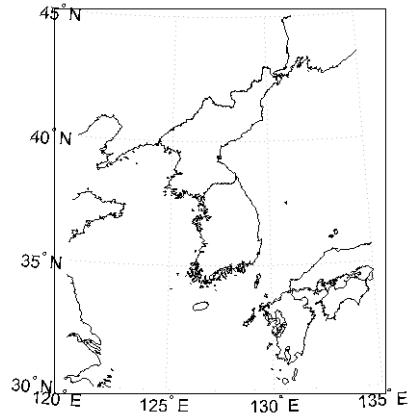
Creating Monochrome Shaded Relief Maps Using surflm

As stated above, `surflm` simulates a single light source instead of inserting light objects in a figure. Conduct the following exercise with the `korea` data set to see how `surflm` behaves. It uses `worldmap` to set up an appropriate map axes and reference outlines.

1. Set up a projection and display a vector map of the Korean peninsula with `worldmap`:

   ```
   hk = worldmap('korea','lineonly'); framem off;
   ```

   `worldmap` chooses a projection and map bounds to make this map:



2. Load the `korea` terrain model:

   ```
   load korea
   ```

3. Generate the grid of latitudes and longitudes to transform the regular data grid to a geolocated one:

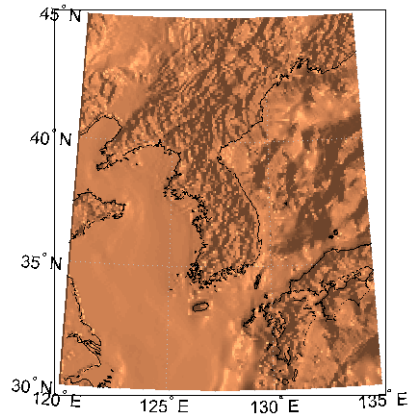
   ```
   [klat,klon] = meshgrat(map,maplegend);
   ```

4. Use `surflm` to generate a default shaded relief map, and change the colormap to a monochromatic scale, such as `gray`, `bone`, or `copper`.

   ```
   ht = surflm(klat,klon,map);
   ```

```
colormap('copper')
```

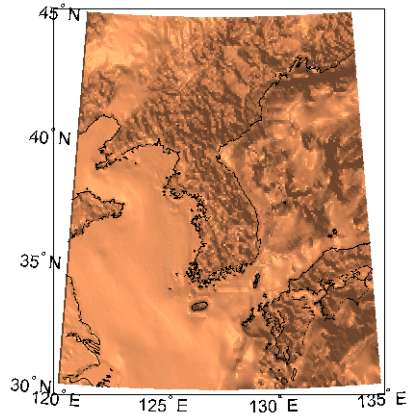
In this default case, the lighting direction is set at 45° counterclockwise from the viewing direction; thus the “sun” is in the southeast. This map is shown below.



5 To make the light come from some other direction, you can specify the light source’s azimuth and elevation as the fourth argument to `surflm`. Clear the terrain map and redraw it, specifying an azimuth of 135° (northeast) and an elevation of 60° above the horizon:

```
clmo(ht); ht=surflm(klat,klon,map,[135,60]);
```

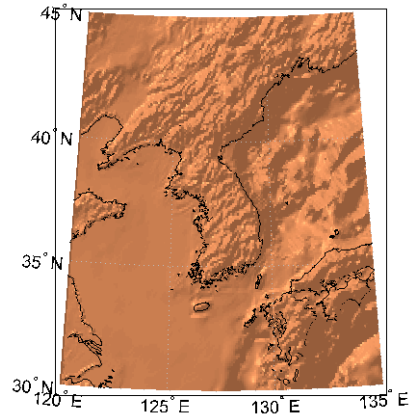
The surface lightens and has a new character because it is lit closer to overhead and from a different direction:



6 Now shift the light to the northwest (-135° azimuth), and lower it to 40° above the horizon. Because a lower "sun" decreases the overall reflectance when viewed from straight above, also specify a more reflective surface as a fifth argument to `surflm`. This is a 1-by-4 vector describing relative contributions of ambient light, diffuse reflection, specular reflection, and a specular shine coefficient. It defaults to `[.55 .6 .4 10]`.

```
clmo(ht); ht=surflm(klat,klon,map,[-135, 30],[.65 .4 .3 10]);
```

This is a good choice for lighting this terrain, because of the predominance of mountain ridges that run from northeast to southwest, more or less perpendicular to the direction of illumination. Here is the final map:



For further information see the reference pages for `surflm` and `surfl`.

Shaded relief representations can highlight the fine structure of the land and sea floor, but because of the monochromatic coloration, it is difficult to distinguish land from sea. The next section describes how to color such maps to set off land from water.

Colored Surface Shaded Relief

The functions `meshlsrm` and `surflsrm` display maps as shaded relief with surface coloring as well as light source shading. You can think of them as extensions to `surflm` that combine surface coloring and surface light shading. Use `meshlsrm` to display regular data grids and `surflsrm` to render geolocated data grids.

These two functions construct a new colormap and associated `CData` matrix that uses grayscales to lighten or darken a matrix component based on its calculated surface normal to a light source. While there are no analogous MATLAB display functions that work like this, you can obtain similar results using MATLAB light objects, as “Relief Mapping with Light Objects” on page 5-31 explains.

Coloring Shaded Relief Maps and Viewing Them in 3-D

In this exercise, you use `surflsrm` in a similar way to how you used `surflm` in the preceding exercise, “Creating Monochrome Shaded Relief Maps Using

surflm" on page 5-26. In addition, you will set a vertical scale and view the map from various perspectives.

1. Start with a new map axes and the `korea` data, then georeference the regular data grid:

   ```
   close all; clear all;
   load korea
   [klat,klon] = meshgrat(map,maplegend);
   axesm miller
   ```

2. Create a colormap for DEM data; it is transformed by `surflsm` to shade relief according to how you specify the sun's altitude and azimuth:

   ```
   [cmap,clim] = demcmap(map);
   ```

3. Plot the colored shaded relief map, specifying an azimuth of -135° and an altitude of 50° for the light source:

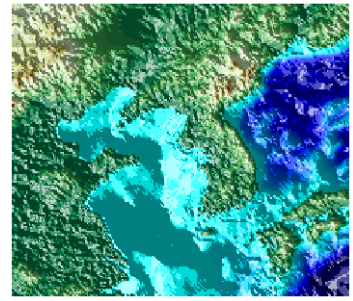
   ```
   surflsrm(klat,klon,map,[-130 50],cmap,clim)
   ```

   You could also achieve the same effect with `meshlsrm`, which operates on regular data grids (it first calls `meshgrat`, just as you just did), e.g., `meshlsrm(map,maplegend)`.

4. The surface will have more contrast than if it were not shaded, and it might help to lighten it uniformly by 25% or so:

   ```
   brighten(.25)
   ```

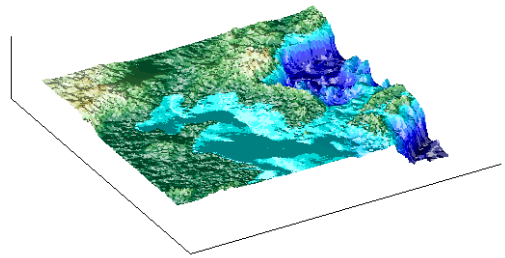
   The map, which has an overhead view, looks like this:



5 Plot an oblique view of the surface by hiding its bounding box, exaggerating terrain relief by a factor of 50, and setting the view azimuth to -30° (south-southwest) and view altitude to 30° above the horizon:

```
set(gca,'Box','off')
daspectm('meters',50)
view(-30,30)
```

The map now looks like this:



6 You can continue to rotate the perspective with the `view` function (or interactively with the **Rotate 3D** tool in the figure window), and to change the vertical exaggeration with the `daspectm` function. You cannot change the built-in lighting direction without generating a new view using `surflsrm`.

For further information, see the reference pages for `surflsrm`, `meshlsrm`, `daspectm`, and `view`.

Relief Mapping with Light Objects

In the exercise "Lighting a Global Terrain Map with lightm and lightmui" on page 5-22, you created light objects to illuminate a globe projection. In the following one, you create a light object to mimic the map produced in the previous exercise ("Coloring Shaded Relief Maps and Viewing Them in 3-D" on page 5-29), which uses shaded relief computations rather than light objects.

The `meshlsrm` and `surflsrm` functions simulate lighting by modifying the colormap with bands of light and dark. The map matrix is then converted to indices for the new "shaded" colormap based on calculated surface normals.

Using light objects allows for a wide range of lighting effects. The Mapping Toolbox manages light objects with the `lightm` function, which depends upon the MATLAB `light` function. Lights are separate MATLAB graphic objects, each with its own object handle.

Colored 3-D Relief Maps Illuminated with Light Objects

As a comparison to the lighted shaded relief example shown earlier, add a light source to the surface colored data grid of the Korean peninsula region:

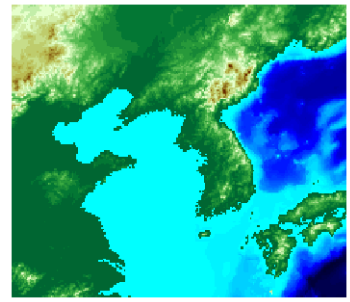
1. If you need to, load the `korea` DEM, and create a map axes using the Miller projection:

   ```
   load korea
   figure; axesm('MapProjection','miller',...
        'MapLatLimit',[30 45],'MapLonLimit',[115 135])
   ```

2. Display the DEM with `meshm`, and color it with terrain hues:

   ```
   meshm(map,maplegend,size(map),map);
   demcmap(map)
   ```

   The map, without lighting effects, looks like this:

3. Create a light object with lightm (similar to the MATLAB light function, but specifies position with latitude and longitude rather than x,y,z). The light is placed at the northwest corner of the grid, one degree high:

   ```
   h=lightm(45,115,1)
   ```

   The figure becomes darker.

4 To see any relief in perspective, it is necessary to exaggerate the vertical dimension. Use a factor of 50 for this:

```
daspectm('meters',50)
```

The figure becomes darker still, with highlights at peaks.

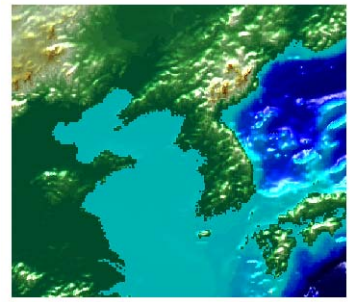
5 Set the ambient (direct), diffuse (skylight) and specular (highlight) surface reflectivity characteristics, respectively:

```
material ([.7, .9, .8])
```

6 By default the lighting is flat (plane facets). Change this to phong shading (interpolated normal vectors at facet corners):

```
lighting phong
```

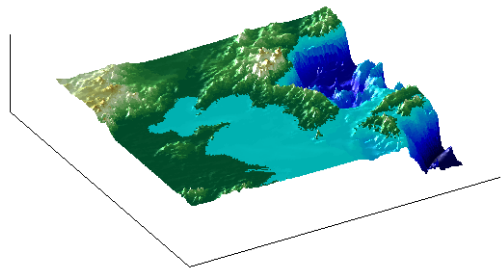
The map now looks like this:



7 Finally, remove the edges of the bounding box and set a viewpoint of -30° azimuth, 30° altitude:

```
set(gca,'Box','off')
view(-30,30)
```

The view from (-30,30) with one light at (45,115,1) and phong shading is shown below. Compare it to the final map in the previous exercise, “Coloring Shaded Relief Maps and Viewing Them in 3-D” on page 5-29.



To remove a light (when there is only one) from the current figure, type

```
clmo(handlem('light'))
```

For more information, consult the reference pages for `lightm`, `daspectm`, `material`, `lighting`, and `view`, along with the section on lighting in the MATLAB graphics documentation.

Draping Data on Elevation Maps

Lighting effects can provide important visual cues when elevation maps are combined with other kinds of data. The shading resulting from lighting a surface makes it possible to "drape" satellite data over a grid of elevations. It is common to use this kind of display to overlay georeferenced land cover images from Earth satellites such as LANDSAT and SPOT on topography from digital elevation models. The Mapping Toolbox can generate such displays using variations of techniques described in the previous section.

When the elevation and image data grids correspond pixel-for-pixel to the same geographic locations, you can build up such displays using the optional altitude arguments in the surface display functions. If they do not, you can interpolate one or both source grids to a common mesh. See "Draping via Converting a Regular Grid to a Geolocated Data Grid" on page 5-38 and "Draping a Geolocated Grid on Regular Data Grid via Texture Mapping" on page 5-40, below for further details on regridding.

Draping Geoid Heights over Topography

The following example shows the figure of the Earth (the `geoid` data set) draped on topographic relief (the `topo` data set). That is, the geoid data is shown as an attribute (using a color scale) rather than being depicted as a 3-D surface itself. The two data sets are both 1-by-1-degree meshes sharing a common origin.

Note The geoid can be described as the surface of the ocean in the absence of waves, tides, or land obstructions. It is influenced by the gravitational attraction of denser or lighter materials in the Earth's crust and interior and by the shape of the crust. A model of the geoid is required for converting ellipsoidal heights (such as might be obtained from GPS measurements) to orthometric heights. Geoid heights vary from a minimum of about 105 meters below sea level to a maximum of about 85 meters above sea level.

---

1 Begin by loading the `topo` and `geoid` regular data grids:

```
load topo
load geoid
```

2 Create a map axes using a Gall stereographic cylindrical projection (a perspective projection):

```
axesm gstereo
```

3 Use `meshm` to plot a colored display of the geoid's variations, but specify `topo` as the final argument, to give each `geoid` grid cell the height (*z*-value) of the corresponding `topo` grid cell:

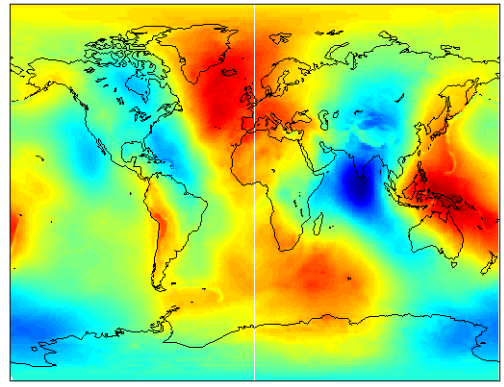
```
meshm(geoid,geoidlegend,size(geoid),topo)
```

Low geoid heights are shown as blue, high ones as red.

4 For reference, plot the world coastlines in black, raise their elevation to 1000 meters (high enough to clear the surface in their vicinity), and expand the map to fill the frame:

```
load coast
plotm(lat,long,'k')
zdatam(handlem('allline'),1000)
tightmap
```

At this point the map looks like this:



5 Due to the vertical view and lack of lighting, the topographic relief is not visible, but it is part of the figure's surface data. Bring it out by exaggerating relief greatly, then setting a view from the south-southeast:

```
daspectm('m',200); tightmap
```

```
view(20,35)
```

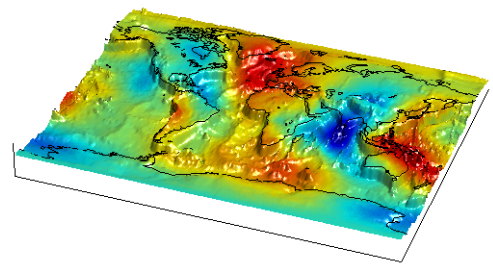
6 Remove the bounding box, shine a light on the surface (using the default position, offset to the right of the viewpoint), and re-render with phong shading:

```
set(gca,'Box','off')
camlight;
lighting phong
```

7 Finally, set the perspective to converge slightly (the default perspective is orthographic):

```
set(gca,'projection','perspective')
```

The final map is shown below. From it, you can see that the geoid mirrors the topography of the major mountain chains such as the Andes, the Himalayas, and the Mid-Atlantic Ridge. You can also see that large areas of high or low geoid heights are not simply a result of topography.



Draping Data over Terrain with Different Gridding

If you want to combine elevation and attribute (color) data grids that cover the same region but are gridded differently, you must resample one matrix to be consistent with the other. It helps if at least one of the grids is a geolocated data grid, because their explicit horizontal coordinates allow them to be resampled using the `ltln2val` function. To combine dissimilar grids, you can either

- Construct a geolocated grid version of the regular data grid values
- Construct a regular grid version of the geolocated data grid values.

The following two examples illustrate these closely related approaches.

Draping via Converting a Regular Grid to a Geolocated Data Grid

This example drapes slope data from a regular data grid on top of elevation data from a geolocated data grid. Although the two data sets actually have the same origin (the geolocated grid derives from the `topo` data set), the approach being demonstrated will work with any dissimilar grids. The example uses the geolocated data grid as the source for surface elevations and transforms the regular data grid into slope values, which are then sampled to conform to the geolocated data grid (creating a set of slope values for the diamond-shaped grid) and color-coded for surface display.

Note When you use `ltln2val` to resample a regular data grid over an irregular area, it is important that the regular data grid completely covers the area of the geolocated data grid.

---

1. Begin by loading the geolocated data grids from `mapmtx`, which contains two regions. You will only use the diamond-shaped portion of `mapmtx` (`lt1`, `lg1`, and `map1`) centered on the Middle East, not the `lt2`, `lg2`, and `map2` data:

   ```
   load mapmtx lt1
   load mapmtx lg1
   load mapmtx map2
   ```

   Also load the `topo` global regular data grid:

   ```
   load topo
   ```

2. Compute surface aspect, slope, and gradients for `topo`. You will use only the slopes in subsequent steps:

   ```
   [aspect,slope,gradN,gradE] = gradientm(topo,topolegend);
   ```

3. Use `ltln2val` to interpolate slope values to the geolocated grid specified by `lt1`, `lg1`:

```
slope1 = ltln2val(slope,topolegend,lt1,lg1);
```

The output is a 50-by-50 grid of elevations matching the coverage of the `map1` variable.

4 Set up a figure with a Miller projection and use `surfm` to display the slope data. Specify the *z*-values for the surface explicitly as the `map1` data, which is terrain elevation:

```
figure; axesm miller
surfm(lt1,lg1,slope1,map1)
```

The map mainly depicts steep cliffs, which represent mountains (the Himalayas in the northeast), and continental shelves and trenches.

5 The coloration depicts steepness of slope. Change the colormap to make the steepest slopes magenta, gentler slopes dark blue, and flat areas light blue:

```
colormap cool;
```

6 Use `view` to get a southeast perspective of the surface from a low viewpoint:

```
view(20,30); daspectm('m',200)
```

In 3-D, you immediately see the topography as well as the slope.

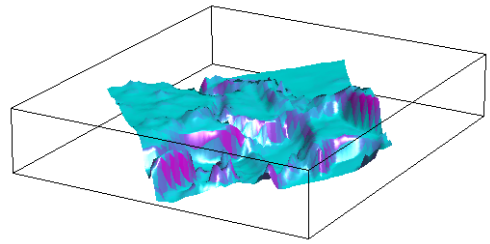
7 The default rendering uses faceted shading (no smooth interpolation); re-render the surface as shiny with phong shading and lighting from the east (the default of `camlight` lights surfaces from over the viewer's right shoulder):

```
material shiny;camlight;lighting phong
```

8 Finally, remove white space and re-render the figure in perspective mode:

```
axis tight; set(gca,'Projection','Perspective')
```

Here is the mapped result:



Draping a Geolocated Grid on Regular Data Grid via Texture Mapping

The second way to combine a regular and a geolocated data grid is to construct a regular data grid of your geolocated data grid's *z*-data. This approach has the advantage that more computational functions are available for regular data grids than for geolocated ones. Another aspect is that the color and elevation grids do not have to be the same size. If the resolutions of the two are different, you can create the surface as a three-dimensional elevation map and later apply the colors as a texture map. You do this by setting the surface `Cdata` property to contain the color matrix, and setting the surface face color to `'TextureMap'`.

In the following steps, you create a new regular data grid that covers the region of the geolocated data grid, then embed the color data values into the new matrix. The new matrix might need to have somewhat lower resolution than the original, to ensure that every cell in the new map receives a value.

1 Clear the workspace and load the `topo` and terrain data from `mapmtx`:

```
clear; load topo;
load mapmtx lt1
load mapmtx lg1
load mapmtx map2
```

2 Identify the geographic limits of one of the `mapmtx` geolocated grids:

```
latlim = [min(lt1(:)) max(lt1(:))];
lonlim = [min(lg1(:)) max(lg1(:))];
```

3 Trim the `topo` data to the rectangular region enclosing the smaller grid:

```
[topo1,topo1ref] = maptrims(topo,topolegend,latlim,lonlim);
```

4 Create a regular grid filled with `NaN`s to receive texture data:

```
[curve1,curve1ref] = nanm(latlim,lonlim,.5);
```

The last parameter establishes the grid at 1/.5 cells per degree.

5 Use `imbedm` to embed values from `map1` into the `curve1` grid; the values are the discrete Laplacian transform (the difference between each element of the `map1` grid and the average of its four orthogonal neighbors):

```
curve1 = imbedm(lt1,lg1,del2(map1),curve1,curve1ref);
```

6 Set up a map axes with the Miller projection and use `meshm` to draw the `topo1` extract of the `topo` DEM:

```
figure; axesm miller
h = meshm(topo1,topo1ref,size(topo1),topo1);
```

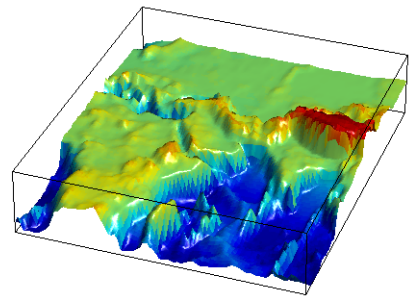
7 Render the figure as a 3-D view from a 20° azimuth and 30° altitude, and exaggerate the vertical dimension by a factor of 200:

```
view(20,30); daspectm('m',200)
```

8 Light the view and render with Phong shading in perspective:

```
material shiny; camlight; lighting phong
axis tight; set(gca,'Projection','Perspective')
```

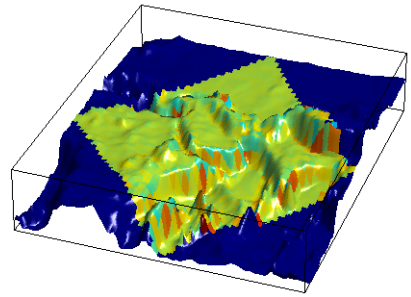
So far, both the surface relief and coloring represent topographic elevation, and appear as follows:



9 Now apply the `curve1` matrix as a texture map directly to the figure using the `set` function:

```
set(h,'Cdata',curve1,'FaceColor','TextureMap')
```

The area originally covered by the `[lt1, lg1, map1]` geolocated data grid, and recoded via the Laplacian transform as `curve1` now controls color symbolism, with the `NaN`-coded outside cells rendered in black.



Working with the Globe Display

The *Globe display* is a three-dimensional view of geospatial data capable of mapping terrain relief or other data for an entire planet viewed from space. Its underlying transformation maps latitude, longitude, and elevation to a three-dimensional Cartesian frame. All projections in the Mapping Toolbox transform latitudes and longitudes to map *x*- and *y*-coordinates. The `globe` function is special because it can render relative relief of elevations above, below or on a sphere. In Earth-centered Cartesian *(x,y,z)* coordinates, *z* not an optional elevation; rather, it is an axis in Cartesian three-space. `globe` is useful for geospatial applications which require three-dimensional relationships between objects to be maintained, such as when one simulates flybys, and or views planets as they rotate.

The Globe display is based on a *coordinate transformation*, and is not a map projection. Note that while it has none of the distortions inherent in planar projections, it is a three-dimensional model of a planet that cannot be displayed without distortion or in its entirety. That is, in order to render the globe in a figure window, either a perspective or orthographic transformation must be applied, both of which necessarily involve setting a viewpoint, hiding the back side, and distortions of shape, scale, and angles.

The globe transform is applied only to the sphere, not to ellipsoids of rotation. However, you are free to impose some flattening on the figure axes by changing the aspect ratio.

The Globe Display Compared with the Orthographic Projection

The following example illustrates differences between the two-dimensional orthographic projection, which looks spherical but is really flat, and the three-dimensional globe display. You use the **Rotate 3D** tool to manipulate the display.

1. First load the `topo` data set and render it with an orthographic map projection:

   ```
   load topo
   axesm ortho; framem
   meshm(topo,topolegend);demcmap(topo)
   ```

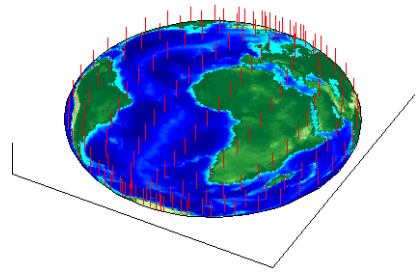
2. View the map obliquely:

   ```
   view(3); daspectm('m',1)
   ```

3 You can view it in 3-D from any perspective, even from underneath. To help visualize this, define a geolocated data grid with `meshgrat`, populate it with a constant *z*-value, and render it as a stem plot with `stem3m`:

```
[latgrat,longrat] = meshgrat(topo,topolegend,[20 20]);
stem3m(latgrat,longrat,500000*ones(size(latgrat)),'r')
```

Use the **Rotate 3D** tool on the figure window toolbar to change your viewpoint. You see that no matter how you position the view, you are looking at a disc with stems protruding perpendicularly. Here is the type of view you can see:



4 Now create another figure using the globe transform rather than orthographic projection:

```
figure
axesm('globe','Geoid',almanac('earth','radius','m'))
```

5 Display the `topo` surface in this figure and view it in 3-D:

```
meshm(topo,topolegend); demcmap(topo)
view(3)
```

6 Also include the stem plot to visualize the difference in surface normals on a sphere:

```
stem3m(latgrat,longrat,500000*ones(size(latgrat)),'r')
```

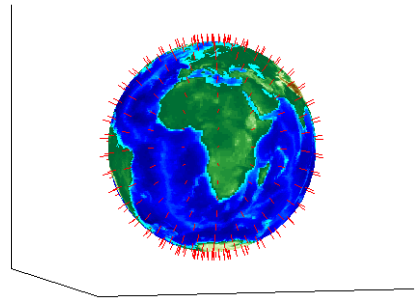
7 You can apply lighting to the display, but its location is fixed, and does not move as the camera position is shifted:

```
camlight('headlight','infinite')
```

8 If you prefer a more unobstructed view, you can hide the 3-D axes:

```
set(gca,'Box','off')
```

Here is a representative view using the Globe dispaly without lighting:



For additional details, see the reference pages for `view`, `camlight`, `meshgrat`, and `stem3m`.

Using Opacity and Transparency in Globe Displays

Because Globe displays depict 3-D objects, you can see into and through them as long as no opaque surfaces (e.g., patches or surfaces) obscure your view. This can be particularly disorienting for point and line data, because features on the back side of the world are reversed and can overlay features on the front side.

Here is one way to create an opaque surface over which you can display line and point data:

1. Create a figure and set up a Globe display:

   ```
   figure; axesm('globe')
   ```

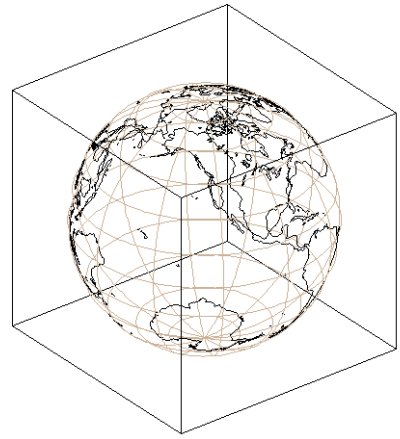
2. Draw a graticule in a light color, slightly raised from the surface:

   ```
   gridm('GLineStyle','-','Gcolor',[.8 .7 .6],'Galtitude', .02)
   ```

3. Load and plot the coast data in black, and set up a 3-D perspective:

   ```
   load coast
   plot3m(lat,long,.01,'k')
   view(3)
   ```

   The 3D view looks like this:



4 Use the **Rotate 3D** tool on the figure's toolbar to rotate the view. Note how confusing the display is because of its transparency.

5 Make a uniform 1-by-1-degree grid and a referencing vector for it:

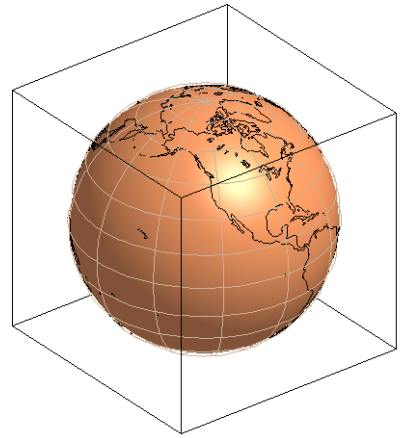
```
base = zeros(180,360); baseref = [1 90 0];
```

6 Render the grid onto the globe, color it copper, light it from camera right, and make the surface reflect more light:

```
hs = meshm(base,baseref,size(base));
colormap copper
camlight right
material([.8 .9 .4])
```

Note Another way to make the surface of the globe one color is to change the `FaceColor` property of a displayed surface mesh (e.g., `topo`).

The display (if you haven't rotated it) looks like this:



When you manually rotate this map, its movement can be jerky due to the number of vectors that must be redisplayed. In any position, however, the copper surface effectively hides all lines on the back side of the globe.

Note The technique of using a uniform surface to hide rear-facing lines has limitations for the display of patch symbolism (filled polygons). As patch polygons are represented as planar, in three-space the interiors of large patches can intersect the spherical surface mesh, allowing its symbolism to show through.

Over-the-Horizon 3-D Views Using Camera Positioning Functions

You can create dramatic 3-D views using the globe display. The `camtargm` and `camposm` functions (Mapping Toolbox versions of `camtarget` and `campos`) enable you to position focal point and a viewpoint, respectively, in geographic coordinates, so you do not need to deal with 3-D Cartesian figure coordinates.

In this exercise, you display national boundaries from the `worldlo` data set over topographic relief, and then view the globe from above Washington, D.C., looking toward Moscow, Russia.

1 Set up a globe display and obtain topographic data for the map:

```
clear; axesm globe
```

```
load topo
```

2 Display `topo` without the vertical component (by omitting the fourth argument to `meshm`):

```
hs = meshm(topo,topolegend,size(topo)); demcmap(topo);
```

The default view is from above the North Pole with the central meridian running parallel to the *x*-axis.

3 Use `displaym` to pull in national outlines stored in the `POline` geostruct within the `worldlo` data set and plot them in light grey:

```
hl = displaym(worldlo('POline')); set(hl,'color',[.7 .7 .7])
```

4 Use `extractm` to identify the coordinate locations for Moscow and Washington from the `worldlo` gazetteer:

```
[tlat,tlon] = extractm(worldlo('gazette'),'Moscow');
[plat,plon] = extractm(worldlo('gazette'),'Washington');
```

5 Create a great circle track to connect Washington with Moscow and plot it in red:

```
[latc,lonc] = track2('gc',tlat,tlon,plat,plon);
plotm(latc,lonc,'r')
```

6 Point the camera at Moscow. Wherever the camera is subsequently moved, it always looks toward `[tlat,tlon]`:

```
camtargm(tlat,tlon,0)
```

7 Station the camera at `(plat,plon,3)`. The third argument is an altitude in Earth radii:

```
camposm(plat,plon,3)
```

8 Establish the camera up vector with the camera target's coordinates. The great circle joining Washington and Moscow now runs vertically:

```
camupm(tlat,tlon)
```

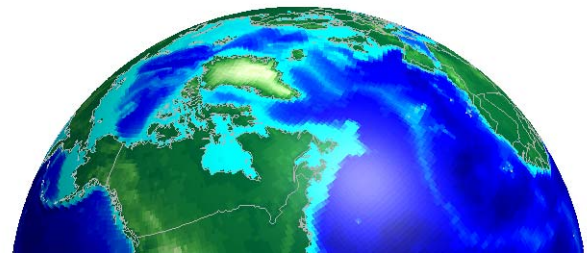
9 Set the field of view for the camera to 20° for the final view:

```
camupm(20)
```

10 Add a light, specify a relatively nonreflective surface material, and hide the map background:

```
camlight; material(0.6*[ 1 1 1])
hidem(gca)
```

Here is the final view:



For additional information, see the reference pages for `displaym`, `extractm`, `camtargm`, `camposm`, `camupm`, `camupm`, `globe`, and `camlight`.

Displaying a Rotating Globe

Because the globe display can be viewed from any angle without the need to recompute a projection, you can easily animate it to produce a rotating globe. If the displayed data is simple enough, such animations can be edrawn at relatively fast rates. In this exercise, you progressively add or replace features on a globe display and rotate it under the control of an M-file that resets the view to rotate the globe from west to east in one-degree increments.

1 In the MATLAB editor, create an M-file containing the following code:

```
% spin.m: Rotates a view around the equator one revolution
% in 5-degree steps. Negative step makes it rotate normally
% (west-to-east).
for i=360:-5:0
```

```
    view(i,0);
    drawnow
end
```

Save this as `spin.m` in your current directory or on the MATLAB path. Note that the azimuth parameter for the figure does not have the same origin as geographic azimuth: it is 90 degrees to the west.

2 Set up a Globe display with a graticule, as follows:

```
axesm('globe','Grid','on','Gcolor',[.7 .8 .9],'GlineStyle','-')
```

The view is from above the North Pole.

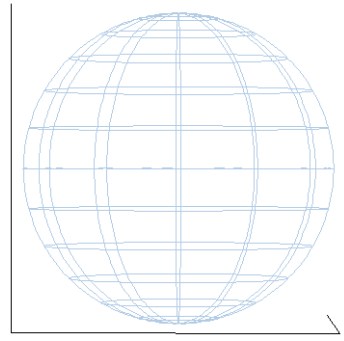
3 Hide the edges of the figure's box, and view it in perspective rather than orthographically (the default perspective):

```
set(gca, 'Box','off', 'Projection','perspective')
```

4 Spin the globe one revolution with your M-file:

```
spin
```

The globe spins rapidly. The last position looks like this:



5 To make the globe opaque, create a sea-level data grid as you did for the previous exercise, "Using Opacity and Transparency in Globe Displays" on page 5-45:

```
base = zeros(180,360); baseref = [1 90 0];
hs = meshm(base,baseref,size(base));
```

```
colormap copper
```

The globe now is a uniform dark copper color with the grid overlaid.

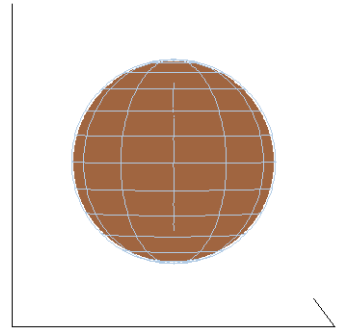
6 Pop up the grid so it appears to float 2.5% above the surface:

```
setm(gca, 'Galtitude',0.025)
```

7 Spin the globe again:

```
spin
```

The motion is much slower, due to the need to re-render the 180-by-360 mesh: The last frame looks like this:



8 Get ready to replace the uniform sphere with topographic relief:

```
clmo(hs)
load topo
```

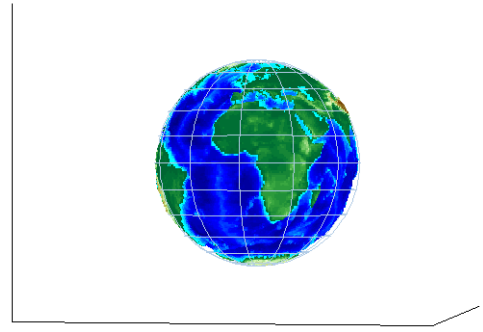
9 Scale the elevations to have an exaggeration of 50 (in units of Earth radii) and plot the surface:

```
topo = topo / (almanac('earth','radius')* 20);
hs = meshm(topo,topolegend,size(topo),topo);
demcmap(topo)
```

10 Spin again:

```
spin
```

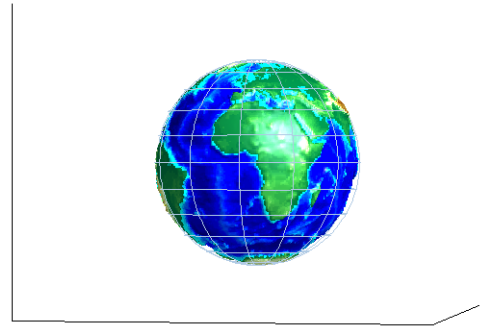
Here is a representative view, showing the Himalayas rising on the Eastern limb of the planet and the Andes on the Western limb:



11 You can apply lighting as well, which will shift as the planet rotates. Try the following settings, or experiment with others:

```
camlight right
lighting phong;
material ([.7, .9, .8])
```

Here is the illuminated version of the preceding view:



For additional information, see the reference pages for `globe`, `camlight`, and `view`.

6

Customizing and Printing Maps

Using the Mapping Toolbox you can place several types of map annotations in addition to those previously described (tracks, circles, grids, meridian and parallel labels, and other text objects). The following sections describe some of this additional functionality for defining annotation elements and for making a variety of thematic maps.

Inset Maps

Inset maps are often used to display widely separated areas, generally at the same scale, or to place a map in context by including overviews at smaller scales. You can create inset maps by nesting multiple axes in a figure and defining appropriate map projections for each. To ensure that the scale of each of the maps is the same, use `axesscale` to resize them. As an example, create an inset map of California at the same scale as the map of South America, to relate the size of that continent to a more familiar region:

1. Begin by defining a map of South America using `worldmap`:

   ```
   close all; clear all; h1 = worldmap('south america');
   setm(h1,'FFaceColor','w')  % set the frame fill to white
   ```

2. Place axes for an inset in the lower middle, and project a line map of California:

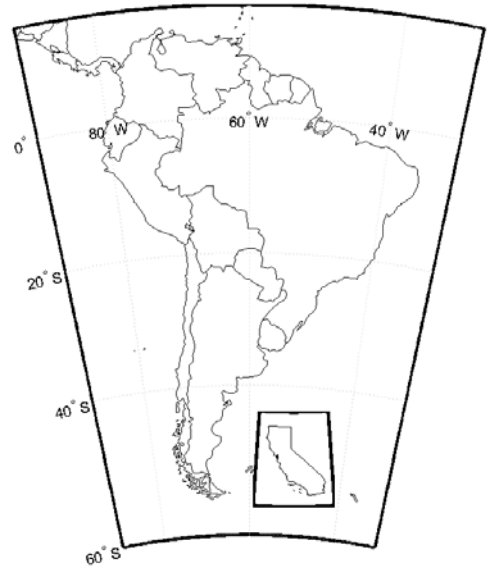
   ```
   h2 = axes('pos',[.5 .2 .1 .1]);
   usamap('californiaonly','lineonly')
   ```

3. Set the frame fill color and set the labels

   ```
   setm(h2,'FFaceColor','w')
   mlabel; plabel; gridm     % toggle off
   ```

4. Set the scale for child axes to that of the parent axes, and hide map borders:

   ```
   axesscale(h1)
   hidem([h1 h2])
   ```



Note that the Mapping Toolbox chose a different projection and appropriate parameters for each region based on its location and shape. You can override these choices to make the two projections the same.

5 Find out what map projections are used, and then make South America's projection the same as California's:

```
getm(h1, 'mapprojection')
ans =
    eqdconic
getm(h2, 'mapprojection')
ans =
    lambert
setm(h1, 'mapprojection', getm(h2, 'mapprojection'))
```

Note that the parameters for South America defaulted properly (those appropriate for California were not used).

6 Finally, experiment with changing properties of the inset, such as its color:

```
setm(h2, 'ffacecolor', 'y')
```

Graphic Scales

Graphic scale elements are used to provide indications of size even more frequently than insets are. These are ruler-like objects that show distances on the ground at the nominal scale of the projection. You can use the `scaleruler` function to add a graphic scale to the current map. You can check and modify the `scaleruler` settings using `getm` and `setm`. You can also move the graphic scale to a new position by dragging its baseline.

Try this by creating a map, adding a graphic scale with the default settings, and shifting its location. Then add a second scale in nautical miles, and change the tick mark style and direction:

1 Plot a patch map of Guatemala:

```
clear all; close all;
worldmap('lo','guatemala','patchonly')
```

2 Add a default graphic scale and then bump it up:

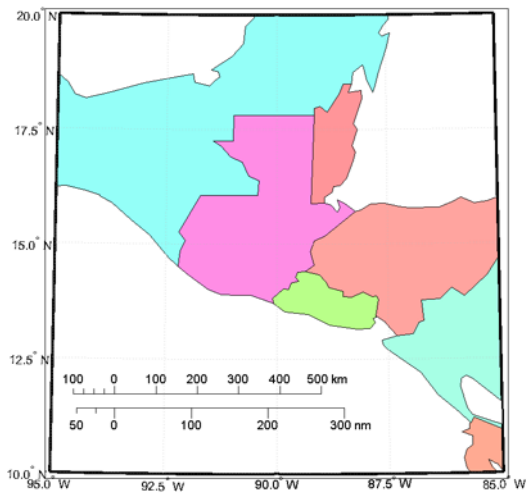
```
scaleruler
setm(handlem('scaleruler1'),'YLoc',.205)
```

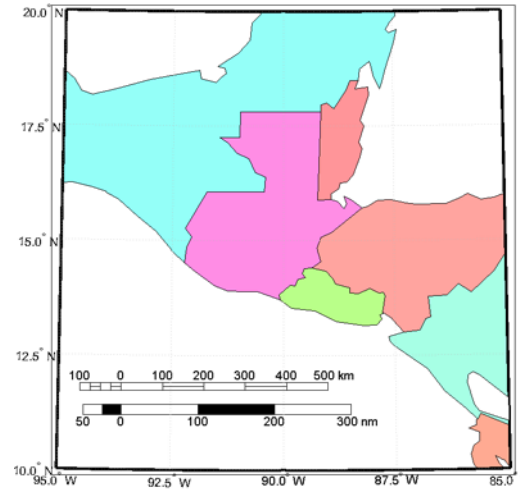
3 Place a second graphic scale. It will look wrong until you adjust it manually:

```
scaleruler('units','nm')
setm(handlem('scaleruler2'),'MajorTick',0:100:300,...
     'MinorTick',0:25:50,'TickDir','down',...
     'MajorTickLength',km2nm(25),...
     'MinorTickLength',km2nm(12.5))
```

4 Experiment with the two other ruler styles available:

```
setm(handlem('scaleruler1'),'RulerStyle','lines')
setm(handlem('scaleruler2'),'RulerStyle','patches')
```


20.0° N
17.5° N
15.0° N
12.5° N
10.0° N
95.0° W
92.5° W
90.0° W
87.5° W
85.0° W
100 0 100 200 300 400 500 km
50 0 100 200 300 nm


20.0° N
17.5° N
15.0° N
12.5° N
10.0° N
95.0° W
92.5° W
90.0° W
87.5° W
85.0° W
100 0 100 200 300 400 500 km
50 0 100 200 300 nm

North Arrows

The north arrow element provides the orientation of a map by pointing to the geographic North Pole. You can use the `northarrow` function to display a symbol indicating the direction due north on the current map. The north arrow symbol can be repositioned by clicking and dragging its icon. The orientation of the north arrow is computed, and does not need manual adjustment no matter where you move the symbol. **Ctrl**+clicking on the icon creates an input dialog box with which you can change the location of the north arrow:

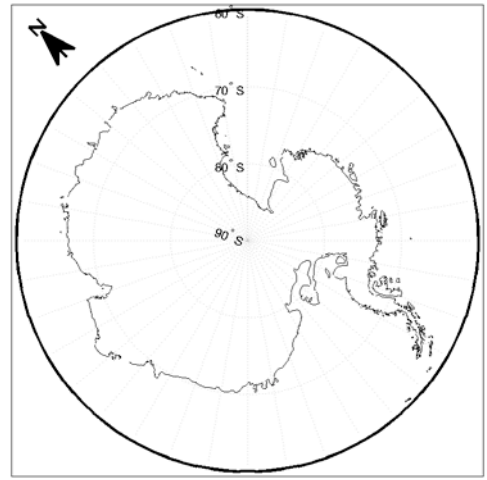
1. To illustrate the use of north arrows, create a map centered at the South Pole and add a north arrow symbol at a specified geographic position.

   ```
   close all; clear all;
   figure; worldmap('south pole')
   northarrow('latitude',-57,'longitude',135);
   ```

2. Click on and drag the north arrow symbol to another corner of the map. Note that it always points to the North Pole.

3. Drag the north arrow back to the top left corner.

4. Right-click or **Ctrl**+click on the north arrow. The **Inputs for North Arrow** dialog opens which lets you specify the line weight, edge and fill colors, and relative size of the arrow. Set the `LineWidth` to `2` and click `OK`. Here is what the map now looks like:



5 Set some north arrow properties manually, just to get a feel for them:

```
h = handlem('NorthArrow');
set(h,'FaceColor',[1.000 0.8431 0.0000],...
'EdgeColor',[0.0100 0.0100 0.9000])
```

6 Make three more north arrows, to show that from the South Pole, every direction is north.

```
northarrow('latitude',-57,'longitude',45)
northarrow('latitude',-57,'longitude',225)
northarrow('latitude',-57,'longitude',315)
```

Note North arrows are created as objects in the MATLAB axes (and thus have Cartesian coordinates), not as 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.

Thematic Maps

Rather than showing physical features on the ground, such as shorelines, roads, settlements, topography, and vegetation, a thematic map displays quantified facts (a "theme"), such as statistics for a region or sets of regions. Examples include the locations of traffic accidents in a city, or election results by state. Thematic maps have a wide vocabulary of cartographic symbols, such as point symbols, dot distributions, "quiver" vectors, isolines, colored zones, raised prisms, and continuous 3-D surfaces. The Mapping Toolbox provides functions to produce most of these types of map symbology.

Choropleth Maps

The most familiar form of thematic map is probably the choropleth map (from the Greek *choros*, for place, and *plethos*, for magnitude). Often used to present data in newspapers, magazines, and reports, choropleth maps fill geographic zones (such as countries or states, but also matrices) with colors and/or patterns to represent nominal, ordinal, or cardinal data values. As there are usually more possible data values than unique symbols or colors capable of differentiating them, choropleth maps usually classify their data into value ranges.

The Mapping Toolbox uses patch objects to construct choropleth maps. It assigns a color to each patch face to represent a specified variable, one value per patch. When the variable is scalar (as opposed to nominal) it generally represents a density (such as population per unit area), intensity (such as income per family), or incidence rate (such as fatalities per thousand persons). It can also convey extensive measurements or counts (such as electoral votes per state) if used carefully.

To make a choropleth map you need to input or compute a vector of values, one for each patch in a vector data set. Symbolizing such data values with the Mapping Toolbox is straightforward. It involves assigning the data values to the `CData` property of a set of patches, and then setting up a colormap with an appropriate color scheme and range. Colormaps usually map N or fewer values (for N patches) to M colors. M can be any number between 2 and N, but typically ranges between 5 and 10.

In the following example, patches representing the 50 states of the U.S. (and the District of Columbia) are displayed and colored according to the surface areas calculated by the `areaint` function. An equal-area projection is

appropriate for this and other choropleth maps. This is because data is often computed or normalized over the patches being displayed, and thus area distortion should be minimized, even at the expense of shape distortion:

1. Set up the U.S. state patch data:

   ```
   close all; clear all; load usalo
   ```

   This data set includes patch data for individual states, the United States, and its Great Lakes.

2. Set up map axes with a projection suitable to display all 50 states with equal areas, a graticule, and grid labels:

   ```
   axesm('MapProjection','eqaconic','MapParallels',[],...
         'MapLatLimit',[15 75],'MapLonLimit',[-175 -60],...
         'MLineLocation',15,'MLabelParallel','south',...
         'MeridianLabel','on','ParallelLabel','on',...
         'GLineStyle','-','GColor',0.5*[1 1 1],...
         'Grid','on','Frame','on')
   ```

3. Draw the basic patch map of the polygons (patches) in the `state` structure:

   ```
   displaym(state)
   ```

4. The colors assigned to patches are based on the default colormap and the ordering of patches, which is alphabetical. See this for yourself as you set up a cell array containing their names:

   ```
   tags = {state.tag}
   tags =
     Columns 1 through 5
       'Alabama'    'Alaska'    'Arizona'    'Arkansas'    'California'
       ...
   ```

5. Choose an ellipsoid for computing spherical area (`geoid` defaults to `wgs80`):

   ```
   refvec = almanac('earth', 'geoid');
   ```

6. Use a `for` loop to compute areas for all U.S. states plus D.C. (this loop could also include calculations to classify values for areas into intervals of varying size):

```
maxarea = 0.0;
for i=1:length(state)
      lat = state(i).lat;
      long = state(i).long;
      surfarea = sum(areaint(lat, long, refvec));
      set(handlem(tags{i}),'CData', surfarea);
      maxarea = max(surfarea, maxarea);
end
```

7 Set the range of value space for the colormap:

```
caxis([0 maxarea])
```

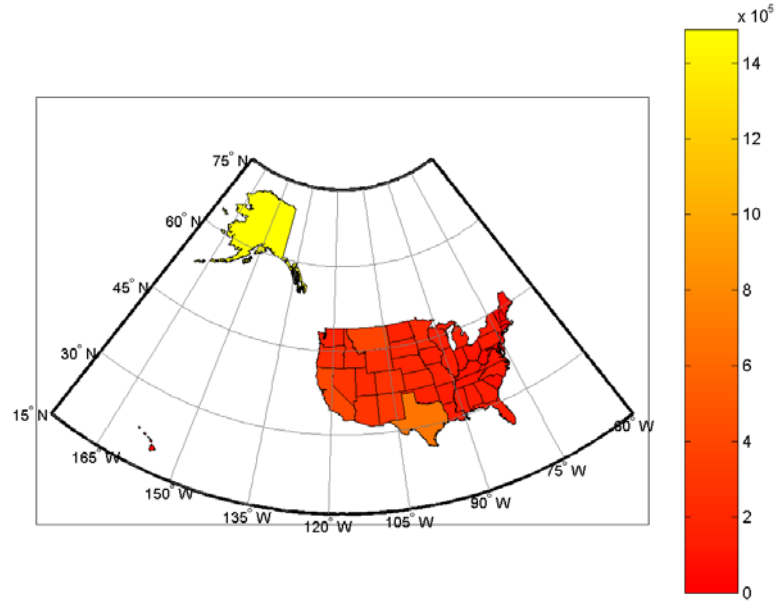
8 Show a colorbar as a key to the symbology, in its default location. This legend relates patch color to area in km^2:

```
colorbar
```

9 Choose a colormap. Here use a monotonic change from brown to yellow.

```
colormap('autumn')
```

10 The map is mostly red, as the following figure shows. Experiment with other colormaps. Some names of predefined colormaps are `autumn`, `cool`, `copper`, `gray`, `pink`, and `jet`.



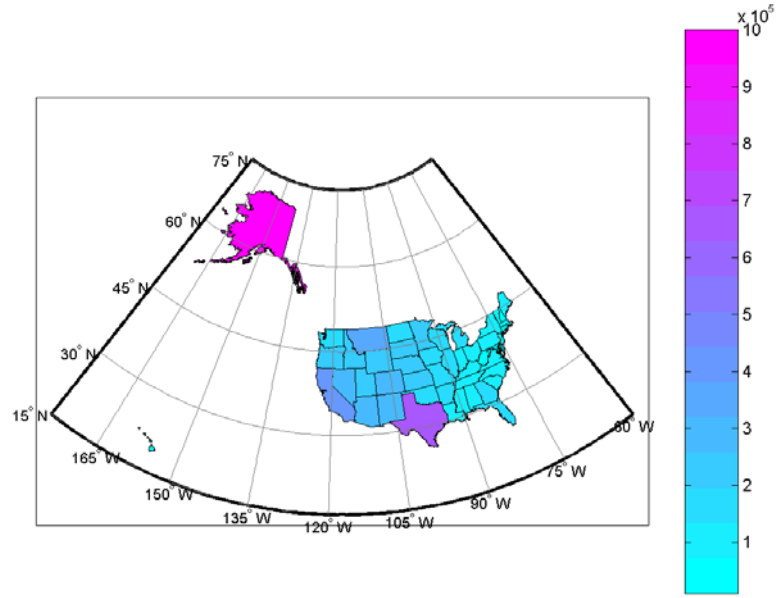
Note that while the color scale varies continuously, many states appear to be the same color. This is because of the skewed distribution of state areas. One way to differentiate the symbology is to clamp the lower end (because the smallest patches, such as District of Columbia and Rhode Island, are much smaller than average) and the upper end (because Alaska's area is so much larger than that of any other state).

11 Issue the following commands to reset the *z*-axis (color range), change the colormap to one that has more hues and a smaller number of steps, and redraw the colorbar to display the new value range:

```
caxis([10000 1000000])
colormap(cool(16))
colorbar
```

Note how you can specify the size of a colormap with the `colormap` syntax used above. Be aware that, because you clamped the value range, the numeric limits of the colorbar overstate the minimum area and understate the maximum area. However, the map give much more information overall

because more states have distinct symbology, as the resulting map below depicts.



Special Thematic Mapping Functions

In addition to choropleth maps, the Mapping Toolbox provides other display and symbology functions. These include the following:

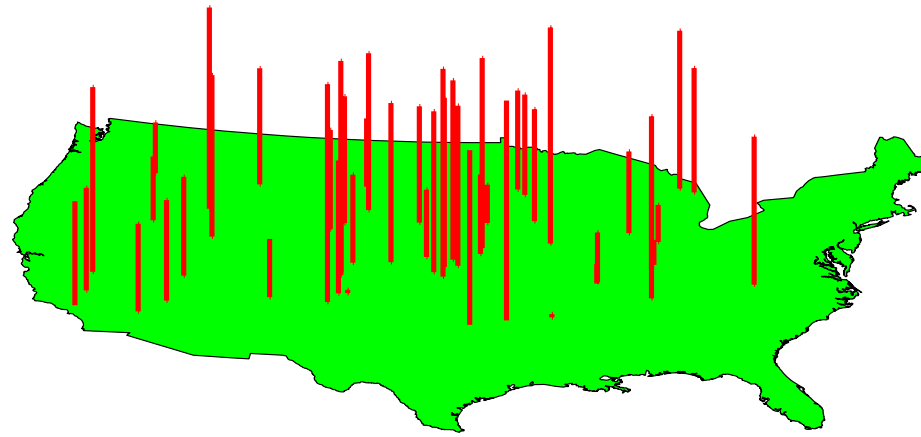
Function	Used For
`cometm`	Traces (animates) vectors slowly from a comet head
`comet3m`	Traces (animates) vectors in 3-D slowly from a comet head
`quiverm`	Plots directed vectors in 2-D from specified latitudes and longitudes with lengths also specified as latitudes and longitudes

Function	Used For
`quiver3m`	Plots directed vectors in 3-D from specified latitudes, longitudes, and altitudes with lengths also specified as latitudes and longitudes, and altitudes
`scatterm`	Draws fixed or proportional symbol maps for each point in a vector with specified marker symbol. Similar maps can be generated using `geoshow` and `mapshow` using appropriate symbol specifications ("symbolspecs").
`stem3m`	Projects a 3-D stem plot map on the current map axes

The `cometm` and `quiverm` functions operate like their MATLAB counterparts `comet` and `quiver`. The `stem3m` function allows you to display geographic bar graphs. Like the MATLAB `scatter` function, the `scatterm` function allows you to display a thematic map with proportionally sized symbols. The `tissot` function calculates and displays Tissot Indicatrices, which graphically portray the shape distortions of any map projection. For more information on these capabilities, consult the descriptions of these functions in the reference pages.

Stem Maps

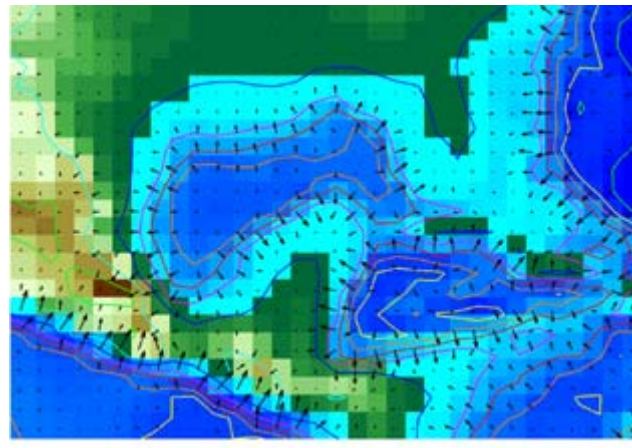
Stem plots are 3-D geographic bar graphs portraying numeric attributes at point locations, usually on vector base maps. Below is an example of a stem plot over a map of the continental United States. The bars could represent anything from selected city populations to the number of units of a product purchased at each location:



Contour Maps

Contour and quiver plots can be useful in analyzing matrix data. In the following example, contour elevation lines have been drawn over a topographical map. The region displayed is the Gulf of Mexico, obtained from the `topo` matrix. Quiver plots have been added to visualize the gradient of the topographical matrix.

Here is the displayed map:



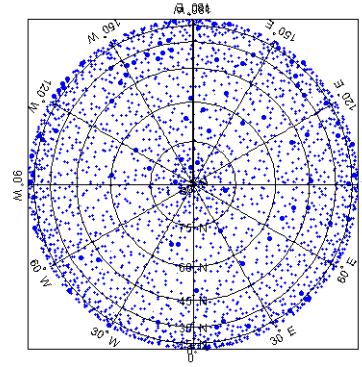
Scatter Maps

The `scatterm` function plots symbols at specified point locations, like the MATLAB `scatter` function. If the symbols are small and inconspicuous and do not vary in size, the result is a *dot-distribution map*. If the symbols vary in size and/or shape according to a vector of attribute values, the result is a *proportional symbol map*.

Below is an example of using `scatterm` to create a star chart of the northern sky. The stars are represented by filled circles whose size is proportional to visual magnitude. To execute the following commands, select them all by dragging over the list in the Help Browser, then click the right mouse button and choose `Evaluate Selection`:

```
close all; clear all
load stars
% Set all visual magnitude zero values to eps
index = find(vmag <= 0);
vmag(index) = eps;
% View the sky orthographically
axesm('MapProjection','ortho','Origin',[90 0])
setm(gca,'FLatLimit',[90 0],'MapLatLimit',[90 0])
gridm on
setm(gca,'LabelFormat','compass','LabelRotation','on')
setm(gca,'MLabelParallel',0,'PLabelMeridian',0)
```

```
setm(gca,'MeridianLabel','on','ParallelLabel','on')
setm(gca,'GlineStyle','-')
% Make scatterplot of vmag data with blue filled circles
scatterm(lat, long, vmag, 'b', 'filled')
```



Using Cartesian MATLAB Display Functions

If you cannot find a Mapping Toolbox display function that does what you need, you might be able to use a non-mapping MATLAB function. When placing graphic objects on a map axes, you can use the MATLAB function to add the graphic objects to the display, using latitude and longitude as x and y, and then project the data afterwards.

Note Before applying non mapping functions to geodata, you should take into consideration that performing Cartesian geometric operations on geographic coordinates can yield inaccurate results when the data cover large regions of a planet or lie near one of its poles.

Example 1: Triangulating Data Points

The Mapping Toolbox does not have a function that displays a triangulated surface from random data points, a structure generally known as a *triangulated irregular network* (TIN). However, MATLAB does have a function to create *Delaunay triangles*, a method that is often used to form TINs from projected point coordinate data. Explore triangulating some point data and bringing the result into the Mapping Toolbox:

1. Use the `seamount` data provided with MATLAB:

   ```
   clear all; close all; load seamount
   ```

2. Determine the bounds of the coordinates and add a degree of white space:

   ```
   latlim = [min(y)-.5 max(y)+.5];
   lonlim = [min(x)-.5 max(x)+.5];
   ```

3. Create map axes to contain the seamount region (`worldmap` selects a projection for you):

   ```
   worldmap(latlim,lonlim,'none')
   ```

4. Create a Delaunay triangulation of x and y (longitude and latitude):

   ```
   tri = delaunay(y,x);
   ```

5 Generate a 3-D surface that combines the triangulation and *z*-values:

```
h = trisurf(tri,y,x,z);
```

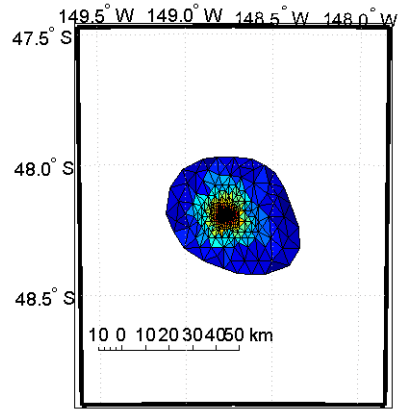
6 Map the surface onto the axes by projecting to the *x-y* plane (`project` is a Mapping Toolbox function especially for this purpose):

```
project(h,'yx')
```

Note that even though the triangulated surface appears to be part of the map, it does not have a geostruct at this point (see "Mapping Toolbox Geographic Data Structures" on page 2-16).

7 Add a default graphic scale to the display:

```
scaleruler
```



If, as in this example, the displayed objects are already in the right place and do not need to be projected, you can trim them to the map frame and convert them to mapped objects (having geostructs) using `trimcart` and `makemapped`. They can then be manipulated as if they had been created with map display functions.

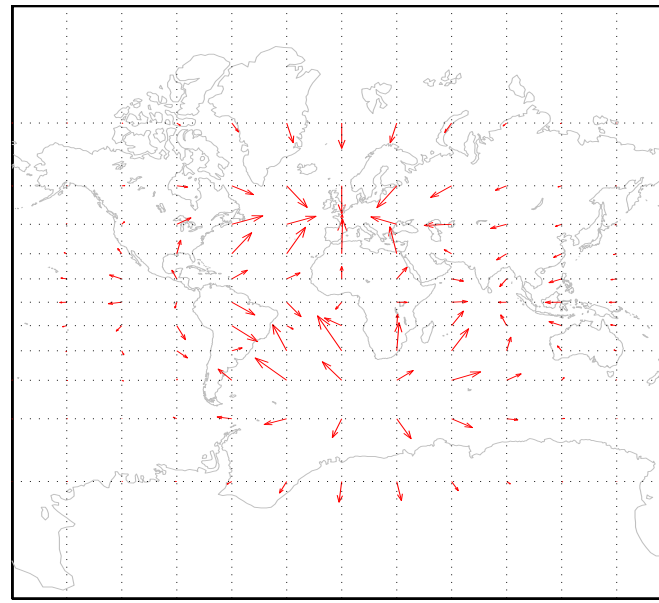
Example 2: Constructing Quiver Maps

As was briefly described for text objects in "Projected and Unprojected Graphic Objects" on page 4-14, you can also combine Mapping Toolbox and MATLAB

functions to mix spherical and Cartesian coordinates. An example would be a quiver plot (sometimes known as a *vector field*) in which the locations of the vectors are geographic, but the lengths, being specified by attributes, are not. In that case, you can use Mapping Toolbox projection calculations and MATLAB graphics functions. Cylindrical projections are the simplest to use because North is up, South is down, and East and West are on an orthogonal axis.

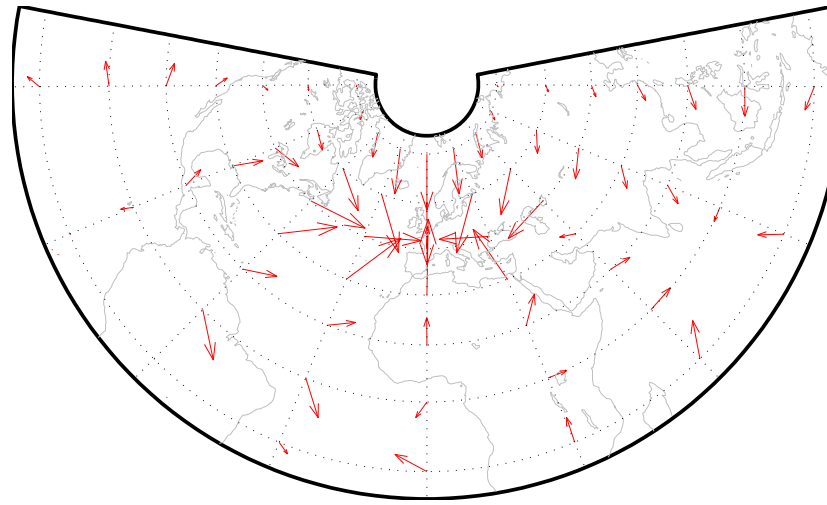
In this example, you will impose a quiver map of the slope of a surface on a world map. The surface is a Gaussian field generated by the MATLAB `peaks` function.

```
figure; axesm mercator; framem; gridm
load coast
plotm(lat,long,'color',[.75 .75 .75])

[u,v] = gradient(peaks(13)/10);
[lat,lon] = meshgrat(-90:15:90,-180:30:180);
[x,y] = mfwdtran(lat,lon);

h = quiver(x,y,u,v,.2,'r');
trimcart(h)
```



An extra step might be required for noncylindrical projections. In these projections, compass directions vary with location. To make the directions agree with the map grid, vectors should be rotated to bring them into alignment. This can be done with the vector transformation function `vfwdtran`. Consider the same data displayed on a conic projection.

```
load coast; figure
axesm('lambert','MapLatLimit',[-20 80])
framem; gridm
plotm(lat,long,'color',[.75 .75 .75])

[x,y] = mfwdtran(lat,lon);
thproj = deg2rad(vfwdtran(lat,lon,90*ones(size(lat))));
[th,r] = cart2pol(u,v);
[uproj,vproj] = pol2cart(th+thproj,r);

h = quiver(x,y,uproj,vproj,0,'r') ;
trimcart(h)
```



Conformal projections, such as this Lambert conformal conic, are often the best choice for quiver displays. They preserve angles, ensuring that the difference between north and east will always be 90 degrees in projected coordinates.

Using Colormaps and Colorbars

Colormap for Terrain Data

In previous examples, the function `demcmap` was used to color several digital elevation model (DEM) topographic displays. This function creates colormaps appropriate to rendering DEMs, although it is certainly not limited to DEMs.

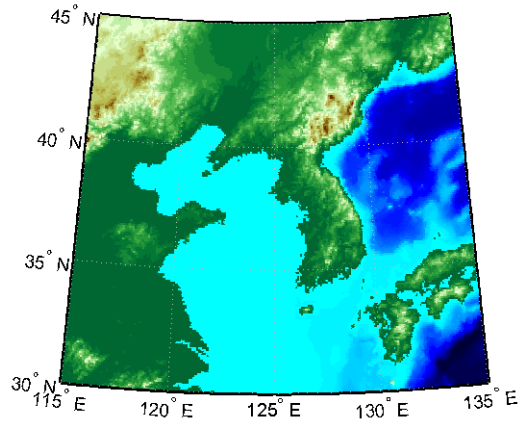
These colormaps, by default, have atlas-like colors varying with elevation or depth that properly preserve the land-sea interface. In cartography, such color schemes are called *hypsometric tints*:

1. Here you explore `demcmap` using the topographic data for the Korean peninsula provided in the `korea` data set:

   ```
   clear all; close all; load korea
   worldmap(map,maplegend,'meshonly')
   ```

2. The Korea DEM is displayed using the default colormap, which is inappropriate and causes the surface to be unrecognizable. Now apply the default DEM colormap and turn off the map frame:

   ```
   demcmap(map)
   hidem(gca)
   ```

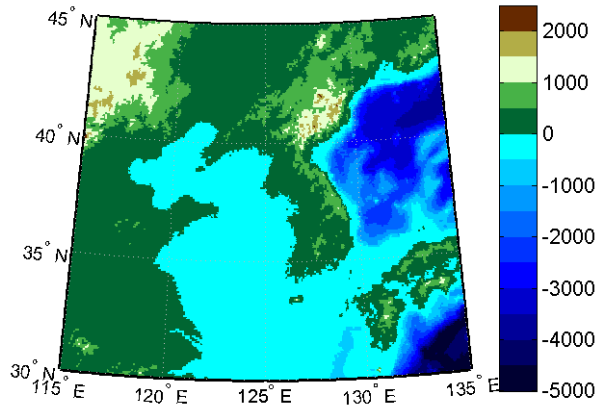


3. You can also make `demcmap` assign all altitudes within a particular range to the same color. This results in a quasi-contour map with breaks at a

constant interval. Now color this map using the same color scheme coarsened to display 500 meter bands:

```
demcmap('inc',map,500)
colorbar
```

Note that the first argument to `demcmap`, `'inc'`, indicates that the third argument should be interpreted as a value range. If you prefer, you could specify the desired number of colors with the third argument by setting the first argument to `'size'`.



Contour Colormaps

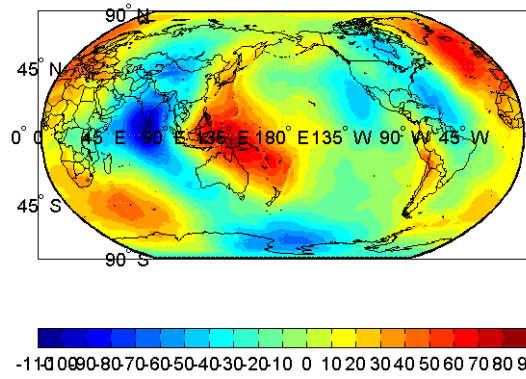
You can create colormaps that make surfaces look like contour maps for other types of data besides terrain. The `contourcmap` function creates a colormap that has color changes at a fixed value increment. Its required arguments are the increment value and the name of a colormap function. Optionally, you can also use `contourcmap` to add and label a colorbar similarly to the MATLAB `colorbar` function:

1 Explore `contourcmap` by loading the world `geoid` data set and rendering it with a default colormap:

```
load geoid
figure; worldmap(geoid,geoidlegend)
```

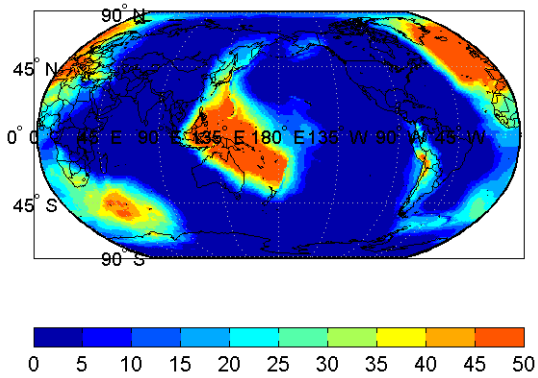
2 Use `contourcmap` to specify a contour interval of 10 (meters), and to place a colorbar beneath the map:

```
contourcmap(10,'jet','colorbar','on','location','horizontal')
```



3 If you want to render a restricted value range, you can enter a vector of evenly spaced values for the first argument. Here you specify a 5-meter interval and truncate symbology at 0 meters on the low end and 50 meters at the high end:

```
contourcmap([0:5:50],...
'jet','colorbar','on','location','horizontal')
```



Should you need to write a custom colormap function, for example, one that has irregular contour intervals, you can easily do so, but it should work like those provided with MATLAB.

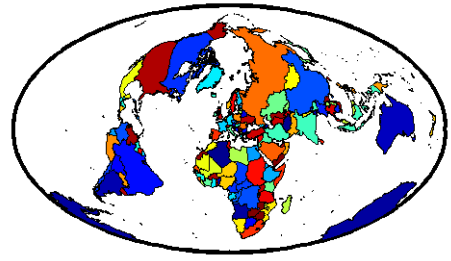
Colormaps for Political Maps

Political maps typically use muted, contrasting colors that make it easy to distinguish one country from its neighbors. You can create colormaps of this kind using the `polcmap` function. The `polcmap` function creates a colormap with randomly selected colors of all hues. Since the colors are random, if you don't like the result, execute `polcmap` again to generate a different colormap:

1. To explore political colormaps, display the `worldlo` data set as patches:

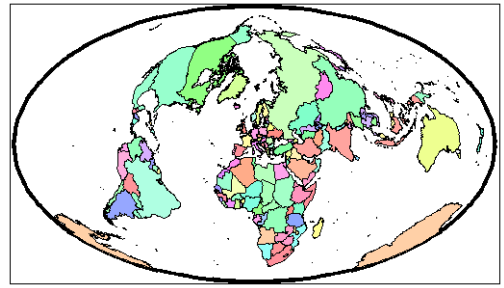
   ```
   figure; axesm bries
   displaym(worldlo('POpatch'))
   framem
   ```

   Note how garish the default coloring is.



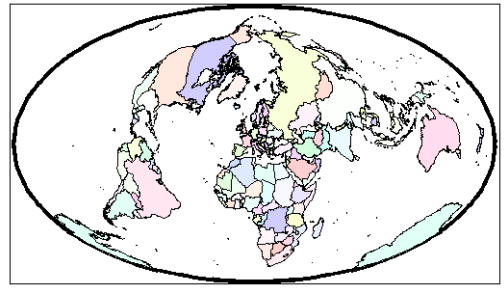
2. Use `polcmap` to randomly recolor the patches and expand the map to fill the frame:

   ```
   polcmap
   tightmap loose
   ```



3 The `polcmap` function can also control the number and saturation of colors. Reissue the command specifying 256 colors and a maximum saturation of 0.2. To ensure that the colormap is always the same, reset the seed on the MATLAB random number function using the `'state'` argument with a fixed value of your choice:

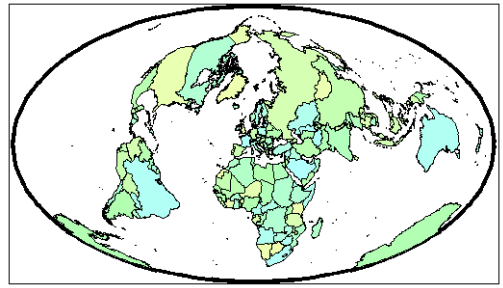
```
rand('state',0)
polcmap(256,.2)
```



4 For maximum control over the colors, specify the ranges of hues, saturations, and values. Use the same set of random color indices as before:

```
rand('state',0)
polcmap(256,[.2 .5],[.3 .3],[1 1])
```

Note that `polcmap` works with `displaym` but not with `geoshow` or `mapshow`.

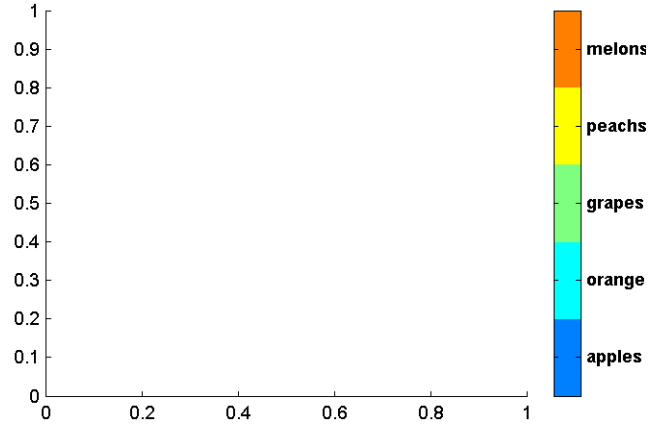


Note The famous Four Color theorem states that any political map can be colored to completely differentiate neighboring patches using only four colors. Experiment to find how many colors it takes to color neighbors differently with `polcmap`.

Labeling Colorbars

Political maps are an example of nominal data display. Many nominal data sets have names associated with a set of integer values, or consist of codes that identify values that are ordinal in nature (such as low, medium and high). The MATLAB function `lcolorbar` creates a colorbar having a text label aligned with each color. Nominal colorbars are customarily used only with small colormaps (where 10 categories or fewer are being displayed).

```
figure; colormap(jet(5))
labels = {'apples','oranges','grapes','peaches','melons'};
lcolorbar(labels,'fontweight','bold');
```



Editing Colorbars

Maps of nominal data often require colormaps with special colors for each index value. To avoid building such colormaps by hand, use the MATLAB GUI for colormaps, `colormapeditor`, or theMapping Toolbox GUI `cmapui`. The `cmapui` panel allows you to select color entries in a colormap one by one by clicking on the colorbar. To change a selected color's hue and saturation, drag the color **Marker** on the color wheel. To control the value (lightness) of the color in HSV space, drag the red **Slider**. Clicking the **Accept** button returns the modified colormap.

Printing Maps to Scale

Maps are often printed at a size that makes objects on paper a particular fraction of their real size. The linear ratio of the mapped to real object sizes is called *map scale*, and it is usually notated with a colon as "1:1,000,000" or "1:24,000". Another way of specifying scale is to call out the printed and real lengths, for example "1 inch = 1 mile."

You can specify the printed scale using the `paperscale` function. It modifies the size of the printed area on the page to match the scale. If the resulting dimensions are larger than your paper, you can reduce the amount of empty space around the map using `tightmap`, `zoom`, or `panzoom`, and by changing the axes position to fill the figure. This also reduces the amount of memory needed to print with the `zbuffer` (raster image) renderer. Be sure to set the paper scale last. For example,

```
set(gca,'Units','Normalized','Position',[0 0 1 1])
tightmap
paperscale(1,'in', 5,'miles')
```

The `paperscale` function also can take a scale denominator as its first and only argument. If you want the map to be printed at 1:20,000,000, type

```
paperscale(2e7)
```

To check the size and extent of text and the relative position of axes, use `previewmap`, which resizes the figure to the printed size.

```
previewmap
```

For more information on printing, see the "Basic Printing and Exporting" section of the MATLAB graphics documentation.

7

Manipulating Geospatial Data

For some purposes, geospatial data is fine to use *as is*. Sooner or later, though, you need to extract, combine, massage, and transform geodata. This chapter discusses some of the tools and techniques that the Mapping Toolbox provides for such purposes.

Units and Notation

Geospatial data always expresses or implies units and types of distance, and in many instances involves time. This section helps you understand the different types and notations used for time, location, and distance, and how to convert data between them easily.

"Converting Latitude and Longitude Notations" on page 7-2	Notations for spherical coordinates and conversion between them
"Converting Distance Units" on page 7-5	Angular distance and conversion to linear distance
"Converting Time Notations" on page 7-8	Expressing time and conversions between time notations

For related documentation on calculating distances, positions, ranges, and angles, see "Planetary Almanac Data" on page 3-24.

Converting Latitude and Longitude Notations

Spherical coordinates such as latitude and longitude are angular measures, and cannot be represented as plane coordinates without projection. Angles can be represented as variables in the Mapping Toolbox in three ways:

- Degrees plus fractions (default; also called *decimal degrees*)
- Radians
- Degrees-minutes-seconds

The toolbox provides functions for converting among these formats.

Regardless of the units used for angles, a pair of them is needed to fix the horizontal location of a point. To manipulate geospatial data given in spherical coordinates, it is necessary to know whether a coordinate tuple represents (latitude, longitude) or (longitude, latitude). This might not always be obvious from inspecting the data.

Degrees-Minutes-Seconds

Degrees-minutes-seconds, or *dms*, notation, is common in atlases and geographic texts, and is sometimes used in digital data sets. Angles in dms are

normally notated as *ddd° mm' ss"*. For example, 142°15'27" is 142 degrees, 15 minutes, and 27 seconds. There are 60 seconds in a minute and 60 minutes in a degree. The Mapping Toolbox internally represents dms angles by a single number, the format of which is *dddmm.ss*. For example, 142°15'27" is 14215.27. Such numbers can be either positive or negative. A special case of the dms format is the *dm* format, in which seconds are not included.

The real value of this notation is in entering data that arrives in this format. The toolbox includes the `mat2dms` function for easily entering dms data.

If you have a three-column matrix in which the columns are degrees, minutes, and seconds, respectively, `mat2dms` converts it to dms format:

```
format long g
dmsmatrix = [45 13 46; 156 45 01; -7 34 12.1]
dmsmatrix =

    45    13    46
   156    45     1
    -7    34    12.1

dmsformat = mat2dms(dmsmatrix)
dmsformat =

    4513.46
   15645.01
   -734.121
```

Note Take care when working with the *dms* format; for example, two angles in this format cannot be added. You should convert dms data to decimal degrees before working with it.

Converting Among Angle Unit Formats

The toolbox includes a variety of angle unit conversion functions. For example, to convert the dms format values to degrees or to radians, you can use `dms2deg` and `dms2rad`, respectively:

```
degformat = dms2deg(dmsformat)
degformat =
 45.2294
```

```
156.7503
 -7.5700
radformat = dms2rad(dmsformat)
radformat =
 0.7894
 2.7358
-0.1321
```

Similar functions include `deg2rad`, `rad2deg`, and `deg2dms`. Another, more general function, `angledim`, converts from one format to another. For example, how many degrees are in one quarter radian?

```
degs = angledim(1/4*pi,'radians','degrees')
degs =
   45
```

Converting Formatted Angle Strings to Numbers

Many sources of geographic data consist of text with the angles in degrees-minutes-seconds format such as *ddd° mm' ss"*. These formatted strings can include the characters for degrees, minutes, and seconds, as well as letters for north, south, east, and west or other special characters. These kinds of angle strings cannot be converted to numbers by using the MATLAB `num2str` function. However, you can convert many of these string formats to numeric decimal degrees using the `str2angle` function. The `str2angle` function accepts string matrices or cell arrays of strings containing values formatted in a number of commonly used angle formats:

```
strs = {'123°30''00"S','123-30-00S','123d30m00sS','1233000S'};
str2angle(strs)
ans =
       -123.5
       -123.5
       -123.5
       -123.5
```

Angular Unit Conversion

Longitudes always increase going eastward and decrease going westward. For longitudes of any magnitude, the function `npi2pi` wraps data to the range (-180 180):

```
longitudes = [-560 125 190];
```

```
newlongitudes = npi2pi(longitudes)
newlongitudes =
   160.0000  125.0000 -170.0000
```

Sometimes it is more natural to consider longitude as strictly positive, proceeding from the prime meridian (0°) eastward around and back to the prime meridian (360°). Any longitude data can be converted to this domain using the `zero22pi` function:

```
positivelongs = zero22pi(newlongitudes)
positivelongs =
   160.0000  125.0000  190.0000
```

If you need this data in radians, you can use an angle conversion function:

```
radianlongs = deg2rad(positivelongs)
radianlongs =
   2.7925    2.1817    3.3161
```

Several angle conversion functions are available in this toolbox, supporting degrees, radians, and degrees-minutes-seconds notation. Some useful utility functions are also included, such as `antipode`. For example, what is the antipodal point (on the opposite side of the Earth) of Natick, Massachusetts (about 42.3°N, 71.35°W)?

```
[antilat,antilong] = antipode(42.3,-71.35)
antilat =
   -42.3000

antilong =
   108.6500
```

The result (42.3°S,108.65°E) lies in the Indian Ocean southwest of Australia.

Converting Distance Units

In spherical coordinates distances are expressed as angles, not lengths. Since there is an infinity of arcs that can connect two points on a sphere or spheroid, by convention the shortest one (the *great circle* distance) is used to measure how far apart points are. To transform an angular distance into linear distance along a great circle, you must specify which ellipsoid vector should be used.

The Mapping Toolbox can express distances in a number of different units. It provides functions to convert between nautical miles (nm), statute miles (sm), feet (ft), kilometers (km), meters (m), degrees of arc length (deg), and radians of arc length (rad). The names of these functions are of the form `sm2km`, `km2rad`, etc. A general distance conversion function, `distdim`, is available as well.

There is no single default unit of distance measurement in the toolbox. Navigation functions use nautical miles as a default, almanac functions use kilometers, and the `distance` function use degrees of arc length. It is essential that you understand the default units of any function you use.

Note When distances are given in terms of angular units (degrees or radians), be careful to remember that these are specified in terms of arc length. While a degree of latitude always subtends one degree of arc length, this is only true for degrees of longitude along the Equator. If this were generally true, the Earth would be cylindrical.

On the Earth, a degree of arc length at the equator is about 60 nautical miles:

```
nauticalmiles = deg2nm(1)
nauticalmiles =
   60.0405
```

The Earth is the default assumption for these conversion functions. You can use other radii, however:

```
nauticalmiles = deg2nm(1,almanac('moon','radius'))
nauticalmiles =
   30.3338
```

The function `deg2sm` returns distances in statute, rather than nautical, miles:

```
deg2sm(1)
ans =
   69.0952
```

The unitsratio Distance Conversion Function

The `unitsratio` function lets you convert plane distances and angular distances from one measurement unit to another. It supports a wide range of linear distance units, from microns to miles. The syntax for `unitsratio` is

```
factor = unitsratio(to-unit, from-unit) * distance
```

By omitting the distance, you can obtain the raw conversion factor.

1 For example, to compute the number of centimeters in an inch, type

```
cm2in = unitsratio('cm','inch')
cm2in =
    2.5400
```

2 To convert this number of centimeters back to inches, type

```
in = unitsratio('in','centimeter') * cm2in
in =
     1
```

Note that `unitsratio` supports various abbreviations for units of length.

1 As another example, first use `almanac` to obtain the grs80 ellipsoid:

```
almanac('earth','grs80', 'km')
ans =
  1.0e+003 *
    6.3781    0.0001
```

2 Compute the difference between the semi-major and semi-minor axis:

```
dkm = ans(1) * ans(2)
dkm =
  521.8540
```

3 Use `unitsratio` to convert this distance from kilometers to meters:

```
dm = unitsratio('m','km')*dkm
dm =
  5.2185e+005
```

4 Now convert from meters to international feet:

```
dft = unitsratio('ft','m')*dm
dft =
  1.7121e+006
```

5 Finally, see how much is this in statute miles:

```
dsmi = unitsratio('statute mile','foot')*dft
dsmi =
  324.2644
```

The `unitsratio` function also converts angles between degrees and radians.

Converting Time Notations

Times can be represented as variables in the Mapping Toolbox in three ways: hours, seconds, and hours-minutes-seconds. The toolbox provides functions for converting among these formats.

Hours

This is the default time unit notation for the toolbox.

Hour notation is simply decimal notation in terms of hours. Two hours and fifteen minutes would be 2.25.

Seconds

Seconds notation is simply decimal notation in terms of seconds. One hour would be 3600.

Hours-Minutes-Seconds

Hours-minutes-seconds, or *hms* notation, is analogous to dms notation for angles. In text, an hms time would be *hh*:*mm*:*ss*. For example, 12:36:15 is 12 hours, 36 minutes, and 15 seconds. In the Mapping Toolbox, when hms times are represented by a single number, the format is *hhmm.ss*. For example, 12:36:15 is 1236.15.

This notation is most useful for entering data provided in this format. The toolbox includes the `mat2hms` function for entering *hms* data, which is similar to the `mat2dms` function described earlier.

Note Exercise care when you use with the hms format; for example, two times in this format cannot simply be added. Always convert data to decimal hours before working with it numerically.

Converting Between Time Unit Formats

Time units can be converted using functions similar to those described for angle unit conversions. These include `hr2sec` and `hms2hr`, as well as a general conversion function, `timedim`, that works like `angledim`.

Manipulating Vector Data

The Mapping Toolbox enables you to manipulate, combine, and separate vector geodata in a variety of ways. This section describes some useful functions for conditioning, selecting, and transforming vector geodata.

"Repackaging Vector Objects" on page 7-11	Separating and combining `NaN`-delimited vectors
"Matching Line Segments" on page 7-12	Forming closed loops that can be represented as patches
"Geographic Interpolation" on page 7-13	Linear point interpolation on the sphere and spheroid; adding detail to lines manually
"Vector Intersections" on page 7-19	Computing where small circles, rhumb lines, and circles intersect
"Polygon Area" on page 7-21	Computing the areas of polygons on the sphere and spheroid
"Overlaying Polygons with Boolean Logic" on page 7-22	Performing geometric intersections of polygons and obtaining logical answers
"Cutting Polygons at the Date Line" on page 7-26	Working around the discontinuity in longitude that happens at 180 degrees east/west
"Building Buffer Zones" on page 7-28	Constructing distance contours around map features for analysis and display
"Trimming Vector Data to a Rectangular Region" on page 7-30	Clipping away line and polygon coordinates that lie outside a region of interest
"Trimming Vector Data to an Arbitrary Region" on page 7-32	Using a data grid to define regions of interest and clip vector data to them
"Simplifying Vector Coordinate Data" on page 7-33	Eliminating visually redundant coordinates to remove unnecessary detail, and to speed and stylize map displays

Repackaging Vector Objects

It can be difficult to identify line or patch segments once they have been combined into large NaN-clipped vectors. You can separate these polygon or line vectors into their component segments using `polysplit`, which take column vectors as inputs:

Extracting and Joining Polygons or Line Segments

1 Enter two `NaN`-delimited arrays in the form of column vectors:

```
lat = [45.6 -23.47 78 NaN 43.9 -67.14 90 -89]';
long = [13 -97.45 165 NaN 0 -114.2 -18 0]';
```

2 Use `polysplit` to create two cell arrays, `latc` and `lonc`:

```
[latc,lonc] = polysplit(lat,long)
latc =
    [3x1 double]    [4x1 double]
lonc =
    [3x1 double]    [4x1 double]
```

3 Inspect the contents of the cell arrays:

```
[latc{1} lonc{1}]
ans =
                      45.6                      13
                    -23.47                  -97.45
                        78                     165
[latc{2} lonc{2}]
ans =
                      43.9                       0
                    -67.14                  -114.2
                        90                     -18
                       -89                       0
```

Note that each cell array element contains a segment of the original line.

4 To reverse the process, use `polyjoin`:

```
[lat2,lon2] = polyjoin(latc,lonc);
```

5 The joined segments are identical with the initial `lat` and `lon` arrays:

```
[lat long] == [lat2 lon2]
ans =
     1     1
     1     1
     1     1
     0     0
     1     1
     1     1
     1     1
     1     1
```

The logical comparison is false for the `NaN` delimiters by definition.

6 You can test for global equality, including `NaN`s, as follows:

```
isequalwithequalnans(lat,lat2) & isequalwithequalnans(long,lon2)
ans =
     1
```

See the reference pages for `polysplit` and `polyjoin` for further information.

Matching Line Segments

A common operation on sets of line segments is the concatenation of segments that have matching endpoints. The `polymerge` command compares endpoints of segments within latitude and longitude vectors to identify endpoints that match exactly or lie within a specified distance. The matching segments are then concatenated, and the process continues until no more coincidental endpoints can be found. The two required arguments are a latitude (or x) vector and a longitude (or y) vector. The following exercise shows this process at work:

Linking Line Segments into Polygons

1 Construct column vectors representing coordinate values:

```
lat = [3 2 NaN 1 2 NaN 5 6 NaN 3 4]';
lon = [13 12 NaN 11 12 NaN 15 16 NaN 13 14]';
```

2 Concatenate the segments that match exactly:

```
[latm,lonm] = polymerge(lat,lon)
ans =
```

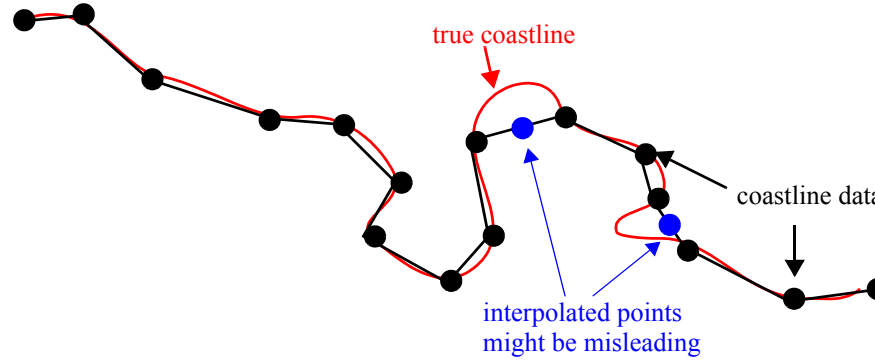
```
     5     15
     6     16
   NaN    NaN
     1     11
     2     12
     2     12
     3     13
     3     13
     4     14
```

The original four segments are merged into two segments.

The `polymerge` function takes an optional third argument, a (circular) distance tolerance that permits inexact matching. A fourth argument enables you to specify whether the function outputs vectors or cell arrays. See the reference page for `polymerge` for further information.

Geographic Interpolation

When using vector data, you must be careful when you make assumptions concerning geographic reality between data points. For instance, when plotting vector data, you might connect each point with a straight line segment. This does not usually indicate any true knowledge about the region between known points. Data consisting of points along a coastline might be sparse; in the absence of other knowledge, filling in data can be misleading.



Interpreting Sparse Vector Data

Despite the dangers of misinterpretation, many circumstances exist in which geographic data interpolation is useful or necessary. Sparser data can be linearly filled in with the `interpm` function.

Consider a set of latitude and longitude points that you want to be separated by no more than one degree in either direction:

```
     lats = [1 2 4 5]; longs = [1 3 4 5]; maxdiff = 1;

[newlats,newlongs] = interpm(lats,longs,maxdiff)
newlats =
   1.0000
   1.5000
   2.0000
   3.0000
   4.0000
   5.0000
newlongs =
   1.0000
   2.0000
   3.0000
   3.5000
   4.0000
   5.0000
```

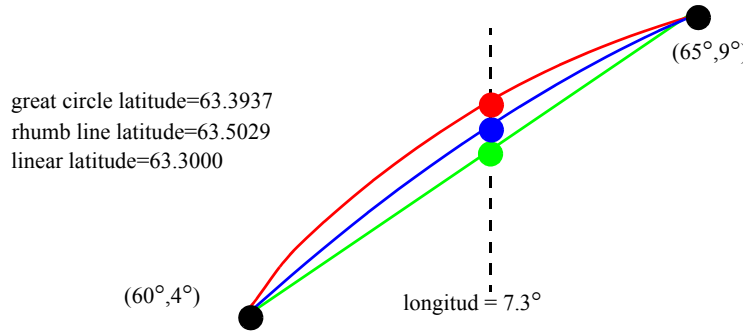
In the original `lats`, there is a gap of 2 degrees between the 2 and the 4. A linearly interpolated point, (3,3.5) was therefore inserted in `newlats` and `newlongs`. Similarly, in the original `longs`, there is a gap of 2 degrees between the 1 and the 3. The point (1.5,2) was therefore interpolated and placed into `newlats` and `newlongs`. Now, no adjacent points in either `newlats` or `newlongs` are greater than `maxdiff` apart.

The `interpm` function returns the original data with new linearly interpolated points inserted. Sometimes, however, only the interpolated values are desired. The commands `intrplat` and `intrplon` provide a capability similar to the MATLAB `interp1` command, allowing for different methods of interpolation.

Use `intrplat` to interpolate a latitude for a given longitude. Given a monotonic set of longitudes and their matching latitude points, you can interpolate a new latitude for a given longitude in a linear, spline, cubic, rhumb line, or great circle sense.

Find the latitude corresponding to a longitude of 7.3° in the following data in a linear, great circle, and rhumb line sense:

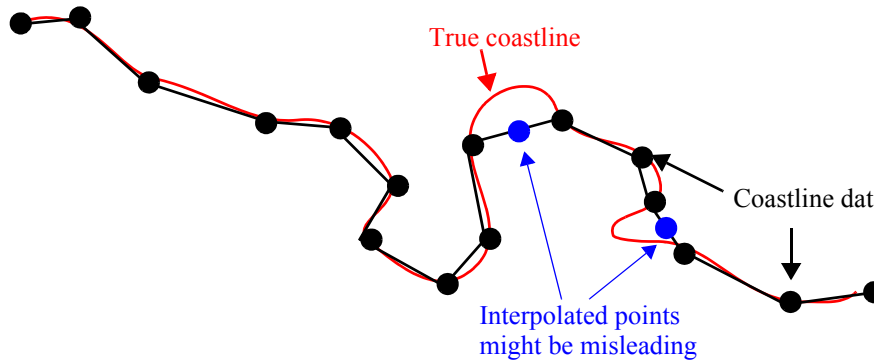
```
longs = [1 3 4 9 13]; lats = [57 68 60 65 56]; newlong = 7.3;

newlat = intrplat(longs,lats,newlong,'linear')
newlat =
   63.3000

newlat = intrplat(longs,lats,newlong,'gc')
newlat =
   63.5029

newlat = intrplat(longs,lats,newlong,'rh')
newlat =
   63.3937
```



The `intrplon` function provides the same capability for interpolating new longitudes for given latitudes.

Interpolating Points Along Line Vectors

When using vector data, remember that like raster data, coordinates are sampled measurements. This involves unavoidable assumptions concerning what the geographic reality is between specified data points. The normal assumption when plotting vector data requires that points be connected with straight line segments, which essentially indicates a lack of knowledge about conditions between the measured points. For lines that are by nature continuous, such as most rivers and coastlines, such piecewise linear interpolation can be false and misleading.



Interpolating Sparse Vector Data

Despite the possibility of misinterpretation, circumstances do exist in which geographic data interpolation is useful or even necessary. The Mapping Toolbox provides the `interpm` function to interpolate between known data points. One value of linearly interpolating points is to fill in lines of constant latitude or longitude (e.g., administrative boundaries) that can curve when projected.

Interpolating Vectors to Achieve a Minimum Point Density

This example interpolates values in a set of latitude and longitude points to have no more than one degree of separation in either direction.

1 Define two fictitious latitude and longitude data vectors:

```
lats = [1 2 4 5]; longs = [1 3 4 5];
```

2 Specify a densification parameter of 1 (the default unit is degrees):

```
maxdiff = 1;
```

3 Call `interpm` to fill in any gaps greater than 1° in either direction:

```
[newlats,newlongs] = interpm(lats,longs,maxdiff)
newlats =
    1.0000
    1.5000
```

```
    2.0000
    3.0000
    4.0000
    5.0000
newlongs =
    1.0000
    2.0000
    3.0000
    3.5000
    4.0000
    5.0000
```

In `lats`, a gap of 2° exists between the values 2 and the 4. A linearly interpolated point, (3,3.5) was therefore inserted in `newlats` and `newlongs`. Similarly, in `longs`, a gap of 2° exists between the 1 and the 3. The point (`1.5, 2`) was therefore interpolated and placed into `newlats` and `newlongs`. Now, the separation of adjacent points is no greater than `maxdiff` in either `newlats` or `newlongs`.

See the reference page for `interpm` for further information.

Interpolating Coordinates at Specific Locations

Both the original data and new linearly interpolated points are returned by `interpm`. Sometimes, however, you might want only the interpolated values. The functions `intrplat` and `intrplon` provide a capability similar to the MATLAB `interp1` function, and give you control over the method used for interpolation. Note that they only interpolate and return one value at a time.

Use `intrplat` to interpolate a latitude for a given longitude. Given a monotonic set of longitudes and their matching latitude points, you can interpolate a new latitude for a longitude you specify, interpolating along linear, spline, cubic, rhumb line, or great circle paths. The longitudes must increase or decrease monotonically. If this is not the case, you might be able to use the companion function `intrplon` if the latitude values are monotonic.

Interpolate a latitude corresponding to a longitude of 7.3° in the following data in a linear, great circle, and rhumb line sense:

1 First define some fictitious latitudes and longitudes:

```
longs = [1 3 4 9 13]; lats = [57 68 60 65 56];
```

2 Specify the longitude for which to compute a latitude:

```
newlong = 7.3;
```

3 Generate a new latitude with linear interpolation:

```
newlat = intrplat(longs,lats,newlong,'linear')
newlat =
   63.3000
```

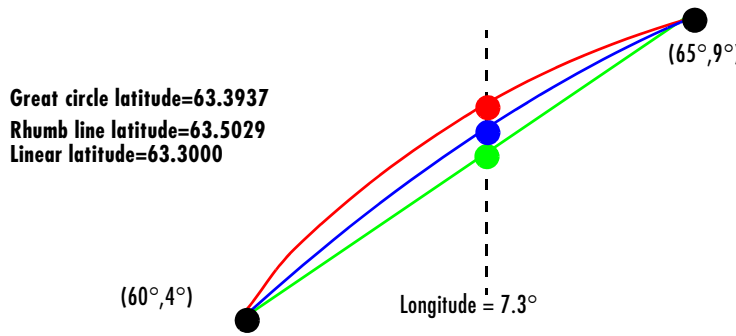
4 Now generate the latitude using great circle interpolation:

```
newlat = intrplat(longs,lats,newlong,'gc')
newlat =
   63.5029
```

5 Generate it again, specifying interpolation along a rhumb line:

```
newlat = intrplat(longs,lats,newlong,'rh')
newlat =
   63.3937
```

The following diagram illustrates these three types of interpolation. The `intrplat` function also can perform spline and cubic spline interpolations.



As mentioned above, the `intrplon` function provides the capability to interpolate new longitudes from given set of longitudes and monotonic latitudes.

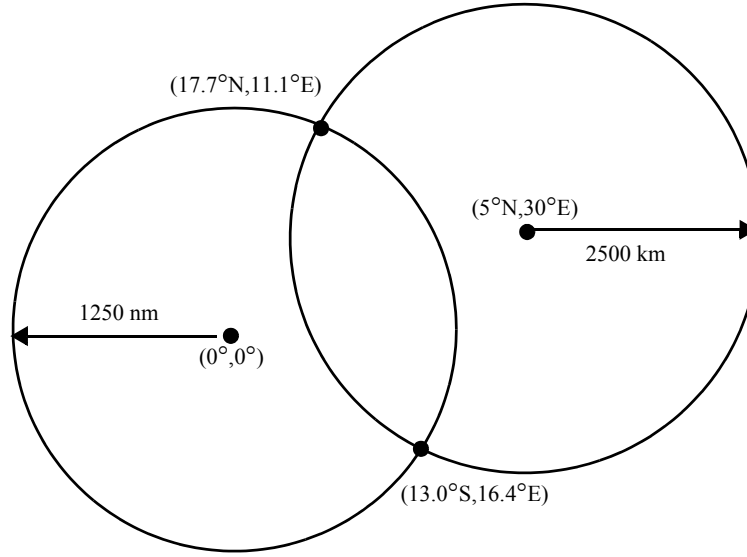
See the reference pages for `intrplat` and `intrplon` for further information.

Vector Intersections

The Mapping Toolbox provides a set of functions to perform intersection calculations on vector data computed by the toolbox, which include great and small circles as well as rhumb line tracks. The functions also determine intersections of arbitrary vector data.

Compute the intersection of a small circle centered at (0°,0°) with a radius of 1250 nautical miles and a small circle centered at (5°N,30°E) with a radius of 2500 kilometers:

```
[lat,long] = scxsc(0,0,nm2deg(1250),5,30,km2deg(2500))
lat =
   17.7487 -12.9839
long =
   11.0624 16.4170
```



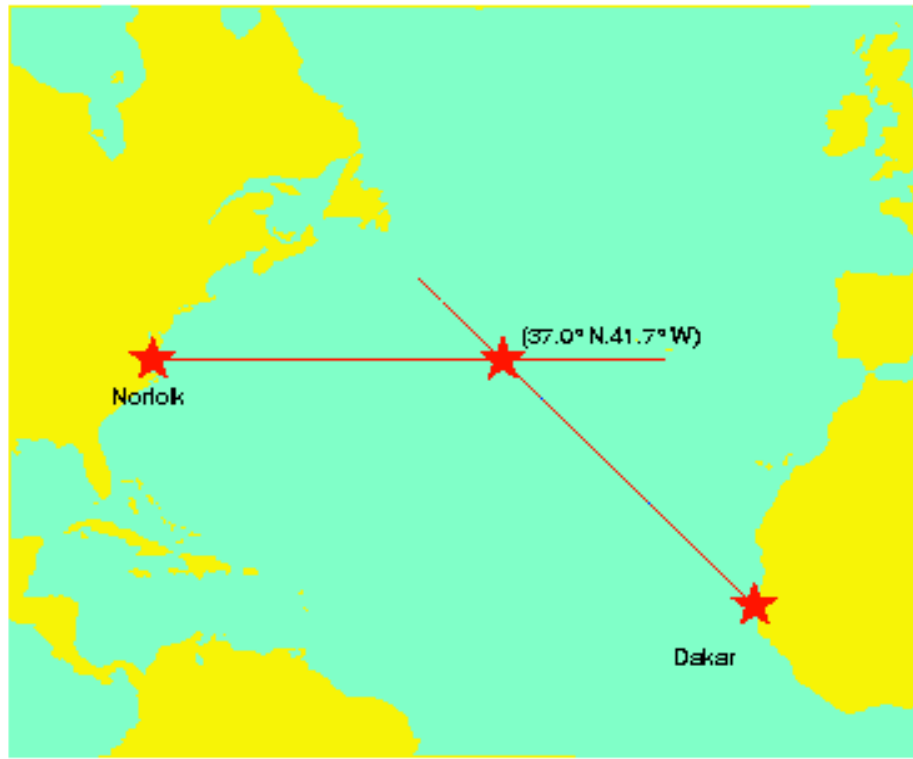
Notice that, in general, small circles intersect twice or never. For the case of exact tangency, `scxsc` returns two identical intersection points. Other similar commands include `rhxrh` for intersecting rhumb lines, `gcxgc` for intersecting great circles, and `gcxsc` for intersecting a great circle with a small circle.

Imagine a ship setting sail from Norfolk, Virginia (37°N,76°W), maintaining a steady due-east course (90°), and another ship setting sail from Dakar, Senegal

(15°N,17°W), with a steady northwest course (315°). Where would the tracks of the two vessels cross?

```
[lat,long] = rhxrh(37,-76,90,15,-17,315)
lat =
   37
long =
   -41.7028
```

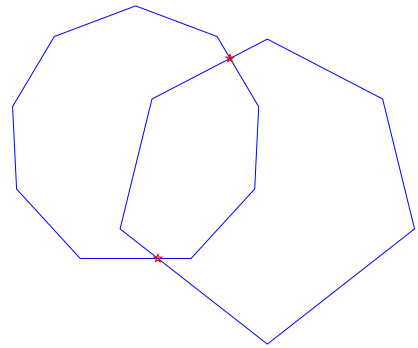
The intersection of the tracks is at (37°N,41.7°W), which is roughly 600 nautical miles west of the Azores in the Atlantic Ocean.



You can also compute the intersection points of arbitrary vectors of latitude and longitude. The `polyxpoly` command finds the segments that intersect and interpolates to find the intersection points. The interpolation is done linearly, as if the points were in a cartesian *x-y* coordinate system. The `polyxpoly`

command can also identify the line segment numbers associated with the intersections:

```
[xint,yint] = polyxpoly(x1,y1,x2,y2);
```



If the spacing between points is large, there can be some difference between the intersection points computed by `polyxpoly` and the intersections shown on a map display. This is a result of the difference between straight lines in the unprojected and projected coordinates. Similarly, there can be differences between the `polyxpoly` result and intersections assuming great circles or rhumb lines between points.

Polygon Area

You can use the function `areaint` to calculate geographic areas for vector data in polygon format. The function performs a numerical integration using Green's Theorem for the area on a surface enclosed by a polygon. Because this is a discrete integration on discrete data, the results are not exact. Nevertheless, the method provides the best means of calculating the areas of arbitrarily shaped regions. Better measures result from better data.

The Mapping Toolbox function `areaint` (for area by integration), like the other area functions, `areaquad` and `areamat`, returns areas as a fraction of the entire planet's surface, unless a radius is provided. Here you calculate the area of the continental United States using the `usalo` workspace. Three areas are returned, because the data contains three polygons: Long Island, Martha's Vineyard, and the rest of the continental U.S.:

```
load usalo
```

```
earthradius = almanac('earth','radius');
area = areaint(uslat,uslon,earthradius)
area =
  1.0e+06 *
        7.9256
        0.0035
        0.0004
```

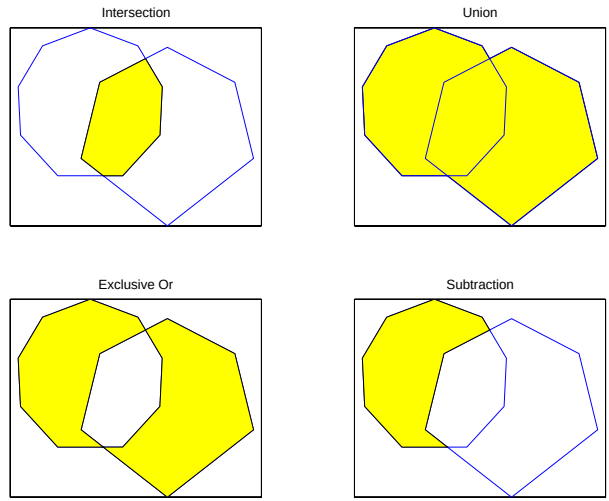
Because the default Earth radius is in kilometers, the area is in square kilometers. From the same workspace, the areas of the Great Lakes can be calculated, this time in square miles:

```
earthradius = almanac('earth','radius','miles');
area = areaint(gtlakelat,gtlakelon,earthradius)
area =
  1.0e+04 *
        8.0124
        1.0382
        0.7635
```

Again, three areas are returned, the largest for the polygon representing Superior, Michigan, and Huron together, the other two for Erie and Ontario.

Overlaying Polygons with Boolean Logic

Polygon Boolean operations are used to answer a variety of questions about logical relationships of vector data polygon objects. Standard Boolean operations include intersection, union, subtraction, and an exclusive OR operation. The `polybool` function performs these operations on two sets of vectors, which can represent *x-y* or latitude-longitude coordinate pairs. In computing points where boundaries intersect, interpolations are carried out on the coordinates as if they were planar. Here is an example that shows all the available operations.



The result is returned as `NaN`-clipped vectors by default. In cases where it is important to distinguish outer contours of polygons from interior holes, `polybool` can also accept inputs and return outputs as cell arrays. In the cell array format, a cell array entry starts with the list of points making up the outer contour. Subsequent `NaN`-clipped faces within the cell entry are interpreted as interior holes.

Intersecting Polygons with the polybool Function

The following exercise demonstrates how you can use `polybool`:

1 Construct a twelve-sided polygon:

```
theta = (0:pi/6:2*pi)';
lat1 = sin(theta);
lon1 = cos(theta);
```

2 Construct a triangle that overlaps it:

```
lat2 = [0 1 -1 0]';
lon2 = [0 2 2 0]';
```

3 Plot the two shapes together with blue and red lines:

```
axesm miller
plotm(lat1,lon1,'b')
plotm(lat2,lon2,'r')
```

4 Compute the intersection polygon and plot it as a green patch:

```
[lati,loni] = polybool('intersection',lat1,lon1,lat2,lon2);
[lati loni]
ans =
      0.44093      0.88185
  1.2246e-016            1
     -0.44093      0.88185
  1.2246e-016  6.1232e-017
      0.44093      0.88185

patchm(lati,loni,'g')
```

5 Compute the union polygon and plot it as a magenta patch:

```
[latu,lonu] = polybool('union',lat1,lon1,lat2,lon2);
[latu lonu]
ans =
      0.44093      0.88185
            1            2
           -1            2
     -0.44093      0.88185
         -0.5      0.86603
     -0.86603          0.5
           -1  6.1232e-017
     -0.86603         -0.5
         -0.5     -0.86603
  1.2246e-016           -1
          0.5     -0.86603
      0.86603         -0.5
            1  6.1232e-017
      0.86603          0.5
          0.5      0.86603
      0.44093      0.88185

patchm(latu,lonu,'m')
```

6 Compute the exclusive OR polygon and plot it as a yellow patch:

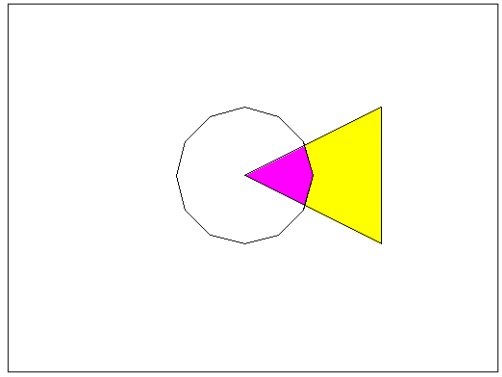
```
[latx,lonx] = polybool('xor',lat1,lon1,lat2,lon2);
[latx lonx]
ans =
     -0.44093      0.88185
  1.2246e-016            1
      0.44093      0.88185
            1            2
           -1            2
     -0.44093      0.88185
          NaN          NaN
      0.44093      0.88185
  1.2246e-016  6.1232e-017
     -0.44093      0.88185
         -0.5      0.86603
     -0.86603          0.5
           -1  6.1232e-017
     -0.86603         -0.5
         -0.5     -0.86603
  1.2246e-016           -1
          0.5     -0.86603
      0.86603         -0.5
            1  6.1232e-017
      0.86603          0.5
          0.5      0.86603
      0.44093      0.88185

patchm(latx,lonx,'y')
```

7 Lastly, subtract the triangle from the 12-sided polygon and plot the resulting concave polygon as a white patch:

```
[latm,lonm] = polybool('minus',lat1,lon1,lat2,lon2);
[latm lonm]
ans =
      0.44093      0.88185
  1.2246e-016  6.1232e-017
     -0.44093      0.88185
         -0.5      0.86603
     -0.86603          0.5
           -1  6.1232e-017
```

```
      -0.86603          -0.5
          -0.5      -0.86603
     1.2246e-016            -1
           0.5      -0.86603
       0.86603          -0.5
             1   6.1232e-017
       0.86603           0.5
           0.5       0.86603
       0.44093       0.88185

patchm(latm,lonm,'w')
```

The final set of colored shapes is shown below.



See the reference page for `polybool` for further information.

Cutting Polygons at the Date Line

Polygon Boolean operations treat input vectors as plane coordinates. The `polyxpoly` function can be confused by geographic data that has discontinuities in longitude coordinates at date line crossings. This can happen when points with longitudes near 180° connect to points with longitudes near -180°, as might be the case for eastern Siberia, Antarctica, and also for small circles and other patch objects generated by toolbox functions.

You can prepare such geographic data for use with `polybool` or for patch rendering by cutting the polygons at the date line with the `flatearthpoly`

function. The result of `flatearthpoly` is a polygon with points inserted to follow the date line up to the pole, traverse the longitudes at the pole, and return to the date line crossing along the other edge of the datel ine.

Removing Discontinuities from a Small Circle

1. Create an Orthographic view of the Earth and plot `coast` on it:

   ```
   close all; clear all;
   axesm ortho
   setm(gca,'Origin', [60 170]); framem on; gridm on
   load coast
   plotm(lat, long)
   ```

2. Generate a small circle that encompasses the North Pole and color it yellow:

   ```
   [latc,lonc] = scircle1(75,45,30);
   patchm(latc,lonc,'y')
   ```

3. Now “latten the small circle with `flatearthpoly`:

   ```
   [latf,lonf] = flatearthpoly(latc,lonc);
   ```

4. Plot the cut circle that you just generated as a magenta line:

   ```
   plotm(latf,lonf,'m')
   ```

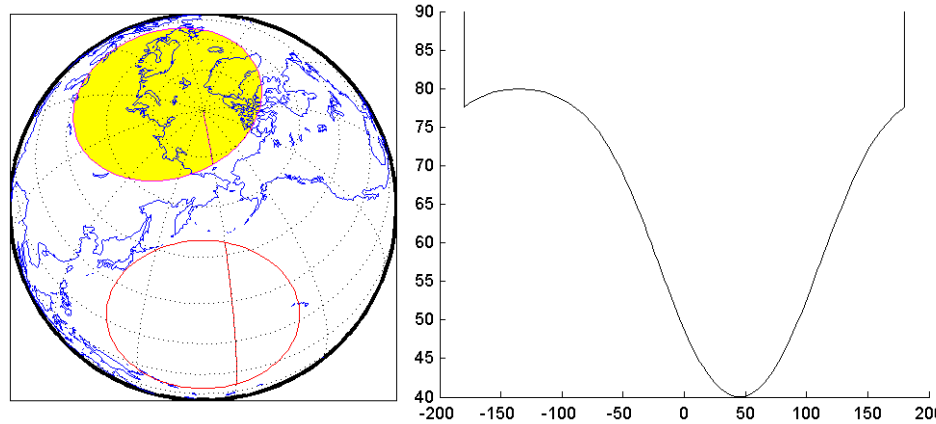
5. Generate a second small circle that does not include a pole:

   ```
   [latc1 lonc1] = scircle1(20, 170, 30)
   ```

6. Flatten it and plot it as a red line:

   ```
   [latf1,lonf1] = flatearthpoly(latc1,lonc1);
   plotm(latf1,lonf1,'r')
   ```

   The following figure shows the result of these operations. Note that the second small circle, which does not cover a pole, has been clipped into two pieces along the date line. On the right, 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` will not cover more than a hemisphere.

Note As the above figure illustrates, you do not need to use `flatearthpoly` to prepare data for a map display. The Mapping Toolbox display functions automatically cut and trim geographic data if required by the map projection. Use this function only when conducting Boolean operations on polygons.

See the reference page for `flatearthpoly` for further information.

Building Buffer Zones

A *buffer zone* is the area within a specified distance of a map feature. For vector geodata, buffer zones are constructed as polygons. For raster geodata, buffer zones are collections of contiguous, identically-coded grid cells. When the feature is a polygon, a buffer zone can be defined as the locus of points within a certain distance of its boundary, either inside or outside the polygon. A buffer zone for a linear object is the locus of points a certain distance away from it. Buffer zones form equidistant contour lines around objects.

The `bufferm` function computes and returns vectors that represent a set of points that define a buffer zone. It forms the buffer by placing small circles at

the vertices of the polygon and rectangles along each of its line segments, and applying the `union` Boolean operation to these objects.

Generating a Buffer Around a Compound Polygon

Demonstrate `bufferm` using a compound polygon representing Italy that you extract from the `worldlo` data set. The boundaries of Italy are provided to `bufferm` as `NaN`-clipped latitude and longitude vectors. Using it, compute a buffer zone at a distance of 1.5 degrees out from the boundaries of Italy:

1 Create a base map of the area surrounding Italy, and hide the border:

```
close all; clear all;
worldmap('lo', [35,50], [3,23], 'lineonly')
hidem(gca)
```

2 Read the `worldlo` data set and extract the polygons representing Italy:

```
load worldlo
[ilat,ilon] = extractm(POpatch,'italy');
```

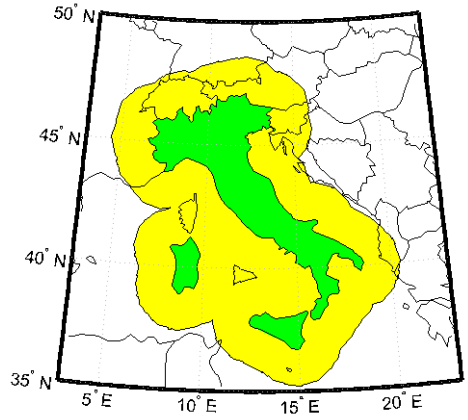
3 User `bufferm` to process the Italy polygons and output a buffer zone 1.5 degrees away from the boundary:

```
[latb,lonb] = bufferm(ilat,ilon,1.5,'out');
```

This can take several minutes, because of the great number of geometric computations that `bufferm` is performing.

4 Draw the buffer zone in yellow, and then draw Italy in green:

```
patchesm(latb, lonb, 'y')
patchesm(ilat, ilon, 'g')
```



Trimming Vector Data to a Rectangular Region

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 (such as the `coast` data set), 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.

Line data and patch data need to be trimmed differently. 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. Patch data requires a more complicated method to ensure that the patch objects are correctly formed.

For the vector data, two functions are available to achieve this. If the vectors are to be handled as line data, the `maptriml` function returns variables containing only those points that lie within the defined region. If, instead, you want to maintain polygon format, use the `maptrimp` function. Be aware, however, that patch-trimmed data is usually larger and more expensive to compute.

Note When drawing maps, the Mapping Toolbox automatically trims vector geodata to the region specified by the frame limits (`FLatLimit` and `FLonLimit` map axes properties) for azimuthal projections, or to frame or map limits (`MapLatLimit` and `MapLonLimit` map axes properties) for nonazimuthal projections. The trimming is done internally in the display routine, keeping the original data intact. For further information on trimming vector geodata, see "Axes for Drawing Maps" on page 4-5, along with the reference pages for the trimming functions.

Trimming Vectors to Form Lines and Polygons

1 Load the `coast` MAT-file for the entire world:

```
close all; clear all;
load coast
```

2 Define a region of interest centered on Australia:

```
latlim = [-50 0]; longlim = [105 160];
```

3 Use `maptriml` to delete all data outside these limits, producing line vectors:

```
[linelat,linelong] = maptriml(lat,long,latlim,longlim);
```

4 Do this again, but use `maptrimp` to produce polygon vectors:

```
[polylat,polylong] = maptrimp(lat,long,latlim,longlim);
```

5 See how much data has been reduced:

```
whos
  Name           Size                    Bytes  Class

  lat         9589x1                     76712  double array
  latlim         1x2                        16  double array
  linelat      870x1                      6960  double array
  linelong     870x1                      6960  double array
  long        9589x1                     76712  double array
  longlim        1x2                        16  double array
  polylat     1020x1                      8160  double array
  polylong    1020x1                      8160  double array
```

```
Grand total is 22962 elements using 183696 bytes
```

Note that the clipped data is only 10% as large as the original data set.

6 Plot the trimmed patch vectors on a Miller projection:

```
axesm('MapProjection', 'miller', 'Frame', 'on',...
'FlatLimit', latlim, 'FlonLimit', longlim)
patchesm(polylat, polylong, 'c')
```

7 Lastly, plot the trimmed line vectors to see that they conform to the patches:

```
plotm(linelat, linelong, 'm')
```

See the reference pages for `maptriml` and `maptrimp` for further information.

Trimming Vector Data to an Arbitrary Region

Often a set of data contains unwanted data mixed in with the desired values. For example, your data might include vectors covering the entire United States, but you only want to work with those falling in Alabama. Sometimes a data set contains noise — perhaps three or four points out of several thousand are obvious errors (for example, one of your city points is in the middle of the ocean). In such cases, locating outliers and errors in the data arrays can be quite tedious.

The `filterm` command uses a data grid to filter a vector data set. Its calling sequence is as follows:

```
[flats,flons] = filterm(lats,lons,grid,refvector,allowed)
```

Each location defined by `lats` and `lons` is mapped to a cell in `grid`, and the value of that grid cell is obtained. If that value is found in `allowed`, that point is output to `flats` and `flons`. Otherwise, the point is filtered out.

The grid might encode political units, and the allowed values might be the code or codes indexing certain states or countries (e.g., Alabama). The grid might also be real-valued (e.g., terrain elevations), although it could be awkward to specify all the values allowed. More often, logical or relational operators will give better results for such grids, enabling the allowed value to be `1` (for `true`). For example, you could use this transformation of the `topo` grid:

```
[flats,flons] = filterm(lats,lons,double(topo>0),topolegend,1)
```

The output would be those points in `lats` and `lons` that occupy dry land (mostly, because some water bodies are above sea level).

For further information, see the reference page for `filterm`. Also see “Data Grids as Logical Variables” on page 7-42.

Simplifying Vector Coordinate Data

Avoiding visual clutter in composing maps is an essential part of cartographic presentation. In cartography, this is described as map generalization, which involves coordinating many techniques, both manual and automated. Limiting the number of points in vector geodata is an important part of generalizing maps, and is especially useful for conditioning cartographic data, plotting maps at small scales, and creating versions of geodata for use at small scales.

An easy, but naive, approach to point reduction would be to discard every *n*th element in each coordinate vector. However, this can result in very poor representations of the original shapes. The Mapping Toolbox provides a function to eliminate insignificant geometric detail in linear and polygonal objects while still maintaining accurate representations of their shapes. The `reducem` function implements a powerful line simplification algorithm (known as Douglas-Peucker) that intelligently selects and deletes visually redundant points.

The `reducem` function takes latitude and longitude vectors plus an optional linear tolerance parameter as arguments, and outputs reduced (simplified) versions of the vectors, in which deviations perpendicular to local “trend lines” in the vectors are all greater than the tolerance criterion. Endpoints of vectors

are preserved. Optional outputs are an error measure and the tolerance value used (which is computed when you do not supply a value).

Note Simplified line data might not always be appropriate for display. If all or most intermediate points in a feature are deleted, then lines that appear straight in one projection can be incorrectly displayed as straight lines in others, and separate lines can be caused to intersect. In addition, when you are reducing data over large world regions, the effective degree of reduction near the poles will be less than that achieved near the equator, due to the fact that the algorithm treats geographic coordinates as if they were planar.

Using reducem to Simplify Lines

The `reducem` function works on both patch and line data. Getting results that look right at an intended scale might require some experimentation with the tolerance parameter. The best way to proceed may be to allow the tolerance to default, and have `reducem` return the tolerance it computed as the fourth value. If the output still has too much detail, then double the tolerance and try again. Similarly, if the output lines do not have enough detail, halve the tolerance and try again. You can also use the third return value, which indicates the percentage of line length that was eliminated by reduction, as a guide to achieve consistent simplification results, although this parameter is sensitive to line geometry and thus can vary by feature type.

To demonstrate the use of `reducem`, this example extracts the outline of the state of Massachusetts from the `usahi` high-resolution data set:

1. Load the U.S. data, extract Massachusetts, and remove the rest of the data:

   ```
   clear all; close all; load usahi
   ```

   The `stateline` structure contains vector data for all 50 U.S. states and the District of Columbia. Retrieve the boundary of Massachusetts, the 19th entry in the structure:

   ```
   malat = stateline(19).lat;
   malon = stateline(19).long;
   ```

   Had you not known the index for Massachusetts, you could have used `extractm`, asking for the state by name:

```
[malat, malon] = extractm(stateline,'massachusetts');
```

2 Note that the Massachusetts state outline consists of 953 points:

```
numel(malat)
ans =
   958
```

3 As you no longer need it, delete the rest of the data from the workspace:

```
clear state*
```

4 Now use `reducem` to simplify the boundary vectors, and inspect the results:

```
[malat1,malon1,cerr,tol] = reducem(malat,malon);
numel(malat1)
ans =
   253
```

5 The number of points used to represent the boundary has dropped from 958 to 253. Compute the degree of reduction:

```
numel(malat1)/numel(malat)
ans =
      0.26409
```

The vectors have been reduced to about a quarter of their original size using the default tolerance.

6 Now examine the error and tolerance values returned by `reducem`:

```
[cerr tol]
ans =
     0.033116    0.0059728
```

The `cerr` value says that only 3.3% of total boundary length was eliminated (despite removing 74% of the points). The tolerance that achieved this was computed by `reducem` as 0.006 decimal degrees, or about 0.66 km.

7 To see the geometric result, plot the reduced outline in red over the original outline in blue:

```
axesm('MapProjection','eqdcyl', 'FlatLim',[41.1 43.0],...
'FlonLim',[-69.8, -73.6], 'Frame','on');
```

```
plotm(malat, malon, 'b')
plotm(malat1, malon1, 'r')
```

You need to zoom in two or three times to see the differences in detail.

8. Double the tolerance, and reduce the original boundary into new variables:

```
[malat2,malon2,cerr2,tol2] = reducem(malat,malon, 0.012);
```

9. Repeat step 3 above with the new data and plot it in dark green:

```
numel(malat2)/numel(malat)
ans =
      0.16493
[cerr2 tol2]
ans =
     0.051734        0.012
plotm(malat2, malon2, 'Color',[0 .6 0])
```

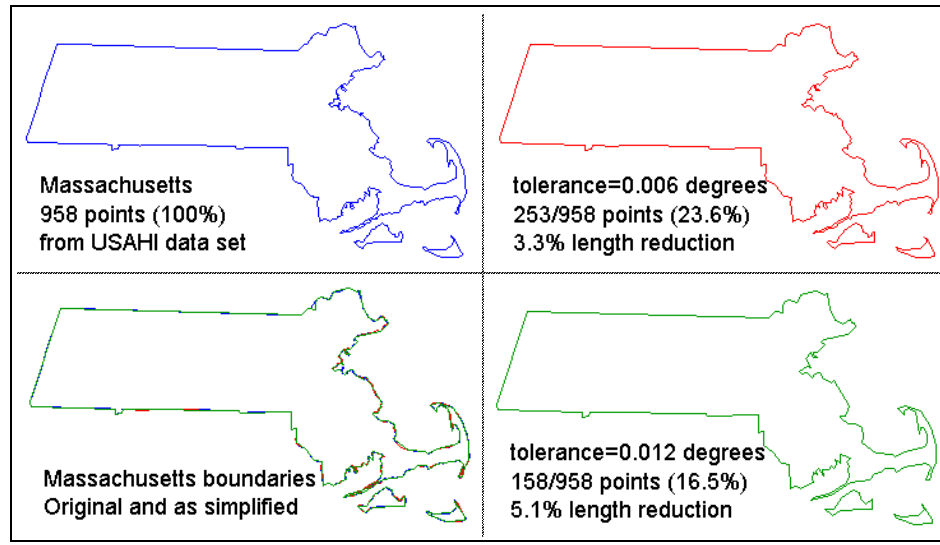
Now you have removed 83% of the points, and 5.2% of total length.

10. Repeat one more time, raising the tolerance to 0.1, and plot the result in black:

```
[malat3,malon3,cerr3,tol3] = reducem(malat,malon, 0.1);
plotm(malat3, malon3, 'Color',[0 0 0])
```

As the composite map shows, the visual effects of point reduction are subtle, up to a point. The choice of a tolerance when reducing line detail depends strongly on the purpose of the map and the scale at which it is to be displayed.

Note This exercise generalized a set of disconnected patches. When patches are contiguous (such as the U.S. state outlines), using `reducem` can result in inconsistencies in boundary representation and gaps at points where states meet. For best results, `reducem` should be applied to line data.



See the reference page for `reducem` for further information.

Manipulating Raster Data

There are some operations on geodata for which raster data is appropriate and in fact makes easier. Among them are logical operations on attributes, extracting attributes along tracks, and computing surface characteristics.

Vector-to-Raster Data Conversion

You can convert latitude-longitude vector data to a grid at any resolution you choose to make a raster base map or grid layer. The Mapping Toolbox provides GUI tools to help you do this, but you can also perform vector-to-raster conversions from the command line. The principal function for gridding vector data is `vec2mtx`, which allocates lines to a grid of any size you indicate, marking the lines with 1's and the unoccupied grid cells with 0's. The grid contains doubles, but if you desire a logical grid (see "Data Grids as Logical Variables" on page 7-42, below) you can cast the result to be a logical array.

If the vector data consists of polygons (patches), the gridded outlines are all hollow. You can differentiate them using the `encodem` function, calling it with an array of rows, columns, and seed values to produce a new grid containing polygonal areas filled with the seed values to replace the binary values generated by `vec2mtx`.

Creating Data Grids from Vector Data

To demonstrate vector-to-raster data conversion, extract patch data for Switzerland from the `worldlo` data set:

1 Use `extractm` to get patch data for the boundary of Switzerland:

```
close all; clear all;
```

```
[swlat,swlon] = extractm(worldlo('POpatch'),'Switzerland');
```

2 Set grid density to be 40 cells per degree, and use `vec2mtx` to rasterize the boundary and generate a referencing vector for it:

```
den = 40;
[swgrid,swrv] = vec2mtx(swlat,swlon,den);
whos
  Name         Size                    Bytes  Class

  den          1x1                         8  double array
  swgrid      80x186                  119040  double array
  swlat       81x1                       648  double array
  swlon       81x1                       648  double array
  swrv         1x3                        24  double array

Grand total is 15046 elements using 120368 bytes
```

The resulting grid contains doubles, and has 80 rows and 186 columns.

3 Make a map of the data grid in contrasting colors:

```
axesm eqdcyl
meshm(swgrid,swrv)
colormap jet(4)
```

4 Set up the map limits:

```
[latlim lonlim]=limitm(swgrid,swrv);
setm(gca, 'Flatlimit', latlim, 'FlonLimit', lonlim)
tightmap
```

5 To fill (recode) the interior of Switzerland, you need a seed point (which must be identified by row and column) and a seed value (to be allocated to all cells within the polygon). Select the middle row and column of the grid and choose an index value of 3 to identify Swiss territory when calling `encodem` to generate a new grid:

```
swpt = [size(swgrid)/2, 3]
swpt =
    40    93     3
```

```
swgrid3 = encodem(swgrid,swpt,1)
```

The last argument (`1`) identifies the code for boundary cells, where filling should halt.

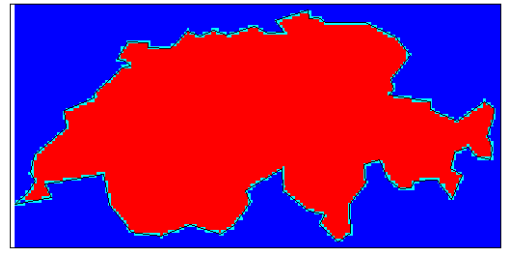
6 Clear and redraw the map using the filled grid:

```
meshm(swgrid3,swrv)
```

7 Plot the original vectors on the grid to see how well data was rasterized:

```
plotm(swlat,swlon,'k')
```

The resulting map is shown below. You can use the zoom tool on the figure window to examine the gridding results more closely.



See the reference pages for `vec2mtx` and `encodem` for further information. A related function for gridding point values is `imbedm`.

Using a GUI to Rasterize Polygons

In the previous example, had you wanted to include the countries that border Switzerland, you could also have extracted Germany, Austria, Italy, and France along with Switzerland, and then deleted unwanted areas of these polygons using `maptrimp` (see "Trimming Vector Data to a Rectangular Region" on page 7-30 for specific details on its use). You can use the `seedm` function with seed points found using the `getseeds` GUI to fill multiple polygons after they are gridded:

1 Extract the data for Switzerland and its neighbors, by passing a cell array of their names to `extractm`:

```
close all; clear all;
```

```
pcs={'Switzerland','Germany','Austria', 'Italy', 'France'};
[celat,celon] = extractm(worldlo('POpatch'),pcs);
```

2 Use `maptrimp` to trim the country boundaries to specified limits:

```
[polylat,polylon] = maptrimp(celat,celon,[45 49],[5 11]);
```

3 Rasterize the trimmed polygons at a 1-arc-minute resolution (60 cells per degree), also producing a referencing vector:

```
[cegrid,cerv] = vec2mtx(polylat,polylon,60);
```

4 Set up a map figure and display the binary grid just created:

```
axesm eqdcyl
meshm(cegrid,cerv)
colormap jet(8)
```

5 Use `getseeds` to interactively pick seed points for Switzerland, Germany, Austria, Italy and France, in that order:

```
[row,col,val] = getseeds(cegrid,cerv,5,[3 4 5 6 7])
row =
   119
   197
   136
    40
    49
col =
   177
   240
   339
   303
    33
val =
     3
     4
     5
     6
     7
```

The MATLAB prompt returns after you pick five locations in the figure window. As you chose them yourself, your `row` and `col` numbers will differ.

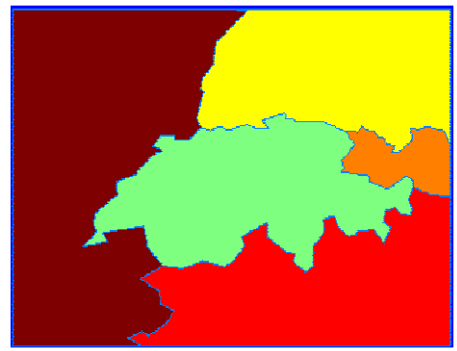
6 Use `encodem` to fill each country with a unique value, producing a new grid, `cegrid5`:

```
cegrid5 = encodem(cegrid,[row col val],1);
```

7 Clear the display and display `cegrid5` to see the result:

```
clma
meshm(cegrid5,cerv)
```

The rasterized map of Switzerland and its neighbors is shown below.



See the reference page for `getseeds` for more information. The GUI tools `maptrim` and `seedm` are also useful in this context.

Data Grids as Logical Variables

You can apply logical criteria to numeric data grids to create *logical grids*. Logical grids are data grids consisting entirely of 1's and 0's. You can create them by performing logical tests on data grid variables. The resulting binary grid is the same size as the original grid(s) and can use the same referencing vector, as the following hypothetical data operation illustrates:

```
logicalgrid = (realgrid>0)
```

This transforms all values greater than zero into 1's and all other values to 0's. You can apply multiple conditions to a grid in one operation:

```
logicalgrid = (realgrid>-100)&(realgrid<100)
```

Should several grids be of the same size and share the same referencing vector (i.e., the grids are in registration), you can create a logical grid by testing joint conditions, treating the individual data grids as map layers:

```
logicalgrid = (population>10000)&(elevation<400)&...
              (country==nigeria)
```

The Mapping Toolbox provides functions enabling the creation of logical grids using logical and relational operators. Grids resulting from such operations contain logical rather than numeric values (which reduce storage by a factor of 8), and might need to be cast to `double` in order to be used in certain functions. The following example shows how you can generate grids of all 1's and all 0's.

Generating "Blank" Logical Grids

Construct a pair of five-cell-per-degree grids. They will contain doubles.

1 Cover the conterminous United States with a grid of 1's; define the country's bounding latitudes and longitudes and the grid resolution:

```
latlims = [25 55]; longlims = [-130 -60]; scale = 5;
```

2 Generate a grid of all 1's over this region at 1/5-degree resolution:

```
onesgrid = onem(latlims,longlims,scale);
```

3 Generate a grid of all 0's over this region at 1/5-degree resolution:

```
zerosgrid = zerom(latlims,longlims,scale);
```

Turn the grids into logical-valued grids and note the difference in size:

```
lonesgrid = logical(onesgrid);
lzerosgrid = logical(zerosgrid);
whos
  Name              Size                    Bytes  Class

  latlims           1x2                        16  double array
  lonesgrid       150x350                   52500  logical array
  longlims          1x2                        16  double array
  lzerosgrid      150x350                   52500  logical array
  onesgrid        150x350                  420000  double array
  scale             1x1                         8  double array
  zerosgrid       150x350                  420000  double array
```

```
Grand total is 210008 elements using 945064 bytes
```

4 Create a referencing vector for mapping the grids:

```
gridref = [5 latlim(2) longlim(1)]
gridref =

     5    55  -130
```

Remember that referencing vectors take the form

```
[cells-per-degree northern-latitude western-longitude]
```

See the references pages for `onem` and `zerom` for more details. You can create grids of all `NaN`s and sparse grids of all 0's in a similar fashion with the commands `nanm` and `spzerom`, respectively.

Obtaining the Area Occupied by a Logical Grid Variable

You can analyze the results of logical grid manipulations to determine the area satisfying one or more conditions (either coded as 1's or an expression that yields a logical value of 1). The command `areamat` can provide the fractional surface area on the globe associated with 1's in a logical grid. Each grid element is a quadrangle, and the sum of the areas meeting the logical condition provides the total area:

1 You can use the `topo` grid and the *greater-than* relational operator to determine what fraction of the Earth lies above sea level:

```
load topo
a = areamat((topo>0),topolegend)
a =
   0.2890
```

The answer is about 30% (note that land areas below sea level are excluded).

2 You can include a planetary radius in specified units if you want the result to have those units. Here is the same query specifying units of square kilometers:

```
a = areamat((topo>0),topolegend,almanac('earth','radius'))
a =
   1.4739e+08
```

3. Use the `worldmtx` data grid that codes countries and water to find the area of a specific country. Here you determine the area of the U.S., which is coded as `183` in this data grid:

```
load worldmtx
a = areamat((map==183),maplegend, almanac('earth','radius'))
a =
  7.6677e+006
```

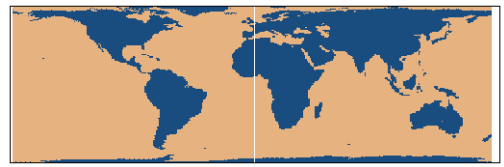
The grid codes 7.67 million square kilometers of land area as belonging to the U.S.

4. You can construct more complex queries if you want to know the extent of coverages over portions of the earth's surface. Using the last example, you can compute what portion of the land area of the earth the U.S. occupies (water is coded with 2's):

```
usland=areamat((map==183),maplegend)/areamat((map~=2),maplegend)
usland =
     0.045293
```

This indicates that the U.S. occupies roughly 4.5% of the Earth's land area.

To interpret this graphically, remember that the logical data grid (`topo>0`) is binary. The following is a display of the grid with purple where the condition is true (i.e., the logical grid entry is 1) and orange where it is false (0). It is displayed in an equal-area cylindrical projection, which vertically shrinks rows near the poles to show their true relative area. The `areamat` command similarly compensates for the changes in area occupied by latitude bands (rows) in computing cell areas.



For further information, see the reference page for `areamat`.

Data Grid Values Along a Path

A common application for gridded geodata is to calculate data values along a path, for example, the computation of terrain height along a transect, a road, or a flight path. The `mapprofile` function does this, based on numerical data defining a set of waypoints or by defining them interactively via graphic input from a map display. Values computed for the resulting profile can be displayed in a new plot or returned as output arguments for further analysis or display.

Using the mapprofile Function

The following example computes the elevation profile along a straight line:

1. Load the Korean elevation data:

   ```
   clear all; close all;
   load korea
   ```

2. Get its latitude and longitude limits using `limitm`: and use them to set up a map frame via `worldmap`:

   ```
   [latlim,lonlim] = limitm(map,maplegend);
   worldmap(latlim,lonlim,'none')
   ```

3. Render the map and apply a digital elevation model (DEM) colormap to it:

   ```
   meshm(map,maplegend,size(map),map)
   demcmap(map)
   ```

4. Define endpoints for a straight-line transect through the region:

   ```
   plat = [40.5 30.7];
   plon = [121.5 133.5];
   ```

5. Now compute the elevation profile, defaulting the track type to `great circle` and the interpolation type to `bilinear`:

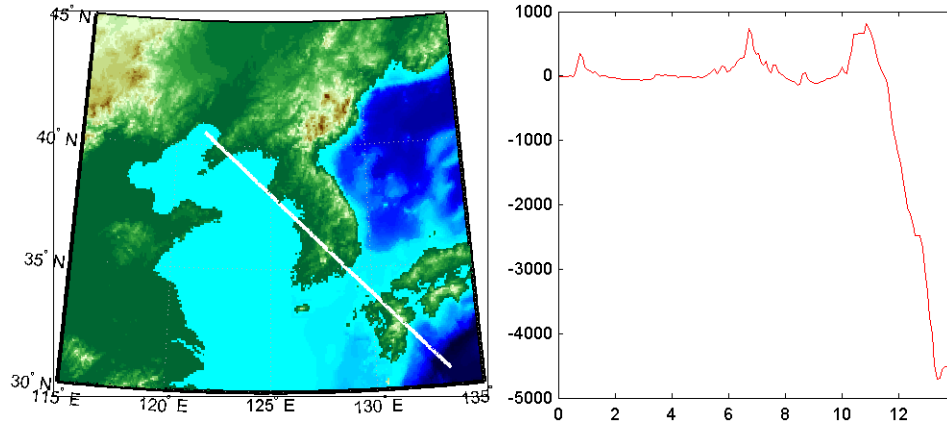
   ```
   [z,rng,lat,lon] = mapprofile(map,maplegend,plat,plon);
   ```

6. Draw the transect in 3-D so it follows the terrain:

   ```
   plot3m(lat,lon,z,'w','LineWidth',2)
   ```

7. Construct a plot of transect elevation and range:

   ```
   figure; plot(rng,z,'r')
   ```



The `mapprofile` function has other useful options, including the ability to interactively define tracks and specify units of distance for them. For further information, see the `mapprofile` reference page.

Data Grid Gradient, Slope, and Aspect

A map profile is often used to determine slopes along a path. A related application is the calculation of slope at all points on a matrix. The `gradientm` command uses a finite-difference approach to compute gradients for either a regular or a georeferenced data grid. The function returns the components of the gradient in the north and east directions (i.e., north-to-south, east-to-west), as well as slope and aspect. The *gradient* components are the change in the grid variable per meter of distance in the north and east directions. If the grid contains elevations in meters, the *aspect* and *slope* are the angles of the surface normal clockwise from north and up from the horizontal. Slope is defined as the change in elevation per unit distance along the path of steepest ascent or descent from a grid cell to one of its eight immediate neighbors, expressed as the arctangent. The angles are in units of degrees by default.

Computing Gradient Data from a Regular Data Grid

The following example illustrates computation of gradient, slope, and aspect data grids for a regular data grid based on the MATLAB `peaks` function:

1. Construct a 100-by-100 grid using the MATLAB `peaks` function and construct a referencing vector for it:

```
clear all; close all;
datagrid = 500*peaks(100);
gridrv = [ 1000 0 0];
```

2. Use `gradientm` to generate grids containing aspect, slope, and gradients to north, and gradients to east:

```
[aspect,slope,gradN,gradE] = gradientm(datagrid,gridrv);
whos
  Name           Size                    Bytes  Class

  aspect       100x100                   80000  double array
  datagrid     100x100                   80000  double array
  gradE        100x100                   80000  double array
  gradN        100x100                   80000  double array
  gridrv         1x3                        24  double array
  slope        100x100                   80000  double array

Grand total is 50004 elements using 400024 bytes
```

3. Map the surface data in a cylindrical equal area projection. Start with the original elevations:

```
axesm eqacyl
meshm(datagrid,gridrv)
colormap (jet(64))
colorbar('vert');
```

4. Clear the frame and display the `slope` grid:

```
clma
meshm(slope,gridrv)
colorbar('vert');
```

5. Map the `aspect` grid:

```
clma
meshm(aspect,gridrv)
colorbar('vert');
```

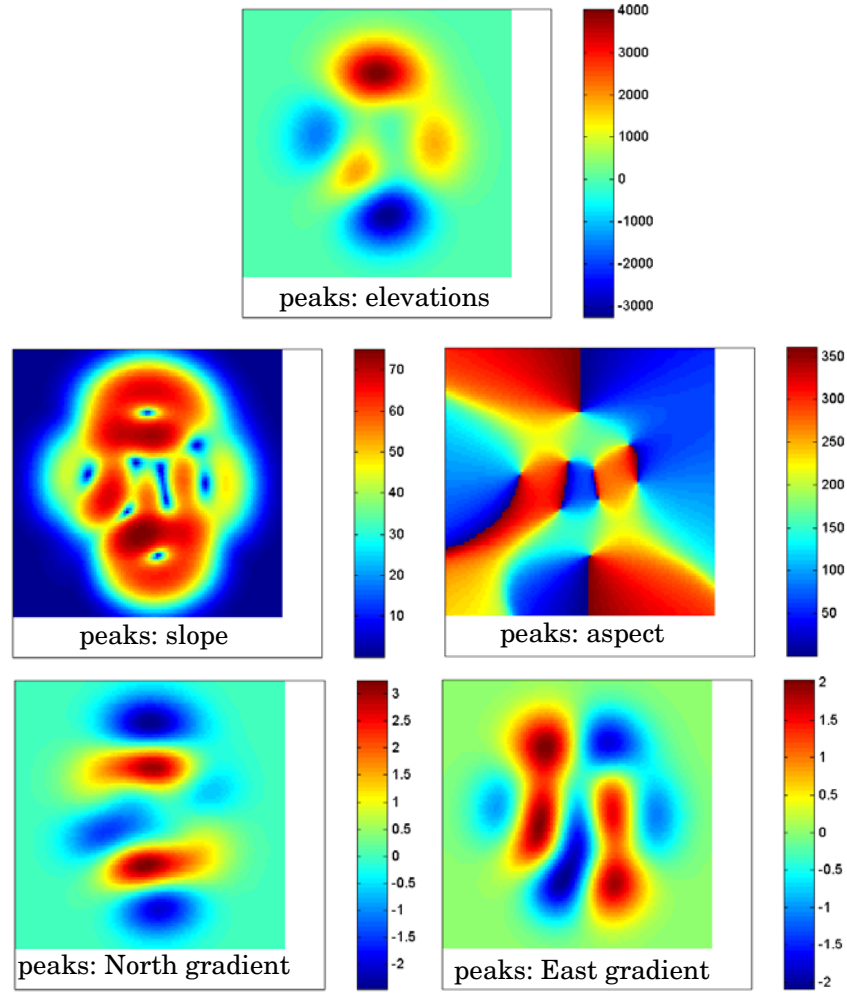
6 Map the gradients to the north:

```
clma
meshm(gradN,gridrv)
colorbar('vert');
```

7 Finally, map the gradients to the east:

```
clma
meshm(gradE,gridrv)
colorbar('vert');
```

The maps of the `peaks` surface elevation and gradient data are shown below. See the `gradientm` reference page for additional information.


peaks: elevations
4000
3000
2000
1000
0
-1000
-2000
-3000
peaks: slope
70
60
50
40
30
20
10
peaks: aspect
350
300
250
200
150
100
50
peaks: North gradient
3
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
peaks: East gradient
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2

8

Using Map Projections and Coordinate Systems

All geospatial data must be flattened onto a display surface in order to visually portray what exists where. The mathematics and craft of map projection is central to this process. Although there is no limit to the ways geodata can be projected, conventions, constraints, standards, and applications generally prescribe its usage. This chapter describes what map projections are, how they are constructed and controlled, their essential properties, and some possibilities and limitations.

If you are not acquainted with the types, properties, and uses of map projections, you should read the first four sections. When constructing maps — especially in an environment such as the Mapping

Toolbox, in which a variety of projections are readily available and easily substituted — it is important to understand how to evaluate projections to select one appropriate to the contents and purpose of a given map. The larger the area being mapped, the more important these choices are.

What Is a Map Projection?

Human beings have known that the shape of the Earth resembles a sphere and not a flat surface since classical times, and possibly much earlier than that. If the world were indeed flat, cartography would be much simpler because map projections would be unnecessary.

To represent a curved surface such as the Earth in two dimensions, you must geometrically transform (literally, and in the mathematical sense, "map") that surface to a plane. Such a transformation is called a *map projection*. The term projection derives from the geometric methods that were traditionally used to construct maps, in the fashion of optical projections made with a device called *camera obscura* that Renaissance artists relied on to render three-dimensional perspective views on paper and canvas.

While many map projections no longer rely on physical projections, it is useful to think of map projections in geometric terms. This is because map projection consists of constructing points on geometric objects such as cylinders, cones, and circles that correspond to homologous points on the surface of the planet being mapped according to certain rules and formulas.

The following sections describe the basic properties of map projections, the surfaces onto which projections are developed, the types of parameters associated with different classes of projections, how projected data can be mapped back to the sphere or spheroid it represents, and details about one very widely used projection system, called Universal Transverse Mercator.

For more detailed information on specific projections, browse the "Projections Reference" chapter. For further reading, the "Bibliography" provides references to books and papers on map projection.

Quantitative Properties of Map Projections

A sphere, unlike a polyhedron, cone, or cylinder, cannot be reformed into a plane. In order to portray the surface of a round body on a two-dimensional flat plane, you must first define a *developable surface* (i.e., one that can be *cut* and *flattened* onto a plane without stretching or creasing) and devise rules for systematically representing all or part of the spherical surface on the plane. Any such process inevitably leads to distortions of one kind or another. Five essential characteristic properties of map projections are subject to distortion: *shape*, *distance*, *direction*, *scale*, and *area*. No projection can retain more than one of these properties over a large portion of the Earth. This is not because a sufficiently clever projection has yet to be devised; the task is physically impossible. The technical meanings of these terms are described below.

- Shape (also called *conformality*)

  Shape is preserved locally (within "small" areas) when the scale of a map at any point on the map is the same in any direction. Projections with this property are called conformal. In them, meridians (lines of longitude) and parallels (lines of latitude) intersect at right angles. An older term for conformal is *orthomorphic* (from the Greek *orthos*, straight, and *morphe*, shape).

- Distance (also called *equidistance*)

  A map projection can preserve distances from the center of the projection to all other places on the map (but from the center only). Such a map projection is called *equidistant*. Maps are also described as equidistant when the separation between parallels is uniform (e.g., distances along meridians are maintained). No map projection maintains distance proportionality in all directions from any arbitrary point.

- Direction

  A map projection preserves direction when azimuths (angles from the central point or from a point on a line to another point) are portrayed correctly in all directions. Many azimuthal projections have this property.

- Scale

  Scale is the ratio between a distance portrayed on a map and the same extent on the Earth. No projection faithfully maintains constant scale over large areas, but some are able to limit scale variation to one or two percent.

8 Area (also called *equivalence*)

A map can portray areas across it in proportional relationship to the areas on the Earth that they represent. Such a map projection is called equal-area or equivalent. Two older terms for equal-area are *homolographic* or *homalographic* (from the Greek *homalos* or *homos*, same, and *graphos*, write), and *authalic* (from the Greek *autos*, same, and *ailos*, area), and *equireal*. Note that no map can be both equal-area and conformal.

For a complete description of the properties that specific map projections maintain, see "Summary and Guide to Projections" on page 8-53.

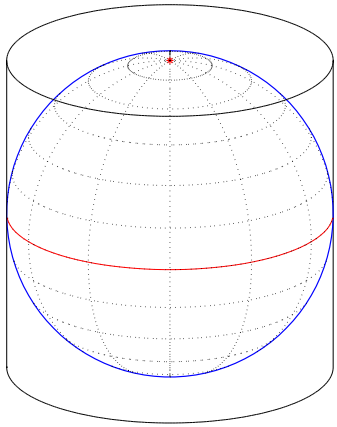
The Three Main Families of Map Projections

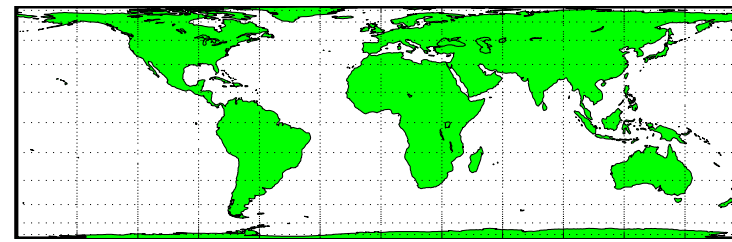
Mapmakers have developed hundreds if not thousands of map projections, over hundreds if not thousands of years. Three large families of map projection, plus several smaller ones, are generally acknowledged. These are based on the types of geometric shapes that are used to transfer features from a sphere or spheroid to a plane. As described above, they are known as *developable surfaces*, and the three traditional families consist of cylinders, cones, and planes. They are used to classify the majority of projections, including some that are not analytically (geometrically) constructed. In addition, a number of map projections are based on polyhedra. While polyhedral projections have interesting and useful properties, they are not described here.

The following sections describe and illustrate the cylindrical, conic and azimuthal families of map projections.

Cylindrical Projections

A *cylindrical* projection is produced by wrapping a cylinder around a globe representing the Earth. The map projection is the image of the globe projected onto the cylindrical surface, which is then unwrapped into a flat surface. When the cylinder aligns with the polar axis, parallels appear as horizontal lines and meridians as vertical lines. Cylindrical projections can be either equal-area, conformal, or equidistant. The following figure shows a regular cylindrical or *normal aspect* orientation in which the cylinder is tangent to the Earth along the Equator and the projection radiates horizontally from the axis of rotation. The projection method is diagrammed on the left, and an example is given on the right (Equal-area cylindrical projection, normal/equatorial aspect).





For a description of projection aspect, see “Projection Aspect” on page 8-10.

Some widely-used cylindrical map projections are

- Equal-area cylindrical projection
- Equidistant cylindrical projection
- Mercator projection
- Miller projection
- Plate carree projection
- Universal transverse Mercator projection

Pseudocylindrical Map Projections

All cylindrical projections fill a rectangular plane. *Pseudocylindrical* projection outlines tend to be barrel-shaped rather than rectangular. However, they do resemble cylindrical projections, with straight and parallel latitude lines, and can have equally spaced meridians, but meridians are curves, not straight lines. Pseudocylindrical projections can be equal-area, but are not conformal or equidistant.

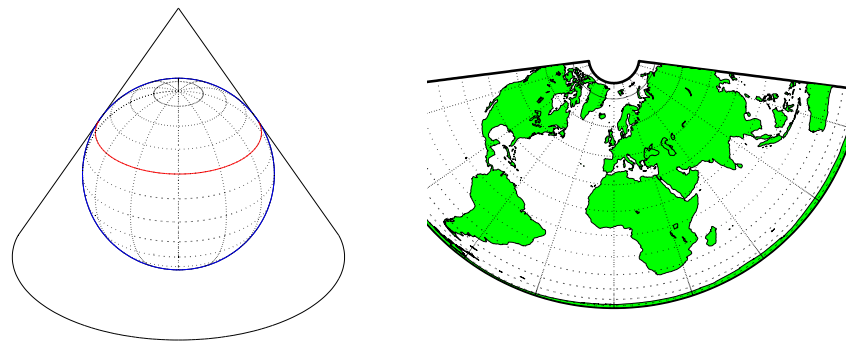
Some widely-used pseudocylindrical map projections are

- Ekert projections (I-VI)
- Goode homolosin projection

- Mollweide projection
- Quartic authalic projection
- Robinson projection
- Sinusoidal projection

Conic Projections

A *conic* projection is derived from the projection of the globe onto a cone placed over it. For the *normal aspect*, the apex of the cone lies on the polar axis of the Earth. If the cone touches the Earth at just one particular parallel of latitude, it is called *tangent*. If made smaller, the cone will intersect the Earth twice, in which case it is called *secant*. Conic projections often achieve less distortion at mid- and high latitudes than cylindrical projections. A further elaboration is the *polyconic* projection, which deploys a family of tangent or secant cones to bracket a succession of bands of parallels to yield even less scale distortion. The following figure illustrates conic projection, diagramming its construction on the left, with an example on the right (Albers equal-area projection, polar aspect).

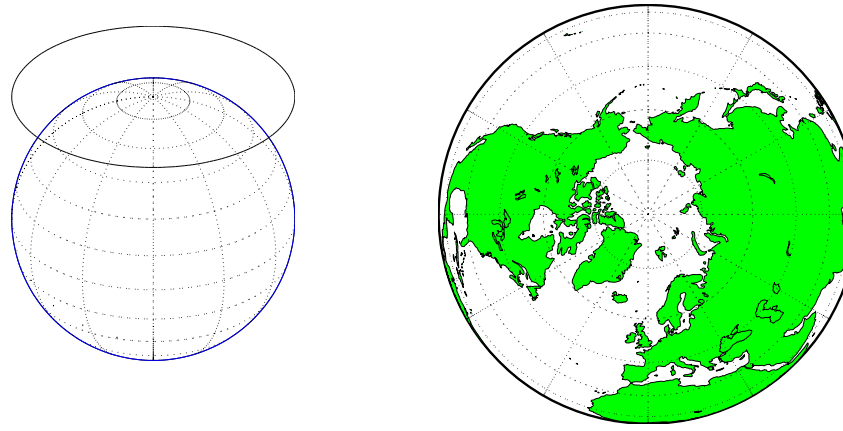


Some widely-used conic projections are:

- Albers Equal-area projection
- Equidistant projection
- Lambert conformal projection
- Polyconic projection

Azimuthal Projections

An *azimuthal* projection is a projection of the globe onto a plane. In polar aspect, an azimuthal projection maps to a plane tangent to the Earth at one of the poles, with meridians projected as straight lines radiating from the pole, and parallels shown as complete circles centered at the pole. Azimuthal projections (especially the orthographic) can have equatorial or oblique aspects. The projection is centered on a point, that is either on the surface, at the center of the Earth, at the antipode, some distance beyond the Earth, or at infinity. Most azimuthal projections are not suitable for displaying the entire Earth in one view, but give a sense of the globe. The following figure illustrates azimuthal projection, diagramming it on the left, with an example on the right (orthographic projection, polar aspect).



Some widely used azimuthal projections are

- Equidistant azimuthal projection
- Gnomonic projection
- Lambert equal-area azimuthal projection
- Orthographic projection
- Stereographic projection
- Universal polar stereographic projection

For additional information on families of map projections and specific map projections, see the “Projections Reference” chapter.

Projection Aspect

A map projection's *aspect* is its orientation on the page or display screen. If north or south is straight up, the aspect is said to be *equatorial*; for most projections this is the *normal* aspect. When the central axis of the developable surface is oriented east-west, the projection's aspect is *transverse*. Projections centered on the North Pole or the South Pole have a *polar* aspect, regardless of what meridian is up. All other orientations have an *oblique* aspect. So far, the examples and discussions of map displays have focused on the normal aspect, by far the most commonly used. This section discusses the use of *transverse*, *oblique*, and *skew-oblique* aspects.

Projection aspect is primarily of interest in the display of maps. However, this section also discusses how the idea of projection aspect as a coordinate system transformation can be applied to map variables for analytical purposes.

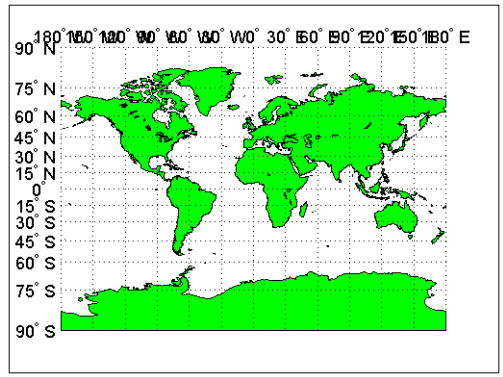
The Orientation Vector

A map axes `Origin` property is a vector describing the geometry of the displayed projection. The Mapping Toolbox calls this property the *orientation vector* (prior versions called it the *origin vector*). The vector takes this form:

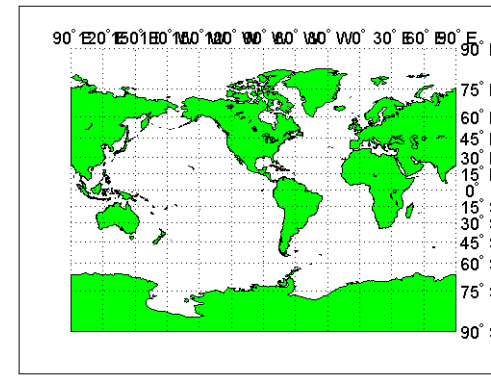
```
orientvec = [latitude longitude orientation]
```

The latitude and longitude represent the geographic coordinates of the center point of the display from which the projection is calculated. The orientation refers to the clockwise angle from *straight up* at which the North Pole points from this center point. The default orientation vector is [0 0 0]; that is, the projection is centered on the geographic point (0°,0°) and the North Pole is *straight up* from this point. Such a display is in a *normal* aspect. Changes to only the longitude value of the orientation vector do not change the aspect; thus, a normal aspect is one centered on the Equator in latitude with an orientation of 0°.

Both of these Miller projections have normal aspects, despite having different orientation vectors:



Origin at (0°,0°), with a 0° Orientation.
(orientation vector = [0 0 0])



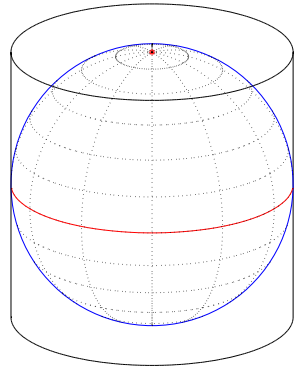
Origin at (0°,90°W), with a 0° Orientation.
(orientation vector = [0 -90 0])

This makes sense if you think about a simple, true cylindrical projection. This is the projection of the globe onto a cylinder wrapped around it. For normal aspects, this cylinder is tangent to the globe at the Equator, and changing the origin longitude simply corresponds to rotating the sphere about the longitudinal axis of the cylinder. If you continue with the wrapped-cylinder model, you can understand the other aspects as well.

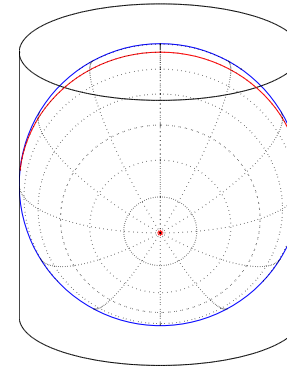
Following this description, a *transverse* projection can be thought of as a cylinder wrapped around the globe tangent at the poles and along a meridian and its antipodal meridian. Finally, when such a cylinder is tangent along any great circle other than a meridian, the result is an *oblique* projection.

Here are diagrams of the four cylindrical map orientations, or aspects:

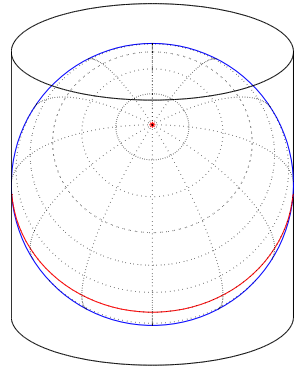
Normal



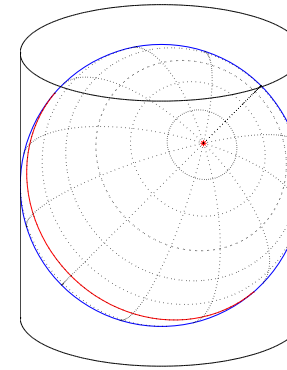
Transverse



Oblique



Skew-Oblique



Of course, few projections are true cylindrical projections, but the concept of the wrapped cylinder is nonetheless a convenient way to describe aspect.

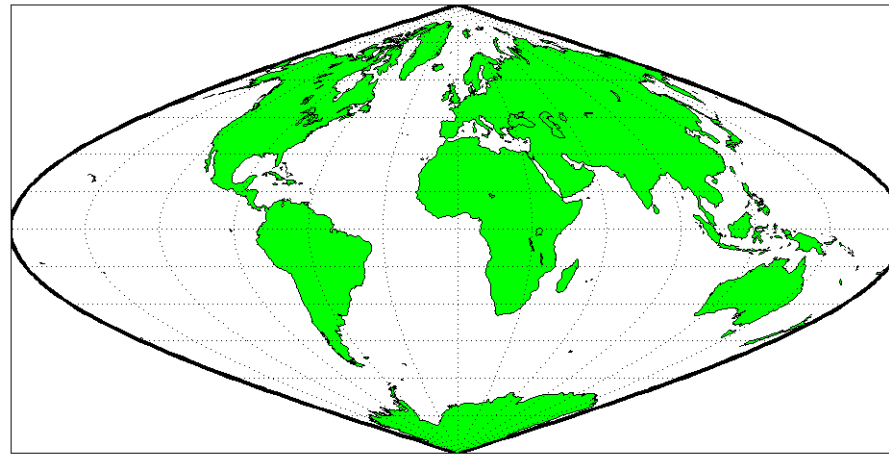
Exploring Projection Aspect

Perhaps the best way to gain an understanding of projection aspect is to experiment with orientation vectors. For the following exercise, use a pseudocylindrical projection, the sinusoidal.

1 Create a default map axes in a sinusoidal projection, turn on the graticule, and display the `coast` data set as filled polygons:

```
close all; clear all;
axesm sinusoid
framem on; gridm on; tightmap tight
load coast
patchm(lat, long,'g')
```

The continents and graticule appear in normal aspect, as shown below.



Normal aspect: origin at (0°,0°), orientation 0°
(orientation vector = [0 0 0])

2 Inspect the orientation vector from the map axes:

```
getm(gca,'Origin')
ans =
     0     0     0
```

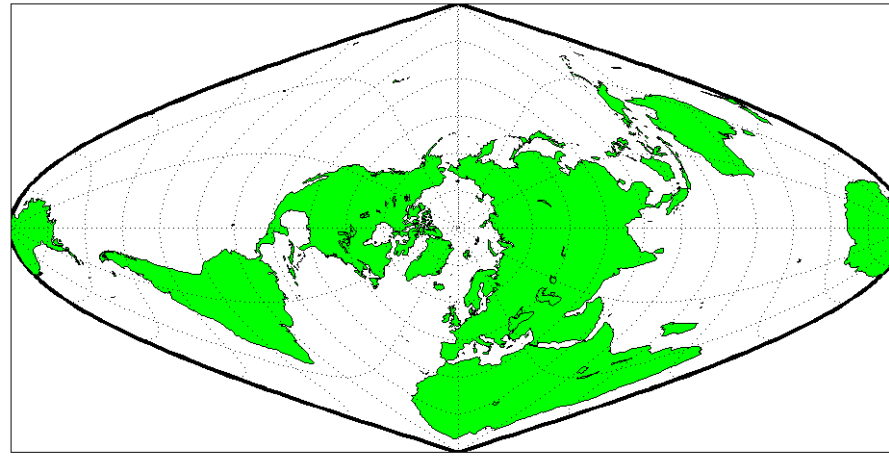
By default, the origin is set at (0°E, 0°N), oriented 0° from vertical.

3 In the normal aspect, the North Pole is at the *top* of the image. To create a transverse aspect, imagine pulling the North Pole down to the center of the

display, which was originally occupied by the point (0°,0°). Do this by setting the first element of `Origin` parameter to a latitude of 90°N:

```
setm(gca,'Origin',[90 0 0])
```

The shape of the frame is unaffected; this is still a sinusoidal projection.

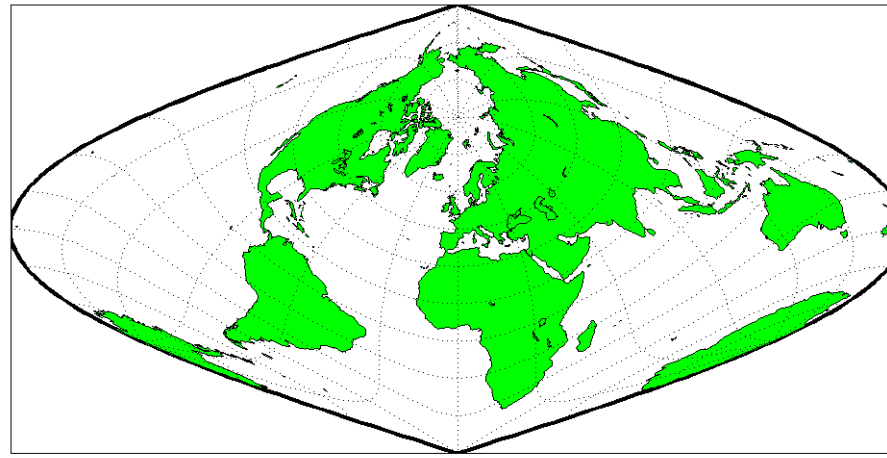


Transverse aspect: origin at (90°N, 0°), orientation 0° (orientation vector = [90 0 0])

4 The normal and transverse aspects can be thought of as limiting conditions. Anything else is an oblique aspect. Conceptually, if you push the North Pole halfway back to its original position (to the position originally occupied by the point (45°N, 0°E) in the normal aspect), the result is a simple oblique aspect.

```
setm(gca,'Origin',[45 0 0])
```

The oblique sinusoidal projection centered at (45°N, 0°E) is shown below.



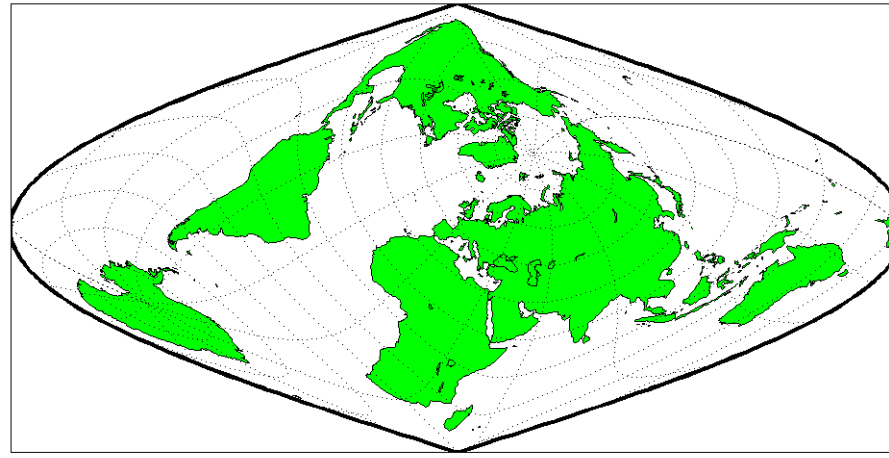
Oblique aspect: origin at (45°N,0°), orientation 0°
(orientation vector = [45 0 0])

You can think of this as pulling the new origin (45°N, 0°) to the center of the image, the place that (0°,0°) occupied in the normal aspect.

5 The previous examples of projection aspect kept the aspect orientation at 0°. If the orientation is altered, an oblique aspect becomes a *skew-oblique*. Imagine the previous example with an orientation of 45°. Think of this as pulling the new origin (45°N,0°E), down to the center of the projection and then rotating the projection until the North Pole lies at an angle of 45° clockwise from straight up with respect to the new origin.

```
setm(gca,'Origin',[45 0 45])
```

As in the previous example, the location (45°N,0°E) still occupies the center of the map.



Skew-oblique aspect: origin at (45°N,0°), orientation 45° (orientation vector = [45 0 45])

Any projection can be viewed in alternate aspects. Some of these are quite useful. For example, the transverse aspect of the Mercator projection is widely used in cartography, especially for mapping regions with predominantly north-south extent. One candidate for such handling might be Chile. Oblique Mercator projections might be used to map long regions that run neither north and south nor east and west, such as New Zealand.

Note The projection aspect discussed in this section is different from the map axes `Aspect` property. The map axes `Aspect` property controls the orientation of the figure axes. For instance, if a map is in a `normal` setting with a *landscape* orientation, a switch to a `transverse` aspect rotates the axes by 90°, resulting in a *portrait* orientation. To display a map in the transverse aspect, combine the `transverse` aspect property with a -90° skew angle. The skew angle is the last element of the `Origin` parameter. For example, a `[0 0 -90]` vector would produce a transverse map.

The base projection can be thought of as a standard coordinate system, and the normal aspect conforms to it. The features of a projection are maintained in any

aspect, *relative to the base projection*. As the preceding illustrations show, the *outline* (frame) does not change. Nondirectional projection characteristics also do not change. For example, the sinusoidal projection is equal-area, no matter what its aspect. Directional characteristics must be considered carefully, however. In the normal aspect of the sinusoidal projection, scale is true along every parallel and the central meridian. This is not the case for the skew-oblique aspect; however, scale is true along the paths of the transformed parallels and meridian.

Projection Parameters

Every projection has at least one parameter that controls how it transforms geographic coordinates into planar coordinates. Some projections are rather fixed, and aside from the orientation vector and nominal scale factor, have no parameters that the user should vary, as to do so would violate the definition of the projection. For example, the Robinson projection has one standard parallel that is fixed by definition at 38° North and South; the Cassini and Wetch projections cannot be constructed in other than Normal aspect. In general, however, projections have several variable parameters. The following section discusses map projection parameters and provides guidance for setting them.

Projection Characteristics Maps Can Have

In addition to the name of the projection itself, the parameters that a map projection can have are:

- *Aspect* — Orientation of the projection on the display surface
- *Center* or *Origin* — Latitude and longitude of the midpoint of the display
- *Scale Factor* — Ratio of distance on the map to distance on the ground
- *Standard Parallel(s)* — Chosen latitude(s) where scale distortion is zero
- *False Northing* — Planar offset for coordinates on the vertical map axis
- *False Easting* — Planar offset for coordinates on the horizontal map axis
- *Zone* — Designated latitude-longitude quadrangle used to systematically partition the planet for certain classes of projections

While not all projections require all these parameters, there will always be a projection aspect, origin, and scale.

Other parameters are associated with the graphic expression of a projection, but do not define its mathematical outcome. These include

- Map latitude and longitude limits
- Frame latitude and longitude limits

However, as certain projections are unable to map an entire planet, or become very distorted over large regions, these limits are sometimes a necessary part of setting up a projection.

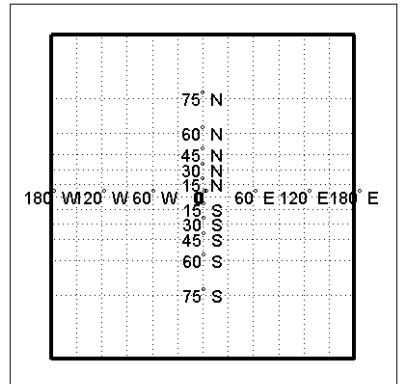
Determining Projection Parameters

In the following exercise, you define a map axes and examine default parameters for a cylindrical, a conic, and an azimuthal projection.

1 Set up a default Mercator projection (which is cylindrical) and pass its handle to the `getm` function to query projection parameters:

```
close all; clear all;
h=axesm('Mapprojection','mercator','Grid','on','Frame','on',...
'MlabelParallel',0, 'PlabelMeridian',0, 'mlabellocation',60,...
'meridianlabel','on', 'parallellabel','on')
```

The graticule and frame for the default map projection are shown below.



2 Query the map axes handle using `getm` to inspect the properties that pertain to map projection parameters. The principal ones are `aspect`, `origin`, `scalefactor`, `nparallels`, `mapparallels`, `falsenorthing`, `falseeasting`, `zone`, `maplatlimit`, `maplonlimit`, `rlatlimit`, and `flonlimit`:

```
getm(h,'aspect')
ans =
     normal
getm(h,'origin')
ans =
     0     0     0
getm(h,'scalefactor')
ans =
     1
```

```
getm(h,'nparallels')
ans =
     1
getm(h,'mapparallels')
ans =
     0
getm(h,'falsenorthing')
ans =
     0
getm(h,'falseeasting')
ans =
     0
getm(h,'zone')
ans =
     []
getm(h,'maplatlimit')
ans =
   -86    86
getm(h,'maplonlimit')
ans =
  -180   180
getm(h, 'Flatlimit')
ans =
   -86    86
getm(h, 'Flonlimit')
ans =
  -180   180
```

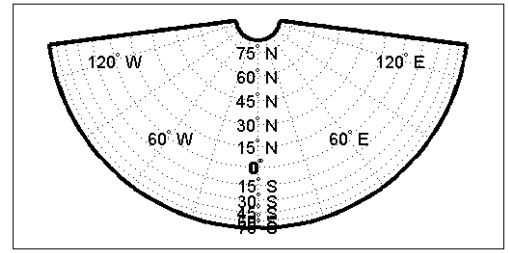
For more information on these and other map axes properties, see the reference page for `axesm`.

3 Reset the projection type to equal-area conic (`'eqaconic'`). The figure is redrawn to reflect the change. Determine the parameters the Mapping Toolbox changes in response:

```
setm(h,'Mapprojection', 'eqaconic')
getm(h,'aspect')
ans =
normal
getm(h,'origin')
ans =
```

```
     0     0     0
getm(h,'scalefactor')
ans =
     1
getm(h,'nparallels')
ans =
     2
getm(h,'mapparallels')
ans =
    15    75
getm(h,'falsenorthing')
ans =
     0
getm(h,'falseeasting')
ans =
     0
getm(h,'zone')
ans =
     []
getm(h,'maplatlimit')
ans =
   -86    86
getm(h,'maplonlimit')
ans =
  -135   135
getm(h, 'Flatlimit')
ans =
   -86    86
getm(h, 'Flonlimit')
ans =
  -135   135
```

The `eqaconic` projection has two standard parallels, at 15° and 75°. It also has reduced longitude limits (covering 270° rather than 360°). The resulting `eqaconic` graticule is shown below.

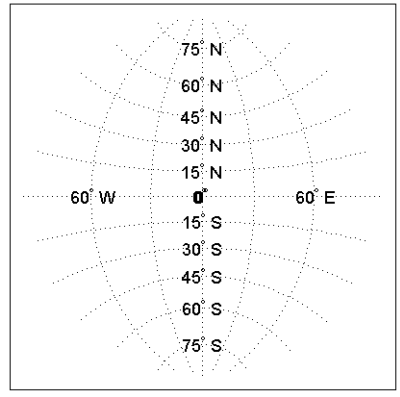


4 Now set the projection type to Stereographic (`'stereo'`) and examine the same properties as you did for the previous projections:

```
setm(h,'Mapprojection', 'stereo')
getm(h,'aspect')
ans =
normal
getm(h,'origin')
ans =
     0     0     0
getm(h,'scalefactor')
ans =
     1
getm(h,'nparallels')
ans =
     0
getm(h,'mapparallels')
ans =
     []
getm(h,'falsenorthing')
ans =
     0
getm(h,'falseeasting')
ans =
     0
getm(h,'zone')
ans =
     []
getm(h,'maplatlimit')
ans =
```

```
   -86    86
getm(h,'maplonlimit')
ans =
  -135   135
getm(h, 'Flatlimit')
ans =
   -86    86
getm(h, 'Flonlimit')
ans =
  -135   135
```

The stereographic projection, being azimuthal, does not have standard parallels, so none are indicated. The map limits do not change from the previous projection. The map figure is shown below.



The “Projections Reference” chapter lists all map projections supported by the Mapping Toolbox, including suggestions for parameter usage.

Visualizing and Quantifying Projection Distortions

Because no projection can preserve all directional and nondirectional geographic characteristics, it is useful to be able to estimate the degree of error in direction, area, and scale for a particular projection type and parameters used. The Mapping Toolbox provides several functions that map projection distortions, and one that computes distortion metrics for specified locations.

Displays of Spatial Error in Maps

A standard method of visualizing the distortions introduced by the map projection is to display small circles at regular intervals across the globe. After projection, the small circles appear as ellipses of various sizes, elongations, and orientations. The sizes and shapes of the ellipses reflect the projection distortions. Conformal projections have circular ellipses, while equal-area projections have ellipses of the same area. This method was invented by Nicolas Tissot in the 19th century, and the ellipses are called *Tissot indicatrices* in his honor. The measure is a tensor function of location that varies from place to place, and reflects the fact that, unless a map is conformal, map scale is different in every direction at a location.

Visualizing Projection Distortions via Tissot Indicatrices

As the following example illustrates, you can add the indicatrices to a map display with the command `tissot` and remove them with `clmo tissot`:

1. Set up a Sinusoidal projection in a skewed aspect, plotting the graticule:

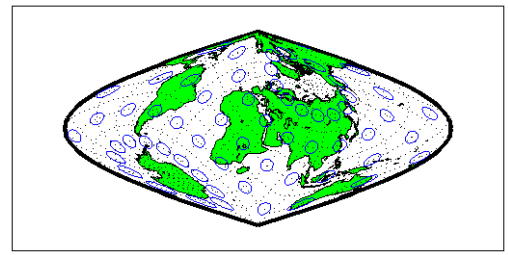
```
close all; clear all;
axesm sinusoid
gridm on;framem on;
setm(gca,'Origin', [20 30 45])
```

2. Load the `coast` data set and plot it as green patches:

```
load coast
patchm(lat, long,'g')
```

3. Plot the default Tissot diagram, shown below:

```
tissot
```



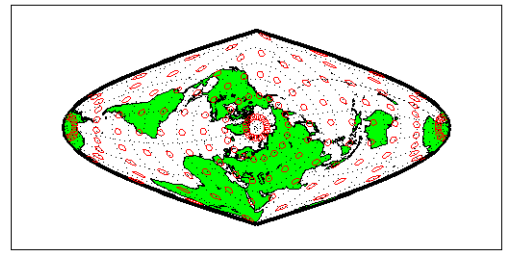
Notice that the circles vary considerably in shape. This indicates that the Sinusoidal projection is not conformal. Despite the distortions, however, the circles all cover equal amounts of area on the map, because the projection has the equal-area property.

Default Tissot diagrams are drawn with blue unfilled 100-point circles spaced 30 degrees apart in both directions. The default circle radius is 1/10 of the current radius of the referencing vector (by default that radius is 1).

4 Now clear the Tissot diagram, rotate the projection to a polar aspect, and plot a new Tissot diagram using circles paced 20 degrees apart, half as big as before, drawn with 20 points, and drawn in red:

```
clmo tissot
setm(gca, 'Origin', [90 0 45])
tissot([20 20 .05 20], 'Color','r')
```

The result is shown below. Note that circles are drawn faster because fewer points are computed for each one. Also note that the distortions are still smallest close to the map origin, and still greatest near the map frame.



Try changing the map projection to a conformal one such as Mercator or Stereographic to see what Tissot indicatrices look like on shape-preserving maps.

For further information, see the reference page for `tissot`.

Visualizing Projection Distortions via Isolines

Most map projection distortions are rather orderly and vary continuously, making them suitable for display via isolines (contour lines). In addition to Tissot diagrams, the Mapping Toolbox enables you to plot isolines of variations of several parameters associated with map projections, using `mdistort`.

The `mdistort` function can plot variations in angles, areas, maximum and minimum scale, and scale along parallels and meridians, in units of percent deviation (except for angles, for which degrees are used). Use this function in selecting projections and projection parameters when you are concerned about keeping specific types of distortion within limits. Below are some examples of `mdistort` using the Hammer modified azimuthal projections and the Bonne pseudoconic projection.

1. Create a Hammer projection map axes in normal aspect, and plot a graticule, frame, and coastlines on it:

   ```
   close all; clear all;
   axesm('MapProjection','hammer','Grid', 'on', 'Frame','on')
   ```

2. Load the `coast` data set and plot it as green patches:

   ```
   load coast
   patchm(lat, long,'g')
   ```

3. Call `mdistort` to plot contours of minimum-to-maximum scale ratios:

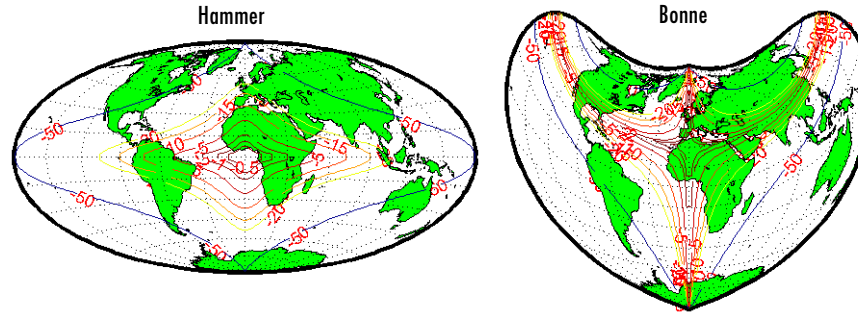
```
mdistort('scaleratio')
```

Notice that the region of minimum distortion is centered around `(0,0)`.

4 Repeat this diagram with a Bonne projection in a new figure window:

```
figure;
axesm('MapProjection','bonne','Grid', 'on', 'Frame','on')
patchm(lat, long,'g')
mdistort('scaleratio')
```

Notice that the region of minimum distortion is centered around `(30,0)`, which is where the single standard parallel is.



Isolines of maximum/minimum scale ratio

5 You can toggle the isolines by typing `mdistort` or `mdistort off`. Look at some other types of distortion. The types you can request are

- `area` — Percent departures from equal area
- `angles` — Angular distortion of right angles
- `scale` or `maxscale` — Percent of maximum scale
- `minscale` — Percent of minimum scale
- `parscale` — Percent of scale along the parallels
- `merscale` — Percent of scale along the meridians
- `scaleratio` — Percent of maximum-to-minimum scale ratio

For further information see the reference page for `mdistort`.

Quantifying Map Distortions at Point Locations

The `tissot` and `mdistort` functions described above provide synoptic visual overviews of different forms of map projection error. Sometimes, however, you need numerical estimates of error at specific locations in order to quantify or correct for map distortions. This is useful, for example, if you are sampling environmental data on a uniform basis across a map, and want to know precisely how much area is associated with each sample point, a statistic that will vary by location and be projection dependent. Once you have this information, you can adjust environmental density and other statistics you collect for areal variations induced by the map projection.

The Mapping Toolbox provides a function to return location-specific map error statistics from the current projection or an mstruct. The `distortcalc` function computes the same distortion statistics as `mdistort` does, but for specified locations provided as arguments. You provide the latitude-longitude locations one at a time or in vectors. The general form is

```
[areascale,angdef,maxscale,minscale,merscale,parscale] = ...
    distortcalc(mstruct,lat,long)
```

However, if you are evaluating the current map figure, omit the mstruct. You need not specify any return values following the last one of interest to you.

Using distortcalc to Determine Map Projection Error

The following exercise uses `distortcalc` to compute the maximum area distortion for a map of Argentina from the `worldlo` data set.

1 Load `worldlo`, extract the outline of Argentina, and delete the rest:

```
close all; clear all;
load worldlo
[alat alon] = extractm(POpatch,'argentina');
clear P* D* d* g*
```

2 Set the spatial extent (map limits) to contain Argentina and also include an area closer to the South Pole:

```
mlatlim = [-72.0 -20.0];
mlonlim = [-75.0 -50.0];
```

3 Create a Mercator cylindrical conformal projection using these limits, specify a five-degree graticule, and then plot the outline for reference:

```
axesm('MapProjection','mercator', 'grid','on', ...
'MapLatLimit',mlatlim, 'MapLonLimit',mlonlim,...
'MLineLocation',5, 'PLineLocation',5)
plotm(alat,alon,'b')
```

4 Sample every tenth point of the patch outline for analysis:

```
alats = alat(1:10:numel(alat));
alons = alon(1:10:numel(alat));
```

5 Compute the area distortions (the first statistic returned by `distortcalc`) at the sample points:

```
aerr = distortcalc(alats, alons);
```

6 Find the range of area distortion across Argentina (percent of a unit area on, in this case, the Equator):

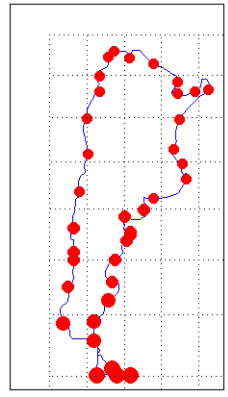
```
aerrmm = [min(aerr) max(aerr)]
aerrmm =
       1.1641       3.0317
```

As Argentina occupies mid southern latitudes, its area on a Mercator map is overstated, and the errors vary noticeably from North to South.

7 Plot the full set of error statistics as proportional circles using `scatterm`:

```
scatterm(alats,alons,20*aerr,'r','filled')
```

The resulting map is shown below:



8 The degree of area overstatement would be considerably larger if it extended farther toward the pole. To see how much larger, get the area distortion statistic for 50°S, 60°S, and 70°S:

```
a=distortcalc(-50,-60)
a =
       2.4203
a=distortcalc(-60,-60)
a =
            4
>> a=distortcalc(-70,-60)
a =
       8.5485
```

Note You can only use `distortcalc` to query locations that are within the current map frame or mstruct limits. Outside points yield `NaN` as a result.

---

9 You can also query distortion at arbitrary locations you select with the graphic curso,r using `inputm`, for example:

```
[plat plon] = inputm(1)
plat =
      -62.225
plon =
      -72.301
>> a=distortcalc(plat,plon)
a =
       4.6048
```

Naturally the answer you get will vary depending on what point you pick. Using this technique, you can write a simple script that lets you query a map repeatedly to determine any distortion statistic at any desired location.

Try changing the map projection or even the orientation vector to see how different projection errors can be. For further information, see the reference page for `distortcalc`.

Accessing, Computing, and Inverting Map Projection Data

Most of the examples in this document assume that the end product of a map projection is a graphical representation as a map, and that the planar coordinates yielded by projection are of little interest. However, there might be times when you need access to projected coordinate data. You might also have projected data that you want to transform back to latitude and longitude (assuming you know its projection parameters). The following sections describe how to retrieve projected data, project it without displaying it, and invert projections.

Accessing Projected Coordinate Data

A MATLAB figure generally contains coordinate data only in its axes child object and in children of axes objects, such as line, patch, and surface objects. See the reference page for `axes` for an overview of this object hierarchy. Note that a map axes can have multiple patch children objects when created with `patchesm`.

You can retrieve projected data from a map axes, but you can also obtain it without having to plot the data or even creating a map axes. The following two exercises illustrate each of these approaches.

Retrieving Projected Coordinates from a Figure

An easy way to retrieve the projected coordinates of a map occupying a figure window is with the MATLAB `get` command. The projected coordinates are stored in the object's `XData` and `YData` properties. The `XData` and `YData` can belong to a child object rather than to the axes themselves, however, as the following exercise demonstrates.

1. Create a Mollweide projection map axes and obtain its handle:

```
close all; clear all;
ha = axesm('mollweid')
```

2. Observe that the axes has no `XData`, `YData`, or `children` information:

```
get(ha,'XData')
??? Error using ==> get
Invalid axes property: 'XData'.

get(ha,'YData')
??? Error using ==> get
Invalid axes property: 'YData'.

get(ha,'children')
ans =
   Empty matrix: 0-by-1
```

3. Display a map frame for the Mollweide projection, obtaining its handle. Confirm that the frame is a child of the axes:

```
hf = framem
hf =
          105
get(ha,'children')
ans =
          105
```

4. Use `get` to extract the *x*-*y* coordinates of the map frame:

```
xf = get(hf,'XData');
yf = get(hf,'YData');
```

   The `xf` and `yf` coordinates are 398-by-1 column vector arrays.

5. Load the `coast` data set and render it with `plotm`, obtaining a handle:

```
load coast
hl = plotm(lat,long)
hl =
          106
get(ha, 'children')
```

```
ans =
          106
          105
```

Note that the line data is also a child of the axes.

6 Retrieve the projected coastline coordinates using handle `hl`:

```
xline = get(hl,'XData');
yline = get(hl,'YData');
```

The `xline` and `yline` coordinates are 1-by-9591 row vector arrays. Inspect their contents before proceeding.

7 The units for projected coordinates are established by the ellipsoid vector. By default, these units are Earth radii, but you can change them at any time using `setm` to control the `geoid` property. For example, set the units to kilometers on a spherical earth with

```
setm(gca,'Geoid', almanac('earth','sphere','kilometers'))
```

Repeat step 6 above to see how this affects coordinate values. For further information on specifying coordinate units and ellipsoids, see "The Ellipsoid Vector" on page 3-4.

Projecting Coordinates Without a Map Axes

You do not need to display a map object to obtain its projected coordinates. You can perform the same projection computations that are done within the Mapping Toolbox display commands by calling the `defaultm` and `mfwdtran` functions.

Using mfwdtran with a Geographic Data Structure

Before projecting the data, you must define projection parameters, just as you would prepare a map axes with `axesm` before displaying a map. The projection parameters are stored in a map projection structure that normally resides in the `UserData` property of a MATLAB axes object, but you can directly create and use the structure for projection computations.

1 Begin by starting afresh with the `coast` data set:

```
close all; clear all;
```

```
load coast
```

2. Use `defaultm` to create an empty map projection structure for a Sinusoidal projection:

   ```
   mstruct = defaultm('sinusoid');
   ```

   The structure mstruct appears in the workspace. Use the property editor to view its fields and contents.

3. Just as you can change the property settings of a map axes with `setm`, you can assign values to the entries of the map projection structure to control the projection properties. Change the map orientation to define a transverse aspect, and set the ellipsoid and coordinate units:

   ```
   mstruct.origin = [-90 180 0];
   mstruct.geoid = almanac('earth','grs80','kilometers');
   ```

4. Repopulate the rest of the structure fields with default property values.

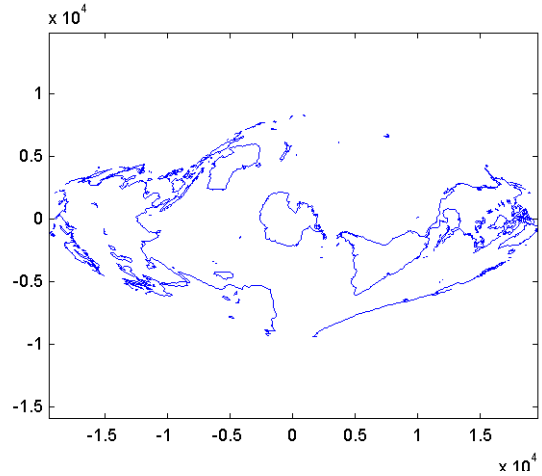
   ```
   mstruct = defaultm(sinusoid(mstruct));
   ```

   You must invoke `defaultm` a second time (recursively) to ensure that any side effects of properties you change are properly handled. For example, changing the origin can constrict the map limits on some projections.

5. Having defined the map projection parameters, project the latitude and longitude vectors into plane coordinates with the Sinusoidal projection and display the result using nonmapping MATLAB graphic commands.

   ```
   [x,y] = mfwdtran(mstruct,lat,long,[],'line');
   plot(x,y); axis equal
   ```

   The plot shows that resulting data are projected in the specified aspect.



For additional information, see the reference pages for `defaultm` and `mfwdtran`. It is also possible to reverse the process using `minvtran`, as the next section, "Inverse Map Projection" on page 8-35, describes. You may also use `projfwd` and `projinv`, which are newer Mapping Toolbox functions that use the `proj.4` map projection library to do forward and inverse projections, respectively. See the references pages for `projfwd` and `projinv` for details.

Inverse Map Projection

The process of obtaining latitudes and longitudes from geodata with planar coordinates is called *inverse projection*. Most, but not all, map projections have inverses. The Mapping Toolbox transforms plane coordinates into geodetic coordinates with the `minvtran` function, a mirror image of `mfwdtran`, which is described in "Using mfwdtran with a Geographic Data Structure" on page 8-33. Like its twin, `minvtran` operates on a geographic data structure that you can explicitly create. If the coordinate data originates from outside the Mapping Toolbox, you need to know its correct projection parameters in order for inverse projection to be successful.

Recovering Geodetic Coordinates with minvtran

In the following exercise exploring the use of `minvtran`, you again work with the `coast` data set, using the projected coordinates created in the previous exercise, "Using mfwdtran with a Geographic Data Structure" on page 8-33.

1 If you do not have the results of the previous exercise in the workspace, perform it now and go on to step 2. You have the following variables:

```
Name          Size                    Bytes  Class

  lat        9589x1                    76712  double array
  long       9589x1                    76712  double array
  mstruct       1x1                     7360  struct array
  x          9599x1                    76792  double array
  y          9599x1                    76792  double array

Grand total is 38563 elements using 314368 bytes
```

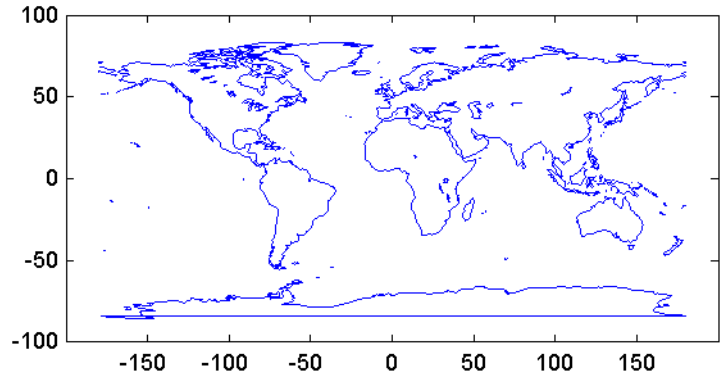
The difference in size between `lat` and `long` and `x` and `y` are due to clipping the *x-y* data to the map frame (`NaN`s are inserted at clip locations).

2 Transform the projected *x-y* data back into geographic coordinates with the inverse transformation function:

```
[lat2,long2] = minvtran(mstruct,x,y);
```

3 In a new figure, plot the resulting latitudes and longitudes as if they were plane coordinates, and set the frame larger than default:

```
figure; plot(long2,lat2); axis equal
set(gca,'XLim',[-200 200],'YLim',[-100 100])
```



Notice the wraparound in Antarctica. This occurred because its coastline crosses the International Date Line. In the projection transformation

process, longitude data outside `[-180 180]` degrees is projected back into this range because angles differing by 360° are geographically equivalent. The data from the inverse transformation process therefore jumps from 180° to -180°, as depicted by the horizontal lines in the figure above.

Obtaining Angular Directions in a Projection Space

In addition to projecting geographic positions into Cartesian coordinates, you can project angles between the sphere and the plane. For cylindrical projections in normal aspect, north maps to up on the *y*-axis, and east maps to right on the *x*-axis. This is not necessarily true of other projection types. In the normal aspect of conic projections, for example, north may skew to the left or right of vertical, depending on longitude. The `vfwdtran` function, which takes latitudes, longitudes, and azimuths, computes angles that geographic vectors make on the projection plane.

To illustrate, define vectors pointing north (0°) and east (90°) at three locations and use `vfwdtran` to compute the angles of north and east in projected coordinates on an equidistant conic projection.

Note Geographic angles are measured clockwise from north, while projected angles are measured counterclockwise from the *x*-axis.

---

1 Set up an equidistant conic projection for the northern hemisphere:

```
close all; clear all;
axesm('eqdconic','maplatlim',[-10 45],'maplonlim',[-55 55])
gridm; framem; mlabel; plabel; tightmap
```

2 Define three locations along the equator:

```
lats = [0 0 0];
lons = [-45 0 45];
```

3 Define north and east azimuths for each point:

```
northazs = [0 0 0];
eastazs = [90 90 90];
```

4 Compute the projected direction of north for each location:

```
pnorth = vfwdtran(lats,lons,northazs)
ans =
       59.614           90       120.39
```

North varies from about 60° from the *x*-axis, to vertical, to 120° from the *x*-axis, quite symmetrically.

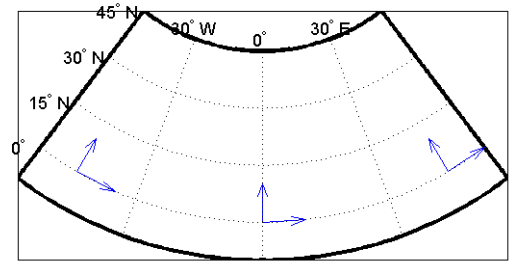
5 Compute projected direction of east for each location:

```
peast = vfwdtran(lats,lons,eastazs)
ans =
      -30.385    0.0001931       30.386
pnorth - peast
ans =
           90           90           90
```

The projected east vectors show a similar symmetry, and as expected form complementary angles to north.

6 Use `quiverm` to plot the six vectors on the projection; note their plane angles:

```
quiverm(lats, lons, [0 0 0], [10 10 10], 0)
quiverm(lats, lons, [10 10 10], [0 0 0], 0)
```



For more information, see the reference pages for `vfwdtrans` and `quiverm`.

Coordinate Transformations

In "The Orientation Vector" on page 8-10, you explored the concept of altering the aspect of a map projection in terms of pushing the North Pole to new locations. Another way to think about this is to redefine the coordinate system, and then to compute a normal aspect projection based on the new system. For example, you might redefine a spherical coordinate system so that your home town occupies the origin. If you calculated a map projection in a normal aspect with respect to this *transformed* coordinate system, the resulting display would look like an oblique aspect of the *true* coordinate system of latitudes and longitudes.

This transformation of coordinate systems can be useful independent of map displays. If you transform the coordinate system so that your home town is the new *North Pole*, then the transformed coordinates of all other points will provide interesting information.

Note The types of coordinate transformations described here are appropriate for the spherical case only. Attempts to perform them on an ellipsoid will produce incorrect answers on the order of several to tens of meters.

When you place your home town at a pole, the spherical distance of each point from your hometown becomes 90° minus its transformed latitude (also known as a *colatitude*). The point antipodal to your town would become the *South Pole*, at -90°. Its distance from your hometown is 90°-(-90°), or 180°, as expected. Points 90° distant from your hometown all have a transformed latitude of 0°, and thus make up the transformed *equator*. Transformed longitudes correspond to their respective great circle azimuths from your home town.

Reorienting Vector Data with rotatem

The `rotatem` function uses an orientation vector to transform latitudes and longitudes into a new coordinate system. The orientation vector can be produced by the `newpole` or `putpole` functions, or can be specified manually.

As an example of transforming a coordinate system, suppose you live in Midland, Texas, at (32°N,102°W). You have a brother in Tulsa (36.2°N,96°W) and a sister in New Orleans (30°N,90°W).

1. Define the three locations:

```
midl_lat = 32;   midl_lon = -102;
tuls_lat = 36.2; tuls_lon = -96;
newo_lat = 30;   newo_lon = -90;
```

2. Determine great circle distances of Tulsa and New Orleans from Midland:

```
dist2tuls = distance(midl_lat,midl_lon,tuls_lat,tuls_lon)
dist2tuls =
   6.5032

dist2newo = distance(midl_lat,midl_lon,newo_lat,newo_lon)
dist2newo =
   10.4727
```

   Tulsa is about 6.5 degrees distant, New Orleans about 10.5 degrees distant.

3. Determine the great circle azimuths from Midland:

```
az2tuls = azimuth(midl_lat,midl_lon,tuls_lat,tuls_lon)
az2tuls =
   48.1386

az2neworl = azimuth(midl_lat,midl_lon,newo_lat,newo_lon)
az2neworl =
   97.8644
```

4. Compute the absolute difference in azimuth, a fact you will use later.

```
azdif = abs(az2tuls-az2neworl)
azdif =
   49.7258
```

5. Today, you feel on top of the world, so make Midland, Texas, the *north pole* of a transformed coordinate system. To do this, first determine the origin required to put Midland at the pole using `newpole`:

```
origin = newpole(midl_lat,midl_lon)
origin =
   58    78     0
```

   The origin of the new coordinate system is (58°N, 78°E). Midland is now at a *new latitude* of 90°.

6 Determine the transformed coordinates of Tulsa and New Orleans using the `rotatem` command. Because its units default to radians, be sure to include the `degrees` keyword:

```
[tuls_lat1,tuls_lon1] = rotatem(tuls_lat,tuls_lon,...
                                origin,'forward','degrees')
tuls_lat1 =
   83.4968
tuls_lon1 =
   -48.1386

[newo_lat1,newo_lon1] = rotatem(newo_lat,newo_lon,...
                                origin,'forward','degrees')
newo_lat1 =
   79.5273
newo_lon1 =
   -97.8644
```

7 Show that the new colatitudes of Tulsa and New Orleans equal their distances from Midland computed in step 2 above:

```
tuls_colat1 = 90-tuls_lat1
tuls_colat1 =
    6.5032
newo_colat1 = 90-newo_lat1
newo_colat1 =
   10.4727
```

8 Recall from step 4 that the absolute difference in the azimuths of the two cities from Midland was 49.7258°. Verify that this equals the difference in their new longitudes:

```
tuls_lon1-newo_lon1
ans =
   49.7258
```

You might note small numerical differences in the results (on the order of 10^{-6}), due to roundoff error and trigonometric functions.

For further information, see the reference pages for `rotatem`, `newpole`, `putpole`, `neworig`, and `org2pol`.

Reorienting Gridded Data with neworig

You can transform coordinate systems of data grids as well as vector data. When regular data grids are manipulated in this manner, distance and azimuth calculations with the map variable become row and column operations.

It is easy to transform a regular data grid to create a new one with its data rearranged to correspond to a new coordinate system using the `neworig` function. To demonstrate this, do the following:

1. Load the `topo` data set and transform it to a new coordinate system in which a point in Sri Lanka (7°N, 80°E) is the *north pole*:

```
close all; clear all;
load topo
origin = newpole(7,80)
origin =
   83.0000 -100.0000         0
```

2. Reorient the data grid with `neworig`, using this orientation vector:

```
[map,lat,lon] = neworig(topo,topolegend,origin);
```

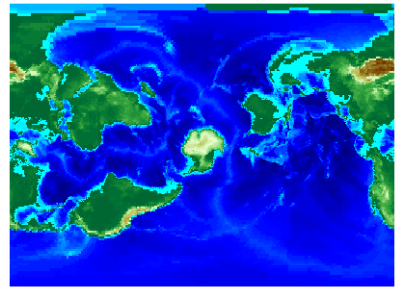
   Note that the result, `[map,lat,lon]`, is a *geolocated data grid*, not a regular data grid like the original `topo` data.

3. Display the new map:

```
axesm miller
surfm(map,[30 30]); demcmap(topo)
```

4. This map is displayed in normal aspect, as its orientation vector shows:

```
mapprops  = get(gca,'UserData');
mapprops.origin
ans =
     0     0     0
```



An interesting feature of this new grid is that every cell in its first row is 0°-1° distant from the point (7°N,80°E), and every cell in its second row is 1°-2° distant, etc. Another feature is that every cell in a particular column has the same great circle azimuth from the point.

Working with the UTM System

So far, this chapter has described types and parameters of specific projections, treating each in isolation. The following section discusses how the Transverse Mercator and Polar Stereographic projections are used to organize a worldwide coordinate grid. This system of projections is generally called Universal Transverse Mercator (UTM). This system supports many military, scientific, and surveying applications.

The UTM system divides the world into a regular non-overlapping grid of quadrangles, called *zones*, each 8 by 6 degrees in extent. Each zone uses formulas for a transverse version of the Mercator projection with projection and ellipsoid parameters designed to limit distortion. The Transverse Mercator projection is defined between 80 degrees south and 84 degrees north. Beyond these limits, the Universal Polar Stereographic (UPS) projection applies.

The UPS has two zones only, `north` and `south`, which also have special projection and ellipsoid parameters.

In addition to the zone identifier — a grid reference in the form of a number followed by a letter (e.g., `31T`) — each UTM zone has a *false northing* and a *false easting*. These are offsets (in meters) that enable each zone to have positive coordinates in both directions. For UTM, they are constant, as follows:

- False easting (for every zone): 500,000 m
- False northing (all zones in the Northern Hemisphere): 0 m
- False northing (all zones in the Southern Hemisphere): 1,000,000 m

For UPS (in both the `north` and `south` zones), the false northing and false easting are both 2,000,000.

Understanding UTM Parameters

You can create UTM maps with `axesm`, just like any other projection. However, you will note that unlike other projections, the map frame is limited to an 8-by-6 degree map window (the UTM zone), as the following steps illustrate.

1. First create a UTM map axes:

   ```
   axesm utm
   ```

2. Get the map axes properties and inspect them in the Command Window or with the Array Editor. The first few illustrate the projection defaults:

```
h = getm(gca)

mapprojection: 'utm'
              zone: '31N'
        angleunits: 'degrees'
            aspect: 'normal'
     falsenorthing: 0
      falseeasting: 500000
       fixedorient: []
             geoid: [6.3782e+006 0.082483]
       maplatlimit: [0 8]
       maplonlimit: [0 6]
      mapparallels: []
        nparallels: 0
            origin: [0 3 0]
       scalefactor: 0.9996
           trimlat: [-80 84]
           trimlon: [-180 180]
             frame: 'off'
             ffill: 100
       fedgecolor: [0 0 0]
       ffacecolor: 'none'
        flatlimit: [0 8]
       flinewidth: 2
        flonlimit: [-3 3]
                ...
```

Note that the default `zone` is `31N`. This is selected because the map `origin` defaults to `[0 3 0]`, which is on the equator and at a longitude of 3° E. This is the center longitude of zone `31N`, which has a latitude limit of `[0 8]`, and a longitude limit of `[0 6]`.

3 Move the `zone` one to the east, and inspect the other parameters again:

```
setm(gca,'zone','32n')
h = getm(gca)

mapprojection: 'utm'
              zone: '32N'
        angleunits: 'degrees'
            aspect: 'normal'
```

```
falsenorthing: 0
 falseeasting: 500000
  fixedorient: []
        geoid: [6.3782e+006 0.082483]
  maplatlimit: [0 8]
  maplonlimit: [6 12]
 mapparallels: []
   nparallels: 0
       origin: [0 9 0]
  scalefactor: 0.9996
      trimlat: [-80 84]
      trimlon: [-180 180]
        frame: 'off'
        ffill: 100
   fedgecolor: [0 0 0]
   ffacecolor: 'none'
    flatlimit: [0 8]
   flinewidth: 2
    flonlimit: [-3 3]
              ...
```

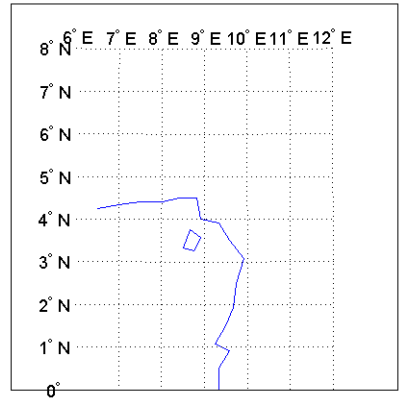
Note that the map origin and limits are adjusted for zone `32N`.

4 Draw the map grid and label it:

```
setm(gca,'grid','on','meridianlabel','on','parallellabel','on')
```

5 Load and plot the `coast` data set to see a close-up of the Gulf of Guinea and Bioko Island in UTM:

```
load coast
plotm(lat,long)
```



Setting UTM Parameters with a GUI

The easiest way to use the UTM projection is through graphical user interfaces. You can create or modify a UTM area of interest with the `axesmui` projection control panel, and get further assistance form the `utmzoneui` control panel.

1 You can Shift+click on a map axes window, or type `axesmui` to display the projection control panel. Here you start from scratch:

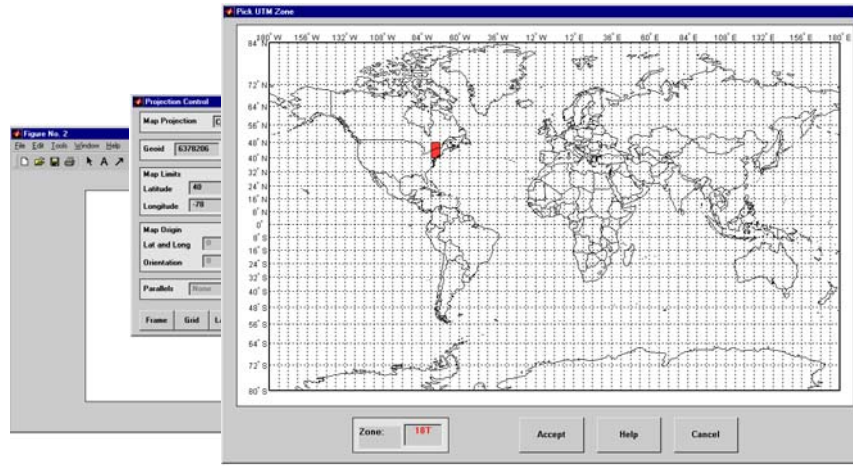
```
close all; clear all;
axesm utm
axesmui
```

The **Map Projection** field is set to `cyln: Universal Transverse Mercator (UTM)`.

Note For UTM and UPS maps, the **Aspect** field is set to `normal` and cannot be changed. If you attempt to specify `transverse`, an error results.

---

2 Click on the **Zone** button to open the `utmzoneui` panel. Click on the map near your area of interest to pick the zone:

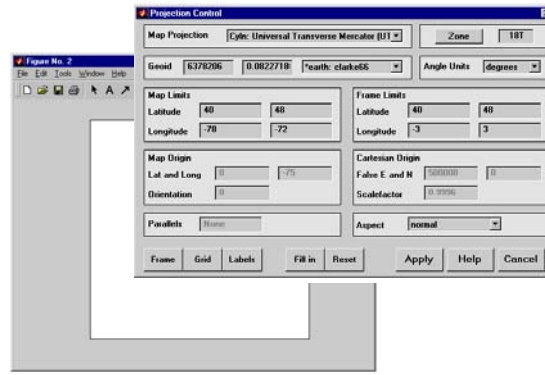


Note that while you can open the `utmzoneui` control panel from the command line, you then have to manually update the figure with the zone name it returns with a `setm` command:

```
setm(gca,'zone',ans)
```

3 Click the **Accept** button.

The `utmzoneui` panel closes, and the `zone` field is set to the one you picked. The map limits are updated accordingly, and the geoid parameters are automatically set to an appropriate ellipsoid definition for that zone. You can override the default choice by selecting another ellipsoid from the list or by typing the parameters in the **Geoid** field.

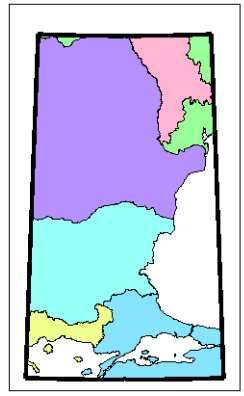


4 Click **Apply** to close the projection control panel.

The projection is then ready for projection calculations or map display commands.

5 Now view basemap data from the `worldhi` high-resolution atlas data for the area within the zone that you just selected:

```
displaym(worldhi(getm(gca,'maplatlim'),getm(gca,'maplonlim')))
polcmap
framem
```



What you see depends on the zone you selected. The preceding display is for zone 31T, which contains the Sea of Marmara and part of the Black Sea.

You can also calculate projected UTM grid coordinates from latitudes and longitudes.

```
[x,y] = mfwdtran(40.5,-73.5)

x =

     627106.47


y =

    4484124.43
```

Working in UTM Without a Map Axes

You can set up UTM to calculate coordinates without generating a map display, using the `defaultm` function. The `utmzone` and `utmgeoid` functions help you select a zone and an appropriate ellipsoid. In the following exercise,you generate UTM coordinate data for a location in New York City, using that point to define the projection itself.

1 Define a location in New York City:

```
close all; clear all;
p1 = [40.7, -74.0]
```

2 Obtain the UTM zone for this point:

```
z1 = utmzone(p1)
z1 =
18T
```

3 Obtain the suggested ellipsoid vector and name for this zone:

```
[ellipsoid,estr] = utmgeoid(z1)
ellipsoid =
  6.3782e+006     0.082272
estr =
clarke66
```

4 Set up the UTM projection based on this information:

```
utmstruct = defaultm('utm');
```

```
utmstruct.zone = '18T';
utmstruct.geoid = ellipsoid;
utmstruct = defaultm(utm(utmstruct));
```

5 Now you can calculate the grid coordinates, without a map display:

```
[x,y] = mfwdtran(utmstruct,p1(1),p1(2))
x =
  5.8448e+005
y =
  4.5057e+006
```

More on utmzone. You can also use the `utmzone` function to compute the zone limits for a given zone name. For example, using the preceding data, the latitude and longitude limits for zone 18T are

```
utmzone('18T')
ans =
    40    48   -78   -72
```

Therefore, you can call `utmzone` recursively to obtain the limits of the UTM zone within which a point location falls:

```
[zonelats zonelons] = utmzone(utmzone(40.7, -74.0))
zonelats =
    40    48
zonelons =
   -78   -72
```

For further information, see the reference pages for `utmzone`, `utmgeoid`, and `defaultm`.

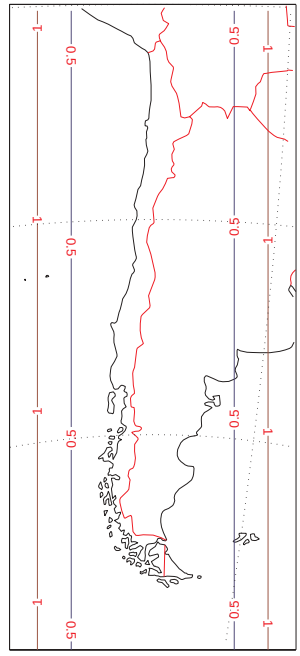
Mapping Across UTM Zones

Because UTM is a zone-based coordinate system, it is designed to be used like a map series, selecting from the appropriate sheet. While it is possible to extend one zone's coordinates into a neighboring zone's territory, this is not normally done.

To display areas that extend across more than one UTM zone, it might be appropriate to use the Mercator projection in a transverse aspect. Of course, you do not obtain coordinates in meters that would match those of a UTM projection, but the results will be nearly as accurate. Here is an example of a

transverse Mercator projection appropriate to Chile. Note how the projection's line of zero distortion is aligned with the predominantly north-south axis of the country. The zero distortion line could be put exactly on the midline of the country by a better choice of the orientation vector's central meridian and orientation angle.

```
latlim = [-60 -15];centralMeridian = -70; width = 20;
axesm('mercator',...
    'Origin',[0 centralMeridian -90],...
    'Flatlimit',[-width/2 width/2],...
    'Flonlimit',sort(-latlim),...
    'Aspect','transverse')
displaym(worldlo('POline'));framem
gridm; setm(gca,'plinefill',1000)
tightmap
mdistort scale
```



Summary and Guide to Projections

Cartographers often choose map projections by determining the types of distortion they want to minimize or eliminate. They can also determine which of the three projection types (cylindrical, conic, or azimuthal) best suits their purpose and region of interest. They can attach special importance to certain projection properties such as equal areas, straight rhumb lines or great circles, true direction, conformality, etc., further constricting the choice of a projection.

The Mapping Toolbox provides about 60 different map projections. To list them all, type `maps`. The following table also summarizes them and identifies their properties. Notes for Special Features are lovated at the end of the table. Detailed information on all map projections provided by the Mapping Toolbox can be found in the “Projections Reference” chapter

Projection	Syntax	Type	Equal-Area	Conformal	Equidistant	Special Features
Balthasart	`balthsrt`	Cylindrical	•			
Behrmann	`behrmann`	Cylindrical	•			
Bolshoi Sovietskii Atlas Mira	`bsam`	Cylindrical				
Braun Perspective	`braun`	Cylindrical				
Cassini	`cassini`	Cylindrical			•	
Central	`ccylin`	Cylindrical				
Equal-Area Cylindrical	`eqacylin`	Cylindrical	•			
Equidistant Cylindrical	`eqdcylin`	Cylindrical			•	
Gall Isographic	`giso`	Cylindrical			•	
Gall Orthographic	`gortho`	Cylindrical	•			

Projection	Syntax	Type	Equal-Area	Conformal	Equidistant	Special Features
Gall Stereographic	`gstereo`	Cylindrical				
Lambert Equal-Area Cylindrical	`lambcyln`	Cylindrical	•			
Mercator	`mercator`	Cylindrical		•		1
Miller	`miller`	Cylindrical				
Plate Carree	`pcarree`	Cylindrical			•	
Trystan Edwards	`trystan`	Cylindrical	•			
Universal Transverse Mercator (UTM)	`utm`	Cylindrical		•		
Wetch	`wetch`	Cylindrical				
Apianus II	`apianus`	Pseudocylindrical				
Collignon	`collig`	Pseudocylindrical	•			
Craster Parabolic	`craster`	Pseudocylindrical	•			
Eckert I	`eckert1`	Pseudocylindrical				
Eckert II	`eckert2`	Pseudocylindrical	•			
Eckert III	`eckert3`	Pseudocylindrical				
Eckert IV	`eckert4`	Pseudocylindrical	•			
Eckert V	`eckert5`	Pseudocylindrical				
Eckert VI	`eckert6`	Pseudocylindrical	•			
Fournier	`fournier`	Pseudocylindrical	•			

Projection	Syntax	Type	Equal-Area	Conformal	Equidistant	Special Features
Goode Homolosine	`goode`	Pseudocylindrical	•			
Hatano Asymmetrical Equal-Area	`hatano`	Pseudocylindrical	•			
Kavraisky V	`kavrsky5`	Pseudocylindrical	•			
Kavraisky VI	`kavrsky6`	Pseudocylindrical	•			
Loximuthal	`loximuth`	Pseudocylindrical				2
McBryde-Thomas Flat-Polar Parabolic	`flatplrp`	Pseudocylindrical	•			
McBryde-Thomas Flat-Polar Quartic	`flatplrq`	Pseudocylindrical	•			
McBryde-Thomas Flat-Polar Sinusoidal	`flatplrs`	Pseudocylindrical	•			
Mollweide	`mollweid`	Pseudocylindrical	•			
Putnins P5	`putnins5`	Pseudocylindrical				
Quartic Authalic	`quartic`	Pseudocylindrical	•			
Robinson	`robinson`	Pseudocylindrical				
Sinusoidal	`sinusoid`	Pseudocylindrical	•			
Tissot Modified Sinusoidal	`modsine`	Pseudocylindrical	•			
Wagner IV	`wagner4`	Pseudocylindrical	•			
Winkel I	`winkel`	Pseudocylindrical				
Albers Equal-Area Conic	`eqaconic`	Conic	•			
Equidistant Conic	`eqdconic`	Conic			•	

Projection	Syntax	Type	Equal-Area	Conformal	Equidistant	Special Features
Lambert Conformal Conic	`lambert`	Conic		•		
Murdoch I Conic	`murdoch1`	Conic			•	3
Murdoch III Minimum Error Conic	`murdoch3`	Conic			•	3
Bonne	`bonne`	Pseudoconic	•			
Werner	`werner`	Pseudoconic	•			
Polyconic	`polycon`	Polyconic				
Van Der Grinten I	`vgrint1`	Polyconic				
Breusing Harmonic Mean	`breusing`	Azimuthal				
Equidistant Azimuthal	`eqdazim`	Azimuthal			•	
Gnomonic	`gnomonic`	Azimuthal				4
Lambert Azimuthal Equal-Area	`eqaazim`	Azimuthal	•			
Orthographic	`ortho`	Azimuthal				
Stereographic	`stereo`	Azimuthal		•		5
Universal Polar Stereographic (UPS)	`ups`	Azimuthal		•		5
Vertical Perspective Azimuthal	`vperspec`	Azimuthal				
Wiechel	`wiechel`	Pseudoazimuthal	•			
Aitoff	`aitoff`	Modified Azimuthal				
Briesemeister	`bries`	Modified Azimuthal	•			

Projection	Syntax	Type	Equal-Area	Conformal	Equidistant	Special Features
Hammer	`hammer`	Modified Azimuthal	•			
Globe	`globe`	Spherical	•	•	•	6

1 Straight rhumb lines.

2 Rhumb lines from central point are straight, true to scale, and correct in azimuth.

3 Correct total area.

4 Straight line great circles.

5 Great and small circles appear as circles or lines.

6 Three-dimensional display (not a map projection).

9

Mapping Applications

This chapter describes several types of numerical applications for geospatial data, including computing and spatial statistics, and calculating tracks, routes, and other information useful for solving navigation problems.

Geographic Statistics

The Mapping Toolbox provides functions for computing basic geographical measures for spatial analysis and for filtering and conditioning data, described in the following sections:

"Geographic Means" on page 9-2	Mean location on a sphere or spheroid
"Geographic Standard Deviation" on page 9-4	Dispersion around a geographic location
"Equal-Areas in Geographic Statistics" on page 9-6	Equalizing areas for histograms and point pattern analysis
"Geographically Filtering Datasets" on page 9-9	Selecting geographic points on the basis of values in a corresponding grid

Classical statistical formulas typically assume that data is one-dimensional (and, often, normally distributed). As this is not true for geospatial data, spatial analysts have developed statistical measures that extend conventional statistics to higher dimensions. However, such formulas often assume that data occupies a two-dimensional Cartesian coordinate system. Computing statistics for geospatial data with geographic coordinates as if it were in a Cartesian framework can give statistically inappropriate results. While this assumption can sometimes yield reasonable numerical approximations within small geographic regions, for larger areas it can lead to incorrect conclusions because of distance measures and area assumptions that are inappropriate for spheres and spheroids. The Mapping Toolbox provides functions for appropriately computing statistics for geospatial data, avoiding these potential pitfalls.

Geographic Means

Consider the problem of calculating the mean position of a collection of geographic points. Taking the arithmetical mean of the latitudes and longitudes using the standard MATLAB `mean` function may seem reasonable, but doing this could yield misleading results.

Take two points at the same latitude, 180° apart in longitude, for example (30°N,90°W) and (30°N,90°E). The *mean* latitude is (30+30)/2=30, which seems right. Similarly, the mean longitude must be (90+(-90))/2=0. However, as one can also express 90°W as 270°E, (90+270)/2=180 is also a valid mean

longitude. Thus there are two correct answers, the prime meridian and the dateline. This demonstrates how the sphericity of the Earth introduces subtleties into spatial statistics.

This problem is further complicated when some points are at different latitudes. Because a degree of longitude at the Arctic Circle covers a much smaller distance than a degree at the equator, distance between points having a given difference in longitude varies by latitude.

Is in fact 30°N the right mean latitude in the first example? The mean position of two points should be equidistant from those two points, and should also minimize the total distance. Does (30°N,0°) satisfy these criteria?

```
dist1 = distance(30,90,30,0)
dist1 =
   75.5225
dist2 = distance(30,-90,30,0)
dist2 =
   75.5225
```

Consider a third point, (`lat`,`lon`), that is also equidistant from the above two points, but at a lesser distance:

```
dist1 = distance(30,90,lat,lon)
dist1 =
   60.0000
dist2 = distance(30,-90,lat,lon)
dist2 =
   60.0000
```

What is this mystery point? The `lat` is 90°N, and any `lon` will do. The North Pole is the true geographic mean of these two points. Note that the great circle containing both points runs through the North Pole (a great circle represents the shortest path between two points on a sphere).

The Mapping Toolbox function `meanm` determines the geographic mean of any number of points. It does this using three-dimensional vector addition of all the points. For example, try the following:

```
lats = [30 30];
longs = [-90 90];
[latbar,longbar] = meanm(lats,longs)
latbar =
```

```
   90
longbar =
   0
```

This is the answer you now expect. This geographic mean can result in one oddity; if the vectors all cancel each other, the mean is the center of the planet. In this case, the returned mean point is (`NaN,NaN`) and a warning is displayed. This phenomenon is highly improbable in *real* data, but can be easily constructed. For example, it occurs when all the points are equally spaced along a great circle. Try taking the geographic mean of (0°,0°), (0°,120°), and (0°,240°), which trisect the equator.

```
elats = [0 0 0];
elons = [60 120 240];
meanm(elats, elons)
ans =
         0  120.0000
```

Geographic Standard Deviation

As you might now expect, the Cartesian definition of standard deviation provided in the standard MATLAB function `std` is also inappropriate for geographic data that is unprojected or covers a significant portion of a planet. Depending upon your purpose, you might want to use the separate geographic deviations for latitude and longitude provided by the function `stdm`, or the single standard distance provided in `stdist`. Both methods measure the deviation of points from the mean position calculated by `meanm`.

The Meaning of stdm

The `stdm` function handles the latitude and longitude deviations separately.

```
[latstd,lonstd] = stdm(lat,lon)
```

The function returns two deviations, one for latitudes and one for longitudes.

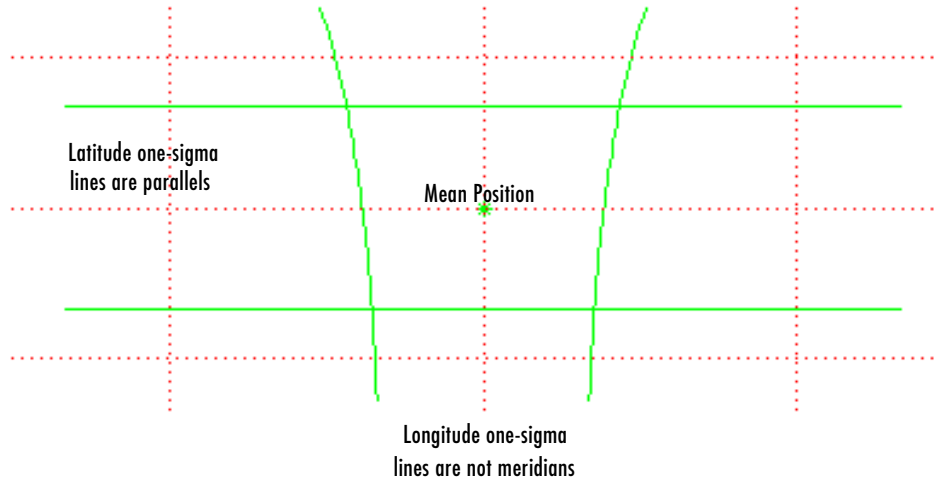
Latitude deviation is a straightforward standard deviation calculation from the mean latitude (mean parallel) returned by `meanm`. This is a reasonable measure for most cases, since on a sphere at least, a degree of latitude always has the same arc length.

Longitude deviation is another matter. Simple calculations based on sum-of-squares angular deviation from the mean longitude (mean meridian)

are misleading. The arc length represented by a degree of longitude at extreme latitudes is significantly smaller than that at low latitudes.

The term *departure* is used to represent the arc length distance along a parallel of a point from a given meridian. For example, assuming a spherical planet, the departure of a degree of longitude at the Equator is a degree of arc length, but the departure of a degree of longitude at a latitude of 60° is one-half a degree of arc length. The `stdm` function calculates a sum-of-squares departure deviation from the mean meridian.

If you want to plot the one-sigma lines for `stdm`, the latitude sigma lines are parallels. However, the longitude sigma lines are not meridians; they are lines of constant departure from the mean parallel.



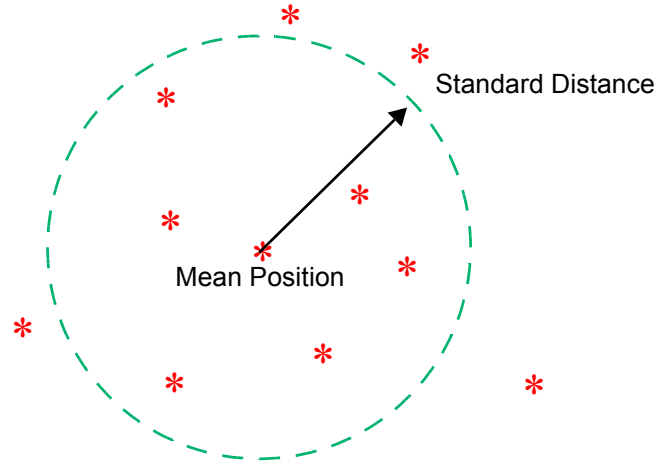
This handling of deviation has its problems. For example, its dependence upon the logic of the coordinate system can cause it to break down near the poles. For this reason, the standard distance provided by `stdist` is often a better measure of deviation. The `stdm` handling is useful for many applications, especially when the data is not global. For instance, these potential difficulties would not be a danger for data points confined to the country of Mexico.

The Meaning of stdist

The standard distance of geographic data is a measure of the dispersion of the data in terms of its distance from the geographic mean. Among its advantages are its applicability anywhere on the globe and its single value:

```
dist = stdist(lat,lon)
```

In short, the standard distance is the average, norm, or *cubic norm* of the distances of the data points in a great circle sense from the mean position. It is probably a superior measure to the two deviations returned by `stdm` except when a particularly latitude- or longitude-dependent feature is under examination.



Equal-Areas in Geographic Statistics

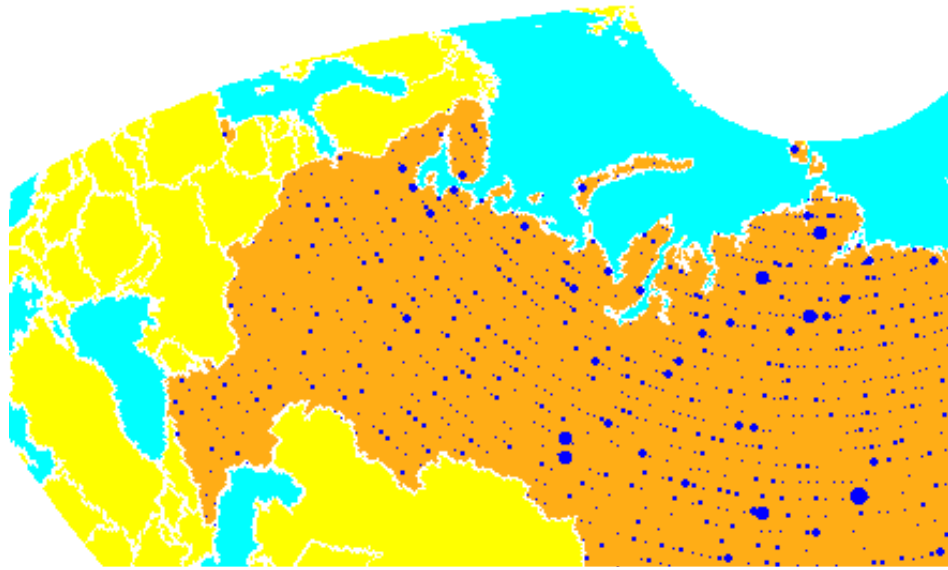
A common error in applying two-dimensional statistics to geographic data lies in ignoring equal-area treatment. It is often necessary to *bin-up* data to statistically analyze it. In a Cartesian plane, this is easily done by dividing the space into equal *x-y* squares. The geographic equivalent of this is to bin up the data in equal latitude-longitude *squares*. Since such squares at high latitudes cover smaller areas than their low-latitude counterparts, the observations in these regions are underemphasized. The result can be conclusions that are biased toward the equator.

Geographic Histograms

The geographic histogram function `histr` allows you to display *binned-up* geographic observations. The `histr` function results in equirectangular binning. Each bin has the same angular measurement in both latitude and longitude, with a default measurement of 1 degree. The center latitudes and longitudes of the bins are returned, as well as the number of observations per bin:

```
[binlat,binlon,num] = histr(lats,lons)
```

As previously noted, these equirectangular bins result in counting bias toward the equator. Here is a display of the one-degree-by-one-degree binning of approximately 5,000 random data points in Russia. The relative size of the circles indicates the number of observations per bin:

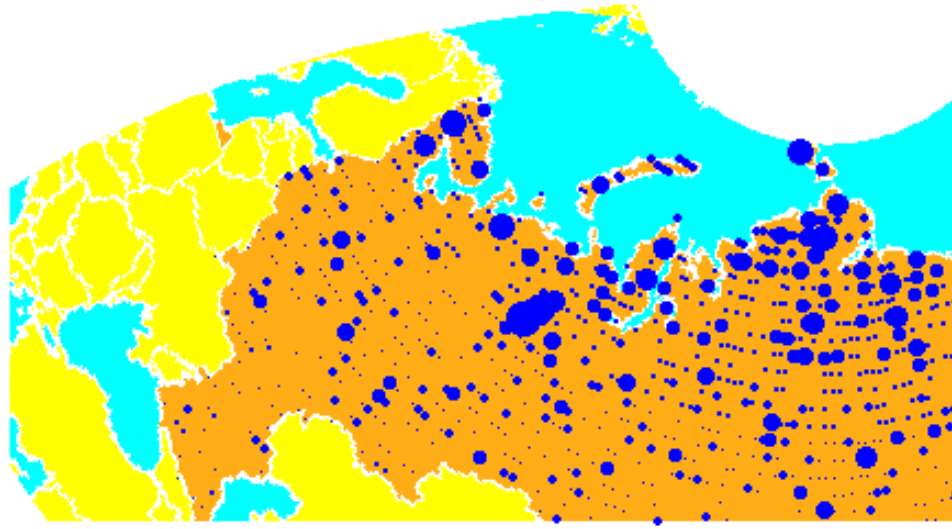


This is a portion of the whole map, displayed in an equal-area Bonne projection. The first step in creating data displays without area bias is to choose an equal-area projection. The proportionally sized symbols are a result of the specialized display function `scatterm`.

You can eliminate the area bias by adding a fourth output argument to `histr`, that will be used to weight each bin's observation by that bin's area:

```
[binlat,binlon,num,wnum] = histr(lats,lons)
```

The fourth output is the weighted observation count. Each bin's observation count is divided by its normalized area. Therefore, a high-latitude bin will have a larger weighted number than a low-latitude bin with the same number of actual observations. The same data and bins look much different when they are area-weighted:



Notice that there are larger symbols to the north in this display. The previous display suggested that the data was relatively uniformly distributed. When equal-area considerations are included, it is clear that the data is skewed to the north. In fact, the data used is northerly skewed, but a simple equirectangular handling failed to demonstrate this.

The `histr` function, therefore, does provide for the display of area-weighted data. However, the actual bins used are of varying areas. Remember, the one-degree-by-one-degree bin near a pole is much smaller than its counterpart near the equator.

The `hista` function provides for actual equal-area bins.

Converting to an Equal-Area Coordinate System

The actual data itself can be converted to an equal-area coordinate system for analysis with other statistical functions. It is easy to convert a collection of geographic latitude-longitude points to an equal-area *x-y* Cartesian coordinate system. The `grn2eqa` function applies the same transformation used in calculating the Equal-Area Cylindrical projection:

```
[x,y] = grn2eqa(lat,lon)
```

For each Greenwich `lat - long` pair, an equal-area `x - y` is returned. The variables `x` and `y` can then be operated on under the equal-area assumption, using a variety of two-dimensional statistical techniques. Tools for such analysis can be found in the Statistics Toolbox and elsewhere. The results can then be converted back to Greenwich coordinates using the `eqa2grn` function:

```
[lat,lon] = eqa2grn(x,y)
```

Remember, when converting back and forth between systems, latitude corresponds to *y* and longitude corresponds to *x*.

Geographically Filtering Datasets

Often, a set of data containsunwanted data mixed in with the desired values. For example, your data might include points for the entire United States, but you only want to work with those points falling in Alabama, or perhaps the data set is *untidy* – out of 4,000 points, you notice that 3 or 4 obviously fall outside of reality (for example, one of your city points is in the middle of the ocean). It can be quite a chore to look at each data set element individually. Perhaps selecting a portion of the data is part of your analysis.

The `filterm` function works with a data grid to filter a vector data set. The form is the following:

```
[flats,flons] = filterm(lats,lons,datagrid,refvec,allowed]
```

Each location defined by `lats` and `lons` is compared to the value at that point in `datagrid`. If the value is `allowed`, that point is included in `flats` and `flons`.

The grid might be politically indexed, and the allowed values might be the code or codes corresponding to the states or countries desired (e.g., Alabama). The grid might also contain cardinal values, or a logical condition thereon,

and the allowed value might be 1 for `true`. Here's what an example might look like using the `topo` data grid:

```
[flats,flons] = filterm(lats,lons,topo>0,topolegend,1)
```

The result would be those points corresponding to land.

Navigation

One field that makes extensive use of geographic information is navigational science and practice. The Mapping Toolbox includes specialized functions for navigation, which are described in the following sections:

Navigating watercraft and aircraft involves a variety of tasks: establishing position, using known, fixed landmarks (piloting); using the stars, Sun, and Moon (celestial navigation); using technology to fix positions (inertial guidance, radio beacons, and satellite navigation, including GPS); or deducing net movement from a past known position (dead reckoning).

Another navigational task involves planning a voyage or flight, which includes determining a short route (great circle approximation), weather avoidance (optimal track routing), and setting out a plan of intended movement (track laydown). The Mapping Toolbox contains functions to support these navigational activities.

Conventions for Navigational Functions

Units

The Mapping Toolbox is, in general, very flexible in allowing a variety of angular and distance measurement units. The navigational support functions are

- `dreckon`
- `gcwaypts`
- `legs`
- `navfix`

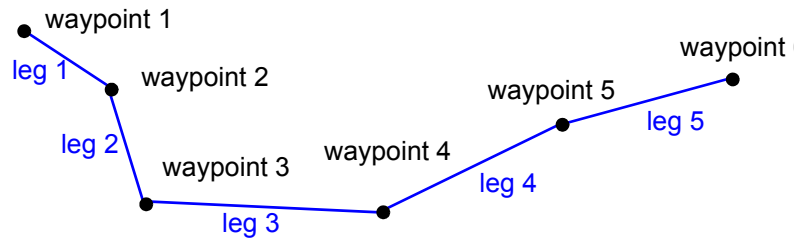
To make these functions easy to use and to conform to common navigational practice, and *for these specific functions only*, certain conventions are used:

- angles are always in degrees
- distances are always in nautical miles
- speeds are always in knots (nautical miles per hour)

Related functions that do not carry this restriction include `rhxrh`, `scxsc`, `gcxgc`, `gcxsc`, `track`, `timezone`, and `crossfix`, because of their potential for application outside navigation.

Navigational Track Format

Navigational track format requires column-vector variables for the latitudes and longitudes of track waypoints. A *waypoint* is a point through which a track passes, usually corresponding to a course (or speed) change. Navigational tracks are made up of the line segments connecting these waypoints, which are called *legs*. In this format, therefore, n legs are described using $n+1$ waypoints, because an endpoint for the final leg must be defined. In Mapping Toolbox navigation functions, angle units are always in degrees.



Here, five track legs require six waypoints. In navigational track format, the waypoints are represented by two 6-by-1 vectors, one for the latitudes and one for the longitudes.

Fixing Position

The fundamental objective of navigation is to determine at a given moment how to proceed to your destination, avoiding hazards on the way. The first step in accomplishing this is to establish your current position. Early sailors kept within sight of land to facilitate this. Today, navigation within sight (or radar range) of land is called *piloting*. Positions are fixed by correlating the bearings and/or ranges of landmarks. In real-life piloting, all sighting bearings are treated as rhumb lines, while in fact they are actually great circles.

Over the distances involved with visual sightings (up to 20 or 30 nautical miles), this assumption causes no measurable error and it provides the significant advantage of allowing the navigator to plot all bearings as straight lines on a Mercator projection.

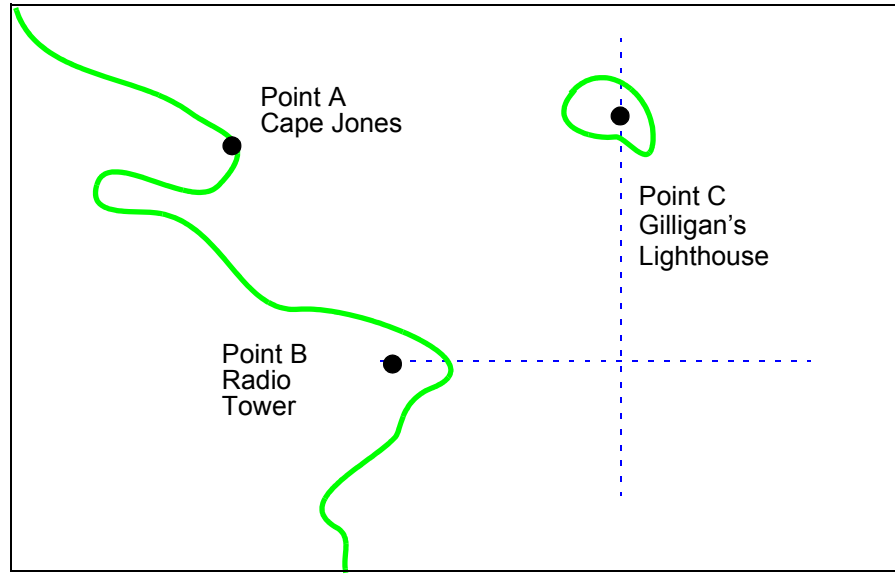
The Mercator was designed exactly for this purpose. Range circles, which might be determined with a radar, are assumed to plot as true circles on a Mercator chart. This allows the navigator to manually draw the range arc with a compass.

These assumptions also lead to computationally efficient methods for fixing positions with a computer. The Mapping Toolbox includes the `navfix` function, which mimics the manual plotting and fixing process using these assumptions.

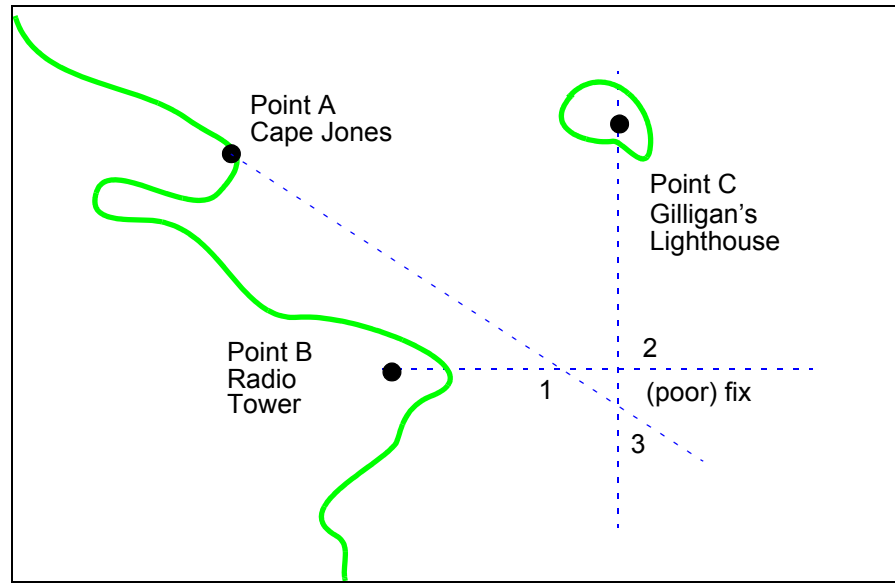
To obtain a good navigational fix, your relationship to at least three known points is considered necessary. A questionable or poor fix can be obtained with two known points.

Some Possible Situations

In this imaginary coastal region, you take a visual bearing on the radio tower of 270°. At the same time, Gilligan's Lighthouse bears 0°. If you plot a 90°-270° line through the radio tower and a 0°-180° line through the lighthouse on your Mercator chart, the point at which the lines cross is a fix. Since you have used only two lines, however, its quality is questionable.



But wait; your port lookout says he took a bearing on Cape Jones of 300°. If that line exactly crosses the point of intersection of the first two lines, you will have a perfect fix.



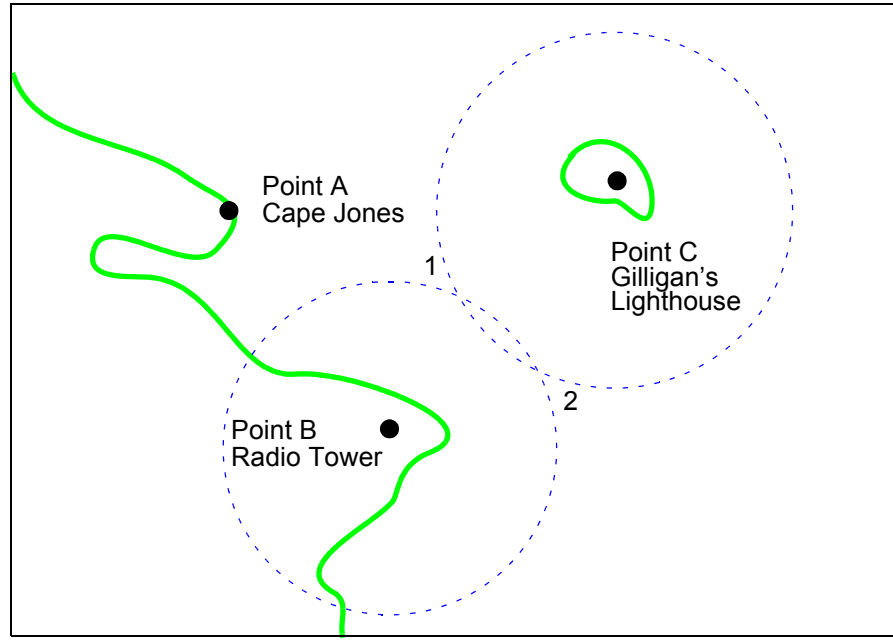
Whoops. What happened? Is your lookout in error? Possibly, but perhaps one or both of your bearings was slightly in error. This happens all the time. Which point, 1, 2, or 3, is correct? As far as you know, they are all equally valid.

In practice, the little triangle is plotted, and the fix position is taken as either the center of the triangle or the vertex closest to a danger (like shoal water). If the triangle is large, the quality is reported as *poor*, or even as *no fix*. If a fourth line of bearing is available, it can be plotted to try to resolve the ambiguity. When all three lines appear to cross at exactly the same point, the quality is reported as *excellent* or *perfect*.

Notice that three lines resulted in three intersection points. Four lines would return six intersection points. This is a case of combinatorial counting. Each intersection corresponds to choosing two lines to intersect from among n lines.

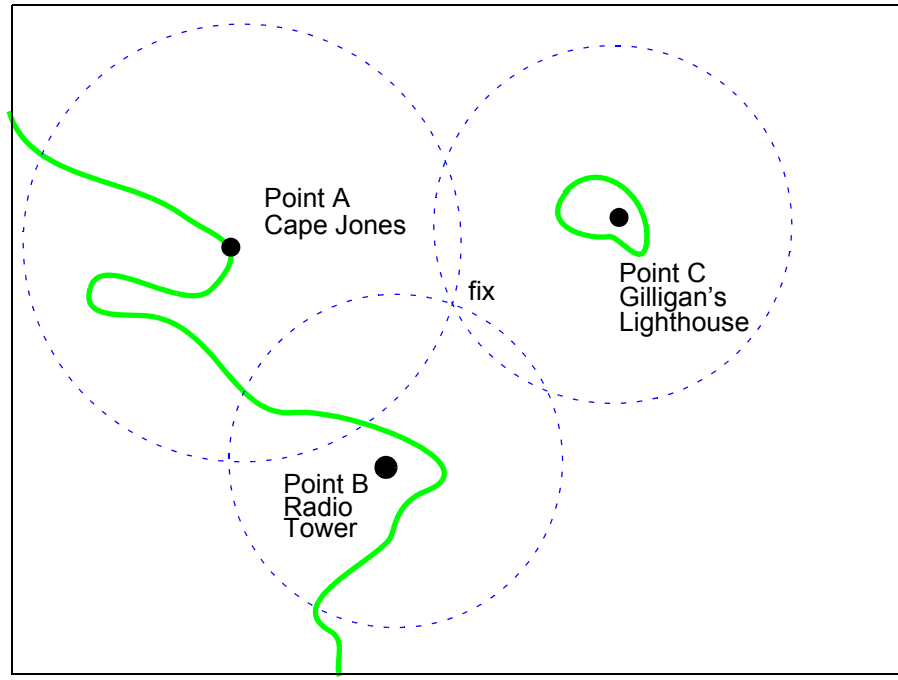
The next time you traverse these straits, it is a very foggy morning. You can't see any landmarks, but luckily, your navigational radar is operating. Each of these landmarks has a good radar signature, so you're not worried. You get a

range from the radio tower of 14 nautical miles and a range from the lighthouse of 15 nautical miles.



Now what? You took ranges from only two objects, and yet you have two possible positions. This ambiguity arises from the fact that circles can intersect twice.

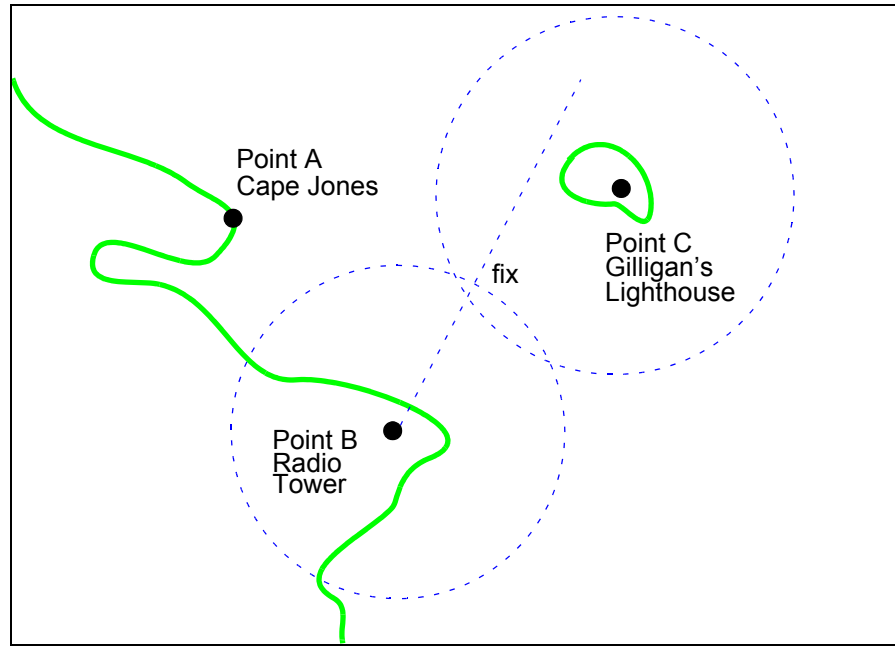
Luckily, your radar watch reports that he has Cape Jones at 18 nautical miles. This should resolve everything.



You were lucky this time. The third range resolved the ambiguity and gave you an excellent fix. Three intersections practically coincide. Sometimes the ambiguity is resolved, but the fix is still poor because the three closest intersections form a sort of circular triangle.

Sometimes the third range only adds to the confusion, either by bisecting the original two choices, or by failing to intersect one or both of the other arcs at all. In general, when *n* arcs are used, 2x(*n*-choose-2) possible intersections result. In this example, it is easy to tell which ones are *right*.

Bearing lines and arcs can be combined. If instead of reporting a third range, your radar watch had reported a bearing from the radar tower of 20°, the ambiguity could also have been resolved. Note, however, that in practice, lines of bearing for navigational fixing should only be taken visually, except in desperation. A radar's beam width can be a degree or more, leading to uncertainty.



As you begin to wonder whether this manual plotting process could be automated, your first officer shows up on the bridge with a laptop and the Mapping Toolbox.

Using navfix

The `navfix` function can be used to determine the points of intersection among any number of lines and arcs. Be warned, however, that due to the combinatorial nature of this process, the computation time grows rapidly with the number of objects. To illustrate this function, assign positions to the landmarks. Point A, Cape Jones, is at (`latA`,`lonA`). Point B, the radio tower, is at (`latB`,`lonB`). Point C, Gilligan's Lighthouse, is at (`latC`,`lonC`).

For the bearing-lines-only example, the syntax is:

```
[latfix,lonfix] = navfix([latA latB latC],[lonA lonB lonC],...
                         [300 270 0])
```

This defines the three points and their bearings as taken *from the ship*. The outputs would look something like this, with actual numbers, of course:

```
latfix =
   latfix1       NaN          % A intersecting B
   latfix2       NaN          % A intersecting C
   latfix3       NaN          % B intersecting C
lonfix =
   lonfix1       NaN          % A intersecting B
   lonfix2       NaN          % A intersecting C
   lonfix3       NaN          % B intersecting C
```

Notice that these are two-column matrices. The second column consists of `NaN`s because it is used only for the two-intersection ambiguity associated with arcs.

For the range-arcs-only example, the syntax is

```
[latfix,lonfix] = navfix([latA latB latC],[lonA lonB lonC],...
                         [16 14 15],[0 0 0])
```

This defines the three points and their ranges as taken from the ship. The final argument indicates that the three cases are all ranges.

The outputs have the following form:

```
latfix =
   latfix11  latfix12          % A intersecting B
   latfix21  latfix22          % A intersecting C
   latfix31  latfix32          % B intersecting C
lonfix =
   lonfix11  lonfix12          % A intersecting B
   lonfix21  lonfix22          % A intersecting C
   lonfix31  lonfix32          % B intersecting C
```

Here, the second column is used, because each pair of arcs has two potential intersections.

For the bearings and ranges example, the syntax requires the final input to indicate which objects are lines of bearing (indicated with a 1) and which are range arcs (indicated with a 0):

```
[latfix,lonfix] = navfix([latB latB latC],[lonB lonB lonC],...
                         [20 14 15],[1 0 0])
```

The resulting output is mixed:

```
latfix =
   latfix11       NaN          % Line B intersecting Arc B
   latfix21  latfix22          % Line B intersecting Arc C
   latfix31  latfix32          % Arc B intersecting Arc C
lonfix =
   lonfix11       NaN          % Line B intersecting Arc B
   lonfix21  lonfix22          % Line B intersecting Arc C
   lonfix31  lonfix32          % Arc B intersecting Arc C
```

Only one intersection is returned for the line from B with the arc about B, since the line originates inside the circle and intersects it once. The same line intersects the other circle twice, and hence it returns two points. The two circles taken together also return two points.

Usually, you have an idea as to where you are before you take the fix. For example, you might have a dead reckoning position for the time of the fix (see below). If you provide `navfix` with this estimated position, it chooses from each pair of ambiguous intersections the point closest to the estimate. Here's what it might look like:

```
[latfix,lonfix] = navfix([latB latB latC],[lonB lonB lonC],...
                         [20 14 15],[1 0 0],drlat,drlon)
latfix =
   latfix11                    % the only point
   latfix21                    % the closer point
   latfix31                    % the closer point
lonfix =
   lonfix11                    % the only point
   lonfix21                    % the closer point
   lonfix31                    % the closer point
```

A Numerical Example of Using navfix

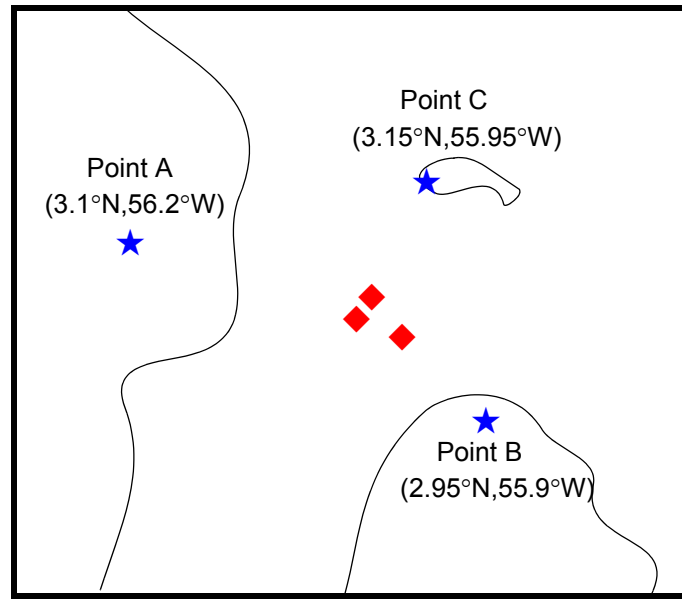
1 Define some specific points in the middle of the Atlantic Ocean. These are strictly arbitrary; perhaps they correspond to points in Atlantis:

```
lata = 3.1;  lona = -56.2;
latb = 2.95; lonb = -55.9;
latc = 3.15; lonc = -55.95;
```

2 Plot them on a Mercator projection:

```
axesm('MapProjection','mercator','Frame','on',...
    'MapLatLimit',[2.8 3.3],'MapLonLimit',[-55.8 -56.3])
plotm([lata latb latc],[lona lonb lonc],...
    'LineStyle','none','Marker','pentagram',...
    'MarkerEdgeColor','b','MarkerFaceColor','b',...
    'MarkerSize',12)
```

Here is what it looks like (the labeling and imaginary coastlines are added after the fact for illustration).



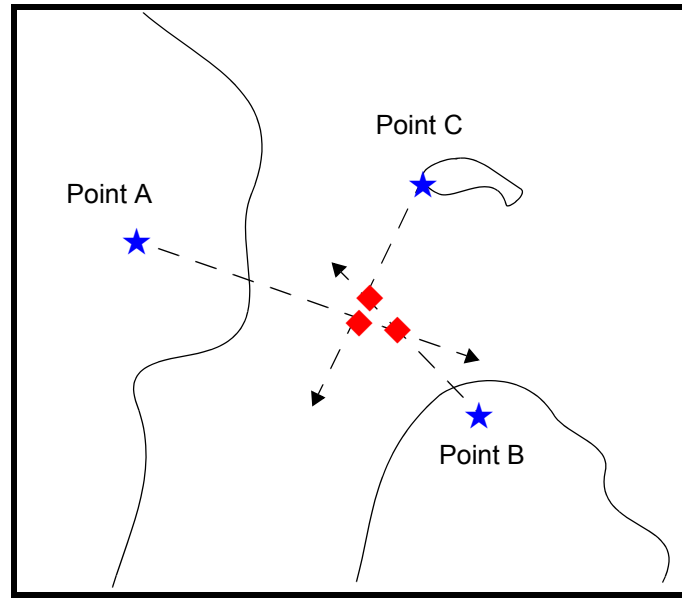
3 Take three visual bearings: Point A bears 289°, Point B bears 135°, and Point C bears 026.5°. Calculate the intersections:

```
[newlat,newlong] = navfix([lata latb latc],[lona lonb lonc],...
                          [289 135 26.5],[1 1 1])
newlat =
   3.0214       NaN
   3.0340       NaN
   3.0499       NaN
```

```
newlong =
   -55.9715       NaN
   -56.0079       NaN
   -56.0000       NaN
```

4 Add the bearing lines and intersection points to the map:

```
plotm(newlat,newlong,'LineStyle','none',...
   'Marker','diamond','MarkerEdgeColor','r',...
   'MarkerFaceColor','r','MarkerSize',9)
```



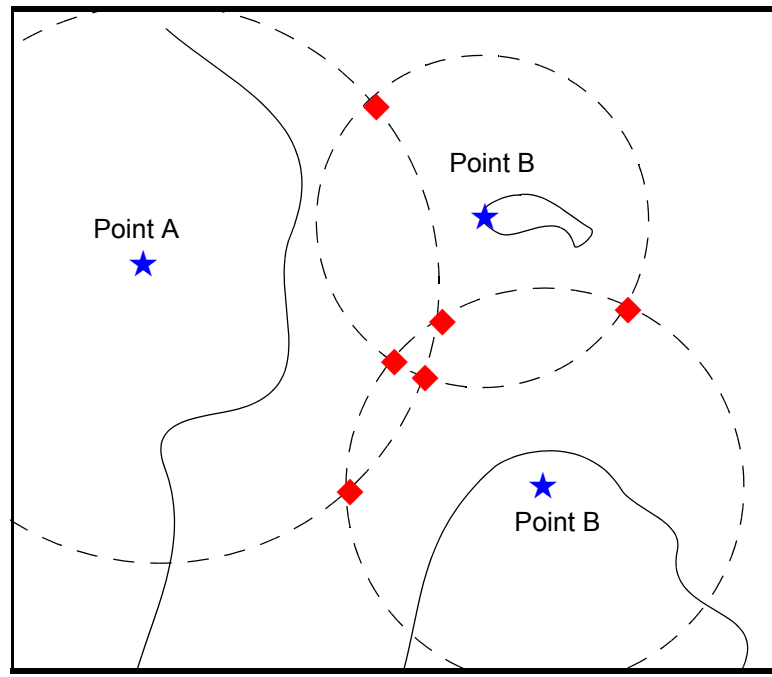
Notice that each pair of objects results in only one intersection, since all are lines of bearing.

5 What if instead, you had ranges from the three points, A, B, and C, of 13 nmi, 9 nmi, and 7.5 nmi, respectively?

```
[newlat,newlong] = navfix([lata latb latc],[lona lonb lonc],...
                          [13 9 7.5],[0 0 0])
newlat =
   3.0739    2.9434
```

```
    3.2413    3.0329
    3.0443    3.0880
newlong =
   -55.9846  -56.0501
   -56.0355  -55.9937
   -56.0168  -55.8413
```

Here's what these points look like:

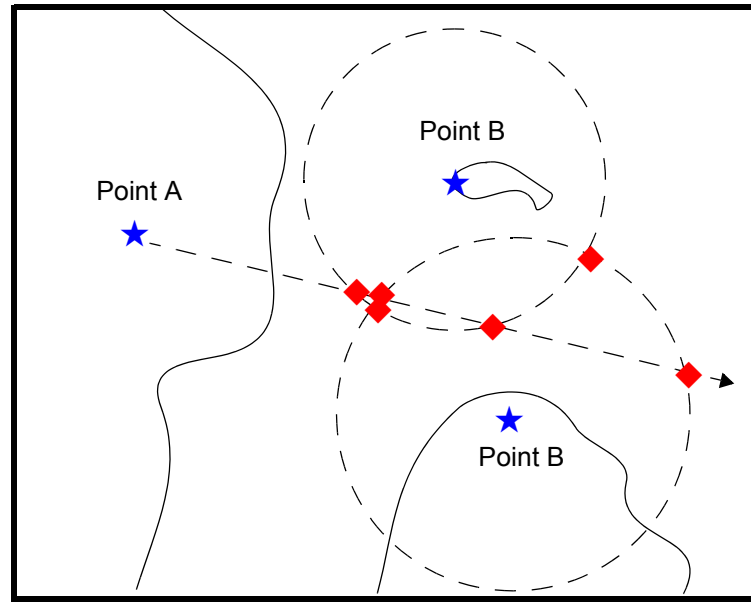


Three of these points look reasonable, three do not.

6 What if, instead of a range from Point A, you had a bearing to it of 284°?

```
[newlat,newlong] = navfix([lata latb latc],[lona lonb lonc],...
                          [284 9 7.5],[1 0 0])
newlat =
    3.0526    2.9892
    3.0592    3.0295
```

```
    3.0443    3.0880
newlong =
   -56.0096  -55.7550
   -56.0360  -55.9168
   -56.0168  -55.8413
```

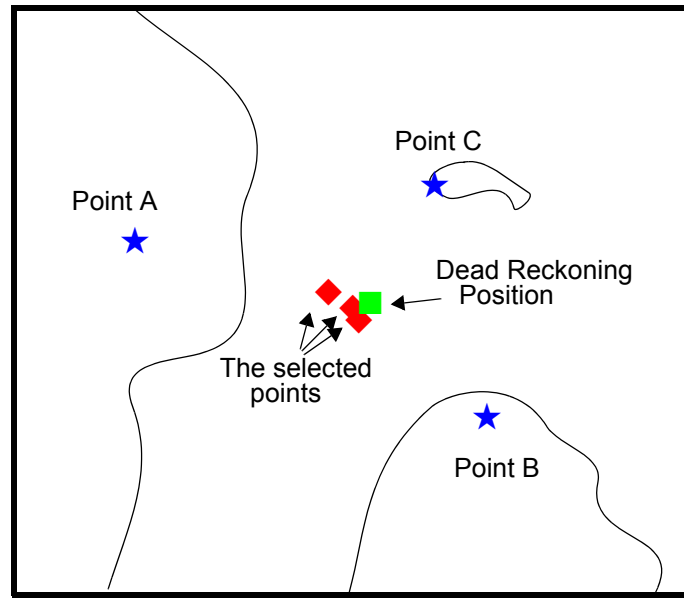


Again, visual inspection of the results indicates which three of the six possible points seem like *reasonable* positions.

7 When using the dead reckoning position (3.05°N,56.0°W), the closer, more reasonable candidate from each pair of intersecting objects is chosen:

```
drlat = 3.05; drlon = -56;
[newlat,newlong] = navfix([lata latb latc],[lona lonb lonc],...
                          [284 9 7.5],[1 0 0],drlat,drlon)
newlat =
   3.0526
   3.0592
   3.0443
newlong =
```

```
-56.0096
-56.0360
-56.0168
```



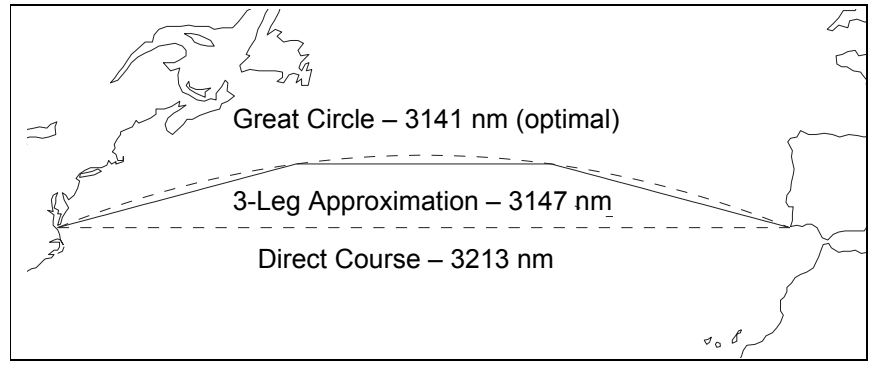
Planning

You know that the shortest path between two geographic points is a great circle. Sailors and aviators are interested in minimizing distance traveled, and hence time elapsed. You also know that the rhumb line is a path of constant heading, the *natural* means of traveling. In general, to follow a great circle path, you would have to continuously alter course. This is impractical. However, you can approximate a great circle path by rhumb line segments so that the added distance is minor and the number of course changes minimal.

Surprisingly, very few rhumb line *track legs* are required to closely approximate the distance of the great circle path.

Consider the voyage from Norfolk, Virginia (37°N,76°W), to Cape St. Vincent, Portugal (37°N,9°W), one of the most heavily trafficked routes in the Atlantic. A due-east rhumb line track is 3,213 nautical miles, while the optimal great circle distance is 3,141 nautical miles.

Although the rhumb line path is only a little more than 2% longer, this is an additional 72 miles over the course of the trip. For a 12-knot tanker, this results in a 6-hour delay, and in shipping, time is money. If just three rhumb line segments are used to approximate the great circle, the total distance of the trip is 3,147 nautical miles. Our tanker would suffer only a half-hour delay compared to a continuous rhumb line course.



The Mapping Toolbox provides the function `gcwaypts` to quickly calculate waypoints in navigation track format in order to approximate a great circle with rhumb line segments. The syntax is simple:

```
[latpts,lonpts] = gcwaypts(lat1,lon1,lat2,lon2,numlegs)
```

All the inputs for this function are scalars. The `numlegs` input is the number of equa-length legs desired, which is 10 by default. The outputs are column vectors representing waypoints in navigational track format. The size of each of these vectors is `[(numlegs+1) 1]`. Here are the points for this example:

```
[latpts,lonpts] = gcwaypts(37,-76,37,-9,3)
latpts =
   37.0000
   41.5076
   41.5076
37.0000
lonpts =
   -76.0000
   -54.1777
   -30.8223
```

```
    -9.0000
```

These points represent waypoints along the great circle between which the approximating path follows rhumb lines. Four points are needed for three legs, because the final point at Cape St. Vincent must be included.

Track Laydown – Displaying Navigational Tracks

Navigational tracks are most useful when graphically displayed. Traditionally, the navigator identifies and plots waypoints on a Mercator projection and then connects them with a straightedge, which on this projection results in rhumb line tracks. In the previous example, waypoints were chosen to approximate a great circle route, but they can be selected for a variety of other reasons.

Let's say that after arriving at Cape St. Vincent, your tanker must traverse the Straits of Gibraltar and then travel on to Port Said, the northern terminus of the Suez Canal. On the scale of the Mediterranean Sea, following great circle paths is of little concern compared to ensuring that the many straits and passages are safely transited. The navigator selects appropriate waypoints and plots them.

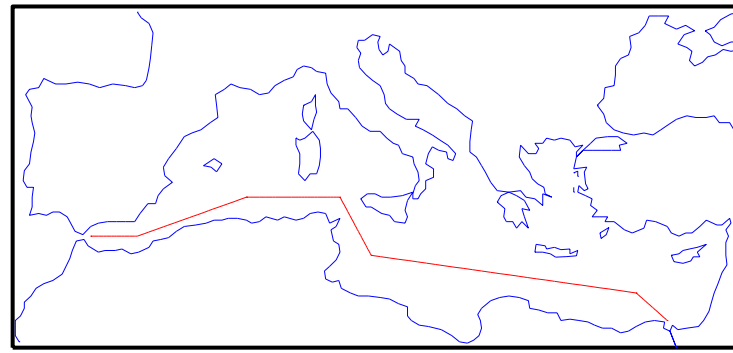
To do this with the Mapping Toolbox, you can display a map axes with a Mercator projection, select appropriate map latitude and longitude limits to isolate the area of interest, plot coastline data, and interactively mouse-select the waypoints with the `inputm` function. The `track` function will generate points to connect these waypoints, which can then be displayed with `plotm`.

For illustration, assume that the waypoints are known (or were gathered using `inputm`). To learn about using `inputm`, see "Interacting with Displayed Maps" on page 4-42, or `inputm` in the Mapping Toolbox reference pages.

```
waypoints = [36 -5; 36 -2; 38 5; 38 11; 35 13; 33 30; 31.5 32]
waypoints =
   36.0000   -5.0000
   36.0000   -2.0000
   38.0000    5.0000
   38.0000   11.0000
   35.0000   13.0000
   33.0000   30.0000
   31.5000   32.0000
load coast
```

```
axesm('MapProjection','mercator',...
'MapLatLimit',[30 47],'MapLonLimit',[-10 37])
framem
plotm(lat,long)

[lttrk,lntrk] = track(waypoints);
plotm(lttrk,lntrk,'r')
```

Although these track segments are straight lines on the Mercator projection, they are curves on others:



The segments of a track like this are called *legs*. Each of these legs can be described in terms of course and distance. The function `legs` will take the waypoints in navigational track format and return the course and distance required for each leg. Remember, the order of the points in this format determines the direction of travel. Courses are therefore calculated from each waypoint to its successor, not the reverse.

```
[courses,distances] = legs(waypoints)
courses =
   90.0000
   70.3132
   90.0000
   151.8186
   98.0776
   131.5684
distances =
   145.6231
   356.2117
```

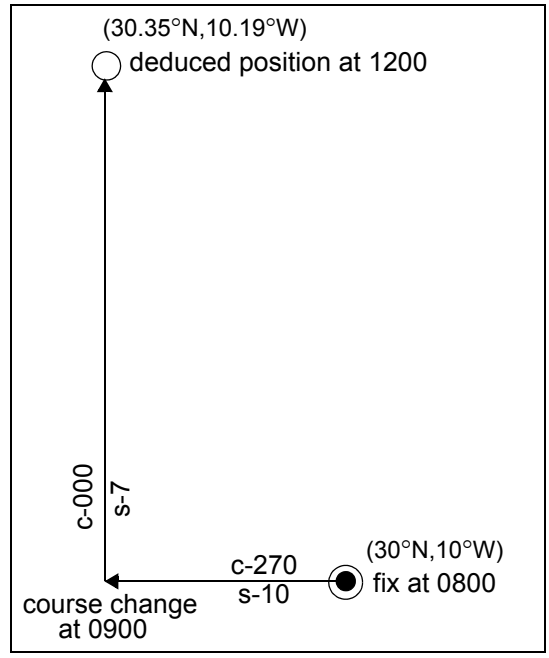
```
283.6839
204.2073
854.0092
135.6415
```

Since this is a navigation function, the courses are all in degrees and the distances are in nautical miles. From these distances, speeds required to arrive at Port Said at a given time can be calculated. Southbound traffic is allowed to enter the canal only once per day, so this information might be economically significant, since unnecessarily high speeds can lead to high fuel costs.

Dead Reckoning

When sailors first ventured out of sight of land, they faced a daunting dilemma. How could they find their way home if they didn't know where they were? The practice of *dead reckoning* is an attempt to deal with this problem. The term is derived from *deduced reckoning*.

Briefly, dead reckoning is vector addition plotted on a chart. For example, if you have a fix at (30°N,10°W) at 0800, and you proceed due west for 1 hour at 10 knots, and then you turn north and sail for 3 hours at 7 knots, you should be at (30.35°N,10.19°W) at 1200.



However, a sailor *shoots the sun* at local apparent noon and discovers that the ship's latitude is actually 30.29°N. What's worse, he lives before the invention of a reliable chronometer, and so he cannot calculate his longitude at all from this sighting. What happened?

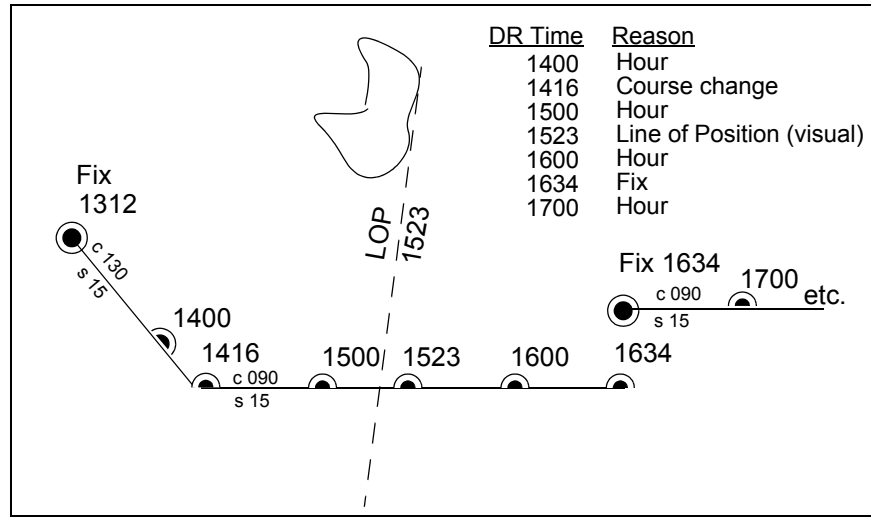
Leaving aside the difficulties in speed determination and the need to tack off course, even modern craft have to contend with winds and currents. However, despite these limitations, dead reckoning is still used for determining position between fixes and for forecasting future positions. This is because dead reckoning provides a certainty of assumptions that estimations of wind and current drift cannot.

When navigators establish a fix from some source, be it from piloting, celestial, or satellite observations, they plot a dead reckoning (DR) track, which is a plot of the intended positions of the ship forward in time. In practice, dead reckoning is usually plotted for 3 hours in advance, or for the time period covered by the next three expected fixes. In open ocean conditions, hourly fixes are sufficient; in coastal pilotage, three-minute fixes are common.

Specific DR positions, which are sometimes called *DR*s, are plotted according to the *Rules of DR*:

- DR at every course change
- DR at every speed change
- DR every hour on the hour
- DR every time a fix or running fix is obtained
- DR 3 hours ahead or for the next three expected fixes
- DR for every line of position (LOP), either visual or celestial

For example, the navigator plots these DRs:



Notice that the 1523 DR does not coincide with the LOP at 1523. Although note is taken of this variance, one line is insufficient to calculate a new fix.

The Mapping Toolbox includes the function `dreckon`, which calculates the DR positions for a given set of courses and speeds. The function provides DR positions for the first three rules of dead reckoning. The approach is to provide a set of waypoints in navigational track format corresponding to the plan of intended movement.

The time of the initial waypoint, or fix, is also needed, as well as the speeds to be employed along each leg. Alternatively, a set of speeds and the times for which each speed will apply can be provided. `dreckon` returns the positions and times required of these DRs:

- `dreckon` calculate the times for position of each course change, which will occur at the waypoints
- `dreckon` calculates the positions for each whole hour
- If times are provided for speed changes, `dreckon` calculates positions for these times if they do not occur at course changes

Imagine you have a fix at midnight at the point (10°N,0°):

```
waypoints(1,:) = [10 0]; fixtime = 0;
```

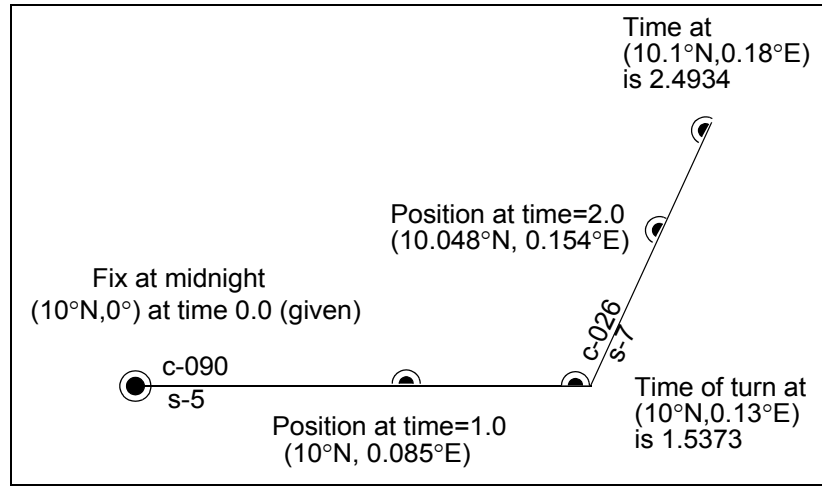
You intend to travel east and alter course at the point (10°N,0.13°E) and head for the point (10.1°N,0.18°E). On the first leg, you will travel at 5 knots, and on the second leg you will speed up to 7 knots.

```
waypoints(2,:) = [10 .13];
waypoints(3,:) = [10.1 .18];
speeds = [5;7];
```

To determine the DR points and times for this plan, use `dreckon`:

```
[drlat,drlon,drtime] = dreckon(waypoints,fixtime,speeds);
[drlat drlon drtime]
ans =
   10.0000    0.0846    1.0000     % Position at 1 am
   10.0000    0.1301    1.5373     % Time of course change
   10.0484    0.1543    2.0000     % Position at 2 am
   10.1001    0.1801    2.4934     % Time at final waypoint
```

Here is an illustration of this track and its DR points:



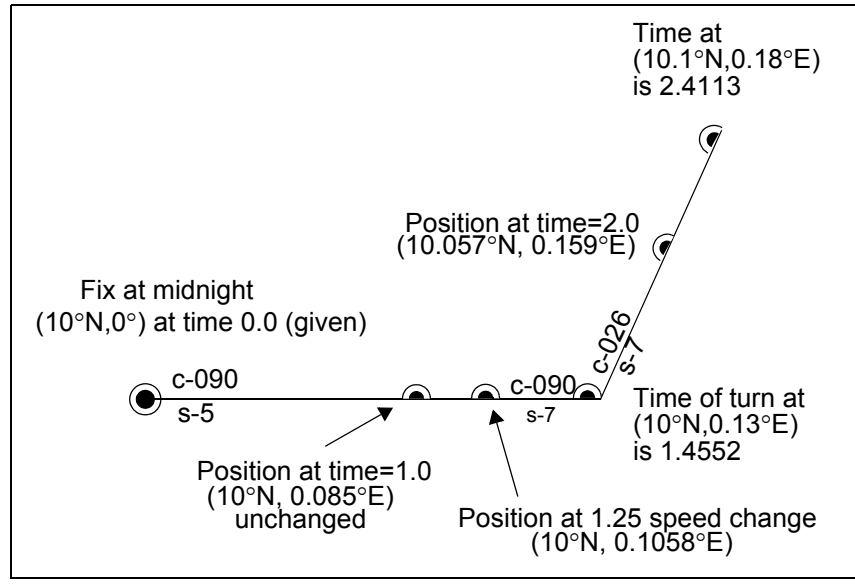
However, you would like to get to the final point a little earlier to make a rendezvous. You decide to recalculate your DRs based on speeding up to 7 knots a little earlier than planned. The first calculation tells you that you were going to increase speed at the turn, which would occur at a time 1.5373 hours after midnight, or 1:32 a.m. (at time 0132 in navigational time format). What time would you reach the rendezvous if you increased your speed to 7 knots at 1:15 a.m. (0115, or 1.25 hours after midnight)?

To indicate times for speed changes, another input is required, providing a time interval after the fix time at which each ordered speed is to end. The first speed, 5 knots, is to end 1.25 hours after midnight. Since you don't know when the rendezvous will be made under these circumstances, set the time for the second speed, 7 knots, to end at infinity. No DRs will be returned past the last waypoint.

```
spdtimes = [1.25; inf];
[drlat,drlon,drtime] = dreckon(waypoints,fixtime,...
                               speeds,spdtimes);
[drlat,drlon,drtime]
ans =
   10.0000    0.0846    1.0000   % Position at 1 am
   10.0000    0.1058    1.2500   % Position at speed change
   10.0000    0.1301    1.4552   % Time of course change
   10.0570    0.1586    2.0000   % Position at 2 am
```

```
    10.1001    0.1801    2.4113   % Time at final waypoint
```

This following illustration shows the difference:



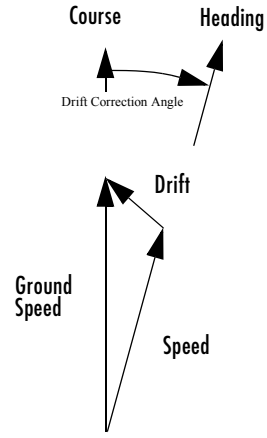
The times at planned positions after the speed change are a little earlier; the position at the known time (2 a.m.) is a little farther along. With this plan, you will arrive at the rendezvous about 4 1/2 minutes earlier, so you may want to consider a greater speed change.

Drift Correction

Dead reckoning is a reasonably accurate method for predicting position if the vehicle is able to maintain the planned course. Aircraft and ships can be pushed off the planned course by winds and current. An important step in navigational planning is to calculate the required drift correction.

In the standard drift correction problem, the desired course and wind are known, but the heading needed to stay on course is unknown. This problem is well suited to vector analysis. The wind velocity is a vector of known magnitude and direction. The vehicle's speed relative to the moving air mass is a vector of known magnitude, but unknown direction. This heading must

be chosen so that the sum of the vehicle and wind velocities gives a resultant in the specified course direction. The ground speed can be larger or smaller than the air speed because of headwind or tailwind components. A navigator would like to know the required heading, the associated wind correction angle, and the resulting ground speed.



What heading puts an aircraft on a course of 250° when the wind is 38 knots from 285°? The aircraft flies at an airspeed of 145 knots.

```
course = 250; airspeed = 145; windfrom = 285; windspeed = 38;
[heading,groundspeed,windcorrangle] = ...
driftcorr(course,airspeed,windfrom,windspeed)

heading =
        258.65

groundspeed =
        112.22

windcorrangle =
          8.65
```

The required heading is about 9° to the right of the course. There is a 33-knot headwind component.

A related problem is the calculation of the wind speed and direction from observed heading and course. The wind velocity is just the vector difference of the ground speed and the velocity relative to the air mass.

```
[windfrom,windspeed] = ...
driftvel(course,groundspeed,heading,airspeed)

windfrom =
        285.00

windspeed =
         38.00
```

Time Notation

General Time Notation

Times can be represented as variables in the Mapping Toolbox in three ways: hours, seconds, and hours-minutes-seconds. The toolbox provides functions for converting among these formats.

Hours. This is the default time unit notation for the toolbox.

Hour notation is simply decimal notation in terms of hours. Two hours and fifteen minutes would be 2.25.

Seconds. Seconds notation is simply decimal notation in terms of seconds. One hour would be 3600.

Hours-Minutes-Seconds. Hours-minutes-seconds, or *hms* notation, is analogous to dms notation for angles. In text, an hms time would be *hh*:*mm*:*ss*. For example, 12:36:15 is 12 hours, 36 minutes, and 15 seconds. In the Mapping Toolbox, when hms times are represented by a single number, the format is *hhmm.ss*. For example, 12:36:15 is 1236.15.

The primary value of this notation is in entering data already in this format. The toolbox includes the `mat2hms` function to easily input hms data, which functions very similarly to the `mat2dms` function described in Converting Latitude and Longitude Notations (p. 7-2).

Note You must exercise care when working with the hms format; for example, two times in this format cannot simply be added. You should convert hms data to decimal hours before working extensively with it.

Converting Between Time Unit Formats. Time units can be converted using functions similar to those described for angle unit conversions. These include `hr2sec` and `hms2hr`, as well as a general conversion function, `timedim`, which works just like `angledim`.

Navigational Time Notation

Navigational practice has its own peculiar notation for times. Time labels on navigation plots are always in a special format. Times are given in four digits, hours from 00 to 23 followed by minutes from 00 to 59. So, one minute before noon is 1159, or 1159Z or 1159Q, etc., based on time zone. Similarly, one minute after midnight is 0001. When more precision is required, the seconds are rounded to the nearest quarter minute and zero, one, two or three apostrophes are suffixed to the time, one for each 15-second block. So, 15 seconds before noon would be 1159'''; 14 seconds before noon would have the exact same notation.

The Mapping Toolbox includes the function `time2str` that returns a string in a variety of formats corresponding to a given time. These strings can then be plotted on map displays as desired. Two other clock formats are also allowed — the 12-hour and the 24-hour digital clock readouts. Consider some string notations for the time 13.21 hours after midnight. The default 24-hour clock is

```
time2str(13.21)
ans =
13:12:36
```

The 12-hour clock reads

```
time2str(13.21,'12')
ans =
01:12:36 PM
```

And the navigation format for this time is

```
time2str(13.21,'nav')
ans =
1312''
```

Each of these can be rounded to the nearest minute with the third argument `hm` (for hours-minutes – the default is `hms`)

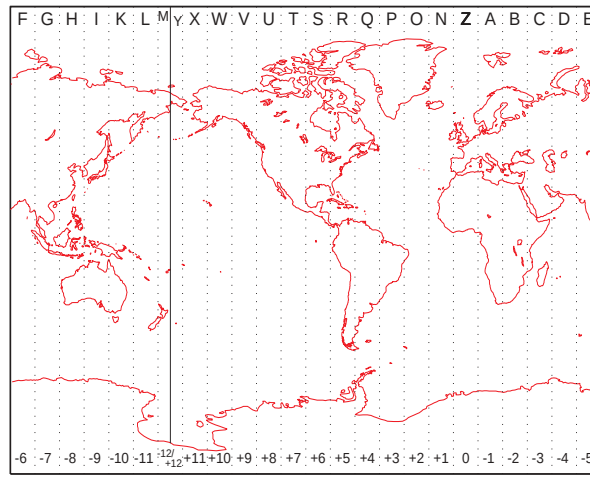
```
time2str(13.21,'nav','hm')
ans =
1313
```

Time Zones

Time zones used for navigation are uniform 15° extents of longitude. The `timezone` function returns a navigational time zone, that is, one based solely on longitude with no regard for statutory divisions. So, for example, Chicago, Illinois, lies in the statutory U.S. Central time zone, which has irregular boundaries devised for political or convenience reasons. However, from a navigational standpoint, Chicago's longitude places it in the *S* (Sierra) time zone. The zone's *description* is +6, which indicates that 6 hours must be added to local time to get Greenwich, or *Z* (Zulu) time. So, if it is noon, standard time in Chicago, it is 12+6, or 6 p.m., at Greenwich.

Each 15° navigational time zone has a distinct description and designating letter. The exceptions to this are the two zones on either side of the Date Line, *M* and *Y* (Mike and Yankee). These zones are only 7-1/2° wide, since on one side of the Date Line, the description is +12, and on the other, it is -12.

Navigational time zones are very important for celestial navigation calculations. Although the Mapping Toolbox does not contain any functions designed specifically for celestial navigation, a simple example can be devised.



It is possible with a sextant to determine *local apparent noon*. This is the moment when the Sun is at its zenith from your point of view. At the exact center longitude of a time zone, the phenomenon occurs exactly at noon, local time. Since the Sun traverses a 15° time zone in 1 hour, it crosses one degree every 4 minutes. So if you observe local apparent noon at 11:54, you must be 1.5° east of your center longitude.

You must know what time zone you are in before you can even attempt a fix. This concept has been understood since the spherical nature of the Earth was first accepted, but early sailors had no ability to keep accurate time on ship, and so were unable to determine their longitude. The invention of accurate chronometers in the 18th century solved this problem.

The `timezone` function is quite simple. It returns the description, `zd`, an integer for use in calculations, a string, `zltr`, of the zone designator, and a string fully naming the `zone`. For example, the information for a longitude 123°E is the following:

```
[zd,zltr,zone] = timezone(123)
zd =
    -8
zltr =
H
zone =
    -8 H
```

Returning to the simple celestial navigation example, the center longitude of this zone is:

```
-(zd*15)
ans =
    120
```

This means that at our longitude, 123°E, we should experience local apparent noon at 11:48 a.m., 12 minutes early.

10

Reference

Guide to Reference Pages

The Mapping Toolbox reference material includes the following:

Mapping Function Reference

- Functions — Categorical List
- Functions — Alphabetical List

Projections Reference

- Map Projections — Alphabetical List

GUI Reference

- Graphical User Interface Functions — Categorical List
- Graphical User Interface Functions — Alphabetical List

Mapping Toolbox Data Reference

- Atlas Data

Bibliography

Geographic Terms

- Glossary

Functions — Categorical List

The following table indexes categories of functions that are grouped together in tables below. Each function has a one-line description and a link to its reference page.

Geospatial Data Import and Access

Standard File Formats

Gridded Terrain and Bathymetry Products

Vector Map Products

Miscellaneous Data Sets

Graphical User Interfaces for Data Import

File Reading Utilities

Ellipsoids, Radii, Areas, and Volumes

Atlas Data Functions

Atlas Data MAT-Files

Vector Map Data and Geographic Data Structures

Geographic Data Structures

Data Manipulation

Georeferenced Images and Data Grids

Spatial Referencing

Visibility in Terrain

Other Analysis/Access

Construction and Modification

Initialization

Map Projections and Coordinates

Available Map Projections

Map Projection Transformations

Angles, Scales, and Distortions

Visualizing Map Distortions

Cylindrical Projections

Pseudocylindrical Projections

Conic Projections

Polyconic and Pseudoconic Projections

Azimuthal, Pseudoazimuthal, and Modified Azimuthal Projections

UTM and UPS Systems

Three-Dimensional Globe Display

Longitude Wrapping

Rotating Coordinates on the Sphere

Trimming and Clipping

Map Display and Interaction

Map Creation and High-Level Display

Vector Symbolization

Automated Base Map Creation

Displaying Lines and Contours

Displaying Patch Data

Displaying Data Grids

Displaying Light Objects and Lighted Surfaces

Dislaying Thematic Maps

Annotating Map Displays

Colormaps for Map Displays

Interactive Map Positions

Interactive Track and Circle Definition

Graphical User Interfaces

Map Object and Projection Properties

Controlling Map Appearance

Clearing Map Displays/Managing Visibility

Geographic Calculations

Geometry of Sphere and Ellipsoid

Ellipsoids and Latitudes

Intersections in the Cartesian Plane

Geographic Statistics

Navigation

Utilities

Map Trimming

Data Precision

Conversion Factors for Angles and Distances

Angle Conversions

Distance Conversions

Time Conversions

Geospatial Data Import and Access

Standard File Formats

`arcgridread`	Read a gridded data set in Arc ASCII Grid Format.
`geotiffinfo`	Information about a GeoTIFF file.
`geotiffread`	Read a georeferenced image from GeoTIFF file.
`getworldfilename`	Derive a worldfile name from an image file name
`sdtsdemread`	Read data from an SDTS raster/DEM data set.
`sdtsinfo`	Information about an SDTS data set.
`shapeinfo`	Information about a shapefile.
`shaperead`	Read vector feature coordinates and attributes from a shapefile.
`worldfileread`	Read a worldfile and return a referencing matrix.
`worldfilewrite`	Construct a worldfile from a referencing matrix.

Gridded Terrain and Bathymetry Products

`dted`	Read U.S. Dept. of Defense Digital Terrain Elevation Data (DTED).
`dteds`	Return DTED data file names covering a latitude-longitude box.
`etopo5`	Read 5-minute gridded terrain/bathymetry from global ETOPO5 data set.
`globedem`	Read Global Land One-km Base Elevation (GLOBE) elevation data.
`globedems`	Return GLOBE data file names covering a latitude-longitude box.
`gtopo30`	Read 30-arc-second global digital elevation model (GTOPO30).

`gtopo30s`	Return GTOPO30 data file names covering a latitude-longitude box.
`satbath`	Read 2-minute global terrain/bathymetry from Smith and Sandwell.
`tbase`	Read 5-minute global terrain elevations from TerrainBase.
`usgs24kdem`	Read a USGS 7.5-minute (30-meter) Digital Elevation Model.
`usgsdem`	Read a USGS 1-degree (3-arc-second) Digital Elevation Model.
`usgsdems`	Return USGS 1-degree DEM file names covering a latitude-longitude box.

Vector Map Products

`dcwdata`	Read selected data from the Digital Chart of the World.
`dcwgaz`	Search for entries in a Digital Chart of the World gazette file.
`dcwrdx`	Read the Digital Chart of the World index file.
`dcwread`	Read a Digital Chart of the World file.
`dcwrhead`	Read Digital Chart of the World file headers.
`fipsname`	Read the name file used to index the TIGER thinned boundary files.
`gshhs`	Read Global Self-Consistent Hierarchical High-Resolution Shoreline.
`tgrline`	Read TIGER/Line data.
`tigermif`	Read a TIGER MIF thinned boundary file.
`tigerp`	Read TIGER p and pa thinned boundary files.
`vmap0data`	Read selected data from the Vector Map Level 0 CD-ROMs.

`vmap0rdx`	Read the Vector Map Level 0 index file.
`vmap0read`	Read a Vector Map Level 0 file.
`vmap0rhead`	Read Vector Map Level 0 file headers.

Miscellaneous Data Sets

`avhrrgoode`	Read AVHRR data product stored in Goode projection.
`avhrrlambert`	Read AVHRR data product stored in Lambert projection.
`egm96geoid`	Read 15-minute gridded geoid heights from EGM96 global geoid model.
`readfk5`	Read the Fifth Fundamental Catalog of stars and its extension.

Graphical User Interfaces for Data Import

`demdataui`	Interactively select elevation data from external sources.
`vmap0ui`	Extract selected data from Vector Map Level 0 CD-ROMs.

File Reading Utilities

`grepfields`	Identify matching records in fixed record length files.
`readfields`	Read fields or records from a fixed format file.
`readmtx`	Read a matrix stored in a file.
`spcread`	Read columns of data from an ASCII text file.

Ellipsoids, Radii, Areas, and Volumes

`almanac`	Parameters for Earth and other objects in the solar system.

Atlas Data Functions

`country2mtx`	Create a raster map grid of a country from `worldlo` data.
`usahi`	Return high-resolution vector data for the United States.
`usalo`	Return vector data for the United States.
`worldhi`	Return high-resolution vector data for the world.
`worldlo`	Return vector data for the world or oceans.

Atlas Data MAT-Files

See Appendix A for descriptions of these data sets.

`coast`	World coastline latitude and longitude arrays.
`geoid`	Global geoid height grid in meters at one cell/degree.
`oceanlo`	Ocean mask patches.
`usahi`	High-resolution United States vector data.
`usalo`	United States vector data.
`usamtx`	Data grid at 5 cells/degree for states in the USA.
`worldhi`	High-resolution world vector data.
`worldlo`	World vector data.
`worldmtx`	Data grid at one cell/degree for countries of the world.
`worldmtxmed`	Data grid at 4 cells/degree for countries of the world.

Vector Map Data and Geographic Data Structures

Geographic Data Structures

`extractfield`	Extract the field values from a structure.
`extractm`	Extract coordinates from a v1 geographic data structure.
`updategeostruct`	Update a geographic data structure.

Data Manipulation

`bufferm`	Compute buffer zones for vector data.
`flatearthpoly`	Insert points along the date line to the pole.
`interpm`	Interpolate vector data to a specified data separation.
`intrplat`	Interpolate a latitude for a given longitude.
`intrplon`	Interpolate a longitude for a given latitude.
`nanclip`	Clip vector data with NaNs at specified pen-down locations.
`polybool`	Perform Boolean operations on polygons.
`polycut`	Compute branch cuts for holes in polygons.
`polyjoin`	Convert polygon segments from cell array to vector format.
`polymerge`	Merge line segments with matching endpoints.
`polysplit`	Extract segments of NaN-delimited polygon vectors to cell arrays.
`polyxpoly`	Compute line or polygon intersection points.
`reducem`	Reduce the density of points in vector data.

Georeferenced Images and Data Grids

Spatial Referencing

`latlon2pix`	Convert latitude-longitude coordinates to pixel coordinates.
`limitm`	Calculate latitude/longitude bounds for a regular data grid.
`makerefmat`	Construct an affine spatial-referencing matrix.
`map2pix`	Convert map coordinates to pixel coordinates.
`mapbbox`	Compute bounding box of a georeferenced image or data grid.
`mapoutline`	Compute outline of a georeferenced image or data grid.
`meshgrat`	Construct a graticule for a surface map object.
`pix2latlon`	Convert pixel coordinates to latitude-longitude coordinates.
`pix2map`	Convert pixel coordinates to map coordinates.
`pixcenters`	Compute pixel centers for georeferenced image or data grid.
`refmat2vec`	Convert a referencing matrix to a referencing vector.
`refvec2mat`	Convert a referencing vector to a referencing matrix.
`setltln`	Convert data grid rows and columns to latitude-longitude.
`setpostn`	Convert latitude-longitude to data grid rows and columns.

Visibility in Terrain

`viewshed`	Areas visible from a point on a digital elevation model.
`los2`	Line of sight visibility between two points in terrain.

Other Analysis/Access

`areamat`	Surface area covered by nonzero values in regular data grid.
`findm`	Return latitude/longitude of nonzero data grid elements.
`gradientm`	Calculate gradient, slope, and aspect of data grid.
`ltln2val`	Extract data grid values for specified locations.
`mapprofile`	Interpolate between waypoints on a regular data grid.

Construction and Modification

`changem`	Substitute values in a data array.
`encodem`	Fill in regular data grid from seed values and locations.
`geoloc2grid`	Convert a geolocated data array to a regular data grid.
`imbedm`	Encode data points into a regular data grid.
`neworig`	Rotate a regular data grid on the sphere.
`resizem`	Resize a regular data grid.
`sizem`	Row and column dimension needed for a regular data grid.
`vec2mtx`	Convert latitude-longitude vectors to a regular data grid.

Initialization

`nanm`	Construct a regular data grid of all NaNs.
`onem`	Construct a regular data grid of all ones.
`spzerom`	Construct a sparse regular data grid of all zeros.
`zerom`	Construct a regular data grid of all zeros.

Map Projections and Coordinates

Available Map Projections

`maps`	List available map projections and verify names.
`maplist`	Return a structure containing the map projections available in the Mapping Toolbox.
`projlist`	List map projections supported by `projfwd` and `projinv`.

Map Projection Transformations

`mfwdtran`	Process forward transformation.
`minvtran`	Process inverse transformation.
`projfwd`	Forward map projection using the PROJ.4 library.
`projinv`	Inverse map projection using the PROJ.4 library.

Angles, Scales, and Distortions

`vfwdtran`	Transform azimuth to direction angle on map plane.
`vinvtran`	Transform direction angle from map plane to azimuth.
`distortcalc`	Calculate distortion parameters for a map projection.

Visualizing Map Distortions

`mdistort`	Display contours of constant map distortion.
`tissot`	Project Tissot indicatrices on a map.

Cylindrical Projections

`balthsrt`	Balthasart Projection.
`behrmann`	Behrmann Projection.
`bsam`	Bolshoi Sovietskii Atlas Mira Projection.
`braun`	Braun Perspective Projection.
`cassini`	Cassini Projection.
`ccylin`	Central Cylindrical Projection.
`eqacylin`	Equal Area Projection.
`eqdcylin`	Equidistant Projection.
`giso`	Gall Isographic Projection.
`gortho`	Gall Orthographic Projection.
`gstereo`	Gall Stereographic Projection.
`lambcyln`	Lambert Projection.
`mercator`	Mercator Projection.
`miller`	Miller Projection.
`pcarree`	Plate Carrée Projection.
`tranmerc`	Transverse Mercator Projection.
`trystan`	Trystan Edwards Projection.
`wetch`	Wetch Projection.

Pseudocylindrical Projections

`apianus`	Apianus II Projection.
`collig`	Collignon Projection.
`craster`	Craster Parabolic Projection.
`eckert1`	Eckert I Projection.
`eckert2`	Eckert II Projection.
`eckert3`	Eckert III Projection.
`eckert4`	Eckert IV Projection.
`eckert5`	Eckert V Projection.
`eckert6`	Eckert VI Projection.
`flatplrp`	Flat-Polar Parabolic Projection.
`flatplrq`	Flat-Polar Quartic Projection.
`flatplrs`	Flat-Polar Sinusoidal Projection.
`fournier`	Fournier Projection.
`goode`	Goode Homolosine Projection.
`hatano`	Hatano Assymmetrical Equal Area Projection.
`kavrsky5`	Kavraisky V Projection.
`kavrsky6`	Kavraisky VI Projection.
`loximuth`	Loximuthal Projection.
`modsine`	Modified Sinusoidal Projection.
`mollweid`	Mollweide Projection.
`putnins5`	Putnins P5 Projection.
`quartic`	Quartic Authalic Projection.
`robinson`	Robinson Projection.
`sinusoid`	Sinusoidal Projection.

`wagner4`	Wagner IV Projection.
`winkel`	Winkel I Projection.

Conic Projections

`eqaconic`	Albers Equal Area Conic Projection.
`eqdconic`	Equidistant Conic Projection.
`lambert`	Lambert Conformal Conic Projection.
`murdoch1`	Murdoch I Conic Projection.
`murdoch3`	Murdoch III Minimum Error Conic Projection.

Polyconic and Pseudoconic Projections

`bonne`	Bonne Projection.
`polycon`	Polyconic Projection.
`vgrint1`	Van Der Grinten I Projection.
`werner`	Werner Projection.

Azimuthal, Pseudoazimuthal, and Modified Azimuthal Projections

`aitoff`	Aitoff Projection.
`breusing`	Breusing Harmonic Mean Projection.
`bries`	Briesemeister's Projection.
`eqaazim`	Lambert Equal Area Azimuthal Projection.
`eqdazim`	Equidistant Azimuthal Projection.
`gnomonic`	Gnomonic Azimuthal Projection.
`hammer`	Hammer Projection.
`ortho`	Orthographic Azimuthal Projection.
`stereo`	Stereographic Azimuthal Projection.

`vperspec`	Vertical Perspective Azimuthal Projection.
`wiechel`	Weichel Equal Area Projection.

UTM and UPS Systems

`ups`	Universal Polar Stereographic (UPS) Projection.
`utm`	Universal Transverse Mercator (UTM) Projection.
`utmgeoid`	Select ellipsoid for a given UTM zone.
`utmzone`	Select a UTM zone.

Three-Dimensional Globe Display

`globe`	Render Earth as a sphere in 3-D graphics.

Longitude Wrapping

`eastof`	Wrap longitudes to values east of a meridian.
`npi2pi`	Wrap latitudes to the [-180 180] degree interval.
`smoothlong`	Remove discontinuities in longitude data.
`westof`	Wrap longitudes to values west of a meridian.
`zero22pi`	Wrap longitudes to the [0 360) degree interval.

Rotating Coordinates on the Sphere

`newpole`	Compute origin vector to rotate a point to the pole.
`org2pol`	Compute location of the North Pole in a rotated map.
`putpole`	Compute origin vector to rotate North Pole to a specific point.

Trimming and Clipping

`clipdata`	Clip map data at the -pi to pi border of a display.
`trimdata`	Trim map data exceeding projection limits.
`undoclip`	Remove object clips introduced by CLIPDATA.
`undotrim`	Remove object trims introduced by TRIMDATA.

Map Display and Interaction

Map Creation and High-Level Display

`axesm`	Create a new map axes/define a map projection.
`displaym`	Project features from a v1 geographic data structure.
`geoshow`	Display map latitude and longitude data.
`grid2image`	Display a regular data grid as an image.
`mapshow`	Display map data.
`mapview`	Interactive map viewer.

Vector Symbolization

`makesymbolspec`	Construct a vector symbolization specification.

Automated Base Map Creation

`usamap`	Map the United States of America using atlas data.
`worldmap`	Map a country, region, or the world using atlas data.

Displaying Lines and Contours

`contourm`	Project a contour plot of map data.
`contour3m`	Project a contour plot of map data in 3-D space.
`contourfm`	Project a filled contour plot of map data.
`linem`	Create and project a line.
`plotm`	Project lines and points.
`plot3m`	Project lines and points in 3-D space.

Displaying Patch Data

`fillm`	Project filled 2-D map polygons.
`fill3m`	Project filled 3-D map polygons in 3-D space.
`patchesm`	Project patches as individual objects.
`patchm`	Project patch objects.

Displaying Data Grids

`meshm`	Warp a regular data grid to a projected graticule mesh.
`pcolorm`	Project a regular data grid in the z = 0 plane.
`surfacem`	Warp geolocated data to a projected graticule mesh.
`surfm`	Project a geolocated data grid on a map axes.

Displaying Light Objects and Lighted Surfaces

`lightm`	Project a light source onto the current map.
`meshlsrm`	Project 3-D lighted shaded relief for regular data grid.
`surflm`	Project a geolocated data grid with lighting.
`surflsrm`	Project 3-D lighted shaded relief for geolocated data.
`shaderel`	Construct cdata and colormap for colored shaded relief.

Dislaying Thematic Maps

`cometm`	Project a 2-D comet plot.
`comet3m`	Project a 3-D comet plot.
`quiverm`	Project a 2-D quiver plot.
`quiver3m`	Project a 3-D quiver plot.

`scatterm`	Project point markers with variable color and area.
`stem3m`	Project a stem map.
`symbolm`	Project point markers with variable size.

Annotating Map Displays

`clabelm`	Add contour labels to a map contour plot.
`clegendm`	Add legend labels to a map contour plot.
`degchar`	Return the LaTeX degree symbol character.
`framem`	Toggle and control the display of the map frame.
`gridm`	Toggle and control the display of the map grid.
`lcolorbar`	Append a colorbar with text labels.
`mlabel`	Toggle and control the display of meridian labels.
`mlabelzero22pi`	Convert meridian labels to the range [0,360] degrees.
`northarrow`	Add graphic element pointing to the geographic North Pole.
`plabel`	Toggle and control the display of parallel labels.
`rotatetext`	Rotate text to the projected graticule.
`scaleruler`	Add graphic scale.
`textm`	Project text annotation on a map.

Colormaps for Map Displays

`contourcmap`	Create a contour colormap for a projected data grid.
`demcmap`	Create a colormap appropriate to terrain elevation data.
`polcmap`	Create a colormap appropriate to a political map.

Interactive Map Positions

`gcpmap`	Get current mouse point from the map.
`gtextm`	Place text on a 2-D map using a mouse.
`inputm`	Return latitudes and longitudes of mouse click positions.

Interactive Track and Circle Definition

`scircleg`	Display a small circle defined via mouse input.
`sectorg`	Display a small circle sector defined via mouse input.
`trackg`	Display a great circle or rhumb line by mouse input.

Graphical User Interfaces

`axesmui`	Interactively define map axes properties.
`clrmenu`	Add a colormap menu to a figure window.
`cmapui`	Create custom colormap.
`colorm`	Create index map colormaps.
`colorui`	Interactively define an RGB color.
`getseeds`	Get seed locations and values for encoding maps.
`lightmui`	Control position of lights on a globe or 3-D map.
`maptrim`	Customize map data sets.
`maptool`	Add menu activated tools to a map figure.
`mlayers`	Manipulate map layers defined with structure data.
`mobjects`	Manipulate object sets displayed on an axes.
`originui`	Interactively modify map origin.
`panzoom`	Pan and zoom on a 2-D plot.

`parallelui`	Interactively modify map parallels.
`qrydata`	Create queries associated with map axes.
`rootlayr`	Construct mlayer cell array input for user workspace.
`scirclui`	Interactively add small circles to a map.
`seedm`	Seed regular matrix maps.
`trackui`	Interactively add great circles and rhumb lines to a map.
`uimaptbx`	Process button down callbacks in Mapping Toolbox.
`utmzoneui`	Choose or identify a UTM zone by clicking on a map.

Map Object and Projection Properties

`cart2grn`	Transform from projected coordinates to Greenwich frame.
`defaultm`	Initialize or reset projection properties to default values.
`gcm`	Get current map projection structure.
`geotiff2mstruct`	Convert GeoTIFF info to a map projection structure.
`getm`	Get map object properties.
`handlem`	Get handle of displayed map objects.
`ismap`	True if axes have a map projection defined.
`ismapped`	True if object is projected on a map axes.
`makemapped`	Make an object a mapped object.
`namem`	Determine the names for valid graphics objects.
`project`	Project a displayed graphics object.
`restack`	Restack objects within the axes.
`rotatem`	Transform map data to new origin and orientation.
`setm`	Set and modify properties of a map.
`tagm`	Assign a name to a graphics object using the tag property.
`zdatam`	Adjust the *z* plane of displayed map objects.

Controlling Map Appearance

`axesscale`	Resize axes for equivalent scale.
`camposm`	Set axes camera position using geographic coordinates.
`camtargm`	Set axes camera target using geographic coordinates.

`camupm`	Set axes camera up vector using geographic coordinates.
`daspectm`	Set the figure `DataAspectRatio` property for a map.
`paperscale`	Set the figure paper size for a given map scale.
`previewmap`	Preview map at printed size.
`tightmap`	Remove white space around a map.

Clearing Map Displays/Managing Visibility

`clma`	Clear current map axes.
`clmo`	Clear specified graphic objects from map axes.
`hidem`	Hide specified graphic objects on map axes.
`showaxes`	Toggle display of map coordinate axes.
`showm`	Show specified graphic objects.
`trimcart`	Trim graphic objects to the map frame.

Geographic Calculations

Geometry of Sphere and Ellipsoid

`antipode`	Point on the opposite side of the globe.
`areaint`	Surface area of a polygon on a sphere or ellipsoid.
`areaquad`	Surface area of a latitude-longitude quadrangle.
`azimuth`	Azimuth between points on a sphere/ellipsoid.
`departure`	Compute departure of longitudes at specific latitudes.
`distance`	Distance between points on a sphere/ellipsoid.
`elevation`	Elevation angle between points on a sphere/ellipsoid.
`ellipse1`	Construct ellipse from center, semimajor axes, eccentricity, and azimuth.
`gc2sc`	Compute center and radius of a great circle.
`gcxgc`	Compute intersection points between great circles.
`gcxsc`	Compute intersection points between great and small circles.
`reckon`	Point at specified azimuth, range on a sphere/ellipsoid.
`rhxrh`	Compute intersection points between rhumb lines.
`scircle1`	Construct small circle from center, range, and azimuth.
`scircle2`	Construct small circle from center and perimeter.
`scxsc`	Compute intersection points between small circles.

`track1`	Construct track lines from starting point, azimuth, and range.
`track2`	Construct track lines from starting and ending points.

Ellipsoids and Latitudes

`axes2ecc`	Compute eccentricity from semimajor and semiminor axes.
`convertlat`	Convert between geodetic and auxiliary latitudes.
`ecc2flat`	Compute flattening of an ellipse from eccentricity.
`ecc2n`	Compute parameter n of an ellipse from eccentricity.
`flat2ecc`	Compute eccentricity of an ellipse from flattening.
`majaxis`	Compute semimajor axis from semiminor axis and eccentricity.
`minaxis`	Compute semiminor axis from semimajor axis and eccentricity.
`n2ecc`	Compute eccentricity of an ellipse from parameter n.
`rcurve`	Compute radii of curvature for an ellipsoid.
`rsphere`	Compute radii for auxiliary spheres.

Intersections in the Cartesian Plane

`circcirc`	Intersections of circles in a Cartesian plane.
`linecirc`	Intersections of circles and lines in a Cartesian plane.

Geographic Statistics

`combntns`	Compute all combinations of a given set of values.
`eqa2grn`	Convert equal-area coordinates to Greenwich coordinates.
`filterm`	Filter data points geographically.
`grn2eqa`	Convert Greenwich coordinates to equal-area coordinates.
`hista`	Histogram for geographic points with equal-area bins.
`histr`	Histogram for geographic points with equirectangular bins.
`meanm`	Compute mean for geographic point locations.
`stdist`	Compute standard distance for geographic point locations.
`stdm`	Compute standard deviation for geographic point locations.

Navigation

`crossfix`	Compute cross fix positions for bearings and ranges.
`dreckon`	Compute dead reckoning positions for a track.
`driftcorr`	Compute heading to correct for wind or current drift.
`driftvel`	Compute drift speed and direction.
`gcwaypts`	Compute equally spaced waypoints along a great circle.
`legs`	Compute courses and distances between waypoints along a track.
`navfix`	Perform mercator-based navigational fixing.
`timezone`	Compute time zone description from longitude.
`track`	Connect navigational waypoints with track segments.

Utilities

Map Trimming

`maptriml`	Trim a line map to a specified region.
`maptrimp`	Trim a patch map to a specified region.
`maptrims`	Trim surface map to a specified region.

Data Precision

`epsm`	Return accuracy in angle units of certain map computations.
`roundn`	Round to specified power of 10.

Conversion Factors for Angles and Distances

`unitsratio`	Unit conversion factors.

Angle Conversions

`angl2str`	Format an angle string.
`angledim`	Convert angles from one unit or format to another.
`deg2dm`	Convert angles from degrees to deg:min vector format.
`deg2dms`	Convert angles from degrees to deg:min:sec vector format.
`deg2rad`	Convert angles from degrees to radians.
`dms2deg`	Convert angles from deg:min:sec to degrees.
`dms2dm`	Convert angles from deg:min:sec to deg:min vector format.
`dms2mat`	Convert a dms vector format to a [deg min sec] matrix.
`dms2rad`	Convert angles from deg:min:sec to radians.
`mat2dms`	Convert a [deg min sec] matrix to vector format.
`rad2deg`	Convert angles from radians to degrees.
`rad2dm`	Convert angles from radians to deg:min vector format.
`rad2dms`	Convert angles from radians to deg:min:sec vector format.
`str2angle`	Convert formatted DMS angle strings to numbers.

Distance Conversions

`deg2km`	Convert distances from degrees to kilometers.
`deg2nm`	Convert distances from degrees to nautical miles.
`deg2sm`	Convert distances from degrees to statute miles.
`dist2str`	Format a distance string.
`distdim`	Convert distances from one unit or format to another.
`km2deg`	Convert distances from kilometers to degrees.
`km2nm`	Convert distances from kilometers to nautical miles.
`km2rad`	Convert distances from kilometers to radians.
`km2sm`	Convert distances from kilometers to statute miles.
`nm2deg`	Convert distances from nautical miles to degrees.
`nm2km`	Convert distances from nautical miles to kilometers.
`nm2rad`	Convert distances from nautical miles to radians.
`nm2sm`	Convert distances from nautical miles to statute miles.
`rad2km`	Convert distances from radians to kilometers.
`rad2nm`	Convert distances from radians to nautical miles.
`rad2sm`	Convert distances from radians to statute miles.
`sm2deg`	Convert distances from statute miles to degrees.
`sm2km`	Convert distances from statute miles to kilometers.
`sm2nm`	Convert distances from statute miles to nautical miles.
`sm2rad`	Convert distances from statute miles to radians.

Time Conversions

`hms2hm`	Convert time from hrs:min:sec to hr:min vector format.
`hms2hr`	Convert time from hrs:min:sec to hours.
`hms2mat`	Convert a hms vector format to a [hrs min sec] matrix.
`hms2sec`	Convert time from hrs:min:sec to seconds.
`hr2hm`	Convert time from hours to hrs:min format.
`hr2hms`	Convert time from hours to hrs:min:sec vector format.
`hr2sec`	Convert time from hours to seconds.
`mat2hms`	Convert a [hrs min sec] matrix to vector format.
`sec2hm`	Convert time from seconds to hrs:min vector format.
`sec2hms`	Convert time from seconds to hrs:min:sec vector format.
`sec2hr`	Convert time from seconds to hours.
`time2str`	Format a time string.
`timedim`	Convert times from one unit or format to another.

Functions — Alphabetical List

This section provides reference entries with detailed descriptions of all the Mapping Toolbox functions, listed alphabetically.

The following headings are used in this section. Not every function will have descriptions for all of these entries, but the information that is given will be ordered as shown.

Purpose

Syntax

Background

Description

Axes Definition

Examples

Object Properties

Limitations

Remarks

See Also

almanac

Purpose

Display planetary data for the nine planets, the Sun, and the Moon

Syntax

```
almanac
almanac(body)
data = almanac(body,parameter)
data = almanac(body,parameter,units)
data = almanac(body,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 each celestial body. Valid *`body`* strings are

```
'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`* strings 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 valid predefined ellipsoid string. In this case, the two-element ellipsoid vector for that ellipsoid model is returned. Valid ellipsoid definition strings for the Earth are

`'everest'`	1830 Everest ellipsoid
`'bessel'`	1841 Bessel ellipsoid
`'airy'`	1849 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

For the Earth, the *`parameter`* strings `'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 string. 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 *`referencebody`* string of `'actual'` returns a tabulated value rather than one dependent upon a ellipsoid model assumption. Other possible *`referencebody`* strings 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 strings 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.

Examples

The radius of the Earth (treated as a sphere) in kilometers is

```
almanac('earth','radius')
ans =
   6371
```

The default ellipsoid model for the Earth (`[semimajor axis eccentricity]`) is

```
almanac('earth','ellipsoid')
```

```
ans =
   1.0e+03 *
       6.3781    0.0001
```

Note that the radius returned for any ellipsoid model reference body is the semimajor axis:

```
almanac('earth','radius','kilometers','ellipsoid')
Warning: Semimajor axis returned for radius parameter
ans =
   6.3781e+03
```

Compare the tabulated values of the Earth's surface area with a spherical assumption and with the 1966 World Geodetic System ellipsoid model:

```
almanac('earth','surfarea','statutemiles','actual')
ans =
     1.969499232704451e+008

almanac('earth','surfarea','statutemiles','sphere')
ans =
     1.969362058529953e+008

almanac('earth','surfarea','statutemiles','wgs66')
ans =
     1.969371331484438e+008
```

Note that these values are so close that long notation is required to differentiate them.

Some lunar measurements are

```
almanac('moon','radius')
ans =
        1738

almanac('moon','surfarea')
ans =
   3.7959e+07

almanac('moon','volume')
ans =
```

```
2.1991e+10
```

Remarks

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`	Distance between points
`distdim`	Convert distance units

angl2str

Purpose

Convert angular values to strings

Syntax

`str = angl2str(angin)` converts the input vector of angles, `angin`, to a string matrix.

`str = angl2str(angin,format)` uses the *`format`* string to specify the notation to be used with the string matrix. The default, `'none'`, results in simple numerical representation (no indicator for positive angles, minus signs for negative angles); `'pm'` (for *plus-minus*) adds a + for positive angles; `'ns'` (for *north-south*) appends an S for negative angles and an N for positive angles; `'ew'` (for *east-west*) appends a W for negative angles and an E for positive angles.

`str = angl2str(angin,format,units)` uses the input *`units`* to define the angle units of the `angin` input. *`units`* is any valid angle string (`'degrees'` are the default). The *`units`* input also determines the unit symbol to suffix to the output strings.

`str = angl2str(angin,format,units,digits)` determines how many digits to display. `digits` is the power of 10 representing the last place of significance in the resulting output. For example, if `digits` = 2, the *hundreds* slot is the last significant figure. In general, the 10^{digits} slot is the last significant figure, rounded appropriately depending upon the value in the $10^{\text{digits}-1}$ slot. `digits` is -2 by default.

Description

The purpose of this function is to make angular-valued variables into strings suitable for map display.

Examples

Create a string matrix to represent a series of values in dms units, using the north-south format:

```
a = -3:1.5:3;
str = angl2str(deg2dms(a),'ns','dms')
str =
3^{\circ} 00' 00.00" S
1^{\circ} 30' 00.00" S
0^{\circ} 00' 00.00"
1^{\circ} 30' 00.00" N
3^{\circ} 00' 00.00" N
```

These LaTeX strings are displayed (using either `text` or `textm`) as

```
  3° 00' 00.00" S
  1° 30' 00.00" S
  0° 00' 00.00"
  1° 30' 00.00" N
  3° 00' 00.00" N
```

See Also

`str2angle`	Convert strings containing DMS formatted angles to numbers
`angledim`	Convert angle units
`dist2str`	Convert distance values to strings
`time2str`	Convert time values to strings

angledim

Purpose

Convert angles between different units

Syntax

`anglout = angledim(anglin,from,to)` returns the value of the input angle `anglin`, which is in units specified by the valid angle units string *`from`*, in the desired units given by the valid angle units string *`to`*. Valid angle units strings are

```
'degrees' for decimal degrees
'radians' for radians
'dms' for degrees-minutes-seconds
'dm' for degrees-minutes
```

Examples

Convert from degrees to radians:

```
angledim(23.45134,'degrees','radians')
ans =
    0.4093
```

What is the difference between dms and dm? (best displayed in *bank format*)

```
format bank
angledim(23.45134,'degrees','dms')
ans =
       2327.05
angledim(23.45134,'degrees','dm')
ans =
       2327.00
```

The dm answer is the dms answer correctly rounded to whole minutes (that is, rounded based on 60 seconds per minute, not 100).

See Also

`angl2str`	Convert angle values to strings
`azimuth`	Angular relationship between points

`deg2dms`
`dms2rad`
`deg2rad`
	Direct angle conversion functions

`distdim`	Convert distance units
`timedim`	Convert time units

Purpose

Determine the antipodes of a geographic point

Syntax

`[newlat,newlong] = antipode(lat,long)` returns the geographic coordinates of the points exactly opposite on the globe from the input points given by `lat` and `long`.

`[newlat,newlong] = antipode(lat,long,`*`units`*`)` specifies the standard angle units string, where *`units`* is any valid angle units string. The default value is `'degrees'`.

Examples

Given a point (43°N, 15°E), find its antipode:

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

or (43°S, 165°W). 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.

arcgridread

Purpose

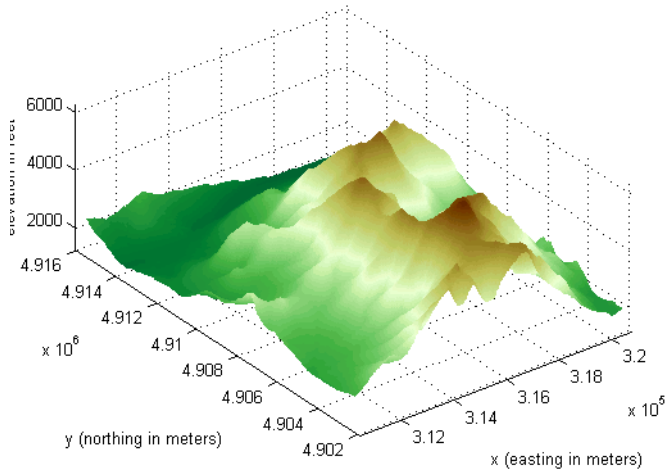
Read a gridded data set in Arc ASCII Grid Format

Syntax

`[Z,R] = arcgridread(filename)` reads a grid from a file in Arc ASCII Grid format. `Z` is a 2-D array containing the data values. `R` is a referencing matrix (see `makrefmat`). `NaN` is assigned to elements of `V` corresponding to null data values in the grid file.

Example

```
[Z,R] = arcgridread('MtWashington-ft.grd');
mapshow(Z,R,'DisplayType','surface');
xlabel('x (easting in meters)'); ylabel('y (northing in meters)')
colormap(demcmap(Z))

% View the terrain in 3D
axis normal; view(3); axis equal; grid on
zlabel('elevation in feet')
```



See Also

`makerefmat`, `mapshow`, `sdtsdemread`

Purpose

Calculate spherical surface area enclosed by a polygon

Syntax

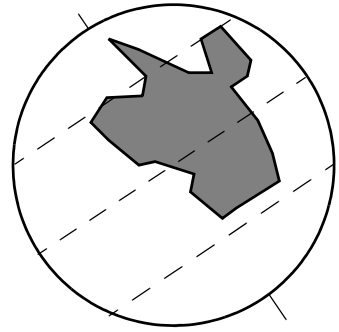
`area = areaint(lats,longs)` returns the surface area enclosed by the polygon defined by the column vectors `lats` and `longs`. Multiple polygons can be delineated by `NaN`s. The output `area` is a fraction of the unit sphere's area of 4π, so the result ranges from 0 to 1.

`area = areaint(lats,longs,ellipsoid)` allows the specification of the ellipsoid model with the two-element ellipsoid vector `ellipsoid`. When an `ellipsoid` is input, the resulting `area` is given in terms of the (squared) units of the `ellipsoid`. For example, if the ellipsoid `almanac('earth','ellipsoid','kilometers')` is used, the resulting area is in km^2. The default ellipsoid is the unit sphere.

`area = areaint(lats,longs,ellipsoid,`*`units`*`)` specifies the units of the inputs `lats` and `longs`, which are `'degrees'` by default.

Description

This function allows 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.



Arbitrarily shaped polygons can be measured.

Examples

Consider the area enclosed by a 30° lune from pole to pole and bounded by the prime meridian and 30°E. 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 30° of latitude:

```
lats = [-90:30:90,60:-30:-60]';
longs = [zeros(1,7),30*ones(1,5)]';
area = areaint(lats,longs)
area =
    0.0792
```

Now, a little finer, perhaps one point every 1° of latitude:

```
lats = [-90:1:90,89:-1:-89]';
longs = [zeros(1,181),30*ones(1,179)]';
area = areaint(lats,longs)
area =
    0.0833
```

Limitations

As noted above, this is a line integral estimation, only as good as the accuracy and the density of the polygon vertex data. However, 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`	Planetary data
`areamat` `areaquad`	Other area calculations

Purpose

Determine geographic area of matrix element

Syntax

`[area,areavec] = areamat(map,refvec)` returns the surface area corresponding to the entries equal to 1 in the regular data grid, `map`, with a three-element referencing vector vector `refvec`. The output `area` is a fraction of the unit sphere's area of 4π, so the result ranges from 0 to 1. Since the area of a given cell is the same within any row of the matrix, the second output, `areavec`, can be useful. It is a vector, having the length of a column of `map`, that provides the cell areas for each row, regardless of whether any element of that row is a 1.

`[area,areavec] = areamat(map,refvec,ellipsoid)` allows the specification of the ellipsoid model with the two-element ellipsoid vector `ellipsoid`. When a `ellipsoid` is input, the resulting `area` is given in terms of the (squared) units of the `ellipsoid`. For example, if the ellipsoid `almanac('earth','ellipsoid','kilometers')` is used, the resulting area is in km^2. The default ellipsoid is the unit sphere.

`area = areaint(lats,longs,ellipsoid,`*`units`*`)` specifies the units of the inputs of the referencing vector, which are `'degrees'` by default.

Description

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 `(topo>0)`, which is 1 everywhere that `topo` is greater than 0 meters, and 0 everywhere else. This is an illustration of that matrix:

Examples

```
load topo
area = areamat((topo>127),topolegend)
```

```
area =
    0.2411
```

Approximately 24% of the Earth has an altitude greater than 127 meters. What is the surface area of this portion of the Earth in square kilometers if a spherical ellipsoid is assumed? (Use the `almanac` function with the sphere as its reference body.)

```
earthgeoid = almanac('earth','ellipsoid','km','sphere');
area = areamat((topo>127),topolegend,earthgeoid)
area =
   1.2299e+08
```

To illustrate the `areavec` output, consider a smaller map:

```
map = ones(9,18);
refvec = [.05 90 0] % each cell 20x20 degrees

[area,areavec] = areamat(map,refvec)
area =
    1.0000
areavec =
    0.0017
    0.0048
    0.0074
    0.0091
    0.0096
    0.0091
    0.0074
    0.0048
    0.0017
```

Each entry of `areavec` represents the portion of the unit sphere's total area a cell in that row of `map` would contribute. Since the column extends from pole to pole in this case, it is symmetric.

Remarks

This calculation is based on the `areaquad` function and is therefore limited only by the granularity of the cellular data resolution and the spherical Earth assumption. For nonspherical ellipsoids, the latitude data is converted to the auxiliary authalic sphere.

See Also

`almanac`	Planetary data
`areaint` `areaquad`	Other area calculations

Purpose

Compute area of a latitude-longitude quadrangle

Syntax

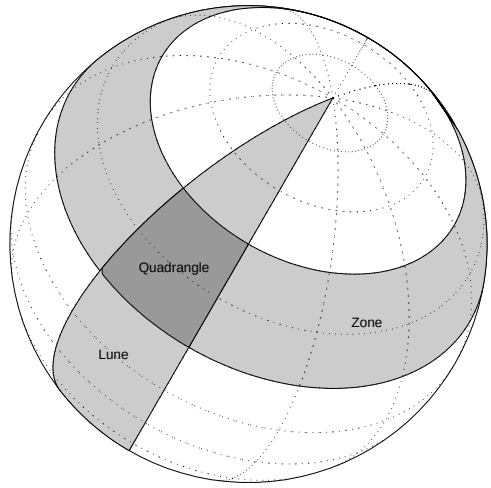
`area = areaquad(lat1,lon1,lat2,lon2)` returns the surface area bounded by the parallels `lat1` and `lat2` and the meridians `lon1` and `lon2`. The output `area` is a fraction of the unit sphere's area of 4π, so the result ranges from 0 to 1.

`area = areaquad(lat1,lon1,lat2,lon2,ellipsoid)` allows the specification of the ellipsoid model with the two-element ellipsoid vector `ellipsoid`. When a `ellipsoid` is input, the resulting `area` is given in terms of the (squared) units of the `ellipsoid`. For example, if the ellipsoid `almanac('earth','ellipsoid','kilometers')` is used, the resulting area is in km^2. The default ellipsoid is the unit sphere.

`area = areaquad(lat1,lon1,lat2,lon2,ellipsoid,`*`units`*`)` specifies the units of the inputs, which are `'degrees'` by default.

Description

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).



Examples

What fraction of the Earth's surface lies between 30°N and 45°N, and also between 25°W and 60°E?

```
area = areaquad(30,-25,45,60)
area =
    0.0245
```

About 2.5%. What is the surface area of the Earth in square kilometers if a spherical ellipsoid is assumed (use the `almanac` function with the sphere as its reference body)?

```
earthellipsoid = almanac('earth','ellipsoid','km','sphere');
area = areaquad(-90,-180,90,180,earthellipsoid)
area =
   5.1006e+08
```

For comparison,

```
almanac('earth','surfarea','km')
ans =
   5.1006e+08
```

Remarks

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

See Also

`almanac`	Planetary data
`areaint` `areamat`	Other area calculations

avhrrgoode

Purpose

Read AVHRR data stored in the Goode Projection

Syntax

`[latgrat,longrat,z] = avhrrgoode` reads data from an AVHRR data set with a nominal resolution of 1 km. These files have 17347 rows and 40031 columns of data, or somewhat more than the capacity of one CD-ROM. The file is selected interactively. Data is returned as a general data grid with the graticule matrices in units of degrees.

`avhrrgoode(`*`region`*`)` reads data from a file with data covering the specified *`region`*. Valid regions are `'g'` or `'global'`, `'af'` or `'africa'`, `'ap'` or `'australia/pacific'`, `'ea'` or `'eurasia'`, `'na'` or `'north america'`, and `'sa'` or `'south america'`. The file is selected interactively. If omitted, `'global'` is assumed.

`avhrrgoode(`*`region`*`,`*`filename`*`)` uses the provided *`filename`*.

`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. If omitted, 100 is assumed.

`avhrrgoode(`*`region`*`,`*`filename`*`,scalefactor,latlim,lonlim)` returns data for the specified region. The returned data will extend somewhat 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]`.

`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. If omitted or empty, a graticule the size of the grid is returned.

`avhrrgoode(`*`region`*`,`*`filename`*`,scalefactor,latlim,lonlim,gsize,fnrows,fncols)` overrides the standard file format for the selected region. This is useful for data stored on CD-ROM, which might have been truncated to fit. Some data was distributed with 16347 rows and 40031 columns of data on CD-ROMs. Nondimensional vegetation index data at 8 km spatial resolution has 2168 rows and 5004 columns.

`avhrrgoode(`*`region`*`,`*`filename`*`,scalefactor,latlim,lonlim,gsize,fnrows,fncols,resolution)` reads a data set with the spatial resolution specified in

meters. If omitted, the full resolution of 1000 meters is assumed. Data is also available at 8000 meter resolution.

`avhrrgoode(`*`region`*`,`*`filename`*`,scalefactor,latlim,lonlim,gsize,fnrows, fncols,resolution,`*`precision`*`)` reads a data set with the integer *`precision`* specified. If omitted, `'uint8'` is assumed. `'uint16'` is appropriate for some files. Check the data's README file 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, 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 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. This function 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.

Remarks

The AVHRR project and data sets are described in

```
http://daac.gsfc.nasa.gov/DATASET_DOCS/avhrr_dataset.html

http://daac.gsfc.nasa.gov/CAMPAIGN_DOCS/FTP_SITE/readmes/pal.html

http://edcwww.cr.usgs.gov/landdaac/1KM/1kmhomepage.html

http://edcwww.cr.usgs.gov/landdaac/glcc/glcc_na.html
```

Some sources for the data include

```
ftp://daac.gsfc.nasa.gov/data/avhrr/

ftp://edcftp.cr.usgs.gov/pub/data/glcc/
```

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

Examples

Read a 1 km Global Land Cover Classification (GLCC) file using the default parameters. Select the file `'gusgs1_2.img'` interactively. This file is available from <ftp://edcftp.cr.usgs.gov/pub/data/glcc/globe/gusgs1_2.img.gz>.

```
[latgrat,longrat,z] = avhrrgoode;
```

Read the same file at full resolution for just the island of Cyprus.

```
[latgrat,longrat,z] = avhrrgoode('g','gusgs1_2.img',1,...
       [34.2 35.9],[32 35]);
```

Read the GLCC urban areas file covering North America in the Goode projection for just the area of eastern Massachusetts. This file is available from <ftp://edcftp.cr.usgs.gov/pub/data/glcc/na/goode/naurbang.img.gz>.

```
[latgrat,longrat,z] = avhrrgoode('north america',...
       'naurban.img',1,[41.13 42.75],[-71.7 -69.8]);
```

Read the global data on the "Global Land 1-km AVHRR Data Set – Vegetation Index 6/21-30, 1992" CD-ROM (distributed by the Land Processes Distributed Active Archive Center, EROS Data Center, Sioux Falls, South Dakota, 57198, USA). Sample every 100th point for the entire globe, returning one lat and long for value. Provide the nonstandard number of rows and columns in the file.

```
[latgrat,longrat,z] = avhrrgoode('global','NDVI.IMG',...
       100,[-90 90],[-180 180],[],16347,40031);
```

Read the global 8 km resolution nondimensional vegetation index available from <ftp://daac.gsfc.nasa.gov/data/avhrr/global_8km/.1994_1997/1994/jun/avhrrpf.ndvi.1ntfgl.940621.gz>. Sample every 10th point for the entire globe, returning one lat and long for value. Provide the nonstandard number of rows, columns, and resolution in the file.

```
[latgrat,longrat,z] = avhrrgoode('global',...
       'avhrrpf.ndvi.1ntfgl.940621',10,[-90 90],...
       [-180 180],[],2168,5004,8000);
```

Read the global 8 km resolution data for AVHRR sensor channel 4 available from <ftp://daac.gsfc.nasa.gov/data/avhrr/global_8km/.1981_1985/1984/feb/avhrrpf.ch4.1ntfgl.840201.gz>. Read at the full 8 km resolution for the

island of Cyprus, returning one lat and long for value. Provide the nonstandard number of rows, columns, resolution, and integer precision in the file.

```
[latgrat,longrat,z] = avhrrgoode('global',...
       'avhrrpf.ch4.1ntfgl.840201',10,[34.2 35.9],...
       [32 35],[],2168,5004,8000,'uint16');
```

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.

Subsets of the land cover data are available in both the Goode and the uninterrupted Lambert azimuthal projections. Data can be read more quickly from the Lambert projection using `avhrrlambert`.

This function does not have the proper projection parameters to read the regional 8 km resolution data sets.

See Also

`avhrrlambert`	Read AVHRR data in the Lambert Azimuthal Equal-Area projection

avhrrlambert

Purpose

Read AVHRR data stored in the Lambert Azimuthal Projection

Syntax

`[latgrat,longrat,z] = avhrrlambert(`*`region`*`)` 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 projection. Data of this type includes the Global Land Cover Characteristics (GLCC). The region specifies the coverage of the file. Valid regions are `'g'` or `'global'`, `'af'` or `'africa'`, `'ap'` or `'australia/pacific'`, `'e'` or `'europe'`, `'a'` or `'asia'`, `'na'` or `'north america'`, `'sa'` or `'south america'`. Data is returned as a geolocated data grid with the graticule matrices in units of degrees.

`avhrrlambert(`*`region`*`,`*`filename`*`)` uses the provided *`filename`*.

`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. If omitted, 100 is assumed.

`avhrrlambert(`*`region`*`,`*`filename`*`,scalefactor,latlim,lonlim)` returns data for the specified region. The returned data will extend somewhat 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]`.

`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 data's README file 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 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 azimuthal projection at 1 km.

Remarks

The AVHRR project and data sets are described in

```
http://daac.gsfc.nasa.gov/DATASET_DOCS/avhrr_dataset.html
```

```
http://edcwww.cr.usgs.gov/landdaac/1KM/1kmhomepage.html
```

The Global Land Cover Characteristics data set is described in

```
http://edcwww.cr.usgs.gov/landdaac/glcc/glcc_na.html
```

The data can be obtained from

```
ftp://edcftp.cr.usgs.gov/pub/data/glcc/
```

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

Example

Read the Global Land Cover Characteristics (GLCC) file covering North America in the Lambert projection with the USGS classification scheme. Use the default parameters to read the entire file, taking just every 100th point. This file is available from <ftp://edcftp.cr.usgs.gov/pub/data/glcc/na/lambert/nausgs1_2l.img.gz>.

```
[latgrat,longrat,z] = avhrrlambert('north america',...
       'nausgs1_2l.img');
```

Read the same file at full resolution for just the area of eastern Massachusetts.

```
[latgrat,longrat,z] = avhrrlambert('north america',...
       'nausgs1_2l.img',1,[41.2 41.5],[-70.9 -70.4]);
```

See Also

`avhrrgoode` Read AVHRR data in the Goode projection

Purpose

Calculate eccentricity from semimajor and semiminor axes

Syntax

```
eccentricity = axes2ecc(semimajor,semiminor)
eccentricity = axes2ecc(axes)
```

Description

Eccentricity, the second element of the standard ellipsoid vector in the Mapping Toolbox, can be determined given both the semimajor and semiminor axes.

`eccentricity = axes2ecc(semimajor,semiminor)` returns the eccentricity associated with the input axes.

`eccentricity = axes2ecc(axes)` allows the axes inputs to be packed into a single two-column input of the form `[semimajor, semiminor]`.

Examples

Using the axes for the default GRS 80 Earth model,

```
ecc = axes2ecc(6378.1370,6356.7523)
ecc =
   0.08181921804834
```

This is the eccentricity returned by `almanac('earth','ellipsoid')`.

See Also

`almanac`	Planetary data
`ecc2n majaxis minaxis`	Related conversion functions

axesm

Purpose

Define map axes and set map properties

Syntax

`axesm`, with no input arguments, initiates the map axes graphical user interface, which can be used to set map axes properties. This is detailed on the `axesm, axesmui` reference page in Chapter 12, "GUI Reference."

`axesm(handle)` makes the map axes with the given handle the current map axes.

`axesm(`*`PropertyName`*`,PropertyValue,...)` creates map axes with the specified property values. The `MapProjection` property must be the first input pair.

`axesm(`*`ProjectionFile`*`,`*`PropertyName`*`,PropertyValue,...)` allows omission of the `MapProjection` property name. The first input must be the identifying string of an available projection.

Description

The `axesm` function creates a map axes object complete with a map data structure. Maps must be displayed in map axes. All standard MATLAB axes properties of map axes are controlled by the `axes` function, along with `set` and `get`. Map axes properties are defined on creation with `axesm` and can be queried and changed after creation of a map axes using `getm` and `setm`.

Axes Definition

Map axes are standard MATLAB axes with different default settings for some properties and with a map data structure in the `UserData` slot. The main differences are

- Axes properties `XGrid`, `YGrid`, `XTick`, `YTick` are set to `'off'`.
- The properties `XColor`, `YColor`, and `ZColor` are set to the background color.
- The `hold` mode is on.

The map structure is assigned to the `UserData` property. Do not overwrite this entry if map axes are desired. The map structure contains the map axes properties, which, in addition to the special standard axes settings described here, constitute a map axes. The map axes properties are described later.

Examples

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

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

In the preceding example, all properties not explicitly addressed in the call are set to either fixed or calculated defaults. The M-file `mercator.m` is a projection file, so the same result could have been achieved with the function

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

A projection file includes default data 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 frame limits is not displayed.

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

Object Properties

Properties That Control the Map

`MapProjection` *`projection_name`* `{no default}`

Map projection — Sets the projection, and hence all transformation calculations, for the map axes object. It is required in the creation of map axes. The projection name is a string corresponding to an M-file appropriate to the projection. 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 documentation. Some projections set their own defaults for other properties, such as parallels and trim limits.

`Zone` `ZoneSpec | {[] or 31N}`

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.

AngleUnits `{degrees} | radians | dms`

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 a more detailed description of the angle unit options, see the Mapping Toolbox User's Guide documentation.

Aspect `{normal} | 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 `scalar {0}`

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 `scalar {0}`

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 `scalar {[]} (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` `[semimajor_axis eccentricity]`

Planet ellipsoid definition — Sets the ellipsoid for calculating the projections of any displayed map objects. In the Mapping Toolbox, the ellipsoid is approximated by a spheroid. The default ellipsoid is a sphere with a radius of 1. This is represented as `[1 0]`. Any semimajor axis, in any distance units, can be entered; eccentricity lies between 0 and 1.

`MapLatLimit` `[south north] | [north south]`

Latitude limits of the displayed map — Sets the north and south latitude limits of the map data. This information is useful for two purposes. The default extents for the texture mapping functions `meshm`, `surfm`, `surfacem`, `surflm`, and `pcolorm` are set for the map axes; if the latitude limits match the actual data grid data limits, no graticule definitions are required when calling the above functions. Secondly, establishing map latitude limits sets the absolute limit on the extent of displayed meridians, regardless of the values of the meridian limits or the meridian exceptions. For nonazimuthal projections in the normal aspect, the map limits are truncated to the smaller of the map and frame limits. The default map latitude limits for most projections are at the poles, `[-90 90]`.

`MapLonLimit` `[east west] | [west east]`

Longitude limits of the displayed map — Sets the east and west longitude limits of the map data. This information is useful for two purposes. The default extents for the texture mapping functions `meshm`, `surfm`, `surfacem`, `surflm`, and `pcolorm` are set for the map axes; if the longitude limits match the actual data grid data limits, no graticule definitions are required when calling the above functions. Secondly, establishing map longitude limits sets the absolute limit on the extent of displayed parallels, regardless of the values of the parallel limits or the parallel exceptions. For nonazimuthal projections in the normal aspect, the map limits are truncated to the smaller of the map and frame limits. The default map longitude limits for most projections are at the International Date Line, `[-180 180]`.

`MapParallels` `[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 15°N and 75°N, which biases the projection towards 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 documentation for more information on standard parallels.

`Parallels` `0, 1, or 2 (read-only, projection-dependent)`

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 documentation for more information on standard parallels.

`Origin` `[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 0° 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 0°, 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 documentation.

`ScaleFactor` `scalar {1}`

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.

TrimLat `[south north] (read-only, projection-dependent)`

Latitude trimming for certain projections — This read-only property indicates the limits in latitude beyond which no plotting is attempted. This property is set by the projection and is usually used to avoid blowups to infinity. For example, the Mercator projection trims all data outside the range `[–86 86]`. `TrimLat` is `[–90 90]` for most projections.

TrimLon `[west east] (read-only, projection-dependent)`

Longitude trimming for certain projections — This read-only property indicates the limits in longitude beyond which no plotting is attempted. This property is set by the projection and is usually used to avoid blowups to infinity. It is less commonly used than the latitude trimming and is `[–180 180]` for most projections.

Properties That Control the Frame

Frame `on | {off}`

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 `scalar plotting point density {100}`

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 ColorSpec | {[0 0 0]}

Color of the displayed frame edge — Specifies the color used for the displayed frame. You can specify a color using a vector of RGB values or one of the MATLAB predefined names. By default, the frame edge is displayed in black ([0 0 0]).

FFaceColor ColorSpec | {none}

Color of the displayed frame face — Specifies the color used for the displayed frame face. You can specify a color using a vector of RGB values or one of the MATLAB predefined names. By default, the frame face is 'none', meaning no face color is filled in. Another useful color is 'cyan' ([0 1 1]), which looks like water.

FLatLimit [south north] | [north south]

Latitude limits of the base projection frame — Sets the north and south latitude limits of the map frame. Latitudes refer to the base projection, and are [–90 90] by default for most projections. The frame latitude limits determine where data is trimmed in a north-south sense. Data lying outside the latitude limits is not displayed. These limits also determine the latitude positions of the frame for its own display.

For nonazimuthal projections in the normal aspect, the frame limits are truncated to the smaller of the map and frame limits. Frame limits for nonazimuthal projections are specified in Cartesian coordinates with respect to the map origin specified in the Origin property. Frame latitude limits for azimuthal projections are specified by -Inf and a radius in polar coordinates with respect to the map origin specified in the Origin property.

FLineWidth scalar {2}

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 [east west] | [west east]

Longitude limits of the base projection frame — Sets the east and west longitude limits of the map frame. Longitudes refer to the base projection, and are [–180 180] by default for most projections. The frame longitude limits determine where data is trimmed in an east-west sense. Data lying outside the longitude limits is not displayed. These limits also determine the longitude positions of the frame for its own display.

For nonazimuthal projections in the normal aspect, the frame limits are truncated to the smaller of the map and frame limits. Frame limits for nonazimuthal projections are specified in Cartesian coordinates with respect to the map origin specified in the `Origin` property. Frame longitude limits are ignored for azimuthal projections.

Properties That Control the Grid

`Grid` `on | {off}`

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` `scalar z-axis value {Inf}`

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` `ColorSpec | {[0 0 0]}`

Color of the displayed grid — Specifies the color used for the displayed grid. You can specify a color using a vector of RGB values or one of the MATLAB predefined names. By default, the map grid is displayed in black (`[0 0 0]`).

`GLineStyle` `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` `scalar {0.5}`

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` `vector of longitudes {[]}`

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 scalar plotting point density {100}

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 [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 scalar interval or specific vector {30°}

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 0° longitude and repeating every interval in both directions, which by default is 30°. Alternatively, you can enter a vector of longitudes, in which case a meridian is displayed for each element of the vector.

PLineException vector of latitudes {[]}

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 scalar plotting point density {100}

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 `[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 `scalar interval or specific vector {15°}`

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 0° latitude and repeating every interval in both directions, which by default is 15°. 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 `{normal} | 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 `ColorSpec | {black}`

Text color for all grid labels — Sets the color of all displayed grid labels. `ColorSpec` is a three-element vector specifying an RGB triple or a predefined MATLAB color string.

FontName `courier | {helvetica} | 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 `scalar in units specified in FontUnits {9}`

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 {points} | 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 bold | {normal}

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

LabelFormat {compass} | 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 on | {off}

Label Rotation — Determines whether the meridian and parallel labels are displayed without rotation (the default), or rotated to align to the graticule.

LabelUnits {degrees} | radians | dms | dm

Specify units for labels — Grid labels are displayed in the desired angular units as specified in `LabelUnits`, regardless of the map axes `AngleUnits`. The default value, however, does match the `AngleUnits` property.

MeridianLabel on | {off}

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

MLabelLocation scalar interval or vector of longitudes

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` `{north} | south | equator | scalar latitude`

Specify parallel for meridian label placement — Specifies the latitude location of the displayed meridian labels. If a 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 0° is used.

`MLabelRound` `integer scalar {0}`

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` `on | {off}`

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

`PLabelLocation` `scalar interval or vector of latitudes`

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` `east | {west} | prime | scalar longitude`

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` `integer scalar {0}`

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

`gcm`	Get current map structure
`getm`	Get map object properties
`setm`	Set map object properties

Purpose

Make scale the same between axes

Syntax

`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`.

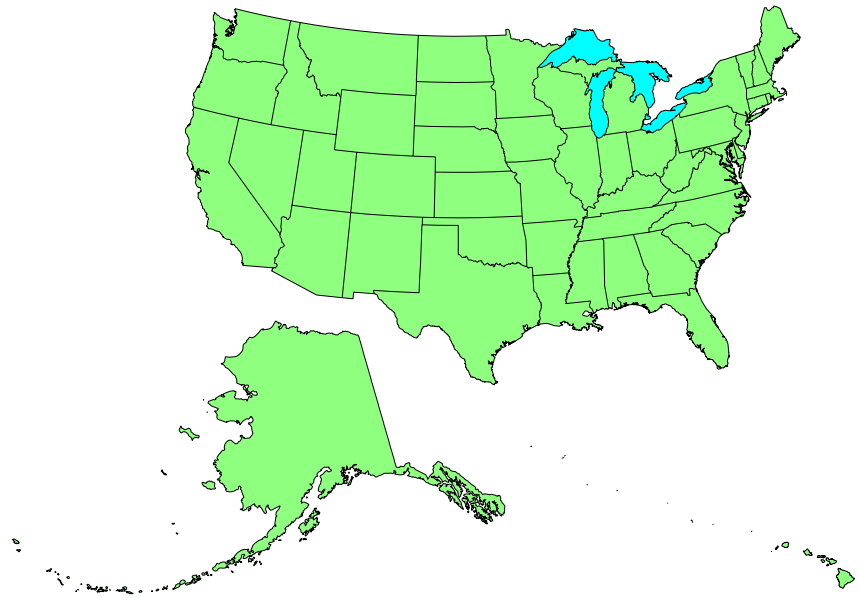
Examples

Display the conterminous United States, Alaska, and Hawaii in separate axes.

```
hconus = usamap('conus'); tightmap

halaska= axes; usamap('alaskaonly'); tightmap;
framem off; mlabel off; plabel off

hhawaii = axes; usamap('hawaiionly'); tightmap;
framem off; mlabel off; plabel off

% Arrange the axes as desired
set(hconus,'Position',[0.1 0.25 0.85 0.6])
set(halaska,'Position',[0.019531 -0.020833 0.2 0.2])
set(hhawaii,'Position',[0.5 0 .2 .2])
```

Make the axes have the same scale.

```
axesscale(hconus) % resize alaska and hawaii axes
hidem([halaska hhawaii])
```



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.

Remarks

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

See Also

`paperscale` Figure paper size for a given map scale

Purpose

Compute azimuth between two points on the globe

Syntax

`az = azimuth(pt1,pt2)` calculates the great circle azimuths from `pt1` to `pt2`. These two-column matrices should be of the form `[latitude longitude]`.

`az = azimuth(lat1,lon1,lat2,lon2)` performs the same calculation for two pairs of latitude and longitude matrices.

`az = azimuth(pt1,pt2,ellipsoid)` specifies the elliptical definition of the Earth to be used with the two-element `ellipsoid` vector. The default ellipsoid model is a unit sphere, which is sufficient for most applications.

`az = azimuth(pt1,pt2,`*`units`*`)` specifies the standard angle unit string. The default value is `'degrees'`.

`az = azimuth(track,pt1,...)` specifies whether great circle azimuths or rhumb line azimuths are desired. Great circle azimuths, the default, are indicated with the standard *`track`* string `'gc'`. Rhumb line azimuths are indicated with the standard *`track`* string `'rh'`.

Background

Azimuths are the bearings, or directions, between pairs of points. Azimuths are measured as angles, clockwise from true north. The North Pole has an azimuth of 0° from every other point on the globe.

Azimuth can be calculated in two ways. For great circles, the azimuth is the angle made between true north and the great circle passing through the two points *at the first point*. For rhumb lines, the azimuth is the *constant* angle made between true north and the entire rhumb line passing through the two points. For more information on this distinction, see the Mapping Toolbox User's Guide documentation.

Examples

Consider two points on the same parallel, for example, (10°N,10°E) and (10°N,40°E). The azimuth between these two points depends upon the *`track`* string selected. Using the `pt1,pt2` notation, the two cases result in

```
az = azimuth('gc',[10,10],[10,40])
az =
   87.3360

az = azimuth('rh',[10,10],[10,40])
az =
    90
```

The great circle path begins on an azimuth north of east to take the shortest route to the second point; the rhumb line proceeds along the parallel, on a constant due east heading.

Rhumb lines and great circles coincide along meridians and the equator. Consider two points on the same meridian, say (10°N,10°E) and (40°N,10°E), this time using the `lat1,lon1,lat2,lon2` notation:

```
az = azimuth(10,10,40,10) % great circle sense
az =
     0

az = azimuth('rh',10,10,40,10)
az =
     0
```

The azimuths are the same because the paths coincide.

See Also

`distance`	Distance between points
`elevation`	Elevation angle to a point
`reckon`	New point from an azimuth

`track`
`track1`
`track2`
	Tracing paths on the globe

Purpose

Compute buffer zones for vector data

Syntax

`[latb,lonb] = bufferm(lat,lon,dist,`*`direction`*`)` computes the buffer zone around a polygon. A buffer zone for a closed polygon is defined as the locus of points that are a certain distance in or out of the polygon. A buffer zone for an open polygon is the locus of points a certain distance out from the polygon. The polygon is specified as vectors of latitude and longitude in units of degrees. The distance is a scalar specified in degrees of arc along the surface. Valid direction strings are `'in'` and `'out'`. The result is returned as `NaN`-clipped vectors in units of degrees.

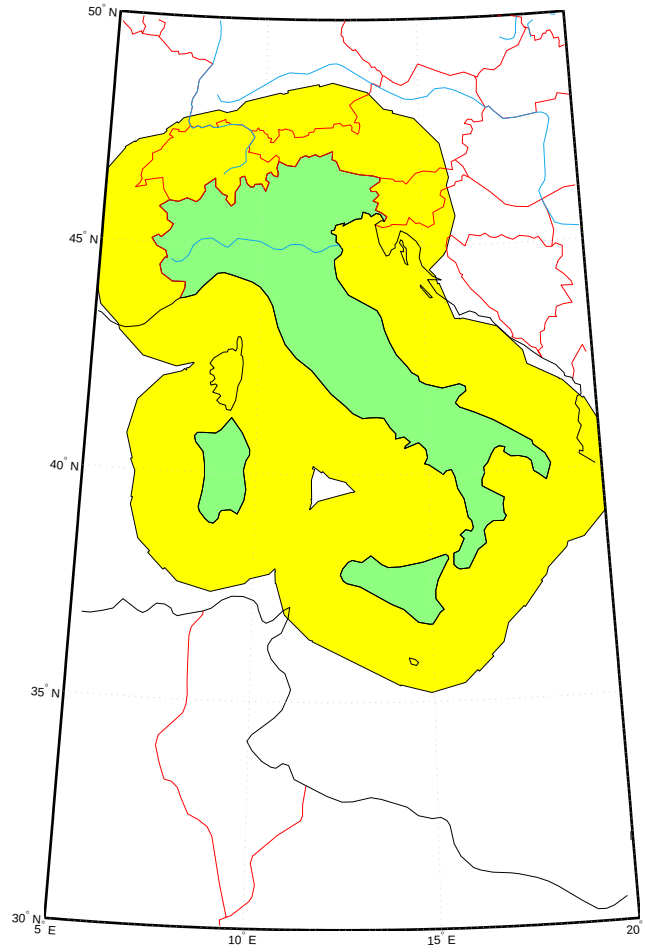
`[latb,lonb] = bufferm(lat,lon,dist,`*`direction`*`,npts)` controls the number of points used to construct circles about the vertices of the polygon. A larger number of points produces smoother buffers, but requires more time. If omitted, 13 points per circle are used.

`[latb,lonb] = bufferm(lat,lon,dist,`*`direction`*`,npts,`*`outputformat`*`)` controls the format of the returned buffer zones. `outputformat 'vector'` returns `NaN`-clipped vectors. `outputformat 'cutvector'` returns `NaN`-clipped vectors with cuts connecting holes to the exterior of the polygon. `outputformat 'cell'` returns cell arrays in which each element of the cell array is a separate polygon. Each polygon can consist of an outer contour followed by holes separated with `NaN`s.

Examples

Extract the coastline for Italy from the low-resolution database and reduce the polygon. Construct a buffer polygon 1.5 degrees of arc along the earth's surface out from the Italian coast. Display the result on a base map of the Italian peninsula along with the original polygon.

```
[lat,lon]=extractm(worldlo('POpatch'),'italy');
[lat,lon]=reducem(lat,lon);

[latb,lonb] = bufferm(lat,lon,1.5,'out')

worldmap('lo','italy','lineonly')
hp = patchesm(latb,lonb,'y');             zdatam(hp,-2)
set(hp(2),'facecolor','w');               zdatam(hp(2),1) % hole
hp2 = geoshow(updategeostruct(worldlo('POpatch'),'italy'), ...
'FaceColor','green')
```



See Also

`polybool` Perform polygon Boolean operations

Purpose

Camera position in geographic coordinates

Syntax

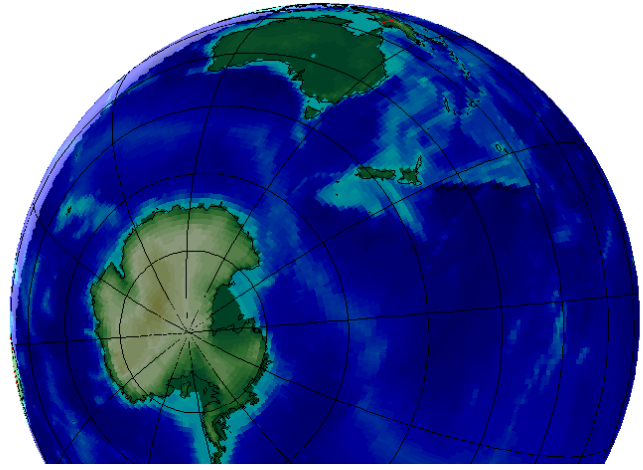
`camposm(lat,long,alt)` sets the axes `CameraPosition` property of the current map axes to the position specified in geographic coordinates. The inputs `lat` and `long` are assumed to be in the angle units of the current map axes.

`[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:

```
figure; axesm('globe','galt',0); gridm('glinestyle','-')

load topo; meshm(topo,topolegend,size(topo)); demcmap(topo)
displaym(worldlo('POline'))
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)
```



See Also

`camtargm`	Camera target from geographic coordinates
`camupm`	`CameraUpVector` from geographic coordinates

`campos`	Camera position
`camva`	Camera view angle

Purpose

Camera target in geographic coordinates

Syntax

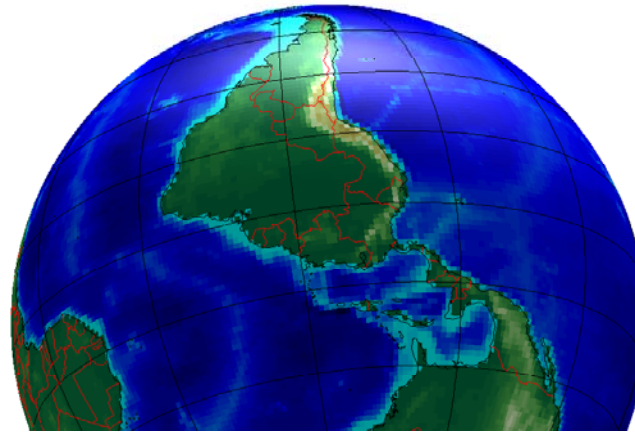
`camtargm(lat,long,alt)` sets the axes `CameraTarget` property of the current map axes to the position specified in geographic coordinates. The inputs `lat` and `long` are assumed to be in the angle units of the current map axes.

`[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:

```
figure; axesm('globe','galt',0); gridm('glinestyle','-')

load topo; meshm(topo,topolegend,size(topo)); demcmap(topo)
displaym(worldlo('POline'))
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)
```



See Also

`camposm`	Camera position from geographic coordinates
`camupm`	`CameraUpVector` from geographic coordinates

`camtarget`	Camera target
`camva`	Camera view angle

Purpose

Camera up vector in geographic coordinates

Syntax

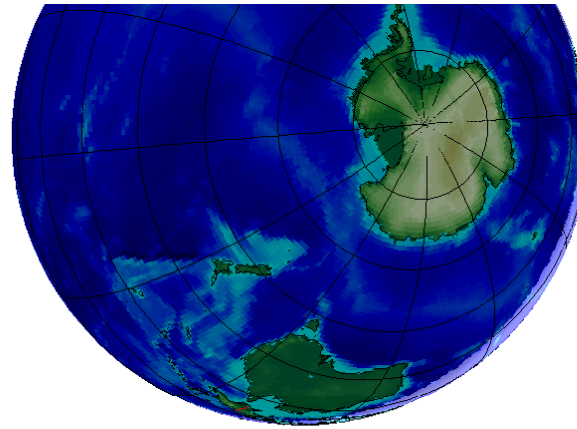
`camupm(lat,long)` sets the axes `CameraUpVector` property of the current map axes to the position specified in geographic coordinates. The inputs `lat` and `long` are assumed to be in the angle units of the current map axes.

`[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:

```
figure; axesm('globe','galt',0); gridm('glinestyle','-')

load topo; meshm(topo,topolegend,size(topo)); demcmap(topo)
displaym(worldlo('POline'))
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)
```



See Also

`camtargm`	Camera target from geographic coordinates
`camposm`	Camera position from geographic coordinates
`camup`	Camera up vector
`camva`	Camera view angle

Purpose

Return Greenwich coordinates for displayed map objects

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.

`[lat,lon,alt] = cart2grn` returns the latitude, longitude, and altitude data in Greenwich 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`	Get current map structure
`mfwdtran`	Forward map projection transformation
`minvtran`	Inverse map projection transformation
`project`	Project existing graphics objects onto a map axes

changem

Purpose

Substitute values in a data array

Syntax

`B = changem(A,newval)`, for scalar `newval`, replaces all zero-valued entries in `A` with `newval`.

`B = changem(A,newval,oldval)` replaces all occurrences of `newval(k)` in `A` with `oldval(k)`. `newval` and `oldval` must match in size.

Description

`mapout = changem(map,newcode,oldcode)` returns a data grid `mapout` identical to the input data grid, except that each element of `map` 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 0's:

```
B = changem(A,[0 0],[9 8])
B =
     0     1     6
     3     5     7
     4     0     2
```

Purpose

Find the intersections of two circles in Cartesian space

Syntax

`[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`

clabelm

Purpose

Label map contours

Syntax

`h1 = clabelm(c,h)` rotates the labels and inserts them in line with the contour lines. The handles of the labels can be returned in `h1`.

`h1 = clabelm(c,h,v)` creates inline labels only for those levels specified in the vector `v`.

`h1 = clabelm(c,h,'manual')` places contour labels at locations you select with a mouse. You press the left mouse button (the only mouse button on a single-button mouse), or the space bar to label a contour at the closest location beneath the center of the cursor. Press the **Return** key while the cursor is within the figure window to terminate labeling. The labels are inserted in line with the contour lines.

`h1 = clabelm(c)`, `h1 = clabelm(c,v)`, and `h1 = clabelm(c,'manual')` operate as above, except that instead of rotating the labels and placing them in line with the contours, the labels are upright, and a + indicates the contour line the label is annotating.

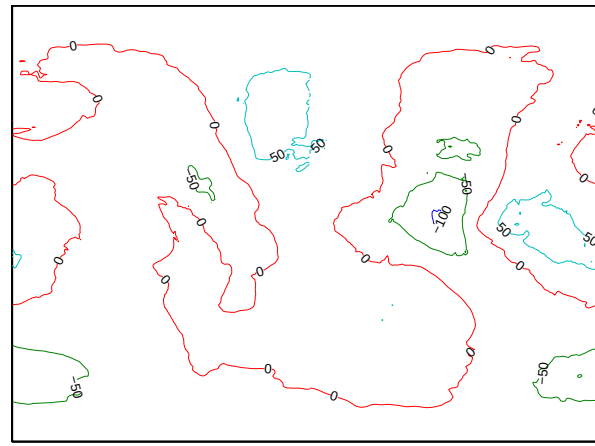
Description

The `clabelm` function adds height labels to a two-dimensional contour plot. By default, `clabelm` labels all displayed contours and randomly selects label positions.

`c` is the contour matrix as described on the `contourm` reference page of this guide; `h` is the vector of handles for the displayed contours.

Examples

```
axesm miller
framem
tightmap
[c,h] = contourm(geoid,geoidlegend,-100:50:80);
clabelm(c,h)
```



See Also

`clegendm`	Display a legend for a contour plot of map data
`contourm`	Project a contour plot of map data
`contour3m`	Project a 3-D contour plot of map data
`clabel`	Contour plot elevation labels (see the online MATLAB Function Reference documentation)

clegendm

Purpose

Add legend labels to a map contour display

Syntax

```
clegendm(cs,h)
clegendm(cs,h,pos)
clegendm(...,unitstr)
clegendm(...,str)
```

Description

The `clegendm` function displays a legend for a displayed contour map.

`clegendm(cs,h)` displays a legend for the contour map defined by the two-column contour definition matrix, `cs`, and the handle(s) `h`. Both of these inputs are produced as the outputs of either `contourm` or `contour3m`.

`clegendm(cs,h,pos)` allows you to specify the position of the legend in the display. The input `pos` can be any of the following integers, with the indicated result:

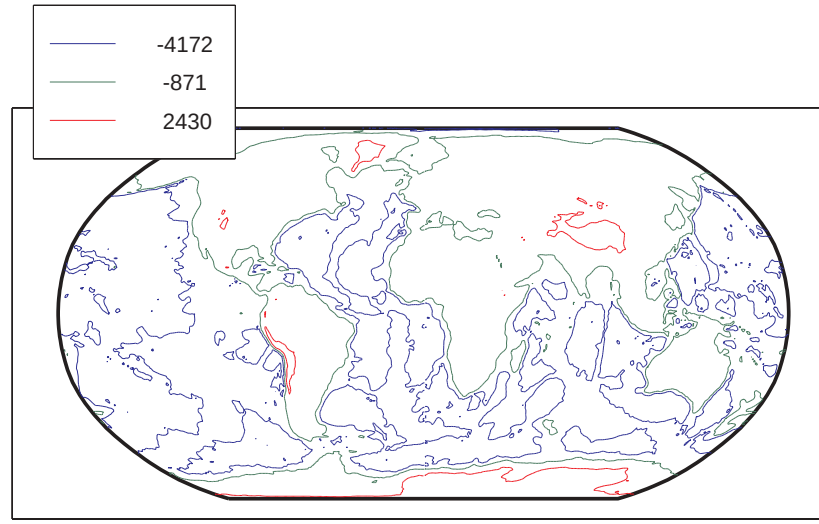
0	Automatic placement (this is the default)
1	Upper right corner
2	Upper left corner
3	Lower left corner
4	Lower right corner
–1	To the right of the plot

`clegendm(...,unitstr)` appends the character string `unitstr` to each entry in the legend.

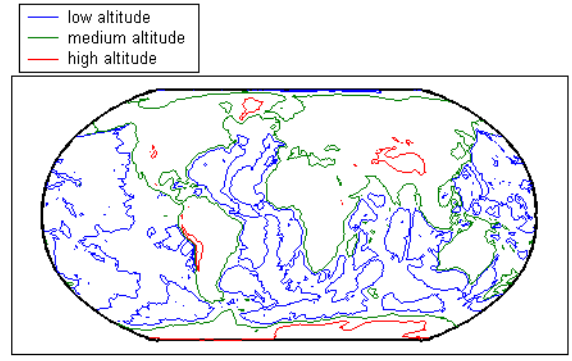
`clegendm(...,str)` uses the strings specified in cell array `str` to label the legend. The cell array must have same number of entries as `h`.

Examples

```
load topo
axesm robinson; framem
[cs,h] = contourm(topo,topolegend,3);
clegendm(cs,h,2)
```




```
% Example showing legend string usage
% Load topographic data measured in meters
load topo;
axesm robinson; framem
[cs,h] = contorm(topo,topopolegend,3);
% Create Legend with user specified string
str = {'low altitude','medium altitude','high altitude'}
clegendm(cs,h,2,str);
```



See Also

`clabelm`	Label a contour plot of map data
`contourm`	Project a contour plot of map data
`contour3m`	Project a 3-D contour plot of map data
`contourc`	Low-level contour plot computation (see the online MATLAB Function Reference documentation)

Purpose

Clip map data at the `-pi` to `pi` border of a display

Syntax

`[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.

Description

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 clips 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.

Allowable object strings are:

- `surface` for clipping graticules
- `light` for clipping lights
- `line` for clipping lines
- `patch` for clipping patches
- `text` for clipping text object location points
- `point` for clipping point data
- `none` to skip all clipping operations

See Also

`trimdata`, `undoclip`, `undotrim`

clma

Purpose

Clear current map axes

Syntax

`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 in the axes `UserData` property, thereby retaining the map axes.

`clma purge` clears all displayed map objects and clears the axes `UserData` slot, effectively changing the map axes to standard axes. This is equivalent to `cla reset`.

See Also

`cla`	Clear current axes (see the online MATLAB Function Reference documentation)
`clmo`	Clear specified graphics objects
`handlem`	Return handles of displayed map objects
`hidem`	Hide specified graphics objects
`namem`	Determine names of valid graphics objects
`showm`	Show specified graphics objects
`tagm`	Assign name to graphics object tag property

Purpose

Clear specified graphics object

Syntax

`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 input string. This can be any string 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`	Clear current map
`handlem`	Return handles of displayed map objects
`hidem`	Hide specified graphics objects
`namem`	Determine names of valid graphics objects
`showm`	Show specified graphics objects
`tagm`	Assign name to graphics object tag property

cmapui

Purpose

A GUI to generate colormaps by interactively picking colors

Syntax

```
cmap5 = cmapui
cmap32 = cmapui(32)
```

Description

`cmapui` is a graphical user interface to create a colormap. The default size is five colors.

You select color slots in the colormap by clicking in the colorbar on the right side of the panel. The current color slot is outlined in black. The color components for that color in HSV space are shown by the position of the dot in the color wheel and of the red bar in the value slider. To change the color, use the mouse to drag the dot and/or the red bar. To close the GUI and return the matrix of colors as RGB components, click the **Accept** button. Clicking **Cancel** closes the GUI and returns an empty matrix.

`cmapui` is a modal GUI. There is no access to the MATLAB command line while `cmapui` is active.

Examples

```
cmap = cmapui(20);
cmap = cmapui(colorcube(10));
```

See Also

`colormapeditor` MATLAB colormap editor

Purpose

Determine combinations of a set of values

Syntax

`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.

Description

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
```

Remarks

This is a recursive function.

Purpose

Project three-dimensional comet plot on map axes

Syntax

`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.

Description

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 coast
z = (1:length(lat))'/3000;
axesm miller
framem; gridm;
setm(gca,'galtitude',max(z)+.5)
view(3)
comet3m(lat,long,z,0.01)
```

See Also

`comet3`	3-D comet-like trajectories (see the online MATLAB Function Reference documentation)
`cometm`	Two-dimensional comet plot projected on map axes

cometm

Purpose

Project two-dimensional comet plot on map axes

Syntax

`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.

Description

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 coast
axesm miller
framem
cometm(lat,long,0.01)
```

See Also

`comet`	Comet-like trajectories (see the online MATLAB Function Reference documentation)
`comet3m`	Three-dimensional comet plot projected on map axes

Purpose

Project a contour plot of map data in 3-D space

Syntax

`contour3m(lat,lon,map)` produces a 3-D contour plot of map data projected onto the current map axes. The input latitude and longitude vectors can be the size of `map` (as in a geolocated data grid), or can specify the corresponding row and column dimensions for the map.

`contour3m(map,refvec)` produces a contour plot of map data in a regular data grid.

`contour3m(lat,lon,map,`*`LineSpec`*`)` uses any valid *`LineSpec`* string to draw the contour lines.

`contour3m(lat,lon,map,`*`PropertyName`*`,PropertyValue,...)` uses the line properties specified to draw the contours.

`contour3m(lat,lon,map,n,...)` draws `n` contour levels, where `n` is a scalar.

`contour3m(lat,lon,map,v,...)` draws contours at the levels specified by the input vector `v`.

`contour3m(map,refvec,...)` takes any of the optional arguments described above.

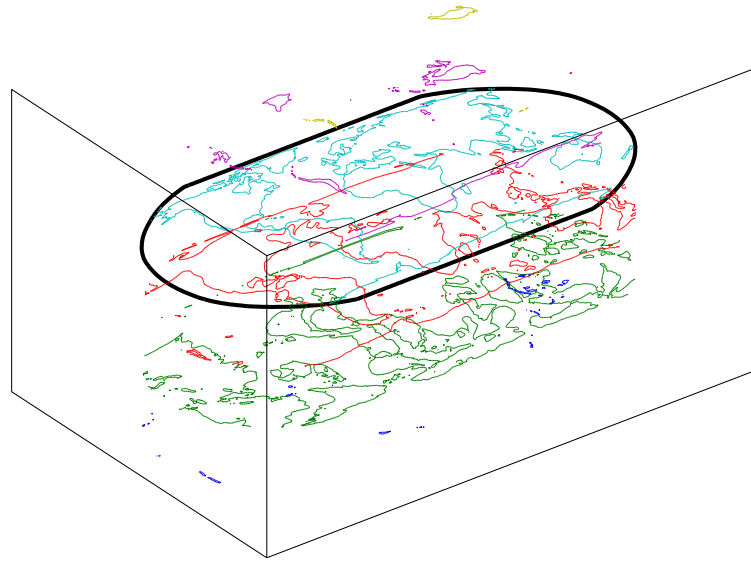
`c = contour3m(...)` returns a standard contour matrix, with the first row representing longitude data and the second row representing latitude data.

`[c,h] = contour3m(...)` returns the contour matrix and the handles to the contour lines drawn.

`contour3m`, without any inputs, activates a GUI to project contour lines onto the current map axes.

Examples

```
load topo
axesm robinson; framem; view(3)
contour3m(topo,topolegend)
set(gca,'DataAspectRatio',[1 1 3000])
```



See Also

`clabelm`	Label a contour plot of map data
`clegendm`	Display a legend for a contour plot of map data
`contourc`	Contour computation (see the online MATLAB Function Reference documentation)
`contourm`	Project a contour plot of map data

Purpose

Project a contour plot of map data

Syntax

`contourm(lat,lon,map)` produces a contour plot of map data projected onto the current map axes. The input latitude and longitude vectors can be the size of `map` (as in a geolocated data grid), or can specify the corresponding row and column dimensions for the map.

`contourm(map,refvec)` produces a contour plot of map data in a regular data grid.

`contourm(lat,lon,map,`*`LineSpec`*`)` uses any valid *`LineSpec`* string to draw the contour lines.

`contourm(lat,lon,map,`*`PropertyName`*`,PropertyValue,...)` uses the line properties specified to draw the contours.

`contourm(lat,lon,map,n,...)` draws `n` contour levels, where `n` is a scalar.

`contourm(lat,lon,map,v,...)` draws contours at the levels specified by the input vector `v`.

`contourm(map,refvec,...)` takes any of the optional arguments described above.

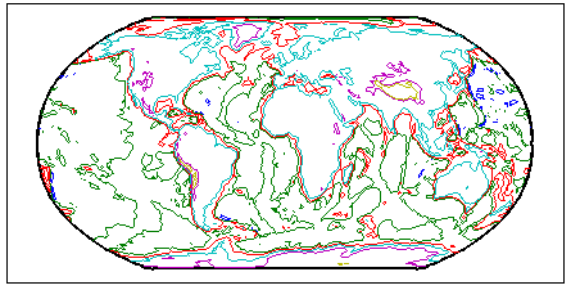
`c = contourm(...)` returns a standard contour matrix, with the first row representing longitude data and the second row representing latitude data.

`[c,h] = contourm(...)` returns the contour matrix and the handles to the contour lines drawn.

`contourm`, without any inputs, activates a GUI to project contour lines onto the current map axes.

Examples

```
load topo
axesm robinson; framem
contourm(topo,topolegend)
```



See Also

`clabelm`	Label a contour plot of map data
`clegendm`	Display a legend for a contour plot of map data
`contourc`	Contour computation (see the online MATLAB Function Reference documentation)
`contour3m`	Project a 3-D contour plot of map data

Purpose

Contour colormap and colorbar for surfaces

Syntax

`contourcmap(cdelta,cmap)` creates a contour colormap for the current axes. A contour colormap is a colormap with color changes aligned to the color data. If `cdelta` is a scalar, contours are generated at multiples of `cdelta`. If `cdelta` is a vector of evenly spaced values, contours are generated at those values. The string input *`cmap`* is the name of the colormap function used in the surface. Valid entries for *`cmap`* include `'pink'`, `'hsv'`, `'jet'`, or any similar colormap function.

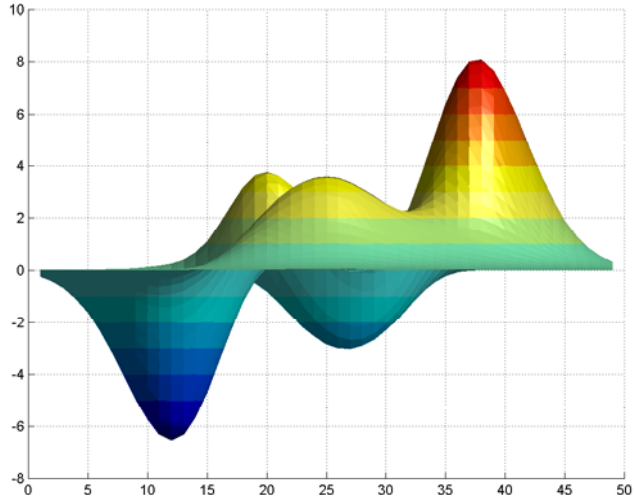
`contourcmap(cdelta,cmap,property,value,...)` allows you to add a colorbar and control the colorbar's properties. You turn the colorbar on with the property-value pair `'Colorbar'` and `'on'`. The location of the colorbar is controlled by the `'Location'` property. Valid entries for `Location` are `'vertical'` (the default) or `'horizontal'`. Properties `'TitleString'`, `'XLabelString'`, `'YLabelString'` and `'ZLabelString'` set the respective strings. Property `'ColorAlignment'` controls whether the colorbar labels are centered on the color bands or the color breaks. Valid values for `ColorAlignment` are `'center'` or `'ends'`. Property `'SourceObject'` controls which object is used to determine the color limits for the colormap. The `SourceObject` value is the handle of a currently displayed object. If omitted, `gca` is used. Other valid property-value pairs are any properties and values that can be applied to the title and labels of the colorbar axes.

`hcb = contourcmap(...)` returns a handle to the colorbar.

Example

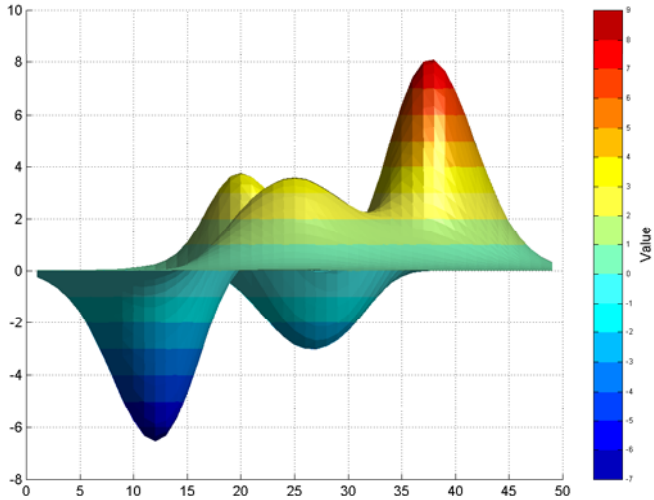
Create a colormap and set color limits to make the color changes occur at multiples of one for the peaks surface.

```
surf(peaks)
contourcmap(1,'jet')
view(90,0)
shading interp
camlight
```



Add a colorbar, controlling the labels and font properties.

```
h = contourcmap(1,'jet','colorbar','on', ...
'YLabelString','Value','FontSize',6)
```



Delete the colorbar.

```
delete(h)
```

See Also

`contourfm`	Filled contour map
`lcolorbar`	Labeled color bar
`demcmap`	Digital elevation data color map

contourfm

Purpose

Project a filled contour map

Syntax

`contourfm(lat,lon,map)` produces a contour plot of map data projected onto the current map axes. The input latitude and longitude vectors can be the size of `map` (as in a geolocated data grid), or can specify the corresponding row and column dimensions for the map.

`contourfm(map,refvec)` produces a contour plot of map data in a regular data grid.

`contourfm(lat,lon,map,`*`LineSpec`*`)` uses any valid *`LineSpec`* string to draw the contour lines.

`contourfm(lat,lon,map,`*`PropertyName`*`,PropertyValue,...)` uses the line properties specified to draw the contours.

`contourfm(lat,lon,map,n,...)` draws `n` contour levels, where `n` is a scalar.

`contourfm(lat,lon,map,v,...)` draws contours at the levels specified by the input vector `v`.

`contourfm(map,refvec,...)` takes any of the optional arguments described above.

`c = contourfm(...)` returns a standard contour matrix, with the first row representing longitude data and the second row representing latitude data.

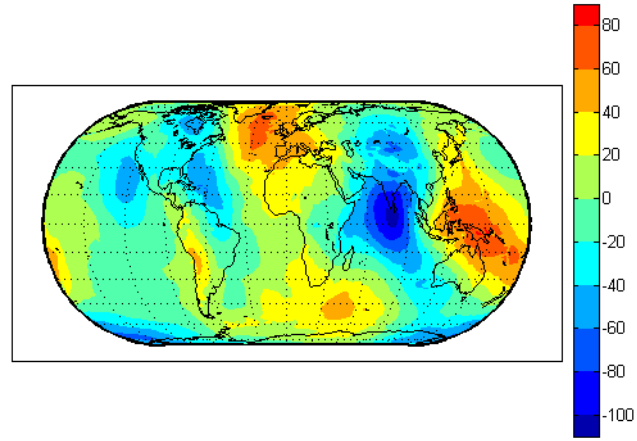
`[c,h] = contourfm(...)` returns the contour matrix and the handles to the contour lines drawn.

`contourfm`, without any inputs, activates a GUI to project contour lines onto the current map axes.

Examples

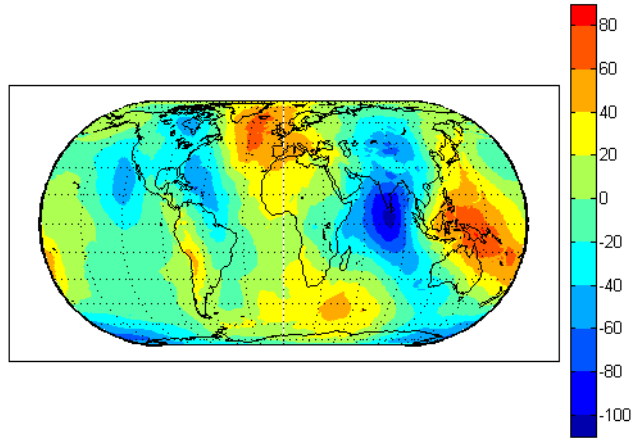
Plot the Earth's geoid with filled contours. The data is in meters.

```
load geoid
figure
axesm eckert4
framem;gridm
load coast
plotm(lat,long,'k')

caxis([-120 100]);colormap(jet(11));colorbar
contourfm(geoid,geoidlegend,-120:20:100);
```



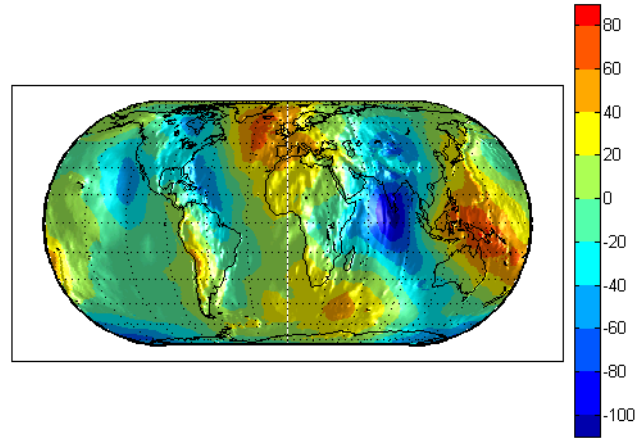
You can reproduce the filled contour display by using a surface instead of the patches created by `contourfm`.

```
figure
axesm eckert4
framem;gridm
load coast
plotm(lat,long,'k')

meshm(geoid,geoidlegend,size(geoid),'Facecolor','interp')
contourcmap(20,'jet');colorbar
```



Surfaces also allow use of lighting to bring out the smaller variations in the data.

```
clmo surface
meshm(geoid,geoidlegend,size(geoid),geoid,'Facecolor','interp')
light;lighting phong; material(0.6*[ 1 1 1])
set(gca,'dataaspectratio',[ 1 1 200])
gridm reset
zdatam(handlem('line'),max(geoid(:)))
```



Limitations

`contourfm` might not fill properly with azimuthal projections.

Remarks

The patches are drawn at a range of z-levels < 0 to ensure proper display. Contours are displayed with no edge colors. To combine contour fill with contour lines, use both `contourfm` and `contourm`.

In most circumstances, contour plots made with surfaces are preferable to the filled patches created by `contourfm`. Surfaces are rendered more quickly and take less time to project and reproject. The use of surfaces also allows surface lighting to create shaded 3-D maps.

See Also

`contourm`	Project a contour plot of data onto the current map axes
`contour3m`	Project a 3-D contour plot of data onto the current map axes
`clabelm`	Add contour labels to a map contour plot

country2mtx

Purpose

Construct a data grid for a country in the `worldlo` database

Syntax

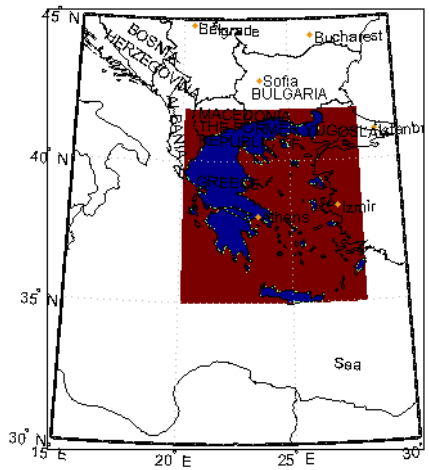
`[map,refvec] = country2mtx(countryname,scale)` constructs a regular data grid of a country in the `worldlo` `POpatch` database. The scale factor represents the number of matrix entries per degree of latitude and longitude (e.g., 10 entries per degree, 100 entries per degree). The scale input must be scalar. The returned matrix has values of 0 in the interior of the country, 1 on the border, and 2 in the exterior.

`[map,refvec] = country2mtx(countryname,scalelatlim,lonlim)` uses the two-element vector latitude and longitude limits to define the extent of the map. If omitted, the limits are computed automatically.

`[map,refvec] = country2mtx(countryname,map1,refvec1)` uses the limits of the provided map.

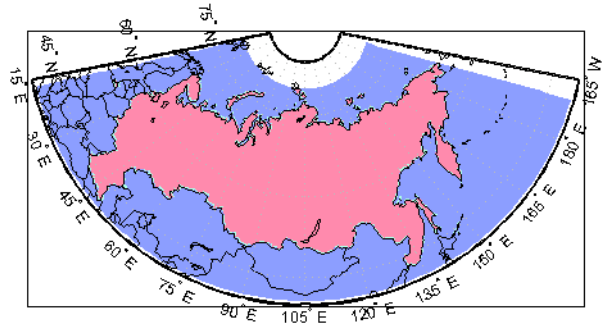
Examples

```
[map,refvec] = country2mtx('greece',50);
worldmap('greece');
meshm(map,refvec)
```




```
load russia
[map2,refvec2] = country2mtx('russia',map,refvec);
figure
```

```
worldmap('russia');
meshm(map2,refvec2)
polcmap
```



See Also

`vec2mtx`	Regular data grid from vector data
`ltln2val`	Return map code value associated with positions
`imbedm`	Encode data points into a regular data grid
`encodem`	Fill in indexed maps with specified seeds
`interpm`	Interpolate vector data to a specified data separation

convertlat

Purpose

Convert between geodetic and auxiliary latitudes

Syntax

`latout = convertlat(ellipsoid, latin, from, to, units)` converts latitude values in `latin` from type `from` to type `to`. `ellipsoid` is a 1-by-2 ellipsoid vector of the form `[semimajoraxis eccentricity]`. (The `almanac` function offers a set of built-in ellipsoids covering most widely available map data.)

Description

`latin` is an array of input latitude values. `from` and `to` are each one of the latitude type strings listed below (or unambiguous abbreviations). `latin` has the angle units specified by `units`: either `'degrees'`, `'radians'`, or unambiguous abbreviations. The output array, `latout`, has the same size and units as `latin`.

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.

Latitude Type	Description
`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.

To properly project rectified latitudes, the radius must also be scaled to ensure the equal meridional distance property. This is accomplished by `rsphere`.

Example

```
% 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 = almanac('earth','ellipsoid','m','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');
```

See Also

almanac, rsphere

Purpose

Determine cross fix positions from bearings and ranges

Syntax

`[newlat,newlon] = crossfix(lat,long,az)` returns the intersection points of all pairs of great circles passing through the points given by the column vectors `lat` and `long` 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,long,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,long,az_range,case,drlat,drlong)` resolves the ambiguities when there is more than one intersection between two objects. The scalar-valued `drlat` and `drlong` provide the location of an estimated (dead reckoned) position. The outputs `newlat` and `newlong` 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,long,az,`*`units`*`)`, `[newlat,newlon] = crossfix(lat,long,az_range,case,`*`units`*`)`, `[newlat,newlon] = crossfix(lat,long,az_range,drlat,drlong,`*`units`*`)`, and `[newlat,newlon] = crossfix(lat,long,az_range,case,drlat,drlong,`*`units`*`)` allow the specification of the angle units to be used for all angles and ranges, where *`units`* is any valid angle units string. The default value of *`units`* is `'degrees'`.

`mat = crossfix(...)` returns the output in a two- or four-column matrix `mat`.

Description

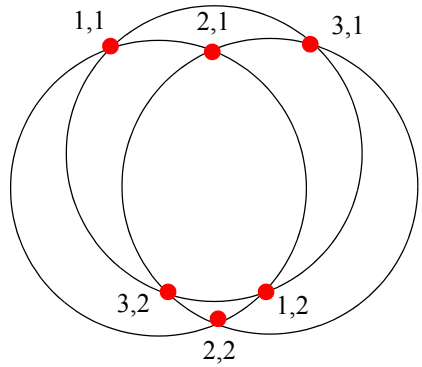
This function calculates the points of intersection between a set of objects taken in pairs. Given great circle azimuths and/or ranges from input points, the locations of the possible intersections are returned. This is different from the navigational function `navfix` in that `crossfix` uses great circle measurement, while `navfix` uses rhumb line azimuths and nautical mile distances.

Examples

Where do the small circles defined as all points 8° in distance from the points (0°,0°), (5°N,5°E), and (0°,10°E)" intersect?

```
[newlat,newlong] = crossfix([0 5 0]',[0 5 10]',[8 8 8]',[0 0 0]')
newlat =
    7.5594   -2.5744
    6.2529   -6.2529
    7.5594   -2.5744
newlong =
   -2.6260    7.5770
    5.0000    5.0000
   12.6260    2.4230
```

Here is an illustration to show why there are six intersections:



If a dead reckoning position is provided, say (0°,5°E), then one from each pair is returned (the closest):

```
[newlat,newlong] = crossfix([0 5 0]',[0 5 10]',...
                            [8 8 8]',[0 0 0]',0,5)
newlat =
   -2.5744
    6.2529
   -2.5744
newlong =
    7.5770
    5.0000
    2.4230
```

See Also

`gcxgc`
`gcxsc`
`scxsc`
`rhxrh`
`polyxpoly`
	Other intersection functions

`navfix`	Mercator-based navigational fixing

daspectm

Purpose

Control vertical exaggeration in a map display

Syntax

`daspectm(`*`zunits`*`)` sets the figure `'DataAspectRatio'` property 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 string *`zunits`* specifies the units of the elevation data, and can be any string recognized by `distdim`.

`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,`*`gunits`*`)` also specifies the units in which the geographic position and direction are given. If omitted, `'degrees'` is assumed.

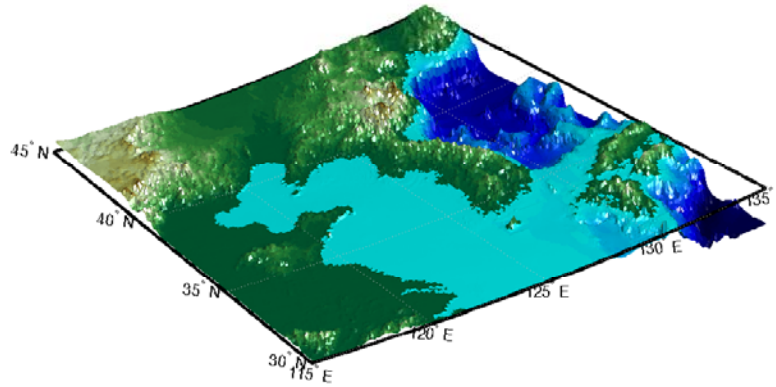
`daspectm(`*`zunits`*`,vfac,lat,long,az,`*`gunits`*`,radius)` uses the last input to determine the radius of the sphere. If `radius` is a string, then it is evaluated as an `almanac` body to determine the spherical radius. If numerical, it is the radius of the desired sphere in `zunits`. If omitted, the default radius of the Earth is used.

Examples

Show the elevation map of the Korean peninsula with a vertical exaggeration factor of 30:

```
load korea
[latlim,lonlim] = limitm(map,refvec);

worldmap(latlim,lonlim,'none')
meshm(map,refvec,size(map),map)
demcmap(map)

view(3)
daspectm('m',30)
tightmap
```

```
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`	Data aspect ratio
`paperscale`	Figure paper size for a given map scale

dcwdata

Purpose

Read selected data from the Digital Chart of the World

Syntax

`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 string. 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 Mapping Toolbox geographic data 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).

Remarks

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 strings describing the objects. 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.

Examples

On the Macintosh,

```
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'});
```

See Also

`vmap0data`	Read selected data from the Vector Map Level 0
`dcwgaz`	Search for entries in the Digital Chart of the World gazette
`dcwread`	Read a Digital Chart of the World file

`dcwrhead`	Read a Digital Chart of the World file header
`displaym`	Project data contained in a map structure
`extractm`	Extract vector data from a map structure
`mlayers`	GUI for manipulating map layers

References

The format and the history of the DCW are described in references `[1]`, `[2]`, and `[3]` located in the Bibliography at the end of this chapter.

dcwgaz

Purpose

Search for entries in the Digital Chart of the World gazette

Syntax

`dcwgaz(`*`library`*`,`*`object`*`)` searches the DCW library for items beginning with the *`object`* string. 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`* string 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 geographic data 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 string.

Remarks

The search is not case sensitive. Items that match are those that begin with the *`object`* string. Spaces are significant.

Examples

On the Macintosh,

```
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
```

See Also

`dcwdata`	Read selected data from the Digital Chart of the World
`dcwread`	Read a Digital Chart of the World file
`dcwrhead`	Read a Digital Chart of the World file header
`displaym`	Project data contained in a map structure
`mlayers`	GUI for manipulating map layers

dcwread

Purpose

Read a Digital Chart of the World file

Syntax

`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 filename.

`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 a string describing 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.

Remarks

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.

Examples

The following examples use the Macintosh directory system and file separators for the pathname:

```
s = dcwread('NOAMER:DCW:NOAMER:','GRT')
s =
                    ID: 1
             DATA_TYPE: 'GEO'
                 UNITS: 'O14'
             ELLIPSOID: 'WGS 84'
       ELLIPSOID_DETAIL: 'A=6378137,B=6356752 Meters'
         VERT_DATUM_REF: 'MEAN SEA LEVEL'
        VERT_DATUM_CODE: 'O15'
            SOUND_DATUM: 'MEAN SEA LEVEL'
       SOUND_DATUM_CODE: 'O15'
         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
```

See Also

`dcwdata`	Read selected data from the Digital Chart of the World
`dcwgaz`	Search for entries in the Digital Chart of the World gazette
`dcwrhead`	Read a Digital Chart of the World file header

Purpose

Read the header of a Digital Chart of the World file

Syntax

`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 filename.

`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.

`str = dcwrhead(...)` returns a string containing the DCW header.

Background

The Digital Chart of the World (DCW) uses header strings in most files to document the contents and format of that file. This function reads the header string, displays a formatted version in the command window, or returns it as a string.

Remarks

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.

Examples

The following example uses the Macintosh file separators and pathname:

```
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,-,-,:;
```

See Also

`dcwdata`	Read selected data from the Digital Chart of the World
`dcwgaz`	Search for entries in the Digital Chart of the World gazette
`dcwread`	Read a Digital Chart of the World file

Purpose

Initialize a default map projection structure

Syntax

`mstruct = defaultm` creates an empty map projection structure.

`mstruct = defaultm(`*`projection`*`)` initializes the map structure for the specified map projection. *`projection`* is any valid projection string, such as `'sinusoid'`.

`mstruct = defaultm(mstruct)` sets appropriate defaults based on existing parameter values in the map structure `mstruct`.

`[mstruct,msg] = defaultm(...)` returns the string `msg`, indicating any error encountered.

Description

The map projection structure contains all the information needed to project and display geographic data. It normally resides in the `UserData` property of a map axes, but it can also be used directly to project data without display.

Examples

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: [-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: 0.50000000000000
    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: 9
          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
```

See Also

`axesm`	Define map axes and set map properties
`gcm`	Get current map data structure
`mfwdtran`	Map forward transformation
`minvtran`	Map inverse transformation
`setm`	Set and modify map properties

Purpose

Convert angle units from degrees to dms or dm format

Syntax

`anglout = deg2dms(anglin)` converts angles input in degrees to the equivalent measure in the degrees-minutes-seconds (*dms*) format.

`angleout = deg2dm(anglin)` converts angles input in degrees to the equivalent measure in the degrees-minutes (*dm*) format. This is the dms format, properly rounded to just degrees and minutes.

Example

```
deg2dms(23.561)
ans =
       2333.40

deg2dm(23.561)
ans =
       2334.00
```

See Also

`angledim`	Convert angle units
`dms2mat`	Convert from dms to separated matrix components
`deg2rad` `dms2rad`	Other direct angle conversion functions
`mat2dms`	Convert from separated matrices input to dms

deg2km, deg2nm, deg2sm

Purpose

Convert distance from degrees to kilometers, nautical miles, or statute miles

Syntax

`distout = deg2km(distin)` converts the input distance given in degrees to kilometers.

`distout = deg2nm(distin)` and `distout = deg2sm(distin)` work identically, except that the output units are nautical miles and statute miles, respectively.

`distout = deg2km(distin,radius)` specifies the radius of the sphere to use, since a degree of arc length covers less distance, for example, on Mars than it would on the Earth. You can enter the radius as a number in kilometers, as a call to the `almanac` function (e.g., `almanac('mars','radius','km')`), again in the appropriate units, or you can pass in a string planet name (e.g., `'mars'`), and the function makes the appropriate call to the `almanac` function. The radius of the Earth is the default.

For `distout = deg2nm(distin,radius)` and `distout = deg2sm(distin,radius)`, make sure your input radius is in the appropriate units, or just use the planet name string.

Examples

A degree of arc length is about 60 nautical miles:

```
deg2nm(1)
ans =
   60.0405
```

This is not true on Mercury, of course:

```
deg2nm(1,'mercury')
ans =
   22.9852
```

See Also

`deg2rad`	Convert degrees to radians
`distdim`	Convert distance units
`nm2km` `sm2deg`	Other direct distance conversion functions

Purpose

Convert angle (or distance) units from degrees to radians

Syntax

`anglout = deg2rad(anglin)` converts angles input in degrees to the equivalent measure in radians.

Remarks

This is both an angle conversion function and a distance conversion function, since arc length can be a measure of distance in either radians or degrees, provided that the radius is known.

Example

Show that there are 2¼ radians in a full circle:

```
2*pi - deg2rad(360)
ans =
     0
```

See Also

`angledim`	Convert angle units
`deg2dms` `dms2rad`	Other direct angle conversion functions
`distdim`	Convert distance units
`nm2km` `sm2deg`	Other direct distance conversion functions
`rad2deg`	Convert from radians to degrees

demcmap

Purpose

Create colormaps for digital elevation maps

Syntax

`demcmap(map)` creates and assigns a colormap for elevation data. The colormap has the number of land and sea colors in the same proportions as the maximum elevations and depths in the data grid. With no output arguments, the colormap is applied to the current figure and the color axis is set so that the interface between the land and sea is in the right place.

`demcmap(map,ncolors)` makes a colormap with a length of `ncolors`. The default value is 64.

`demcmap(map,ncolors,cmapsea,cmapland)` allows the default colors for sea and land to be replaced. The colors in the created colormap are interpolated from the RGB color matrix inputs, which can be of any length. You can retain default colors for either land or sea by providing an empty matrix in place of the color matrices. You can specify the current figure colormap by entering the string `'window'` in place of either RGB matrix.

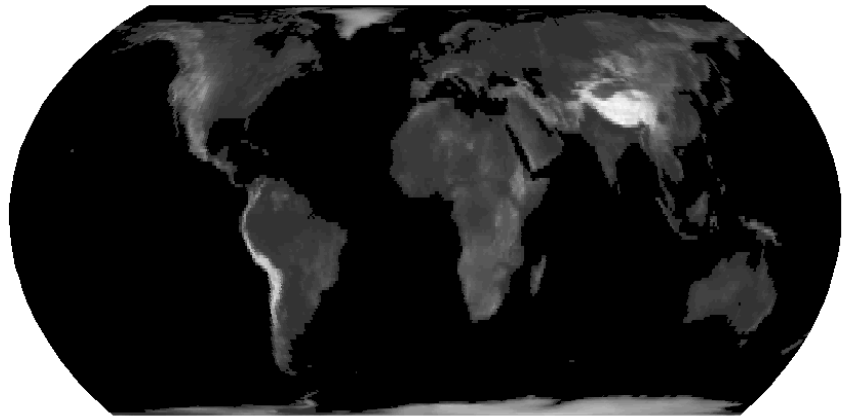
`demcmap(`*`color`*`,map,spec)` uses the *`color`* string to define a colormap. If the string is set to `'size'`, `spec` is the length of the colormap. If it is set to `'inc'`, `spec` is the size of the altitude range assigned to each color. If omitted, *`color`* is `'size'` by default.

`demcmap(`*`color`*`,map,spec,cmapsea,cmapland)` allows for both coloring options along with specified colors.

Examples

Display the world topographical map using grayscale colors:

```
load topo
axesm hatano
meshm(topo,topolegend)
demcmap(topo,64,[0 0 0],[.2 .2 .2; 1 1 1])
```



See Also

`caxis`	Color axis scaling (see the online MATLAB Function Reference documentation)
`colormap`	Set and get the current colormap (see the online MATLAB Function Reference documentation)
`meshlsrm`	Project 3-D lighted shaded relief of regular data grid
`meshm`	Display a regular data grid warped to a projected graticule
`surflsrm`	Project 3-D lighted shaded relief of geolocated data grid
`surfm`	Display a data grid warped to a projected graticule

demdataui

Purpose

Digital elevation map data user interface

Activation

```
demdataui
```

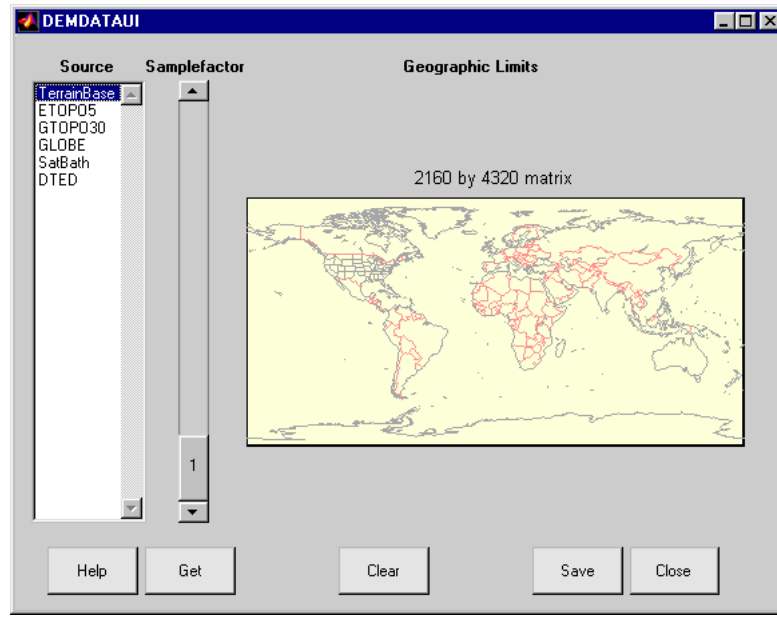
Description

`demdataui` is a graphical user interface to extract digital elevation map data from a number of external data files.

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, and DTED data. See the links under "See also" for more information on these data sets. `demdataui` looks for these external data files on the MATLAB path and, for some operating systems, on CD-ROM disks.

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: new_etopo5.bil or etopo5.northern.bat and etopo5.southern.bat files. tbase: tbase.bin file. satbath: topo_6.2.img file. gtopo30: a directory that contains subdirectories with the data files. For example, demdataui would detect gtopo30 data if a directory on the path contained the directories E060S10 and E100S10, each of which holds the uncompressed data files. globedem: a directory that contains data files and in the subdirectory "/esri/hdr" the "*.hdr" header files. dted: a directory that has a subdirectory named DTED. The contents of the DTED directory are more subdirectories organized by longitude and, below that, the DTED data files for each latitude tile. See the help for functions with the data source names for more on the data attributes and internet locations.

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 results are stored as a geographic data structure. 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,'ldem3d')). 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.

See Also

`etopo5`, `tbase`, `gtopo30`, `globedem`, `dted`, `satbath`, `vmap0ui`

departure

Purpose

Compute departure between longitudes at specified latitudes

Syntax

`dist = departure(long1,long2,lat)` returns the departure between two longitudes at a given latitude in degrees. Departure is dimensionless; the shorter of the two directions is taken from the first longitude to the second. The distance is given in degrees of arc length.

`dist = departure(long1,long2,lat,`*`units`*`)` specifies the valid angle units string to apply to the latitude, longitudes, and output distance.

`dist = departure(long1,long2,lat,ellipsoid)` specifies the elliptical definition of the Earth to be used with the two-element `ellipsoid` vector. The default ellipsoid model is a unit sphere, which is sufficient for most applications. When a ellipsoid model is input, the resulting distance is given in terms of the distance units in the `ellipsoid` vector, regardless of the angle units used.

Description

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.

Examples

On a spherical Earth, the departure is proportional to the cosine of the latitude:

```
distance = departure(0,10,0)
distance =
    10
distance = departure(0,10,60)
distance =
     5
```

When an ellipsoid is used, the result is more complicated. The distance at 60° is not exactly twice the 0° value:

```
distance = departure(0,10,0,almanac('earth','ellipsoid','nm'))
distance =
  601.0772
distance = departure(0,10,60,almanac('earth','ellipsoid','nm'))
distance =
  299.7819
```

See Also

`distance`	Distance between two points
`stdm`	Standard deviation for geographic data

displaym

Purpose

Project data in a geographic data structure

Syntax

`displaym(mstruct)` projects the data contained in the input structure onto the current axes. The current axes must have a valid map definition. The input mstruct must be a valid Mapping Toolbox geographic data structure.

`displaym(mstruct,'object')` displays vector data from entries in the Mapping Toolbox geographic data structure whose tags begin with the `'object'` string. The output vectors use NaNs to separate the individual entries in the map structure. Matches of the tag string must be vector data (lines and patches) to be included in the output. The search is not case sensitive.

`displaym(mstruct,objects)`, where `objects` is a character array or a cell array or strings, allows more than one object to be the basis for the search. Character array objects have trailing spaces stripped before matching.

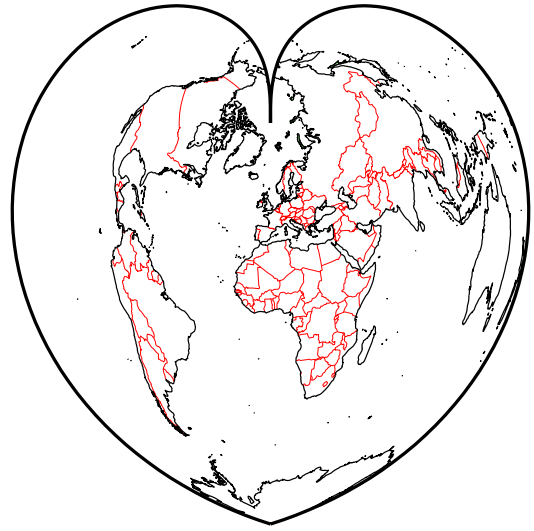
`[lat,lon] = displaym(mstruct,objects,'exact')` requires an exact match to extract data.

`h = displaym(mstruct,...)` returns the handles to the objects projected.

Examples

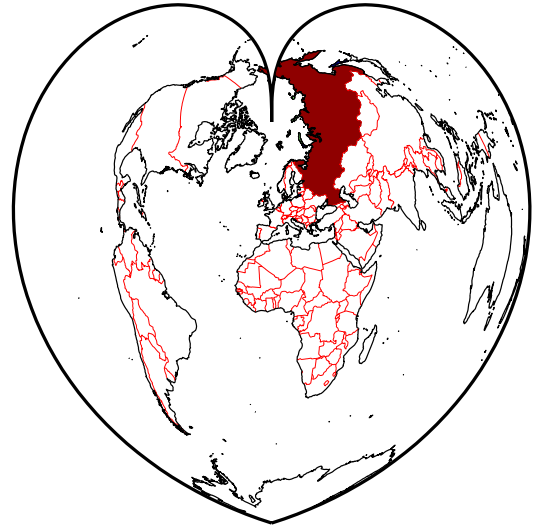
Display political and coastlines contained in structure format:

```
axesm werner; framem
displaym(POline)
```



Highlight Russia by displaying a patch map of the country:

```
displaym(POpatch,'russia')
```



Remarks

A Mapping Toolbox geographic data structure is a MATLAB structure that can contain line, patch, text, regular data grid, geolocated data grid, and light objects.

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.

See Also

`extractm`	Extract vector data from geographic data structures
`mlayers`	GUI for manipulating layers of a geographic data structure

Purpose

Convert distance values to strings

Syntax

`str = dist2str(distin)` converts the input vector of distances, `distin`, to a string matrix.

`str = dist2str(distin,`*`format`*`)` uses the *`format`* string to specify the notation to be used for the string matrix. The default, `'none'`, results in simple numerical representation (no indicator for positive distances, minus signs for negative distances); `'pm'` (for *plus-minus*) prefixes a + for positive distances.

`str = dist2str(distin,`*`format`*`,`*`units`*`)` uses the input *`units`* to define the units in which the input distances are supplied. *`units`* is any valid distance string (`'kilometers'` is the default). *`units`* also determines the unit symbol to suffix to the strings.

`str = dist2str(distin,`*`format`*`,`*`units`*`,digits)` determines how many digits to display. `digits` is the power of 10 representing the last place of significance in the resulting output. For example, if `digits` = 2, the *hundreds* slot is its last significant figure. In general, the 10^{digits} slot is the last significant figure, rounded appropriately depending upon the value in the $10^{\text{digits}-1}$ slot. `digits` is -2 by default.

Description

The purpose of this function is to make distance-valued variables into strings suitable for map display.

Examples

Create a vector of values and convert to strings:

```
d = [-3.7 2.95 87]
str = dist2str(d,'none','km')
str =
-3.70 km
 2.95 km
87.00 km
```

Now change the units, add +'s to positive values, and truncate to the tenths (10^{-1}) slot:

```
str = dist2str(d,'pm','sm',-1)
str =
 -3.7 mi
 +3.0 mi
```

```
   +87.0 mi
```

See Also

`angl2str`	Convert angle values to strings
`distdim`	Convert distance units
`time2str`	Convert time values to strings

Purpose

Compute distance between two points on the globe

Syntax

```
dist = distance(pt1,pt2)
dist = distance(pt1,pt2,ellipsoid)
dist = distance(pt1,pt2,units)
dist = distance(pt1,pt2,ellipsoid,units)

dist = distance(track,pt1,...)

dist = distance(lat1,lon1,lat2,lon2)
dist = distance(lat1,lon1,lat2,lon2,ellipsoid)
dist = distance(lat1,lon1,lat2,lon2,units)
dist = distance(lat1,lon1,lat2,lon2,ellipsoid,units)

dist = distance(track,lat1,...)
```

Background

Distance between two points can be calculated in two ways. For great circles, 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. For more information on this distinction, see the Mapping Toolbox User's Guide documentation.

Description

`dist = distance(pt1,pt2)` calculates the great circle distance from `pt1` to `pt2`. These two-column matrices should be of the form `[latitude longitude]`. The resulting distance is returned in terms of angle units of arc length (degrees by default).

`dist = distance(lat1,lon1,lat2,lon2)` performs the same calculation for two pairs of latitude and longitude matrices.

`dist = distance(pt1,pt2,ellipsoid)` specifies the elliptical definition of the Earth to be used with the two-element `ellipsoid` vector. The default ellipsoid model is a unit sphere, which is sufficient for most applications. When an ellipsoid is input, the resulting distance is given in terms of the distance units used in the `ellipsoid` vector.

`dist = distance(pt1,pt2,units)` specifies the standard angle unit string. The default value is `'degrees'`. These units are also the distance units of the result (e.g., degrees of arc length) unless a `ellipsoid` vector is specified.

`dist = distance(track,pt1,...)` specifies whether great circle distances or rhumb line distances are desired. Great circle distances, the default, are indicated with the standard *`track`* string `'gc'`. Rhumb line distances are indicated with the standard *`track`* string `'rh'`.

Examples

Imagine a trip 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`* string selected. Using the `pt1,pt2` notation, the two cases result in

```
dist = distance('gc',[37,-76],[37,-9])
dist =
   52.3094

dist = distance('rh',[37,-76],[37,-9])
dist =
    53.5086
```

The difference between these two tracks is 1.992 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 course of 090°, due east, while in order to follow the great circle path, the course must be changed continuously.

When a great circle and rhumb line coincide, the distances are the same. Using two points on the same meridian, this time in the `lat1,lon1,lat2,lon2` notation,

```
dist = distance(37,-76,67,-76) % great circle sense
dist =
     30.0000

dist = distance('rh',37,-76,67,-76)
dist =
     30.0000
```

The distances are the same, about 1800 nautical miles (there are about 60 nautical miles in a degree of arc length).

See Also

`azimuth`	Azimuth between two points on the globe
`elevation`	Elevation between two points on the globe
`distdim`	Convert distance units
`reckon`	New point with an azimuth and distance

`track`
`track1`
`track2`
`trackg`
	Trace paths on the globe

distortcalc

Purpose

Calculate distortion parameters for a map projection

Syntax

`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
```

Remarks

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`	Contours of constant distortion on a map
`tissot`	Graphic depiction of distortion characteristics

distdim

Purpose

Convert distances between different units

Syntax

`distout = distdim(distin,from,to)` returns the value of the input distance `distin`, which is in units specified by the valid distance units string *`from`*, in the desired units given by the valid distance units string *`to`*. Valid distance units strings are

```
'kilometers' or 'km' for kilometers
'meters' or 'm' for meters
'nauticalmiles' or 'nm' for nautical miles
'statutemiles' or 'sm' for statute miles
'feet' or 'ft' for feet
'degrees' or 'deg' for degrees (arc length)
'radians' or 'rad' for radians (arc length)
```

`distout = distdim(distin,from,to,radius)` specifies the radius of a sphere to use when one of *`from`* or *`to`* is a unit string associated with arc length (radians or degrees). A degree of arc length covers more kilometers, for example, on Jupiter than it would on the Earth. You can enter the radius as a number (the radius of the sphere in the non-arc-length units), as a call to the `almanac` function (e.g., `almanac('jupiter','radius','units')`), again in the appropriate units, or as a string planet name (e.g., `'mars'`), and the function makes the appropriate call to the `almanac` function. The radius of the Earth is the default.

Remarks

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 necessarily 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 6371 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. In the Mapping Toolbox, 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.

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
```

See Also

`almanac`	Planetary data
`angledim`	Convert angle units

`deg2km`
`sm2nm`
`nm2rad`
	Direct distance conversion functions

`dist2str`	Convert distances to display strings
`distance`	Distance between points
`timedim`	Convert time units

dms2deg, dms2rad

Purpose

Convert angle units from dms format to degrees or radians

Syntax

`anglout = dms2deg(anglin)` converts angles input in degrees-minutes-seconds (*dms*) format to the equivalent measure in decimal degrees.

`anglout = dms2rad(anglin)` converts angles input in degrees-minutes-seconds (*dms*) format to the equivalent measure in radians.

Remarks

The inputs can be in degrees-minutes (*dm*) format, because numerically they look like dms format in which seconds are always zero.

Example

```
dms2deg(430.00)
ans =
          4.50
```

See Also

`angledim`	Convert angle units
`deg2rad` `dms2rad`	Other direct angle conversion functions
`dms2dm`	Round dms format angles to dm format
`dms2mat`	Convert from dms to separated matrix components
`mat2dms`	Convert from separated matrices to dms format

Purpose

Convert the elements of dms format to distinct matrix elements

Syntax

`[d,m,s] = dms2mat(anglin)` takes angles input in dms inputs and splits their components into three outputs, one each for degrees, minutes, and seconds.

`[d,m,s] = dms2mat(anglin,n)` specifies the power of 10, `n`, to which the resulting seconds output should be rounded (that is, if a result is 12.567 seconds, and `n` = -2, the resulting seconds output would be 12.57). The default value of `n` is -5.

`matout = dms2mat(anglin,n)` returns a three-column matrix, `matout`, in which the columns represent degrees, minutes, and seconds, respectively. In this case, `anglin` must be a vector.

Examples

```
anglin = [12547.34; 54323.17];

[d,m,s] = dms2mat(anglin)
d =
   125
   543
m =
    47
    23
s =
    34
    17

matout = dms2mat(anglin)
matout =
   125    47    34
   543    23    17
```

See Also

`mat2dms` Convert from separated matrices to dms format

dms2dm

Purpose

Round from dms format to dm format

Syntax

`anglout = dms2deg(anglin)` rounds angles input in degrees-minutes-seconds (*dms*) format to the appropriate value in degrees-minutes (*dm*) format. This special handling is needed because there are 60, and not 100, seconds in a minute.

Example

Round 4°45'29” and 4°45'31" to dm format:

```
dms2dm(445.29)
ans =
        445.00

dms2dm(445.31)
ans =
        446.00
```

See Also

`angledim`	Convert angle units
`deg2rad` `dms2rad`	Other direct angle conversion functions
`dms2dm`	Round dms format angles to dm format
`dms2mat`	Convert from dms to separated matrix components
`mat2dms`	Convert from separated matrices input to dms

Purpose

Compute dead reckoning positions for a track

Syntax

`[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 the “Navigation” section of the Mapping Toolbox User’s Guide documentation for further information.

Examples

Assume that a navigator gets a fix at noon, 1200Z, which is (10.3°N, 34.67°W). He's in a hurry to make a 1330Z rendezvous with another ship at (9.9°N, 34.5°W), so he plans on a speed of 25 knots. After the rendezvous, both ships head for (0°, 37°W). 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`	Find courses and distances between waypoints
`navfix`	Mercator-based navigational fixing
`track`	Connect navigational waypoints with track segments

Purpose

Heading to correct for wind or current drift

Syntax

`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.

Example

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` Wind or current from heading, course, and speeds

driftvel

Purpose

Wind or current from heading, course, and speeds

Syntax

`[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`).

Example

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` Heading to correct for wind or current drift

Purpose

Read U. S. Department of Defense Digital Terrain Elevation Data (DTED) data

Syntax

`[datagrid,refvec] = dted` returns all of the elevation data in a DTED file as a regular data grid with elevations in meters. The file is selected interactively. This function reads the DTED elevation files, which generally have filenames ending in `.dtN`, where N is 0,1,2,3,... .

`[datagrid,refvec] = dted(`*`filename`*`)` returns all the elevation data in the specified DTED file. The file must be found on the MATLAB path. If not found, you can select the file interactively.

`[datagrid,refvec] = dted(`*`filename`*`,scalefactor)` returns data from the specified DTED file, downsampled by the `scalefactor`. If omitted, a `scalefactor` of 1 is assumed.

`[datagrid,refvec] = dted(`*`filename`*`,scalefactor,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.

`[datagrid,refvec] = dted(dirname,scalefactor,latlim,lonlim)` reads and concatenates data from multiple files within a DTED CD-ROM or directory structure. The `dirname` input is a string with the name of a directory containing the DTED directory. Within the DTED directory are subdirectories for each degree of longitude, each of which contains files for each degree of latitude. For DTED CD-ROMs, `dirname` is the device name of the CD-ROM drive.

`[datagrid,refvec,UHL,DSI,ACC] = dted(...)` also returns the DTED User Header Label (UHL), Data Set Identification (DSI) and Accuracy (ACC) metadata records as structures. Documentation describing the meaning of these records is available online at `http://www.nima.mil/publications/specs/printed/DTED/dted1-2.doc`.

Background

The U. S. Department of Defense, through the National Imagery and Mapping 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. Higher resolution data is restricted to the U.S. Department of Defense and its contractors.

Examples

```
[datagrid,refvec] = dted('n38.dt0');

[datagrid,refvec,UHL,DSI,ACC] = dted('n38.dt0',1,[38.5 38.8],...
       [-76.8 -76.6]);

[datagrid,refvec,UHL,DSI,ACC] = dted('f:',1,[38.5 38.8],...
       [-76.8 -76.6]);
```

Limitations

At higher latitudes the files have fewer longitude records. In those cases a warning is issued, and the coarser spacing is used in both directions.

Remarks

This function reads the DTED elevation files in the format used for files delivered on CD-ROM. The filenames generally end in `.dtN`, where N is 0,1,2,3, etc.

The 1 kilometer data is available online at `http://www.nima.mil/geospatial/products/DTED/dted.html`.

The higher resolution data is available to the U.S. Department of Defense and its contractors from the National Imagery and Mapping Agency (NIMA).

DTED0 files have 120-by-120 points. DTED1 files have 1201-by-1201. The edges of adjacent tiles have redundant records. Maps extend a half a cell outside the requested map limits.

A third-party description of the DTED data can be found at

```
http://www.fas.org/irp/program/core/dted.htm
```

Detailed official documentation is available at

```
http://www.nima.mil/publications/specs/printed/DTED/dted1-2.doc
```

The DTED files are binary. No line ending conversion or byte-swapping is required.

See Also

`usgsdem`	USGS 1-Degree (3-arc-sec resolution) digital elevation data
`gtopo30`	30-Arc-Sec global digital elevation data
`tbase`	TerrainBase Global 5-Min digital terrain data
`etopo5`	ETOPO5 Global 5-Min digital terrain data

Purpose

DTED file names

Syntax

`fname = dteds(latlim,lonlim)` returns 'Level 0' DTED file names (directory and name) required to cover the geographic region specified by `latlim` and `lonlim`.

`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, through the National Imagery and Mapping 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. 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

`dted` Read DTED digital elevation map data

eastof

Purpose

Wrap longitudes to values east of a meridian

Syntax

`ang = eastof(angin,meridian)` transforms input angles into equivalent angles east of the specified meridian.

`ang = eastof(angin,meridian,`*`units`*`)` uses the units defined by the input string *`units`*. If omitted, default units of `'degrees'` are assumed.

Example

```
eastof(1,360)
ans =
   361
```

Remarks

This function can be used to prepare vector data for use with regular data grids. Regular data grids use geographic locations that are strictly east of the left edge of the map.

See Also

`westof`	Wrap longitudes to values west of a meridian
`zero22pi`	Truncate angles into the 0 deg to 360 deg range
`npi2pi`	Truncate angles into the -180 deg to 180 deg range
`smoothlong`	Remove discontinuities for longitude data
`angledim`	Convert angles from one unit system to another

Purpose

Convert from eccentricity to flattening representation of the ellipsoid

Syntax

`flattening = ecc2flat(eccentricity)` returns the equivalent flattening for the input eccentricities. If the input, `eccentricity`, is a two-column vector, only the second column is used. This allows the standard two-element ellipsoid vectors to be used as rows of the input, because the second element of these vectors is the eccentricity. In all other cases, all columns of the input are used.

Description

Flattening and eccentricity are two methods of defining an ellipsoid.

Example

```
flattening = ecc2flat(almanac('earth','ellipsoid'))
flattening =
    0.0034
```

See Also

`almanac`	Planetary data
`ecc2n` `majaxis`	Other ellipsoid functions
`flat2ecc`	Convert from flattening to eccentricity

ecc2n

Purpose

Convert from eccentricity to the *n* representation of the ellipsoid

Syntax

`n = ecc2n(eccentricity)` returns the equivalent *n* for the input eccentricities. If the input, `eccentricity`, is a two-column vector, only the second column is used. This allows the standard two-element ellipsoid vectors to be used as rows of the input, because the second element of these vectors is the eccentricity. In all other cases, all columns of the input are used.

Description

Eccentricity and the parameter *n* are two methods of defining an ellipsoid. The definition of *n* is

(semimajor axis – semiminor axis)/(semimajor axis + semiminor axis)

Example

```
n = ecc2n(almanac('earth','ellipsoid'))
n =
   0.00167922039463
```

See Also

`almanac`	Planetary data
`ecc2flat` `majaxis`	Other ellipsoid functions
`n2ecc`	Convert from *n* to eccentricity

Purpose

Read 15-minute gridded geoid heights from the EGM96 geoid model of the Earth

Syntax

`[datagrid,refvec] = egm96geoid(scalefactor)` reads the data for the entire world, downsampling the data by the scale factor. The result is returned as a regular data grid and associated referencing vector. Heights are given in meters in the tide-free system.

`[datagrid,refvec] = egm96geoid(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. Longitude limits can be defined in the range `[–180 180]` or `[0 360]`. For example, `lonlim = [170 190]` returns data centered on the date line, while `lonlim = [–10 10]` returns data centered on the prime meridian.

Background

Although the Earth is round, it is not exactly a sphere. The shape of the Earth is usually defined by the geoid, which is defined as a gravitational equipotential surface, but can be conceptualized as the shape the ocean surface would take in the absence of waves, weather, and land. For cartographic purposes it is generally sufficient to treat the Earth as a sphere or ellipsoid of revolution. For other applications, a more detailed model of the geoid such as EGM 96 may be required. EGM 96 is a spherical harmonic model of the geoid complete to degree and order 360. This function reads from a file of gridded geoid heights derived from the EGM 96 harmonic coefficients.

Examples

Read the EGM 96 geoid grid for the world, taking every 10th point.

```
[datagrid,refvec] = egm96geoid(10);
```

Read a subset of the geoid grid at full resolution and interpolate to find the geoid height at a point between grid points.

```
[datagrid,refvec] = egm96geoid(1,[-10 -12],[129 132]);
z = ltln2val(datagrid,refvec,-11.1,130.22,'bicubic')
z =
   53.4809
```

Remarks

This function reads the 15-minute EGM96 grid file `WW15MGH.GRD`. The grid is available from

```
<http://www.nima.mil/GandG/wgs-84/egm96.html>
```

as either a DOS self-extracting compressed file or a UNIX compressed file. Do not modify the file once it has been extracted.

Information on the data set can be found at

```
<http://cddisa.gsfc.nasa.gov/926/egm96/egm96.html>
```

Maps will extend a half a cell outside the requested map limits.

There are 721 rows and 1441 columns of values in the grid at full resolution. The lower resolution atlas data in `GEOID.MAT` is derived from the EGM 96 grid.

See Also

`ltln2val` Returns map code value associated with positions

Purpose

Calculate elevation angle between points on an ellipsoid

Syntax

`elevang = elevation(lat1,lon1,alt1,lat2,lon2,alt2)` computes the elevation angle of point 2 as viewed from point 1. The elevation angle is the angle of the line of sight above the local horizontal at point 1. Points are specified as latitude, longitude, and altitude above the surface. Angles are in units of degrees, altitudes are in meters. The figure of the Earth is assumed to be the default ellipsoid. Inputs can be vectors of points. You can compute elevations from a point to a line by providing a scalar point 1 and vector point 2.

`[elevang,slantrange] = elevation(...)` also returns the slant range between the points in units of meters.

`[elevang,slantrange] = elevation(lat1,lon1,alt1,lat2,lon2,alt2,`*`angleunits`*`)` uses the inputs *`angleunits`* to define the angle units of the inputs and outputs. If omitted, `'degrees'` is assumed.

`[elevang,slantrange] = elevation(lat1,lon1,alt1,lat2,lon2,alt2,`*`angleunits`*`,`*`zunits`*`)` uses the inputs *`zunits`* to define the units of the altitude inputs and slant range outputs. If omitted, `'meters'` is assumed. All distance units recognized by `distdim` are valid inputs.

`[elevang,slantrange] = elevation(lat1,lon1,alt1,lat2,lon2,alt2,`*`angleunits`*`,ellipsoid)` computes the elevation and slant range of the ellipsoid defined by the input ellipsoid. The ellipsoid vector is of the form `[semimajor axes, eccentricity]`. If omitted, the default ellipsoid for the Earth is assumed. The altitudes are input and the slant range is returned in the units of `ellipsoid(1)`.

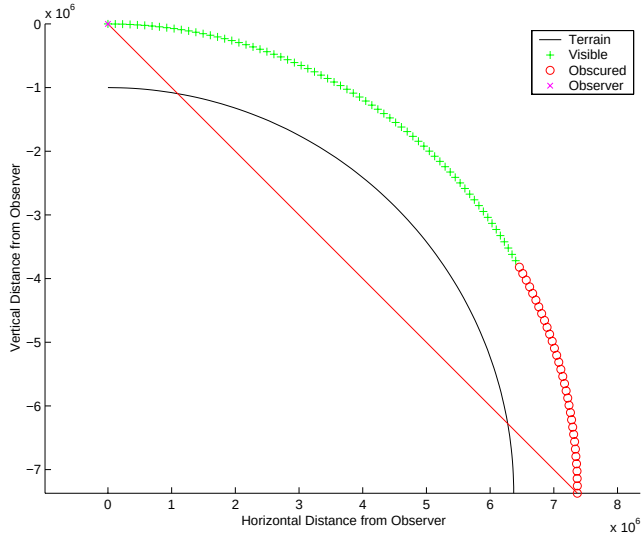
Example

What is the elevation angle of a point 90 degrees distant when both the observer and target are 1000 km altitude 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.

```
map = zeros(180,360); refvec = [1 90 -180];
los2(map,refvec,lat1,lon1,lat2,lon2,alt1,alt1);
```



See Also

`azimuth` Calculate azimuth between points on an ellipsoid

Purpose

Geographic ellipse defined by its center, semimajor axes, eccentricity, and azimuth

Syntax

`[lat,lon] = ellipse1(lat0,lon0,ellipse)` computes ellipses with a center at the point `lat0, lon0`. The ellipse is defined by the third input, which is of the form `[semimajor-axis, eccentricity]`. The `lat0, lon0` inputs can be scalar or column vectors. 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 `lat0` and `lon0`. The input semimajor axis is in degrees of arc length on a sphere. All ellipses are oriented so that their semimajor axis lies due north.

`[lat,lon] = ellipse1(lat0,lon0,ellipse,offset)` computes the ellipses where the semimajor 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` vector, of the form `[semimajor-axis, eccentricity]`. If omitted, the unit sphere, `ellipsoid=[1 0]`, is assumed. When a 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(lat0,lon0,ellipse,offset,`*`units`*`)`, `[lat,lon] = ellipse1(lat0,lon0,ellipse,offset,az,`*`units`*`)`, and `[lat,lon] = ellipse1(lat0,lon0,ellipse,offset,az,ellipsoid,`*`units`*`)` are all valid calling forms, which use the input *`units`* to define the angle units of the inputs and outputs. If omitted, `'degrees'` is assumed.

`[lat,lon] = ellipse1(lat0,lon0,ellipse,offset,az,ellipsoid,`*`units`*`,npts)` uses the

input `npts` to determine the number of points per ellipse computed. The input `npts` is a scalar, and if omitted, `npts=100`.

`[lat,lon] = ellipse1(track,...)` uses the *`track`* string to define either a great circle or rhumb line distance from the ellipse center. If *`track`* = `'gc'`, then great circle distances are computed. If *`track`* = `'rh'`, then rhumb line distances are computed. If omitted, `'gc'` is assumed.

`mat = ellipse1(...)` returns a single output argument where `mat=[lat lon]`. This is useful if only one ellipse is computed.

Example

Create and plot the small ellipse centered at (0°,0°), with a semimajor axis of 10° and a semiminor axis of 5°.

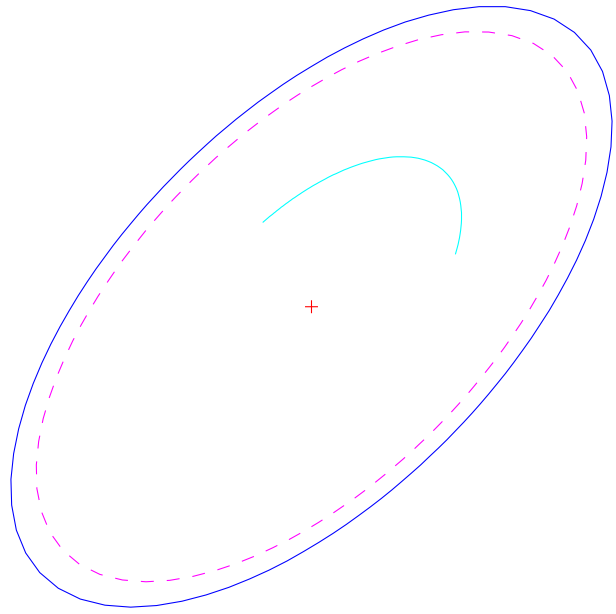
```
axesm mercator
ecc = axes2ecc(10,5);
plotm(0,0,'r+')
[elat,elon] = ellipse1(0,0,[10 ecc],45);
plotm(elat,elon)
```

If the desired radius is known in some nonangular distance unit, use the radius returned by the `almanac` function as the ellipsoid input to set the range units (use an empty azimuth entry to specify a full ellipse).

```
earthradius = almanac('earth','radius','nm');
[elat,elon] = ellipse1(0,0,[550 ecc],45,[],earthradius);
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-')
```



Remarks

This function extends the concept of the small circle, which is the locus of all points at an equal surface distance, to a “small ellipse.” You construct the small ellipse by computing the locus of points for which the distance from the center point varies as the parametric description of the ellipse.

You can define multiple circles from a single starting point by providing scalar `lat0`, `lon0` inputs and a two-column matrix for the ellipse definitions.

See Also

`scircle1`	Small circle defined by its center, range, and azimuth
`track1`	Track lines defined by starting point, azimuth, and range
`c`	Compute eccentricity given semimajor, semiminor axes

encodem

Purpose

Fill in regions of indexed data grids with specified values

Syntax

`newmap = encodem(map,seedmat)` fills in regions of the input data grid, `map`, 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]`.

`newmap = encodem(map,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.

Description

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 7's and the lower left region with 3's:

```
map = eye(4)
map =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1

newmap = encodem(map,[4,1,3; 1,4,7])
newmap =
     1     7     7     7
     3     1     7     7
     3     3     1     7
     3     3     3     1
```

See Also

`getseeds` Interactively create seed matrix

`imbedm` Encode data points into a regular data grid

Purpose

Show map precision

Syntax

`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 string. 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` Significant figures

eqa2grn

Purpose

Convert from equal area to Greenwich coordinates

Syntax

`[lat,lon] = eqa2grn(x,y)` converts the equal-area coordinate points `x` and `y` to the Greenwich 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 (0°,0°).

`[lat,lon] = eqa2grn(x,y,origin,ellipsoid)` specifies the two-element ellipsoid vector describing the ellipsoidal model of the figure of the Earth. The `ellipsoid` is spherical by default.

`[lat,lon] = eqa2grn(x,y,origin,`*`units`*`)` specifies the units for the outputs, where *`units`* is any valid angle units string. The default value is `'degrees'`.

`mat = eqa2grn(x,y,origin...)` packs the outputs into a single variable.

Description

This function converts data from equal-area *x-y* coordinates to Greenwich (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`	Convert Greenwich coordinates to equal-area coordinates
`hista`	Equal area histogram

Purpose

Read data from the ETOPO5 global 5-minute Digital Terrain Model

Syntax

`[datagrid,refvec] = etopo5(scalefactor)` reads the data for the entire world, downsampling the data by the scale factor. The result is returned as a regular data grid and an associated referencing vector.

`[datagrid,refvec] = etopo5(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

ETOPO5 is a global database of elevations and depths on a regular 5-minute grid. It is a compilation of data from a variety of different sources, including the U.S. Naval Oceanographic Office, U.S. Defense Mapping Agency, U.S. Navy Fleet Numerical Oceanographic Center, Bureau of Mineral Resources, Australia, and the Department of Industrial and Scientific Research, New Zealand. These databases were assembled by Margo Edwards at Washington University, St. Louis, Missouri.

Remarks

Data values are in whole meters, representing the elevation of the center of each cell. Some parts of the world are represented by data with a horizontal resolution as coarse as 1 degree by 1 degree. The vertical resolution varies from 1 meter for Australia and New Zealand to as much as 150 meters for parts of Africa, Asia, and South America. Oceanographic data in areas shallower than 200 meters contains little detail, because of how depth contours were converted to gridded depths.

ETOPO5 is being superseded by the development of a new digital terrain model called TerrainBase. See the `tbase` external interface function for more information.

The `etopo5` function reads the version of the database contained in two text files, `etopo5.southern.bat` and `etopo5.northern.bat`. These files are available over the Internet from The MathWorks:

```
ftp://ftp.mathworks.com
```

An overview of the data can be found at the ETOPO5 information page at the U.S. Geological Survey:

```
http://edcwww.cr.usgs.gov/glis/hyper/guide/etopo5
```

and from the U.S. National Geophysical Data Center Web page:

```
http://www.ngdc.noaa.gov/mgg/global/etopo5.HTML
```

Examples

Read every tenth point in the data set:

```
scalefactor = 10;
[datagrid,refvec] = etopo5(scalefactor);
whos
  Name              Size         Bytes  Class
  datagrid          216x432     746496  double array
  refvec            1x3             24  double array
  scalefactor       1x1              8  double array
limitm(datagrid,refvec)
ans =
   -90    90     0   360
```

Read in data for Korea and Japan at the full resolution:

```
scalefactor = 1; latlim = [30 45]; lonlim = [115 145];
[datagrid,refvec] = etopo5(scalefactor,latlim,lonlim);
whos datagrid
  Name      Size         Bytes  Class
  datagrid 180x360       518400  double array
```

See Also

`gtopo30`	Read elevation data from GTOPO30
`tbase`	Read data from the TerrainBase model
`usgsdem`	Read USGS digital elevation maps

References

More information on ETOPO5 can be found in reference `[4]` located in the Bibliography at the end of this appendix.

Purpose

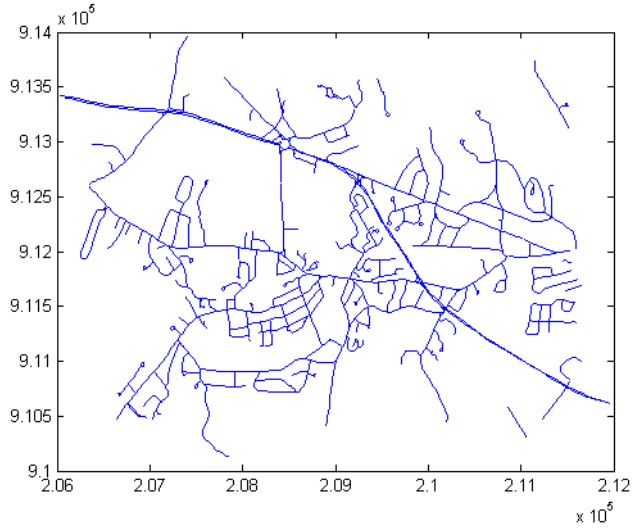
Extract the field values from a structure

Syntax

`a = extractfield(s, name)` returns the field values specified by the field named `name` into the 1-by-n output array `a`. `n` is the total number of elements in the field `name` of structure `s`, that is, `n = numel([s(:).(name)])`. `name` is a case-sensitive string defining the field name of the structure `s`. `a` is a cell array if any field values in the field name contain a string 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`.

Examples

```
% Plot the X, Y coordinates of the road's shape
roads = shaperead('concord_roads.shp');
plot(extractfield(roads,'X'),extractfield(roads,'Y'));

% Extract the names of the roads
roads = shaperead('concord_roads.shp');
names = extractfield(roads,'STREETNAME');

% Extract a mix-type field into a cell array
S(1).Type = 0;
S(2).Type = logical(0);
mixedType = extractfield(S,'Type');
```



See Also `struct`, `shaperead`

Purpose

Extract vector data from geographic data structures

Syntax

`[lat,lon] = extractm(gstruct,object)` extracts vector data from those entries in the Mapping Toolbox geographic data structure that have tags beginning with the *`object`* string. The output vectors use `NaN`s to separate the entries in the map structure. Matches of the tag string must be vector data (lines and patches) to be included in the output.

`[lat,lon] = extractm(gstruct,objects)`, where `objects` is a character array, allows more than one object to be the basis for the search.

`[lat,lon] = extractm(gstruct,objects,'exact')` requires an exact match to extract data.

`[lat,lon,indx] = extractm(gstruct)` extracts all vector data from the input map structure.

`[lat,lon,indx] = extractm(...)` also returns the vector `indx` identifying the entries in the structure that meet the selection criteria.

`mat = extractm(...)` returns the vector data in a single, two-column matrix, in which the first column contains latitudes and the second column longitudes.

Examples

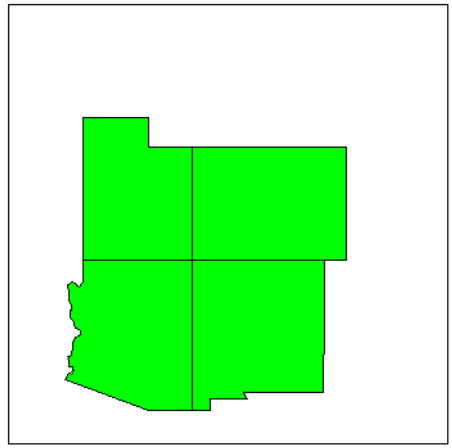
Extract the District of Columbia from the low-resolution U.S. vector data:

```
load usalo
extractm(state,'district of columb')
ans =
   38.9000  -77.0700
   38.9000  -77.0500
   38.9000  -77.0700
   38.8700  -77.0200
   38.8000  -77.0200
   38.7800  -77.0300
   38.9000  -76.9000
   39.0000  -77.0300
   38.9500  -77.1200
   38.9000  -77.0700
```

Extract the states that meet at the Four Corners and plot them:

```
states4 = strvcat('colo','new mex','ariz','utah');
```

```
[lat,long,indx] = extractm(state,states4);

axesm mercator
patchm(lat,long,'g')
```



Remarks

A Mapping Toolbox geographic data structure is a MATLAB structure that can contain line, patch, text, regular data grid, geolocated data grid, and light objects.

See Also

`extractfield`, `geoshow`, `mapshow`, `updategeostruct`, `mlayers`, `displaym`

Purpose

Project 3-D patch objects onto the current map axes

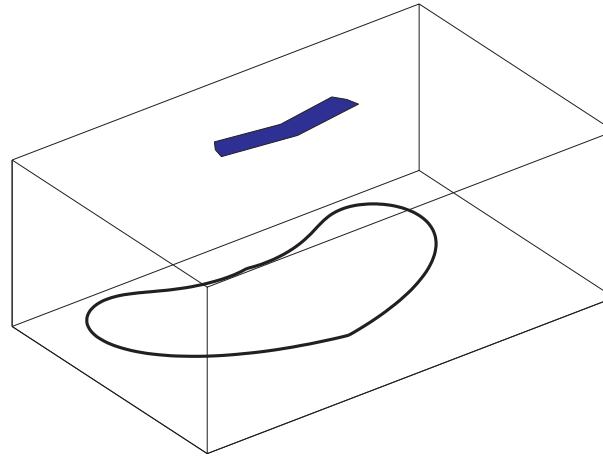
Syntax

`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`	Project 2-D patch objects onto the current map axes
`patchesm`	Project multiple patch objects more rapidly
`patchm`	Project and display patch objects on the current map axes

Purpose

Project 2-D patch objects onto the current map axes

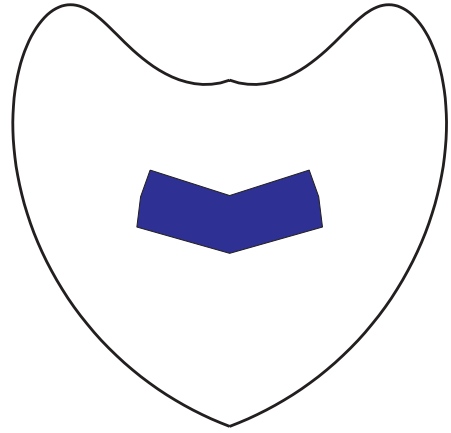
Syntax

`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`	Project 3-D patch objects onto the current map axes
`patchesm`	Project multiple patch objects more rapidly
`patchm`	Project and display patch objects on the current map axes

filterm

Purpose

Filter geographic data sets

Syntax

`[newlat,newlong] = filterm(lat,long,map,refvec,allowed)` filters geographic data based upon the corresponding entries of a regular data grid, `map`, with a three-element referencing vector `refvec`. The data locations to be filtered are input in the vectors `lat` and `lon`. For those locations corresponding to entries of `map` equal to one of the values contained in the vector `allowed`, an output location is returned in `newlat` and `newlon`. Those locations not corresponding to such entries of `map` are not returned in the outputs.

Examples

Filter a random set of 100 geographic points. Use the `topo` map for starters:

```
load topo
```

Then generate 100 random points:

```
lat = -90+180*rand(100,1);
long = -180+360*rand(100,1);
```

Make a land map, which is `1` where `topo>0` elevation:

```
land = topo>0;
[newlat,newlong] = filterm(lat,long,land,topolegend,1);
size(newlat)
ans =
    15     1
```

15 of the 100 random points fall on *land*.

See Also

`hista`	Spatial equal area histogram
`histr`	Spatial equirectangular histogram

Purpose

Find latitude and longitude coordinates for nonzero map entries

Syntax

`[lat,lon] = findm(map,refvec)` returns latitude and longitude vectors `lat` and `lon`, which provide the locations of all nonzero entries of the regular data grid `map`, with three-element referencing vector `refvec`.

`[lat,lon,val] = findm(map,refvec)` also returns the values `val` of the data grid corresponding to the `lat` and `lon` locations.

`[lat,lon,val] = findm(latin,lonin,map)` removes the *regular* matrix restriction. Two matrices, `latin` and `lonin`, the same size as `map`, must provide cell-by-cell latitude and longitude coordinates matched with the corresponding entries of `map`.

`mat = findm(...)` returns a single output `mat` of the form `[lat,lon]`.

Description

This function works in two modes: with a regular matrix restriction and without.

Examples

The entered map can also be the result of a logical statement. Where is elevation greater than 5500 meters?

```
load topo
mat = findm((topo>5500),topolegend)
mat =
   34.5000   79.5000
   34.5000   80.5000
   30.5000   84.5000
   28.5000   86.5000
```

These points are in the Himalayas.

See Also

`find` Find indices and values of nonzero elements (see the online MATLAB Function Reference documentation)

fipsname

Purpose

Read the FIPS (Federal Information Processing Standard) name file used with the TIGER thinned boundary files

Syntax

`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`*.

Remarks

The FIPS name files, along with the TIGER thinned boundary files, are available over the Internet at

```
ftp://ftp.census.gov/pub/tiger/boundary/
```

Example

```
struc = fipsname('st_name.dat')
struc =
1x57 struct array with fields:
    name
    id

s(1)
ans =
    name: 'Alabama'
      id: 1
```

See Also

`tigermif`	Read TIGER MapInfo Interchange Format thinned boundary files
`tigerp`	Read TIGER ArcInfo Format thinned boundary files

Purpose

Convert from flattening to eccentricity representation of the ellipsoid

Syntax

`eccentricity = flat2ecc(flattening)` returns the equivalent eccentricity for the input `flattening`. If the input, `flattening`, is a two-column vector, only the second column is used. This allows two-element vectors to be used as rows of the input, since the form `[semimajor-axis, flattening]` is a complete representation of an ellipsoid (but is not the standard form for ellipsoid vectors in the Mapping Toolbox). In all other cases, all columns of the input are used.

Description

Flattening and eccentricity are two methods of defining an ellipsoid.

Example

```
e = flat2ecc(0.003353)
e =
   0.08182149712026
```

This eccentricity is the default value for the Earth.

See Also

`almanac`	Planetary data
`ecc2flat`	Convert from eccentricity to flattening
`ecc2n` `majaxis`	Other ellipsoid functions

flatearthpoly

Purpose

Insert points along the date line to the pole

Syntax

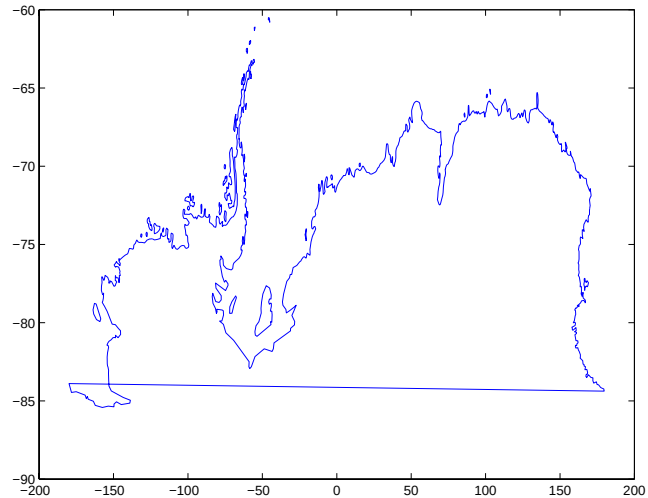
`[lat2,lon2] = flatearthpoly(lat,lon)` inserts points in the input latitude and longitude vectors at +/- 180 longitude and to the poles. The resulting vectors look like the result of `patchm` on a cylindrical projection and do not encompass the poles. Inputs and outputs are in degrees.

`[lat2,lon2] = flatearthpoly(lat,lon,origin)` centers the polygon on the provided origin. The origin is a scalar longitude or a three-element vector containing latitude, longitude, and orientation in units of degrees.

Example

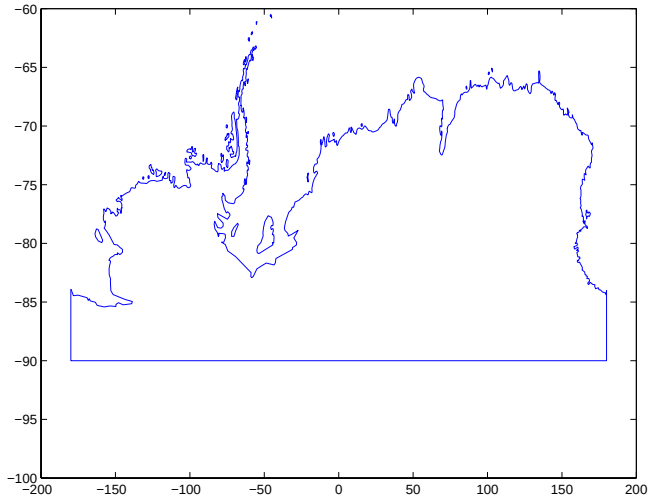
Vector data for geographic objects that encompass a pole will inevitably encounter or cross the date line. While the Mapping Toolbox properly displays such polygons, they can cause problems for functions like the polygon intersection and Boolean operations that work with Cartesian coordinates. When these 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.

```
[lat,long]=extractm(worldlo('POpatch'),'Antarctica');
plot(long,lat)
```



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.

```
[lat2,lon2]=flatearthpoly(lat,long);
plot(lon2,lat2)
ylim([-100 -60])
```



See Also

`polybool`	Polygon Boolean operations
`polyxpoly`	Polygon intersections
`mfwdtran`	Process the map forward transformations

framem

Purpose

Toggle and control the display of the map frame

Syntax

`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 string argument `'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 *`linespec`* string recognized by the MATLAB `line` function.

`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.

Remarks

You can also create or alter map frame properties using the `axesm` or `setm` functions.

See Also

`axesm`	Define map axes and set map properties
`setm`	Set map properties

Purpose

Convert great circles to small circle notation

Syntax

`[centerlat,centerlong,radius] = gc2sc(lat,long,az)` returns the *small circle notation* for great circles entered in *great circle notation*.

`[centerlat,centerlong,radius] = gc2sc(lat,long,az,units)` specifies the standard angle unit string. The default value is `'degrees'`.

Description

Great circles are a subcategory of small circles, having a radius of 90°. Because of the computational circumstances under which these objects often arise, however, two different notations are convenient.

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 (25°S,70°W) on an azimuth of 45°, how can it be represented in small circle notation?

```
[newlat,newlong,range] = gc2sc(-25,-70,45)
newlat =
  -39.8557
newlong =
   42.9098
range =
    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 0° and proceeds on an azimuth of 90° or 270°. In small circle notation, this is

```
[newlat,newlong,range] = gc2sc(0,-70,270)
newlat =
    90
newlong =
 -145.9638
range =
    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.

Remarks

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 90°. In the above example, the South Pole would also be a suitable center for the equator in small circle notation.

See Also

`antipode`	Find antipodal points
`distdim`	Distance unit conversions

`gcxgc`
`gcxsc`
`rhxrh`
`crossfix`
	Intersection functions

Purpose

Get current map structure

Syntax

`mapstruct = gcm` returns the map axes *map structure*, which contains the settings for all the current map axes properties.

`mapstruct = gcm(hndl)` specifies the map axes by axes handle.

Examples

Establish a map axes with default values, then look at the structure:

```
axesm mercator
mapstruct = gcm
mapstruct =
     mapprojection: 'mercator'
              zone: []
        angleunits: 'degrees'
            aspect: 'normal'
       fixedorient: []
              geoid: [1 0]
       maplatlimit: [-86 86]
       maplonlimit: [-180 180]
      mapparallels: 0
        nparallels: 1
            origin: [0 0 0]
     falsenorthing: 0
      falseeasting: 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: 0.50000000000000
```

```
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: 9
     fontunits: 'points'
    fontweight: 'normal'
   labelformat: 'compass'
    labelunits: 'degrees'
 labelrotation: 'off'
 meridianlabel: 'off'
mlabellocation: 30
mlabelparallel: 86
   mlabelround: 0
 parallellabel: 'off'
plabellocation: 15
plabelmeridian: -180
   plabelround: 0
```

Remarks

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`	Create map axes object
`getm`	Query map axes map structure
`setm`	Modify map axes map structure

Purpose

Get current mouse point from the map

Syntax

`pt = gcpmap` returns the current point of the current map axes in the form `[latitude longitude z-altitude]`.

`pt = gcpmap(hndl)` specifies the map axes in question by its handle.

Remarks

`gcpmap` works much like the standard MATLAB `get(gca,'CurrentPoint')`, except that the returned matrix is in `[lat lon z]`, not `[x y z]`.

See Also

`inputm` Mouse selection of position

gcwaypts

Purpose

Find equally spaced waypoints along a great circle

Syntax

`[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.

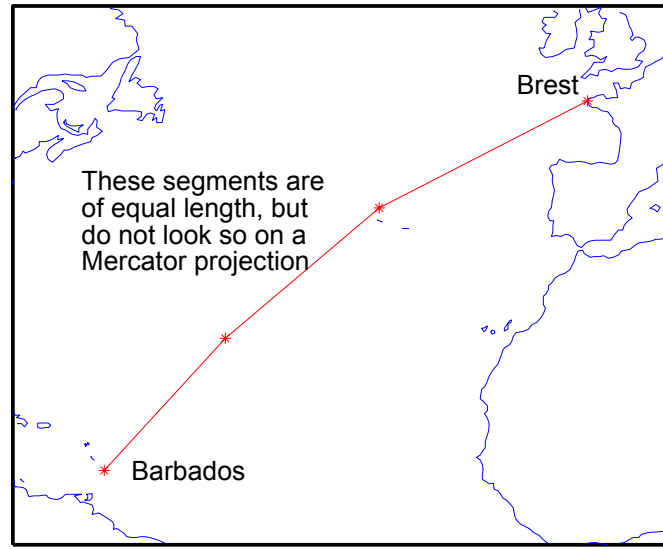
In navigational practice, great circle paths are often approximated by rhumb line segments. This is done to come reasonably close to the shortest distance between points without requiring course changes too frequently. The `gcwaypts` function provides an easy means of finding waypoints along a great circle path that can serve as endpoints for rhumb line segments (track legs).

Examples

Imagine you own a sailing yacht and are planning a voyage from North Point, Barbados (13.33° N,59.62°W), to Brest, France (48.33°N,4.83°W). To divide the track into three equal-length segments,

```
[l,g] = gcwaypts(13.33,-59.62,48.33,-4.83,3)
l =
   13.3300
   27.3316
   39.6250
   48.3300
g =
  -59.6200
  -45.8919
  -28.4459
```

```
   -4.8300
```



See Also

`dreckon`	Dead reckon points for a track
`legs`	Courses and distances between waypoints
`navfix`	Mercator-based navigational fixing
`track`	Connect waypoints

Purpose

Provide intersection coordinates for pairs of great circles

Syntax

`[newlat,newlong] = gcxgc(lat1,long1,az1,lat2,long2,az2)` returns the two intersection points of pairs of great circles input in *great circle notation*. When the two great circles are identical (which is not, in general, apparent by inspection), two `NaN`s are returned instead and a warning is displayed. For multiple pairings, the inputs must be column vectors.

`[newlat,newlong] = gcxgc(lat1,long1,az1,lat2,long2,az2,`*`units`*`)` specifies the standard angle unit string. The default value is `'degrees'`.

Description

For any pair of great circles, there are two possible intersection conditions: the circles are identical or they intersect exactly twice on the sphere.

Great circle notation consists of a point on the great circle and the azimuth at that point along which the great circle proceeds.

Examples

Given a great circle passing through (10°N,13°E) and proceeding on an azimuth of 10°, where does it intersect with a great circle passing through (0°, 20°E), on an azimuth of -23° (that is, 337°)?

```
[newlat,newlong] = gcxgc(10,13,10,0,20,-23)
newlat =
   14.3105  -14.3105
newlong =
   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 0° or 180°:

```
[newlat,newlong] = gcxgc(10,13,0,0,20,180)
newlat =
   -90    90
newlong =
   -174.4504   12.5094
```

The two meridians intersect at the North and South Poles, which is exactly correct.

See Also

`antipode`	Find antipodal points
`gc2sc`	Convert great circle to small circle notation

`scxsc`
`gcxsc`
`rhxrh`
`crossfix`
`polyxpoly`
	Other intersection functions

Purpose

Provide intersection coordinates for great circles paired with small circles

Syntax

`[newlat,newlong] = gcxsc(gclat,gclong,gcaz,sclat,sclong,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,newlong] = gcxsc(...,`*`units`*`)` specifies the standard angle unit string. The default value is `'degrees'`.

Description

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 (43°N,0°) and proceeding on an azimuth of 10°, where does it intersect with a small circle centered at (47°N,3°E) with an arc length radius of 12°?

```
[newlat,newlong] = gcxsc(43,0,10,47,3,12)
newlat =
   35.5068   58.9143
newlong =
   -1.6159    5.4039
```

See Also

`gc2sc`	Convert great circle to small circle notation

`gcxgc`
`scxsc`
`rhxrh`
`crossfix`
`polyxpoly`
	Other intersection functions

geoloc2grid

Purpose

Convert a geolocated data array to a regular data grid

Syntax

`[Z, refvec] = 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 referencing vector `refvec`. `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`.

Remarks

`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.

Example

```
% Load the geolocated data array 'map1'
% and grid it to 1/2-degree cells.
load mapmtx
cellsize = 0.5;
[Z, refvec] = 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 in subplot 1
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');

% Display 'Z' as a regular data grid in subplot 2
subplot(1,2,2)
```

```
ax =
axesm('mercator','MapLatLimit',latlim,'MapLonLimit',lonlim,...
 'Grid','on','MeridianLabel','on','ParallelLabel','on');
set(ax,'Visible','off'
geoshow(Z, refvec, 'DisplayType', 'texturemap');
```

geoshow

Purpose

Display map latitude and longitude data

Syntax

`geoshow(s)` displays the graphic features stored in the geographic data structure `s`. If `Lat` and `Lon` fields are present, then their coordinate values are projected to map coordinates if the axes has a projection. Otherwise, `Lon` is plotted as latitude and `Lat` as longitude. If `s` includes X and Y fields, then they are used directly to plot features in map coordinates.

`geoshow(lat,lon)` or
`geoshow(lat,lon, ..., 'DisplayType', displaytype, ...)` displays the equal length coordinate vectors `lat` and `lon`. `lat` and `lon` can contain embedded `NaN`s, delimiting coordinates of lines or polygons. In this case, `displaytype` can be `'point'`, `'line'`, or `'polygon'` and defaults to `'line'`.

`geoshow(lat,lon,Z, ..., 'DisplayType', displaytype, ...)`, where `lat` and `lon` are M-by-N coordinate arrays, `Z` is an M-by-N array of class double, and `displaytype` is `'texturemap'` or `'contour'`, displays a geolocated data grid. `Z` can contain `NaN` values.

```
geoshow(lat,lon,I)
geoshow(lat,lon,BW)
geoshow(lat,lon,X,cmap)
geoshow(lat,lon,RGB)
```

where `I` is an intensity image, `BW` is a logical image, `X` is an indexed image with colormap `cmap`, or `RGB` is a true-color image, displays a geolocated image. The image is rendered as a texture map on a zero-elevation surface. If specified, `'DisplayType'` must be set to `'image'`. Examples of geolocated images include a color composite from a satellite swath or an image originally referenced to a different coordinate system.

`geoshow(Z,R, ..., 'DisplayType', displaytype,...)`, where `Z` is of class double and `DisplayType` is `'contour'` or `'texturemap'`, displays a regular M-by-N data grid. `R` is a referencing matrix or referencing vector.

```
geoshow(I,R)
geoshow(BW,R)
geoshow(RGB,R)
geoshow(A,CMAP,R)
```

displays an image georeferenced to latitude/longitude. It is rendered as an image object if the display geometry permits; otherwise, the image is rendered

as a texture map on a zero-elevation surface. If specified, `'DisplayType'` must be set to `'image'`.

`geoshow(filename)` displays data from `filename` according to the type of file format. The `DisplayType` parameter is automatically set, according to the following table:

Format	DisplayType
Shape file	`'point'`, `'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)

`geoshow(ax, ...)` sets the axes parent to `ax`. This is equivalent to `geoshow(..., 'Parent', ax, ...)`.

`h = geoshow(...)` returns a handle to a MATLAB graphics object, an array of object handles, or in the case of vector data, a map graphics object.

`geoshow(..., param1, val1, param2, val2, ...)` specifies parameter/value pairs that modify the type of display or set MATLAB graphics properties.

Parameters

Parameter names can be abbreviated and are case insensitive. Parameters include

- `'DisplayType'`: The `DisplayType` parameter specifies the type of graphic display for the data. The value must be consistent with the type of data being displayed, as shown in the following table:

Data Type	Value(s)
Vector	`'point'`, `'line'`, or `'polygon'`
Image	`'image'`
Grid	`'texturemap'` or `'contour'`

Graphics Properties

In addition to specifying a parent axes, you can set the following properties for line, point, and polygon:

- `DisplayType`:

DisplayType	Property Name
`'line'`	`'Color'`, `'LineStyle'`, `'LineWidth'`, and `'Visible'`
`'point'`	`'Marker'`, `'Color'`, `'MarkerEdgeColor'`, `'MarkerFaceColor'`, `'MarkerSize'`, and `'Visible'`
`'polygon'`	`'FaceColor'`, `'FaceAlpha'`, `'LineStyle'`, `'LineWidth'`, `'EdgeColor'`, `'EdgeAlpha'`, and `'Visible'`

  Refer to the MATLAB Graphics documentation on line, patch, image, surface, and mesh for a complete description of these properties and their values.

- `SymbolSpec`:

  The `SymbolSpec` parameter specifies the symbolization rules used for vector data through a structure returned by `makesymbolspec`. It is used only for vector data.

  In cases where both `SymbolSpec` and one or more graphics properties are specified, the graphics properties override any settings in the symbol spec structure. See example 3 below.

  To change the default symbolization rule for a property name/property value pair in the symbol spec, prefix the word `'Default'` to the graphics property name (listed in the preceding table). See example 2 below.

Remarks

You can use `geoshow` to render vector data in an `axesm` figure. However, you cannot subsequently change the map projection using `setm`.

`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.

Examples

Example 1

Display world coastlines, without a projection.

```
load coast
figure
geoshow(lat,long);

% Add the international boundaries as black lines
boundaries = updategeostruct(worldlo('POline'));
symbols = makesymbolspec('Line',{'Default','Color','black'});
hold on
geoshow(gca,boundaries(1),'SymbolSpec',symbols);
```

Example 2

Override the SymbolSpec default rule.

```
% Create a SymbolSpec to display Alaska and Hawaii as red
polygons.
symbols = makesymbolspec('Polygon', ...
          {'tag','Alaska','FaceColor','red'}, ...
          {'tag','Hawaii','FaceColor','red'});

% Display all the other states in blue.
figure;worldmap('na');
geoshow(usahi('statepatch'),'SymbolSpec',symbols, ...
                            'DefaultFaceColor','blue', ...
                            'DefaultEdgeColor','black');
```

Example 3

Display the Korean data grid, with the worldhi boundaries.

```
% Display the Korean data grid as a texture map.
load korea
figure;axesm mercator
```

```
geoshow(gca,map,maplegend,'DisplayType','texturemap');
colormap(demcmap(map))

% Set the display to the bounding box of the data grid.
[latlim,lonlim] = limitm(map,maplegend);
[x,y]=mfwdtran(latlim,lonlim);
set(gca,'Xlim',[min(x(:)), max(x(:))]);
set(gca,'Ylim',[min(y(:)), max(y(:))]);

% Get the region's worldhi data.
[korea_lat, korea_lon]= extractm(worldhi(latlim, lonlim));

% Display the worldhi boundaries.
hold on
geoshow(korea_lat, korea_lon);

% Mask the ocean.
geoshow(worldlo('oceanmask'),'EdgeColor','none','FaceColor','c')
```

Example 4

Display the EGM96 geoid heights.

```
% Display the geoid as a texture map.
load geoid
figure
axesm eckert4; framem; gridm;
h=geoshow(geoid, geoidlegend, 'DisplayType','texturemap');
axis off

% Set the Z data to the geoid height values, rather than a
% surface with zero elevation.
set(h,'ZData',geoid);
light; material(0.6*[ 1 1 1]);
 set(gca,'dataaspectratio',[ 1 1 200]);
hcb = colorbar('horiz');
set(get(hcb,'Xlabel'),'String','EGM96 geoid heights in m.')

% Mask out all the land.
geoshow(worldlo('POpatch'),'FaceColor','black');
zdatam(handlem('patch'),max(geoid(:)));
```

Example 5

Display the moon albedo image as a texture map.

```
load moonalb
figure
axesm ortho
geoshow(moonalb, moonalblegend, 'DisplayType','image');
axis off
```

See Also

makesymbolspec, mapshow, mapview, updategeostruct

geotiff2mstruct

Purpose

Convert GeoTIFF information to a map projection structure

Syntax

`mstruct = geotiff2mstruct(info)` converts the GeoTIFF info structure `info` to a map projection structure, `mstruct`.

Example

```
% Verify that the info structure from ‘boston.tif’
% converts to an mstruct.

% Obtain the info structure of 'boston.tif'.
info = geotiffinfo('boston.tif');

% Get the projection list structure for conversion from
% GeoTIFF to a map projection structure.
S = projlist('all');

% Verify info converts to a mstruct.
id = strmatch(info.CTProjection,{S.GeoTIFF},'exact');
if ~isempty(id) && S(id).mstruct
  mstruct = geotiff2mstruct(info);
else
  fprintf('Unable to convert %s to an mstruct.\n',...
  info.CTProjection);
end
```

See Also

`axesm`, `defaultm`, `geotiffinfo`, `projfwd`, `projinv`, `projlist`

Purpose

Information about a GeoTIFF file

Syntax

`info = geotiffinfo(filename)` returns a structure whose fields contain file and cartographic information about a GeoTIFF file.

`filename` is a string that specifies the name of the GeoTIFF file. `filename` can include the directory name; otherwise, the file must be in the current directory or in a directory on the MATLAB path. If the named file includes the extension `.TIF` or `.TIFF` (either upper- or lowercase), the extension can be omitted from `filename`.

If `filename` is a file containing more than one GeoTIFF image, `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 more than one image exists in the file, it is assumed that each image will have the same cartographic information and the same image width and height.

`info = geotiffinfo(url)` reads the GeoTIFF image from an Internet URL. The `url` must include the protocol type (e.g., `"http://"`).

Field Description

The `info` structure contains the following fields:

`Filename`	String containing the name of the file
`FileModDate`	String containing the modification date of the file
`FileSize`	Integer indicating the size of the file in bytes
`Format`	String containing the file format, which should always be `'tiff'`
`FormatVersion`	String or number specifying the 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
`ColorType`	String indicating the type of image: `'truecolor'` for a true-color (RGB) image, `'grayscale'` for a grayscale intensity image, or `'indexed'` for an indexed image

`ModelType`	String indicating the type of coordinate system used to georeference the image: `'ModelTypeProjected'`, `'ModelTypeGeographic'`, or `''`
`PCS`	String describing the projected coordinate system
`Projection`	String describing the EPSG identifier for the underlying projection method
`MapSys`	String indicating the 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, zero if not applicable or unknown
`CTProjection`	String containing the GeoTIFF identifier for the underlying projection method
`ProjParm`	An N-by-1 double 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`	An 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`	String indicating the geographic coordinate system
`Datum`	String indicating the projection datum type, such as 'North American Datum 1927' or 'North American Datum 1983'
`Ellipsoid`	String indicating the 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`	String indicating the prime meridian location, for example, `'Greenwich'` or `'Paris'`
`PmLongToGreenwich`	Double indicating the decimal degrees of longitude between this prime meridian and Greenwich. Prime meridians to the west of Greenwich are negative.
`UOMLength`	String indicating the units of length used in the projected coordinate system
`UOMLengthInMeters`	Double defining the `UOMLength` unit in meters
`UOMAngle`	String indicating the 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:

- ImagePoints — Structure containing the image coordinates of the tiepoints
- WorldPoints — Structure containing the world coordinates of the tiepoints

  The `ImagePoints` and `WorldPoints` structures each contain these fields:
  - `X` — A double array of size N-by-1 for the X values
  - `Y` — A double array of size N-by-1 for the Y values

- `Z` — A double array of size N-by-1 for the Z values

`PixelScale`	3-by-1 double array that specifies the X, Y, Z pixel scale values
`RefMatrix`	3-by-2 double referencing matrix that must be unambiguously defined by the GeoTIFF file; otherwise it is returned 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
`CornerCoords`	Contains the GeoTIFF image corners in projected and latitude-longitude coordinates. The corner coordinate values are stored counterclockwise starting at the upper left corner followed by lower left, lower right, and ending at the upper right corner.

The `CornerCoords` structure contains four fields. Each is a 4-by-1 double array, or empty (`[]`), if unknown.

- `PCSX` — Coordinates in the Projected Coordinate System; equals `LON` if the model type is `'ModelTypeGeographic'`
- `PCSY` — Coordinate in the Projected Coordinate System; equals `LAT` if the model type is `'ModelTypeGeographic'`
- `LON` — Longitudes of the corner
- `LAT` — Latitudes of the corner

`ImageDescription`	String describing the image; omitted if not included
`GeoTIFFCodes`	Structure containing raw numeric values for those GeoTIFF fields that are encoded numerically in the file. These raw values, converted to a string elsewhere in the INFO structure, are provided here for reference.

The following fields are included in the `GeoTIFFCodes` structure:

- `Model`
- `PCS`

- GCS
- UOMLength
- UOMAngle
- Datum
- PM
- Ellipsoid
- ProjCode
- Projection
- CTProjection
- ProjParmId
- MapSys

Each is scalar except for ProjParmId, which is a column vector.

Example

```
info = geotiffinfo('boston.tif');
```

See Also

imfinfo, geotiffread, makerefmat, projfwd, projinv, projlist

geotiffread

Purpose

Read a georeferenced image from GeoTIFF file

Syntax

`A = geotiffread(filename)` reads the GeoTIFF image in `filename` into `A`. If the file contains a grayscale intensity image, `A` is a two-dimensional array. If the file contains a true-color (RGB) image, `A` is a three-dimensional (M-by-N-by-3) array.

`filename` is a string that specifies the name of the GeoTIFF file. `filename` can include the directory name; otherwise, the file must be in the current directory or in a directory on the MATLAB path. If the named file includes the extension `.TIF` or `.TIFF` (either upper- or lowercase), the extension can be omitted from `filename`.

`[X, cmap] = geotiffread(filename)` reads the indexed image in `filename` into I and its associated colormap into `cmap`. Colormap values in the image file are automatically rescaled into the range `[0,1]`.

`[X, cmap, R, bbox] = geotiffread(filename)` reads the indexed image into I, the associated colormap into `cmap`, the referencing matrix into `R`, 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 ([]).

`[A, R, bbox] = geotiffread(filename)` reads the image into `A`, the referencing matrix into `R`, and the bounding box into `bbox`.

`[...] = geotiffread(filename, idx)` reads in one image from a multiimage GeoTIFF file. `idx` is an integer value that specifies the order that the image appears in the file. For example, if `idx` is 3, `geotiffread` reads the third image in the file. If you omit this argument, `geotiffread` reads the first image in the file.

`[...] = geotiffread(url, ...)` reads the GeoTIFF image from an Internet URL. The URL must include the protocol type (e.g., "`http://`").

Example

1 Read and display the Boston GeoTIFF image:

```
[boston_X, boston_cmap, boston_R, bbox] =
geotiffread('boston.tif');
figure
mapshow(boston_X,boston_cmap,boston_R);
```

2 Read and display the Boston GeoTIFF panachromatic image:

```
[pan_I, pan_R, bbox] = geotiffread('boston_pan.tif');
figure
mapshow(pan_I, pan_R);
```

3 Overlay the Boston GeoTIFF panchromatic image with the Boston GeoTIFF multispectral image.

```
figure
mapshow(boston_X,boston_cmap,boston_R);
mapshow(gca,pan_I, pan_R);
```

See Also

`geotiffinfo`, `imread`, `mapview`, `mapshow`, `geoshow`

getm

Purpose

Get map object properties

Syntax

`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`	Define map axes and set map properties
`setm`	Set map properties

Purpose

Interactively assign seeds for data grid encoding

Syntax

`[row,col,val] = getseeds(map,refvec,nseeds)` prompts the user for a number, `nseeds`, of mouse-input locations on the current map axes. After the locations are selected, the user is prompted for a value to associate with each location. The outputs are the row and column, `row` and `col`, of the input regular data grid, `map`, with its associated referencing vector, `refvec`, corresponding to the input locations. The third output, `val`, returns the selected value for each location.

`[row,col,val] = getseeds(map,refvec,nseeds,seedval)` predefines the values of the locations. If `seedval` is a scalar, the same value is assigned to all points. If it is a vector with a length of `nseeds`, each entry corresponds to a particular location.

`seedmat = getseeds(...)` packs the outputs into a single, three-column matrix, `seedmat`, that is a suitable input for the `encodem` function. The form of this matrix is `[lat lon val]`.

Description

The `getseeds` function allows you to interactively create the seed matrix values used by the `encodem` function to fill in regions of data grids.

Examples

Demonstrate this for yourself by typing the following and interactively selecting points:

```
load topo
axesm('gortho','grid','on')
seedmat = getseeds(topo,topolegend,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` Fill in regions of indexed data grids with specified values

getworldfilename

Purpose

Derive a worldfile name from an image filename

Syntax

`worldfilename = getworldfilename(imagefilename)` returns the name of the corresponding worldfile derived from the name of an image file.

The worldfile and the image file have the same base name. If `imagefilename` follows the ".3" convention, then you create the worldfile 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 worldfile name.

If `imagefilename` has no extension, then `'.wld'` is appended to construct a worldfile name.

Examples

Given the following image filenames, `worldfilename` would return these worldfile names:

Image File Name	Worldfile Name
`myimage.tif`	`myimage.tfw`
`myimage.jpeg`	`myimage.jpegw`
`myimage`	`myimage.wld`

See Also

`worldfileread`, `worldfilewrite`

Purpose

Read elevation data from GLOBE Digital Elevation Map files into a regular data grid

Syntax

`[datagrid,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 that does not include an extension. GLOBEDEM first reads the ESRI header file found in the subdirectory `'/esri/hdr/'` and then the binary data file filename. 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 `n`th point is returned. The map data is returned as an array of elevations and associated referencing vector. Elevations are given in meters above mean sea level, using WGS 84 as a horizontal datum.

`[datagrid,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.

`[datagrid,refvec] = globedem(dirname,scalefactor,latlim,lonlim)` reads and concatenates data from multiple files within a GLOBE directory tree. The `dirname` input is a string with the name of the directory 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.

Remarks

The Mapping Toolbox 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 `NaN`s.

The data is available over the Internet via anonymous FTP from

```
<ftp://ftp.ngdc.noaa.gov/GLOBE_DEM/data/elev/>
```

The data and some documentation is also available over the World Wide Web from

```
<http://www.ngdc.noaa.gov/seg/topo/globe.shtml>
```

Examples

Determine the file that contains the area around Cape Cod.

```
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. Provide an empty file name, and select the file interactively.

```
[datagrid,refvec] = globedem([],20);
size(datagrid)
ans =
   300   540
```

Extract a subset of the data for Massachusetts at the full resolution.

```
latlim = [41 42.5]; lonlim = [-73 -69.9];
[datagrid,refvec] = globedem('f10g',1,latlim,lonlim);
size(datagrid)
ans =
   181 373
```

Replace the `NaN`s in the ocean with -1 to color them blue.

```
datagrid(isnan(datagrid)) = -1;
```

Extract some data for southern Louisiana in an area that straddles two tiles. Provide the name of the directory 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'

[datagrid,refvec] =
globedem('d:\externalData\globe\elev',1,latlim,lonlim);
size(datagrid)

ans =

        325.00        315.00
```

See Also

`demdataui`	Interactive tool to read digital elevation data
`dted`	Read Digital Terrain Elevation Data (DTED) data
`gtopo30`	Read GTOPO30 digital elevation data
`satbath`	Read global 2-minute topography from satellite bathymetry
`tbase`	Read data from the TerrainBase data set
`usgsdem`	Read USGS digital elevation maps

References

<http://www.ngdc.noaa.gov/seg/topo/globe.shtml>

globedems

Purpose

GLOBE DEM filenames

Syntax

`fname = globedems(latlim,lonlim)` returns a cell array of the filenames covering the geographic region for GLOBE DEM digital elevation maps. The region is specified by scalar latitude and longitude points, or two-element vectors of latitude and longitude limits 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 files 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 filenames for a given geographic region.

Remarks

The Mapping Toolbox reads data from GLOBE Version 1.0. GLOBE DEM first reads the corresponding ESRI header file found in the subdirectory `'/esri/hdr/'` and then the binary data file (with no extension).

Examples

Which files are needed for southern Louisiana?

```
latlim =[28.61 31.31]; lonlim = [-91.24 -88.62];
globedems(latlim,lonlim)

ans =

    'e10g'
    'f10g'
```

See Also

`globedem` Read GLOBE digital elevation map data

References

<http://www.ngdc.noaa.gov/seg/topo/globe.shtml>

Purpose

Numerical gradient, slope, and aspect of data grids

Syntax

`[aspect,slope,gradN,gradE] = gradientm(map,refvec)` computes the slope, aspect, and north and east components of the gradient for a regular data grid. If the map contains elevations in meters, the resulting aspect and slope are in units of degrees clockwise from north and up from the horizontal. The north and east gradient components are the change in the map variable per meter of distance in the north and east directions. The computation uses finite differences for the map variable on the default Earth ellipsoid.

`[aspect,slope,gradN,gradE] = gradientm(lat,lon,map)` does the computation for a geolocated data grid.

`[aspect,slope,gradN,gradE] = gradientm(map,refvec,ellipsoid)` and `[aspect,slope,gradN,gradE] = gradientm(lat,lon,map,ellipsoid)` use the provided ellipsoid definition. The ellipsoid vector is of the form `[semimajor axes, eccentricity]`. If the map contains elevations in the same units as `ellipsoid(1)`, the slope and aspect are in units of degrees. This calling form is most useful for computations on bodies other than the Earth.

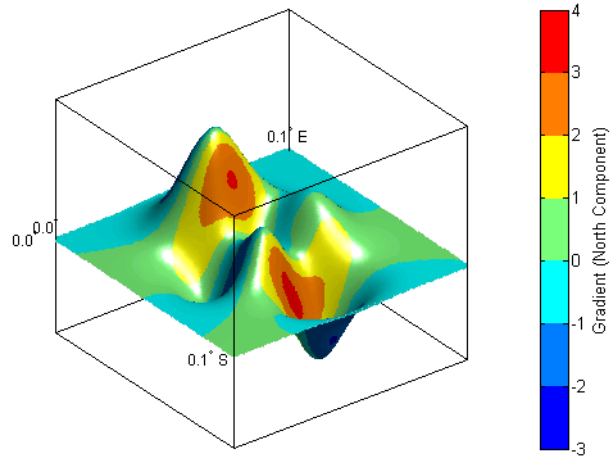
`[aspect,slope,gradN,gradE] = gradientm(lat,lon,map,ellipsoid,units)` specifies the angle units of the latitude and longitude inputs. If omitted, `'degrees'` is assumed. For elevation maps in the same units as `ellipsoid(1)`, the resulting slope and aspect are in the specified units. The components of the gradient are the change in the map variable per unit of `ellipsoid(1)`.

Example

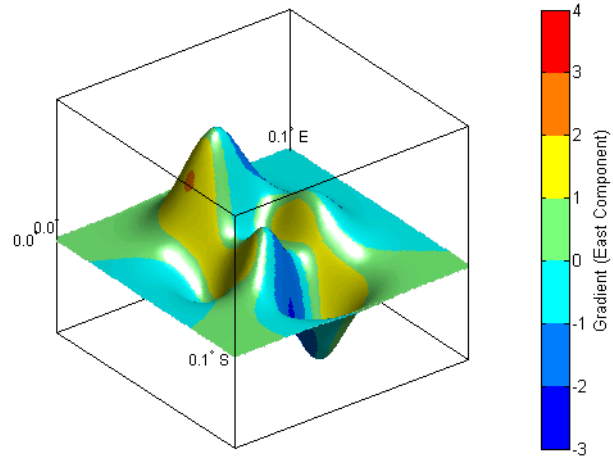
Compute the gradient, slope, and aspect for a regular data grid based on the MATLAB `peaks` matrix. Show the `gradientm` outputs as colors on a three-dimensional surface of elevation. The surfaces are shown with no vertical exaggeration.

```
clear all; close all;
datagrid = 500*peaks(100);
gridrv = [ 1000 0 0];
[aspect,slope,gradN,gradE] = gradientm(datagrid,gridrv);
figure
worldmap(gradN,gridrv,'none')
meshm(gradN,gridrv,size(datagrid),datagrid)
hcb = contourcmap(1,'jet','colorbar','on',...
'ylabelstring','Gradient (North Component)');
```

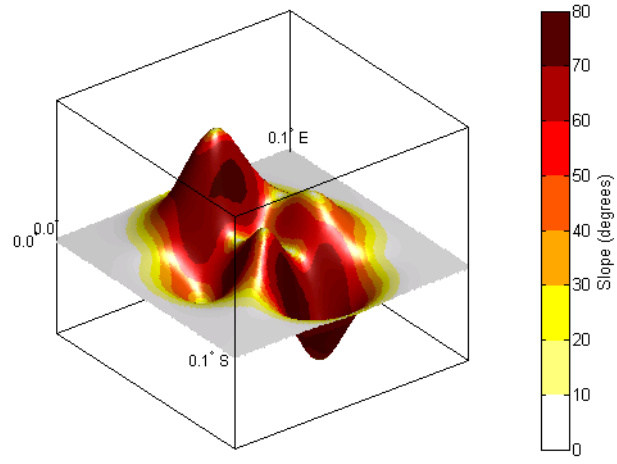
```
view(3)
daspectm meters
camlight
shading interp; lighting phong
framem off
```



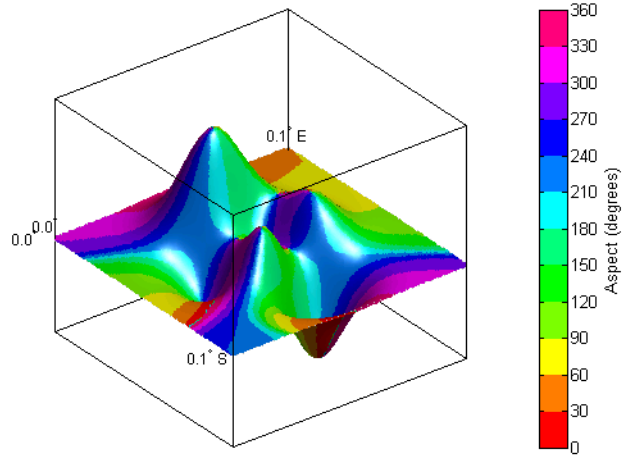

```
clmo surface
meshm(gradE,gridrv,size(datagrid),datagrid)
set(get(hcb,'ylabel'),'String','Gradient (East Component)');
shading interp; lighting phong
```



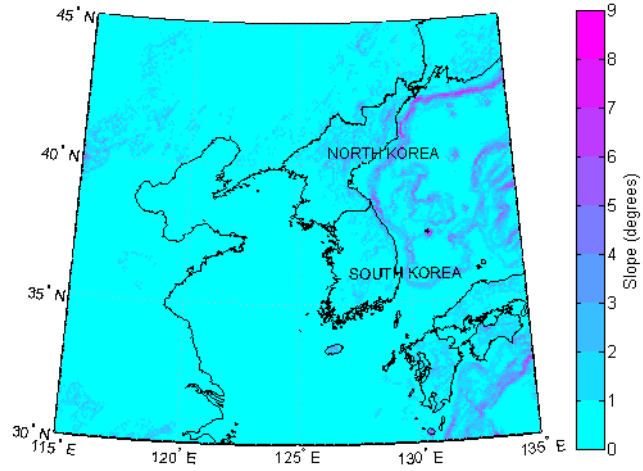

```
clmo surface
meshm(slope,gridrv,size(datagrid),datagrid)
contourcmap(10,'hot','colorbar','on',...
'ylabelstring','Slope (degrees)')
colormap(flipud(colormap))
shading interp; lighting phong
```



```
clmo surface
meshm(aspect,gridrv,size(datagrid),datagrid)
contourcmap(30,'hsv','colorbar','on',...
'ylabelstring','Aspect (degrees)')
shading interp; lighting phong
```




```
clear; close all;
load korea
[aspect,slope,gradN,gradE] = gradientm(map,refvec);
worldmap(slope,refvec);
contourcmap(1,'cool','colorbar','on',...
'ylabelstring','Slope (degrees)')
hidem(gca)
```



Remarks

Coarse digital elevation models can considerably underestimate the local slope. For the preceding map, the elevation points are separated by about 10 kilometers. The terrain between two adjacent points is modeled as a linear variation, while actual terrain can vary much more abruptly over such a distance.

See Also

`gradient`	Approximate gradient
`viewshed`	Visible areas from a point on a digital elevation map

grepfields

Purpose

Identify matching fields in fixed record length files

Syntax

`grepfields(`*`filename`*`,`*`searchstring`*`)` displays lines in the file that begin with the search string. 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 string 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.

Example

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 string `'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.

Remarks

See `readfields` for a complete discussion of the format and contents of the `fields` argument.

See Also

`readfields` Read fields or records from a fixed-format file

Purpose

Toggle and control the display of the map grid

Syntax

`gridm` toggles the visibility of the map grid by setting the map axes property `Grid` to `'on'` or `'off'`. The default setting for map axes is `'off'`.

`gridm('on')` sets the map axes `Grid` property to `'on'`.

`gridm('off')` sets the map axes `Grid` property to `'off'`.

`gridm('reset')` resets the entire grid using the current properties. This is essentially a refresh option.

`gridm(`*`linestyle`*`)` sets the map axes `GridLineStyle` property to any line style string recognized by the MATLAB `line` function.

`gridm(`*`PropertyName,PropertyValue`*`,...)` sets the appropriate map axes properties to the desired values. These property names and values are described on the `axesm` reference page of this guide.

Remarks

You can also create or alter map grid properties using the `axesm` or `setm` functions.

See Also

`axesm`	Define map axes and set map properties
`setm`	Set map properties

gridmap

Purpose

Grid a geolocated data grid

Syntax

`[griddedmap, maprefvec] = gridmap(lat, lon, map, cellsize)` grids the geolocated data grid `map`, given geolocation points in `lat` and `lon`, to produce a regular data grid `griddedmap` and the corresponding referencing vector `maprefvec`. `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 `griddedmap` falling outside the area covered by `map` are set to `NaN`s.

Description

`gridmap` provides an easy-to-use alternative to gridding geolocated data grids 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.

Examples

```
% Load the geolocated data grid 'map1'
% and grid it to 1/2-degree cells
load mapmtx
cellsize = 0.5;
[griddedmap, maprefvec] = gridmap(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 grid in subplot 1
subplot(1,2,1)
ax =
axesm('mercator','MapLatLimit',latlim,'MapLonLimit',lonlim,...
  'Grid','on','MeridianLabel','on','ParallelLabel','on');
set(ax,'Visible','off')
surfm(lt1, lg1, map1);
% Display 'griddedmap' as a regular data grid in subplot 2
subplot(1,2,2)
ax =
axesm('mercator','MapLatLimit',latlim,'MapLonLimit',lonlim,...
  'Grid','on','MeridianLabel','on','ParallelLabel','on');
set(ax,'Visible','off')
```

```
meshm(griddedmap, maprefvec);
```

See Also

imbedm, meshm

grid2image

Purpose

Display a regular data grid as an image

Syntax

`grid2image(grid,R)` displays a regular data grid as an image. `grid` can be a matrix of dimension M-by-N or M-by-N-by-3, and can contain double, `uint8`, or `uint16` data. `R` is a 1-by-3 referencing vector defined as [cells/angle units north-latitude west-longitude], or a 3-by-2 referencing matrix, defining a two-dimensional affine transformation from pixel coordinates to spatial coordinates. The displayed map is a Plate Carrée projection, treating longitude as X and latitude as Y. This projection produces significant distortion near the poles.

`grid2image(grid,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.

See Also

`image`, `mapshow`, `mapview`, `meshm`, `surfacem`, `surfm`

Purpose

Convert from Greenwich to equal area coordinates

Syntax

`[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 (0°,0°).

`[x,y] = grn2eqa(lat,lon,origin,ellipsoid)` specifies the two-element ellipsoid vector describing the ellipsoidal model of the figure of the Earth. 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 string. The default value is `'degrees'`.

`mat = grn2eqa(lat,lon,origin...)` packs the outputs into a single variable.

```
mat = grn2eqa(lat,lon,origin...)
```

Description

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`	Convert from equal area to Greenwich coordinates
`hista`	Equal-area histogram

gshhs

Purpose

Read the Global Self-Consistent Hierarchical High-Resolution Shoreline data

Syntax

`struc = gshhs(filename)` reads the specified GSHHS file and extracts data for the entire world. The result is returned as a geographic data structure. Each element of `struc` represents a unique polygon. The tag field for each element contains the topographic level represented by the polygon, and is `'land'`, `'lake'`, `'island'` for an island in a lake, or `'pond'` for a pond on an island in a lake. GSHHS files have filenames of the form `'gshhs_X.b'`, where `X` is one of the letters `'c'`, `'l'`, `'i'`, `'h'`, and `'f'`, corresponding to increasing resolution (and file size).

`struc = gshhs(filename,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. Longitude limits should be between `[–180 195]`.

`gshhs(filename,'createindex')` creates an index file for faster reading. The index file has the same name as the GSHHS data file, but with the extension `'i'` instead of `'b'`. The file is written in the present working directory, which can be identified with the command `pwd`. This file is needed for acceptable performance with the larger data sets. No map data is returned while the index is created.

Background

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.

Example

Read all of the lowest resolution database.

```
s = gshhs('gshhs_c.b')
```

Read the intermediate resolution database for South America.

```
s = gshhs('gshhs_i.b',[-60 -15],[-90 -30])
```

Read the full-resolution file for East and West Falkland Islands (Islas Malvinas).

```
s = gshhs('gshhs_f.b',[-55 -50],[-65 -55])
```

Create the index file for the high-resolution database.

```
gshhs('gshhs_h.b','createindex')
```

Limitations

Polygons with more than 1 million points are treated as line objects rather than patches. This occurs only for the full-resolution file.

Remarks

The GSHHS data in various resolutions is available over the Internet from

```
ftp://ftp.ngdc.noaa.gov/MGG/shorelines
```

Information on the data sets is available from

```
http://www.ngdc.noaa.gov/mgg/shorelines/gshhs.html
```

If you are extracting data within smaller geographic limits, it is much faster to create the index file first, and then extract the data. With very large amounts of data, you might want to plot the data as `NaN`-clipped lines rather than as a very large number of patches. Use `extractm` to combine the data into one vector.

See Also

`extractm`	Extract vector data from geographic data structures
`vmap0data`	Extract selected data from the Vector Map Level 0 CD-ROMs
`dcwdata`	Extract selected data from the Digital Chart of the World
`tgrline`	TIGER/Line data extraction
`tigermif`	Read the TIGER MIF thinned boundary file
`tigerp`	Read the TIGER p and pa thinned boundary file (ArcInfo format)

gtextm

Purpose

Place text on map using mouse

Syntax

`h = gtextm(`*`string`*`)` places the text object string at the position selected by mouse input. When this function is called, the current map axes are brought up and the cursor is activated for mouse-click position entry. The text object's handle is returned.

`h = gtextm(string,`*`PropertyName`*`,`*`PropertyValue`*`,...)` allows the specification of any properties supported by the MATLAB `text` function.

Example

Create map axes:

```
axesm('sinusoid','FEdgeColor','red')
gtextm('hello world','FontWeight','bold')
```

Click inside the frame and the text appears.

See Also

`axesm`	Create map axes object
`textm`	Project text objects

Purpose

Read elevation data from GTOPO30 into a regular data grid

Syntax

`[datagrid,refvec] = gtopo30(`*`filename`*`,scalefactor)` reads the GTOPO30 files and returns a regular data grid. The *`filename`* is given as a string that does not include the extension. If the files are not found on the MATLAB path, they can be selected though a dialog box. `scalefactor` is an integer that when equal to 1 gives the data at its full resolution. When `scalefactor` is an integer number larger than 1, every `n`th point is returned. This scale factor must divide evenly into the number of rows and columns in the data file. The map data is returned as a matrix of elevations and an associated referencing vector.

`[datagrid,refvec] = gtopo30(`*`filename`*`,scalefactor,latlim,lonlim)` allows you to read a subset of the map data. The limits of the desired data are specified as vectors of latitudes and longitudes in degrees. The elements of `latlim` and `lonlim` must be in ascending order.

Background

A global model of elevations on a 30 arc-second grid (approximately 1 km) has been developed by the Earth Resources Observation System Data Center and is available over the Internet. The model uses a variety of international data sources, but is primarily based on raster data from the Digital Terrain Elevation Model (DTED) and vector data from the Digital Chart of the World (DCW). Gridded elevations are computed from DCW elevation contours, points, and drainage lines using ANUDEM, a Digital Elevation Model generator developed by the Australian National University. The data set consists of a total of 21,600 rows and 43,200 columns on 33 tiles, generally covering 40 by 50 degrees. Each tile requires about 15 MB compressed, 80 MB uncompressed, and twice that to extract.

Remarks

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 `NaN`s.

The data is available over the Internet via anonymous FTP from

```
ftp://edcftp.cr.usgs.gov/pub/data/gtopo30/global
```

The data and some documentation is also available over the World Wide Web from

`http://edcwww.cr.usgs.gov/landdaac/gtopo30/gtopo30.html` and

```
http://edcwww.cr.usgs.gov/landdaac/gtopo30/README.html
```

Examples

Extract every 20th point from the tile covering the northeastern United States and eastern Canada. Provide an empty filename and select the file interactively.

```
[datagrid,refvec] = gtopo30([],20);
size(datagrid)
ans =
   300   240
```

Extract a subset of the data for Massachusetts at the full resolution.

```
latlim = [41 42.5]; lonlim = [-73 -69.9];
[datagrid,refvec] = gtopo30('W100N90',1,latlim,lonlim);
size(datagrid)
ans =
   180   391
```

Replace the `NaN`s in the ocean with -1 to color them blue.

```
datagrid(isnan(datagrid)) = -1;
```

Extract the data for Thailand. This area straddles two tiles. The data is on CD number 3 distributed by the USGS.

```
latlim = [5.22 20.90]; lonlim = [96.72 106.38];
gtopo30s(latlim,lonlim)

ans =

    'e060n40'
    'e100n40'

[datagrid,refvec] = gtopo30('f:\',3,latlim,lonlim);
size(datagrid)

ans =

        628 388
```

See Also

`gtopo30s`	Filenames for GTOPO30 data from geographic area
`demdataui`	Digital elevation map data user interface
`dted`	Read Digital Terrain Elevation Data (DTED) data
`globedem`	Read 1 km GLOBE DEM data
`satbath`	Global 2-minute topography from satellite bathymetry
`tbase`	Read data from the TerrainBase data set
`usgsdem`	Read USGS digital elevation maps

gtopo30s

Purpose

Obtain 30-arc-sec resolution DEM file names

Syntax

`fname = gtopo30s(latlim,lonlim)` returns a cell array of the filenames covering the geographic region for GTOPO30 digital elevation maps (also referred to as "30-arc second" DEMs). `latlim` and `lonlim` specify the region as scalar latitude and longitude points, or two-element vectors of latitude and longitude limits in units of degrees.

Remarks

The data is available over the Internet via anonymous FTP from

```
<ftp://edcftp.cr.usgs.gov/pub/data/gtopo30/global>
```

The data and some documentation is also available over the Web from

```
<http://edcdaac.usgs.gov/gtopo30/gtopo30.html>
```

Specifically, see

```
<http://edcdaac.usgs.gov/gtopo30/README.html>
```

See Also

`gtopo30`

Purpose

Get handles of graphics objects

Syntax

`handlem` or `handlem('taglist')` displays a dialog box for selecting the objects for which you want handles.

`h = handlem('prompt')` displays another dialog box, which allows greater control of object selection.

`h = handlem(`*`object`*`)` returns the handles of those objects specified by the input string. The options for the *`object`* string are

`'all'`	All children of the current axes
`'clabel'`	Contour labels on the current map axes
`'contour'`	Contour lines on the current map axes
`'frame'`	Map frame
`'grid'`	Map grid lines
`'hidden'`	Hidden objects on the current axes
`'image'`	Image objects on the current axes
`'light'`	Light objects on the current axes
`'line'`	Line objects on the current axes
`'map'`	All objects on the map, excluding the frame (default)
`'meridian'`	Longitude grid lines
`'mlabel'`	Longitude labels
`'parallel'`	Latitude grid lines
`'patch'`	Patch objects on the current axes
`'plabel'`	Latitude labels
`'surface'`	Surface objects on the current axes
`'text'`	Text objects on the current axes
`'tissot'`	Tissot indicatrices on the current map axes
`'visible'`	Visible objects on the current axes

Or any user-defined object tag string.

A prefix of `'all'` can be applied to strings defining a Handle Graphics object type (`'allimage'`, `'allline'`, `'allsurface'`, `'allpatch'`, `'alltext'`) to determine all object handles that meet the type criteria. Without the `'all'` prefix, those objects named by the user with the `tagm` function are not included (e.g., a line with the tag `'route'` would not be included for object string `'line'`, but would be for `'allline'`).

`handlem('object',axesh)` searches within the axes specified by the input handle `axesh`.

`handlem('object',axesh,'searchmethod')` controls the method used to match the `'str'` input. If omitted, `'exact'` is assumed. Search method `'strmatch'` searches for matches at the beginning of the tag, similar to the MATLAB `STRMATCH` function. Search method `'findstr'` searches within the tag, similar to the MATLAB `FINDSTR` function.

`h = handlem(handles)` returns those elements of an input vector of handles that are still valid.

See Also

`clma`	Clear current map
`clmo`	Clear specified graphics objects
`hidem`	Hide specified graphics objects
`namem`	Determine names of valid graphics objects
`showm`	Show specified graphics objects
`tagm`	Assign a name to graphics object `Tag` property

Purpose

Hide specified graphic object

Syntax

`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`* string, which can be any string recognized by the `handlem` function.

See Also

`clma`	Clear current map
`clmo`	Clear specified graphics objects
`handlem`	Get handle of displayed map objects
`namem`	Determine names of valid graphics objects
`showm`	Show specified graphics objects
`tagm`	Assign name to graphics object `Tag` property

hista

Purpose

Create spatial equal-area histogram

Syntax

`[lat,lon,num] = hista(lats,lons)` returns the center coordinates of equal-area bins and the number of observations falling in each based on the geographically distributed input data.

`[lat,lon,num] = hista(lats,lons,binarea)` specifies the equal-area bin size, in square kilometers. It is 100 km^2 by default.

`[lat,lon,num] = hista(lats,lons,binarea,ellipsoid)` specifies the elliptical definition of the Earth to be used with the two-element `ellipsoid` vector. The default ellipsoid model is a spherical Earth, which is sufficient for most applications.

`[lat,lon,num] = hista(lats,lons,binarea,`*`units`*`)` specifies the standard angle unit string. The default value is `'degrees'`.

Examples

Create random data:

```
lats = rand(4)
lats =
    0.4451    0.8462    0.8381    0.8318
    0.9318    0.5252    0.0196    0.5028
    0.4660    0.2026    0.6813    0.7095
    0.4186    0.6721    0.3795    0.4289

longs = rand(4)
longs =
    0.3046    0.3028    0.3784    0.4966
    0.1897    0.5417    0.8600    0.8998
    0.1934    0.1509    0.8537    0.8216
    0.6822    0.6979    0.5936    0.6449
```

Bin the data in 50-by-50 km cells (2500 sq km):

```
[lat,lon,num] = hista(lats,longs,2500);
[lat lon num]
ans =
    0.2574    0.3757    4.0000
    0.7070    0.3757    5.0000
   -0.1923    0.8253    1.0000
    0.2573    0.8253    2.0000
```

```
    0.7070    0.8254    4.0000
```

See Also

`eqa2grn` `grn2eqa`	Greenwich/equal-area conversion
`histr`	Spatial equirectangular histogram

histr

Purpose

Create spatial equirectangular histogram

Syntax

`[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`*`)` specifies the standard angle unit string. 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.

Description

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., 1°-by-1°). 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

Create random data:

```
lats = rand(4)
lats =
    0.4451    0.8462    0.8381    0.8318
    0.9318    0.5252    0.0196    0.5028
    0.4660    0.2026    0.6813    0.7095
    0.4186    0.6721    0.3795    0.4289

longs = rand(4)
longs =
    0.3046    0.3028    0.3784    0.4966
    0.1897    0.5417    0.8600    0.8998
    0.1934    0.1509    0.8537    0.8216
    0.6822    0.6979    0.5936    0.6449
```

Bin the data in 0.5-by-0.5 degree cells (two bins per degree):

```
[lat,lon,num,wnum] = histr(lats,longs,2);
[lat,lon,num,wnum]
ans =
    0.2500    0.2500    3.0000    3.0000
    0.7500    0.2500    4.0000    4.0003
    0.2500    0.7500    4.0000    4.0000
    0.7500    0.7500    5.0000    5.0004
```

The bins centered at 0.75°N are slightly smaller in area than the others. `wnum` reflects the relative count per normalized unit area.

See Also

`filterm`	Geographic filter for data sets
`hista`	Spatial equal-area histogram

hms2hm

Purpose

Round from hms format to hm format

Syntax

`timeout = hms2hm(timein)` rounds times input in hours-minutes-seconds (*hms*) format to the appropriate value in hours-minutes (*hm*) format. This special handling is needed because there are 60, not 100, seconds in a minute.

Example

Round 12:34:29 and 12:34:31 to hm format:

```
timeout = hms2hm([1234.29 1234.31])
timeout =
        1234        1235
```

See Also

`hms2hr` `sec2hr`	Other direct time conversion functions
`hms2mat`	Convert from hms to separated matrix components
`mat2hms`	Convert from separated matrices to hms format
`timedim`	Convert time units

Purpose

Convert time units from hms format to hours or seconds

Syntax

`timeout = hms2hr(timein)` converts times input in hours-minutes-seconds (*hms*) format to the equivalent measure in decimal hours.

`timeout = hms2sec(timein)` converts times input in hours-minutes-seconds (*hms*) format to the equivalent measure in seconds.

Remarks

The inputs can be in hours-minutes (*hm*) format, since numerically they look like hms format, in which seconds are always zero.

Example

```
hms2hr(1230)
ans =
   12.5000
hms2sec(100.10)
ans =
        3610
```

See Also

`hms2hm`	Round hms format to hm format
`hms2mat`	Convert from hms to separated matrix components
`sec2hr` `hr2hms`	Other direct time conversion functions
`mat2hms`	Convert from separated matrices to hms format
`timedim`	Convert time units

hms2mat

Purpose

Convert the elements of hms format to distinct matrix elements

Syntax

`[h,m,s] = hms2mat(timein)` takes times in hms format and splits their components into three outputs, one each for hours, minutes, and seconds.

`[h,m,s] = hms2mat(timein,n)` specifies the power of 10, `n`, to which the resulting seconds output should be rounded (that is, if a result is 12.567 seconds and `n` = -2, the resulting seconds output would be 12.57). The default value of `n` is -5.

`matout = hms2mat(timein,n)` returns a three-column matrix, `matout`, in which the columns represent hours, minutes, and seconds, respectively. In this case, `timein` must be a vector.

Examples

```
[h,m,s] = hms2mat(1234.567)
h =
    12
m =
    34
s =
   56.7000

matout = hms2mat(1234.567)
matout =
   12.0000   34.0000   56.7000
```

See Also

`mat2hms` Convert from separated matrices to hms format

Purpose

Convert time units from hours to hms or hm

Syntax

`timeout = hr2hms(timein)` converts times input in hours to the equivalent measure in the hours-minutes-seconds (*hms*) format.

`timeout = hr2hm(timein)` converts times input in hours to the equivalent measure in the hours-minutes (*hm*) format. This is the hms format, properly rounded to just hours and minutes.

Example

```
hr2hms(12.51)
ans =
       1230.36

hr2hm(12.51)
ans =
       1231.00
```

See Also

`hms2mat`	Convert from hms to separated matrix components
`sec2hr` `hr2hms`	Other direct time conversion functions
`mat2hms`	Convert from separated matrices to hms format
`timedim`	Convert time units

hr2sec

Purpose

Convert time from hours to seconds

Syntax

`timeout = hr2sec(timein)` converts times input in hours to the equivalent measure in seconds.

Example

```
hr2sec(1)
ans =
        3600
```

See Also

`sec2hr` `hr2hms`	Other direct time conversion functions
`timedim`	Convert time units

Purpose

Encode data points into regular data grid

Syntax

`newmap = imbedm(lat,long,value,map,refvec)` resets certain entries of a regular data grid, `map`. The entries to be reset correspond to the locations defined by the latitude and longitude position vectors `lat` and `lon`. The entries are reset to the same number if `value` is a scalar, or to individually specified numbers if `value` is a vector the same size as `lat` and `lon`. If any points lie outside the input `map`, a warning is displayed.

`newmap = imbedm(lat,lon,value,map,refvec,`*`units`*`)` specifies the units of the vectors `lat` and `lon`, where *`units`* is any valid angle units string (`'degrees'` by default).

`[newmap,badindx] = imbedm(lat,lon,value,map,refvec,`*`units`*`)` returns the indices of `lat` and `lon` corresponding to points outside the `map` in the variable `badindx`.

Examples

Create a simple map and embed new values in it:

```
map = ones(3,6)
map =
     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

newmap = imbedm([23 -23], [45 -45],[5 5],map,refvec)
newmap =
     1     1     1     1     1     1
     1     1     5     5     1     1
     1     1     1     1     1     1
```

See Also

`ltln2val`	Latitude and longitude to matrix entry value
`setpostn`	Latitude and longitude to row and column

ind2rgb8

Purpose

Convert an indexed image to a uint8 RGB image.

Syntax

`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.

Example

```
% Convert the 'boston.tif' image to RGB.
[X, cmap, R, bbox] = geotiffread('boston.tif');
RGB = ind2rgb8(X, cmap);
mapshow(RGB, R);
```

See also

`ind2rgb`

Purpose

Select latitude and longitude coordinates using mouse

Syntax

`[lat,long] = inputm` returns the latitude-longitude point of a mouse-selected point of the current map axes.

`[lat,long] = inputm(npts)` specifies the number of points, `npts`, to be mouse-selected.

`[lat,long] = inputm(npts,hndl)` specifies the handle of the map axes on which mouse selection is desired.

`pts = inputm(npts)` returns the points in a single two-column output.

Remarks

`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.

See Also

`gcpmap`	Map current point
`ginput`	Request user input (see the online MATLAB Function Reference documentation)

interpm

Purpose

Linearly interpolate latitude and longitude data to a given resolution

Syntax

`[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 method strings 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 units string. The default is `'degrees'`.

Examples

```
lat = [1 2 4 5]; lon = [7 8 9 11];
[latout,lonout] = interpm(lat,lon,1);
[latout lonout]
ans =
    1.0000    7.0000
    2.0000    8.0000
    3.0000    8.5000
    4.0000    9.0000
    4.5000   10.0000
    5.0000   11.0000
```

See Also

`intrplat`	Interpolated latitudes for given longitudes
`intrplon`	Interpolated longitudes for given latitudes

Purpose

Interpolate latitude for a given longitude

Syntax

`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. The default value of the *`method`* string is `'linear'`, which results in linear, or Cartesian, interpolation between the numerical values entered. This is really just a pass-through to the MATLAB `interp1` function. Similarly, `'spline'` and `'cubic'` perform cubic spline and cubic interpolation, respectively. The `'rh'` method returns interpolated points that lie on rhumb lines between input data. Similarly, the `'gc'` method 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. The default is `'degrees'`.

Description

The function `intrplat` is a geographic data analogy of the standard MATLAB function `interp1`.

Examples

Compare the results of the various methods:

```
lats = [25 45]; longs = [30 60];
newlat = intrplat(longs,lats,45,'linear')
newlat =
    35

newlat = intrplat(longs,lats,45,'rh')
newlat =
   35.6213

newlat = intrplat(longs,lats,45,'gc')
newlat =
   37.1991
```

Remarks

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

`interpm`	Linear interpolation of latitude and longitude
`intrplon`	Interpolate longitudes for given latitudes

Purpose

Interpolate longitude for a given latitude

Syntax

`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. The default value of the *`method`* string is `'linear'`, which results in linear, or Cartesian, interpolation between the numerical values entered. This is really just a pass-through to the MATLAB `interp1` function. Similarly, `'spline'` and `'cubic'` perform cubic spline and cubic interpolation, respectively. The `'rh'` method returns interpolated points that lie on rhumb lines between input data. Similarly, the `'gc'` method 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. The default is `'degrees'`.

Description

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
```

Remarks

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

`interpm`	Linear interpolation of latitude and longitude
`intrplat`	Interpolate longitudes for given longitude

Purpose

Test whether axes have a map definition

Syntax

`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 a string message if the axes is not a map axes, specifying why not.

Description

The `ismap` function tests an axes object to determine whether it is a map axes.

See Also

`gcm`	Get current map data structure
`ismapped`	Test whether object is projected on map axes

ismapped

Purpose

Test whether object is projected on map axes

Syntax

`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 a string message if the axes is not projected on a map axes, specifying why not.

Description

The `ismapped` function tests an object to determine whether it is projected on map axes.

See Also

`gcm`	Get current map data structure
`ismap`	Test whether axes have a map definition

Purpose

Convert distance from kilometers to other units

Syntax

```
distout = km2deg(distin)
distout = km2deg(distin,radius)

distout = km2nm(distin)

distout = km2rad(distin)
distout = km2rad(distin,radius)

distout = km2sm(distin)
```

Description

`distout = km2deg(distin)` converts the input distance given in kilometers to degrees. `distout = km2nm(distin)`, `distout = km2rad(distin)`, and `distout = km2sm(distin)` perform analogously, converting to nautical miles, radians, and statute miles, respectively.

`distout = km2deg(distin,radius)`and `distout = km2rad(distin,radius)` specify the radius of the sphere to use, because a degree (or radian) of arc length covers less distance, for example, on Mars than it does on the Earth. You can enter the radius as a number in kilometers, as a call to the `almanac` function (e.g., `almanac('mars','radius','km')`), or you can pass in a string planet name (e.g., `'mars'`), and the function will make the appropriate call to the `almanac` function. The radius of the Earth is the default.

Examples

How many miles is a *10k run*?

```
distout = km2sm(10)
distout =
    6.2139
```

See Also

`distdim`	Convert distance units
`nm2km` `sm2deg`	Other direct distance conversion functions

latlon2pix

Purpose

Convert latitude-longitude coordinates to pixel coordinates

Syntax

`[row, col] = latlon2pix(R,lat,lon)` calculates pixel coordinates `row`, `col` from latitude-longitude coordinates `lat`, `lon`. `R` is a 3-by-2 referencing matrix defining a two-dimensional affine transformation from pixel coordinates to spatial coordinates. `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.

Description

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.

Example

```
% Find the pixel coordinates of the upper left and lower right
% outer corners of a 2-by-2 degree gridded data set.
R = makerefmat(1, 89, 2, 2);
[UL_row, UL_col] = latlon2pix(R,  90, 0)     % Upper left
[LR_row, LR_col] = latlon2pix(R, -90, 360)   % Lower right
[LL_row, LL_col] = latlon2pix(R, -90, 0)     % Lower left
% Note that the in both the 2nd case and 3rd case we get a column
% value of 0.5, because the left and right edges are on the same
% meridian and (-90, 360) is the same point as (-90, 0).
```

See Also

`makerefmat`, `pix2latlon`, `map2pix`

Purpose

Determine courses and distances between navigational track waypoints

Syntax

`[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 string *`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)` and `[course,dist] = legs(pts,`*`method`*`)` allow you to input the waypoints in a single two-column matrix, `pts`.

`mat = legs(lat,...)` packs up the outputs into a single two-column matrix, `mat`.

Description

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 (42.3°N,71°W) and traveling to LAX in Los Angeles (34°N,118°W). The pilot wants to file a flight plan that takes the plane over O'Hare Airport in Chicago (42°N,88°W) 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 269° for 756 nautical miles, then alter course to 251° 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`	Dead reckon points for a track
`gcwaypts`	Find waypoints along a great circle
`navfix`	Mercator-based navigational fixing
`track`	Connect waypoints

Purpose

Project light objects on the current map axes

Syntax

`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

```
load topo
axesm globe; view(120,30)
meshm(topo,topolegend); demcmap(topo)
lightm(0,90,'color','yellow')
material([.5 .5 1]); lighting phong
```

See Also

`light`	Create light (see the online MATLAB Function Reference documentation)
`lightmui`	GUI to control position of lights on a globe or 3-D map

lightmui

Purpose

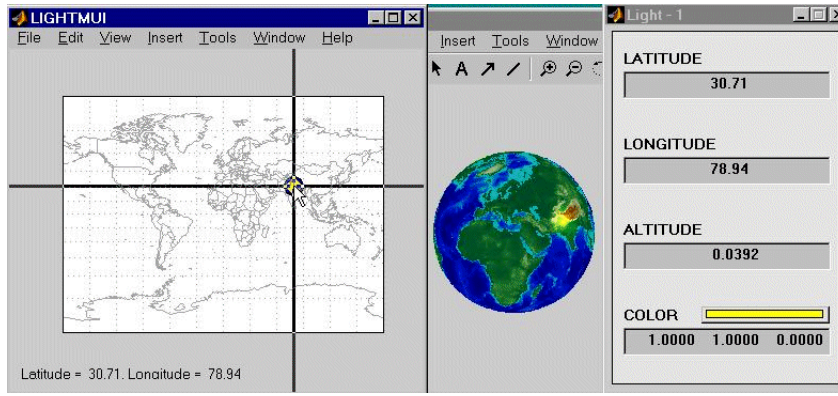
GUI to control position of lights on a globe or 3-D map

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` Project light objects on the current map axes

Purpose

Determine latitude and longitude limits of a regular data grid

Syntax

`[latlimits,lonlimits] = limitm(map,refvec)` returns two-element limit vectors `latlimits` and `lonlimits` describing the extremes of the input regular data grid with a legend vector `refvec`.

`latlimits` and `lonlimits` are of the form `[south-limit north-limit]` and `[west-limit east-limit]`, respectively. All elements are in degrees, because this function deals only with regular data grids.

`limvec = limitm(map,refvec)` returns a single four-element output vector of the form `[south-limit north-limit west-limit east-limit]`.

Examples

Using a familiar data grid,

```
load topo
[latlimits,lonlimits] = limitm(topo,topolegend)
latlimits =
   -90    90
lonlimits =
     0   360
```

Which is expected, because `topo` covers the whole globe.

See Also

`nanm`
`onem`
`spzerom`
`zerom`
	Create special data grids

`sizem`	Row and column dimensions needed for data grid

linecirc

Purpose

Find the intersections of a circle and a line in Cartesian space

Syntax

`[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`

Purpose

Project line objects onto current map axes

Syntax

`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,`*`linetype`*`)` allows the specification of the line style, where *`linetype`* is any string recognized by the MATLAB `line` function.

`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`.

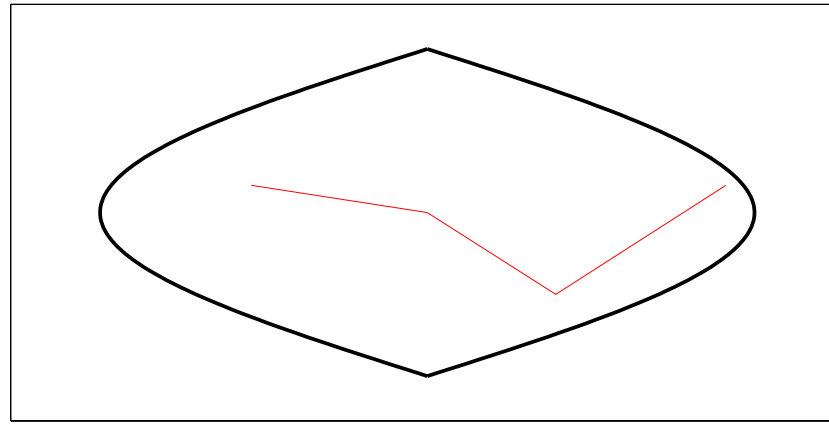
`h = linem(lat,lon,z)` displays a line object in three dimensions, where `z` is the same size as `lat` and `lon` and contains the desired altitude data. `z` is independent of `AngleUnits`. If omitted, all points are assigned a *z*-value of 0 by default.

Description

`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-')
```



See Also

`line`	Create line (see the online MATLAB Function Reference documentation)
`plot3m`	Project lines onto current map axes in 3-D space
`plotm`	Project 2-D lines onto current map axes

Purpose

Line of sight visibility between two points in terrain

Syntax

`vis = los2(map,refvec,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 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 1 when the two points are visible to each other, and 0 when the line of sight is obscured by terrain. If any of the input arguments is 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.

`vis = los2(map,refvec,lat1,lon1,lat2,lon2,alt1)` places the first point at the specified altitude in meters above the surface. This is equivalent to putting the point on a tower. If omitted, the point is assumed to be on the surface.

`vis = los2(map,refvec,lat1,lon1,lat2,lon2,alt1,alt2)` also places the second point at a specified altitude in meters above the surface. If omitted, the point is assumed to be on the surface.

`vis = los2(map,refvec,lat1,lon1,lat2,lon2,alt1,alt2,`*`alt1opt`*`)` controls whether the first point is at a relative or absolute altitude. If the altitude option is `'AGL'`, the point's altitude is in meters above the terrain ground level. If *`alt1opt`* is `'MSL'`, `alt1` is interpreted as altitude above zero, or mean sea level. If omitted, `'AGL'` is assumed.

`vis = los2(map,refvec,lat1,lon1,lat2,lon2,...`

`alt1,alt2,`*`alt1opt`*`,`*`alt2opt`*`)` also controls the interpretation of the second point's altitude.

`vis = los2(map,refvec,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, 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.

`vis = los2(map,refvec,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 4/3rds 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, the elevations, and the radii should be in the same units.

`[vis,visprofile,dist,z,lattrk,lontrk] = los2(...)` also returns vectors of points along the path between the two points. The `visprofile` is a vector containing 1's where the intermediate points are visible. `dist` is the distance along the path (in meters or the units of the radius). `z` contains the terrain profile relative to the z datum along the path. `lattrk` and `lontrk` are the latitudes and longitudes of the points along the path.

`los2(...)`, with no output arguments, displays the visibility profile between the two points in a new figure.

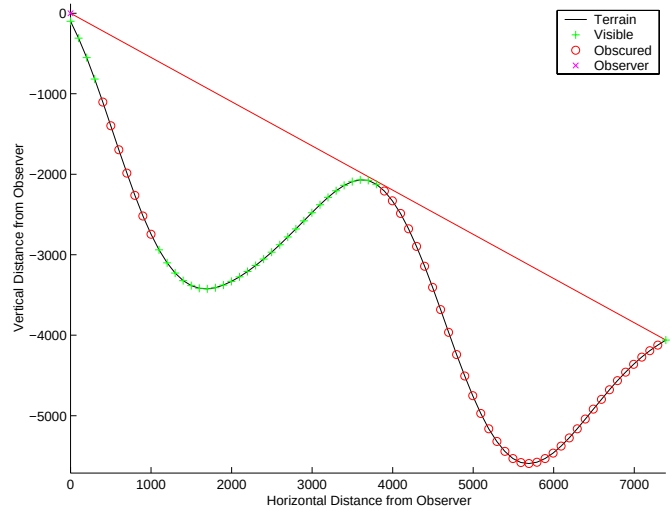
Description

`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 map's `zdata` is used for the profile. The color data is used in the absence of data in `z`. Select the two points by clicking 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.

Example

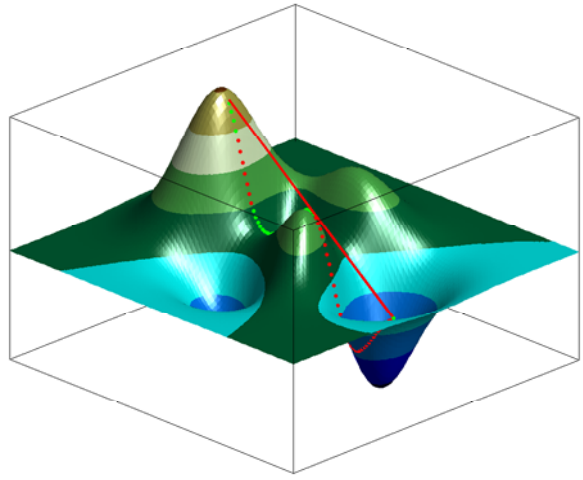
Check the visibility between two points on the peaks map.

```
map = 500*peaks(100);
refvec = [ 1000 0 0];
[lat1,lon1,lat2,lon2]=deal(-0.027,0.05,-0.093,0.042);

los2(map,refvec,lat1,lon1,lat2,lon2,100)

ans =

     1
```



Display the same data together with the `peaks` surface.

```
figure
axesm('globe','geoid',almanac('earth','sphere','meters'))
meshm(map,refvec,size(map),map); axis tight
camposm(-10,-10,1e6); camupm(0,0)
demcmap('inc',map,1000); shading interp; camlight

[vis,visprofile,dist,z,lattrk,lontrk] =
los2(map,refvec,lat1,lon1,lat2,lon2,100);
plot3m(lattrk([1;end]),lontrk([1; end]),z([1; end])+[100;
0],'r','linewidth',2)
plotm(lattrk(~visprofile),lontrk(~visprofile),z(~visprofile),'r.
','markersize',10)
plotm(lattrk(visprofile),lontrk(visprofile),z(visprofile),'g.','
markersize',10)
```



See Also

`viewshed`	Compute visible areas from a point on a digital elevation map
`mapprofile`	Interpolate values between waypoints on a regular data grid

Purpose

Determine data grid entries or interpolated values associated with latitude-longitude points

Syntax

`value = ltln2val(map,refvec,lat,lon)` returns the values of the regular data grid `map` corresponding to the locations specified by the vectors `lat` and `lon`.

`value = ltln2val(map,refvec,lat,lon,`*`method`*`)` specifies the method for determining the returned value. The default *`method`* is `'nearest'`, which returns the unaltered value of the cell containing the coordinates `lat` and `lon`. Using a *`method`* of `'linear'` or `'cubic'` results in values that are linearly and cubically interpolated between cells, respectively.

Examples

Find the elevations in `topo` associated with three European cities — Milan, Bern, and Prague (`topo` elevations are in meters):

```
load topo
```

The city locations, [Milan Bern Prague],

```
lats = [45.45; 46.95; 50.1];
longs = [9.2; 7.4; 14.45];

elevations = ltln2val(topo,topolegend,lats,longs)
elevations =
   313
   1660
   297
```

See Also

`findm` Coordinates of nonzero map entries

majaxis

Purpose

Calculate semimajor axis from semiminor axis and eccentricity

Syntax

`semimajor = majaxis(semiminor,eccentricity)` returns the semimajor axis length corresponding to the input semiminor axis and eccentricity.

`semimajor = majaxis([semiminor eccentricity])` allows the inputs to be packed into a single two-column input of the form `[semiminor eccentricity]`.

Description

The semimajor axis, the first element of the standard ellipsoid vector in the Mapping Toolbox, can be determined given both the semiminor axis and the eccentricity.

Examples

Using the default values for the Earth,

```
semimajor = majaxis(6356.7523,0.0818192)
semimajor =
   6.3781e+03
```

This is the default semimajor axis.

See Also

`almanac`	Planetary data
`axes2ecc` `minaxis`	Related conversion functions

Purpose

Make an object a mapped object

Syntax

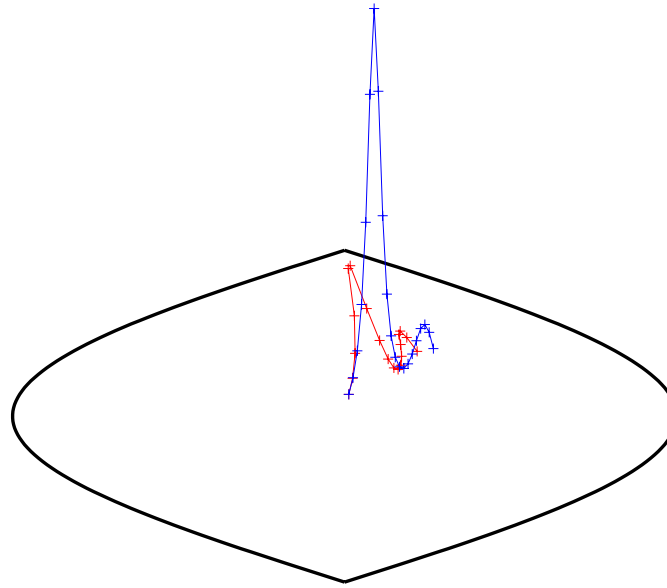
`makemapped(h)` adds a Mapping Toolbox structure to the displayed objects associated with `h`. `h` can be a handle, vector of handles, or any name string recognized by `handlem`. The objects are then considered to be geographic data. Objects extending outside the map frame should first be trimmed to the map frame using `trimcart`.

Background

The Mapping Toolbox identifies displayed objects that are projected from geographic coordinates by a special structure in that object's `UserData` property. Objects created using standard MATLAB display functions lack this structure, and retain the same Cartesian coordinates, regardless of the projection. This function adds the structure that makes an object mapped. The coordinates are unchanged in the process, but will change if the map projection is modified.

Examples

```
axesm('miller','geoid',[25 0])
framem
plot(humps,'b+-')

h = plot(humps,'r+-');
trimcart(h)
makemapped(h)

setm(gca,'MapProjection','sinusoid')
```



Remarks

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

`trimcart`	Trim graphic objects to the map frame
`handlem`	Return the graphics handle for identified objects
`cart2grn`	Transform from projected Cartesian coordinates to Greenwich frame

Purpose

Construct an affine spatial-referencing matrix

Description

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 third syntax below. This is one of the few places where longitude precedes latitude in a function call.

Syntax

`R = makerefmat(x11, y11, dx, dy)` with scalar `dx` and `dy` constructs a referencing matrix that aligns image/data grid rows to map `x` and columns to map `y`. `x11` and `y11` are scalars that specify the map location of the center of the first (1,1) pixel in the image or first element of the data grid, so that

```
[x11 y11] = pix2map(R,1,1)
```

`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. More abstractly, R is defined such that

```
[x11 + (col-1) * dx, y11 + (row-1) * dy] = pix2map(R, row, col)
```

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, for which

```
[x11 + ([row col]-1) * dx(:), y11 + ([row col]-1) * dy(:)] ...
 = pix2map(R, row, col)
```

Remarks

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 `[dx1) 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 scalar `dy` 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. In this case,

```
[lat11,lon11] = pix2geo(R,1,1),
[lat11+(row-1)*dlat,lon11+(col-1)*dlon] = pix2geo(R,row,col)
```

for scalar `dlat` and `ddlon`, and

```
[lat11+[row col]-1)*dlat,lon11+([row col]-1)*dlom] = ...
pix2geo(R, row,col)
```

for vector `dlat` and `dlon`. Note that images or data grids aligned with latitude and longitude might already have referencing vectors. In this case you can use function `refvec2mat` to convert to a referencing matrix.

Examples

Example 1

```
% 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)
```

Example 2

```
% Create a referencing matrix for a global geoid grid.

load geoid  % Adds array 'geoid' to the workspace

% 'geoid' contains a model of the Earth's geoid sampled in
% one-degree-by-one-degree cells.  Each column of 'geoid'
% contains geoid heights in meters for 180 cells starting at
% latitude -90 degrees and extending to +90 degrees, for a
% given latitude.
% Each row contains geoid heights for 360 cells starting at
% longitude 0 and extending 360 degrees.

lat11 = -89.5;  % Cell-center latitude corresponding to geoid(1,1)
lon11 =   0.5;  % Cell-center longitude corresponding to
geoid(1,1)
dLat = 1;  % From row to row moving north by one degree
dLon = 1;  % From column to column moving east by one degree
geoidR = makerefmat(lon11, lat11, dLon, dLat)

% It's well known that at its most extreme the geoid reaches
% a minimum of slightly less than -100 meters, and that the
% minimum occurs in the Indian Ocean at approximately
% 4.5 degrees latitude, 78.5 degrees longitude. Check the
% geoid height at this location by using LATLON2PIX with
% the new referencing matrix:

[row, col] = latlon2pix(geoidR, 4.5, 78.5)
geoid(round(row),round(col))
```

Example 3

```
% Create a half-resolution version of a georeferenced TIFF
% image, using Image Processing Toolbox functions IND2GRAY
% and IMRESIZE.

% Read the indexed-color TIFF image and convert it to grayscale.
% The size of the image is 2000-by-2000.
[X, cmap] = imread('1_209910_sub.tif');
I_orig = ind2gray(X, cmap);

% Read the corresponding worldfile.  Each image pixel covers a
% one-meter square on the map.
R_orig = worldfileread('1_209910_sub.tfw')

% Halve the resolution, creating a smaller (1000-by-1000) image.
I_half = imresize(I_orig, size(I_orig)/2, 'bicubic');

% Find the map coordinates of the center of pixel (1,1) in the
% resized image: halfway between the centers of pixels (1,1) and
% (2,2) in the original image.
[x11_orig, y11_orig] = pix2map(R_orig, 1, 1)
[x22_orig, y22_orig] = pix2map(R_orig, 2, 2)

% Average these to determine the center of pixel (1,1) in the new
% image.
x11_half = (x11_orig + x22_orig) / 2
y11_half = (y11_orig + y22_orig) / 2

% Make a referencing matrix for the new image, noting that its
% pixels are each two meters square.
R_half = makerefmat(x11_half, y11_half, 2, -2)

% Display each image in map coordinates.
figure;
subplot(2,1,1); h1 = mapshow(I_orig,R_orig); ax1 =
get(h1,'Parent');
subplot(2,1,2); h2 = mapshow(I_half,R_half); ax2 =
get(h2,'Parent');
set(ax1, 'XLim', [208000 208250], 'YLim', [911800 911950])
set(ax2, 'XLim', [208000 208250], 'YLim', [911800 911950])
```

```
% Mark the same map location on top of each image.
x = 208202.21;
y = 911862.70;
line(x, y, 'Parent', ax1, 'Marker', '+', 'MarkerEdgeColor', 'r');
line(x, y, 'Parent', ax2, 'Marker', '+', 'MarkerEdgeColor', 'r');

% Graphically, they coincide, even though the same map location
% corresponds to two different pixel coordinates.
[row1, col1] = map2pix(R_orig, x, y)
[row2, col2] = map2pix(R_half, x, y)
```

See Also

latlon2pix, map2pix, pix2latlon, pix2map, refvec2mat, worldfileread, worldfilewrite

makesymbolspec

Purpose

Construct layer symbolization specification

Syntax

`symbolspec = makesymbolspec(geometry,rule1,rule2,...ruleN)` constructs a symbol specification structure (`symbolspec`) for symbolizing a (vector) shape layer in the Map Viewer or when using `mapshow`. `geometry` is one of `'Point'`, `'Line'`, `'PolyLine'`, `'Polygon'`, or `'Patch'`. Rules, defined in detail below, specify the graphics properties for each feature of the layer. A rule can be a default rule that is applied to all features in the layer or it may limit the symbolization to only those features that have a particular value for a specified attribute. Features that do not match any rules are displayed using the default graphics properties.

To create a rule that applies to all features, a default rule, use the following syntax:

```
{'Default',Property1,Value1,Property2,Value2,...
           PropertyN,ValueN}
```

To create a rule that applies only to features that have a particular value or range of values for a specified attribute, use the following syntax:

```
{AttributeName,AttributeValue,
Property1,Value1,Property2,Value2,...,PropertyN,ValueN}
```

`AttributeValue` and `ValueN` can each be a two-element vector, `[low high]`, specifying a range. If `AttributeValue` is a range, `ValueN` might or might not be a range.

The following is a list of allowable values for `PropertyN`.

- Points or Multipoints: `'Marker'`, `'Color'`, `'MarkerEdgeColor'`, `'MarkerFaceColor'`, `'MarkerSize'`, and `'Visible'`
- Lines or PolyLines: `'Color'`, `'LineStyle'`, `'LineWidth'`, and `'Visible'`
- Polygons: `'FaceColor'`, `'FaceAlpha'`, `'LineStyle'`, `'LineWidth'`, `'EdgeColor'`, `'EdgeAlpha'`, and `'Visible'`

Examples

The following examples import a shapefile containing road data and symbolize it in several ways using symbol specifications.

Example 1 — Default Color

```
roads = shaperead('concord_roads.shp');
```

```
blueRoads = makesymbolspec('Line',{'Default','Color',[0 0 1]});
mapshow(roads,'SymbolSpec',blueRoads);
```

Example 2 — Discrete Attribute Based

```
roads = shaperead('concord_roads.shp');
roadColors = ...
makesymbolspec('Line',{'CLASS','Primary','Color','r'},...
                      {'CLASS','Secondary','Color','g'},...
                      {'CLASS','Improved','Color','y'},...
                      {'Default','Color','k'});
mapshow(roads,'SymbolSpec',roadColors);
```

Example 3 — Using a Range of Attribute Values

```
roads = shaperead('concord_roads.shp');
lineStyle = makesymbolspec('Line',...
 {'ID',[0 5], 'LineStyle',':'},...
 {'ID',[6 10],'LineStyle','-.'});
mapshow(roads,'SymbolSpec',lineStyle);
```

Example 4 — Using a Range of Attribute Values and a Range of Property Values

```
roads = shaperead('concord_roads.shp');
colorRange = makesymbolspec('Line',...
 {'ID',[1 10],'Color',jet(10)});
mapshow(roads,'SymbolSpec',colorRange);
```

See Also

mapshow, geoshow, mapview

map2pix

Purpose

Convert map coordinates to pixel coordinates

Syntax

`[row,col] = map2pix(R,x,y)` calculates pixel coordinates `row, col` from map coordinates `x,y`. `R` is a 3-by-2 referencing matrix defining a two-dimensional affine transformation from pixel coordinates to map coordinates. `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.

Example

```
% Find the pixel coordinates for the spatial coordinates
 % (207050, 912900)
R = worldfileread('concord_ortho_w.tfw');
[r,c] = map2pix(R, 207050, 912900);
```

See Also

`latlon2pix`, `makerefmat`, `pix2map`, `worldfileread`

Purpose

Compute bounding box of a georeferenced image or data grid

Syntax

`bbox = mapbbox(R, height, width)` computes the 2-by-2 bounding box of a georeferenced image or regular gridded data set. `R` is a 3-by-2 affine referencing matrix. `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 struct array with the fields

`'RefMatrix'`	3-by-2 referencing matrix
`'Height'`	Scalar number
`'Width'`	Scalar number

See Also

`geotiffinfo`, `makerefmat`, `mapoutline`, `pixcenters`, `pix2map`

maplist

Purpose

List the map projections available in the Mapping Toolbox

Syntax

`list = maplist` returns a structure that defines the map projections available in the Mapping Toolbox. The list structure is `list.Name`, `list.IdString`, `list.Classification`, `list.ClassCode`. This list structure is used by the functions `maps` and `axesmui` when processing map projection identifiers during operation of the toolbox functions.

`[list,defproj] = maplist` also returns the `IdString` of the default projection.

`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` defines the name of the M-file that computes the projection.

`list.Classification` defines the projection classification that is used in the command-line table display.

`list.ClassCode` defines the character string 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

If map projections are to be added to the toolbox, the list structure must be extended and the appropriate field data entered. For example, if a new projection is added to the default list, then a new entry in the list structure would be

```
list.Name(61)           = 'My Projection'
list.IdString(61)       = 'newprojection';
list.Classification(61) = 'New Projection Type';
list.ClassCode(61)      = 'Code';
```

See Also

`maps`, `axesmui`

Purpose

Compute outline of a georeferenced image or data grid

Syntax

`[x,y] = mapoutline(R, height, width)` computes the outline of a georeferenced image or regular gridded data set in map coordinates. `R` is a 3-by-2 affine referencing matrix. `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 struct 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

Example 1

Draw an outline delineating a TIFF image with a world file

```
R = worldfileread('concord_ortho_w.tfw');
info = imfinfo('concord_ortho_w.tif');
[x,y] = mapoutline(R, info.Height, info.Width);
plot(x,y)
```

Example 2

Draw an outline delineating a GeoTIFF image

```
info = geotiffinfo('boston.tif');
```

```
[x,y] = mapoutline(info, 'close');
plot(x,y)
```

See Also makerefmat, mapbbox, pixcenters, pix2map

Purpose

Values between waypoints on a regular data grid

Syntax

`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 map's zdata is used for the profile. The color data is used in the absence of data in `z`. The result is displayed in a new figure.

`[z,rng,lat,lon] = mapprofile` returns the values of the profile without displaying them. The output `z` contains interpolated values from `map` along great circles between the waypoints. `rng` 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.

`[z,rng,lat,lon] = mapprofile(map,refvec,lat,lon)` uses the provided regular data grid and waypoint vectors. No displayed map is required. Sets of waypoints can be separated by `NaN`s into line sequences. The output ranges are measured from the first waypoint within a sequence.

`[z,rng,lat,lon] = mapprofile(map,refvec,lat,lon,rngunits)` specifies the units of the output ranges along the profile. Valid range units inputs are any distance string recognized by `distdim`. Surface distances are computed using the default radius of the earth. If omitted, `'degrees'` is assumed.

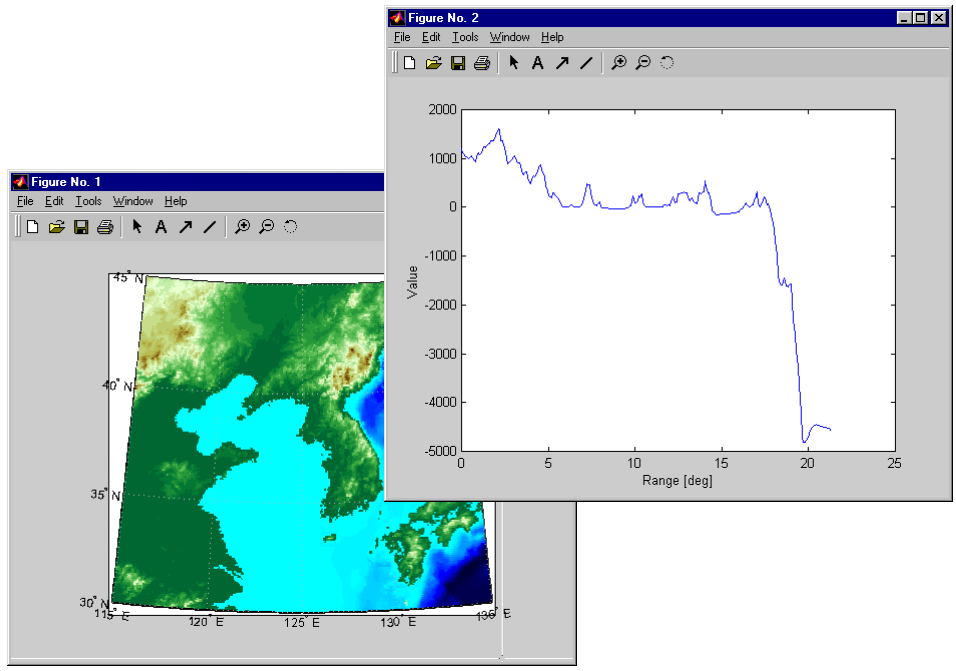
`[z,rng,lat,lon] = mapprofile(map,refvec,lat,lon,ellipsoid)` uses the provided ellipsoid definition in computing the range along the profile. The ellipsoid vector is of the form `[semimajor axes, eccentricity]`. The output range is reported in the same distance units as the semimajor axes of the ellipsoid vector. If omitted, the range vector is for a sphere.

```
[z,rng,lat,lon] = ...
mapprofile(map,refvec,lat,lon,rngunits,trackmethod,interpmethod)
```
and
```
[z,rng,lat,lon] = ...
mapprofile(map,refvec,lat,lon,ellipsoid,trackmethod,interpmethod)
```
control the interpolation methods used. Valid `trackmethods` are `'gc'` for great circle tracks between waypoints, and `'rh'` for rhumb lines. Valid `interpmethods` for interpolation within the data grid are `'bilinear'` for linear interpolation, `'bicubic'` for cubic interpolation, and `'nearest'` for nearest neighbor interpolation. If omitted, `'gc'` and `'bilinear'` are assumed.

Example

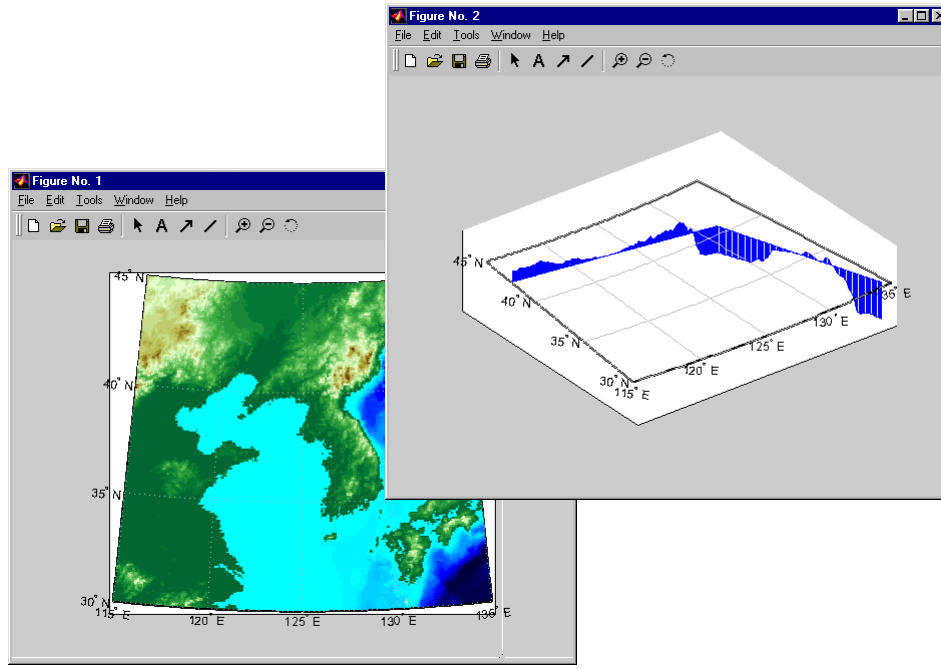
What is the elevation profile across the sample Korean digital elevation data? Take the data and the waypoints from a map display. Click the upper left and the lower right corners of the map, and then press the **Enter** key.

```
load korea
worldmap(map,refvec,'dem')
mapprofile
```



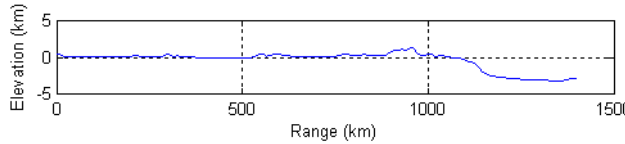
When you select more than two waypoints, the automatically generated figure displays the result in three dimensions. The following example shows the relative sizes of the mountains in northern China compared to the depths of the Sea of Japan.

```
close
mapprofile
```



Adding output arguments suppresses the display of the results in a new figure. You can then use the results in further calculations or display the results yourself. Here the profile from the upper left to lower right is computed from waypoints selected on the map. The ranges and elevations are converted to kilometers and displayed in a new figure. The vertical exaggeration factor is set to 20. With no vertical exaggeration, the changes in elevation would be almost too small to see.

```
[z,rng,lat,lon] = mapprofile;
figure
plot(deg2km(rng),z/1000)
daspect([ 1 1/20 1 ]); grid
xlabel 'Range (km)'
ylabel 'Elevation (km)'
```

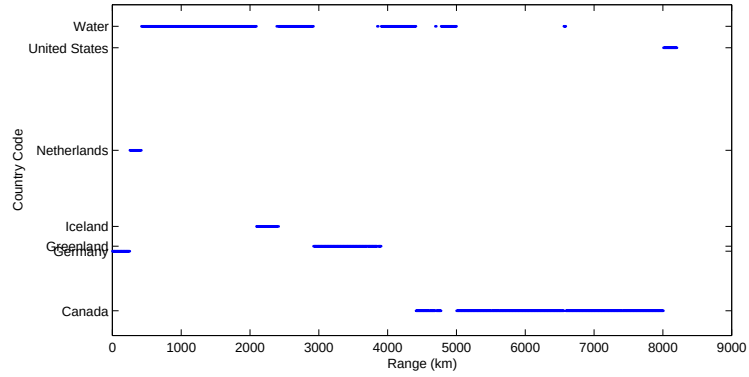


You can compute values along a path without reference to an existing figure by providing a regular data grid and vectors of waypoint coordinates. Optional arguments allow control over the units of the range output and interpolation methods between waypoints and data grid elements.

Find the countries that lie under a great circle track from Frankfurt to Seattle. Use the 15 minute (25 kilometer) `worldmtxmed` political map data.

```
close all; clear all;
load worldmtxmed
[lat,lon] =
extractm(worldlo('PPpoint'),{'Seattle','Frankfurt'});
lat(isnan(lat)) = []; lon(isnan(lon)) = [];
[lat lon]

ans =

   50.1082    8.6732
   47.5070 -122.3589


[valp,rngp,latp,lonp] =
mapprofile(double(map),refvec,lat,lon,'km','gc','nearest');

ucodes = unique(valp)

ucodes =

    35
    71
    74
    86
   132
```

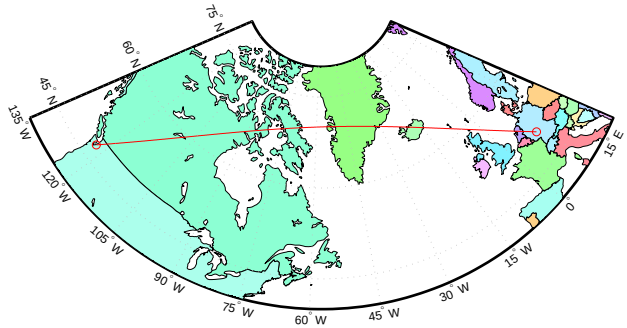
```
   194
   207

unames = names(ucodes)

unames =

    'Canada'
    'Germany'
    'Greenland'
    'Iceland'
    'Netherlands'
    'United States'
    'Water'



plot(rngp,valp,'.')
xlabel 'Range (km)'
ylabel 'Country Code'
set(gca,'ytick',ucodes)
set(gca,'yticklabel',unames)
```



Verify the result by displaying the track on a map.

```
worldmap([40 80],[-135 20],'patch')
```

```
plotm(latp,lonp,'r')
plotm(lat,lon,'ro')
hidem(gca)
```



See Also

`ltln2val`	Values for position on a regular data grid
`los2`	Line of sight between two points

Purpose

List projection names or projection codes

Syntax

`maps` displays in the Command Window a table describing all projections available for use.

`strmat = maps('namelist')` returns the English names for the available projections as a matrix of strings.

`strmat = maps('idlist')` returns the standard projection identification strings for the available projections as a matrix of strings.

`stdstr = maps(`*`id_string`*`)` returns the specific standard projection identification string associated with a unique truncation 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 are the strings used, for example, when setting the `axesm` property `MapProjection`.

The functions `setm` and `axesm` recognize unique abbreviations (truncations) of these strings. The `maps` function can be used to convert such an abbreviation to the standard ID string:

```
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`	Create map axes
`setm`	Modify the properties of a displayed map

Purpose

Display map data without projection

Syntax

`mapshow(s)` displays the graphic features stored in the geographic data structure `s`. If `s` includes X and Y fields, then they are used directly to plot features in map coordinates. If Lat and Lon fields are present instead, then their coordinate values are projected to map coordinates if the axes has a projection. Otherwise, Lon is plotted as X and Lat as Y.

```
mapshow(s,Property1,Value1,Property2,Value2,...)
```

`mapshow(x,y)` or `mapshow(x,y, ..., 'DisplayType', displaytype, ...)` displays the equal-length coordinate vectors `x` and `y`. `x` and `y` can contain embedded NaNs, delimiting coordinates of lines or polygons. `displaytype` can be `'point'`, `'line'`, or `'polygon'` and defaults to `'line'`.

`mapshow(x,y,z, ..., 'DisplayType', displaytype, ...)` where `x` and `y` are M-by-N coordinate arrays, `z` is an M-by-N array of class double, and `displaytype` is `'surface'`, `'mesh'`, `'texturemap'`, or `'contour'`, displays a geolocated data grid, `z`. `z` can contain `NaN` values.

```
mapshow(x,y,I)
mapshow(x,y,BW)
mapshow(x,y,A,cmap)
mapshow(x,y,RGB)
```

where `I` is an intensity image, `BW` is a logical image, `A` is an indexed image with colormap `cmap`, or `RGB` is a true-color image, displays a geolocated image. The image is rendered as a texture map on a zero-elevation surface. If specified, `'DisplayType'` must be set to `'image'`. Examples of geolocated images include a color composite from a satellite swath or an image originally referenced to a different coordinate system.

`mapshow(x,R, ..., 'DisplayType', displaytype,...)` where Z is class double and `displaytype` is `'surface'`, `'mesh'`, `'texturemap'`, or `'contour'`, displays a regular M-by-N data grid. `R` is a referencing matrix or referencing vector.

```
mapshow(I,R)
mapshow(BW,R)
mapshow(RGB,R)
mapshow(A,cmap,R)
```

displays a georeferenced image. It is rendered as an image object if the display geometry permits; otherwise, the image is rendered as a texture map on a zero-elevation surface. If specified, `'DisplayType'` must be set to `'image'`.

`mapshow(filename)` displays data from `filename`, according to the type of file format. The `DisplayType` parameter is automatically set according to the following table:

Format	DisplayType
Shapefile	`'point'`, `'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)

`mapshow(ax, ...)` sets the axes parent to AX. This is equivalent to

```
mapshow(..., 'Parent', ax, ...)
```

`h = mapshow(...)` returns a handle to a MATLAB graphics object, an array of object handles, or in the case of vector data, a map graphics object.

`mapshow(..., param1, val1, param2, val2, ...)` specifies parameter/value pairs that modify the type of display or set MATLAB graphics properties. Parameter names can be abbreviated and are case insensitive.

Parameters

Parameters for `mapshow` include

- `'DisplayType'` – The `DisplayType` parameter specifies the type of graphic display for the data. The value must be consistent with the type of data being displayed, as shown in the following table:

Data Type	Value
`vector`	`'point'`, `'line'`, or `'polygon'`
`image`	`'image'`
`grid`	`'surface'`, `'mesh'`, `'texturemap'`, or `'contour'`

Graphics Properties

In addition to specifying a parent axes, the following properties can be set for line, point, and polygon:

- `DisplayType`

DisplayType	Property Name
`'line'`	`'Color'`, `'LineStyle'`, `'LineWidth'`, and `'Visible'`
`'point'`	`'Marker'`, `'Color'`, `'MarkerEdgeColor'`, `'MarkerFaceColor'`, `'MarkerSize'`, and `'Visible'`
`'polygon'`	`'FaceColor'`, `'FaceAlpha'`, `'LineStyle'`, `'LineWidth'`, `'EdgeColor'`, `'EdgeAlpha'`, and `'Visible'`

Refer to the MATLAB graphics documentation on line, patch, image, surface, and mesh for a complete description of these properties and their values.

- `'SymbolSpec'`— The `SymbolSpec` parameter specifies the symbolization rules used for vector data through a structure returned by `makesymbolspec`. It is used only for vector data.

  When both `SymbolSpec` and one or more graphics properties are specified, the graphics properties will override any settings in the symbol spec structure. See example 5 below.

  To change the default symbolization rule for a property name/property value pair in the symbol spec, prefix the word `'Default'` to the graphics property name (listed in the preceding table). See example 4 below.

Refer to the Handle Graphics documentation on lines and patches for a complete description of these properties and their values.

If `PropertyN` is `'SymbolSpec'`, then `ValueN` must be `symbols`. `symbols` should conform to the structure returned by `makesymbolspec`.

When you use `'SymbolSpec'`/`symbols` and other property name/property value pairs together, the property name/property value pairs override any settings in `symbols`. To append the property name/property value pairs to the symbol spec, prefix the word `'Default'` to the property name. See the example below.

Remarks

You can use `mapshow` to render vector data in an `axesm` figure. However, you cannot subsequently change the map projection using `setm`.

You can generally substitute `mapshow` for `displaym` if no map projection is required. However, there are limitations where display of specific objects is concerned. See the remarks under `updategeostruct` for further information.

Examples

Example 1

Display the roads geographic data structure.

```
roads = shaperead('concord_roads.shp');
figure
mapshow(roads);
```

Example 2

Display the roads shape and change the LineStyle.

```
figure
mapshow('concord_roads.shp','LineStyle',':');
```

Example 3

Display the roads shape, and render using a SymbolSpec.

```
% Create a SymbolSpec to color local roads:
% * (ADMIN_TYPE=0) cyan, state roads (ADMIN_TYPE=3) red.
% Hide very minor roads (CLASS=6).
% Make all roads that are major or larger (CLASS=1-4)
% * have a LineWidth of 2.
roadspec = makesymbolspec('Line',...
                          {'ADMIN_TYPE',0,'Color','cyan'}, ...
```

```
                          {'ADMIN_TYPE',3,'Color','red'},...
                          {'CLASS',6,'Visible','off'},...
                          {'CLASS',[1 4],'LineWidth',2});
figure
mapshow('concord_roads.shp','SymbolSpec',roadspec);
```

Example 4

Override default properties of the SymbolSpec.

```
roadspec = ...
makesymbolspec('Line',{'CLASS','Primary','Color','r'}, ...
                      {'CLASS','Improved','Color','y'}, ...
                      {'CLASS','Primary 4L','Color','m'});
figure
mapshow('concord_roads.shp','SymbolSpec',roadspec,...
'DefaultColor','b', 'DefaultLineStyle','-.');
```

Example 5

Override a graphics property of the SymbolSpec.

```
roadspec = ...
makesymbolspec('Line',{'CLASS','Primary','Color','r'}, ...
                      {'CLASS','Improved','Color','y'}, ...
                      {'CLASS','Primary 4L','Color','m'});
figure
mapshow('concord_roads.shp','SymbolSpec',roadspec,'Color','b');
```

Example 6

Display the waterways and roads shapes in one figure.

```
figure
mapshow('concord_roads.shp');
mapshow(gca,'concord_hydro_line.shp','Color','b');
mapshow(gca,'concord_hydro_area.shp','FaceColor','b', ...
'EdgeColor','b');
```

Example 7

View the Mount Washington SDTS DEM terrain data

```
% View the Mount Washington terrain data as a mesh.
```

```
figure
h = mapshow('9129CATD.ddf','DisplayType','mesh');
Z = get(h,'ZData');
colormap(demcmap(Z))

% View the Mount Washington terrain data as a surface.
figure
mapshow('9129CATD.ddf');
colormap(demcmap(Z))
view(3); % View  as a 3-d surface
axis normal;
```

Example 8

Display the grid and contour lines of Mount Washington and Mount Dartmouth.

```
figure
[Z_W, R_W] = arcgridread('MtWashington-ft.grd');
[Z_D, R_D] = arcgridread('MountDartmouth-ft.grd');
mapshow(Z_W, R_W,'DisplayType','surface');
hold on
mapshow(gca,Z_W, R_W,'DisplayType','contour');
mapshow(gca,Z_D, R_D, 'DisplayType','surface');
mapshow(gca,Z_D, R_D,'DisplayType','contour');
colormap(demcmap(Z_W))
% Set the contour lines to the max surface value
zdatam(handlem('line'),max([Z_D(:)' Z_W(:)']));
```

Example 9

Display an image with a worldfile.

```
figure
mapshow('concord_ortho_e.tif');
```

See Also

`arcgridread`, `geoshow`, `geotiffread`, `makesymbolspec`, `mapview`, `sdtsdemread`, `shaperead`, `updategeostruct`

Purpose

Trim line vector map to a specified region

Syntax

`[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

Following is a simple example:

```
lat0 = [1:10,9:-1:0]; lon0 = -30:-11;
[lat,lon] = maptriml(lat0,lon0,[3 7],[-29 -12]);
[lat lon]
ans =
   NaN   NaN
     3   -28
     4   -27
     5   -26
     6   -25
     7   -24
   NaN   NaN
     7   -18
     6   -17
     5   -16
     4   -15
     3   -14
   NaN   NaN
```

Notice that trimmed line segment ends have `NaN`s inserted at trim points.

See Also

`maptrimp`	Trim patch map data to specified region
`maptrims`	Trim surface data grid to specified region

maptrimp

Purpose

Trim patch map data to a specified region

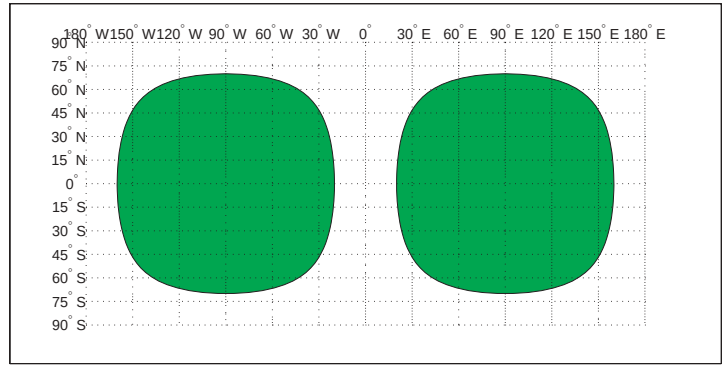
Syntax

`[lat,lon] = maptrimp(lat0,lon0,latlim,lonlim)` returns *adjusted* patch map data sets. The patches must be input in `NaN`-delimited vectors `lat0` and `lon0`. For any patch face completely outside the specified limits, the entire patch face is discarded. When parts of a patch are outside the limits, those data points are moved to lie on the limits. These limits are specified by the two-element vectors `latlim` and `lonlim`, which have the forms `[south-limit north-limit]` and `[west-limit east-limit]`, respectively.

Examples

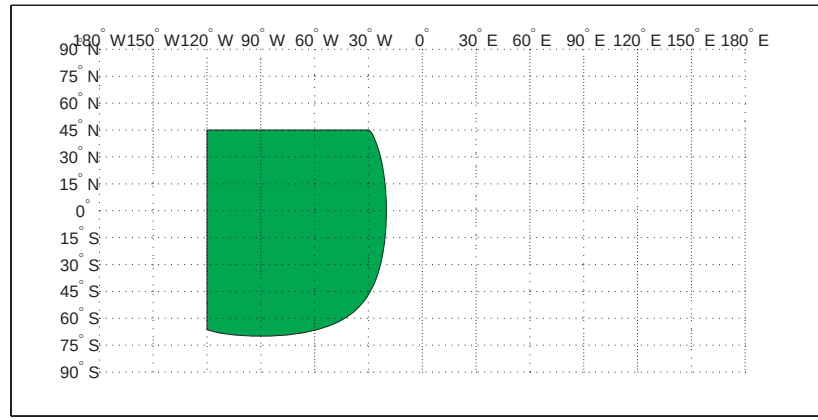
Make two patches using the `scircle1` function, and display them:

```
[lat0,lon0] = scircle1([0 0]',[-90 90]',[70 70]');
axesm('pcarree','Grid','on',...
      'MeridianLabel','on','ParallelLabel','on')
h = fillm(lat0,lon0,'green');
```



Now trim the patch data to lie between 80°S and 45°N latitude, and 120°W and 0° longitude. The patch data is in two columns coming out of `scircle1`, so first you must turn it into `NaN`-delimited vectors.

```
lat0 = [lat0; NaN NaN];
lon0 = [lon0; NaN NaN];
[lat,lon] = maptrimp(lat0(:),lon0(:),[45 -80],[-120 0]);
clmo(h)
fillm(lat,lon,'green')
```



Notice that the patch face to the east, lying completely outside the allowed area, was removed. The western patch was trimmed to the required area.

See Also

`maptriml`	Trim line vector map data to specified region
`maptrims`	Trim surface data grid to specified region

maptrims

Purpose

Trim surface regular data grid data to a specified region

Syntax

`[submap,sublegend] = maptrims(map,refvec,latlim,lonlim)` returns the subset of the input regular data grid between the latitude and longitude limits, in degrees, defined by the two-element vectors `latlim` and `lonlim`. `refvec` is the referencing vector of the input data grid; `sublegend` is the referencing vector of the output data grid.

`[submap,sublegend] = maptrims(map,refvec,latlim,lonlim,scale)` is a means of further reducing the size of the output matrix. The cells-per-degree scale of the original matrix is given by the first element of `refvec`. The desired cells-per-degree scale in the output map is given by `scale`, which must equally divide `refvec(1)`. For example, if `refvec(1)` were 20 (cells per degree), then `scale` could be 1, 2, 4, 5, 10, or 20.

Description

The `maptrims` function selects a portion of a larger data grid defined by a latitude-longitude quadrangle.

The reduced matrix is created using `resizem` with a `'nearest'` interpolation method.

Examples

```
load topo
[submap,sublegend] = maptrims(topo,topolegend,...
                              [80.25 85.3],[165.2 170.7])

submap =

       -2826       -2810       -2802       -2793
       -2915       -2913       -2905       -2884
       -3192       -3186       -3165       -3122
       -3399       -3324       -3273       -3214


sublegend =

     1    85   166
```

The upper left corner of the map might differ slightly from that of the requested region. `maptrims` uses the corner coordinates of the first cell inside the limits.

See Also

`maptriml`	Trim line vector map data to specified region
`maptrimp`	Trim patch map data to specified region
`resizem`	Resize a data grid

mapview

Purpose

Interactive map viewer

Description

Use the Map Viewer to work with vector, image, and raster data grids in a map coordinate system: 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.

Syntax

`mapview` (with no arguments) starts a new Map Viewer in an empty state.

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 Mapping Toolbox, set your working directory or navigate the Map Viewer **Open** dialog to

```
matlabroot/toolbox/map/mapdemos
```

Import From Workspace

Images: Use the **Raster Data -> Image** import dialog to select a **referencing matrix and data name** for the image from the list of workspace variables. If the image type is true-color (RGB), specify which band represents the red, green, and blue intensities.

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 struct 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` or `mfwdtran` 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` or `minvtran`, then reproject with `projfwd` or `mfwdtran`. You might also need to adjust the horizontal datum of your data (using, for example, the free GEOTRANS application from the Geospatial Sciences Division of the U.S. National Imagery and Mapping Agency (NIMA, `http://www.nima.mil`). 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 symbol spec 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 symbol spec to use a different color and/or line width and/or line style for each road class. See the `makesymbolspec` help for examples and to learn how to construct a symbol spec. If you regularly work with data sets sharing a common set of feature attributes, you might want to save one or more symbol specs in a MAT-file (or save calls to `makesymbolspec` in an M-file).

Once you have a symbol spec 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 symbol spec 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. Note that to insert an additional object of the same type, you must reselect the appropriate tool. In addition, the **Insert** menu allows you to insert axis labels and a title. 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.

See Also

`arcgridread`, `geoshow`, `geotiffread`, `makesymbolspec`, `mapshow`, `sdtsdemread`, `shaperead`, `updategeostruct`, `worldfileread`

Purpose

Convert distinct matrix elements to dms format

Syntax

`anglout = mat2dms(d,m,s)` takes angles separated into three inputs, one each for degrees, minutes, and seconds, and converts them to single dms values.

`anglout = mat2dms(d,m,s,n)` specifies the power of 10, `n`, to which the input seconds should be rounded before they are converted (that is, if a result is 12.567 seconds, and n = -2, the resulting seconds output would be 12.57). The default value of `n` is -2.

`anglout = mat2dms([d,m,s],n)` allows the inputs to be packed into a three-column matrix in which the columns represent degrees, minutes, and seconds, respectively.

Examples

```
anglout = mat2dms(23,45,17.5)
anglout =
                  2345.175
```

See Also

`dms2mat` Convert from dms to separated matrices

mat2hms

Purpose

Convert distinct matrix elements to hms format

Syntax

`timeout = mat2hms(h,m,s)` takes times separated into three inputs, one each for hours, minutes, and seconds, and converts them to single hms values.

`timeout = mat2hms(h,m,s,n)` specifies the power of 10, `n`, to which the input seconds should be rounded before they are converted (that is, if a result is 12.567 seconds, and `n = -2`, the resulting seconds output would be 12.57). The default value of `n` is -2.

`timeout = mat2hms([h,m,s],n)` allows the inputs to be packed into a three-column matrix in which the columns represent hours, minutes, and seconds, respectively.

Examples

```
timeout = mat2hms([13 35],[34 18],[29.8 17.0])
timeout =
                  1334.298                    3518.17
```

See Also

`hms2mat` Convert from hms to separated matrices

Purpose

Display contours of constant distortion on a map

Syntax

`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`*`)` or `mdistort` *`parameter`* displays contours of distortion for the specified parameter. Recognized *`parameter`* strings are `'area'`, `'angles'` for the maximum angular distortion of right angles, `'scale'` or `'maxscale'` for the maximum scale, `'minscale'` for the minimum scale, `'parscale'` for scale along the parallels, `'merscale'` for scale along the meridians, and `'scaleratio'` for the ratio of maximum and minimum scale. If omitted, the `'maxscale'` parameter is displayed. All parameters are displayed as percent distortion except angles, which are displayed in degrees.

`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,ht] = mdistort(...)` returns the handles to the line and text objects.

Background

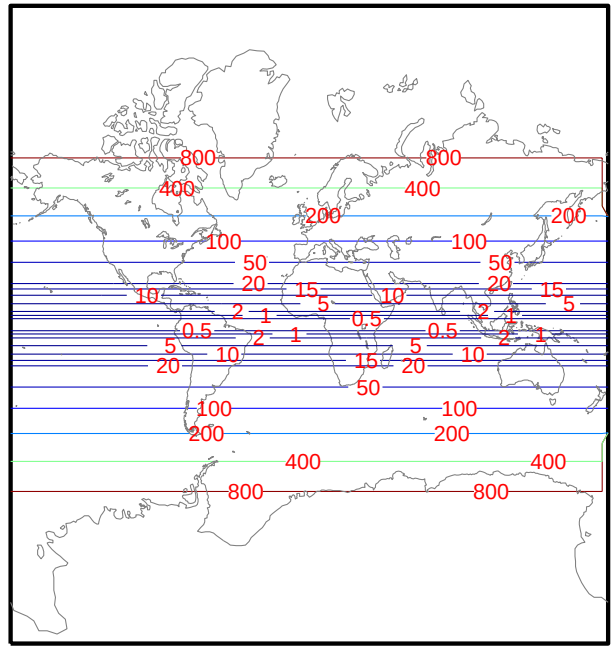
Map projections inevitably introduce distortions in the shape and size of objects as they are transformed from three-dimensional spherical coordinates to two-dimensional Cartesian coordinates. The amount and type of distortion vary between projections, over the projection, and with the selection of projection 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

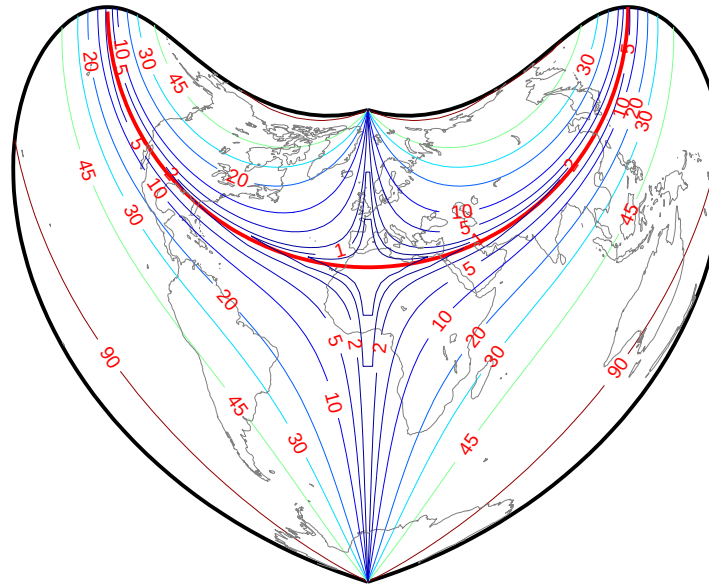
Note the extreme area distortion of the Mercator projection. This makes it ill-suited for global displays.

```
figure
axesm mercator
load coast
framem;plotm(lat, long,'color',.5*[1 1 1])

mdistort area
```



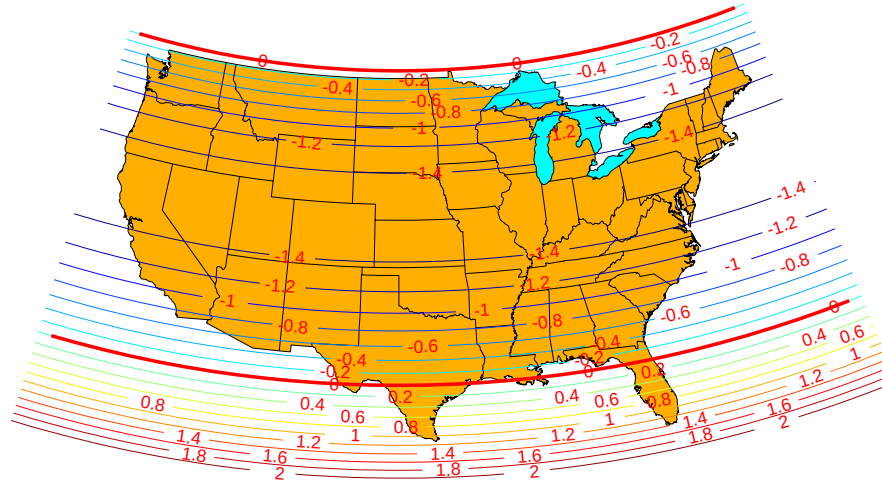
The lines of zero distortion for the Bonne projection follow the central meridian and the standard parallel.

```
figure
axesm bonne
load coast
framem;plotm(lat, long,'color',.5*[1 1 1])

mdistort angles
parallelui
```



An equidistant conic projection with properly chosen parallels can map the conterminous United States with less than 1.5% distortion.

```
figure
usamap conus
mdistort('parscale', -2:.2:2)
parallelui
```



Remarks

`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`	Tissot indicatrices projected onto a map axes
`distortcalc`	Calculate distortion parameters for a map projection
`vfwdtran`	Transform vector azimuths to a projection space angle

Purpose

Compute mean for geographic data

Syntax

`[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. When the standard angle string *`units`* is omitted, `'degrees'` is assumed.

`[latmean,lonmean] = meanm(lat,lon,ellipsoid)` specifies the elliptical definition of the Earth to be used with the two-element `ellipsoid` vector. 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" in the "Mapping Applications" chapter of the Mapping Toolbox User's Guide documentation.

Examples

Create random latitude and longitude matrices:

```
lat = rand(3)
lat =
    0.9501    0.4860    0.4565
    0.2311    0.8913    0.0185
    0.6068    0.7621    0.8214

lon = rand(3)
lon =
    0.4447    0.9218    0.4057
    0.6154    0.7382    0.9355
    0.7919    0.1763    0.9169

[latmean,lonmean] = meanm(lat,lon,'radians')
latmean =
    0.6004    0.7395    0.4448
lonmean =
```

```
       0.6347    0.6324    0.7478
```

See Also

`filterm`	Geographic filter for data sets
`hista`	Spatial equal area histogram
`histr`	Spatial equirectangular histogram
`stdist`	Standard distances for geographic data
`stdm`	Standard deviation for geographic data

Purpose

Construct a map graticule mesh for surface object display

Syntax

`[latgrat,longrat] = meshgrat(map,refvec)` constructs a graticule for the regular data grid `map` with the associated referencing vector `refvec`. The default graticule size is equal to the size of the map matrix.

`[latgrat,longrat] = meshgrat(map,refvec,npts)` returns a graticule mesh of size `npts`. The input `npts` is a two-element vector of the form `[latitude-points longitude-points]`. If `npts` is set to an empty matrix, then the graticule returned is the Mapping Toolbox default graticule size `[50 100]`.

`[latgrat,longrat] = meshgrat(lat,lon)` can be used for data grids that are not regular in spacing (e.g., row one represents 1°, row two represents 1.34°) but are regular in orientation (rows are north-south, columns are east-west). The inputs `lat` and `lon` are vectors describing the latitudes and longitudes on a row-by-row and column-by-column basis for the data grid to be displayed. Regardless of the variable spacing of the matrix, the graticule is evenly spaced. In this form, `meshgrat` is similar to the MATLAB function `meshgrid`.

`[latgrat,longrat] = meshgrat(latlim,lonlim,npts)` returns a graticule mesh of size `npts`. The input vectors `latlim` and `lonlim` are two-element vectors specifying the graticule latitude and longitude limits. The input `npts` is a two-element vector of the form `[latitude-points longitude-points]`. If `npts` is set to an empty matrix, then the graticule returned is the Mapping Toolbox default graticule size `[50 100]`.

`[latgrat,longrat] = meshgrat(lat,lon,`*`units`*`)` and `[latgrat,longrat] = meshgrat(latlim,lonlim,npts,`*`units`*`)` use the input *`units`* to specify the angle units of the input and output parameters. If omitted, `'degrees'` is assumed.

Description

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]; longlim = [-180 180];
```

```
[latgrat,longrat] = meshgrat(latlim,longlim,[3 6])
latgrat =
  -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
longrat =
 -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 a surface object like the `topo` map can be warped to the grid.

See Also

`meshm`	Regular data grid warped to a projected graticule mesh
`pcolorm`	Project a data grid in the z = 0 plane
`surfacem`	Data grid warped to a projected graticule mesh
`surfm`	Data grid projected on a map axes

Purpose

Project 3-D lighted shaded relief of a regular data grid

Syntax

`meshlsrm(map,refvec)` displays the regular data grid colored according to elevation and surface slopes. The current axes must have a valid map projection definition.

`meshlsrm(map,refvec,[azim elev])` displays the regular 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 (90° azimuth) at an elevation of 45°.

`meshlsrm(map,refvec,[azim elev],cmap)` displays the regular 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.

`meshlsrm(map,refvec,[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.

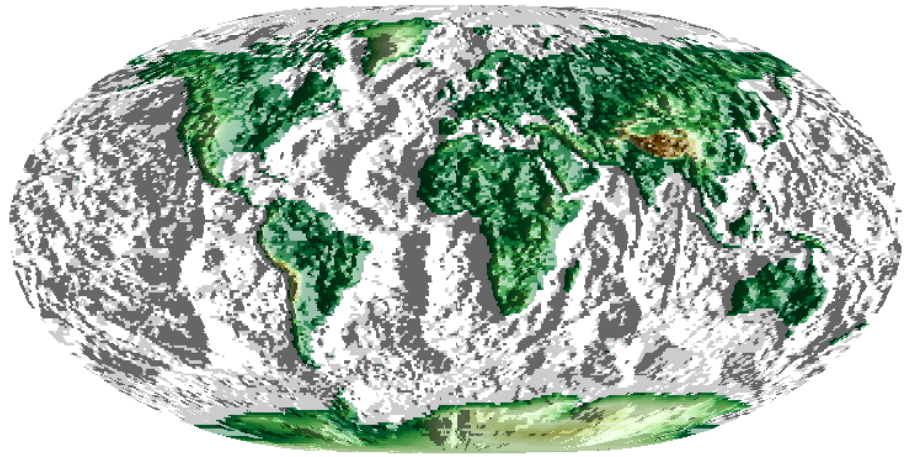
Remarks

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.

Examples

Create a new colormap using `demcmap`, with white colors for the sea and default colors for land. Use this colormap for a lighted shaded relief map of the world.

```
load topo
[cmap,clim] = demcmap(topo,[],[1 1 1],[]);
axesm loximuth
meshlsrm(topo,topolegend,[],cmap,clim)
```



See Also

`meshm`	Regular data grid warped to a projected graticule mesh
`pcolorm`	Project a data grid in the z = 0 plane
`shaderel`	Construct `cdata` and colormap for colored shaded relief
`surfacem`	Data grid warped to a projected graticule mesh
`surflm`	Display a lighted data grid warped to a projected graticule
`surfm`	Data grid projected to a map axes
`surflsrm`	Project a 3-D lighted shaded relief of a geolocated data grid

Purpose

Warp regular data grid to a projected graticule mesh

Syntax

`h = meshm(map,refvec)` projects the regular data grid onto the current map axes. The handle, `h`, of the displayed surface can be returned.

`h = meshm(map,refvec,npts)` specifies the resolution of the graticule grid. The input `npts` is of the form `[latitude-points longitude-points]`. The default value of `npts` is `[50 100]` (the graticule has 50 vertices in the latitude direction and 100 vertices in the longitude direction).

`h = meshm(map,refvec,npts,alt)` sets the *z*-axis altitude of the graticule mesh. `alt` can be a scalar, in which case the map is plotted on a `z = alt` plane, or `alt` can be a matrix of `size(alt) = npts`, in which case the graticule mesh is plotted in 3-D.

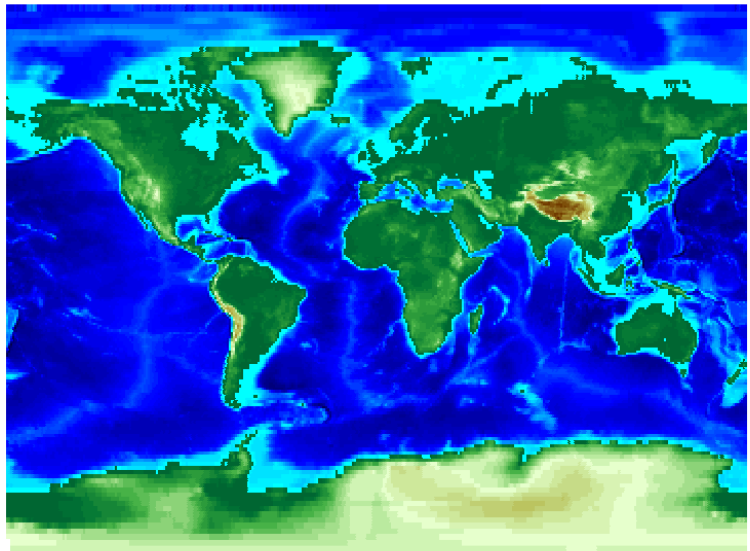
`h = meshm(map,refvec,`*`PropertyName`*`,PropertyValue,...)` allows the input of property name/property value pairs to control the surface object properties. Any property supported by the standard MATLAB function `surface` except `XData`, `YData`, and `ZData` can be altered in this manner.

Description

The `meshm` function warps a regular data grid to a graticule mesh, which is itself projected according to the `MapProjection` property of the current map axes. 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.

Examples

```
load topo
axesm miller
meshm(topo,topolegend,[90 180])
demcmap(topo)
```



See Also

`meshgrat`	Construct map graticule grid
`pcolorm`	Project a data grid in the z = 0 plane
`surfacem`	Data grid warped to a projected graticule mesh
`surfm`	Data grid projected to a map axes

Purpose

Transform unprojected Greenwich data to a projected Cartesian coordinate system

Syntax

`[x,y] = mfwdtran(lat,lon)` transforms unprojected Greenwich data to the projected Cartesian coordinate frame using the map projection defined for the current axes. No clipping or trimming of data is performed with this calling form.

`[x,y,z] = mfwdtran(lat,lon,alt)` transforms the three-dimensional data to the projected Cartesian coordinate frame using the map projection defined for the current axes. If `alt = []` or `alt` is omitted, the default `alt = 0` is used.

`[x,y,z,struct] = mfwdtran(lat,lon,alt,`*`object`*`)` clips and trims the data during the transformation process. Allowable *`object`* strings are `'surface'`, `'line'`, `'patch'`, `'light'`, `'text'`, and `'none'`. `'none'` results in no clipping or trimming of the input data. The output `struct` is a structure containing information about the clips and trims associated with the transformed object. This structure is also found in the displayed object's `UserData` property.

`[...] = mfwdtran(mstruct,...)` requires a valid map projection structure as the first argument. This structure is used to define the map projection calculations performed. No map axes need be displayed when using this calling form.

Examples

The following latitude and longitude data for the District of Columbia is obtained from the `usalo` workspace:

```
load usalo
lat = state(51).lat;
lon = state(51).long;
[lat lon]
ans =
   38.9000  -77.0700
   38.9000  -77.0500
   38.9000  -77.0700
   38.8700  -77.0200
   38.8000  -77.0200
   38.7800  -77.0300
   38.9000  -76.9000
   39.0000  -77.0300
   38.9500  -77.1200
```

```
    38.9000  -77.0700
```

Before projecting the data, it is necessary to define projection parameters. You can do this with the `axesm` function or with the `defaultm` function:

```
mstruct = defaultm('mercator');
mstruct.origin = [38.89 -77.04 0];
mstruct = defaultm(mstruct);
```

Now that the projection parameters have been set, transform the District of Columbia data into the Cartesian frame using the Mercator projection:

```
[x,y] = mfwdtran(mstruct,lat,lon);
[x y]
ans =
   -0.0004    0.0002
   -0.0001    0.0002
   -0.0004    0.0002
    0.0003   -0.0003
    0.0003   -0.0016
    0.0001   -0.0019
    0.0019    0.0002
    0.0001    0.0019
   -0.0011    0.0010
   -0.0004    0.0002
```

See Also

`defaultm`	Initialize a default map data structure
`gcm`	Get current map data structure
`minvtran`	Map inverse transformation
`vfwdtran`	Vector forward transformation
`vinvtran`	Vector inverse transformation

Purpose

Calculate semiminor axis from semimajor axis and eccentricity

Syntax

`semiminor = minaxis(semimajor,eccentricity)` returns the semiminor axis length corresponding to the input semimajor axis and eccentricity.

`semiminor = minaxis([semimajor,eccentricity])` allows the inputs to be packed into a single two-column input of the form `[semimajor, eccentricity]`.

Description

The semiminor axis can be determined given both the semimajor axis and the eccentricity, the two elements of a standard ellipsoid vector in the Mapping Toolbox.

Examples

Using the default values for the Earth,

```
semiminor = minaxis(almanac('earth','ellipsoid'))
semiminor =
   6.3568e+03
```

See Also

`almanac`	Planetary data
`axes2ecc` `majaxis`	Related conversion functions

minvtran

Purpose

Transform projected Cartesian data to an unprojected Greenwich coordinate system

Syntax

`[lat,lon] = minvtran(x,y)` transforms projected Cartesian data to an unprojected Greenwich coordinate frame using the map projection defined for the current axes. No data clips or trims are removed with this calling form.

`[lat,lon,alt] = minvtran(x,y,z)` transforms the three-dimensional data to the unprojected Greenwich coordinate frame using the map projection defined for the current axes. If `z = []` or `z` is omitted, the default `z = 0` is used.

`[lat,lon,alt] = minvtran(x,y,z,`*`object`*`,struct)` removes all clips and trims from the input data. Allowable *`object`* strings are `'surface'`, `'line'`, `'patch'`, `'light'`, `'text'`, and `'none'`. `'none'` results in no removal of any clips or trims of the input data. The output `struct` is a structure containing information about the clips and trims associated with the transformed object, and is created by the function `mfwdtran`.

`[...] = minvtran(mstruct,...)` requires a valid map projection structure as the first argument. This structure is used to define the map projection calculations performed. No map axes need be displayed when using this calling form.

Examples

Before using any transformation functions, 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 `usalo` workspace:

```
load usalo
lat = state(51).lat;
lon = state(51).long;
[lat lon]
ans =
   38.9000  -77.0700
   38.9000  -77.0500
   38.9000  -77.0700
```

```
   38.8700  -77.0200
   38.8000  -77.0200
   38.7800  -77.0300
   38.9000  -76.9000
   39.0000  -77.0300
   38.9500  -77.1200
   38.9000  -77.0700
```

This data can be projected into Cartesian coordinates of the Mercator projection using the `mfwdtran` function:

```
[x,y] = mfwdtran(mstruct,lat,lon);
[x y]
ans =
   -0.0004    0.0002
   -0.0001    0.0002
   -0.0004    0.0002
    0.0003   -0.0003
    0.0003   -0.0016
    0.0001   -0.0019
    0.0019    0.0002
    0.0001    0.0019
   -0.0011    0.0010
   -0.0004    0.0002
```

To transform the projected *x-y* data back into the unprojected Greenwich frame, use the `minvtran` function:

```
[lat2,lon2] = minvtran(mstruct,x,y);
[lat2 lon2]
ans =
   38.9000  -77.0700
   38.9000  -77.0500
   38.9000  -77.0700
   38.8700  -77.0200
   38.8000  -77.0200
   38.7800  -77.0300
   38.9000  -76.9000
   39.0000  -77.0300
   38.9500  -77.1200
   38.9000  -77.0700
```

See Also

`axesm`	Map axes definition and property setting
`defaultm`	Initialize a default map data structure
`gcm`	Get current map data structure
`mfwdtran`	Map forward transformation
`vfwdtran`	Vector forward transformation
`vinvtran`	Vector inverse transformation

Purpose

Project meridian labels on a map axes

Syntax

`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 the strings `'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`	Define map axes and set map properties
`mlabelzero22pi`	Display longitude labels in the range of 0 to 360 degrees
`plabel`	Parallel labels projected on a map axes
`setm`	Set map properties

mlabelzero22pi

Purpose

Display longitude labels in the range of 0 to 360 degrees

Syntax

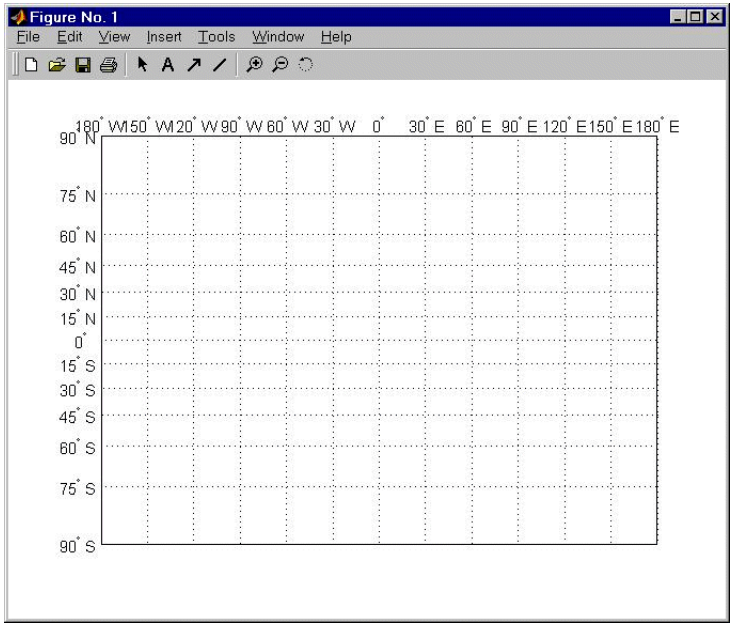
```
mlabelzero22pi
```

Description

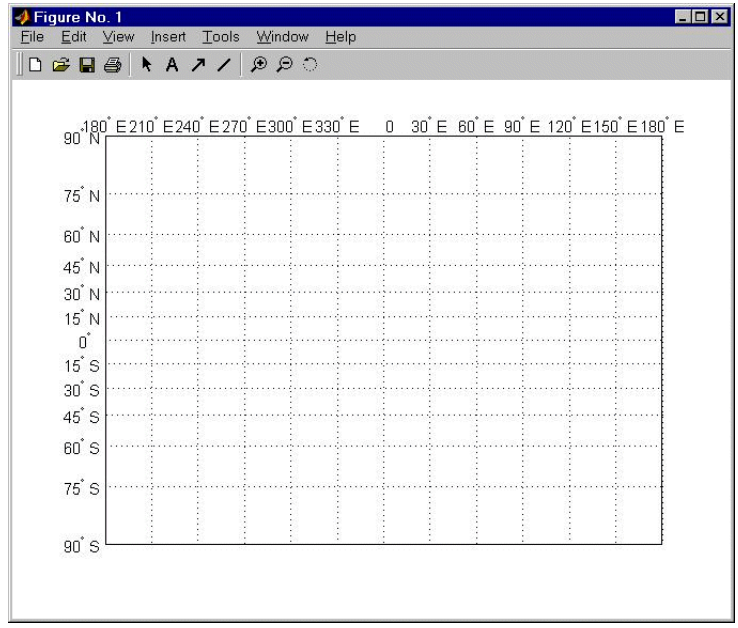
`mlabelzero22pi` displays longitude labels in the range of 0 to 360 degrees east of the prime meridian.

Example

```
% 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` Project meridian labels on a map axes

n2ecc

Purpose

Convert from the *n* to the eccentricity representation of the ellipsoid

Syntax

`eccentricity = n2ecc(n)` returns the equivalent eccentricities for the input *n* parameters. If the input `n` is a two-column vector, only the second column is used. This allows two-element vectors to be used as rows of the input, because the form `[semimajor-axis, n]` is a complete representation of an ellipsoid (but is not the standard form for ellipsoid vectors in the Mapping Toolbox). In all other cases, all columns of the input are used.

Description

Eccentricity and the parameter *n* are two methods of defining an ellipsoid. The definition of *n* is

(semimajor axis – semiminor axis)/(semimajor axis + semiminor axis)

Example

```
ecc = n2ecc(0.00167922039463)
ecc =
   0.08181919104285
```

This eccentricity is the default value for the Earth.

See Also

`almanac`	Planetary data
`ecc2flat` `majaxis`	Similar conversion functions
`ecc2n`	Convert from eccentricity to *n*

Purpose

Determine the names of valid graphics objects

Syntax

`objects = namem` returns the object names for all objects on the current axes. The object name is defined as its tag, if the object `Tag` property is supplied. Otherwise, it is the object `Type`. Duplicate object names are removed from the output string matrix.

`objects = namem(handles)` returns the object names for the objects specified by the input `handles`.

`[objects,message] = namem(...)` returns a string message indicating any error encountered.

The names returned are either set at object creation or defined by the user with the `tagm` function.

See Also

`clma`	Clear current map
`clmo`	Clear specified graphics objects
`handlem`	Get handles of displayed graphics objects
`hidem`	Hide specified graphics objects
`showm`	Show specified graphics objects
`tagm`	Assign a name to graphics object `Tag` property

nanclip

Purpose

Convert pen-down delimited data to NaN-delimited data

Syntax

```
dataout = nanclip(datain)
```

`dataout = nanclip(datain,pendowncmd)` returns 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`.

Description

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` Read space-delimited data

Purpose

Create data grids containing NaNs

Syntax

`map = nanm(latlim,lonlim,scale)` returns a regular data grid consisting entirely of NaNs. The two-element vectors `latlim` and `lonlim` define the latitude and longitude limits of the geographic region. They should be of the form [north south] and [east west], respectively. The number of rows and columns per angle unit is set by the scalar value `scale`.

`[map,refvec] = nanm(latlim,lonlim,scale)` returns the three-element referencing vector for the returned `map`.

Example

```
[map,refvec] = nanm([46,51],[-79,-75],1)
map =
   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN
refvec =
     1    51   -79
```

See Also

`limitm`	Data grid limits
`onem`	Create a data grid of ones
`sizem`	Row and column dimensions needed for map
`spzerom`	Create a sparse data grid of zeros
`zerom`	Create a full data grid of zeros

Purpose

Determine Mercator-based navigational fix

Syntax

`[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 zeros 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 documentation for a more complete description of the use of this function.

Remarks

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.

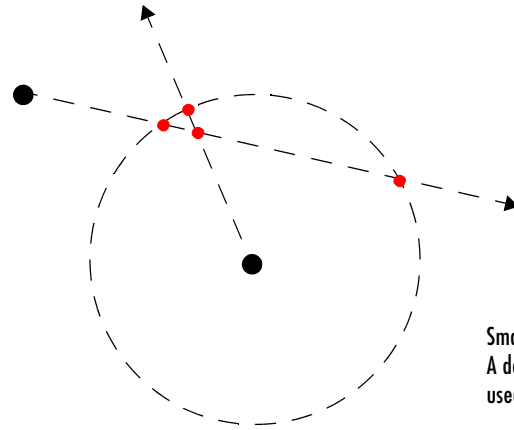
Examples

For a fully illustrated example of the application of this function, refer to the "Navigation" section in the Mapping Toolbox User's Guide documentation.

Imagine you have two landmarks, at (15°N,30.4°W) and (14.8°N,30.1°W). You have a visual bearing to the first of 280° and to the second of 160°. 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:



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.

See Also

`crossfix`	Great circle fixing

`gcxgc`
`gcxsc`
`scxsc`
`rhxrh`
`polyxpoly`
	Other intersection functions

See Also

`dreckon`	Compute dead reckoning positions for a track
`gcwaypts`	Find equally spaced waypoints along a great circle
`legs`	Find courses and distances between waypoints

`navfix`	Mercator-based navigational fixing
`track`	Connect navigational waypoints with track segments

neworig

Purpose

Transform regular data grid to new coordinate system based on a new origin

Syntax

`[map,lat,lon] = neworig(map0,refvec,origin)` returns the data in the original regular data grid `map0`, with its three-element referencing vector `refvec`, reallocated to the cells of the new (same-sized) data grid. This transformation is governed by the input `origin`. This is a three- (or two-) element vector of the form `[latitude longitude orientation]`. The latitude and longitude are the coordinates of the point in the original system that 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 0°. This `origin` vector might be the output of `putpole` or `newpole`. The outputs `lat` and `lon` are matrices the size of `map` that give a cell-by-cell registration of `map` to the coordinates of the original (`map0`) system in latitude and longitude, respectively.

`[map,lat,lon] = neworig(map0,refvec,origin,`*`direction`*`)` allows the specification of the operation. If the string *`direction`* is `'forward'` (the default), the transformation occurs as described above. If the *`direction`* is `'inverse'`, then the output `map` is the *original* system from which a transformed matrix `map0` was derived, via the input `origin`. Note that if the matrix `map1` is transformed forward to `map2`, and `map2` is transformed inversely to `map3`, `map3` will look very much like `map1`, but the two matrices will not be identical. This is because `neworig` is in fact projecting the values of the cells twice, rather than *undoing* the first transformation, and matrix data has granularity.

`[map,lat,lon] = neworig(map0,refvec,origin,`*`direction,units`*`)` allows the specification of the angular units of the origin vector, where *`units`* is any valid angle units string. The default is `'degrees'`.

Description

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.

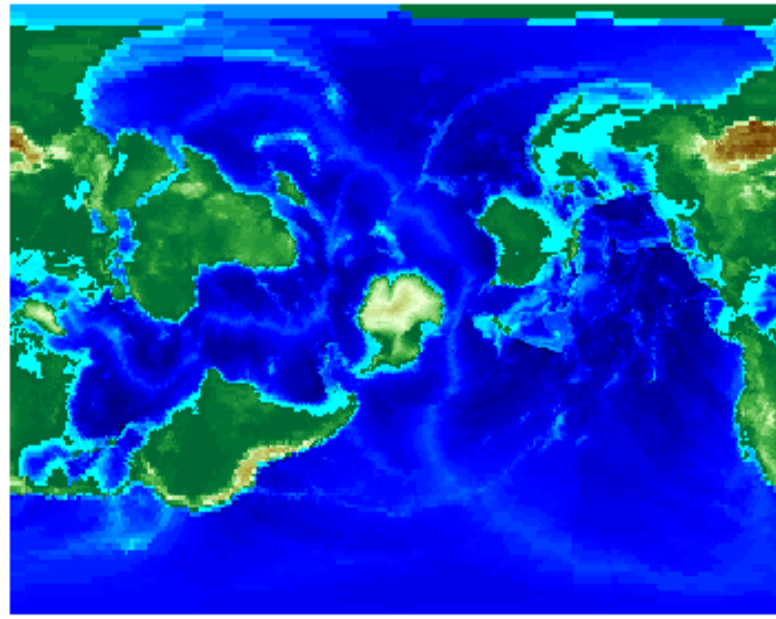
Limitations

`neworig` only supports matrix maps that cover the entire globe.

Examples

This is the `topo` map transformed to put Sri Lanka at the North Pole:

```
load topo
origin = newpole(7,80)
origin =
   83.0000 -100.0000         0
[map,lat,lon] = neworig(topo,topolegend,origin);

axesm miller
surfm(map,[30 30])
demcmap(topo)
```



See Also

`newpole`	Select point to place at North Pole
`org2pol`	Pole of transformed coordinate system
`putpole`	Origin of transformed coordinate system
`rotatem`	Transform vector data to new coordinate system based on a new origin

newpole

Purpose

Compute origin of a transformed coordinate system based on a new pole

Syntax

`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 0°.

`origin = newpole(polelat,polelon,units)` specifies the units of the inputs and output, where *`units`* is any valid angle units string. The default is `'degrees'`.

Description

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 30° beyond the true North Pole on the original origin's meridian is pulled up to become the *pole*, the point originally 30° above the origin is pulled down into the origin spot.

See Also

`neworig`	Transform regular data grid to new coordinate system
`org2pol`	Pole of transformed coordinate system
`putpole`	Origin of transformed coordinate system

Purpose

Convert distance from nautical miles to other units

Syntax

`distout = nm2deg(distin)` converts the input distance given in nautical miles to degrees. `distout = nm2km(distin)`, `distout = nm2rad(distin)`, and `distout = nm2sm(distin)` perform analogously, converting to kilometers, radians, and statute miles, respectively.

`distout = nm2deg(distin,radius)`and `distout = nm2rad(distin,radius)` specify the radius of the sphere to use, because a degree (or radian) of arc length covers less distance, for example, on Mars than it does on the Earth. You can enter the radius as a number in nautical miles, as a call to the `almanac` function (e.g., `almanac('mars','radius','nm')`), or you can pass in a string planet name (e.g., `'mars'`), and the function will make the appropriate call to the `almanac` function. The radius of the Earth is the default.

Examples

How fast is 30 knots (nautical miles per hour) in kph?

```
distout = nm2km(30)
distout =
   55.5600
```

See Also

`distdim`	Convert distances between different units
`km2sm` `sm2deg`	Other direct distance conversion functions

northarrow

Purpose

Add graphic element pointing to the geographic North Pole

Syntax

`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.

Description

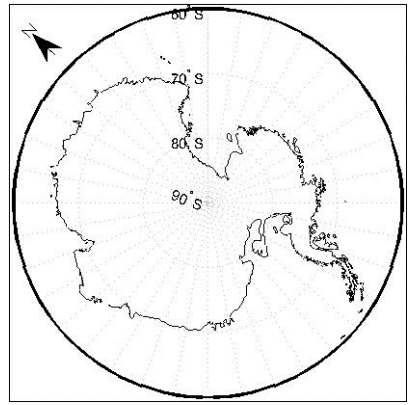
`northarrow` creates a north arrow symbol at the map origin on the displayed map. You can reposition the north arrow symbol by clicking and dragging its icon. Alternate clicking the icon creates an input dialog box that you can also use to change the location of the north arrow.

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.

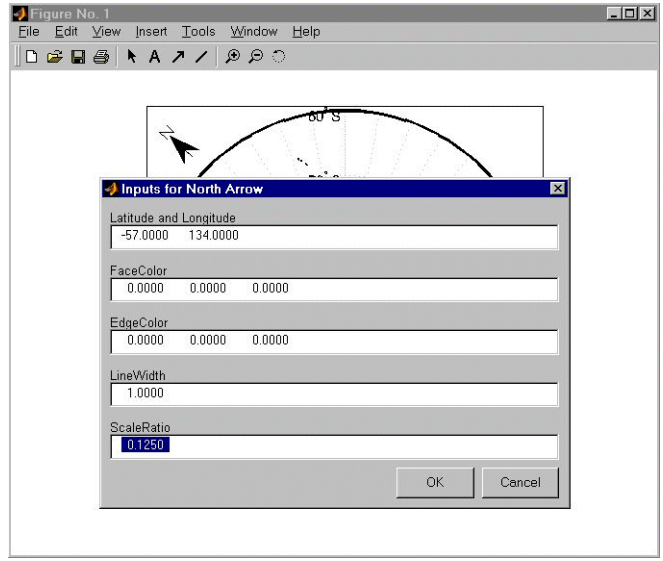
Examples

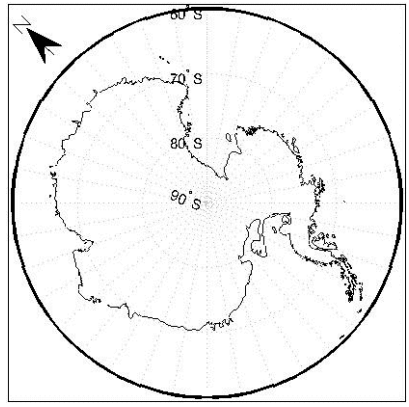
Create a map of the South Pole and then add the north arrow in the upper left of the map.

```
figure; worldmap('south pole')
northarrow('lat',-57,'lon',134);
```



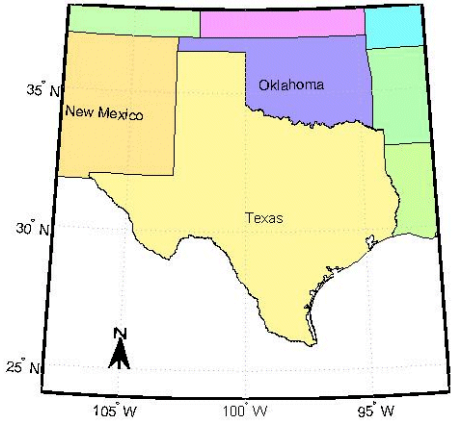
Right-click the north arrow icon to activate the input dialog box. Increase the size of the north arrow symbol by changing the `'ScaleRatio'` property.





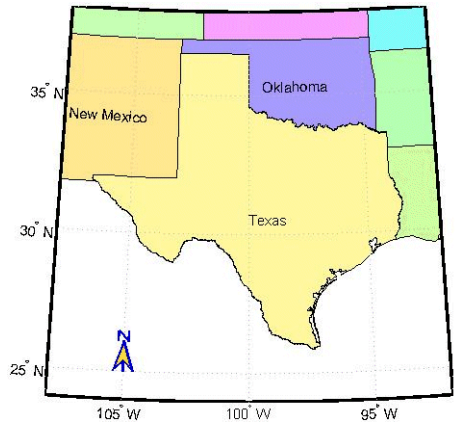
Create a map of Texas and add the north arrow in the lower left of the map.

```
figure; usamap('texas')
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])
```



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.

npi2pi

Purpose

Convert normalized angles to lie between -π and π

Syntax

`anglout = npi2pi(anglin)` *wraps* the input angle `anglin` to lie on the range -180 to 180 (e.g., 270° is renamed -90°).

`anglout = npi2pi(anglin,`*`units`*`)` specifies the angle units with any valid angle units string *`units`*. The default is `'degrees'`.

`anglout = npi2pi(anglin,`*`units`*`,`*`approach`*`)` specifies the approach logic for this wrapping. The *`approach`* string `'exact'` calculates a mathematically precise wrap. `'inward'` and `'outward'` calculate more quickly by shifting the values by an *epsilon* either toward or away from the origin and performing a trigonometric wrap. The trigonometric wrap is inexact to allow for the fact that different computer math processors might give different (although trigonometrically identical) results (180° or -180°, for example). The offset prevents this.

Examples

```
npi2pi(315)
ans =
   -45
npi2pi(181)
ans =
  -179
```

See Also

`zero22pi` Normalize angles to lie between 0 and 2π

Purpose

Create data grids containing ones

Syntax

`map = onem(latlim,lonlim,scale)` returns a regular data grid consisting entirely of ones. 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 number of rows and columns per angle unit is set by the scalar value `scale`.

`[map,refvec] = onem(latlim,lonlim,scale)` returns the three-element referencing vector for the returned `map`.

Examples

```
[map,refvec] = onem([46,51],[-79,-75],1)
map =
     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1
refvec =
     1    51   -79
```

See Also

`limitm`	Data grid limits
`nanm`	Create a data grid of `NaN`s
`sizem`	Row and column dimensions needed for map
`spzerom`	Create a sparse data grid of zeros
`zerom`	Create a full data grid of zeros

org2pol

Purpose

Compute pole of a transformed coordinate system based on a new origin

Syntax

`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 units string. The default is `'degrees'`.

Description

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 (30°N,0°) 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 30° down to the origin, and the North Pole is pulled down 30°. A little less obvious example is the following:

```
pole = org2pol([5 40 30])
pole =
   59.6245   80.0750   40.0000
```

See Also

`neworig` Transform a regular data grid to a new coordinate system

`putpole` Origin of transformed coordinate system based on a new pole

Purpose

Figure paper size for a given map scale

Syntax

`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 string *`sunits`* can be any string recognized by `distdim`. The paper units string *`punits`* can be any dimensional units string 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. If `radius` is a string, then it is evaluated as an `almanac` body to determine the spherical radius. If numerical, it is the radius of the desired sphere in the same units as the surface distance. 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

`pagedlg`	Page position dialog box
`axesscale`	Resize axes for equivalent scale
`daspectm`	Figure `DataAspectRatio` property for a map

Purpose

Project patches onto the current map axes as separate objects

Syntax

The `patchesm` function is very similar to the `patchm` function. The significant difference is that in `patchesm`, separate patches (delineated by `NaN`s 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.

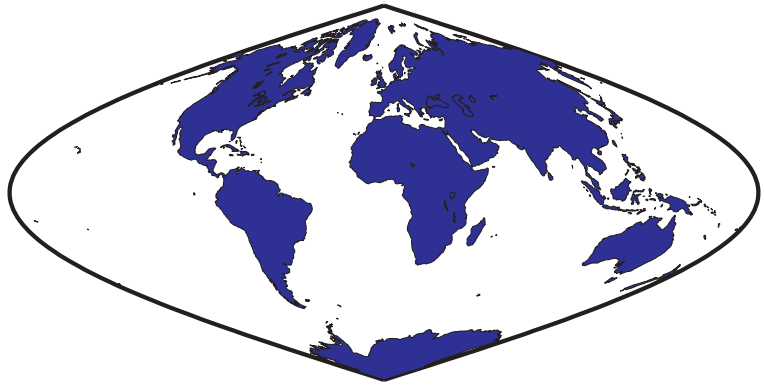
`h = patchesm(lat,lon,cdata)` and `h = patchesm(lat,lon,cdata, PropertyName,PropertyValue,...)` projects and displays patch (polygon) objects defined by their vertices given in `lat` and `lon` on the current map axes. `lat` and `lon` must be vectors. The color data, `cdata`, can be any color data designation supported by the standard MATLAB `patch` function. The object handle or handles, `h`, can be returned.

`h = patchesm(lat,lon,PropertyName,PropertyValue,...)` allows any property name/property value pair supported by `patch` to be assigned to the `patchesm` objects.

`h = patchesm(lat,lon,z,cdata)` and `h = patchesm(lat,lon,z,cdata, PropertyName,PropertyValue,...)` allows the assignment of an altitude, `z`, to each patch object. The default altitude is `z = 0`.

Examples

```
load coast
axesm sinusoid; framem
h = patchesm(lat,long,'b');
```



```
length(h)
ans =
   238
```

See Also

`patchm`	Project patch objects on the current map axes
`fill3m`	Project 3-D patch objects onto the current map axes
`fillm`	Project 2-D patch objects onto the current map axes

Purpose

Project patch objects onto the current map axes

Syntax

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. Except for `XData`, `YData`, and `ZData`, all line properties and styles available through `patch` are supported by `patchm`.

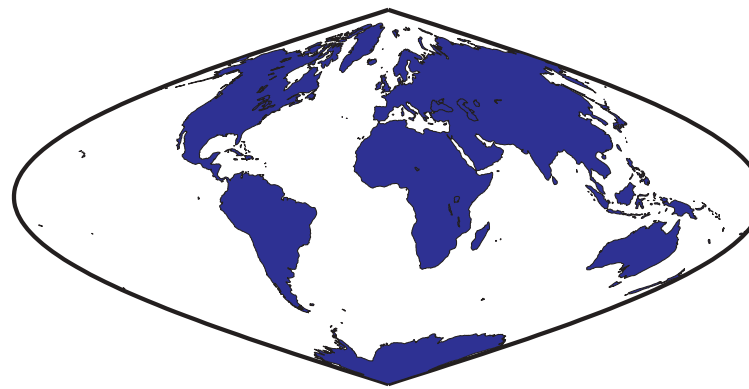
`h = patchm(lat,lon,cdata)` and `h = patchm(lat,lon,cdata,`*`PropertyName`*`,PropertyValue,...)` projects and displays patch (polygon) objects defined by their vertices given in `lat` and `lon` on the current map axes. `lat` and `lon` must be vectors. The color data, `cdata`, can be any color data designation 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.

`h = patchm(lat,lon,z,cdata)` and `h = patchm(lat,lon,z,cdata,`*`PropertyName`*`,PropertyValue,...)` allows the assignment of an altitude, `z`, to each patch object. The default altitude is `z = 0`.

Examples

```
load coast
axesm sinusoid; framem
h = patchm(lat,long,'b');
```



```
length(h)
ans =
     1
```

See Also

`patchesm`	Project patches as separate objects
`fill3m`	Project 3-D patch objects onto the current map axes
`fillm`	Project 2-D patch objects onto the current map axes

Purpose

Project data grid in the `z = 0` plane

Syntax

`h = pcolorm(map)` projects the data grid `map` on a graticule grid the size of `map` between the latitude and longitude limits of the current map axes. The handle `h` of the displayed surface can be returned.

`h = pcolorm(map,npts)` results in a graticule grid defined by npts, which is a two-element vector of the form `[latitude-points longitude-points]`. The default `npts` is `[50 100]`.

`h = pcolorm(lat,lon,map)` allows three other methods of defining the graticule grid. If `lat` and `lon` are matrices, they represent the actual graticule vertices as might be returned by `meshgrat`. If vectors, they are the representative coordinates of the rows and columns, respectively, of such a grid. If they are two-element vectors, they are treated as latitude and longitude limits, and a graticule mesh the size of the default `npts` is calculated.

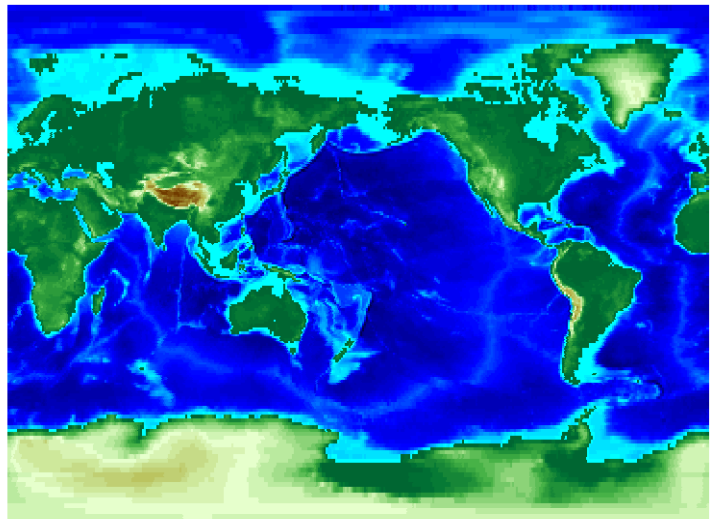
`h = pcolorm(lat,lon,map,`*`PropertyName`*`,PropertyValue,...)` allows the input of property name/property value pairs to control the surface object properties. Any property supported by the standard MATLAB function `surface` except `XData`, `YData`, and `ZData` can be altered in this manner.

Description

This function warps a data grid to a graticule mesh, which itself 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.

Examples

```
load topo
axesm miller
pcolorm(topo,[30 30])
demcmap(topo)
```



See Also

`meshgrat`	Construct map graticule grid
`meshm`	Regular data grid warped to a projected graticule mesh
`surfacem`	Data grid warped to a projected graticule mesh
`surfm`	Data grid projected to a map axes

Purpose

Convert pixel coordinates to latitude-longitude coordinates

Syntax

`[lat, lon] = pix2latlon(r,row,col)` calculates latitude-longitude coordinates `lat`, `lon` from pixel coordinates `row`, `col`. `r` is a 3-by-2 referencing matrix defining a two-dimensional affine transformation from pixel coordinates to spatial coordinates. `row` and `col` are vectors or arrays of matching size. The outputs `lat` and `lon` have the same size as `row` and `col`.

Example

```
% Find the latitude and longitude of the upper left and lower right
% outer corners of a 2-by-2 degree gridded data set.
R = makerefmat([1, 89], 2, 2);
[UL_lat, UL_lon] = pix2latlon(R, .5, .5)
[LR_lat, LR_lon] = pix2latlon(R, 90.5, 180.5)
```

See Also

`latlon2pix`, `makerefmat`, `pix2map`

pix2map

Purpose

Convert pixel coordinates to map coordinates

Syntax

`[x,y] = pix2map(R,row,col)` calculates map coordinates `x,y` from pixel coordinates `row,col`. `R` is a 3-by-2 referencing matrix defining a two-dimensional affine transformation from pixel coordinates to spatial coordinates. `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.

Example

```
% Find the map coordinates for the pixel at (100,50).
R = worldfileread('concord_ortho_w.tfw');
[x,y] = pix2map(R,100,50)
```

See Also

`makerefmat`, `map2pix`, `pix2latlon`, `worldfileread`

Purpose

Compute pixel centers for georeferenced image or data grid

Syntax

`[x,y] = pixcenters(R, height, width)` returns the spatial coordinates of a spatially-referenced image or regular gridded data set. `R` is the 3-by-2 affine referencing matrix. `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 `height`-by-1 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 struct 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.

Notes

For more information on referencing matrices, see the documentation for `makerefmat`.

`pixcenters` is useful for working with `surf`, `mesh`, or `surface`, and for coordinate transformations.

Example

```
[Z,R] = arcgridread('MtWashington-ft.grd');
[x,y] = pixcenters(R, size(Z));
h = surf(x,y,Z); axis equal; colormap(demcmap(Z))
set(h,'EdgeColor','none')
xlabel('x (easting in meters)')
ylabel('y (northing in meters')
zlabel('elevation in feet')colormap(terrain)
```

See Also

`arcgridread`, `makerefmat`, `mapbbox`, `mapoutline`, `pix2map`, `worldfileread`

The help for `mapshow` provides an alternative version of the preceding example.

Purpose

Project parallel labels on a map axes

Syntax

`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 the strings `'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 in this guide.

Parallel label handles can be returned in `h` if desired.

See Also

`axesm`	Define map axes and set map properties
`setm`	Set map properties
`mlabel`	Meridian labels projected on a map axes

plot3m

Purpose

Project line objects onto current map axes in 3-D space

Syntax

`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`.

`h = plot3m(lat,lon,`*`linetype`*`)` allows the specification of the line style, where *`linetype`* is any string recognized by the MATLAB `line` function.

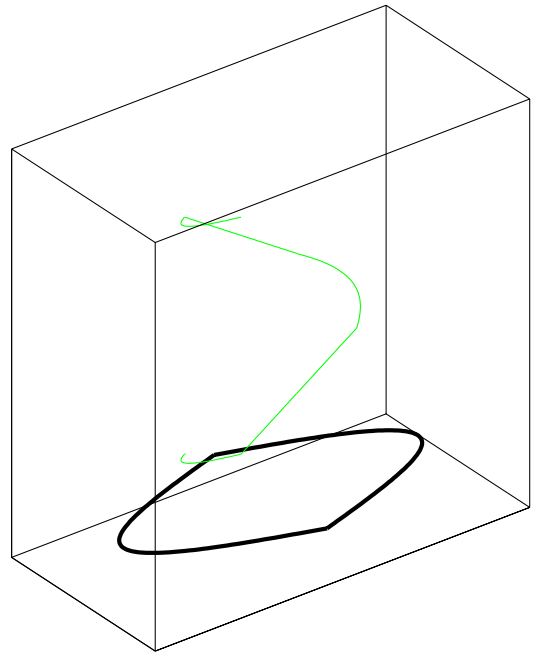
`h = plot3m(lat,lon,`*`PropertyName`*`,PropertyValue,...)` allows the specification of any number of property name/property value pairs for any properties recognized by the MATLAB `line` function except for `XData`, `YData`, and `ZData`.

Description

`plot3m` is the mapping equivalent of the MATLAB `plot3` function.

Example

```
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,'g')
```



See Also

`linem`	Project line objects onto current map axes
`plot3`	Plot lines and points in 3-D space (see the online MATLAB Function Reference documentation)
`plotm`	Project lines onto current map axes in 2-D space

plotm

Purpose

Project 2-D lines onto current map axes

Syntax

`h = 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. 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 = plotm(lat,lon,`*`linetype`*`)` allows the specification of the line style, where *`linetype`* is any string recognized by the MATLAB `line` function.

`h = plotm(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`.

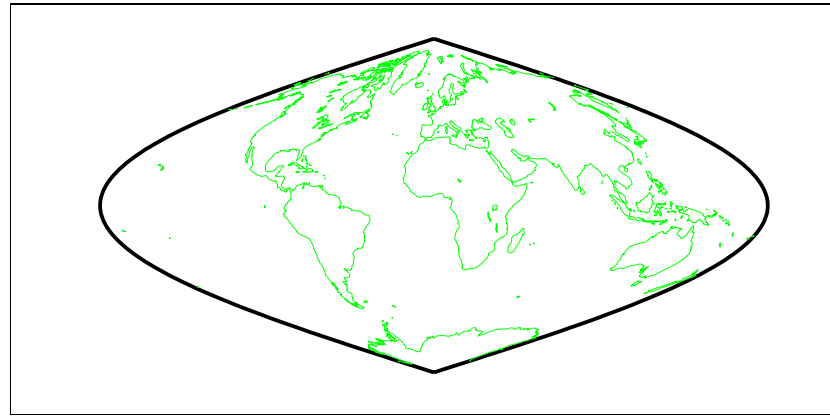
`h = plotm([lat lon],...)` allows the coordinates to be packed into a single two-column matrix.

Description

`plotm` is the mapping equivalent of the MATLAB `plot` function.

Example

```
load coast
axesm sinusoid; framem
plotm(lat,long,'g')
```



See Also

`linem`	Project line objects onto current map axes
`plot`	Linear plot (see the online MATLAB Function Reference documentation)
`plot3m`	Project lines onto current map axes in 3-D space

polcmap

Purpose

Colormap for political maps

Syntax

`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. Larger maximum saturation values produce brighter, more saturated colors. If omitted, the default is 0.5.

`polcmap(ncolors,huelimits,saturationlimits,valuelimits)` controls the colors. Hue, saturation, and value are randomly selected 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.

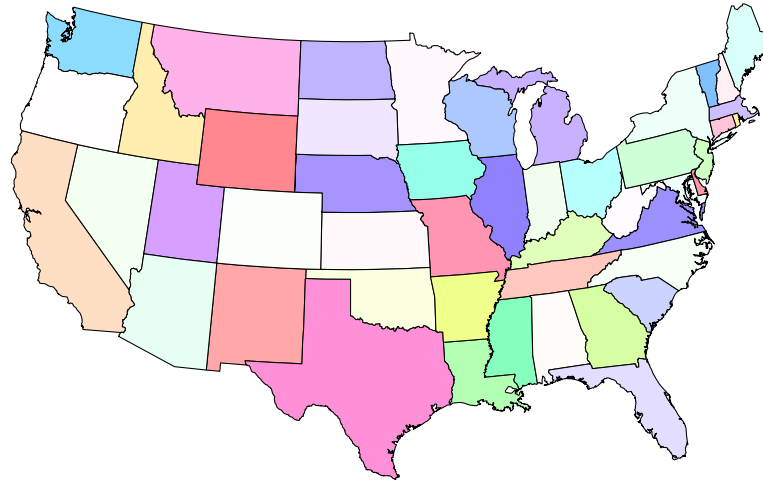
`cmap = polcmap(...)` returns the colormap without applying it to the figure.

Remarks

You cannot use `polcmap` to alter the colors of patches drawn by `geoshow` or `mapshow`. The patches must have been rendered by `displaym`.

Example

```
usamap('conus','none')
framem off; gridm off; mlabel off; plabel off

load usalo
displaym(state)
polcmap
```



See Also

`demcmap`	Colormaps for digital elevation maps
`colormap`	Color lookup table

polybool

Purpose

Perform polygon Boolean operations

Syntax

`[lat,lon] = polybool(`*`flag`*`,x1,y1,x2,y2)` performs the polygon Boolean operation identified by *`flag`*. Valid flag strings are one of the following: (`'intersection'`, `'and'`, `'&'`), (`'union'`, `'or'`, `'|'`, `'+'`, `'plus'`), (`'exclusiveor'`, `'xor'`) and (`'subtraction'`, `'minus'`, `'-'`). The polygon inputs are `NaN`-delimited vectors or cell arrays containing individual polygons in each element with the outer face separated from the subsequent inner faces by `NaN`s. The result is output as `NaN`-delimited vectors.

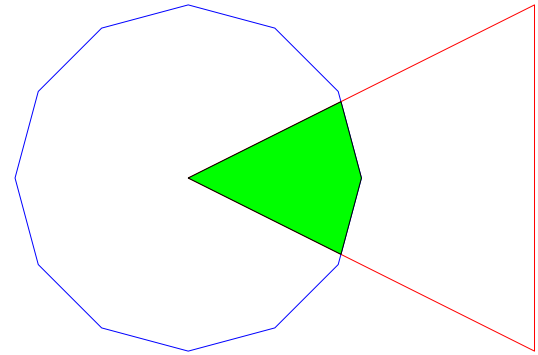
`[lat,lon] = polybool(`*`flag`*`,x1,y1,x2,y2,`*`outputformat`*`)` controls the format of the resulting polygons. If *`outputformat`* is `'vector'`, the result is returned as vectors with `NaN`s separating the faces. No distinction is made between outer and inner faces of polygons. If *`outputformat`* is `'cutvector'`, inner faces are connected to the enclosing polygon face by inserting a cut. `If` *`outputformat`* is `'cell'`, the result is returned as cell arrays containing individual polygons in each element, with the outer face separated from the subsequent inner faces by `NaN`s. If omitted, `'vector'` is assumed.

Limitations

Polygons are assumed to be in a Cartesian coordinate system. Therefore, geographic data that encompasses a pole cannot be used directly. Use `flatearthpoly` to convert polygons to Cartesian coordinates.

Example

```
theta = (0:pi/6:2*pi)';
lat1 = sin(theta);
lon1 = cos(theta);
lat2 = [0 1 -1 0]';
lon2 = [0 2 2 0]';
[latb,lonb] = polybool('intersection',lat1,lon1,lat2,lon2);

axesm miller
plotm(lat1,lon1,'b')
plotm(lat2,lon2,'r')
patchm(latb,lonb,'g')
```



```
[latb2,lonb2] = polybool('xor',lat1,lon1,lat2,lon2,'cell')

latb2 =

    [ 6x1 double]
    [15x1 double]


lonb2 =

    [ 6x1 double]
    [15x1 double]
```

See Also

`bufferm`	Compute buffer zones for vector data
`polyjoin`	Convert polygon segments from cell array to vector format
`polysplit`	Extract segments of `NaN`-delimited polygon vectors to cell arrays

polycut

Purpose

Polygon branch cuts for holes

Syntax

`[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

`polybool`, `polysplit`, `polyjoin`

Purpose

Convert polygon segments from cell array to vector format

Syntax

`[lat,lon] = polyjoin(latcells,loncells)` converts polygons from cell array format to vector format. In cell array format, each element of the cell array is a separate polygon. Each polygon can consist of an outer contour followed by holes separated with `NaN`s. In vector format, each vector can contain multiple faces separated by `NaN`s. There is no distinction between outer contours and holes.

Example

```
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
```

See Also

`polybool`	Perform polygon Boolean operations
`polycut`	Implement polygon branch cuts for holes
`polysplit`	Extract segments of NaN-delimited polygon vectors to cell arrays

polymerge

Purpose

Merge line segments with matching endpoints

Syntax

`[lat2,lonc2 = polymerge(lat,lon)` combines vector line segments with identical endpoints. `polymerge` compares the endpoints of all line segments and combines those that match. The line can be input as vectors of latitude and longitude with `NaN`s delimiting segments. The line can also be input as cell arrays, with each element of a cell array containing a line segment. The resulting line is in the same format as the input.

`[lat2,lonc2 = polymerge(lat,lon,tol)` combines line segments whose endpoints are separated by less than the circular tolerance. If omitted, `tol = 0` is assumed. The tolerance is in the same units as the polygon input.

`[lat2,lonc2 = polymerge(lat,lon,tol,`*`outputformat`*`)` controls the format of the resulting polygons. If *`outputformat`* is `'vector'`, the result is returned as vectors with `NaN`s separating the segments. If *`outputformat`* is `'cell'`, the result is returned as cell arrays containing segments in each element. If omitted, `'vector'` is assumed.

Example

```
lat = [1 2 3 NaN 6 7 8 9 NaN 6 5 4 3 NaN 12 13 14 NaN 9 10 11 12]';
lon = lat;
[lat2,lon2] = polymerge(lat,lon);

[lat2 lon2]
ans =
    14    14
    13    13
    12    12
    12    12
    11    11
    10    10
     9     9
     9     9
     8     8
     7     7
     6     6
     6     6
     5     5
     4     4
     3     3
```

```
     3     3
     2     2
     1     1
```

See Also

`polyjoin`	Convert polygon segments from cell array to vector format
`polysplit`	Extract segments of `NaN`-delimited polygon vectors to cell arrays

polysplit

Purpose

Extract segments of NaN-delimited polygon vectors to cell arrays

Syntax

`[latcells,loncells] = polysplit(lat,lon)` returns the NaN-delimited segments of the vectors `lat` and `lon` as cell arrays. Each element of the cell array contains one segment.

Example

```
lat = [1 2 3 NaN 4 NaN 5 6 7 8 9]';
lon = [9 8 7 NaN 6 NaN 5 4 3 2 1]';
[latcells,loncells] = polysplit(lat,lon);

[latcells' loncells']
ans =
    [3x1 double]    [3x1 double]
    [         4]    [         6]
    [5x1 double]    [5x1 double]
```

See Also

`polybool`	Perform polygon Boolean operations
`polyjoin`	Convert polygon segments from cell array to vector format

Purpose

Compute line or polygon intersection points

Syntax

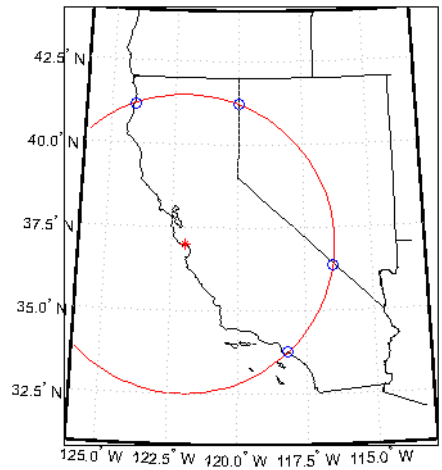
`[xi,yi] = polyxpoly(x1,y1,x2,y2)` returns the intersection points of two sets of lines and/or polygons.

`[xi,yi] = polyxpoly(...,'unique')` returns only unique intersections.

`[xi,yi,ii] = polyxpoly(...)` also returns a two-column index of line segment numbers corresponding to the intersection points.

Example

```
[lat,lon]=extractm(usalo('state'),'california');
[lat,lon]=reducem(lat,lon);

lat0 = 37; lon0 = -122; rad = 500;
[latc,lonc] = scircle1(lat0,lon0,km2deg(rad));

usamap('california','lineonly')
plotm(lat0,lon0,'r*')
plotm(latc,lonc,'r')

[loni,lati] = polyxpoly(lon,lat,lonc,latc);
plotm(lati,loni,'bo')
```



See Also

`crossfix`	Compute cross fix positions for bearings and ranges
`gcxgc`	Compute the intersection points between two great circles
`gcxsc`	Compute the intersection points between a great and a small circle
`navfix`	Perform Mercator-based navigational fixing
`rhxrh`	Compute the intersection point between two rhumb lines
`scxsc`	Compute the intersection points for pairs of small circles

Purpose

View map at printed size

Syntax

```
previewmap
```

Background

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.

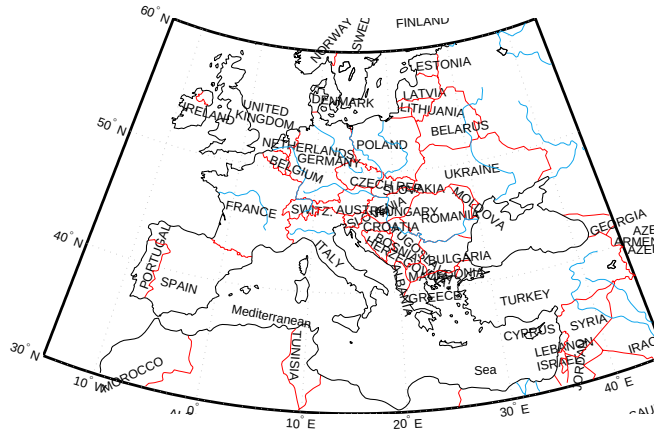
Description

`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.

Examples

Is the text small enough to avoid overlapping in a map of Europe?

```
worldmap europe
h = displaym(worldlo('POtext'));
trimcart(h)
rotatetext(h)

orient landscape
tightmap
hidem(gca)
previewmap
```



Limitations

The figure cannot be made larger than the screen.

See Also

`pagedlg`	Page position dialog box
`paperscale`	Figure paper size for a given map scale
`axesscale`	Resize axes for equivalent scale

Purpose

Project a displayed graphics object on a map axes

Syntax

`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!

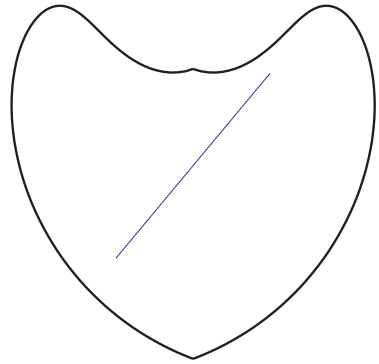
`project(h,'xy')` specifies that the `XData` of the unprojected objects corresponds to longitudes and the `YData` to latitudes. This is the default assumption.

`project(h,'yx')` specifies that the `XData` of the unprojected objects corresponds to latitudes and the `YData` to longitudes.

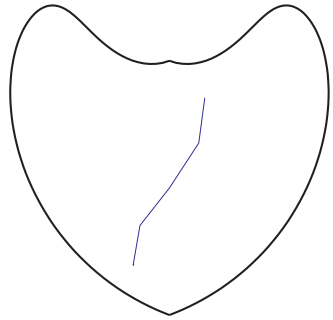
Example

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

`linem`	Project line objects
`patchm`	Project patch objects
`surfacem`	Project data grid
`textm`	Project text objects

Purpose

Forward map projection using the PROJ.4 map projection library

Syntax

`[x, y] = projfwd(proj, lat, lon)` returns the `x` and `y` map coordinates from the forward projection transformation. `proj` is a structure defining the map projection. `proj` can be an mstruct or a GeoTIFF `info` structure. `lat` and `lon` are arrays of the latitude and longitude coordinates.

For a complete list of GeoTIFF `info` and map projection structures that you can use with `projfwd`, see the reference page for `projlist`.

Examples

Example 1

Display a projected image and its corner points.

1. Get the `info` structure for the image:

```
info = geotiffinfo('boston.tif');
```

2. Project the latitude and longitude bounding box corners of the georeferenced image `boston.tif`.

```
[x, y] = projfwd(info, ...
                 info.CornerCoords.LAT, ...
                 info.CornerCoords.LON)
```

3. Display the image and corners:

```
figure
mapshow('boston.tif')
mapshow(gca, [x; x(1)],[y; y(1)],'Color', 'cyan')
```

Example 2

Overlay `boston.tif` on top of `boston_ovr.jpg`.

1. Obtain the `info` structure:

```
info = geotiffinfo('boston.tif')
```

2. Read the `boston_ovr.jpg` image and its worldfile:

```
[I, cmap] = imread('boston_ovr.jpg')
R = worldfileread(getworldfilename('boston_ovr.jpg')
```

3. Create a latitude and longitude grid:

```
[lon, lat] = pixcenters(R, size(I), 'makegrid');
```

4 Project the grid to the same projection as `boston.tif`:

```
[x, y] = projfwd{info, lat, lon);
```

5 Overlay `boston_ovr.jpg` on `boston.tif`:

```
figure
mapshow(x, y, I, cmap);
hold on
mapshow'boston.tif');
```

See Also

`geotiffinfo`, `mfwdtran`, `minvtran`, `projinv`, `projlist`

Purpose

Inverse map projection using the PROJ.4 map projection library

Syntax

`[lat, lon] = projinv(proj, x, y)` returns the latitude and longitude values from the inverse projection transformation. `proj` is a structure defining the map projection. `proj` can be a map projection mstruct or a GeoTIFF `info` structure. `x` and `y` are *x-y* map coordinate arrays. For a complete list of GeoTIFF `info` and map projection structures that you can use with `projinv`, see the reference page for `projlist`.

Example

Display `boston.tif` in a Mercator projection.

1. Obtain the info structure and read the image:

```
info = geotiffinfo('boston.tif');
[I, cmap] = geotiffread('boston.tif';
```

2. Create a grid for the image and convert it to latitude and longitude:

```
[x, y] = pixcenters(info.RefMatrix, size(I), 'makegrid');
[lat, lon] = projinv(info, x, y);
```

3. Use `extractm` to obtain state boundary coordinates for Massachusetts from the `usahi` data base, and create a Mercator projection using its latitude and longitude limits:

```
figure; axesm('mercator')
[slat, slon] = extractm(usahi('stateline'), 'Massachusetts');
setm('maplonlimit', [min(slon(:)) max(slon(:))], ...
     'maplatlimit', [min(slat(:)) max(slat(:))])
```

4. Display the stateline boundary and the image:

```
geoshow(slat, slon, 'color', 'black')
geoshow(lat, lon, ind2rgb8(I, cmap))
tightmap
```

See Also

`geotiffinfo`, `mfwdtran`, `minvtran`, `projfwd`, `projlist`

projlist

Purpose

List map projections supported by `projfwd` and `projinv`

Syntax

`projlist(listmode)` displays a table of projection names, IDs, and availability. *`listmode`* is a string with value `'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 map projection IDs that are available for use with `projfwd` and `projinv`. The output structure contains the fields
  - `Name` — Projection name
  - `MapProjection` — Projection ID string
- `geotiff` — Lists the GeoTIFF projection IDs that are available for use with `projfwd` and `projinv`. The output structure contains the fields
  - `GeoTIFF` — GeoTIFF projection ID string.
  - `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 the fields
  - `GeoTIFF` — GeoTIFF projection ID string
  - `MapProjection` — Projection ID string
- `all`— Lists the map and GeoTIFF projection IDs that are available for use with `projfwd` and `projinv`. The output structure contains the fields
  - `GeoTIFF`— GeoTIFF projection ID string
  - `MapProjection` — Projection ID string
  - `info` — Logical array with values 1 or 0
  - `mstruct` — Logical array with values 1 or 0

Description

`projfwd` and `projinv` can be used to process certain forward or inverse map projections. These functions are implemented in C using the `PROJ.4` library. `projlist` provides a convenient list of the projections that can be used with `projfwd` or `projinv`. Because `projfwd` and `projinv` accept either a map projection structure (mstruct) or a GeoTIFF info structure, `projlist` provides separate lists for each case. It can also list the projections for which a GeoTIFF info structure can be converted to an mstruct.

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

putpole

Purpose

Compute the origin of a transformed coordinate system

Syntax

`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 units string. The default is `'degrees'`.

Description

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.

Examples

Pull the North Pole down the 0° meridian by 30° to 60°N. What is the resulting origin vector?

```
origin = putpole([60 0])
origin =
   30.0000         0         0
```

This makes sense: when the pole slid down 30°, the point that was 30° 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
```

See Also

`neworig`	Transform regular data grid to new coordinate system
`org2pol`	Pole of a transformed coordinate system based on a new origin

quiver3m

Purpose

Project three-dimensional quiver plot on map axes

Syntax

`h = quiver3m(lat,lon,alt,u,v,w)` displays *velocity* vectors with components `(u,v,w)` at the geographic points `(lat,lon)` and altitude `alt` on a displayed map axes. The inputs `u`, `v`, and `w` 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,u,v,w,linespec)` allows the control of the line specification of the displayed vectors with a *`linespec`* string recognized by the MATLAB `line` function. If symbols are indicated in *`linespec`*, they are plotted at the start points of the vectors, i.e., the input points (`lat`,`lon`,`alt`).

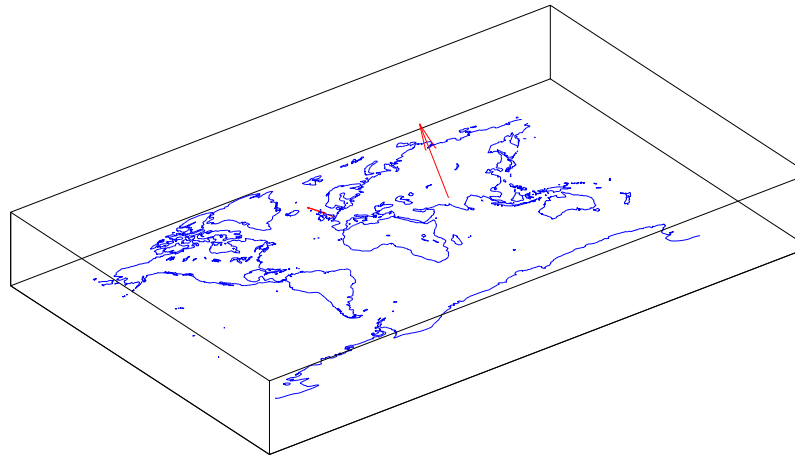
`h = quiver3m(lat,lon,alt,u,v,w,linespec,'filled')` results in the filling in of any symbols specified by *`linespec`*.

`h = quiver3m(lat,lon,alt,u,v,w,scale)`,
`h = quiver3m(lat,lon,alt,u,v,w,linespec,scale)` and
`h = quiver3m(lat,lon,alt,u,v,w,linespec,scale,'filled')` alters 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+u`,`lon+v`,`alt+w`).

Examples

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 coast
axesm miller; view(3)
plotm(lat,long)
lat0 = [51.5,29]; lon0 = [0 77.5]; alt = [0 0];
u = [-40 50]; v = [-70 10]; w = [1 1];
quiver3m(lat0,lon0,alt,u,v,w,'m')
```



See Also

`quiverm`	Two-dimensional quiver plot projected on map axes
`quiver3`	Three-dimensional velocity plot (see the online MATLAB Function Reference documentation)

quiverm

Purpose

Project two-dimensional quiver plot on map axes

Syntax

`h = quiverm(lat,lon,u,v)` displays *velocity* vectors with components `(u,v)` at the geographic points `(lat,lon)` on displayed map axes. All four inputs should be in the `AngleUnits` of the map axes. The inputs `u` and `v` 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,u,v,`*`linespec`*`)` allows the control of the line specification of the displayed vectors with a *`linespec`* string recognized by the MATLAB `line` function. If symbols are indicated in *`linespec`*, they are plotted at the start points of the vectors, i.e., the input points (`lat`,`lon`).

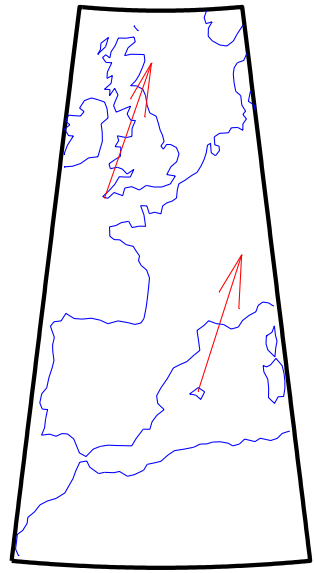
`h = quiverm(lat,lon,u,v,`*`linespec`*`,'filled')` results in the filling in of any symbols specified by *`linespec`*.

`h = quiverm(lat,lon,u,v,scale)` and `h = quiverm(lat,lon,u,v,` *`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+u`,`lon+v`).

Example

Plot quiver vectors from Land's End (50°N,5.4°W) and Majorca (39.7°N,2.9°E) in a direction corresponding to +5° latitude and +3° longitude. Use automatic scaling.

```
load coast
axesm('eqacon','MapLatLimit',[30 60],'MapLonLimit',[-10 10])
framem; plotm(lat,long)
lat0 = [50 39.7]; lon0 = [-5.4 2.9];
u = [5 5]; v = [3 3];
quiverm(lat0,lon0,u,v,'r')
```



See Also

`quiver3m`	Three-dimensional quiver plot projected on map axes
`quiver`	Quiver or velocity plot (see the online MATLAB Function Reference documentation)

rad2deg

Purpose

Convert angle (or distance) units from radians to degrees

Syntax

`anglout = rad2deg(anglin)` converts angles input in radians to the equivalent measure in degrees.

Remarks

This is both an angle conversion function and a distance conversion function, because arc length can be a measure of distance in either radians or degrees (provided the radius is known).

Example

There are 180° in π radians:

```
anglout = rad2deg(pi)
anglout =
   180
```

See Also

`angledim`	Convert angle units
`deg2dms` `dms2rad`	Other direct angle conversion functions
`deg2rad`	Convert degrees to radians
`distdim`	Convert distances between different units
`nm2km` `sm2deg`	Other direct distance conversion functions

Purpose

Convert angle units from radians to dms or dm

Syntax

`anglout = rad2dms(anglin)` converts angles input in radians to the equivalent measure in degrees-minutes-seconds (*dms*) format.

`angleout = rad2dm(anglin)` converts angles input in radians to the equivalent measure in degrees-minutes (*dm*) format. This is the dms format, properly rounded to just degrees and minutes.

Example

```
rad2dms(1)
ans =
       5717.45

rad2dm(1)
ans =
       5718.00
```

See Also

`angledim`	Convert angle units
`deg2rad` `dms2rad`	Other direct angle conversion functions
`dms2mat`	Convert from dms to separated matrix components
`mat2dms`	Convert from separated matrices input to dms

rad2km, rad2nm, rad2sm

Purpose

Convert distance from radians to kilometers, nautical miles, or statute miles

Syntax

`distout = rad2km(distin)` converts the input distance given in radians to kilometers.

`distout = rad2nm(distin)` and `distout = rad2sm(distin)` work identically, except that the output units are nautical miles and statute miles, respectively.

`distout = rad2km(distin,radius)` specifies the radius of the sphere to use, since a radian of arc length covers less distance, for example, on Mars than it would on the Earth. You can enter the radius as a number in kilometers, as a call to the `almanac` function (e.g., `almanac('mars','radius','km')`), again in the appropriate units, or you can pass in a string planet name (e.g., `'mars'`), and the function will make the appropriate call to the `almanac` function. The radius of the Earth is the default.

For `distout = rad2nm(distin,radius)` and `distout = rad2sm(distin,radius)`, make sure your input radius is in the appropriate units, or just use the planet name string.

Examples

How long is a trip around the equator in statute miles?

```
distout = rad2sm(2*pi)
distout =
   2.4874e+04
```

How about on Jupiter?

```
distout = rad2sm(2*pi,'jupiter')
distout =
   2.7284e+05
```

See Also

`distdim`	Convert distances between different units
`nm2km` `sm2deg`	Other direct distance conversion functions
`rad2deg`	Convert radians to degrees

Purpose

Calculate radii of curvature on an ellipsoid

Syntax

`r = rcurve(ellipsoid,lat)` or `r = rcurve('parallel',ellipsoid,lat)` returns the parallel radius of curvature at the latitude `lat` for a given elliptical definition, where `ellipsoid` is a two-element ellipsoid vector. This is the radius of the small circle encompassing the ellipsoid at the given latitude. The radius is a distance in units consistent with the semimajor axis, the first element of `ellipsoid`.

`r = rcurve(ellipsoid,lat,`*`units`*`)` specifies the units of the input `lat`, where *`units`* is any valid angle units string. The default is `'degrees'`.

`r = rcurve('meridian',ellipsoid,lat,`*`units`*`)` returns the meridianal radius, which is the radius of curvature at the latitude `lat` for the ellipse described by a meridian on the ellipsoid.

`r = rcurve('transverse',ellipsoid,lat,`*`units`*`)` returns the transverse radius, which is the radius of a curve described by the intersection of the ellipsoid with a plane normal to the surface of the ellipsoid at the latitude `lat`.

Examples

The radii of curvature of the default ellipsoid at 45°, in kilometers:

```
r = rcurve('transverse',almanac('earth','ellipsoid','km'),45,...
           'degrees')
r =
   6.3888e+03

r = rcurve('meridian',almanac('earth','ellipsoid','km'),45,...
           'degrees')
r =
   6.3674e+03

r = rcurve('parallel',almanac('earth','ellipsoid','km'),45,...
           'degrees')
r =
   4.5024e+03
```

See Also

`rsphere` Radii of auxiliary spheres

readfields

Purpose

Read fields or records from a fixed-format file

Syntax

`struc = readfields(fname,fstruc)` reads all the records from a fixed format file. *`fname`* is a string 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 MATLAB 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/line-feed line ending everywhere except the last record. File sizes are not checked when an open file is provided.

Remarks

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 string containing the field name under which the read data is stored in the output structure, and `type` is a format string 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 string 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 string. Fortran-style double-precision output such as `'0.0D00'` can be read using a type string such as `'%nD'`, where `n` is the number of characters per element. This is an extension to the C-style format strings accepted by `sscanf`. Users unfamiliar with C should note that `'%d'` is preferred over `'%i'` for formatted integers. MATLAB 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 for strings, and the number of elements for formatted data.

You can omit fields from all output by providing an empty string for the `fstruc` name field.

See Also

`grepfields`	Search within fields of a fixed format file
`readmtx`	Read a matrix stored in a file
`textread`	Read formatted text files
`spcread`	Read columns of data from an ASCII text file
`dlmread`	Read ASCII delimited file

readfk5

Purpose

Read the Fifth Fundamental Catalog of Stars

Syntax

`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.

Remarks

Positions are given in terms of right ascension and declination. “Astronomical Data” in Chapter 8 of the Mapping Toolbox User’s Guide documentation shows how to convert these to latitude and longitude for display by the Mapping Toolbox.

The Fifth Fundamental Catalog of Stars (FK5), Parts I and II, is available over the Internet by anonymous FTP at the following:

```
ftp://adc.gsfc.nasa.gov/pub/adc/archives/catalogs/1/1149A/
ftp://adc.gsfc.nasa.gov/pub/adc/archives/catalogs/1/1175/
```

The readme files have more complete descriptions of the data.

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'
```

See Also

`dms2deg`	Convert angles from deg:min:sec to degrees
`hms2hr`	Convert time from hrs:min:sec to hours
`scatterm`	Construct a thematic map with proportional symbols

References

See references [5] and [6] in the Bibliography located at the end of this chapter.

readmtx

Purpose

Read a matrix stored in a file

Syntax

`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`* strings 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 string 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 MATLAB 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.

Remarks

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 string 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 string 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 string such as `'%nD'`, where `n` is the number of characters per element. This is an extension to the C-style format strings accepted by `sscanf`. Users unfamiliar with C should note that `'%d'` is preferred over `'%i'` for formatted integers. MATLAB 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* line-feed (Macintosh, UNIX), or 2 for platforms with carriage returns *and* line-feeds (DOS).

See Also

`readfields`	Read fields or records from a fixed format file
`textread`	Read formatted text files
`spcread`	Read columns of data from an ASCII text file
`dlmread`	Read ASCII delimited file

Purpose

Compute position at a specified azimuth and range

Syntax

`[latout, lonout] = reckon(lat, lon, rng, az)`, for scalar inputs, calculates a position (`latout, lonout`) at a given range `rng` and azimuth `az` along a great circle from a starting point defined by `lat` and `lon`. `lat` and `lon` are in degrees. The range is in degrees of arc length on a sphere. The input azimuth is in degrees, measured clockwise from due north. `reckon` calculates multiple positions when given four non-scalar inputs of matching size.

`[latout, lonout] = reckon(lat, lon, rng, az, units)`, where `units` is any valid angle units string, specifies the angular units of the inputs and outputs, including `rng`. The default value is `'degrees'`.

`[latout, lonout] = reckon(lat, lon, rng, az, ellipsoid)` calculates positions along a geodesic on an ellipsoid, as specified by the two-element vector `ellipsoid`. The range, `rng`, is in linear distance units matching the units of the semimajor axis of the ellipsoid (the first element of `ellipsoid`).

`[latout, lonout] = reckon(lat, lon, rng, 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

What are the coordinates of the point 600 nautical miles northwest of London, UK (51.5°N,0°), in a great circle sense?

```
dist = nm2deg(600)  % convert nm distance to degrees
dist =
    9.9933

pt1 = reckon(51.5,0,dist,315)  % northwest is 315 degrees
pt1 =
   57.8999  -13.3507
```

Now, where would a plane taking off from London and traveling on a constant northwesterly course for 600 nautical miles end up?

```
pt2 = reckon('rh',51.5,0,dist,315)
pt2 =
```

```
    58.5663  -12.3699
```

How far apart are these points (distance in great circle sense)?

```
separation = distance('gc',pt1,pt2)
separation =
    0.8430

nmsep = deg2nm(separation)  % convert answer to nautical miles
nmsep =
   50.6156
```

Over 50 nautical miles separate the two points.

See Also

`azimuth`	Azimuth between two points on the globe
`distance`	Calculate distances between points on a ellipsoid
`distdim`, `distance`	Distance between two points on the globe
`km2deg`	Convert distance units
`dreckon`	Navigational dead reckoning

`track`
`track1`
`track2`
	Tracing paths on the globe

Purpose

Reduce the number of points in vector data

Syntax

`[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.

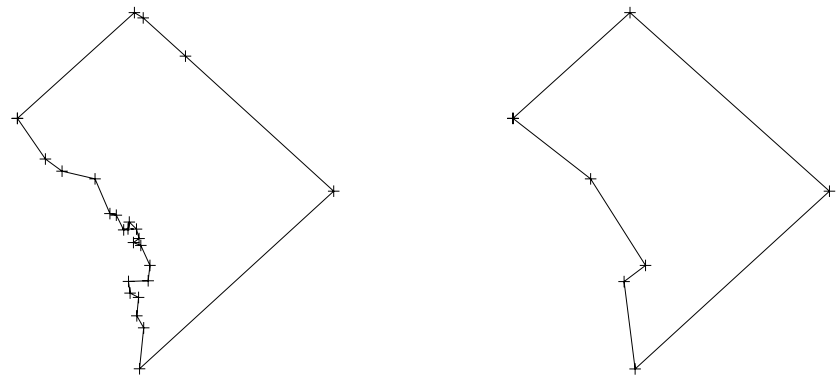
`[latout,lonout,cerr] = reducem(...)` in addition returns a measure of the error introduced by the simplification. The output `cerr` is the difference in the arc length of the original and reduced data, normalized by the original length.

`[latout,lonout,cerr,tol] = reducem(...)` also returns the tolerance used in the reduction, which is useful when the tolerance is computed automatically.

Examples

Compare the original and reduced outlines of the District of Columbia from the `usahi` atlas data:

```
load usahi
[lat,long] = extractm(stateline,'district');
[latout,lonout,cerr] = reducem(lat,long);
```



Remarks

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

`interpm`	Interpolate vector data to a specified data separation
`resizem`	Resize a data grid

Purpose

Convert a referencing matrix to a referencing vector

Syntax

`refvec = refmat2vec(R,s)` converts a referencing matrix, `R`, to the 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 with elements [cells/angleunit north-latitude west-longitude].

Example

```
% Verify the conversion of the geoid referencing vector to a
% referencing matrix.
load geoid;
R = refvec2mat(geoidlegend, size(geoid));
V = refmat2vec(R, size(geoid));
```

See Also

`makerefmat`, `refvec2mat`

refvec2mat

Purpose

Convert a referencing vector to a referencing matrix

Syntax

`R = refvec2mat(refvec,s)` converts a referencing vector, `refvec`, to the referencing matrix `R`. `refvec` is a 1-by-3 referencing vector with elements [cells/angleunit north-latitude west-longitude]. `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.

Example

```
% Convert the geoid referencing vector to a referencing matrix
 load geoid;
R = refvec2mat(geoidlegend, size(geoid));
```

See Also

`makerefmat`, `refmat2vec`

Purpose

Resize a data grid

Syntax

`map = resizem(map0,m)` resizes an original data grid, `map0`, by a resizing factor `m`. For example, if `m` is 0.5, the number of rows and the number of columns will be cut in half. The result is the resized map `map`.

`map = resizem(map0,[r c])` resizes `map0` so that the output map, `map`, has `r` rows and `c` columns.

`map = resizem(map0,m,`*`method`*`)` specifies the method of interpolation. The string *`method`* `'nearest'` results in nearest-neighbor interpolation, the default, `'cubic'` results in bicubic interpolation, and `'linear'` results in bilinear interpolation.

`[map,refvec] = resizem(map0,m,refvec0)` resizes a regular data grid with a referencing vector, `refvec0`, and returns a regular data grid and its referencing vector, `refvec`.

This case requires a resizing factor, `m`, rather than the `[r c]` vector, as referencing vectors only have meaning for regular data grids (that is, rows represent the same angular dimension as columns).

When the map size is being reduced, `resizem` lowpass filters the map before interpolating to avoid aliasing. By default, this filter is designed using FIR1, but can be specified using

- `resizem(...,`*`method`*`,h)` The default filter is 11-by-11
- `resizem(...,`*`method`*`,n)` uses an `n`-by-`n` filter
- `resizem(...,`*`method`*`,0)` turns off the filtering

Unless a filter `h` is specified, `resizem` does not filter when `'nearest'` is used. These filters are associated with the MATLAB Image Processing Toolbox.

Example

Double the size of a map:

```
map = [1 2; 3 4]
map =
      1  2
      3  4

newmap = resizem(map,2)
newmap =
```

```
1   1   2   2
1   1   2   2
3   3   4   4
3   3   4   4
```

Purpose

Restack objects within the axes

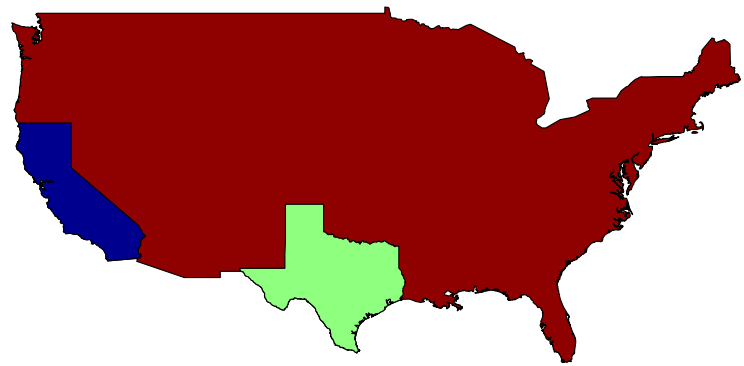
Syntax

`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 string recognized by `handlem`. Recognized *`position`* strings are `'top'`, `'bottom'`, `'bot'`, `'up'`, or `'down'`.

Examples

Restack the patch objects of California and Texas to lie above the U.S.

```
axesm miller
load usalo

h = displaym(state,'california');
displaym(state,'texas');
displaym(conus)

restack(h,'top')
restack('Texas','top')
```



Remarks

This function is the function-line equivalent of the stacking buttons in the `mobjects` graphical user interface. The stacking order is the order of the children of the axes.

See Also

`mobjects` GUI for manipulating objects displayed on an axes

rhxrh

Purpose

Provide intersection coordinates for pairs of rhumb lines

Syntax

`[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 string. The default units are `'degrees'`.

Description

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 discussion of *Limitations* below.

Rhumb line notation consists of a point on the line and the constant azimuth of the line.

Examples

Given a starting point at (10°N,56°W), a plane maintains a constant heading of 35°. Another plane starts at (0°,10°W) and proceeds at a constant heading of 310° (–50°). Where would their two paths cross each other?

```
[newlat,newlong] = rhxrh(10,-56,35,0,-10,310)
newlat =
   26.9774
newlong =
  -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 85° — it is easy to imagine another rhumb line, say one with a heading of 0°, 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`
	Other intersection functions

`navfix`	Navigational fixing

Purpose

Use workspace variables to construct a cell array for input to the `mlayers` tool

Syntax

`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` — Interactive geographic data structure manipulation (see Chapter 12, “GUI Reference”)

Purpose

Transform vector data to a new coordinate system based on a new origin

Syntax

`[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 0°. 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 units string. The default is `'radians'`. Note that this default is different from that of most functions.

Description

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 (90° 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 (23°S,43°W) in a coordinate system in which New York (41°N,74°W) 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`	Transform regular data grid to new coordinate system based on a new origin
`newpole`	Select point to place at North Pole
`org2pol`	Pole of transformed coordinate system
`putpole`	Origin of transformed coordinate system

Purpose

Rotate text to the projected graticule

Syntax

`rotatetext` rotates displayed text objects to account for the curvature of the graticule. The objects are selected interactively from a graphical user interface.

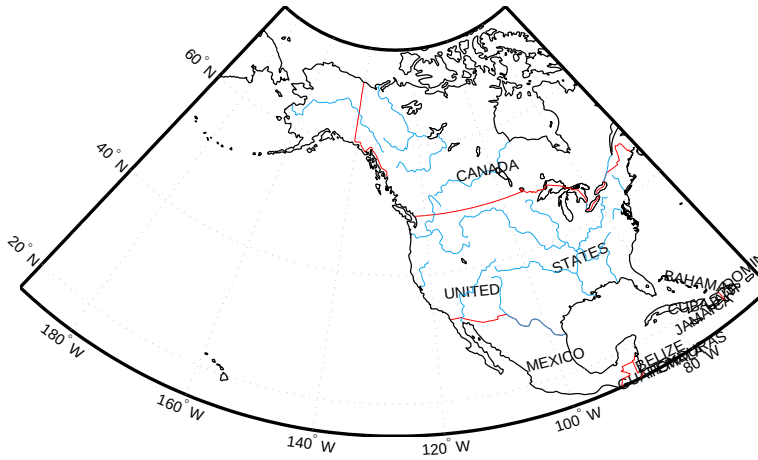
`rotatetext(objects)` rotates the selected objects. `objects` can be a name string recognized by `handlem` or a vector of handles to displayed text objects.

`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.

```
worldmap('usa','lineonly')
h = displaym(worldlo('POtext'));
trimcart(h)
rotatetext(h)
```



Remarks

You can rotate meridian and parallel labels automatically by setting the map axes `LabelRotation` property to `'on'`.

See Also

`vfwdtran`	Transform vector azimuths to a projection space angle
`vinvtran`	Transform vector azimuths from a projection space angle

roundn

Purpose

Round numbers at specified powers of 10

Syntax

`outnum = roundn(innum)` rounds the elements of `innum` to the nearest one-hundredth.

`outnum = roundn(innum,n)` specifies the power of 10 to which the elements of `innum` are rounded. For example, if `n = 2`, round to the nearest hundred (10^2).

Examples

Using generated numbers, round them to significant tenths, ones, and tens figures (note that your original numbers could differ):

```
fullfig = 1000*magic(2)/7
fullfig =
   142.8571  428.5714
   571.4286  285.7143

tenths = roundn(fullfig,-1)
tenths =
   142.9000  428.6000
   571.4000  285.7000

units = roundn(fullfig,0)
units =
   143   429
   571   286

tens = roundn(fullfig,1)
tens =
   140   430
   570   290
```

See Also

`epsm` Map precision

Purpose

Compute auxiliary sphere radii

Syntax

`r = rsphere('biaxial',ellipsoid)` calculates the radius of a biaxial auxiliary sphere for the ellipsoid specified by the two-element ellipsoid vector `ellipsoid`. The output, `r`, is the radius of this sphere in units consistent with the semimajor axis, that is, the first element of `ellipsoid`. The biaxial radius is calculated by averaging the semimajor and semiminor axes of the ellipsoid, giving each equal weight.

`r = rsphere('biaxial',ellipsoid,`*`method`*`)` specifies the averaging method. If the string *`method`* is `'mean'` (the default), an arithmetic mean is used. If *`method`* is `'norm'`, a geometric mean is used.

`r = rsphere('triaxial',ellipsoid)` results in a triaxial radius, which is calculated by averaging the ellipsoidal axes while giving double weight to the semimajor axis to reflect its role in two of the ellipsoid's three dimensions.

`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,`*`method`*`,`*`units`*`)` returns a radius that is the result of averaging the meridional and transverse radii of curvature at the specified latitude, `lat`. The units of the input `lat` can be specified by the valid angle units string *`units`*. The default units are `'degrees'`, the default averaging method is `'mean'`, and the default latitude is 45°.

`r = rsphere('euler',lat1,lon1,lat2,lon2,ellipsoid)` and `r = rsphere('euler',lat1,lon1,lat2,lon2,ellipsoid,`*`units`*`)` calculate a radius using Euler's Theorem. This calculation requires the specification of an arc that is defined by its endpoints, (`lat1`,`lon1`) and (`lat2`,`lon2`).

Description

The `rsphere` function calculates the radii of auxiliary spheres for the ellipsoid. An auxiliary sphere is a sphere that shares certain desired characteristics with the ellipsoid.

Examples

Different criteria result in different spheres:

```
r = rsphere('biaxial',almanac('earth','ellipsoid','km'))
r =
   6.3674e+03

r = rsphere('triaxial',almanac('earth','ellipsoid','km'))
r =
   6.3710e+03

r = rsphere('curve',almanac('earth','ellipsoid','km'))
r =
   6.3781e+03
```

See Also

`rcurve` Radii of curvature for the ellipsoid

Purpose

Read predicted global 2-minute (4 km) topography from satellite bathymetry

Syntax

`[latgrat,longrat,z] = satbath` reads the global topography file for the entire world, returning every 50th point. The result is returned as a general data grid.

`[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.

Remarks

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 is available over the Internet via anonymous FTP from

```
<ftp://topex.ucsd.edu/pub/global_topo_2min/>
```

Download the latest version of file `topo_x.2.img`, where `x` is the version number, and rename it `topo_6.w.img` for compatibility with the `satbath` function.

Some documentation is also available over the World Wide Web from

```
<http://topex.ucsd.edu/marine_topo/mar_topo.html>
```

and

```
<http://www.ngdc.noaa.gov/mgg/announcements/announce_predict.html>
```

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
```

See Also

`tbase`	TerrainBase Global 5-Min digital terrain data extraction
`gtopo30`	30-Arc-Sec global digital elevation data extraction
`egm96geoid`	15-minute gridded geoid heights from the EGM96 geoid model of the Earth

Purpose

Add graphic scale

Syntax

`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.

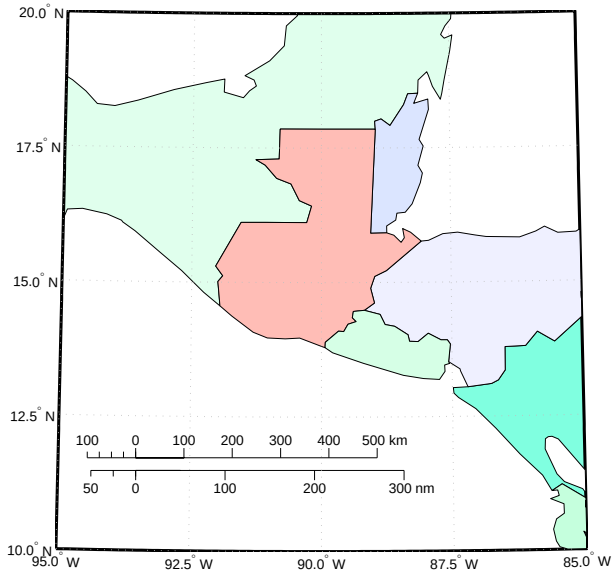
Background

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

Create a map, add a graphic scale with the default settings, and shift it up a bit. Add a second scale showing nautical miles, and change the tick marks and direction.

```
worldmap('guatemala','patchonly')

scaleruler
setm(handlem('scaleruler1'),'YLoc',.205)

scaleruler('units','nm')
setm(handlem('scaleruler2'),...
     'MajorTick',0:100:300,'MinorTick',0:25:50,...
     'TickDir','down','MajorTickLength',km2nm(20),...
     'MinorTickLength',km2nm(12.5))
```



Remarks

You control graphic scale objects using `setm` and `getm`. The function `setm(h)`, where `h` is the handle to a graphic scale object, displays a list of graphic scale properties. 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`.

Modifying the properties of the graphic scale results in the replacement of the original object. For this reason, handles to the graphic scale object will change. Use `handlem('scaleruler')` to get a list of the current handles to all graphic scale objects. Use `handlem('scalerulerN')`, where `N` is an integer, to get the handle to a particular graphic scale. Use `namem` to see the names of existing graphic scale objects. The name of a graphic scale object is also stored in the read-only `'Children'` property, which is accessed using `getm`.

Use `scaleruler off`, `clmo scaleruler`, or `clmo scalerulerN` to remove the scale rulers. The handles to the components of the graphic scale are hidden, so using `delete` removes only the baseline. In that event, you can remove the remaining elements of the graphic scale with the function

`delete(findall(gca,'Tag','scalerulerN'))`, where `N` is the corresponding integer.

You can reposition the graphic scale object by dragging the baseline with the mouse. You can also change the position by modifying the `'XPos'` and `'YPos'` properties using `setm`.

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 font angle string recognized by MATLAB.

`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 `string`

Label text for the graphic scale — Contains a string 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 strings`

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 strings. There must be as many strings 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 `strings`

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 string 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 the MATLAB *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` `almanac body or scalar`

Planetary radius — The radius property controls the scaling between angular and surface distances. If `radius` is a string, then it is evaluated as an `almanac` body to determine the spherical radius. If numerical, it is the radius of the desired sphere in the same units as the `Units` property. The default is `'earth'`.

`Units` `(valid distance unit strings)`

Surface distance units — Defines the distance units displayed in the graphic scale. `Units` can be any distance unit string recognized by `distdim`. The distance string is also used in the last graphic scale text label.

Other Properties

`Children` `(read-only)`

Name string of graphic scale elements — Contains the tag string 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.

See Also

`distance`	Calculate distances between points on an ellipsoid
`surfdist`	Interactive tool for distance, azimuth, and reckoning calculations
`axesscale`	Resize axes for equivalent scale
`paperscale`	Figure paper size for a given map scale
`distortcalc`	Calculate distortion parameters for a map projection
`mdistort`	Display contours of constant distortion on a map

scatterm

Purpose

Construct a thematic map with proportional symbols

Syntax

`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 string.

`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.

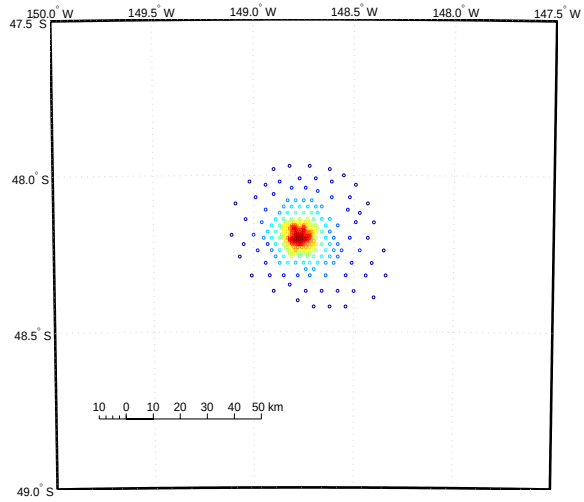
`scatterm`, without any inputs, activates a GUI to project point data onto the current map axes.

`h = scatterm(...)` returns handles of patches created.

Examples

Plot the `seamount` data provided with MATLAB as symbols with the color proportional to the height.

```
load seamount
worldmap([-49 -47.5],[-150 -147.5],'none')
scatterm(y,x,5,z)
scaleruler
```



See Also

`stem3m` Project stem plot on map axes

scircle1

Purpose

Compute coordinates of a small circle path from center, radius, and arc limits

Syntax

`[latc,lonc] = scircle1(lat,lon,rng)` returns the coordinates of points along small circles centered at the points provided in `lat` and `lon` with radii given in `rng`. These radii must in this case be given in the same angle units as the center points (`'degrees'`). The coordinates for multiple small circles are stored in separate columns of `latc` and `lonc`.

`[latc,lonc] = scircle1(lat,lon,rng,az)` specifies the arc section of the small circle for which points are returned. The input `az` is a one- or two-column vector. When `az` has a single column, points are returned for the arc segment from 0° azimuth clockwise to the positive entries in `az` (counterclockwise for negative entries). When `az` has two columns, the returned points correspond to arc segments from the first-column entry clockwise to the second-column entry. When `az` is empty or not provided, points for the entire small circle are returned.

`[latc,lonc] = scircle1(lat,lon,rng,az,`*`units`*`)` specifies the units for the inputs and outputs, where *`units`* is any valid angle units string. The default value is `'degrees'`.

`[latc,lonc] = scircle1(lat,lon,rng,az,ellipsoid,`*`units`*`)` specifies the elliptical definition of the Earth to be used with the two-element `ellipsoid` vector. The default ellipsoid model is the sphere, which is sufficient for most applications. When a ellipsoid is input, the range inputs in `rng` must be in the units of the ellipsoid semimajor axis, rather than in the angle units specified by *`units`*.

`[latc,lonc] = scircle1(lat,lon,rng,az,ellipsoid,`*`units`*`,npts)` specifies the number of output points, `npts`, returned per small circle. The default value of `npts` is 100.

`[latc,lonc] = scircle1(`*`track`*`,lat,lon,rng...)` specifies the logic with which ranges are calculated. If the string track is `'gc'` (the default), great circle distance is used. It track is `'rh'`, rhumb line distance is used.

`pts = scircle1(lat,lon,rng)` returns the points in a two-column output `pts`.

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 `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 90° or its angular equivalent, so all meridians on the globe are small circles as well.

Small circle notation consists of a center point and a radius in units of angular arc length.

Examples

Create and plot a small circle centered at (0°,0°) with a radius of 10°:

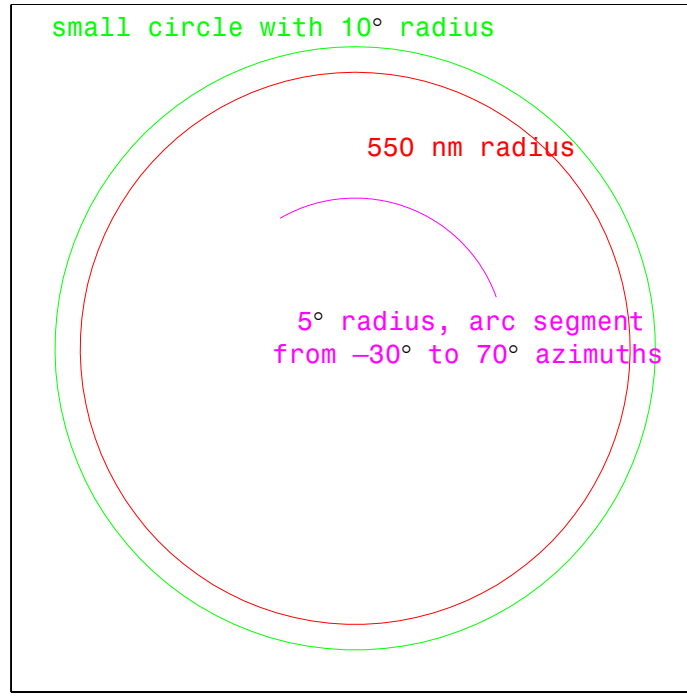
```
axesm('mercator','MapLatLimit',[30 -30],'MapLonLimit',[-30 30]);
[latc,longc] = scircle1(0,0,10);
plotm(latc,longc,'g')
```

If the desired radius is known in some nonangular distance unit, use the radius returned by the `almanac` function as the ellipsoid to set the range units (use an empty azimuth entry to indicate a full circle):

```
earthradius = almanac('earth','radius','nm');
[latc,longc] = scircle1(0,0,550,[],earthradius);
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')
```



See Also

`scircle2`	Small circle from center and perimeter point
`track`	Connect waypoints with track segments
`scircleg`	Interactive small circles
`trackg`	Interactive tracks
`track1` `track2`	Great circles and rhumb lines

Purpose

Compute coordinates of a small circle path from center and perimeter point

Syntax

`[latc,lonc] = scircle2(lat1,lon1,lat2,lon2)` returns the coordinates of points along small circles centered at the points provided in `lat1` and `lon1`, which pass through the points provided in `lat2` and `lon2`. The coordinates of multiple small circles are stored in separate columns of `latc` and `lonc`.

`[latc,lonc] = scircle2(lat1,lon1,lat2,lon2,`*`units`*`)` specifies the units for the inputs and outputs, where *`units`* is any valid angle units string. The default value is `'degrees'`.

`[latc,lonc] = scircle2(lat1,lon1,lat2,lon2,ellipsoid)` specifies the elliptical definition of the Earth to be used with the two-element `ellipsoid` vector. The default ellipsoid model is the sphere, which is sufficient for most applications.

`[latc,lonc] = scircle2(lat1,lon1,lat2,lon2,ellipsoid,`*`units`*`,npts)` specifies the number of output points, `npts`, returned per small circle. The default value of `npts` is 100.

`[latc,lonc] = scircle2(`*`track`*`,lat1,lon1,lat2,lon2...)` specifies the logic with which ranges are calculated. If the string track is `'gc'` (the default), great circle distance is used. If track is `'rh'`, rhumb line distance is used.

`pts = scircle2(lat1,lon1,lat2,lon2)` returns the points in a two-column output `pts`.

Background

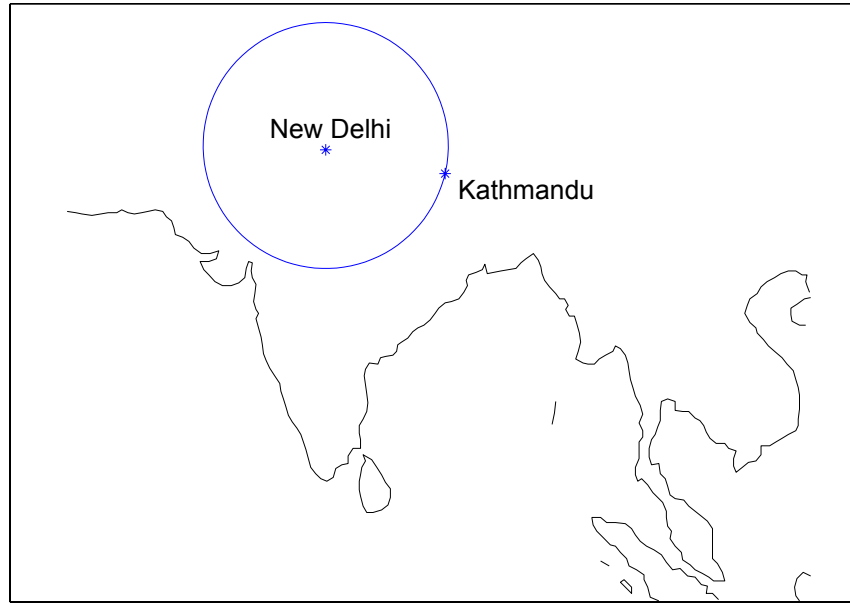
A small circle is the locus of all points an equal surface distance from a given center. For true small circles, this distance is always calculated in a great circle sense; however, the `scircle2` function allows a locus to be calculated using distances in a rhumb line sense as well. An example of a small circle is *all points exactly 100 miles from the Washington Monument*.

Examples

Plot the locus of all points the same distance from New Delhi as Kathmandu:

```
axesm('mercator','MapLatlimit',[0 40],'MapLonLimit',[60 110]);
load coast
plotm(lat,long,'k');    % For reference
lat1 = 29; lon1 = 77.5;    % New Delhi
lat2 = 27.6; lon2 = 85.5;  % Kathmandu
plotm([lat1 lat2],[lon1 lon2],'b*')   % Plot the cities
```

```
[latc,lonc] = scircle2(lat1,lon1,lat2,lon2);
plotm(latc,lonc,'b')
```



See Also

`scircle1`	Small circle from center and radius
`track`	Connect waypoints with track segments
`track1` `track2`	Great circles and rhumb lines

Purpose

Display small circle defined via mouse clicks

Syntax

`h = scircleg(ncirc)` brings forward the current map axes and waits for the user to make (2 x `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,`*`linestyle`*`)` specifies the line style for the displayed small circles, where *`linestyle`* is any line style string recognized by the standard MATLAB function `line`.

`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 the string track is `'gc'` (the default), great circle distance is used. If track is `'rh'`, rhumb line distance is used.

Description

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.

See Also

`scircle1`	Small circle from center, azimuth, and radius
`scircle2`	Small circle from center and perimeter point

Purpose

Provide intersection coordinates for pairs of small circles

Syntax

`[newlat,newlon] = scxsc(lat1,lon1,range1,lat2,lon2,range2)` returns in `newlat` and `newlon` the locations of the points of intersection of two small circles in *small circle notation*. For example, the first small circle in a pair would be centered on the point (`lat1`,`lon1`) with a radius of `range1` (in angle units). The inputs must be column vectors. 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 returned twice.

`[newlat,newlon]=scxsc(lat1,lon1,range1,lat2,lon2,range2,`*`units`*`)` specifies the angle units used for all inputs, where *`units`* is any valid angle units string. The default units are `'degrees'`.

Description

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.

Small circle notation consists of a center point and a radius in units of angular arc length.

Examples

Given a small circle centered at (10°S,170°W) with a radius of 20° (~1200 nautical miles), where does it intersect with a small circle centered at (3°N, 179°E), with a radius of 15° (~900 nautical miles)?

```
[newlat,newlong] = scxsc(-10,-170,20,3,179,15)
newlat =
   -8.8368    9.8526
newlong =
   169.7578 -167.5637
```

Note that in this example, the two small circles cross the date line.

Remarks

Great circles are a subset of small circles — a great circle is just a small circle with a radius of 90°. 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 90° (or its equivalent in radians or dms units).

See Also

`gc2sc`	Convert great circle to small circle notation

`gcxgc`
`gcxsc`
`rhxrh`
`crossfix`
`polyxpoly`
	Other intersection functions

sdtsdemread

Purpose

Read data from an SDTS raster/DEM data set

Syntax

`[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 (see `makerefmat`). `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.

Example

```
[Z, R] = sdtsdemread('9129CATD.ddf');
mapshow(Z,R,'DisplayType','contour')
```

See Also

`arcgridread`, `makerefmat`, `mapshow`, `sdtsinfo`

Purpose

Information about an SDTS data set

Syntax

`info = sdtsinfo(filename)` returns a structure whose fields contain information about the contents of a SDTS data set.

`filename` is a string that specifies the name of the SDTS catalog directory file, such as `7783CATD.DDF`. The filename 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`	String containing the name of the catalog directory file of the SDTS transfer set
`Title`	String containing the name of the data set
`ProfileID`	String containing the Profile Identifier, e.g., `'SRPE: SDTS RASTER PROFILE and EXTENSIONS'`
`ProfileVersion`	String containing the Profile Version Identifier, e.g., `'VER 1.1 1998 01'`
`MapDate`	String specifying the date associated with the cartographic information contained in the data set
`DataCreationDate`	String specifying the creation date of the data set
`HorizontalDatum`	String representing the horizontal datum to which the data is referenced
`MapRefSystem`	String describing the projection and reference system used: `'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`	String specifying the units used for the horizontal coordinate values
`VerticalUnits`	String specifying the units used for the vertical coordinate values
`MinElevation`	Scalar value of the minimum elevation value for the data set
`MaxElevation`	Scalar value of the maximum elevation value for the data set

Example

```
info = sdtsinfo('9129CATD.DDF');
```

See Also

`sdtsdemread`, `makerefmat`

Purpose

Convert time units from seconds to hms or hm

Syntax

`timeout = sec2hms(timein)` converts times input in seconds to the equivalent measure in the hours-minutes-seconds (*hms*) format.

`timeout = sec2hm(timein)` converts times input in seconds to the equivalent measure in the hours-minutes (*hm*) format. This is the hms format, properly rounded to just hours and minutes.

Example

```
sec2hms(3661)
ans =
        101.01

sec2hm(3661)
ans =
        101.00
```

See Also

`hms2mat`	Convert from hms to separated matrix components
`mat2hms`	Convert from separated matrices input to hms
`sec2hm` `hr2sec`	Other direct time conversion functions
`timedim`	Convert times between different units

sec2hr

Purpose

Convert time from seconds to hours

Syntax

`timeout = sec2hr(timein)` converts times input in seconds to the equivalent measure in hours.

Example

```
sec2hr(1000)
ans =
    0.2778
```

See Also

`hr2sec` `hms2sec`	Other direct time conversion functions
`timedim`	Convert times between different units

Purpose

Convert data grid rows and columns to latitude-longitude

Syntax

`[lat,lon] = setltln(map,refvec,row,col)` returns the latitude and longitudes associated with the input row and column coordinates of the input `map` with a referencing vector of `refvec`.

`mat = setltln(map,refvec,row,col)` returns the coordinates in a single two-column matrix of the form `[latitude longitude]`.

`[lat,lon,badindx] = setltln(map,refvec,row,col)` returns the indices of the elements of the `row` and `col` vectors that lie outside the input map. The outputs `lat` and `lon` always ignore these points; the third output accounts for them.

Examples

Find the coordinates of row 45, column 65 of `topo`:

```
load topo
[lat,lon,badindx] = setltln(topo,topolegend,45,65)
lat =
  -45.5000
lon =
   64.5000
badindx =
     []      % Empty because the point is valid
```

See Also

`ltln2val`	Latitude and longitude to matrix entry value
`pix2latlon`	Convert pixel coordinates to latitude-longitude coordinates
`setpostn`	Latitude and longitude to row and column

setm

Purpose

Modify the properties of a displayed map

Syntax

`setm(axishndl,`*`PropertyName`*`,PropertyValue,...)` sets the properties of the map axes specified by its handle to the given values. The map properties must be recognized by `axesm`.

`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.

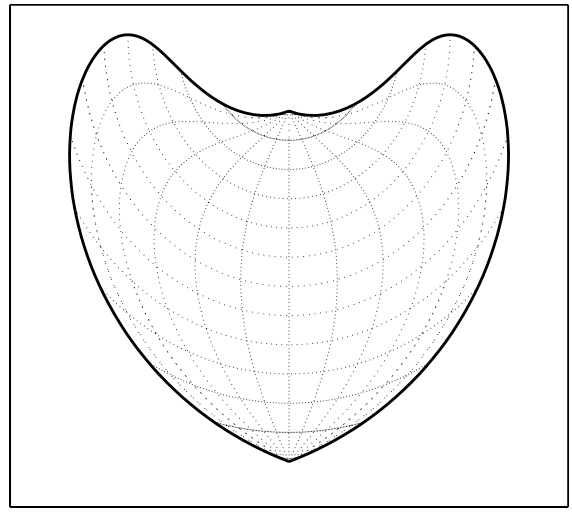
`setm(surfhndl,'MeshGrat',npts,alt)` alters the mesh graticule of projected surface objects displayed using the `meshm` function. In this case, the two-element vector `npts` specifies the graticule size in the manner described under `meshm`. The altitude can be a scalar or a matrix with a size corresponding to `npts`.

Examples

Display a map axes and alter it:

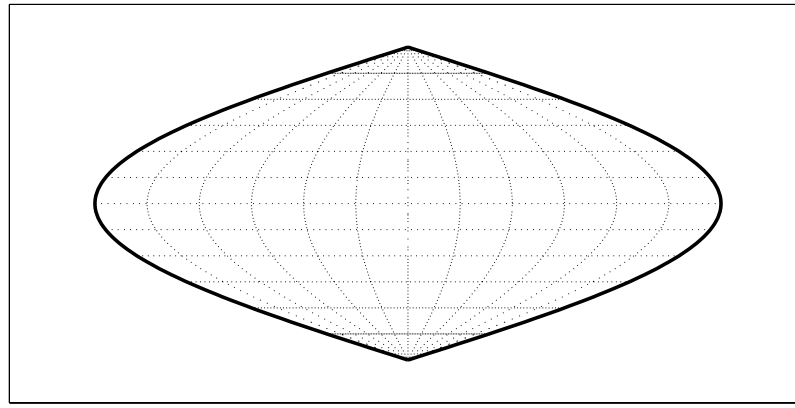
```
axesm('bonne','Frame','on','Grid','on')
```

The standard Bonne projection has a standard parallel at 30°N.



Setting this standard parallel to 0° results in a Sinusoidal projection:

```
setm(gca,'MapParallels',0)
```



See Also

`axesm`	Create and set map axes properties
`getm`	Get map object properties

setpostn

Purpose

Convert latitude-longitude to data grid rows and columns

Syntax

`[row,col] = setpostn(map,refvec,lat,long)` returns the row and column indices of the input regular data grid with a referencing vector of `refvec` for the points specified by the vectors `lat` and `long`. All angles are in degrees.

`indx = setpostn(map,refvec,lat,long)` returns the single-value indices of `map(:)`.

`[row,col,badindx] = setpostn(map,refvec,lat,long)` also returns the indices of `lat` and `long` corresponding to points outside `map`. These points are always ignored in `row` and `col`.

Examples

What are the matrix coordinates in `topo` of Denver, Colorado, at (39.7°N,105°W)?

```
load topo
[row,col] = setpostn(topo,topolegend,39.7,105)
row =
   130
col =
   105
```

See Also

`latlon2pix`	Convert latitude-longitude coordinates to pixel coordinates
`ltln2val`	Latitude and longitude to matrix entry value
`setltln`	Row and column to latitude and longitude

Purpose

Construct cdata and colormap for colored shaded relief

Syntax

`[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 (90° azimuth) at an elevation of 45°.

`[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`.

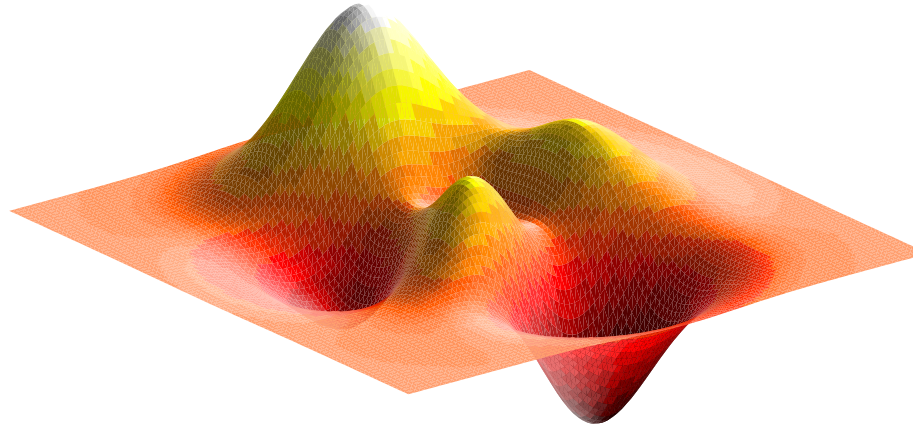
Remarks

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 color map. 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.

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
```



See Also

`caxis`	Pseudocolor axis scaling
`colormap`	Color lookup table
`light`	Create a light object (see the online MATLAB Function Reference documentation)
`meshlsrm`	Project 3-D lighted shaded relief of a regular data grid
`surf`	3-D shaded surface plot (see the online MATLAB Function Reference documentation)
`surflsrm`	Project 3-D lighted shaded relief of a geolocated data grid

Purpose

Information about a shapefile

Syntax

`info = shapeinfo(filename)` returns a structure, `info`, whose fields contain information about the contents of a shapefile.

The shapefile format was defined by the Environmental Systems Research Institute (ESRI) to store nontopological vector geometry and attribute information for spatial features. A shapefile consists of a main file, an index file, and an xBASE table. All three files have the same base name and are distinguished by the extensions `.SHP`, `.SHX`, and `.DBF`, respectively (e.g., given the base name `'roads'` the shapefile filenames would be `'roads.SHP'`, `'roads.SHX'`, and `'roads.DBF'`).

`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 directory and have valid file extensions. 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.

Field Descriptions

The `info` structure contains the following fields:

`Filename`	Char array containing the names of the files that were read
`ShapeType`	String 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
`NumFeatures`	The number of spatial features in the shapefile

The `Attributes` structure contains these fields:

- `Name` — String containing the attribute name as given in the xBASE table
- `Type` — String specifying the MATLAB class of the attribute data returned by `shaperead`. The following attribute (xBASE) types are supported: Numeric, Floating, Character, and Date

See Also shaperead

Purpose

Read vector feature coordinates and attributes from a shapefile

Syntax

`s = shaperead(filename)` returns an N-by-1 version 2 geographic data structure (geostruct2) array, `S`, containing an element for each nonnull spatial feature in the shapefile. `S` combines feature coordinates/geometry and attribute values.

`[s, a] = shaperead(filename)` returns an N-by-1 geostruct2 array, `s`, and a parallel N-by-1 attribute structure array, `a`. Each array contains an element for each nonnull spatial feature in the shapefile.

`[s, a] = shaperead(filename, param1, val1, param2, val2,...)` returns a subset of the shapefile contents in `s` or `[s,a]`, as determined by the parameters `'RecordNumbers'`, `'BoundingBox'`, `'Selector'`, or `'Attributes'`. In addition, the parameter `'UseGeoCoords'` can be used in cases where you know that the *x*- and *y*-coordinates in the shapefile actually represent longitude and latitude.

The shapefile format was defined by the Environmental Systems Research Institute (ESRI) to store nontopological vector geometry and attribute information for spatial features. A shapefile consists of a main file, an index file, and an xBASE table. All three files have the same base name and are distinguished by the extensions `.SHP`, `.SHX`, and `.DBF`, respectively (e.g., given the base name `'concord_roads'` the shapefile filenames would be `'concord_roads.SHP'`, `'concord_roads.SHX'`, and `'concord_roads.DBF'`).

`filename` can be the base name or the full name of any one of the component files. `shaperead` reads all three files as long as they exist in the same directory and have valid file extensions. If the main file (with extension `.SHP`) is missing, `shaperead` returns an error. If either of the other files is missing, `shaperead` returns a warning.

Supported Shape Types

`shaperead` supports the ordinary 2-D shape types: `'Point'`, `'Multipoint'`, `'PolyLine'`, and `'Polygon'`. (`'Null Shape'` features can also be present in a Point, Multipoint, PolyLine, or Polygon shapefile, but are ignored.) `shaperead` does *not* support any 3-D or "measured" shape types: `'PointZ'`, `'PointM'`, `'MultipointZ'`, `'MultipointM'`, `'PolyLineZ'`, `'PolyLineM'`, `'PolygonZ'`, `'PolylineM'`, or `'Multipatch'`.

Output Structure

The fields in the output structure arrays `s` and `a` depend on (1) the type of shape contained in the file and (2) the names and types of the attributes included in the file:

Field Name	Field Contents	Comment
`Geometry`	Shape type string	
`BoundingBox`	`[minX minY; maxX maxY]`	Omitted for shape type `'Point'`
`X` or `Lon`	Coordinate vector	NaN separators used in multipart PolyLine and Polygon shapes
`Y` or `Lat`	Coordinate vector	NaN separators used in multipart PolyLine and Polygon shapes
`attr1`	Value of first attribute	Included in output `s` if output `a` is omitted
`attr2`	Value of second attribute	Included in output `s` if output `a` is omitted
`...`	...	...

The names of the attribute fields (listed above as `Attr1, Attr2, ...`) are determined at run-time from the xBASE table (with extension `.DBF`) and/or optional, user-specified parameters. There can be many attribute fields, or none at all.

Field Descriptions

- **Geometry** — String with one of the following values: `'Point'`, `'MultiPoint'`, `'Line'`, or `'Polygon'`. (Note that these match the standard shapefile types, except that for shape type `'Polyline'` the value of the *`Geometry`* field is simply `'Line'`.)
- **BoundingBox** — Contains a 2-by-2 numerical array specifying the minimum and maximum feature coordinate values in each dimension

(`min([x, y]); max([x, y]`, where x and y are N-by-1 and contain the combined coordinates of all parts of the feature).

- **X and Y / Lon and Lat (Coordinate vector)** — 1-by-N arrays of class double. For `'Point'` shapes, they are 1-by-1. In the case of multipart `'Polyline'` and `'Polygon'` shapes, `NaN`s are added to separate the lines or polygon rings. In addition, one terminating `NaN` is added to support horizontal concatenation of the coordinate data from multiple shapes.
- **Attribute fields** — Attribute names, types, and values are defined within a given shapefile. The following four types are supported: Numeric, Floating, Character, and Date. `shaperead` skips over other attribute types. The field names in the output shape structure are taken directly from the shapefile if they contain no spaces or other illegal characters and there is no duplication of field names (e.g., an attribute named `'BoundingBox'`, `'PointData'`, etc., or two attributes with the same names).

  Otherwise, the following "name mangling" is applied: Illegal characters are replaced by `'_'`. If the first character in the attribute name is illegal, a leading `'Z'` is added. Numerals are appended if needed to avoid duplicate names. The attribute values for a feature are taken from the shapefile and stored as doubles or character arrays:

Attribute Type in Shapefile	MATLAB Storage
Numeric	double (scalar)
Float	double (scalar)
Character	char array
Date	char array

Parameter-Value Options

By default, `shaperead` returns an entry for every nonnull feature and creates a field for every attribute. Use the first three parameters below (`RecordNumbers`, `BoundingBox`, and `Selector`) to be selective about which features to read. Use

the fourth parameter (`Attributes`) to control the attributes to keep. Use the fifth (`GeodeticCoords`) to control the output field names.

Name	Description of Value	Purpose
`RecordNumbers`	Integer-valued vector, class double	Screen out features whose record numbers are not listed.
`BoundingBox`	2-by-(2, 3, or 4) array, class double	Screen out features whose bounding boxes fail to intersect the selected box.
`Selector`	Cell array containing a function handle and one or more attribute names. Function must return a logical scalar.	Screen out features for which the function, when applied to the corresponding attribute values, returns false.
`Attributes`	Cell array of attribute names	Omit attributes that are not listed. Use `{}` to omit all attributes. Also sets the order of attributes in the structure array.
`UseGeoCoords`	Scalar logical	If true, replace X and Y field names with `'Lon'` and `'Lat'`, respectively. Defaults to false.

Examples

```
% Read the entire concord_roads.shp shapefile, including the
% attributes,
% in concord_roads.dbf.
S = shaperead('concord_roads.shp');

% Restrict output based on a bounding box and read only two
% of the feature attributes.
bbox = [2.08 9.11; 2.09 9.12] * 1e5;
S = shaperead('concord_roads','BoundingBox',bbox,...
              'Attributes',{'STREETNAME','CLASS'});
```

See Also

`shapeinfo`, `updategeostruct`

Purpose

Toggle display of Cartesian MATLAB axes

Syntax

`showaxes` toggles the visibility of the axes between the `'on'` and `'off'` conditions.

`showaxes('on')` sets the color of the axes (the `XColor` and `YColor` properties) to black.

`showaxes('off')` sets the color of the axes (the `XColor` and `YColor` properties) to the background color, effectively making them invisible. This is the default condition for map axes.

`showaxes('hide')` sets the `Visible` property of the axes to `'on'`.

`showaxes('show')` sets the `Visible` property of the axes to `'off'`.

`showaxes('reset')` sets the axes properties to the default map axes settings.

`showaxes(`*`color`*`)` sets the color of the axes (the `XColor` and `YColor` properties) to the color specified by any valid color string.

`showaxes(color)` sets the color of the axes (the `XColor` and `YColor` properties) to the color specified by the input RGB triple.

See Also

`axesm` Map axes creation

showm

Purpose

Show specified graphics objects

Syntax

`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`* string, which can be any string recognized by the `handlem` function.

See Also

`clma`	Clear current map
`clmo`	Clear specified graphics objects
`handlem`	Get handles of displayed map objects
`hidem`	Hide specified graphics objects
`namem`	Determine names of valid graphics objects
`tagm`	Assign a name to graphics object tag property

Purpose

Determine row and column dimensions needed for a regular data grid

Syntax

`[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 referencing vector for 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!

See Also

`findm`	Coordinates of nonzero map entries
`limitm`	Matrix map limits

`nanm`
`onem`
`spzerom`
`zerom`
	Create special maps

sm2deg, sm2km, sm2nm, sm2rad

Purpose

Convert distance from statute miles to other units

Syntax

`distout = sm2deg(distin)` converts the input distance given in statute miles to degrees. `distout = sm2km(distin)`, `distout = sm2rad(distin)`, and `distout = sm2nm(distin)` perform analogously, converting to kilometers, radians, and nautical miles, respectively.

`distout = sm2deg(distin,radius)` and `distout = sm2rad(distin,radius)` specify the radius of the sphere to use, since a degree (or radian) of arc length covers less distance, for example, on Mars than it does on the Earth. You can enter the radius as a number in statute miles, as a call to the `almanac` function (e.g., `almanac('mars','radius','sm')`), or you can pass in a string planet name (e.g., `'mars'`), and the function will make the appropriate call to the `almanac` function. The radius of the Earth is the default.

Examples

In track, is the quarter mile exactly the same as the 400-meter?

```
distout = sm2km(0.25)
distout =
    0.4023
```

No, it's 2.3 meters longer.

See Also

`distdim`	Convert distances between different units
`km2sm` `nm2deg`	Other direct distance conversion functions

Purpose

Remove discontinuities in longitudes

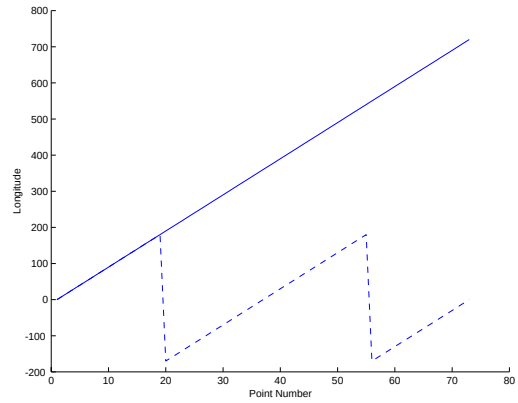
Syntax

`ang = smoothlong(angin)` removes discontinuities in longitude data. The resulting angles can cover more than one revolution.

`ang = smoothlong(angin,`*`units`*`)` uses the units defined by the input string *`units`*. If omitted, default units of `'degrees'` are assumed.

Examples

```
long = npi2pi(0:10:720);
long2 = smoothlong(long);
figure;hold on
plot(long,'--'); plot(long2)
xlabel 'Point Number'; ylabel Longitude
```



Remarks

This function can remove large jumps in longitude that might otherwise result in spurious data when you are interpolating with `interpm`.

See Also

`npi2pi`	Truncate angles into the -180 deg to 180 deg range
`zero22pi`	Truncate angles into the 0 deg to 360 deg range
`interpm`	Interpolate vector data to a specified data separation

Purpose

Read an ASCII file of space-delimited data columns

Syntax

`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 string *`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.

Remarks

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` Convert pen-down delimited data to `NaN`-delimited data

spzerom

Purpose

Create a sparse data grid of zeros

Syntax

`map = spzerom(latlim,lonlim,scale)` returns a sparse regular data grid consisting entirely of zeros. 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 number of rows and columns per angle unit is set by the scalar `scale`.

`[map,refvec] = spzerom(latlim,lonlim,scale)` returns the three-element referencing vector for the returned `map`.

Examples

```
[map,refvec] = spzerom([46,51],[-79,-75],1)
map =
   All zero sparse: 5-by-4
refvec =
     1    51   -79
```

See Also

`limitm`	Matrix map limits
`nanm`	Create a data grid of `NaN`s
`onem`	Create a data grid of ones
`sizem`	Rows and columns needed for `map`
`zerom`	Create a full data grid of zeros

Purpose

Compute standard distance of geographic data

Syntax

`dist = stdist(lat,lon)` returns a row vector of the latitude and longitude geographic standard distance for the data points specified by the columns of `lat` and `lon`.

`dist = stdist(lat,lon,`*`units`*`)` indicates the angular units of the data. When the standard angle string *`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 elliptical definition of the Earth to be used with the two-element `ellipsoid` vector. The default ellipsoid model is a spherical Earth, which is sufficient for most applications. Output measurements are in terms of the distance units of the `ellipsoid` vector.

`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`.

Description

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

Generate latitude and longitude matrices using the `worldlo` data set (compare with the example for `stdm`):

```
load worldlo
lat=[PPpoint(2:5).lat]
lat =
    5.3386   24.5808    5.4619   36.8862

lon=[PPpoint(2:5).long]
lon =
   -3.7602   54.9619   -0.3450   35.5141
```

```
dist= stdist(lat,lon)
dist =
   26.5282
```

See Also

`meanm`	Mean of geographic data
`stdm`	Standard deviation of geographic data

Purpose

Compute standard deviation for geographic data

Syntax

`[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 elliptical definition of the Earth to be used with the two-element `ellipsoid` vector. The default ellipsoid model is a spherical Earth, which is sufficient for most applications. 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 the standard angle string *`units`* is omitted, `'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” in the “Mapping Applications” chapter of the Mapping Toolbox User’s Guide documentation.

Examples

Create latitude and longitude matrices using the `worldlo` data set (compare with the example for `stdist`):

```
load worldlo
lat=[PPpoint(2:5).lat]
lat =
    5.3386   24.5808    5.4619   36.8862

lon=[PPpoint(2:5).long]
lon =
   -3.7602   54.9619   -0.3450   35.5141

[latstd,lonstd]=stdm(lat,lon)
latstd =
```

```
      8.9962
  lonstd =
    159.1153
```

See Also

`departure`	Departure of longitudes at specific latitudes
`filterm`	Geographic filter for data sets
`hista`	Spatial equal-area histogram
`histr`	Spatial equirectangular histogram
`meanm`	Mean for geographic data
`stdist`	Standard distances for geographic data

Purpose

Project a stem plot map on the current map axes

Syntax

`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,`*`LineType`*`)` allows the style of the stem plot's lines to be specified with any string *`LineType`* recognized by the MATLAB `line` function.

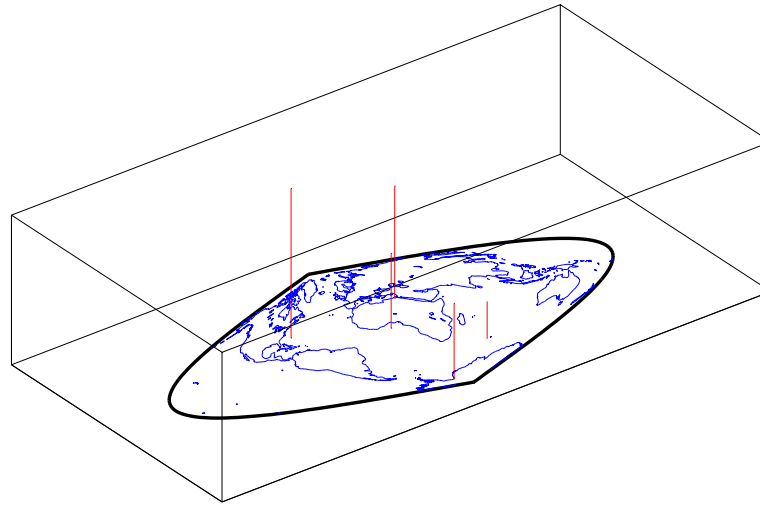
`h = stem3m(lat,lon,z,`*`PropertyName`*`,PropertyValue,...)` allows any property/value pair recognized by the MATLAB line function to be specified for the stems.

Description

A stem plot displays data as lines extending normal to the *xy*-plane, in this case, on a map.

Examples

```
load coast
axesm sinusoid; view(3)
h = framem; set(h,'zdata',zeros(size(lat)))
plotm(lat,long)
ptlat = [0 30 30 -50 -78]';
ptlon = [0 30 -70 65 -35]';
ptz = [1 1.5 2 .5 1]';
stem3m(ptlat,ptlon,ptz,'r-')
```



See Also

`scatterm` Thematic map with proportional symbols

Purpose

Convert formatted dms angle strings to numbers

Syntax

`angles = str2angle(strings)` converts the formatted degrees-minutes-seconds strings to numeric angles in units of degrees. Examples of recognized formats are 123°30'00"S, 123-30-00S, 123d30m00sS, and 1233000S. The seconds field can contain a fractional component in all but the last form. Strings can be a character matrix or a cell array vector of strings.

Example

```
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

strs = strvcat(strs{:})

strs =

23°30'00"N
23-30-00S
123d30m00sE
1233000W

str2angle(strs)

ans =

         23.5
        -23.5
        123.5
```

```
-123.5
```

See Also

`angl2str`	Angle conversion to a string

Purpose

Warp data grid to a projected graticule mesh

Syntax

`h = surfacem(map)` projects the data grid `map` on a graticule grid the size of `map` between the latitude and longitude limits of the current map axes. The handle `h` of the displayed surface can be returned.

`h = surfacem(map,npts)` results in a graticule grid defined by `npts`, which is a two-element vector of the form `[latitude-points longitude-points]`. The default `npts` is `[50 100]` (the graticule has 50 vertices in the latitude direction and 100 vertices in the longitude direction).

`h = surfacem(lat,lon,map)` allows greater graticule control through the inputs `lat` and `lon`. If matrices, they are the graticule vertex coordinates as might be returned by `meshgrat`. If vectors, they are the representative coordinates of the rows and columns, respectively, of such a grid. If they are two-element vectors, they are treated as latitude and longitude limits, and a graticule mesh the size of the default `npts` is calculated internally.

`h = surfacem(lat,lon,map,alt)` sets the *z*-axis altitude of the graticule mesh. `alt` must be the same size as `lat`. If no `alt` is supplied, the mesh is plotted at z = 0, unless `lat` is the same size as `map`, in which case `zdata = map`, and a 3-D topographical map results.

`h = surfacem(lat,lon,map,`*`PropertyName`*`,PropertyValue,...)` allows the input of property name/property value pairs to control the surface object properties. Any property supported by the standard MATLAB function `surface` except `XData`, `YData`, and `ZData` can be altered in this manner.

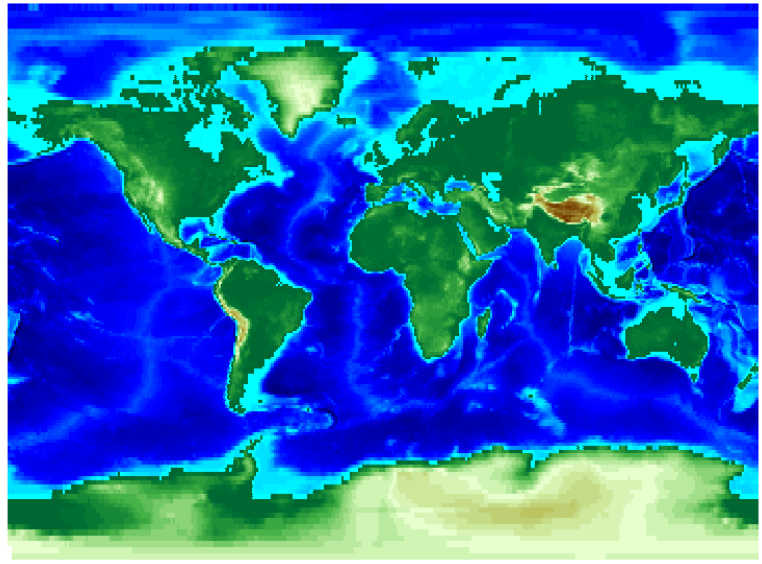
Description

Unlike `meshm` and `surfm`, this function adds surfaces to the current axes, regardless of hold state. This function warps a data grid to a graticule mesh, which itself is projected according to the map axes `MapProjection` property. 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.

Examples

```
load topo
axesm miller
surfacem(topo,[30 30])
```

```
demcmap(topo)
```



See Also

`meshgrat`	Construct map graticule grid
`meshm`	Regular data grid warped to a projected graticule mesh
`pcolorm`	Projected data grid in the z = 0 plane
`surfm`	Matrix map projected on a map axes

Purpose

Project three-dimensional shaded surface with lighting on a map axes

Syntax

`h = surflm(map)` displays the data grid `map` projected to a graticule grid the size of `map` in accordance with the current map axes `MapProjection` property. It is displayed with a default light source. The handle `h` of the displayed surface object can be returned.

`h = surflm(map,s)` specifies the direction of the light source. `s` is a two- or three-element vector that specifies the direction from the surface map to the light source. `s=[sx sy sz]` or `s=[azimuth elevation]`. The default `s` is 45° counterclockwise from the current view direction.

`h = surflm(lat,lon,map)` allows you to specify your graticule. `lat` and `lon` can be vectors with elements corresponding to `map` rows and columns, respectively, or they can be matrices the size of `map`. The resulting graticule is the size of `map`.

`h = surflm(lat,lon,map,s,k)` specifies the reflectance constant. `k` is a four-element vector defining the relative contributions of ambient light, diffuse reflection, specular reflection, and the specular shine coefficient. `k = [ka,kd,ks,shine]` and defaults to `[.55 .6 .4 10]`.

Description

`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 map matrix.

Examples

To see this, type the following. The graticule is the size of `topo` (180 x 360) and is rendered in 3-D, so it might take a while. It is also memory intensive:

```
load topo
axesm miller
surflm(topo)
```

See Also

`surfm` Matrix map projected on map axes

surflsrm

Purpose

Project 3-D lighted shaded relief of a geolocated data grid

Syntax

`surfsrlm(lat,long,map)` displays the geolocated data grid, colored according to elevation and surface slopes. The current axes must have a valid map projection definition.

`surfsrlm(lat,long,map,[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 (90° azimuth) at an elevation of 45°.

`surfsrlm(lat,long,map,[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.

`surfsrlm(lat,long,map,[azim elev],cmap,clim)` uses the provided color axis limits, which are, by default, automatically computed from the data.

`h = surfsrlm(...)` returns the handle to the surface drawn.

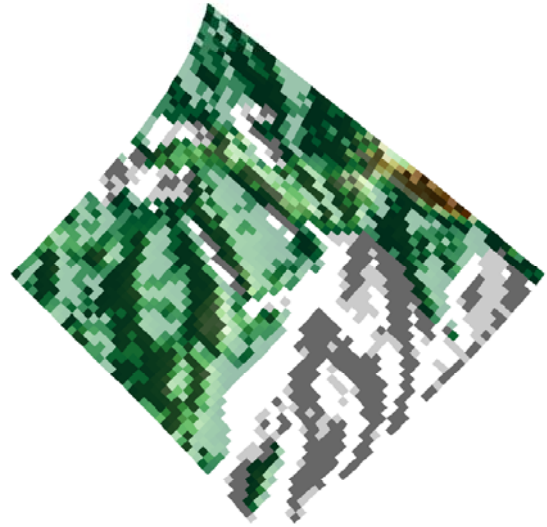
Remarks

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.

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)
```



See Also

`meshlsrm`	Project 3-D lighted shaded relief of a regular data grid
`meshm`	Display a regular data grid warped to a projected graticule
`pcolorm`	Projected data grid in the z = 0 plane
`shaderel`	Construct cdata and colormap for colored shaded relief
`surfacem`	Display a data grid warped to a projected graticule
`surflm`	Display a lighted data grid warped to a projected graticule
`surfm`	Display a data grid warped to a projected graticule

surfm

Purpose

Project data grid on a map axes

Syntax

`h = surfm(map)` projects the data grid `map` on a graticule grid the size of `map` between the latitude and longitude limits of the current map axes. The handle `h` of the displayed surface can be returned.

`h = surfm(map,npts)` results in a graticule grid defined by npts, which is a two element vector of the form `[latitude-points longitude-points]`.

`h = surfm(lat,lon,map)` allows three other methods of defining the graticule grid. If `lat` and `lon` are matrices, they represent the actual graticule vertices as might be returned by `meshgrat`. If vectors, they are the representative coordinates of the rows and columns, respectively, of such a grid. If they are two-element vectors, they are treated as latitude and longitude limits, and a graticule mesh of `size(map)` is calculated.

`h = surfm(lat,lon,map,alt)` sets the *z*-axis altitude of the graticule mesh. `alt` must be the same size as `lat`. If no `alt` is supplied, the mesh is plotted at `z = 0`, unless `lat` is the same size as `map`, in which case `zdata = map`, and a 3-D topological map results. Since the default graticule is the size of `map`, the default condition for `surfm` is to create the topographic map.

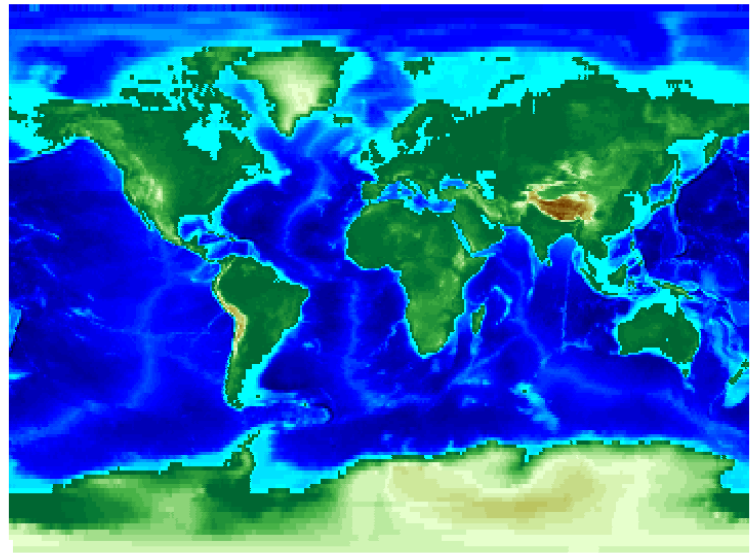
`h = surfm(lat,lon,map,`*`PropertyName`*`,PropertyValue,...)` allows the input of property name/property value pairs to control the surface object properties. Any property supported by the standard MATLAB function `surface` except `XData`, `YData`, and `ZData` can be altered in this manner.

Description

This function warps a data grid to a graticule mesh, which itself 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.

Examples

```
load topo
axesm miller
[meshlat,meshlon] = meshgrat(topo,topolegend,[90 180]);
surfm(meshlat,meshlon,topo)
demcmap(topo)
```



See Also

`meshgrat`	Construct map graticule grid
`meshm`	Regular data grid warped to projected graticule mesh
`pcolorm`	Projected data grid in the z = 0 plane
`surfacem`	Matrix map warped to projected graticule mesh

tagm

Purpose

Assign a name to a graphics object in its `Tag` property

Syntax

`tagm(hndl,tagstr)` sets the `Tag` property of each object designated in the vector of handles `hndl` to the associated string (row) of the matrix of strings `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`	Clear current map
`clmo`	Clear specified graphics objects
`handlem`	Get handles of displayed graphics objects
`hidem`	Hide specified graphics objects
`namem`	Determine names of valid graphics objects
`showm`	Show specified graphics objects

Purpose

TerrainBase global 5-minute digital terrain data extraction

Syntax

`[datagrid,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 referencing vector.

`[datagrid,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 best available public domain data from almost 20 different sources, including the DCW-DEM and ETOPO5. The model is currently under development and will be updated as new data sources become available. The data set was created by the National Geophysical Data Center and World Data Center-A for Solid Earth Geophysics in Boulder, Colorado.

Remarks

Elevations and depths are given in meters above or below mean sea level.

The `tbase.bin` file is available on CD-ROM from

NOAA/NGDC
Mail Code E/GC3
325 Broadway
Boulder, CO 80303
USA
Tel: (303) 497-6338
Fax: (303) 497-6513

and via the Internet at

```
ftp://ftp.ngdc.noaa.gov/Solid_Earth/cdroms/TerrainBase_94/data/
```

No byte-swapping or line-ending conversion is required.

A summary of the model can be found at

```
ftp://ftp.ngdc.noaa.gov/Solid_Earth/Topography/tbase_5min/
   tbase.txt
```

Examples

Read every tenth point in the data set:

```
[datagrid,refvec] = tbase(10);
whos
  Name              Size         Bytes  Class

  datagrid        216x432       746496  double array
  refvec            1x3             24  double array

limitm(datagrid,refvec)
ans =
   -90    90     0   360
```

Read data for Korea and Japan at the full resolution:

```
scalefactor = 1; latlim = [30 45]; lonlim = [115 145];
[datagrid,refvec] = tbase(scalefactor,latlim,lonlim);
whos datagrid
  Name       Size          Bytes  Class

  datagrid 180x360       518400  double array
```

See Also

`gtopo30`	Read elevation data from GTOPO30
`etopo5`	Read data from the ETOPO5 data set
`usgsdem`	Read USGS digital elevation maps

Purpose

Project text annotation on map axes

Syntax

`h = textm(lat,lon,string)` displays the strings (rows) of the matrix of strings *`string`* at the geographic locations specified by the vectors `lat` and `lon`. The handles `h` of the displayed strings can be returned.

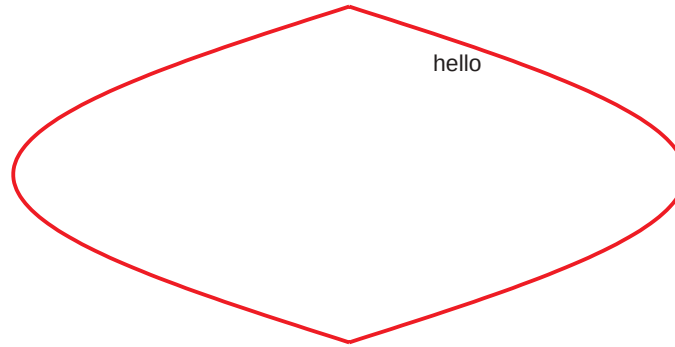
`h = textm(lat,lon,z,string)` displays the strings at the *z*-axis altitudes `z`. The default altitude is 0.

`h = textm(lat,lon,z,string,PropertyName,PropertyValue,...)` sets the text object properties. All properties supported by the MATLAB `text` function are supported by `textm`.

Example

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 string `'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.

See Also

`axesm`	Create map axes object
`text`	Create text object in current axes (see the online MATLAB Function Reference documentation)

Purpose

Read TIGER MIF (MapInfo Interchange Format) thinned boundary files

Syntax

`tigermif(namesstruc)` reads a TIGER thinned boundary file in the MIF format. The user selects the file interactively, but must provide the structure containing the names (as returned by the `fipsname` function). The patch data is returned in a Mapping Toolbox geographic data structure.

`tigermif(namesstruc,`*`filename`*`)` reads the MIF file named in the string *`filename`*. The filename is provided with the `.MIF` extension. If the file is not found, a dialog box is activated to allow the user to select a file interactively.

`tigermif(namesstruc,`*`filename`*`,pstruc)` appends the patch data to the existing structure, `pstruc`.

`tigermif(namesstruc,`*`filename`*`,pstruc,tstruc)` appends the data in the file to the existing patch and text geographic data structures, `pstruc` and `tstruc`. The text structure contains labels for the patches. This form is used with two output arguments. The arguments for the existing structures can be set to empty matrices if none are available.

`tigermif(namesstruc,`*`filename`*`,pstruc,tstruc,getcodes)` returns only the data matching the scalar or vector of numeric FIPS codes.

`pstruc = tigermif(...)` saves the returned patch data in `pstruc`.

`[pstruc,tstruc] = tigermif(...)` saves the returned patch data in `pstruc` and text labels in `tstruc`. Both are geographic data structures.

Background

TIGER Thinned Boundary files are lower resolution extracts from the U.S. Census Bureau's detailed TIGER/Line database. U.S. state and county boundaries are available in the MapInfo Interchange format (MIF).

Remarks

The data files are available over the Internet from

```
ftp://ftp.census.gov/pub/tiger/boundary/
```

Extremely limited documentation on the files is available on the World Wide Web from

```
http://www.census.gov/ftp/pub/geo/www/tiger/mif.txt
```

and

```
http://www.census.gov/ftp/pub/geo/www/tiger/resource.html
```

Examples

Read the names file (contains the names of U.S. states and territories):

```
namestruc = fipsname('st_name.dat')
namestruc =
1x57 struct array with fields:
    name
    id
```

Read the file containing Hawaii's thinned state boundaries and text labels into a Mapping Toolbox geographic data structure:

```
[ps,ts] = tigermif(namestruc,'ST15.MIF')
ps =
              lat: [1585x1 double]
             long: [1585x1 double]
             type: 'patch'
    otherproperty: {}
              tag: 'Hawaii'
         altitude: []
ts =
              lat: 21.1343
             long: -157.9524
             type: 'text'
              tag: 'maptext'
    otherproperty: {1x2 cell}
           string: {1x1 cell}
         altitude: []
```

Read the file containing Alaska's thinned state boundaries, and append it to the Hawaii data:

```
[ps,ts] = tigermif(namestruc,'ST02.MIF',ps,ts)
ps =
1x2 struct array with fields:
    lat
    long
    type
    otherproperty
    tag
    altitude
ts =
1x2 struct array with fields:
    lat
    long
    type
    tag
    otherproperty
    string
    altitude
```

Get the state boundaries and text labels for part of New England. The FIPS codes for Connecticut, Massachusetts, and Rhode Island are 9, 25, and 44, respectively:

```
[ps,ts] = tigermif(namestruc,'ST_LOW48.MIF',[],[],[9 25 44])
ps =
1x3 struct array with fields:
    lat
    long
    type
    otherproperty
    tag
    altitude
ts =
1x3 struct array with fields:
    lat
    long
    type
    tag
```

```
otherproperty
string
altitude
```

See Also

`dcwdata`	Read selected data from the *Digital Chart of the World*
`fipsname`	Read TIGER thinned boundary file FIPS names
`tgrline`	Read data from TIGER/Line files
`tigerp`	Read TIGER ArcInfo Format thinned boundary files

Purpose

Read TIGER p and pa (ArcInfo format) thinned boundary files

Syntax

`tigerp(namesstruc)` reads a TIGER thinned boundary file in the ArcInfo format. The user selects the file interactively, but must provide the structure containing the names (as returned by the `fipsname` function). The patch data is returned in a Mapping Toolbox geographic data structure.

`tigerp(namesstruc,`*`filename`*`)` reads the ArcInfo file named in the string *`filename`*. The filename is provided without the `'_p'` or `'_pa'` extension.

`tigerp(namesstruc,`*`filename`*`,pstruc)` appends the patch data to the existing structure, `pstruc`.

`tigerp(namesstruc,`*`filename`*`,pstruc,tstruc)` appends the data in the file to the existing patch and text geographic data structures, `pstruc` and `tstruc`. The text structure contains labels for the patches. This form is used with two output arguments. The arguments for the existing structures can be set to empty matrices if none are available.

`tigerp(namesstruc,`*`filename`*`,pstruc,tstruc,getcodes)` returns only the data matching the scalar or vector of numeric FIPS codes.

`pstruc = tigerp(...)` saves the returned patch data in `pstruc`.

`[pstruc,tstruc] = tigerp(...)` saves the returned patch data in `pstruc` and text labels in `tstruc`. Both are geographic data structures.

Background

TIGER Thinned Boundary files are lower resolution extracts from the U.S. Census Bureau's more detailed TIGER/Line database. State, county, minor civil division, census tract/block numbering area, American Indian reservation/Alaska native village statistical area, Alaska native regional corporation, urbanized areas, metropolitan areas, and congressional district boundaries are available in the ArcInfo format.

Remarks

Coordinate values are based on Clarke's spheroid of 1866 and the North American Datum, 1927 (NAD27).

The data files are available over the Internet from

```
ftp://ftp.census.gov/pub/tiger/boundary
```

Documentation for the files is available from

```
ftp://ftp.census.gov/pub/tiger/boundary/readme.txt
```

Examples

Read the names file with the names of all counties in the U.S. and territories. This file is in FIPS format:

```
namestruc = fipsname('co_name.dat')
namestruc =
1x3248 struct array with fields:
    name
    id
```

Read the file containing Alaska's thinned county boundaries into a Mapping Toolbox geographic data structure:

```
[ps,ts] = tigerp(namestruc,'co_02_p.dat')
ps =
1x26 struct array with fields:
    lat
    long
    type
    otherproperty
    altitude
    tag
ts =
1x26 struct array with fields:
    lat
    long
    type
    tag
    otherproperty
    altitude
    string
```

Read only the Aleutians East and West:

```
[ps,ts] = tigerp(namestruc,'co_02_p.dat',[],[],[2013 2016])
ps =
1x2 struct array with fields:
    lat
    long
    type
    otherproperty
    altitude
    tag
ts =
1x2 struct array with fields:
    lat
    long
    type
    tag
    otherproperty
    altitude
    string
```

See Also

`dcwdata`	Read selected data from the *Digital Chart of the World*
`fipsname`	Read TIGER thinned boundary file FIPS names
`tgrline`	Read data from TIGER/Line files
`tigermif`	Read TIGER MapInfo Interchange Format thinned boundary files

tightmap

Purpose

Remove white space around a map

Syntax

`tightmap` sets the MATLAB 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 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 coast
plotm(lat, long)
```

Now use `tightmap` to reduce the wasted space:

```
tightmap
```

Limitations

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.

See Also

`panzoom`	Pan and zoom on a 2-D plot
`zoom`	Zoom in and out on a 2-D plot
`paperscale`	Figure paper size for a given map scale
`axesscale`	Resize axes for equivalent scale
`previewmap`	View map at printed size

Purpose

Convert time to a clock string

Syntax

`str = time2str(timein)` converts a numerical vector of times to a string matrix. The output string matrix is useful for the display of times.

`str = time2str(timein,clock)` uses the specified *`clock`* input to construct the string matrix. Allowable *`clock`* strings are `'24'` (default) for a 24-hour clock, `'12'` for a 12-hour clock, and `'nav'` for a navigational hour clock.

`str = time2str(timein,clock,format)` uses the specified *`format`* input to construct the string matrix. Allowable for *`format`* strings are `'hms'`, for hours, minutes, and seconds, and `'hm'` (default), for hours and minutes.

`str = time2str(timein,clock,format,units)` defines the units in which the input times are supplied. Any valid time *`units`* string can be entered. If omitted, `'hours'` is assumed.

`str = time2str(timein,clock,format,digits)` indicates the power of ten to be included for the seconds (if *`format`* `= 'hms'`) or minutes (if *`format`* `= 'hm'`). The default value is 0, so nothing is returned to the right of the decimal (10^0 is the ones column). For example, if `digits = -2`, seconds are returned down to the hundredths column.

Description

The purpose of this function is to make time-valued variables into strings suitable for map display.

Examples

13 hours, 56 minutes, 44 seconds in hms format is `1356.44`.

```
time = 1356.44;
str = time2str(time,'12','hms','hms')
str =
1:56:44 PM
```

For hm format, appropriate rounding occurs:

```
str = time2str(time,'12','hm','hms')
str =
1:57 PM
```

The 24-hour and navigational representations are

```
str = time2str(time,'24','hms','hms')
str =
```

```
13:56:44
str = time2str(time,'nav','hms','hms')
str =
1356'''
```

Navigational times are four digits; if seconds are included, they are rounded to the nearest 15 seconds, which are represented by tick marks (`0 = none`, `15 = '`, `30 = ''`, `45 = '''`).

Consider the hms format time 1356.4456 for rounding purposes:

```
str = time2str(1356.4456,'12','hms','hms',-2)  % hundredths
str =
1:56:44.56 PM
str = time2str(1356.4456,'12','hms','hms',-1)  % tenths
str =
1:56:44.6 PM
```

See Also

`hr2hms` `hr2sec`	Other direct time conversion functions
`timedim`	Convert times between different units

Purpose

Convert times between different units

Syntax

`timeout = timedim(timein,from,to)` returns the value of the input time `timein`, which is in units specified by the valid time units string *`from`*, in the desired units given by the valid time units string *`to`*. Valid time units strings are

```
'hours' or 'hr'        for decimal hours
'seconds' or 'sec'     for seconds
'hms'                  for hours-minutes-seconds
'hm'                   for hours-minutes
```

Examples

Convert from hours to seconds:

```
timedim(2.56,'hours','seconds')
ans =
        9216
```

What is the difference between hms and hm (best displayed in *bank format*)?

```
format bank
timedim(2.56,'hours','hms')
ans =
        233.36

timedim(2.56,'hours','hm')
ans =
        234.00
```

The hm answer is the hms answer correctly rounded to whole minutes (that is, rounded based on 60 seconds per minute, not 100).

See Also

`angledim`	Convert angle units
`distdim`	Convert distance units
`hr2hms` `hr2sec`	Other direct time conversion functions
`time2str`	Convert time to selected display string

timezone

Purpose

Determine time zone based on longitude

Syntax

`[zd,zltr,zone] = timezone(long)` returns an integer zone description, `zd`, an alphabetical string zone indicator, `zltr`, and a string, `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`* string. The default value is `'degrees'`.

Examples

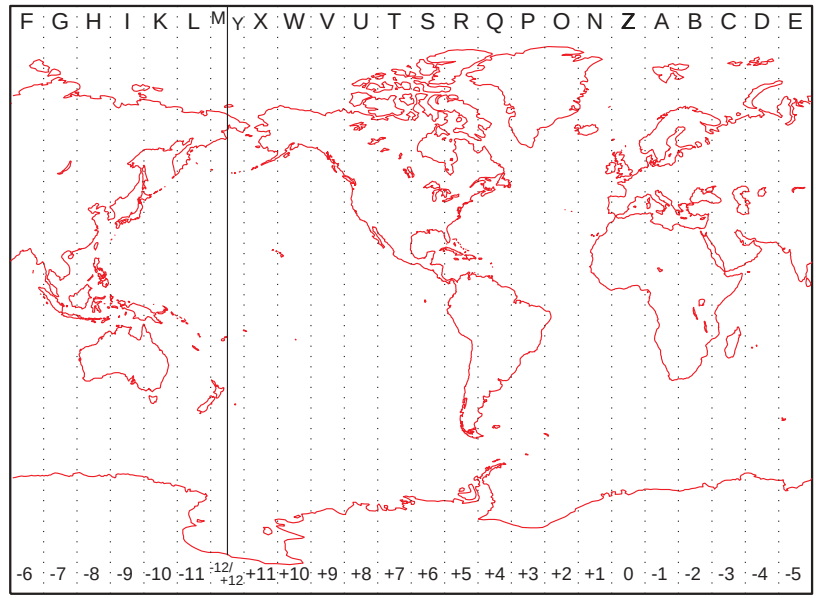
Given that it is locally 1330 (1:30 p.m.) at a longitude of 75°W, determine GMT:

```
[zd,zltr,zone] = timezone(-75,'degrees')
zd =
     5
zltr =
R
zone =
+5 R
```

Greenwich Mean Time (GMT) 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 GMT. For local times elsewhere, the Earth is divided into 15° 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 GMT. For notational convenience, each zone is also given an alphabetical indicator. The indicator at Greenwich is *Z*, so GMT is often called *ZULU time*.



Note that there are actually 25 time zones, because the zone centered on the International Date Line (180° 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.

tissot

Purpose

Project Tissot indicatrices onto map axes

Syntax

`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,`*`linestyle`*`)` and `h = tissot(linestyle)` specify any *`linestyle`* string recognized by the standard MATLAB function `line` to set the line style of the Tissot indicatrices.

`h = tissot(spec,`*`PropertyName`*`,PropertyValue,...)` and `h = tissot(`*`linestyle`*`,`*`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 meridianal 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. Chapter 11, "Projections Reference," of this guide includes Tissot diagrams for every projection on a global scale.

Description

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.

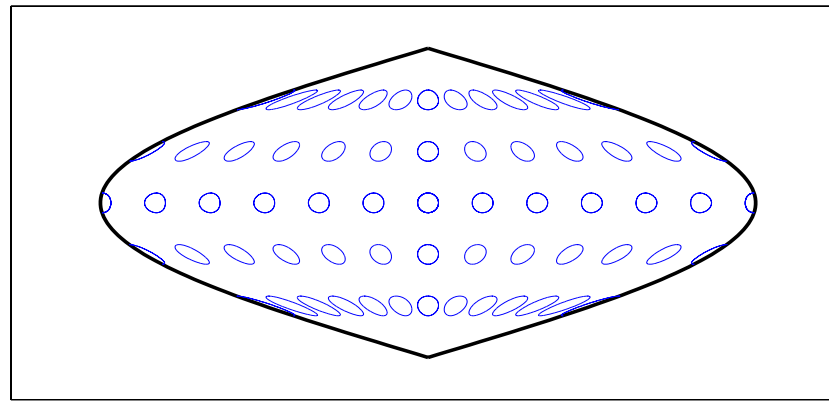
`Latint` is the latitude interval between indicatrix circle centers. If entered it should be in the map axes `AngleUnits`. The default value is one circle every 30° of latitude (that is, 0°, +/-30°, etc.).

`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 30° of latitude (that is, 0°, +/-30°, etc.).

`Points` is the number of plotting points per circle. The default is 100 points.

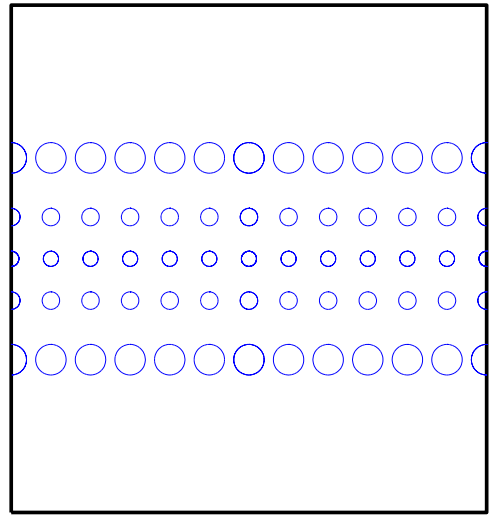
Examples

```
axesm sinusoid; framem
tissot
```



The Sinusoidal projection is equal area.

```
setm(gca,'MapProjection','Mercator')
```



The Mercator projection is conformal.

See Also

`mdistort`	Display contours of constant distortion on a map
`distortcalc`	Calculate distortion parameters for a map projection
`projections`	See Chapter 11, “Projections Reference.”

Purpose

Read data from U.S. Census Bureau TIGER/Line files

Syntax

`[CL,PR,SR,RR,H,AL,PL] = tgrline(`*`filename`*`)` reads the set of 1994 TIGER/Line files that share the same filename, but different extensions. The results are returned in a set of geographic data structures containing the county boundaries, primary roads, secondary roads, railroads, hydrography, area landmarks, and point landmarks, respectively. The data is tagged with the names of the features.

`[CL,PR,SR,RR,H,AL,PL] = tgrline(`*`filename`*`,year)` reads the TIGER/Line files in the format from the specified year. The files are updated periodically, and the format and filename extensions can change from year to year. Valid years are 1990, 1992, 1994, and 1995.

`[CL,PR,SR,RR,H,AL,PL] = tgrline(`*`filename`*`,year,`*`countyname`*`)` uses the string *`countyname`* to tag the county data.

Background

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 Mariana 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.

Remarks

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.

TIGER/line files are available from the U.S. Census Bureau on CD-ROMs. Ordering information and an overview of other Census Bureau data products are available at

```
http://www.census.gov/ftp/pub/geo/www/tiger/
```

Example files are available over the Internet at

```
ftp://ftp.census.gov/pub/tiger/line/
```

Examples

Read from the data for Washington, D.C.:

```
[CL,PR,SR,RR,H,AL,PL] = tgrline('TGR11001',1994,'Wash,DC');
```

See Also

`dcwdata`	Read selected data from the *Digital Chart of the World*
`tigermif`	Read TIGER MapInfo Interchange Format thinned boundary files
`tigerp`	Read TIGER ArcInfo format thinned boundary files

Purpose

Connect navigational waypoints with track segments

Syntax

`[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 elliptical definition of the Earth with a two-element ellipsoid model vector `ellipsoid`. The default ellipsoid is a spherical Earth, which is sufficient for most applications.

`[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

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 coast
```

```
axesm('mercator','MapLatLimit',[30 47],'MapLonLimit',[-10 37])
plotm(lat,long,'b')
```

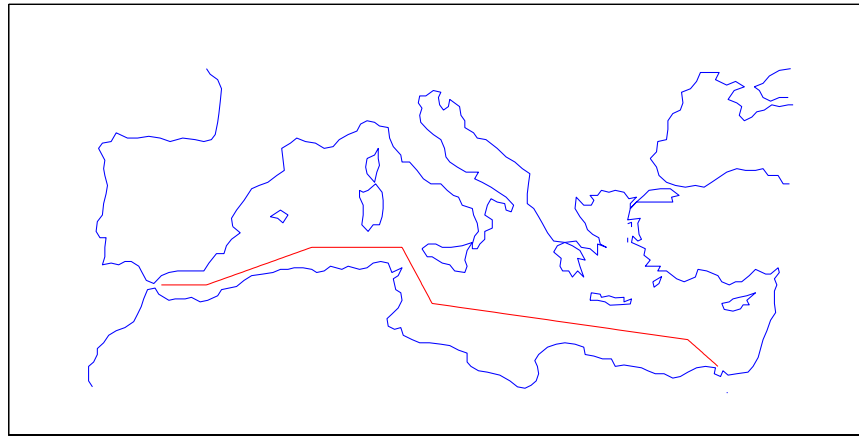
These are the waypoints Lt. Sextant has selected:

```
waypoints = [36,-5; 36,-2; 38,5; 38,11; 35,13; 33,30; 31.5,32]
waypoints =
   36.0000   -5.0000
   36.0000   -2.0000
   38.0000    5.0000
   38.0000   11.0000
   35.0000   13.0000
   33.0000   30.0000
   31.5000   32.0000
```

Now display the track:

```
[lttrk,lntrk] = track('rh',waypoints,'degrees');
plotm(lttrk,lntrk,'r')
```

With a display this clear, the captain gladly approves the plan.



See Also

`dreckon`	Dead reckon points for a track
`gcwaypts`	Find waypoints along a great circle

`legs`	Courses and distances between waypoints
`navfix`	Mercator-based navigational fixing

track1

Purpose

Compute great circle or rhumb line track defined by point, azimuth, and range

Syntax

`[lattrk,lontrk] = track1(lat,lon,az)` returns, in `lattrk` and `lontrk`, points along a complete (great circle) track passing through the point specified by `lat` and `lon` with an initial azimuth at that point of `az`. When the inputs are column vectors, the successive tracks are stored in separate columns of `lattrk` and `lontrk`.

`[lattrk,lontrk] = track1(`*`track`*`,lat,lon,az)` allows the specification of the track logic to be employed. A string *`track`* of `'gc'` is the default, resulting in a great circle track. A *`track`* of `'rh'` results in a complete rhumb line track.

`[lattrk,lontrk] = track1(`*`track`*`,lat,lon,az,`*`units`*`)` specifies the units of the inputs and outputs, where *`units`* is any valid angle unit string. The default is `'degrees'`.

`[lattrk,lontrk] = track1(`*`track`*`,lat,lon,az,rng)` specifies the range of the track. `rng` is a one- or two-column matrix. If `rng` has one column, the track extends from the point `(lat,lon)` at an azimuth of `az` for a distance `rng` if `rng` is positive, or at an azimuth `az+180°` (or its angular equivalent) for a distance of `abs(rng)` if `rng` is negative. If `rng` has two columns, the endpoints are defined as above. In this case, the segment extends from the point associated with the first column of `rng` to the point associated with the second column. `rng` is in *`units`* (unless a `ellipsoid` is input). When no `rng` is provided, or `rng` is empty, a *complete* track is returned.

`[lattrk,lontrk] = track1(track,lat,lon,az,rng,ellipsoid,`*`units`*`)` specifies the elliptical definition of the Earth with a two-element ellipsoid model vector `ellipsoid`. The default ellipsoid is a spherical Earth, which is sufficient for most applications. If used, the units of the semimajor axis of the ellipsoid vector define the units for the `rng` input, overriding *`units`* for this purpose.

`[lattrk,lontrk] = track1(`*`track`*`,lat,lon,az,rng,ellipsoid,`*`units`*`,npts)` specifies the number of points, `npts`, per output track. `npts` is 100 by default.

`pts = track1(lat,lon,az,...)` combines the outputs into a single two-column matrix, `pts`.

Background

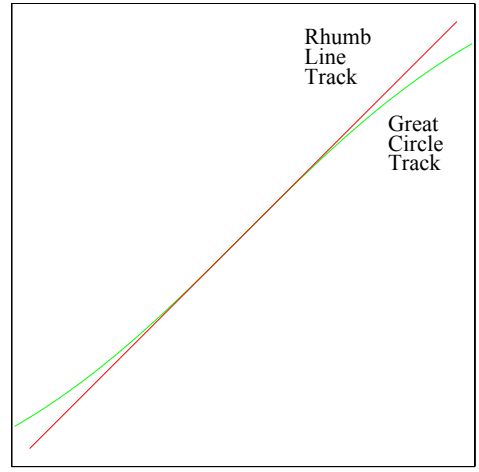
A path along the surface of the Earth connecting two points is a *track*. Two types of track lines are of interest geographically, great circles and rhumb

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.

Examples

```
axesm('mercator','MapLatLimit',[-60 60],'MapLonLimit',[-60 60])
[lattrkgc,lontrkgc] = track1(0,0,45,[-55 55]);
plotm(lattrkgc,lontrkgc,'g')
[lattrkrh,lontrkrh] = track1('rh',0,0,45,[-55 55]);
plotm(lattrkrh,lontrkrh,'r')
```



See Also

`azimuth`	Azimuth between two points on the globe
`distance`	Distance between two points on the globe
`reckon`	New point with an azimuth and distance
`scircle1` `scircle2`	Small circle coordinates
`track`	Connect waypoints with track segments

`track2`	Great circle or rhumb line defined by two points
`trackg`	Interactive tracks

Purpose

Compute great circle or rhumb line track defined by two points

Syntax

`[lattrk,lontrk] = track2(lat1,lon1,lat2,lon2)` returns, in `lattrk` and `lontrk`, points along a (great circle) track between the points specified by `lat1` with `lon1` and `lat2` and `lon2`. When the inputs are column vectors, the successive tracks are stored in separate columns of `lattrk` and `lontrk`.

`[lattrk,lontrk] = track2(track,lat1,lon1,lat2,lon2)` allows the specification of the track logic to be employed. A string *`track`* of `'gc'` is the default, resulting in a great circle track. A *`track`* of `'rh'` results in a rhumb line track.

`[lattrk,lontrk] = track2(`*`track`*`,lat1,lon1,lat2,lon2,`*`units`*`)` specifies the units of the inputs and outputs, where *`units`* is any valid angle unit string. The default is `'degrees'`.

`[lattrk,lontrk] = track2(`*`track`*`,lat1,lon1,lat2,lon2,ellipsoid,`*`units`*`)` specifies the elliptical definition of the Earth with a two-element ellipsoid model vector `ellipsoid`. The default ellipsoid is a spherical Earth, which is sufficient for most applications.

`[lattrk,lontrk] = track2(lat1,lon1,lat2,lon2,ellipsoid,`*`units`*`,npts)` specifies the number of points, `npts`, per output track. `npts` is 100 by default.

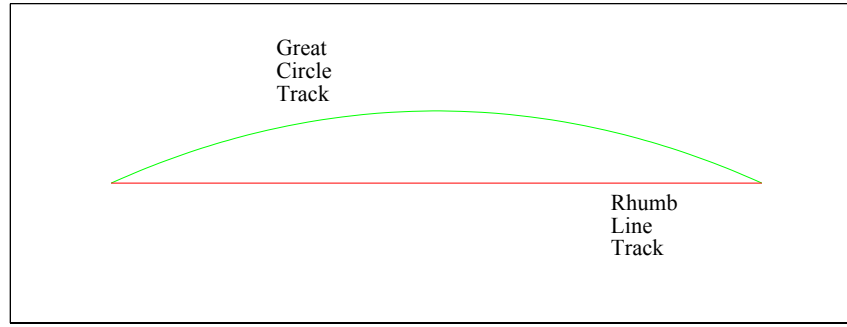
`pts = track2(lat1,lon1,lat2,lon2,...)` combines the outputs into a single two-column matrix, `pts`.

Background

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.

Example

```
axesm('mercator','MapLatLimit',[30 50],'MapLonLimit',[-40 40])
[lattrkgc,lontrkgc] = track2(40,-35,40,35);
[lattrkrh,lontrkrh] = track2('rh',40,-35,40,35);
plotm(lattrkgc,lontrkgc,'g')
plotm(lattrkrh,lontrkrh,'r')
```



See Also

`azimuth`	Azimuth between two points on the globe
`distance`	Distance between two points on the globe
`reckon`	New point with an azimuth and distance
`scircle1` `scircle2`	Small circle coordinates
`track`	Connect waypoints with track segments
`track1`	Great circle or rhumb line from point and azimuth
`trackg`	Interactive tracks

Purpose

Display great circle or rhumb line track defined via mouse input

Syntax

`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,`*`linestyle`*`)` specifies the line style for the displayed track segments, where *`linestyle`* is any line style string recognized by the standard MATLAB function `line`.

`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 the string *`track`* is `'gc'` (the default), a great circle path is used. If *`track`* is `'rh'`, rhumb line logic is used.

Description

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.

See Also

`track1`	Great circle or rhumb line from point, azimuth, and range
`track2`	Great circle or rhumb line from two points
`scircleg`	Interactive small circles

trimcart

Purpose

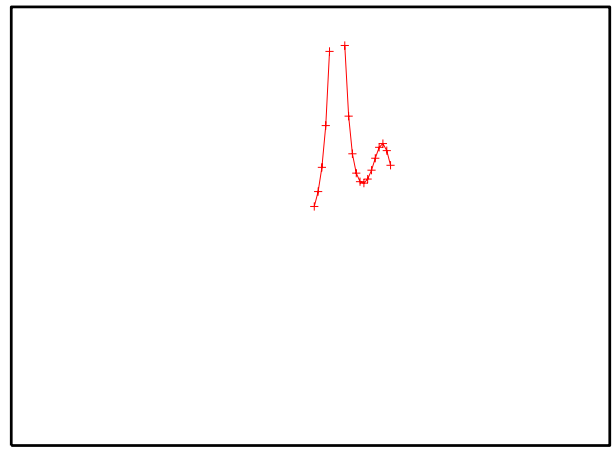
Trim graphic objects to the map frame

Syntax

`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

```
str = unitstr('sm','distances')
axesm('miller','ellipsoid',[25 0])
framem
h = plot(humps,'r+-');
trimcart(h)
```



Limitations

`trimcart` does not trim patch objects.

See Also

`handlem`	Graphics handle for identified objects
`makemapped`	Make an object mapped

Purpose

Trim map data exceeding projection limits

Syntax

`[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 object strings 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`

unitsratio

Purpose

Unit conversion factors

Syntax

`ratio = unitsratio(to, from)` returns the number of `to` units per one `from` unit. For example, `unitsratio('cm', 'm')` returns 100 because there are 100 centimeters per meter. `unitsratio` makes it easy to convert from one system of units to another. Specifically, if `x` is in units `from` and

```
y = unitsratio(to, from) * x
```

then Y is in units `to`.

`to` and `from` can be any strings from the second column of one of the following tables (both must come from the same table). `to` and `from` are case insensitive and can be either singular or plural.

Units of Length

`unitsratio` recognizes the following identifiers for converting units of length:

Unit Name	String(s)
Meter	`'m', 'meter(s)', 'metre(s)'`
Centimeter	`'cm', 'centimeter(s)', 'centimetre(s)'`
Millimeter	`'mm', 'millimeter(s)', 'millimetre(s)'`
Micron	`'micron(s)'`
Kilometer	`'km', 'kilometer(s)', 'kilometre(s)'`
Nautical mile	`'nm', 'nautical mile(s)'`
International foot	`'ft', 'international ft', 'foot', 'international foot', 'feet', 'international feet'`
Inch	`'in', 'inch', 'inches'`
Yard	`'yd', 'yard(s)'`
international mile	`'mi', 'mile(s)', 'international mile(s)'`

Unit Name	String(s)
U.S. survey foot	'sf', 'survey ft', 'U.S. survey ft', 'survey foot', 'U.S. survey foot', 'survey feet', 'U.S. survey feet'
U.S. survey mile (statute mile)	'sm', 'survey mile(s)', 'statute mile(s)', 'U.S. survey mile(s)'

Units of Angle

`unitsratio` recognizes the following identifiers for converting units of angle:

Unit Name	String(s)
radian	'rad', 'radian(s)'
degree	'deg', 'degree(s)'

Examples

```
% Approximate mean earth radius in meters
radiusInMeters = 6371000
% Conversion factor
feetPerMeter = unitsratio('feet', 'meter')
% Radius in (international) feet:
radiusInFeet = feetPerMeter * radiusInMeters
% The following prints a true statement for valid TO, FROM pairs:
to   = 'feet';
from = 'mile';
sprintf('There are %g %s per %s.', unitsratio(to,from), to, from)
% The following prints a true statement for valid TO, FROM pairs:
to   = 'degrees';
from = 'radian';
sprintf('One %s is %g %s.', from, unitsratio(to,from), to)
```

unitstr

Purpose

Test for valid unit strings or abbreviations

Syntax

`unitstr` lists all valid unit strings and all abbreviations that are not simply truncations of the original strings (e.g., `'km'` for `'kilometers'`).

`str = unitstr(`*`str0`*`,`*`measstr`*`)` returns the valid standard string *`str`* corresponding to the recognized abbreviation *`str0`*. The type of string sought is specified by *`measstr`*, which can be `'distances'`, `'angles'`, or `'time'`.

Examples

This function recognizes and standardizes certain abbreviations:

```
str = unitstr('sm','distances')
str =
statutemiles
```

And any unique truncation:

```
str = unitstr('hou','time')
str =
hours
```

See Also

`angledim`	Angle unit conversion
`distdim`	Distance unit conversion
`timedim`	Time unit conversion

Purpose

Update a version 1 geographic data structure to a version 2 geographic data structure

Syntax

`g2 = updategeostruct(g)` accepts a geographic data structure `g`. If `g` is a geostruct1 for which the `'type'` field has value `'line'` or `'patch'`, `updategeostruct` restructures its elements to create a geostruct2, `g2`. If `g` is a geostruct2, it is copied unaltered to `g2`. `updategeostruct` should not be used for geostruct1 arrays of type `'text'`, `'light'`, `'regular'`, or `'surface'`.

`s = updategeostruct(g, str)` selects only elements whose `tag` field begins with the string `str` (and whose `type` field is either `'line'` or `'patch'`). The selection is case insensitive.

`[s,symbolspec] = updategeostruct(g, ...)` restructures a geographic data structure and determines a `symbolspec` based on the graphic properties specified in the `otherproperty` field for each element of `g` and, if necessary, the `jet` colormap.

`[s,symbolspec] = updategeostruct(g, ..., cmap)` specifies a colormap, `cmap`, to define the colors used in `symbolspec`.

Remarks

The Mapping Toolbox supports two ways of encoding vector features in MATLAB structure arrays. In both cases there is one feature per array element, and in both cases the array elements are called “geographic data structures.” Mapping Toolbox Version 1.3.1 and earlier supported the “version 1” geographic data structure (called geostruct1), in which

- 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.
- All coordinates are in latitude-longitude, stored in fields `lat` and `long`.
- An `altitude` coordinate array extends coordinates to 3-D.
- 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 geostruct1 can contain

- Vector geodata (`type` is `'line'` or `'patch'`)
- Raster geodata (`type` is `'surface'` or `'regular'`)
- Graphic objects (`type` is `'text'` or `'light'`)

Beginning with Mapping Toolbox 2.0, geographic data structures can take a more general form (geostruct2) — but only for vector geodata:

- Coordinates can be in either *latitude-longitude* (stored in fields `Lat` and `Lon`) or map *x-y* (stored in fields `X` and `Y`).
- An optional field, `Height` or `Z`, extends coordinates to 3-D.
- A `Geometry` text field designates the geometric nature of the feature: `'Point'`, `'Multipoint'`, `'Line'`, or `'Polygon'` rather than a graphics object type.
- Additional attribute fields, the names and number of which are data-set-specific, describe the nongeometric properties (name, ownership, age, code or identifier, ...).

This is the form of geostruct used for the output of `shaperead`. The version 2 geographic data structures allow for a greater amount of information to be carried about each vector feature. They also separate the graphics display properties from the fundamental properties of the geographic features themselves.

Instead of being assigned in advance, graphics properties are determined at display time by matching up attribute values against rules provided in a symbol spec. 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.

Some version 2 toolbox functions (for example, `mapshow`, `geoshow`, and `mapview`) accept either type of geographic data structure. Other functions (for example, `displaym` and `extractm`) accept only version 1 geographic data structures. The purpose of `updategeostruct`, which supports the implementation of `mapshow` and `geoshow`, is to restructure version 1 geographic data structures containing vector geodata, converting them to the newer form.

You might need to update gstructs in order to use existing Mapping Toolbox data with newer display functions. You must, for example, use `updategeostruct` when displaying objects by name with `geoshow` and `mapshow`. The following commands (`displaym` and `geoshow`) yield similar results (fill color might differ):

```
load usalo; axesm miller
displaym(state,'california');

figure; axesm miller
geoshow(updategeostruct(state,'california'));
```

If you do not reference objects by name, `geoshow` and `mapshow` can use existing mstructs. For example,

```
geoshow(state)
```

draws all state patches, and

```
geoshow(state(5))
```

draws just California.

Example

```
% Draw the United States with colors from the autumn colormap
patches = usahi('statepatch');
cmap = autumn(numel(patches));
[states, spec] = updategeostruct(patches, cmap);
mapshow(states,'SymbolSpec',spec)
```

See Also

`extractm`, `shaperead`, `makesymbolspec`, `mapview`, `mapshow`, `geoshow`

undoclip

Purpose

Remove object clips introduced by `clipdata`

Syntax

`[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`.

Description

Allowable object strings 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`

Purpose

Remove object trims introduced by `trimdata`

Syntax

`[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`.

Description

Allowable object strings 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`

usahi

Purpose

Return data from the `usahi` atlas data file

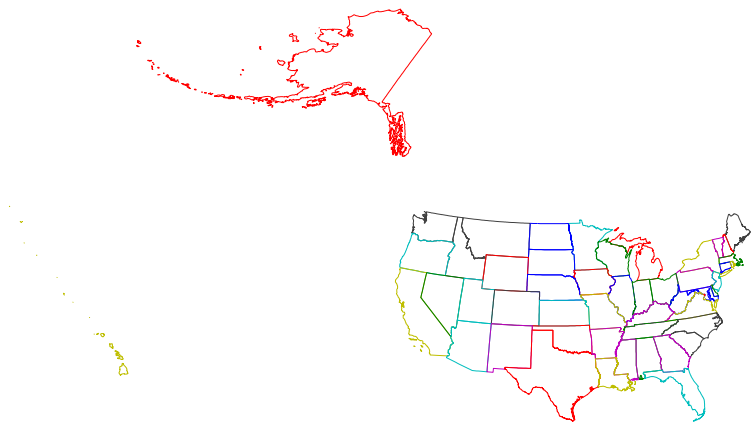
Syntax

`usahi` types a list of the atlas data variables in the `usahi` MAT-file to the screen.

`s = usahi(request)` returns the requested variable. Valid requests are `'stateline'`, `'statepatch'`, and `'statetext'`. This function can be used as an argument to other commands such as `geoshow(usahi('statepatch'))`.

Examples

```
axesm('bonne','origin',-100)
geoshow(usahi('stateline'))
```



See Also

`worldlo`	World atlas data
`usalo`	Low-resolution atlas data for the United States
`usahi.mat`	MAT-file containing high-resolution atlas data for the United States

Purpose

Return data from the `usalo` atlas data file

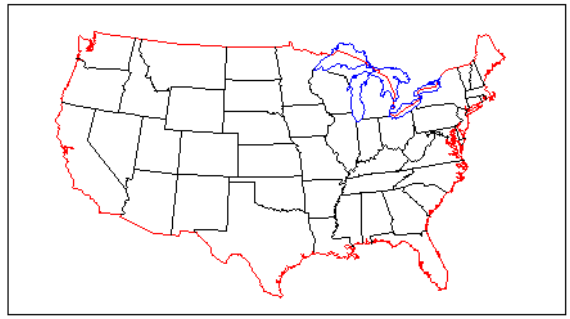
Syntax

`usalo` types a list of requests for the `usalo` MAT-file to the screen.

`s = usalo(request)` returns the requested data. Valid requests for two-column vectors are `'conusvec'`, `'statebvec'`, and `'gtlakevec'`. Valid requests for geographic data structures are `'conus'`, `'state'`, `'stateborder'`, and `'greatlakes'`. This function can be used as an argument to other commands such as `geoshow(usalo('conus'))`.

Examples

```
axesm('bonne','origin',-100)
usvec = usalo('conus')
usvec =
              lat: [4339x1 double]
             long: [4339x1 double]
             type: 'patch'
              tag: 'ContinentalUnitedStates'
    otherproperty: []
         altitude: []
stvec = usalo('stateborder')
stvec =
              lat: [2345x1 double]
             long: [2345x1 double]
             type: 'line'
              tag: 'StateBorder'
    otherproperty: {'k:'}
         altitude: []
glvec = usalo('gtlakevec');
geoshow(usvec.lat,usvec.long,'color','red')
geoshow(stvec.lat,stvec.long,'color','black')
geoshow(glvec(:,1),glvec(:,2),'color','blue')
```



See Also

`worldlo`	World atlas data
`usahi`	High-resolution atlas data for the United States
`usalo.mat`	MAT-file containing low-resolution atlas data for the United States

Purpose

Create a map of the United States of America

Syntax

`usamap` creates a map of all or some of the United States of America. The state or states are selected interactively from a list. A map axes and map are created in the current axes. The axis limits are set tight around the map frame.

`h = usamap('all')` or `usamap all` creates a standard map of the U. S. The conterminous states, Alaska, and Hawaii are each displayed as insets in different axes using projection parameters suggested by the U. S. Geological Survey. The handles for the three map axes are returned in `h`. `h(1)` contains the conterminous states, `h(2)` Alaska, and `h(3)` Hawaii.

`usamap allequal` creates the map with Alaska and Hawaii at the same scale as the conterminous states.

`usamap conus` maps only the conterminous states.

`usamap` *`state`* maps the requested state. For example, `usamap vermont`. *`state`* can also be a padded string matrix or a cell array of strings containing multiple state names.

`usamap` *`stateonly`* maps only that state. Example: `usamap vermontonly`. If any of the state names in a *`state`* string matrix or cell array of strings end with `'only'`, only the requested states are displayed.

`usamap(`*`state`*`,`*`type`*`)` controls the atlas data displayed. Type `'line'` or `'patch'` creates a map with atlas data of those types and text annotation of the state names. Type `'lineonly'` or `'patchonly'` suppresses text annotation. Type `'none'` suppresses all atlas data. If omitted, type `'patch'` is assumed.

`usamap(latlim,lonlim)` creates a map of the states covering the provided latitude and longitude limits. The limits are two-element vectors in units of degrees.

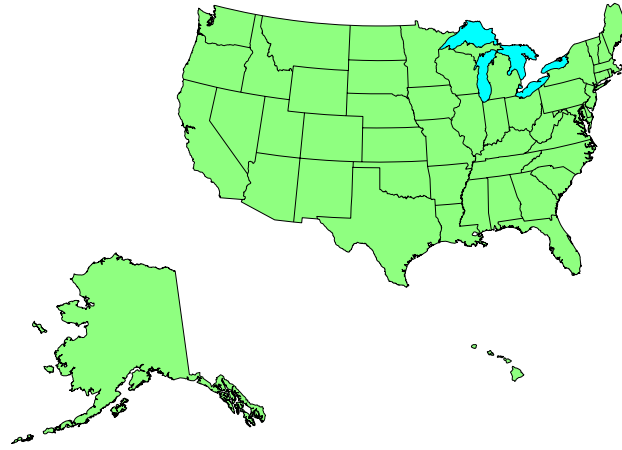
`usamap(latlim,lonlim,`*`type`*`)` is also a valid calling form.

`usamap(lmap,refvec)` and `usamap(lmap,refvec,'type')` use the supplied regular data grid to define the geographic limits. The data grid is displayed using `MESHM` unless `type` is `'none'`, `'lineonly'`, `'patch'`, or `'patchonly'`. Use type `'meshonly'` to display only the data grid.

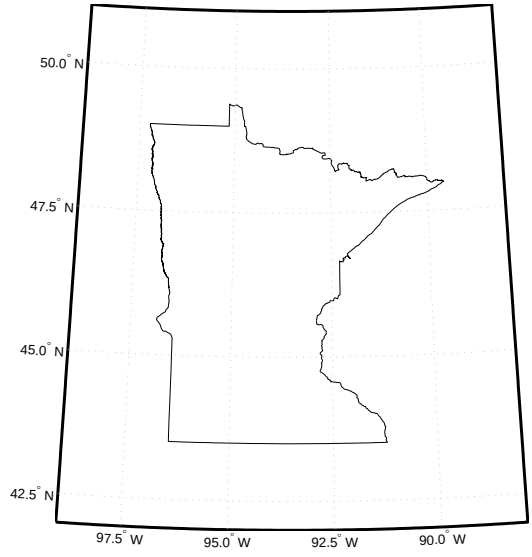
`h = usamap(...)` returns the handle or handles of the axes containing the map.

Examples

```
usamap allequal
```



```
usamap('minnesotaonly','line')
```



Remarks

`usamap` uses `tightmap` to set the axis limits tight around the map. If you change the projection, or just want more white space around the map frame, use `axis auto`.

`axes(h(n))`, where `n = 1`, `2`, or `3`, makes the desired axes current.

`set(h,'Visible','on')` makes the axes visible.

`set(h,'ButtonDownFcn','selectmoveresize')` allows interactive repositioning of the axes. `set(h,'ButtonDownFcn','uimaptbx')` restores the Mapping Toolbox interfaces.

`axesscale(h(1))` resizes the axes containing Alaska and Hawaii to the same scale as the conterminous states.

See Also

`worldmap`	Map a country or region using `worldlo` atlas data
`usalo`	Return data from the `usalo` atlas data file
`selectmoveresize`	Interactively select, move, resize, or copy objects
`axesscale`	Resize axes for equivalent scale
`paperscale`	Figure paper size for a given map scale

Purpose

Read USGS 7.5 minute 1:24,000 (30 m) digital elevation model files

Syntax

`[latgrat,longrat,mat] = usgs24kdem` reads a USGS 1:24,000 digital elevation map (DEM) file in standard format. The file is selected interactively. The entire file is read and subsampled by a factor of 5, and returned as a general data grid.

`[latgrat,longrat,mat] = usgs24kdem(`*`filename`*`)` specifies the name of the DEM file.

`[latgrat,longrat,mat] = usgs24kdem(`*`filename`*`,scalefactor)` subsamples the file by the `scalefactor`. If omitted, the default is 5, which returns every 5th point.

`[latgrat,longrat,mat] = usgs24kdem(`*`filename`*`,scalefactor,latlim,lonlim)` returns data for the requested geographic area. The area is specified as two-element vectors in units of degrees. The data extends somewhat outside the requested area. If omitted, the entire area covered by the DEM file is returned.

`[latgrat,longrat,mat] = usgs24kdem(`*`filename`*`,scalefactor,latlim,lonlim, gsize)` also controls the graticule size. `gsize` is a two-element vector specifying the number of rows and columns in the latitude and longitude matrices. If omitted, a graticule the same size as the map is returned.

`[latgrat,longrat,mat,A,B] = usgs24kdem(...)` also returns the contents of the `A` and `B` records of the DEM file. The `A` record is a header to the file containing descriptions of the data. The `B` record is the raw profile data from which the data grid is constructed.

Background

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 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 contiguous United States, Hawaii, and Puerto Rico. The data has been released in a number of formats. This function reads the data in the “standard” file format.

Examples

Get the 1:24,000 DEM for south San Francisco, available from the Bay Area Regional Database at

```
<http://bard.wr.usgs.gov/htmldir/dem_html/dem-sf.html>
```

The URL for the San Francisco South DEM is

```
<http://bard.wr.usgs.gov/bard/dem/dems24k/sanfrancisco/sanfranci
scos.dem>
```

Be sure you save to a file rather than opening in a window.

Read the entire file, taking every second point.

```
[latgrat,longrat,mat] = usgs24kdem('sanfranciscos.dem',2);
whos
  Name          Size         Bytes  Class

  latgrat     232x186       345216  double array
  longrat     232x186       345216  double array
  mat         232x186       345216  double array
```

Read the DEM at full resolution, limiting the area to the San Bruno mountains. These limits can be defined using `inputm` on a display of the above data. Also return the header record.

```
axesm mercator
surfm(latgrat,longrat,mat,,mat-max(mat(:)))
[latlim,lonlim] = inputm(2);
[latgrat,longrat,mat,A] = usgs24kdem('sanfranciscos.dem',1,...
                                     latlim,lonlim);
whos
  Name          Size         Bytes  Class

  A             1x1           4740  struct array
  latgrat     162x202       261792  double array
  latlim        1x2             16  double array
  longrat     162x202       261792  double array
  lonlim        1x2             16  double array
  mat         162x202       261792  double array

A
A =
```

```
                    Quadranglename: 'SAN FRANCISCO SOUTH, CA  BIG
                                    BASIN DEM'
                       TextualInfo: 'WMC                        CTOG'
                            Filler: ''
                       ProcessCode: ''
                           Filler2: ''
                 SectionalIndicator: ''
                       MCoriginCode: ''
                       DEMlevelCode: 2
              ElevationPatternCode: 'regular'
PlanimetricReferenceSystemCode: 'UTM'
                              Zone: 10
            ProjectionParameters: [15x1  double]
                 HorizontalUnits: 'meters'
                  ElevationUnits: 'feet'
               NsidesToBoundingBox: 4
                        BoundingBox: [ 8x1  double]
                   MinMaxElevations: [ 2x1  double]
                      RotationAngle: 0
                  AccuracyCode: 'accuracy information in record C'
                     XYZresolutions: [ 3x1  double]
                         NrowsCcols: [ 2x1  double]
                     MaxPcontourInt: ''
                  SourceMaxCintUnits: ''
                     SmallestPrimary: ''
                  SourceMinCintUnits: ''
                      DataSourceDate: ''
                     DataInspRevDate: ''
                         InspRevFlag: ''
                  DataValidationFlag: ''
                     SuspectVoidFlag: ''
                       VerticalDatum: ''
                     HorizontalDatum: ''
                         DataEdition: ''
                         PercentVoid: ''
```

Remarks

This function reads USGS DEM files stored in the UTM projection. 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 by contacting the U.S. Geological Survey. Other agencies have made some of the data available online. Sources for data for the San Francisco Bay area are

```
<http://bard.wr.usgs.gov/>
```

and

```
<ftp://bard.wr.usgs.gov/bard/dem/dems24k/>
```

Extensive documentation on the data format and standards is available from

```
<http://mapping.usgs.gov/www/ti/DEM/standards_dem.html>
```

and

```
<ftp://mapping.usgs.gov/pub/ti/DEM/demguide/>
```

The DEM files are ASCII files, and can be transferred as text. Line-ending conversion is not necessarily required.

See Also

`usgsdem`	USGS 1-Degree (3-arc-sec resolution) digital elevation data
`dted`	U. S. Department of Defense Digital Terrain Elevation Data (DTED) data
`gtopo30`	30-Arc-Sec global digital elevation data
`tbase`	TerrainBase Global 5-Min digital terrain data
`etopo5`	ETOPO5 Global 5-Min digital terrain data

Purpose

Read USGS 1-Degree (3-arc-sec resolution) DEM data

Syntax

`[datagrid,refvec] = usgsdem(`*`filename`*`,scalefactor)` reads the specified file and returns the data in a regular data grid. 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.

`[datagrid,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).

Remarks

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.

The DEMs are available over the Internet from

```
ftp://edcftp.cr.usgs.gov/pub/data/DEM/250/
```

The files are available sorted by state or can be selected from an index map at

```
http://edcwww.cr.usgs.gov/doc/edchome/ndcdb/2MIL/2milmap.html
```

Some documentation for the data sets is available at

```
http://edcwww.cr.usgs.gov/glis/hyper/guide/1_dgr_dem
```

Examples

Read every fifth point in the file containing part of Rhode Island and Cape Cod:

```
[datagrid,refvec] = usgsdem('providence-e',5);
```

Read the elevation data for Martha's Vineyard at full resolution:

```
[datagrid,refvec] = usgsdem('providence-e',1,...
   [41.2952 41.4826],[-70.8429 -70.4392]);
whos datagrid
  Name        Size         Bytes  Class

  datagrid  226x485       876880  double array
```

See Also

`usgs24kdem`	Read elevation data from 1:24,000 USGS DEM files
`gtopo30`	Read elevation data from GTOPO30
`etopo5`	Read data from the ETOPO5 data set
`tbase`	Read data from the TerrainBase data set
`usgsdems`	Read USGS digital elevation map filenames

References

See reference `[7]` in the Bibliography located at the end of this chapter.

Purpose

USGS 1-Degree (3-arc-sec resolution) DEM filenames

Syntax

`[fname,qname] = usgsdems(latlim,lonlim)` returns cell arrays of the DEM filenames 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 filenames 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 filenames for a given geographic region.

Remarks

This function only returns filenames for the contiguous United States.

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'
```

See Also

`usgsdem` Read USGS digital elevation maps

References

See reference `[7]` in the Bibliography located at the end of this chapter.

utmzone

Purpose

Define Universal Transverse Mercator projection zone

Syntax

`zone = utmzone` selects a Universal Transverse Mercator (UTM) zone with a graphical user interface. The zone designation is returned as a string.

`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`*`)`, where *`zone`* is a valid UTM zone designation, returns the geographic limits of the zone. 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.

`[zone,msg] = utmzone(...)` and `[latlim,lonlim,msg] = utmzone(...)` return a message if there is an error. `msg` is empty when there are no errors.

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`	Universal Transverse Mercator suggested ellipsoids

utmgeoid

Purpose

Recommend ellipsoids for Universal Transverse Mercator projection zone

Syntax

`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 ellipsoid string used by the `almanac` function.

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

`utmzone` Universal Transverse Mercator projection zones

Purpose

Regular data grid from vector data

Syntax

`[map,refvec] = vec2mtx(lat,lon,scale)` creates a regular data grid from vector data. The returned map has values of one corresponding to the vector data, and zeros otherwise. `lat` and `lon` are vectors of equal length containing geographic locations in units of degrees. The scale factor represents the number of grid entries per single unit of latitude and longitude (e.g., 10 entries per degree, 100 entries per degree). The `scale` input must be scalar.

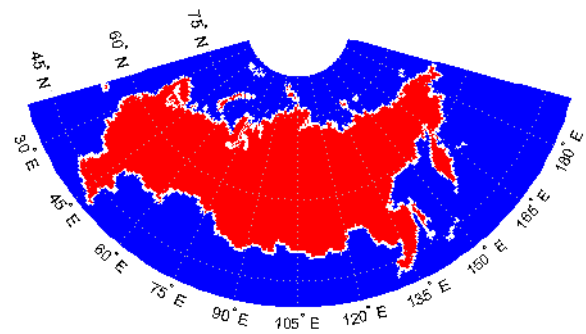
`[map,refvec] = vec2mtx(lat,lon,scale,latlim,lonlim)` uses the two-element vector latitude and longitude limits to define the extent of the map. If omitted, the limits are computed automatically.

`[map,refvec] = vec2mtx(lat,lon,map1,refvec1)` uses the provided `map` and `refvec` to define the extent of the map. If omitted, the limits are computed automatically.

`[...] = vec2mtx(...,'filled')` also fills the area outside the border. The interior then has values of 0, the border 1, and the exterior 2. `lat` and `lon` should contain data that closes on itself.

Examples

```
[lat,long] = extractm(worldlo('POpatch'),'Russia');
[map,refvec] = vec2mtx(lat,long,2,'filled');
[latlim,lonlim] = limitm(map,refvec);
worldmap(latlim,lonlim,'none'); framem off
meshm(map,refvec)
colormap(flag(3))
```



Limitations

The `vec2mtx` function does not fill properly if the vector data extends beyond a pole.

See Also

`country2mtx`	Construct a data grid for a country in the `worldlo` database
`ltln2val`	Return map code value associated with positions
`imbedm`	Encode data points into a regular data grid
`encodem`	Fill in indexed maps with specified seeds
`interpm`	Interpolate vector data to a specified data separation

Purpose

Transform vector azimuths to a projection space angle

Syntax

`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.

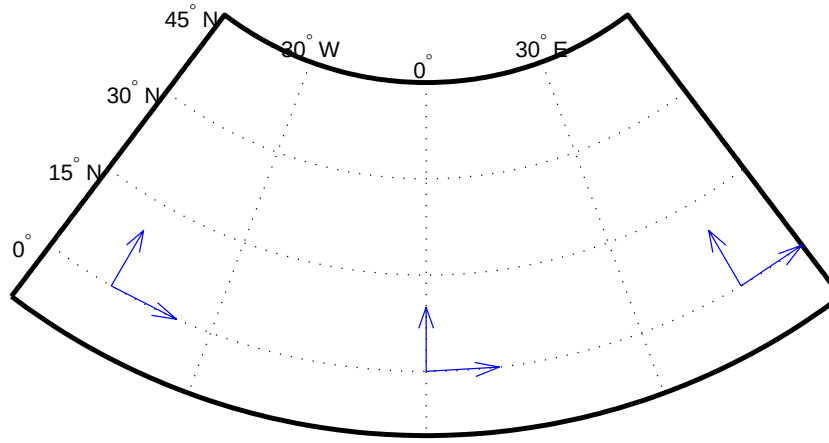
Background

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.

Remarks

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`	Transform azimuths from a projection space angle
`mfwdtran`	Process the map forward transformations
`minvtran`	Process the map inverse transformations
`defaultm`	Initialize the default map data structure

viewshed

Purpose

Compute areas visible from a point on a digital elevation map

Syntax

`[vismap,visrefvec] = viewshed(map,refvec,lat1,lon1)` computes areas visible from a point on a digital elevation map. The elevations are provided as 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 resulting `vismap` contains ones at the surface locations visible from the observer location, and zeros where the line of sight is obscured by terrain.

`viewshed(map,refvec,lat1,lon1,oalt)` 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(map,refvec,lat1,lon1,oalt,talt)` 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(map,refvec,lat1,lon1,oalt,talt,oaltopt)` controls whether the observer is at a relative or absolute altitude. If the observer altitude option is `'AGL'`, the observer altitude `oalt` is in meters above ground level. If `oaltopt` is `'MSL'`, `oalt` is interpreted as altitude above zero, or mean sea level. If omitted, `'AGL'` is assumed.

`viewshed(map,refvec,lat1,lon1,oalt,talt,oaltopt,taltopt)` controls whether the target points are at a relative or absolute altitude. If the target altitude option is `'AGL'`, the target altitude `talt` is in meters above ground level. If `taltopt` is `'MSL'`, `talt` is interpreted as altitude above zero, or mean sea level. If omitted, `'AGL'` is assumed.

`viewshed(map,refvec,lat1,lon1,oalt,talt,oaltopt,taltopt,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.

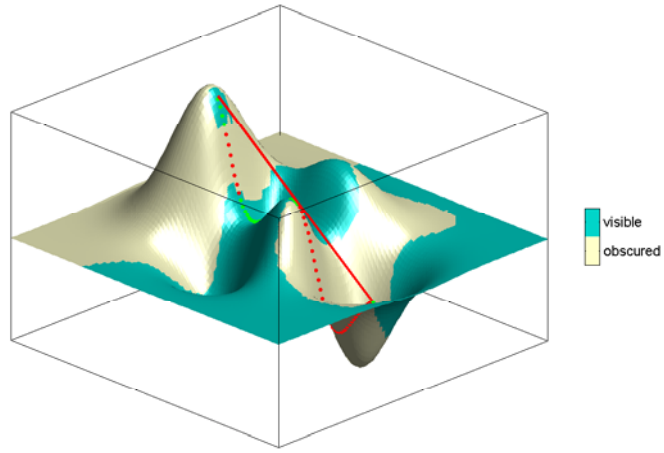
`viewshed(map,refvec,lat1,lon1,oalt,talt,oaltopt,taltopt,actualradius,effectiveradius)` assumes a larger radius for propagation of the line of sight. This can account for the curvature of the signal path due to refraction in the atmosphere. For example, radio propagation in the atmosphere is commonly treated as straight line propagation on a sphere with 4/3rds the

radius of the Earth. In that case the last two arguments would be `R` and `4/3*R_e`, where `R` 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.

Example

Compute visibility for a point on the peaks map. Add the detailed information for the line of sight calculation between two points from `los2`.

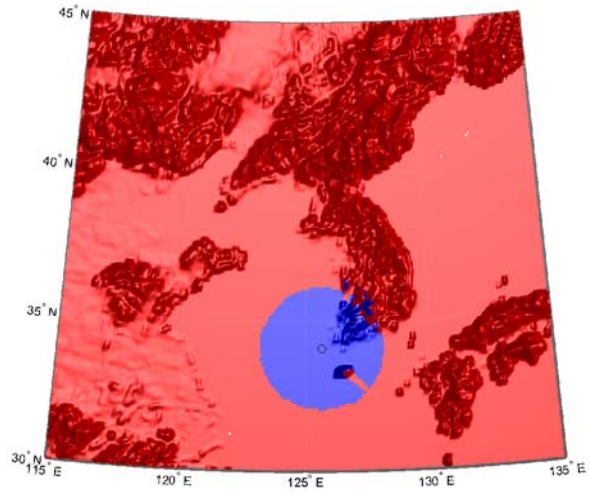
```
map = 500*peaks(100);
refvec = [ 1000 0 0];
[lat1,lon1,lat2,lon2]=deal(-0.027,0.05,-0.093,0.042);

[vismap,vismapleg] = viewshed(map,refvec,lat1,lon1,100);
[vis,visprofile,dist,z,lattrk,lontrk] =
los2(map,refvec,lat1,lon1,lat2,lon2,100);

axesm('globe','geoid',almanac('earth','sphere','meters'))
meshm(vismap,vismapleg,size(map),map); axis tight
camposm(-10,-10,1e6); camupm(0,0)
colormap(flipud(summer(2))); brighten(0.75);
shading interp; camlight
h = lcolorbar({'obscured','visible'});
set(h,'Position',[.875 .45 .02 .1])

plot3m(lattrk([1;end]),lontrk([1; end]),z([1; end])+[100;
0],'r','linewidth',2)
plotm(lattrk(~visprofile),lontrk(~visprofile),z(~visprofile),'r.
','markersize',10)
plotm(lattrk(visprofile),lontrk(visprofile),z(visprofile),'g.','
markersize',10)
```



Compute the surface areas visible by radar from an aircraft 3000 meters above the Yellow Sea. Assume that radio wave propagation in the atmosphere can be modeled as straight lines on a 4/3rds radius Earth. Display the visible areas as blue and the obscured areas as red. Drape the visibility colors on an elevation map, and use lighting to bring out the surface topography. The aircraft's radar can see out a certain radius on the surface of the ocean, but 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.

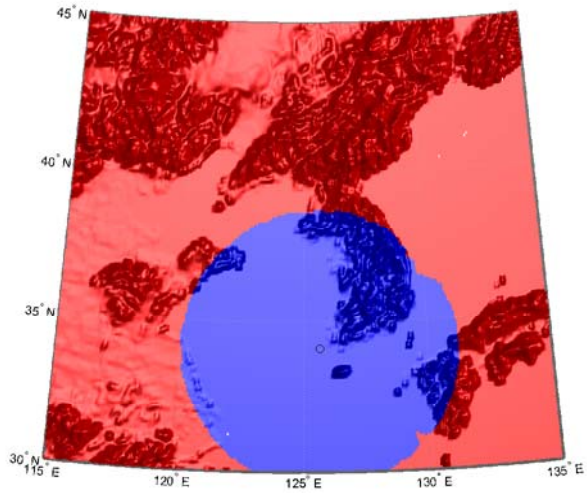
```
load korea
map(map<0) = -1;
figure
worldmap(map,refvec,'ldem3d')
lat = 34.0931; lon = 125.6578; altobs = 3000; alttarg = 0;
plotm(lat,lon,'wo')
Re = almanac('earth','radius','m');
[vmap,vmapl] =
viewshed(map,refvec,lat,lon,altobs,alttarg,'MSL','AGL',Re,4/3*Re
);
clmo surface
meshm(vmap,vmapl,size(map),map)
caxis auto; colormap([1 0 0; 0 0 1])
```

```
lighting phong; material metal
hidem(gca)
```



Over what area can the radar plane flying at an altitude of 3000 meters have line-of-sight to other aircraft flying at 5000 meters? Now the area is much larger. Some edges of the area are reduced by shadowing from Jeju-Do and the mountains on the Korean peninsula.

```
[vmap,vmapl] =
viewshed(map,refvec,lat,lon,3000,5000,'MSL','MSL',Re,4/3*Re);
clmo surface
meshm(vmap,vmapl,size(map),map)
material metal
lighting phong
```



See Also

`los2`	Line of sight visibility between two points in terrain

Purpose

Transform azimuths from a projection space angle

Syntax

`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 Greenwich 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] = vfwdtran(...)` also returns the vector length in the Greenwich coordinate system. A value of 1 indicates no scale distortion for that angle.

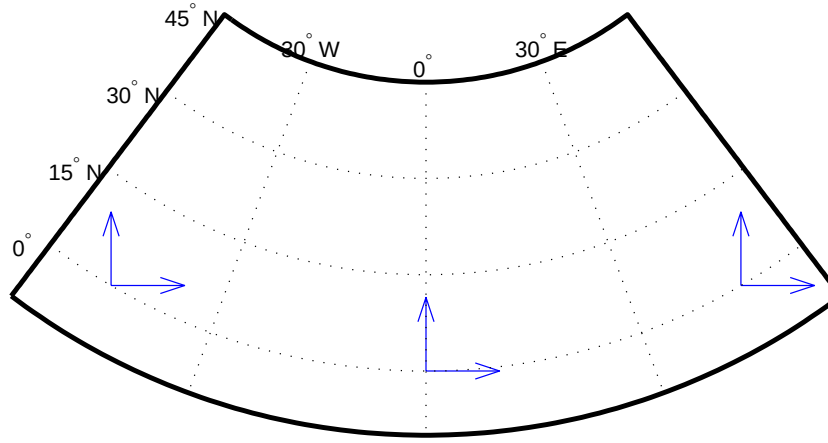
Background

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 conics, azimuthals, and other projections. This function computes the geographic azimuth for angles in the projected space.

Examples

Sample calculations:

```
axesm('eqdconic','maplatlim',[-10 45],'maplonlim',[-55 55])
gridm; framem; mlabel; plabel

[x,y] = mfwdtran([0 0 0],[-45 0 45]);
quiver(x,y,[ .2 .2 .2],[0 0 0],0)
quiver(x,y,[0 0 0],[ .2 .2 .2],0)
```




```
vinvtran(x,y,[ 0 0 0])
ans =
       57.345       90.338       124.98

vinvtran(x,y,[ 90 90 90])
ans =
       331.99            0       28.008
```

Limitations

This transformation is limited to the region specified by the frame limits in the current map definition.

Remarks

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`	Transform vector azimuths to a projection space angle
`mfwdtran`	Process the map forward transformations
`minvtran`	Process the map inverse transformations
`defaultm`	Initialize the default map data structure

vmap0data

Purpose

Read selected data from the Vector Map Level 0

Syntax

`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 string. A list of valid codes is displayed when an invalid code, such as `'?'`, is entered. *`topolevel`* defines the type of data returned. It is a string containing `'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 geographic data 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 National Imagery and Mapping Agency's (NIMA) Operational Navigation Chart (ONC) series. 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. (From the NIMA Metadata referenced below.)

Remarks

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.

The VMAP0 is available from the following sources:

USGS Open-File Section
Box 25286
Denver, CO 80225
USA
Tel: (303) 236-7476

Information on the VMAP0 data can be found at

```
<http://mapping.usgs.gov/esic/cdrom/cdlist.html>
```

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`*.

```
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 other examples:

```
[TRtext,TRline] = vmap0data(devicename,'SASAUS',...
     [-48 -34],[164 180],'trans',{'text','line'});

[BNDpatch,BNDline,BNDpoint,BNDtext] = vmap0data(devicename,...
     'EURNASIA',-48 ,164,'bnd',{'all'});
```

See Also

`vmap0read`	Read a VMAP0 file
`vmap0rhead`	Read a VMAP0 file header
`displaym`	Project data contained in a map structure
`geoshow`	Project data contained in a map structure

`extractm`	Extract vector data from a map structure
`mlayers`	GUI for manipulating map layers

vmap0read

Purpose

Read a Vector Map Level 0 file

Syntax

`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 filename.

`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 a string describing 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.

Remarks

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.

Examples

The following examples use the UNIX directory system and file separators for the pathname:

```
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
```

See Also

`vmap0data`	Read selected data from the VMAP0
`vmap0rhead`	Read a VMAP0 file header

Purpose

Read Vector Map Level 0 file headers

Syntax

`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 *`filename`*.

`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 string containing the VMAP0 header.

Background

The Vector Map Level 0 (VMAP0) uses header strings in most files to document the contents and format of that file. This function reads the header string and displays a formatted version in the Command Window, or returns it as a string.

Remarks

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.

Examples

The following example uses UNIX file separators and pathname:

```
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,-,
```

See Also

`vmap0data`	Read selected data from the VMAP0
`vmap0read`	Read a VMAP0 file

Purpose

Vector Map Level 0 (VMap0) user interface

Activation

`vmap0ui` is a graphical user interface to read Vector Map Level 0 CD-ROMs. The VMAP0 is the most detailed global database of vector map data available to the public. `vmap0ui` attempts to automatically detect which drive contains a VMAP0 CD-ROM. If `vmap0ui` can't find the CD, use the following calling form.

`vmap0ui(devicename)` or `vmap0ui devicename` uses the specified logical device name for the CD-ROM drive containing the VMAP0 CD-ROM. Under the Macintosh operating system, it would be `'VMAP'`. Under the Windows OS, it could be `'f:'` or `'g:'` or some other letter. Under UNIX, it could be `'\cdrom\'`. Check your computer's documentation for the correct device name.

VMAP0 CD-ROMs are available from

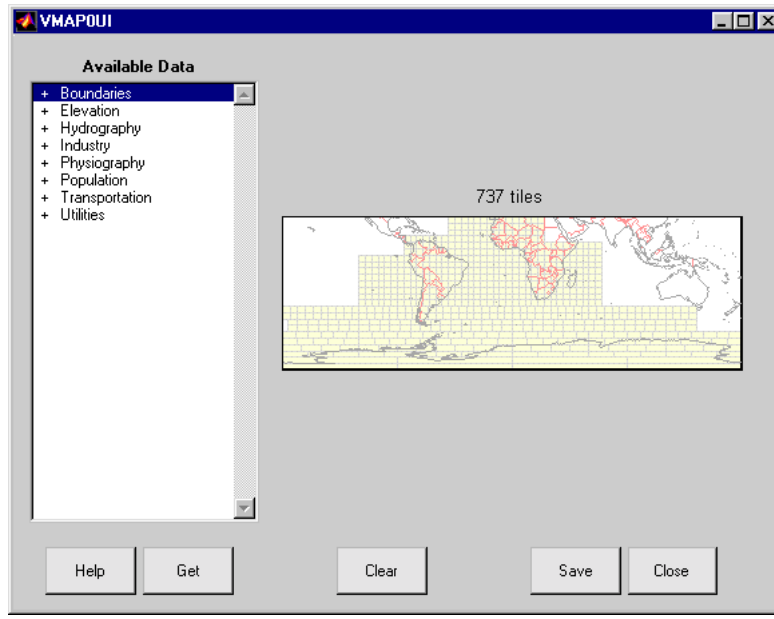
USGS Information Services (Map and Book Sales)
Box 25286
Denver Federal Center
Denver, CO 80225
Telephone: (303) 202-4700
Fax: (303) 202-4693

The price as of early 1998 was $82.50 per four-disk set.

Information on the VMAP0 data can be found at

```
<http://164.214.2.54/mel/metadata/vmap0.meta.html>
```

Controls



The `vmap0ui` screen lets you read data from the Vector Map Level 0 (VMAP0). The VMAP0 is the most detailed 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 filename and location. If you choose to save to the base workspace, you are notified of the variable names that will be overwritten. The results are stored as geographic data structures with variable names based on theme and feature names. 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 geographic data structures, use `rootlayr; displaym(ans)`. To display all of the geographic data structures using the `mlayers` GUI, type `rootlayr; mlayers(ans)`.

The Close Button

The **Close** button closes the `vmap0ui` panel.

See Also

`vmap0datam`, `mlayers`, `displaym`, `extractm`

Purpose

Wrap longitudes to values west of a meridian

Syntax

`ang = westof(angin,meridian)` transforms input angles into equivalent angles west of the specified meridian.

`ang = westof(angin,meridian,`*`units`*`)` uses the units defined by the input string *`units`*. If omitted, default units of `'degrees'` are assumed.

Examples

```
westof(20,0)
ans =
-340

westof(20,-360)
ans =
-700
```

See Also

`eastof`	Wrap longitudes to values east of a meridian
`zero22pi`	Truncate angles into the 0 deg to 360 deg range
`npi2pi`	Truncate angles into the -180 deg to 180 deg range
`smoothlong`	Remove discontinuities in longitude data
`angledim`	Convert angles from one unit system to another

worldfileread

Purpose

Read a worldfile and return a referencing matrix

Syntax

`R = worldfileread(worldfilename)` reads the worldfile data from `worldfilename` and constructs the referencing matrix `R`.

`R` is a 3-by-2 affine transformation matrix that is used in `pix2map` and `map2pix` to transform pixel row and column coordinates to/from map/model coordinates according to `[x y] = [row col 1] * R`.

Example

```
R = worldfileread('concord_ortho_w.tfw');
```

See Also

`getworldfilename`, `makerefmat`, `pix2map`, `map2pix`, `worldfilewrite`

Purpose

Construct a worldfile from a referencing matrix

Syntax

`worldfilewrite(R, worldfilename)` calculates the worldfile entries corresponding to referencing matrix `R` and writes them into the file `worldfilename`.

`R` is a 3-by-2 affine transformation matrix that is used in `pix2map` and `map2pix` to transform pixel row and column coordinates to/from map/model coordinates according to `[x y] = [row col 1] * R`.

Example

```
R = worldfileread('concord_ortho_w.tfw');
worldfilewrite(R,'concord_ortho_w_test.tfw');
```

constructs the referencing matrix `R` from `concord_ortho_w.tfw`, then reconstructs a copy of the worldfile from `R`.

See Also

`getworldfilename`, `pix2map`, `map2pix`, `worldfileread`

Purpose

Return data from the 1:1,000,000 `worldhi` atlas data file

Syntax

`worldhi` types a list of the atlas data variables in the `worldhi` MAT-file to the screen.

`s = worldhi(request)` returns the requested variable. Valid requests are `'POpatch'`, `'POtext'`, `'PPpoint'`, `'PPtext'`, and any country name in the `worldhi` database. You can request multiple countries by using a string matrix or cell array of strings. The result is returned as a geographic data structure. This function can be used as an argument to other commands such as `displaym(worldhi('POtext'))` or `displaym(worldhi('france'))`. The `worldhi` database contains country outlines and text originally compiled at a scale of 1:1,000,000.

`s = worldhi(countryname,minarea)` removes small islands from the output. Only polygon faces with an area greater than `minarea` (in units of square kilometers) are returned.

`s = worldhi(countryname,searchmethod)` controls the method used to match the country name input. Search method `'strmatch'` searches for matches at the beginning of the name, like the MATLAB `strmatch` function. Search method `'findstr'` searches within the name, like the MATLAB `findstr` function. Search method `'exact'` requires an exact match. If omitted, search method `'strmatch'` is assumed. The search is not case sensitive.

`s = worldhi(countryname,searchmethod,minarea)` controls both the minimum area and the string matching method.

`s = worldhi(latlim,lonlim)` returns all countries within a quadrangle. `latlim` and `lonlim` are two-element vectors in units of degrees positive or negative from the prime meridian. Islands or noncontiguous parts of the countries whose bounding boxes fall completely outside the requested region are omitted from the output.

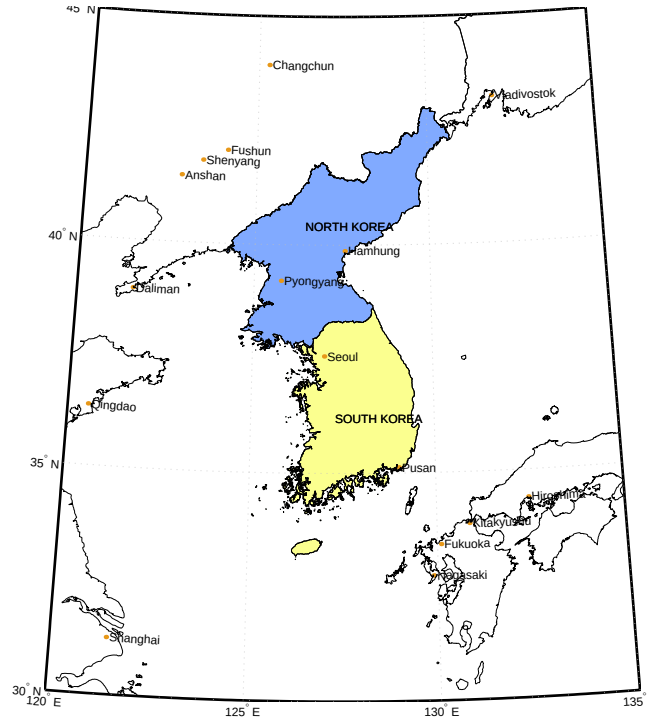
`s = worldhi(latlim,lonlim,minarea)` also removes small islands from the output. Only polygon faces with an area greater than `minarea` (in units of square kilometers) are returned.

Example

Make a base map of the Korean peninsula using lines and display the two Koreas from the `worldhi` database.

```
worldmap korea
```

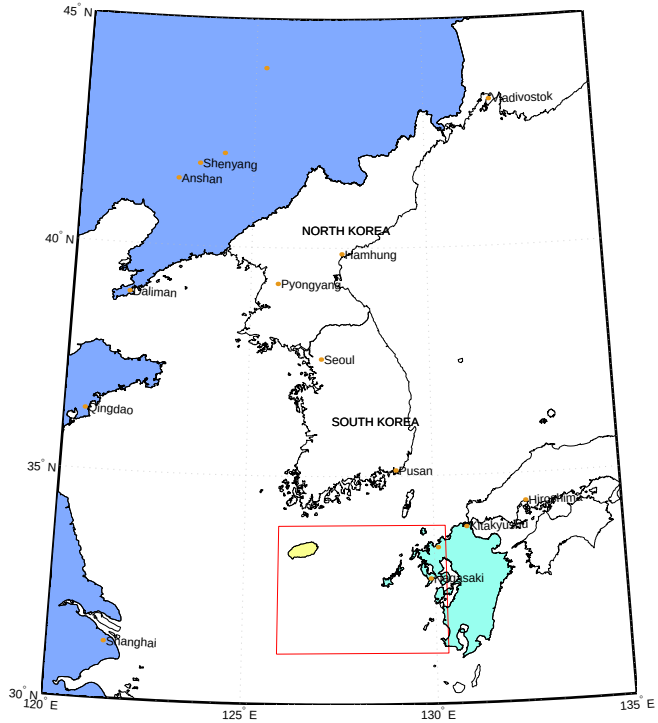
```
s = worldhi('korea');
h = displaym(s);
polcmap
zdatam('allpatch',-1)
```



Delete the Korean patches, and display data from the `worldhi` database that falls within some geographic limits. Mainland China is displayed because its bounding box encompasses the requested quadrangle.

```
delete(h)

latlim = [31.1 33.9]';
lonlim = [125.9 130.3]';
lat = [latlim; flipud(latlim); latlim(1)];
lon = [lonlim(1); lonlim; flipud(lonlim)];
plotm(lat,lon,'r')
```

```
s = worldhi(latlim,lonlim,100);
h = displaym(s);
```



See Also

`worldlo`	Return data from the `worldlo` atlas data file
`usahi`	Return data from the `usahi` atlas data file
`usalo`	Return data from the `usalo` atlas data file
`worldhi.mat`	`worldhi` atlas data MAT-file
`displaym`	Project data in a geographic data structure

Purpose

Return data from the `worldlo` atlas data file

Syntax

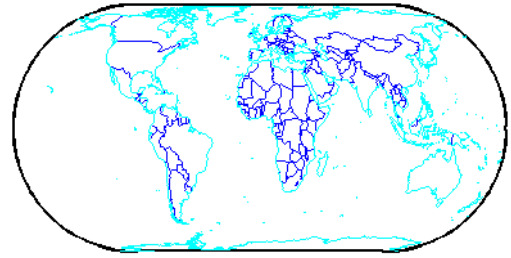
`worldlo` types a list of the atlas data variables in the `worldlo` MAT-file to the screen.

`s = worldlo(request)` returns the requested variable. Valid requests are `'POline'`, `'POpatch'`, `'POtext'`, `'PPpoint'`, `'PPtext'`, `'DNline'`, `'DNpatch'`, `'oceanmask'`, and `'gazette'`. This function can be used as an argument to other commands such as `displaym(worldlo('POline'))`.

Examples

Display the world line data without loading it into the workspace.

```
axesm eckert4
framem
geoshow(worldlo('POline'))
```



See Also

`usahi`	Return data from the `usahi` atlas data file
`usalo`	Return data from the `usalo` atlas data file
`geoshow`	Project data contained in a map structure
`worldlo.mat`	MAT-file containing low-resolution atlas data for the world
`oceanlo.mat`	MAT-file containing an ocean mask for the `worldlo` atlas data

worldmap

Purpose

Map a country or region using the `worldlo` atlas data

Syntax

`worldmap` maps a region or country using the `worldlo` atlas data. The region is selected interactively from a list. The map is created in the current axes.

`worldmap` *`region`* or `worldmap(`*`region`*`)` makes a map of the specified region. Recognized region strings are `'Africa'`, `'Antarctica'`, `'Asia'`, `'Europe'`, `'North America'`, `'North Pole'`, `'Pacific'`, `'South America'`, `'South Pole'`, `'World'`, and the names of the countries in the `worldlo` atlas data. *`region`* can also be a padded string matrix or a cell array of strings containing multiple country names.

`worldmap` *`regiononly`* adds only the specified country to the map. For example, `worldmap italyonly`. If any of the country names in `region` string matrix or cell array of strings end with `'only'`, only the requested countries are displayed.

`worldmap(`*`regiononly`*`,`*`type`*`)` controls the atlas data displayed. Type `'line'` or `'patch'` creates a map with atlas data of those types. If the size of the map permits, point and text annotation of countries and cities is included. Type `'lineonly'` or `'patchonly'` suppresses the point and text annotation. Type `'none'` suppresses all atlas data. If omitted, type `'line'` is assumed.

`worldmap(latlim,lonlim)` and `worldmap(latlim,lonlim,`*`type`*`)` use the supplied geographic limits to define the map. The geographic limits are two-element vectors of the form `[start end]`, in angular units of degrees.

`worldmap(map,refvec)` and `worldmap(map,refvec,`*`type`*`)` use the supplied regular data grid to define the geographic limits. The data grid is displayed using `meshm` unless type is `'none'`, `'lineonly'`, `'patch'`, or `'patchonly'`. Use type `'meshonly'` to display only the data grid. Types `'mesh'` and `'meshonly'` display the data grid as a colored surface in the z = 0 plane with the default colormap. Types `'dem'` and `'demonly'` apply the digital elevation colormap. Types `'dem3d'` and `'dem3donly'` display a three-dimensional surface. Types `'lmesh3d'`, `'lmesh3donly'`, `'ldem3d'`, and `'ldem3donly'` apply lighting effects to the surfaces.

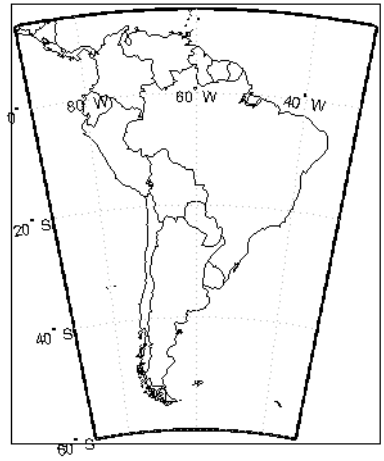
`worldmap('hi',...)` or `worldmap('lo',...)` enforces the use of high- or low-resolution data in the map. If omitted, `worldmap` selects the appropriate data set based on graphic scale. For small-scale maps (1:30,000,000 and under), the `worldlo` database is used. Above 1:30,000,000 (at larger scales), the

`worldhi` database is used. When using the high-resolution data, `worldmap` does not include islands smaller than an appropriate threshold size. Use `worldmap('allhi',...)` to ensure that all islands are included, regardless of size.

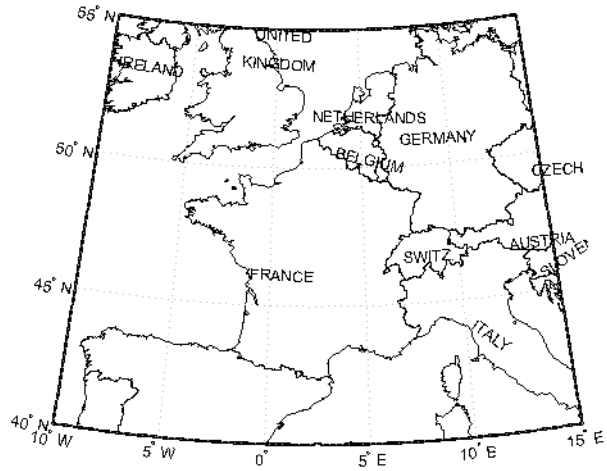
`h = worldmap(...)` returns the handle of the map axes.

Example

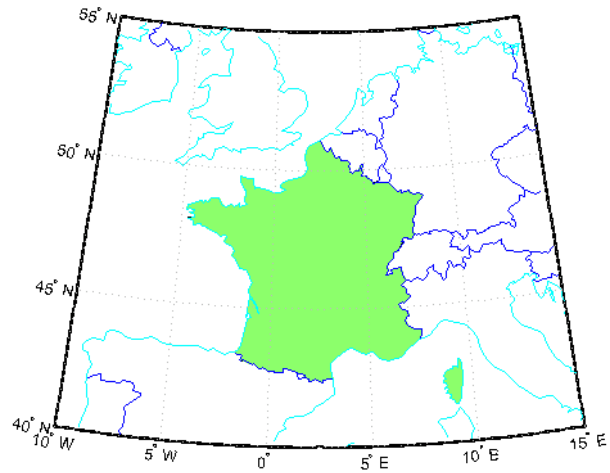
```
worldmap 'south america'
```



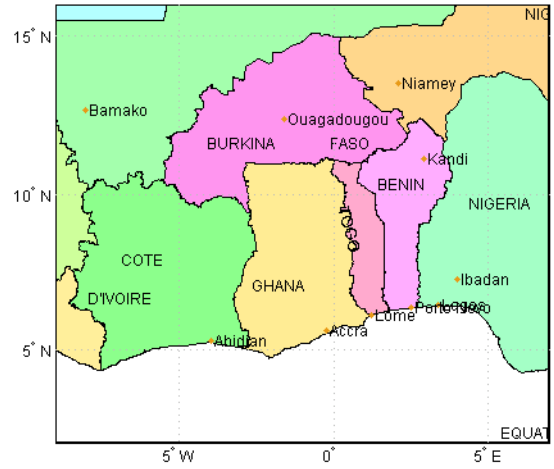
```
figure; worldmap france
```



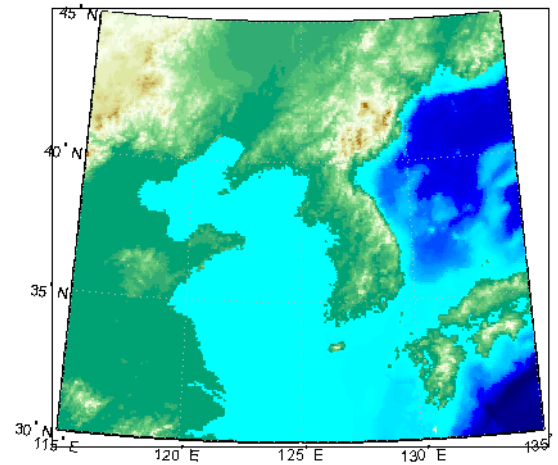

```
figure;
worldmap('lo','franceonly','patch')
geoshow(worldlo('POline'))
hidem(gca)
```



```
figure; worldmap([2 16],[-9 7],'patch');
```




```
figure; load korea; worldmap(map,refvec,'dem');
brighten(.5)
```



Remarks

`worldmap` uses `tightmap` to set 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`.

Use `hidem(gca)` to remove the axes box and `hidem` or `clmo` to remove other unwanted graphic objects. `set(gca,'Color','w')` makes the figure background white.

See Also

`usamap`	Create a map of the United States of America
`gridm`	Toggle and control the display of the map grid
`mlabel`	Meridian labels projected onto a map axes
`plabel`	Parallel labels projected onto a map axes
`framem`	Toggle and control the display of the map frame
`worldlo`	Return data from the `worldlo` atlas data file
`worldhi`	Return data from the `worldhi` atlas data file
`geoshow`	Display vector data in geographic data structure

Purpose

Adjust the *z*-plane of specified graphics objects

Syntax

`zdatam(hndl)` sets the *z*-level of all objects specified by the vector of handles to 0.

`zdatam(object)` sets the *z*-level of all objects identified by the string *`object`* to 0. The string can be any string recognized by the `handlem` function.

`zdatam(hndl,zdata)` sets the *z*-level of all specified objects to the value of a scalar `zdata`, or sets each object at its own level if `zdata` is a vector the same size as `hndl`. When `hndl` is a scalar, `zdata` can also be a matrix with the same size as the object designated by `hndl`.

`zdatam(object,zdata)` sets the *z*-level of the designated object to a scalar `zdata`, or to match a `zdata` matrix the same size as the object.

Description

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`	Handles of graphics objects
`setm`	Set map properties

zero22pi

Purpose

Convert normalized angles to lie between 0 and 2π

Syntax

`anglout = zero22pi(anglin)` *wraps* the input angle `anglin` to lie on the range 0 to 2π (e.g., 450° is renamed 90°).

`anglout = zero22pi(anglin,`*`units`*`)` specifies the angle units with any valid angle units string *`units`*. The default is `'degrees'`.

`anglout = zero22pi(anglin,`*`units`*`,`*`approach`*`)` specifies the approach logic for this wrapping. The *`approach`* string `'exact'` calculates a mathematically precise wrap. `'inward'` and `'outward'` calculate more quickly by shifting the values by an *epsilon* either toward or away from the origin and performing a trigonometric wrap. The trigonometric wrap is inexact, to allow for the fact that different computer math processors might give different (although trigonometrically identical) results (180° or -180°, for example). The offset prevents this.

Examples

```
zero22pi(567.5)
ans =
                     207.5

zero22pi(-567.5)
ans =
                     152.5
```

See Also

`npi2pi` Normalize angles to lie between -π and π

Purpose

Create a matrix map of zeros

Syntax

`map = zerom(latlim,lonlim,scale)` returns a full regular data grid consisting entirely of zeros. 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 number of rows and columns per angle unit is set by the scalar `scale`.

`[map,refvec] = zerom(latlim,lonlim,scale)` returns the three-element referencing vector for the returned `map`.

Examples

```
[map,refvec] = zerom([46,51],[-79,-75],1)
map =
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
refvec =
     1    51   -79
```

See Also

`limitm`	Matrix map limits
`nanm`	Create a data grid of `NaN`s
`onem`	Create a data grid of ones
`sizem`	Rows and columns needed for map
`spzerom`	Create a sparse data grid of zeros

11

Projections Reference

Map Projections — Alphabetical List

The Projections Reference pages are organized in alphabetical order by the name of the map projection. The entries in this chapter contain the following:

Classification Classifies the projection by the geometric or mathematical means of construction.

Syntax Provides the name of the projection M-file used to specify a particular map projection.

Graticule Describes the appearance of meridians, parallels, poles, and map symmetry.

Features Describes the properties of the projection and identifies map distortion.

Parallels Describes the standard parallels of projection.

Remarks Describes the history of the projection and relationships to other projections.

Limitations Describes any restrictions on using the projection.

Cassini Cylindrical Projection

Central Cylindrical Projection

Collignon Projection

Craster Parabolic Projection

Eckert I Projection

Eckert ll Projection

Eckert lll Projection

Eckert IV Projection

Eckert V Projection

Eckert VI Projection

Equal-Area Cylindrical Projection

Equidistant Azimuthal Projection

Equidistant Conic Projection

Equidistant Cylindrical Projection

Fournier Projection

Gall Isographic Projection

Gall Orthographic Projection

Gall Stereographic Projection

Globe

Gnomonic Projection

Goode Homolosine Projection

Hammer Projection

Hatano Asymmetrical Equal-Area Projection

Kavraisky V Projection

Kavraisky VI Projection

Lambert Azimuthal Equal-Area Projection

Lambert Conformal Conic Projection

Lambert Equal-Area Cylindrical Projection

Loximuthal Projection

McBryde-Thomas Flat-Polar Parabolic Projection

McBryde-Thomas Flat-Polar Quartic Projection

McBryde-Thomas Flat-Polar Sinusoidal Projection

Mercator Projection

Miller Cylindrical Projection

Mollweide Projection

Murdoch I Conic Projection

Murdoch III Minimum Error Conic Projection

Orthographic Projection

Plate Carrée Projection

Polyconic Projection

Putnins P5 Projection

Quartic Authalic Projection

Robinson Projection

Sinusoidal Projection

Stereographic Projection

Tissot Modified Sinusoidal Projection

Transverse Mercator Projection

Trystan Edwards Cylindrical Projection

Universal Polar Stereographic Projection

Universal Transverse Mercator Projection

Van der Grinten I Projection

Vertical Perspective Azimuthal Projection

Wagner IV Projection

Werner Projection

Wetch Cylindrical Projection

Wiechel Projection

Winkel I Projection

Aitoff Projection

Classification Modified Azimuthal

Syntax `aitoff`

Graticule Meridians: Central meridian is a straight line half the length of the Equator. Other meridians are complex curves, equally spaced along the Equator, and concave towards the central meridian.

Parallels: Equator is straight. Other parallels are complex curves, equally spaced along the central meridian, and concave towards the nearest pole.

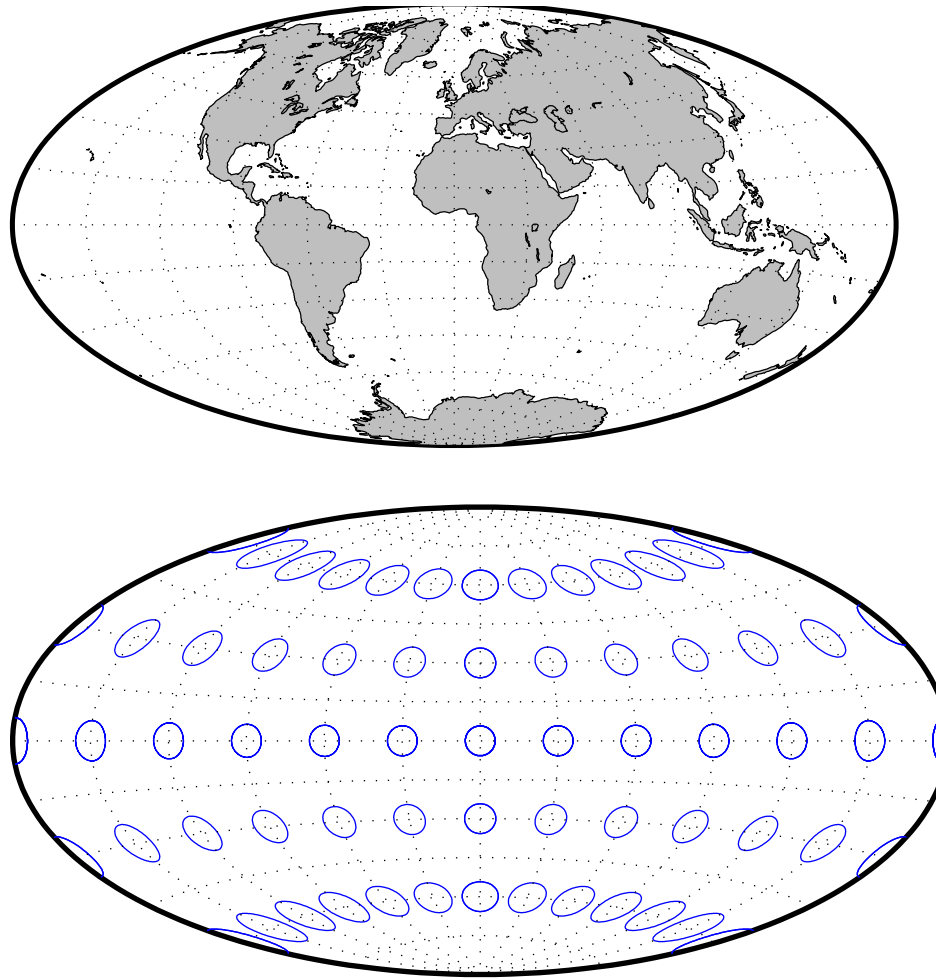
Poles: Points.

Symmetry: About the Equator and central meridian.

Features This projection is neither conformal nor equal area. The only point free of distortion is the center point. Distortion of shape and area are moderate throughout. This projection has less angular distortion on the outer meridians near the poles than pseudoazimuthal projections

Parallels There is no standard parallel for this projection.

Remarks This projection was created by David Aitoff in 1889. It is a modification of the Equidistant Azimuthal projection. The Aitoff projection inspired the similar Hammer projection, which is equal area.



Albers Equal-Area Conic Projection

Classification Conic

Syntax `eqaconic`

Graticule Meridians: Equally spaced straight lines converging to a common point, usually beyond the pole. The angles between the meridians are less than the true angles.

Parallels: Unequally spaced concentric circular arcs centered on the point of convergence. Spacing of parallels decreases away from the central latitudes.

Poles: Normally circular arcs, enclosing the same angle as the displayed parallels.

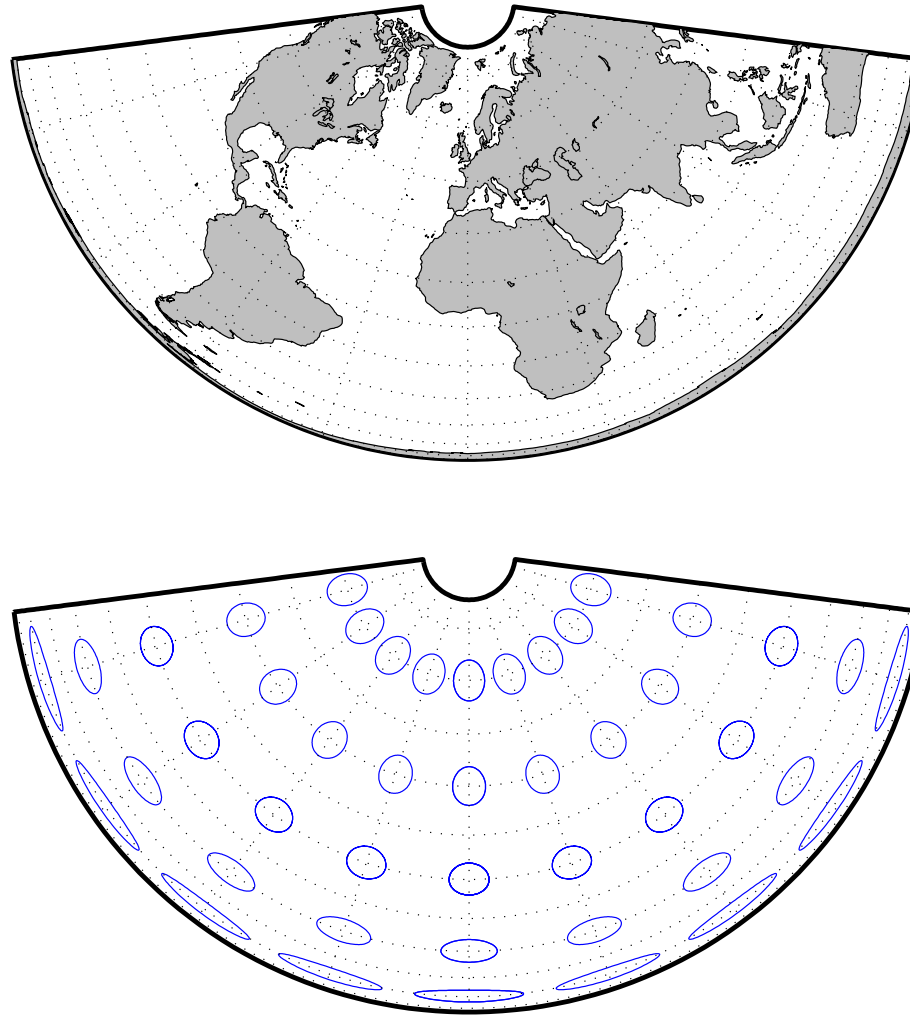
Symmetry: About any meridian.

Features This is an equal-area projection. Scale is true along the one or two selected standard parallels. Scale is constant along any parallel; the scale factor of a meridian at any given point is the reciprocal of that along the parallel to preserve equal-area. This projection is free of distortion along the standard parallels. Distortion is constant along any other parallel. This projection is neither conformal nor equidistant.

Parallels The cone of projection has interesting limiting forms. If a pole is selected as a single standard parallel, the cone is a plane and a Lambert Azimuthal Equal-Area projection results. If two parallels are chosen, not symmetric about the Equator, then a Lambert Equal-Area Conic projection results. If a pole is selected as one of the standard parallels, then the projected pole is a point, otherwise the projected pole is an arc. If the Equator is chosen as a single parallel, the cone becomes a cylinder and a Lambert Equal-Area Cylindrical projection is the result. Finally, if two parallels equidistant from the Equator are chosen as the standard parallels, a Behrmann or other equal-area cylindrical projection is the result. Suggested parallels for maps of the conterminous U.S. are [29.5 45.5]. The default parallels are [15 75].

Remarks This projection was presented by Heinrich Christian Albers in 1805.

Limitations Longitude data greater than 135° east or west of the central meridian is trimmed.



Apianus II Projection

Classification Pseudocylindrical

Syntax `apianus`

Graticule Meridians: Equally spaced elliptical curves converging at the poles.

Parallels: Equally spaced straight lines.

Poles: Points.

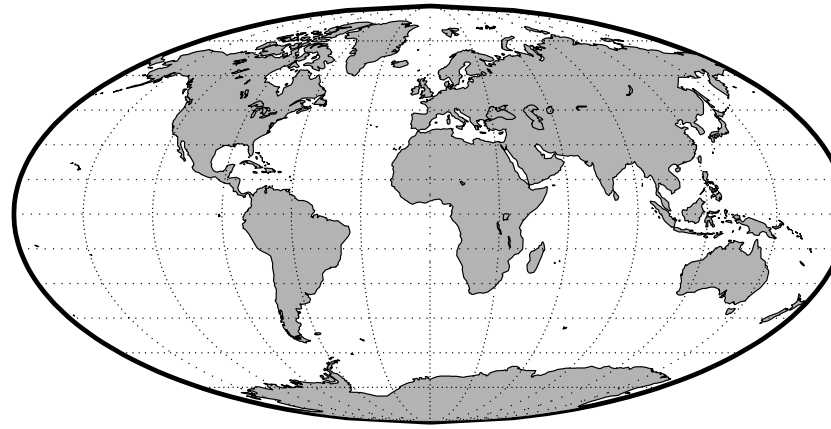
Symmetry: About the Equator and central meridian.

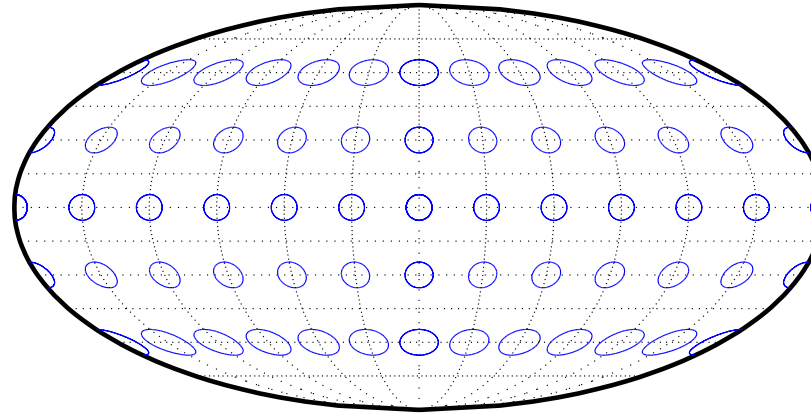
Features Scale is constant along any parallel or pair of parallels equidistant from the Equator, as well as along the central meridian. The Equator is free of angular distortion. This projection is not equal-area, equidistant, or conformal.

Parallels There is no standard parallel for this projection.

Remarks This projection was first described in 1524 by Peter Apian (or Bienewitz).

Limitations This projection is available for the spherical geoid only.





Balthasart Cylindrical Projection

Classification Cylindrical

Syntax `balthsrt`

Graticule Meridians: Equally spaced straight parallel lines.

Parallels: Unequally spaced straight parallel lines, perpendicular to the meridians. Spacing is closest near the poles.

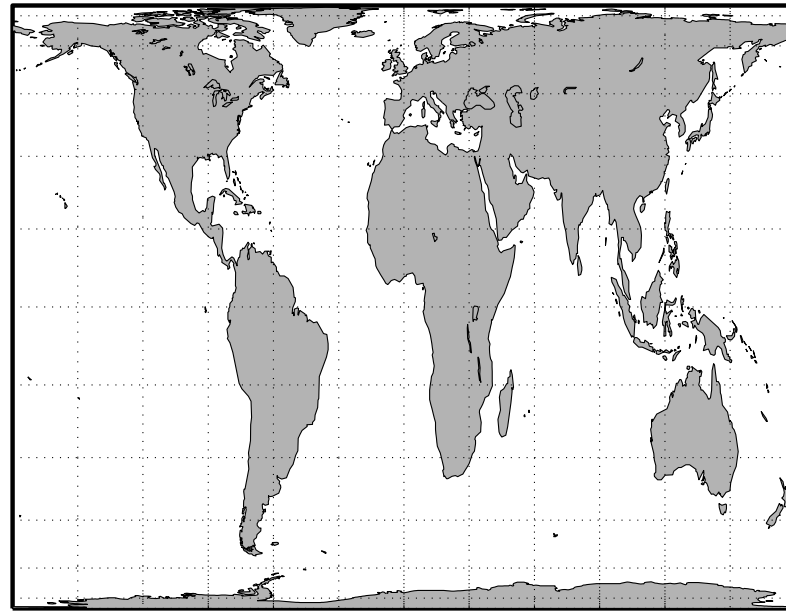
Poles: Straight lines equal in length to the Equator.

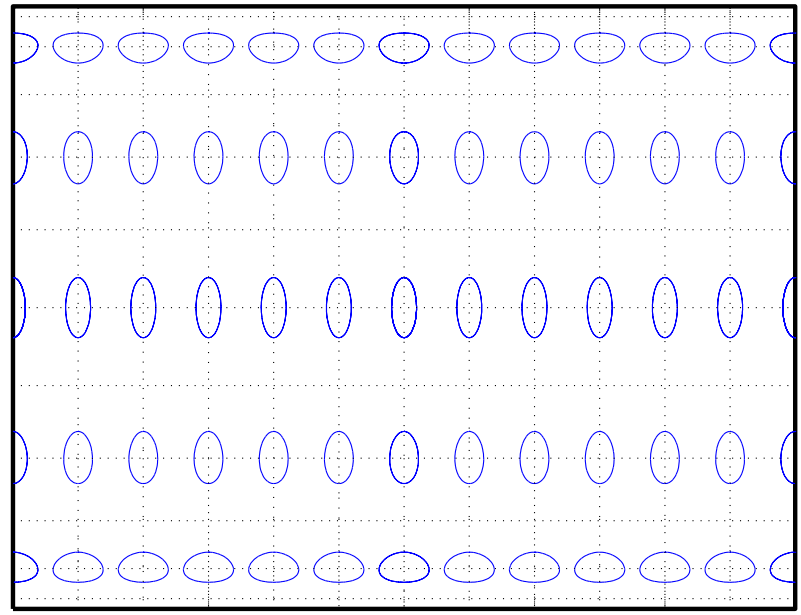
Symmetry: About any meridian or the Equator.

Features This is an orthographic projection onto a cylinder secant at the 50° parallels. It is equal-area, but distortion of shape increases with distance from the standard parallels. Scale is true along the standard parallels and constant between two parallels equidistant from the Equator. This projection is not equidistant.

Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, the standard parallel is by definition fixed at 50°.

Remarks The Balthasart Cylindrical projection was presented in 1935 and is a special form of the Equal-Area Cylindrical projection secant at 50°N and S.





Behrmann Cylindrical Projection

Classification Cylindrical

Syntax `behrmann`

Graticule Meridians: Equally spaced straight parallel lines 0.42 as long as the Equator.

Parallels: Unequally spaced straight parallel lines, perpendicular to the meridians. Spacing is closest near the poles.

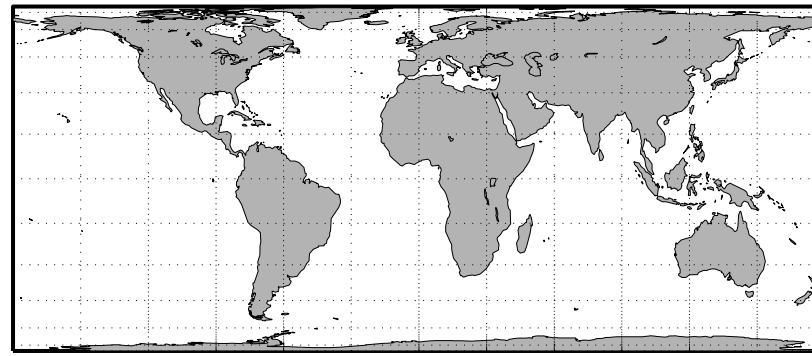
Poles: Straight lines equal in length to the Equator.

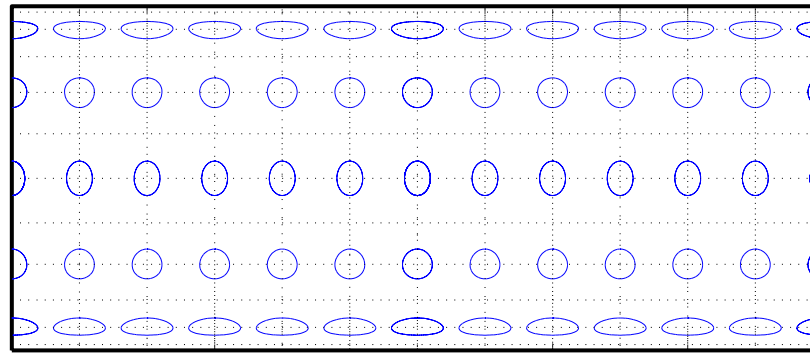
Symmetry: About any meridian or the Equator.

Features This is an orthographic projection onto a cylinder secant at the 30° parallels. It is equal-area, but distortion of shape increases with distance from the standard parallels. Scale is true along the standard parallels and constant between two parallels equidistant from the Equator. This projection is not equidistant.

Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, the standard parallel is by definition fixed at 30°.

Remarks This projection is named for Walter Behrmann, who presented it in 1910 and is a special form of the Equal-Area Cylindrical projection secant at 30°N and S.





Bolshoi Sovietskii Atlas Mira Projection

Classification Cylindrical

Syntax `bsam`

Graticule Meridians: Equally spaced straight parallel lines.

Parallels: Unequally spaced straight parallel lines, perpendicular to the meridians. Spacing increases toward the poles.

Poles: Straight lines equal in length to the Equator.

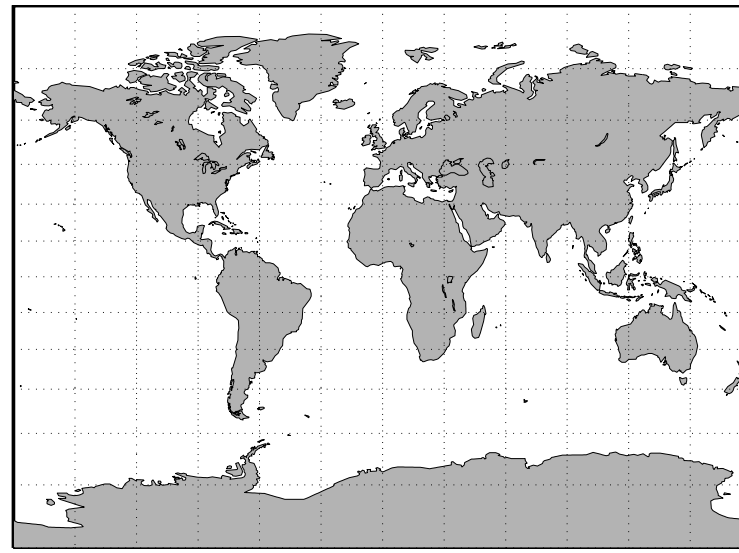
Symmetry: About any meridian or the Equator.

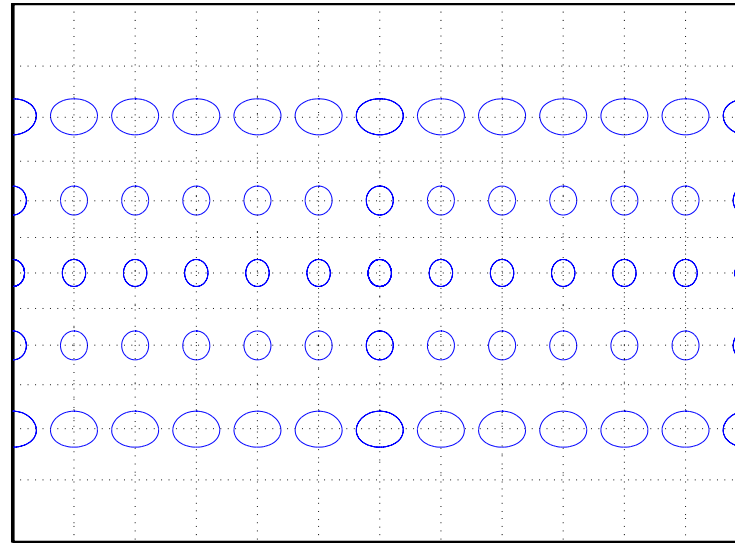
Features This is a perspective projection from a point on the Equator opposite a given meridian onto a cylinder secant at the 30° parallels. It is not equal-area, equidistant, or conformal. Scale is true along the standard parallels and constant between two parallels equidistant from the Equator. There is no distortion along the standard parallels, but it increases moderately away from these parallels, becoming severe at the poles.

Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, the standard parallel is by definition fixed at 30°.

Remarks This projection was first described in 1937, when it was used for maps in the *Bolshoi Sovietskii Atlas Mira* (Great Soviet World Atlas). It is commonly abbreviated as the BSAM projection. It is a special form of the Braun Perspective Cylindrical projection secant at 30°N and S.

Limitations This projection is available for the spherical geoid only.





Bonne Projection

Classification Pseudoconic

Syntax `bonne`

Graticule Central Meridian: A straight line.

Meridians: Complex curves connecting points equally spaced along each parallel and concave toward the central meridian.

Parallels: Concentric circular arcs spaced at true distances along the central meridian.

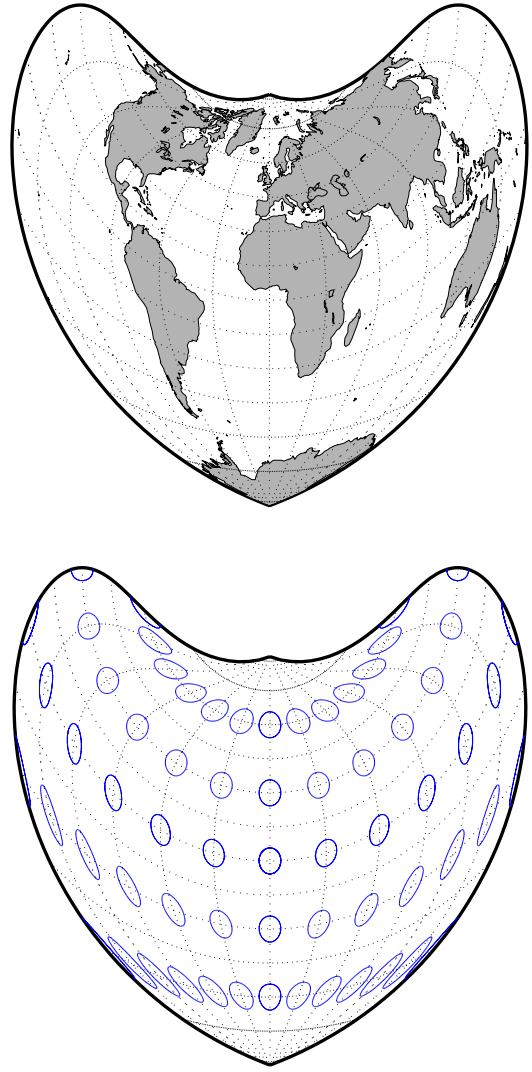
Poles: Points.

Symmetry: About the central meridian.

Features This is an equal-area projection. The curvature of the standard parallel is identical to that on a cone tangent at that latitude. The central meridian and the central parallel are free of distortion. This projection is not conformal.

Parallels This projection has one standard parallel, which is 30°N by default. It has two interesting limiting forms. If a pole is employed as the standard parallel, a Werner projection results; if the Equator is used, a Sinusoidal projection results.

Remarks This projection dates in a rudimentary form back to Claudius Ptolemy (about A.D. 100). It was further developed by Bernardus Sylvanus in 1511. It derives its name from its considerable use by Rigobert Bonne, especially in 1752.



Braun Perspective Cylindrical Projection

Classification Cylindrical

Syntax `braun`

Graticule Meridians: Equally spaced straight parallel lines.

Parallels: Unequally spaced straight parallel lines, perpendicular to the meridians. Spacing increases toward the poles.

Poles: Straight lines equal in length to the Equator.

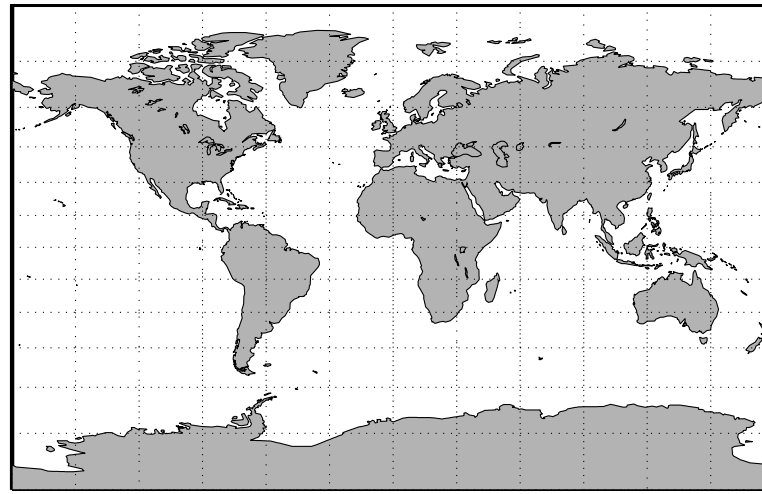
Symmetry: About any meridian or the Equator.

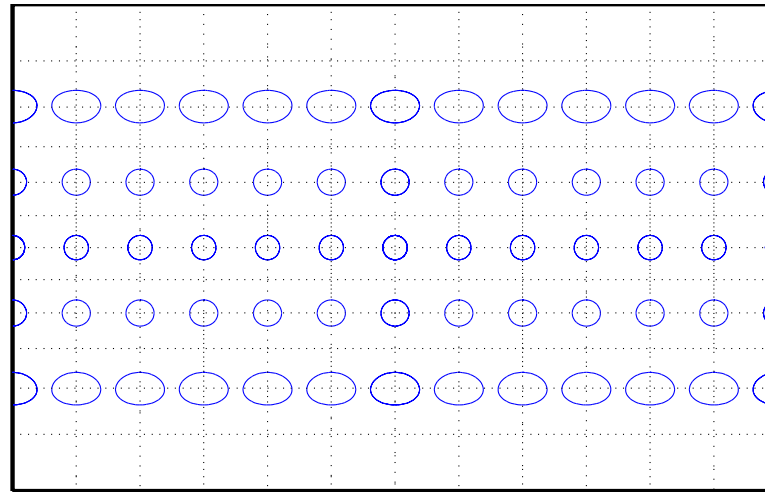
Features This is an perspective projection from a point on the Equator opposite a given meridian onto a cylinder secant at standard parallels. It is not equal-area, equidistant, or conformal. Scale is true along the standard parallels and constant between two parallels equidistant from the Equator. There is no distortion along the standard parallels, but it increases moderately away from these parallels, becoming severe at the poles.

Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, any latitude may be chosen; the default is arbitrarily set to 0°.

Remarks This projection was first described by Braun in 1867. It is less well known than the specific forms of it called the Gall Stereographic and the *Bolshoi Sovietskii Atlas Mira* projections.

Limitations This projection is available for the spherical geoid only.





Breusing Harmonic Mean Projection

Classification Azimuthal

Syntax `breusing`

Graticule The graticule described is for the polar aspect.

Meridians: Equally spaced straight lines intersecting at the central pole.

Parallels: Unequally spaced circles centered on the central pole. The opposite hemisphere cannot be shown. Spacing increases (slightly) away from the central pole.

Poles: The central pole is a point, while the opposite pole cannot be shown.

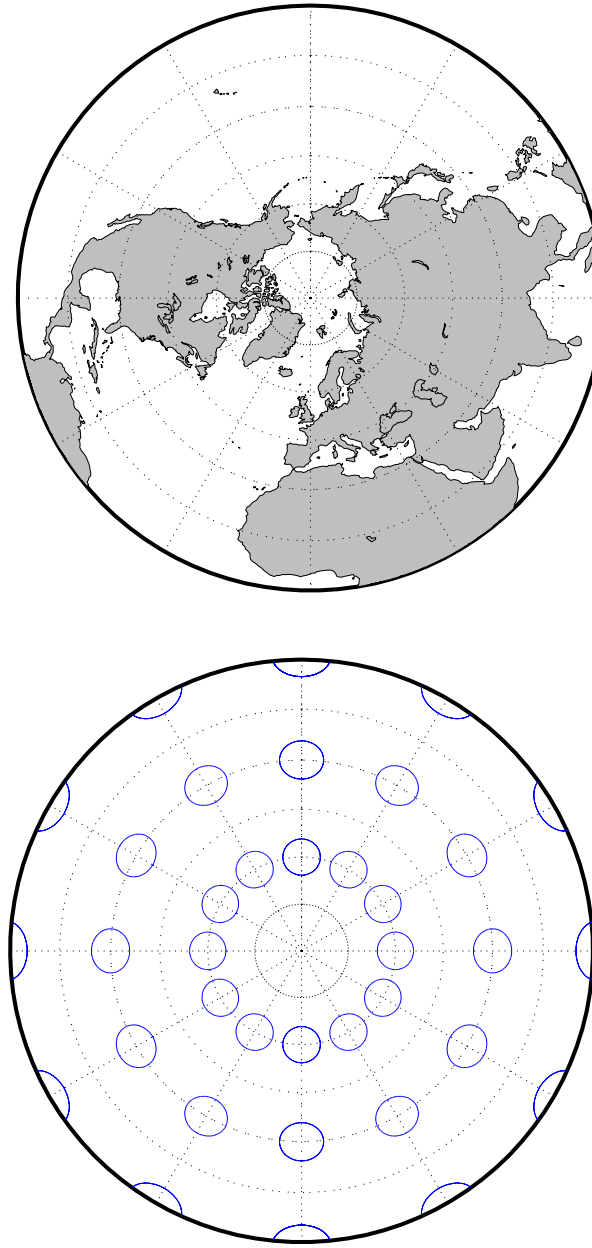
Symmetry: About any meridian.

Features This is a harmonic mean between a Stereographic and Lambert Equal-Area Azimuthal projection. It is not equal-area, equidistant, or conformal. There is no point at which scale is accurate in all directions. The primary feature of this projection is that it is minimum error – distortion is moderate throughout.

Parallels There are no standard parallels for azimuthal projections.

Remarks F. A. Arthur Breusing developed a geometric mean version of this projection in 1892. A. E. Young modified this to the harmonic mean version presented here in 1920. This projection is virtually indistinguishable from the Airy Minimum Error Azimuthal projection, presented by George Airy in 1861.

Limitations This projection is available for the spherical geoid only.



Briesemeister Projection

Classification Modified Azimuthal

Syntax `bries`

Graticule Meridians: Central meridian is straight. Other meridians are complex curves.

Parallels: Complex curves.

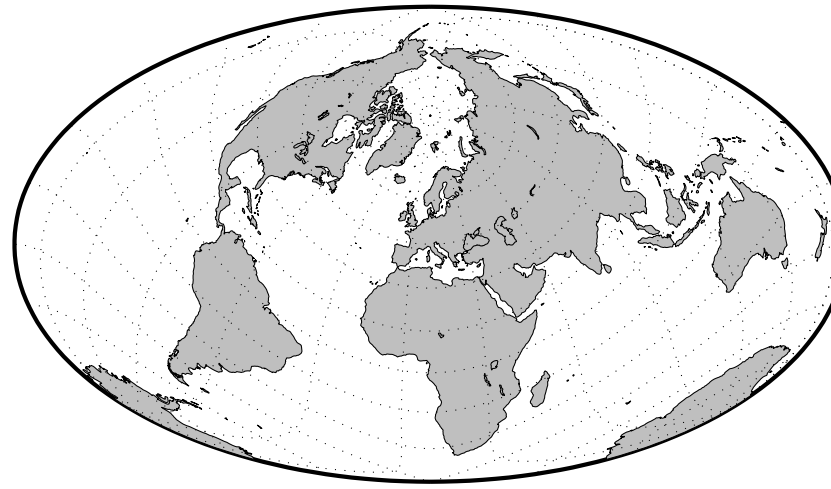
Poles: Points.

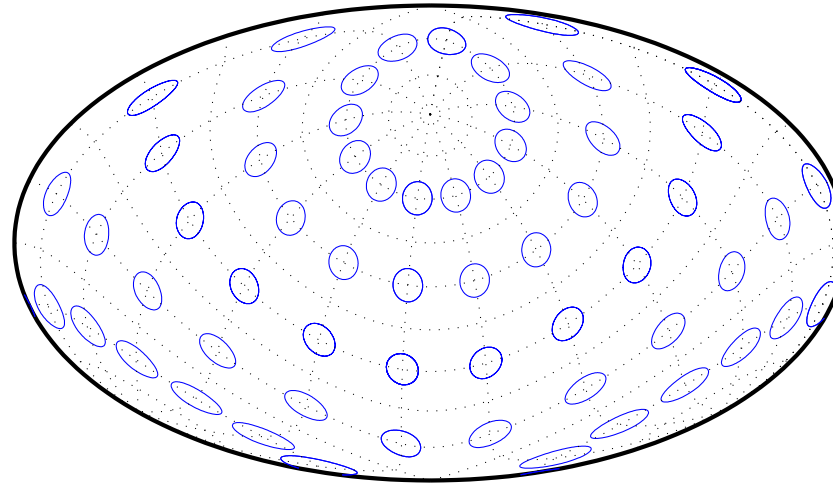
Symmetry: About the central meridian.

Features This equal-area projection groups the continents about the center of the projection. The only point free of distortion is the center point. Distortion of shape and area are moderate throughout.

Parallels There is no standard parallel for this projection.

Remarks This projection was presented by William Briesemeister in 1953. It is an oblique Hammer projection with an axis ratio of 1.75 to 1, instead of 2 to 1.





Cassini Cylindrical Projection

Classification Cylindrical

Syntax `cassini`

Graticule Central Meridian: Straight line (includes meridian opposite the central meridian in one continuous line).

Other Meridians: Straight lines if 90° from central meridian, complex curves concave toward the central meridian otherwise.

Parallels: Complex curves concave toward the nearest pole.

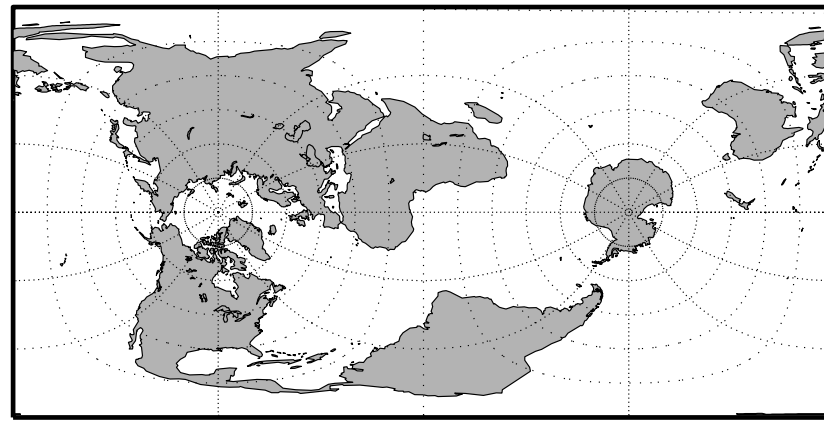
Poles: Points along the central meridian.

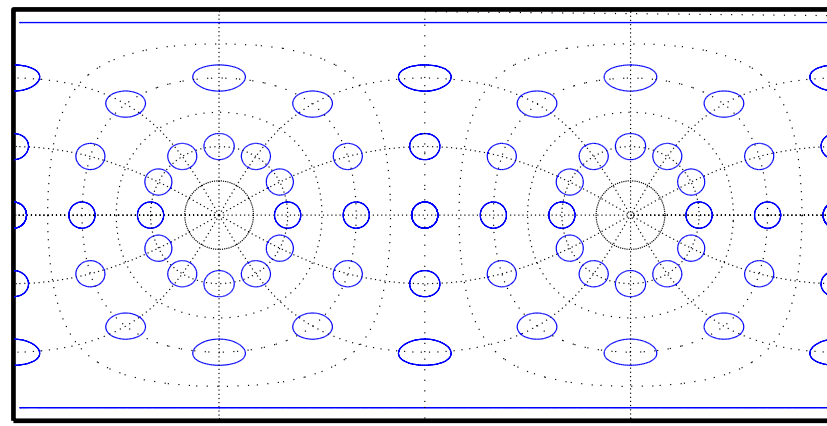
Symmetry: About any straight meridian or the Equator.

Features This is a projection onto a cylinder tangent at the central meridian. Distortion of both shape and area are functions of distance from the central meridian. Scale is true along the central meridian and along any straight line perpendicular to the central meridian (i.e., it is equidistant).

Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, the standard parallel *of the base projection* is by definition fixed at 0°.

Remarks This projection is the transverse aspect of the Plate Carrée projection, developed by César François Cassini de Thury (1714-84). It is still used for the topographic mapping of a few countries.





Central Cylindrical Projection

Classification Cylindrical

Syntax `ccylin`

Graticule Meridians: Equally spaced straight parallel lines.

Parallels: Unequally spaced straight parallel lines, perpendicular to the meridians. Spacing increases toward the poles, more rapidly than that of the Mercator projection.

Poles: Cannot be shown.

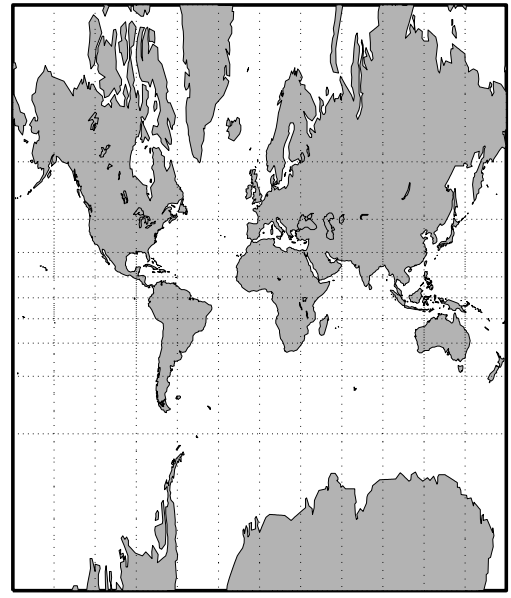
Symmetry: About any meridian or the Equator.

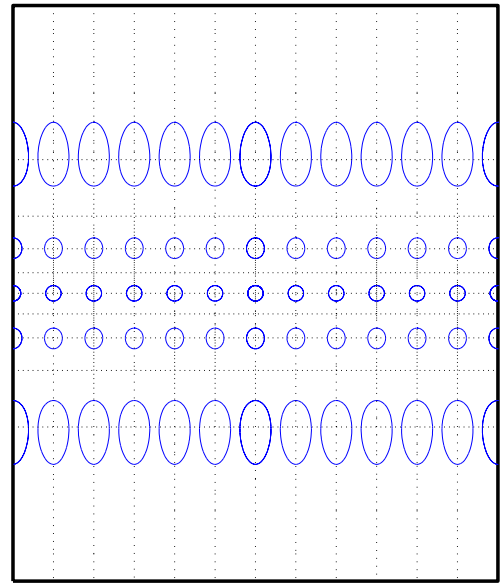
Features This is a perspective projection from the center of the Earth onto a cylinder tangent at the Equator. It is not equal-area, equidistant, or conformal. Scale is true along the Equator and constant between two parallels equidistant from the Equator. Scale becomes infinite at the poles. There is no distortion along the Equator, but it increases rapidly away from the Equator.

Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, the standard parallel is by definition fixed at 0°.

Remarks The origin of this projection is unknown; it has little use beyond the educational aspects of its method of projection and as a comparison to the Mercator projection, which is not perspective. The transverse aspect of the Central Cylindrical is called the Wetch projection.

Limitations This projection is available for the spherical geoid only. Data at latitudes greater than 75° is trimmed to prevent large values from dominating the display.





Collignon Projection

Classification Pseudocylindrical

Syntax `collig`

Graticule Meridians: Equally spaced straight lines converging at the North Pole.

Parallels: Unequally spaced straight parallel lines, farthest apart near the North Pole, closest near the South Pole

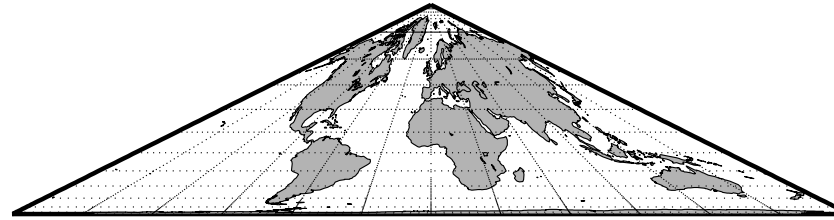
Poles: North Pole is a point, South Pole is a line 1.41 as long as the Equator.

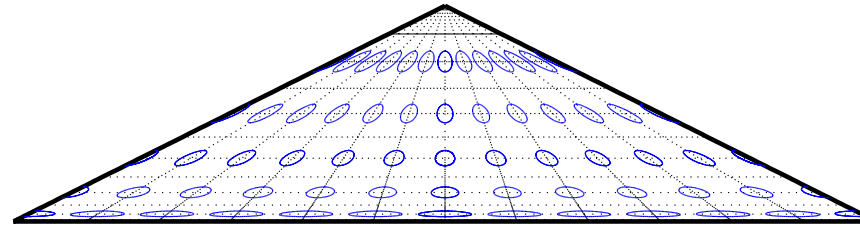
Symmetry: About the central meridian.

Features This is a novelty projection showing a straight-line, equal-area graticule. Scale is true along the 15°51'N parallel, constant along any parallel, and *different* for any pair of parallels. Distortion is severe in many regions, and is only absent at 15°51'N on the central meridian. This projection is not conformal or equidistant.

Parallels This projection has one standard parallel, which is by definition fixed at 15°51'.

Remarks This projection was presented by Édouard Collignon in 1865.





Craster Parabolic Projection

Classification Pseudocylindrical

Syntax `craster`

Graticule Central Meridian: Straight line half as long as the Equator.

Other Meridians: Equally spaced parabolas intersecting at the poles and concave toward the central meridian.

Parallels: Unequally spaced straight parallel lines, perpendicular to the central meridian. Spacing changes very gradually and is greatest near the Equator.

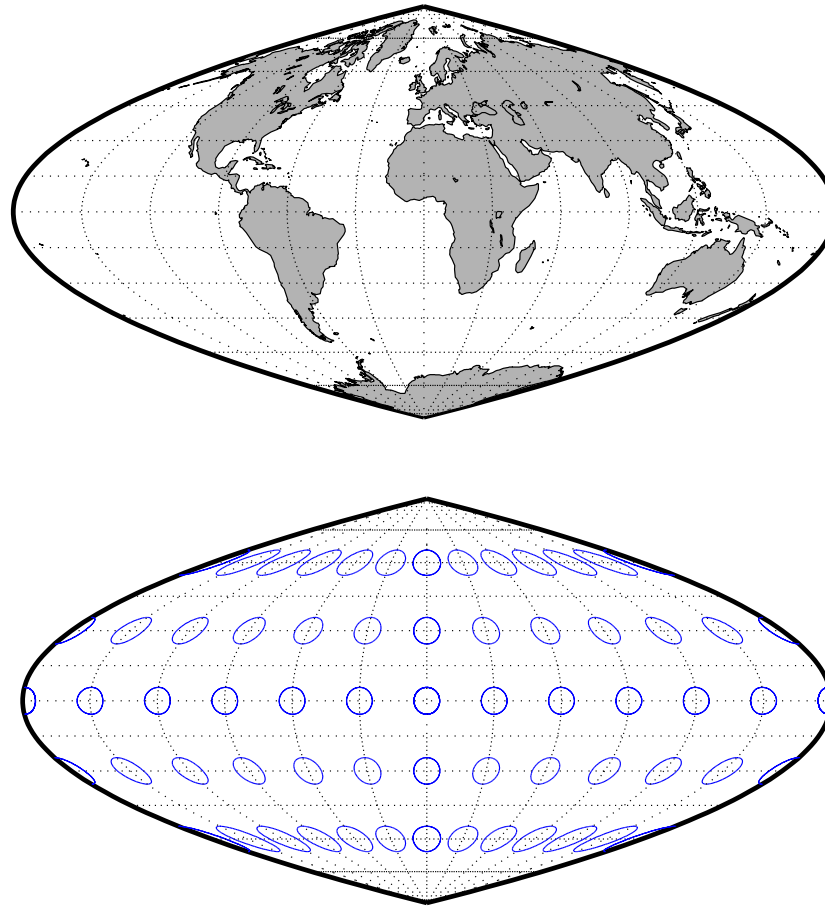
Poles: Points.

Symmetry: About the central meridian or the Equator.

Features This is an equal-area projection. Scale is true along the 36°46' parallels and is constant along any parallel and between any pair of parallels equidistant from the Equator. Distortion is severe near the outer meridians at high latitudes, but less so than the Sinusoidal projection. This projection is free of distortion only at the two points where the central meridian intersects the 36°46' parallels. This projection is not conformal or equidistant.

Parallels For this projection, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. The standard parallel is by definition fixed at 36°46'.

Remarks This projection was developed by John Evelyn Edmund Craster in 1929; it was further developed by Charles H. Deetz and O.S. Adams in 1934. It was presented independently in 1934 by Putnins as his P_4 projection.



Eckert I Projection

Classification Pseudocylindrical

Syntax `eckert1`

Graticule Central Meridian: Straight line half as long as the Equator.

Other Meridians: Equally spaced straight converging lines broken at the Equator.

Parallels: Equally spaced straight parallel lines, perpendicular to the central meridian.

Poles: Lines half as long as the Equator.

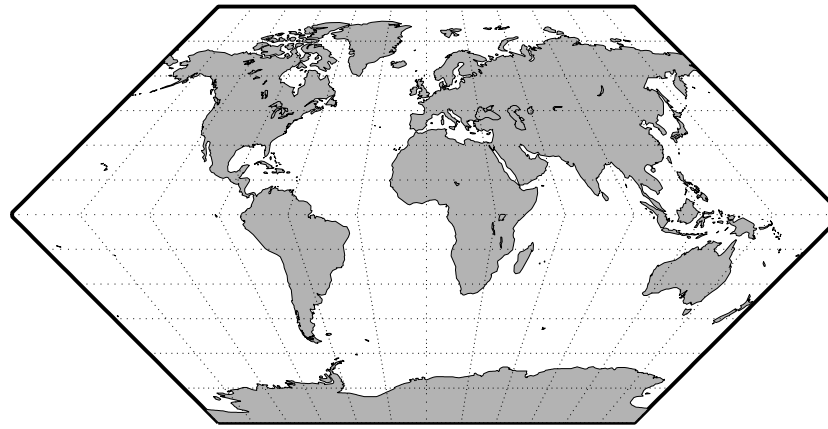
Symmetry: About the central meridian or the Equator.

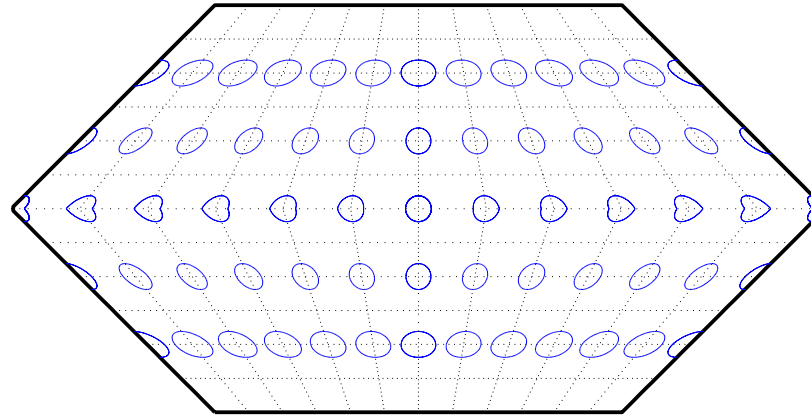
Features Scale is true along the 47°10' parallels and is constant along any parallel, between any pair of parallels equidistant from the Equator, and along any given meridian. It is not free of distortion at any point, and the break at the Equator introduces excessive distortion there; regardless of the appearance here, the Tissot indicatrices are of indeterminate shape along the Equator. This novelty projection is not equal-area or conformal.

Parallels For this projection, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. The standard parallel is by definition fixed at 47°10'.

Remarks This projection was presented by Max Eckert in 1906.

Limitations This projection is available for the spherical geoid only.





Eckert II Projection

Classification Pseudocylindrical

Syntax `eckert2`

Graticule Central Meridian: Straight line half as long as the Equator.

Other Meridians: Equally spaced straight converging lines broken at the Equator.

Parallels: Unequally spaced straight parallel lines, perpendicular to the central meridian. Spacing is widest near the Equator.

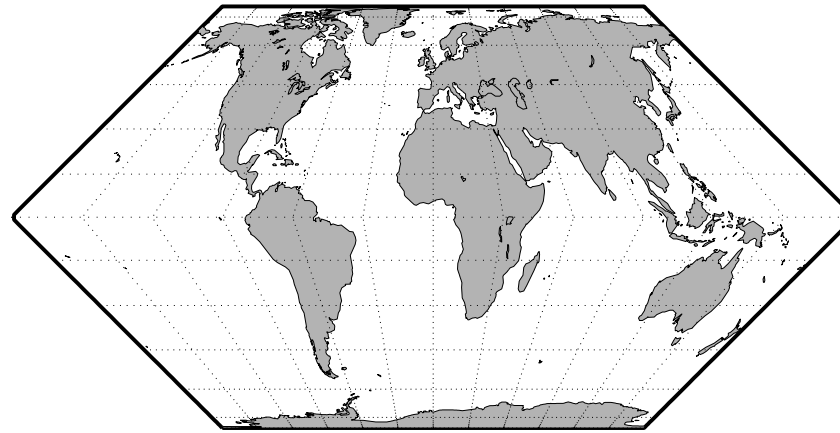
Poles: Lines half as long as the Equator.

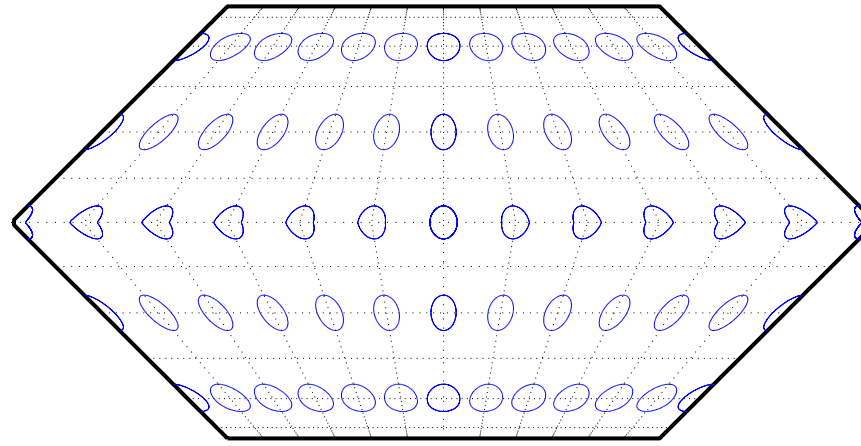
Symmetry: About the central meridian or the Equator.

Features This is an equal-area projection. Scale is true along the 55°10' parallels and is constant along any parallel and between any pair of parallels equidistant from the Equator. It is not free of distortion at any point except at 55°10'N and S along the central meridian; the break at the Equator introduces excessive distortion there. Regardless of the appearance here, the Tissot indicatrices are of indeterminate shape along the Equator. This novelty projection is not conformal or equidistant.

Parallels For this projection, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. The standard parallel is by definition fixed at 55°10'.

Remarks This projection was presented by Max Eckert in 1906.





Eckert III Projection

Classification Pseudocylindrical

Syntax `eckert3`

Graticule Central Meridian: Straight line half as long as the Equator.

Other Meridians: Equally spaced semiellipses concave toward the central meridian. The outer meridians, 180° east and west of the central meridian, are semicircles.

Parallels: Equally spaced straight parallel lines, perpendicular to the central meridian.

Poles: Lines half as long as the Equator.

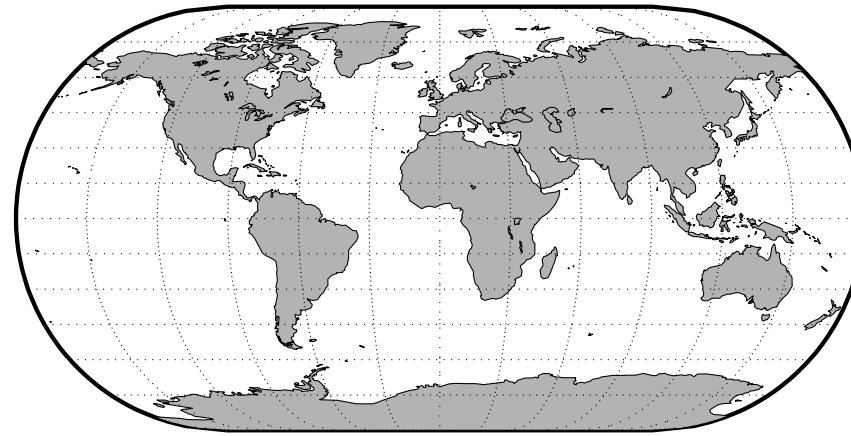
Symmetry: About the central meridian or the Equator.

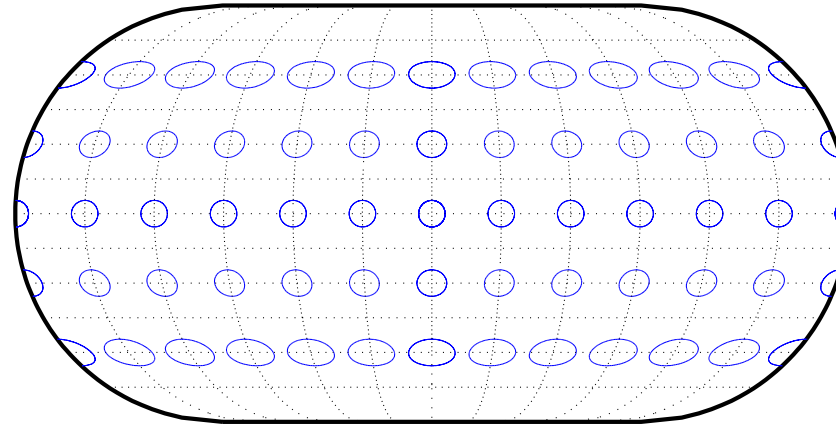
Features Scale is true along the 35°58' parallels and is constant along any parallel and between any pair of parallels equidistant from the Equator. No point is free of all scale distortion, but the Equator is free of angular distortion. This projection is not equal-area, conformal, or equidistant.

Parallels For this projection, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. The standard parallel is by definition fixed at 35°58'.

Remarks This projection was presented by Max Eckert in 1906.

Limitations This projection is available for the spherical geoid only.





Eckert IV Projection

Classification Pseudocylindrical

Syntax `eckert4`

Graticule Central Meridian: Straight line half as long as the Equator.

Other Meridians: Equally spaced semiellipses concave toward the central meridian. The outer meridians, 180° east and west of the central meridian, are semicircles.

Parallels: Unequally spaced straight parallel lines, perpendicular to the central meridian. Spacing is greatest toward the Equator.

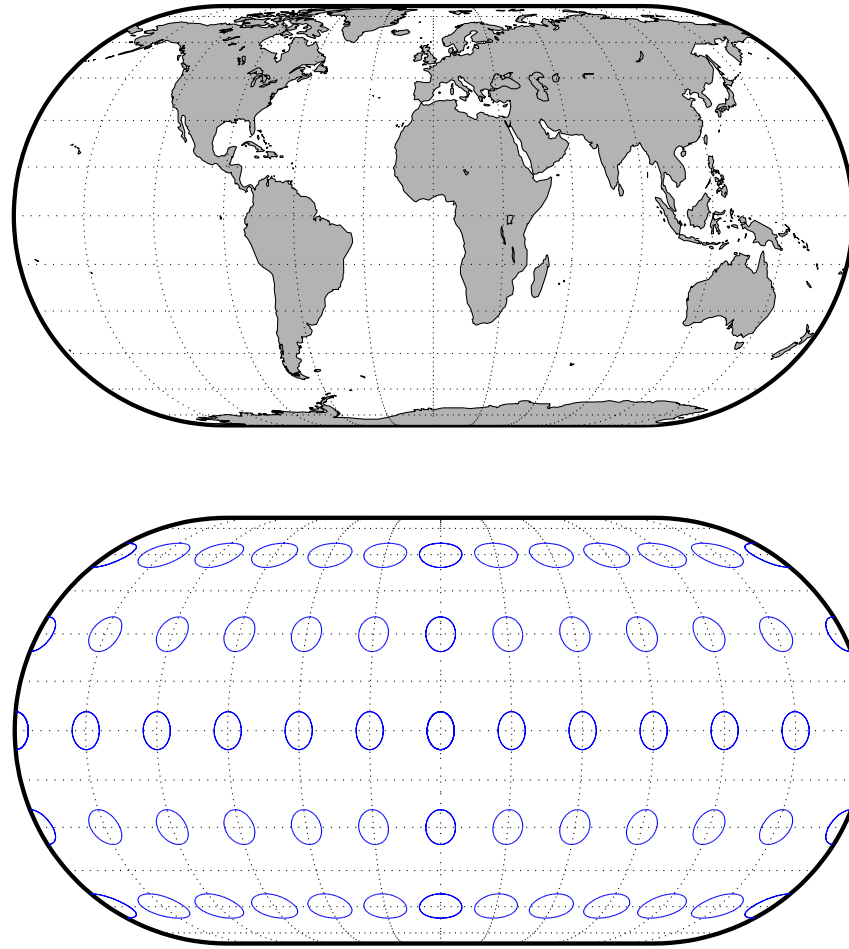
Poles: Lines half as long as the Equator.

Symmetry: About the central meridian or the Equator.

Features This is an equal-area projection. Scale is true along the 40°30' parallels and is constant along any parallel and between any pair of parallels equidistant from the Equator. It is free of distortion only at the two points where the 40°30' parallels intersect the central meridian. This projection is not conformal or equidistant.

Parallels For this projection, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. The standard parallel is by definition fixed at 40°30'.

Remarks This projection was presented by Max Eckert in 1906.



Eckert V Projection

Classification Pseudocylindrical

Syntax `eckert5`

Graticule Central Meridian: Straight line half as long as the Equator.

Other Meridians: Equally spaced sinusoidal curves concave toward the central meridian.

Parallels: Equally spaced straight parallel lines, perpendicular to the central meridian.

Poles: Lines half as long as the Equator.

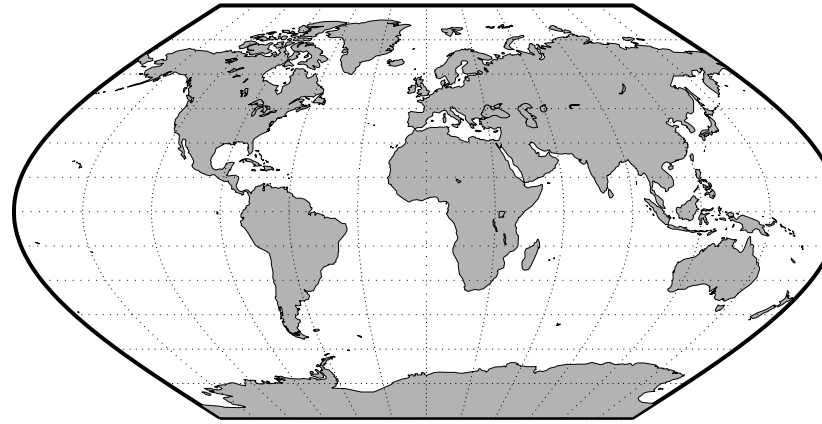
Symmetry: About the central meridian or the Equator.

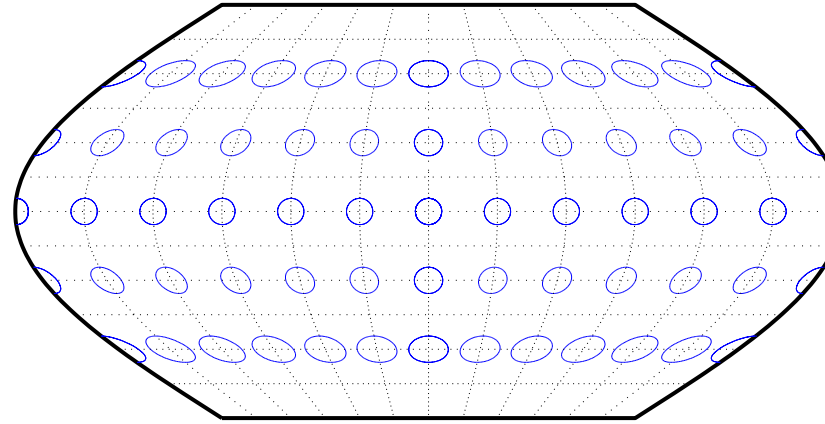
Features This projection is an arithmetic average of the x and y coordinates of the Sinusoidal and Plate Carrée projections. Scale is true along latitudes 37°55'N and S, and is constant along any parallel and between any pair of parallels equidistant from the Equator. There is no point free of all distortion, but the Equator is free of angular distortion. This projection is not equal-area, conformal, or equidistant.

Parallels This projection has one standard parallel, which is by definition fixed at 0°.

Remarks This projection was presented by Max Eckert in 1906.

Limitations This projection is available for the spherical geoid only.





Eckert VI Projection

Classification Pseudocylindrical

Syntax `eckert6`

Graticule Central Meridian: Straight line half as long as the Equator.

Other Meridians: Equally spaced sinusoidal curves concave toward the central meridian.

Parallels: Unequally spaced straight parallel lines, perpendicular to the central meridian. Spacing is greatest toward the Equator.

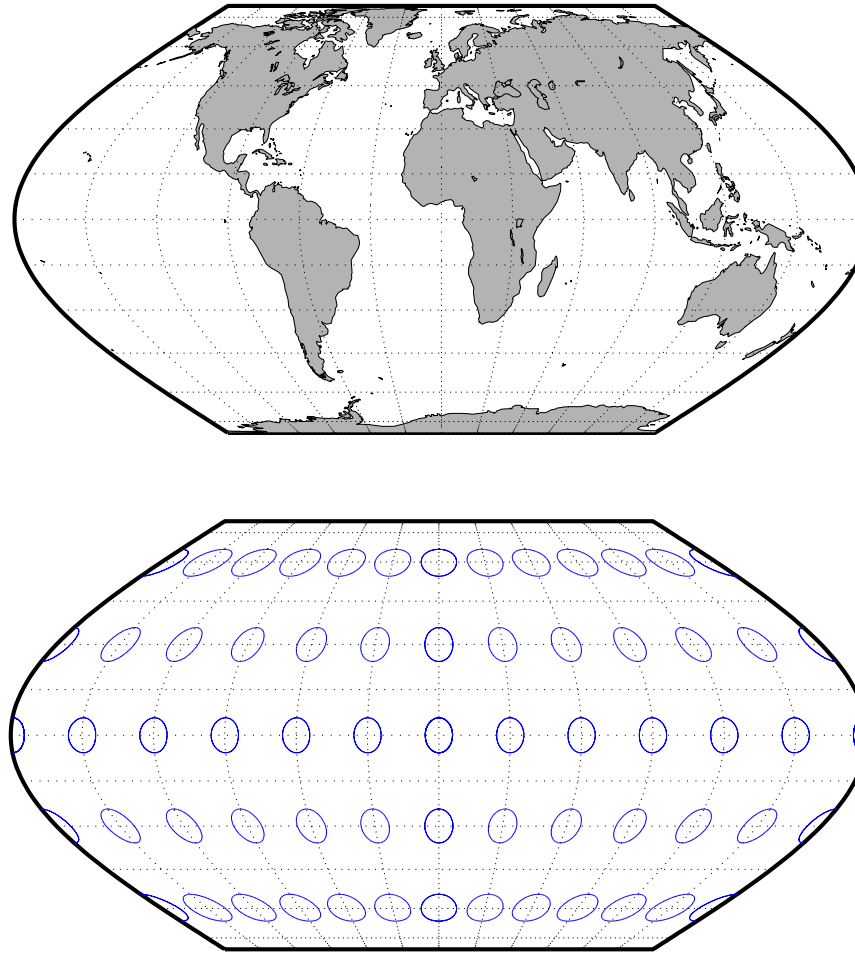
Poles: Lines half as long as the Equator.

Symmetry: About the central meridian or the Equator.

Features This is an equal-area projection. Scale is true along the 49°16' parallels and is constant along any parallel and between any pair of parallels equidistant from the Equator. It is free of distortion only at the two points where the 49°16' parallels intersect the central meridian. This projection is not conformal or equidistant.

Parallels For this projection, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. The standard parallel is by definition fixed at 49°16'.

Remarks This projection was presented by Max Eckert in 1906.



Equal-Area Cylindrical Projection

Classification Cylindrical

Syntax `eqacylin`

Graticule Meridians: Equally spaced straight parallel lines.

Parallels: Unequally spaced straight parallel lines, perpendicular to the meridians. Spacing is closest near the poles.

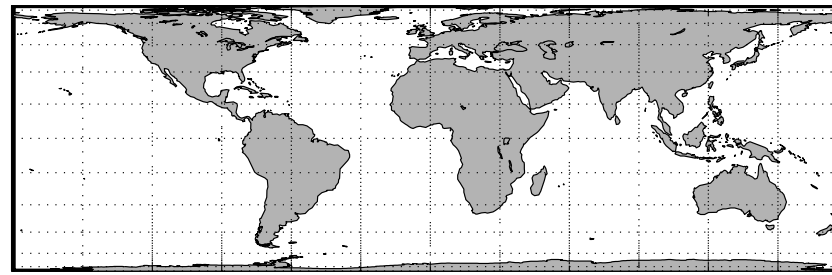
Poles: Straight lines equal in length to the Equator.

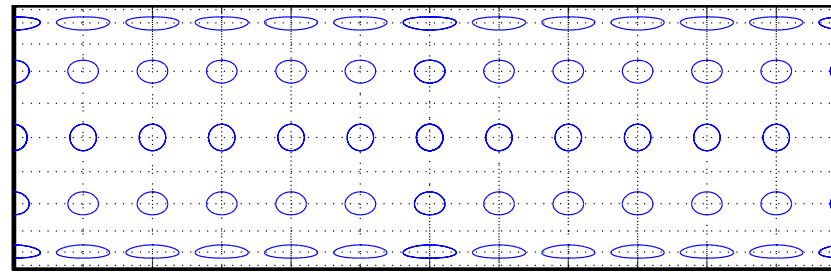
Symmetry: About any meridian or the Equator.

Features This is an orthographic projection onto a cylinder secant at the standard parallels. It is equal-area, but distortion of shape increases with distance from the standard parallels. Scale is true along the standard parallels and constant between two parallels equidistant from the Equator. This projection is not equidistant.

Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, any latitude may be chosen; the default is arbitrarily set to 0° (the Lambert variation).

Remarks This projection was proposed by Johann Heinrich Lambert (1772), a prolific cartographer who proposed seven different important projections. The form of this projection tangent at the Equator is often called the Lambert Equal-Area Cylindrical projection. That and other special forms of this projection are included separately in this guide, including the Gall Orthographic, the Behrmann Cylindrical, the Balthasart Cylindrical, and the Trystan Edwards Cylindrical projections.





Equidistant Azimuthal Projection

Classification Azimuthal

Syntax `eqdazim`

Graticule The graticule described is for the polar aspect.

Meridians: Equally spaced straight lines intersecting at a central pole. The angles between them are the true angles.

Parallels: Equally spaced circles, centered on the central pole. The entire Earth may be shown.

Poles: Central pole is a point. The opposite pole is a bounding circle with a radius twice that of the Equator.

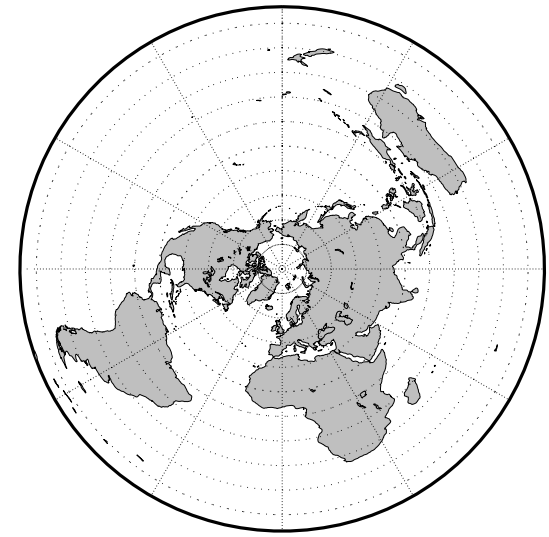
Symmetry: About any meridian.

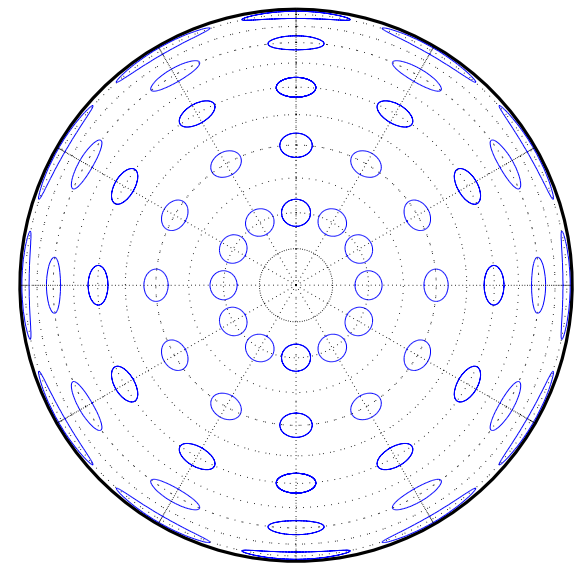
Features This is an equidistant projection. It is neither equal-area nor conformal. In the polar aspect, scale is true along any meridian. The projection is distortion free only at the center point. Distortion is moderate for the inner hemisphere, but it becomes extreme in the outer hemisphere.

Parallels There are no standard parallels for azimuthal projections.

Remarks This projection may have been first used by the ancient Egyptians for star charts. Several cartographers used it during the sixteenth century, including Guillaume Postel, who used it in 1581. Other names for this projection include Postel and Zenithal Equidistant.

Limitations This projection is available for the spherical geoid only.





Equidistant Conic Projection

Classification Conic

Syntax `eqdconic`

Graticule Meridians: Equally spaced straight lines converging to a common point, usually beyond the pole. The angles between the meridians are less than the true angles.

Parallels: Equally spaced concentric circular arcs centered on the point of meridanal convergence.

Poles: Normally circular arcs, enclosing the same angle as the displayed parallels.

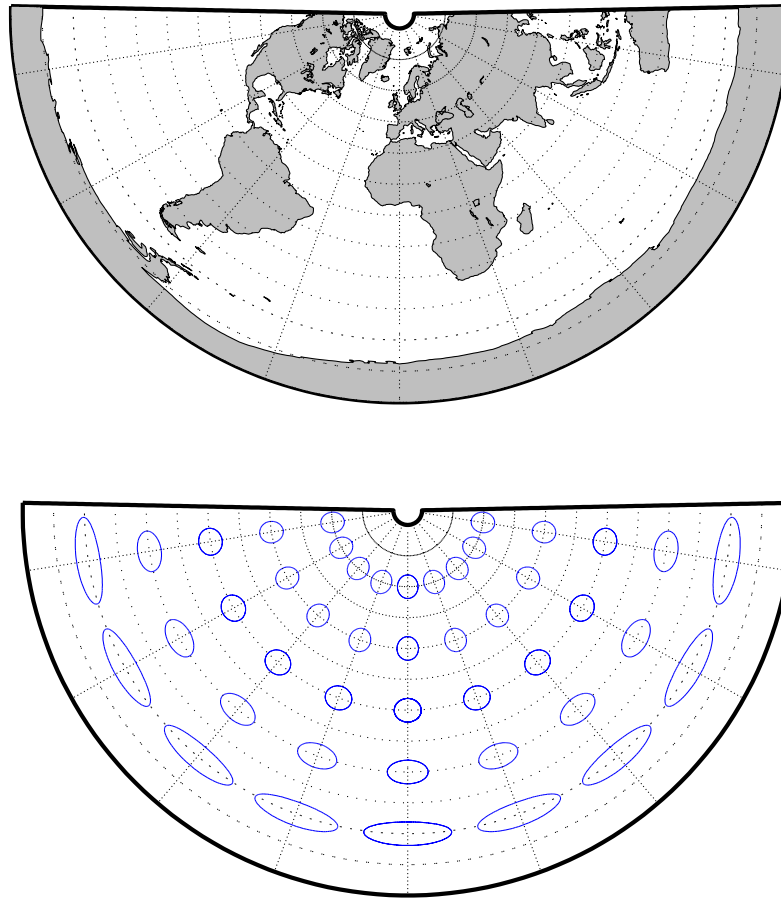
Symmetry: About any meridian.

Features Scale is true along each meridian and the one or two selected standard parallels. Scale is constant along any parallel. This projection is free of distortion along the two standard parallels. Distortion is constant along any other parallel. This projection provides a compromise in distortion between conformal and equal-area conic projections, of which it is neither.

Parallels The cone of projection has interesting limiting forms. If a pole is selected as a single standard parallel, the cone is a plane, and an Equidistant Azimuthal projection results. If two parallels are chosen, not symmetric about the Equator, then an Equidistant Conic projection results. If a pole is selected as one of the standard parallels, then the projected pole is a point, otherwise the projected pole is an arc. If the Equator is so chosen, the cone becomes a cylinder and a Plate Carrée projection results. If two parallels equidistant from the Equator are chosen as the standard parallels, an Equidistant Cylindrical projection results. The default parallels are [15 75].

Remarks In a rudimentary form, this projection dates back to Claudius Ptolemy, about A.D. 100. Improvements were developed by Johannes Ruysch in 1508, Gerardus Mercator in the late 16th century, and Nicolas de l'Isle in 1745. It is also known as the Simple Conic or Conic projection.

Limitations Longitude data greater than 135° east or west of the central meridian is trimmed.



Equidistant Cylindrical Projection

Classification Cylindrical

Syntax `eqdcylin`

Graticule Meridians: Equally spaced straight parallel lines more than half as long as the Equator.

Parallels: Equally spaced straight parallel lines, perpendicular to and having wider spacing than the meridians.

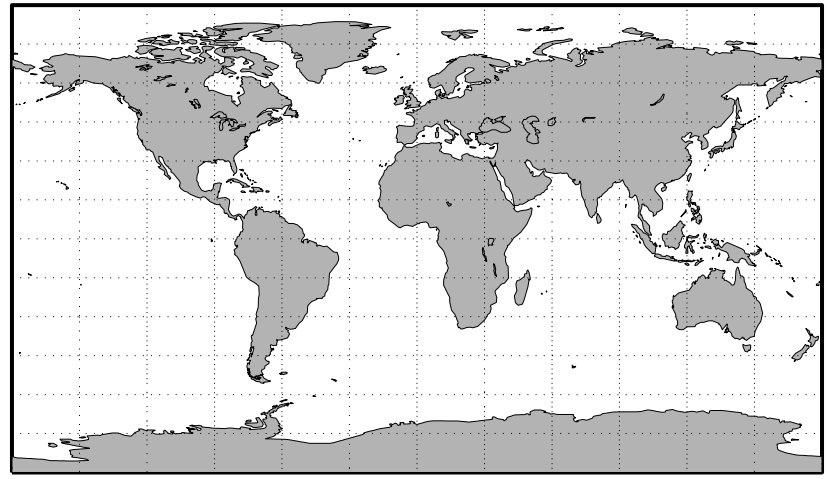
Poles: Straight lines equal in length to the Equator.

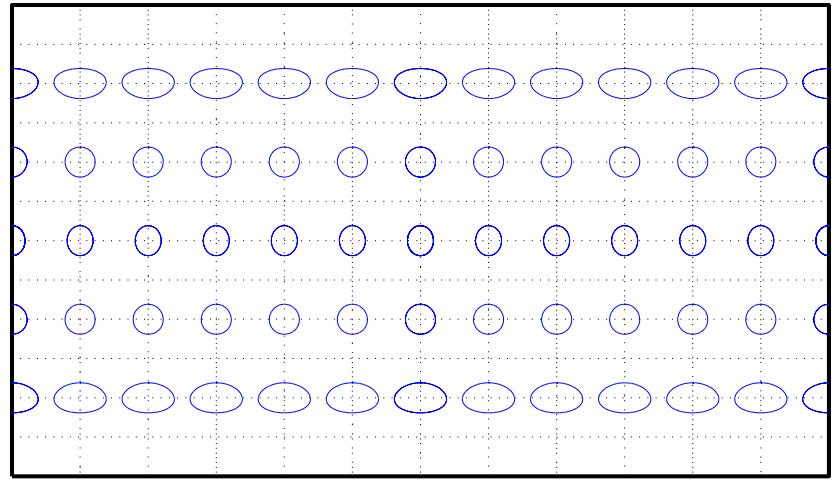
Symmetry: About any meridian or the Equator.

Features This is a projection onto a cylinder secant at the standard parallels. Distortion of both shape and area increase with distance from the standard parallels. Scale is true along all meridians (i.e., it is equidistant) and the standard parallels and is constant along any parallel and along the parallel of opposite sign.

Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, any latitude can be chosen; the default is arbitrarily set to 30°.

Remarks This projection was first used by Marinus of Tyre about A.D. 100. Special forms of this projection are the Plate Carrée, with a standard parallel at 0°, and the Gall Isographic, with standard parallels at 45°N and S. Other names for this projection include Equirectangular, Rectangular, Projection of Marinus, *La Carte Parallélogrammatique*, and *Die Rechteckige Plattkarte*.





Fournier Projection

Classification Pseudocylindrical

Syntax `fournier`

Graticule Meridians: Equally spaced elliptical curves converging at the poles.

Parallels: Straight lines.

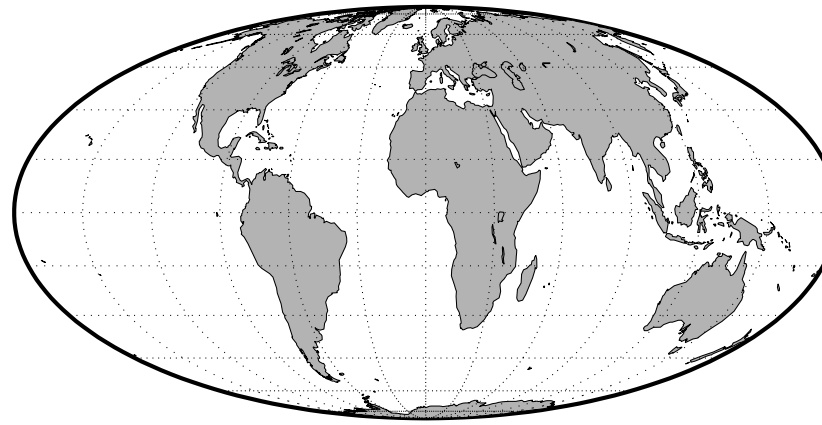
Poles: Points.

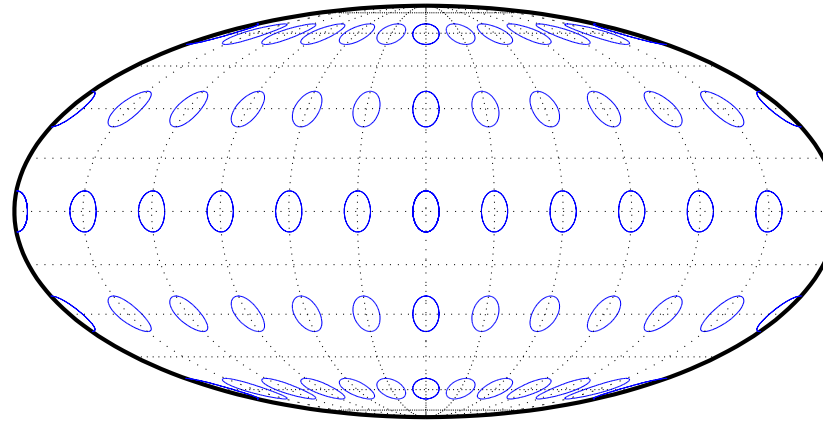
Symmetry: About the Equator and central meridian.

Features This projection is equal-area. Scale is constant along any parallel or pair of parallels equidistant from the Equator. This projection is neither equidistant nor conformal.

Parallels There is no standard parallel for this projection.

Remarks This projection was first described in 1643 by Georges Fournier. This is actually his second projection, the Fournier II.





Gall Isographic Projection

Classification Cylindrical

Syntax `giso`

Graticule Meridians: Equally spaced straight parallel lines more than half as long as the Equator.

Parallels: Equally spaced straight parallel lines, perpendicular to and having wider spacing than the meridians.

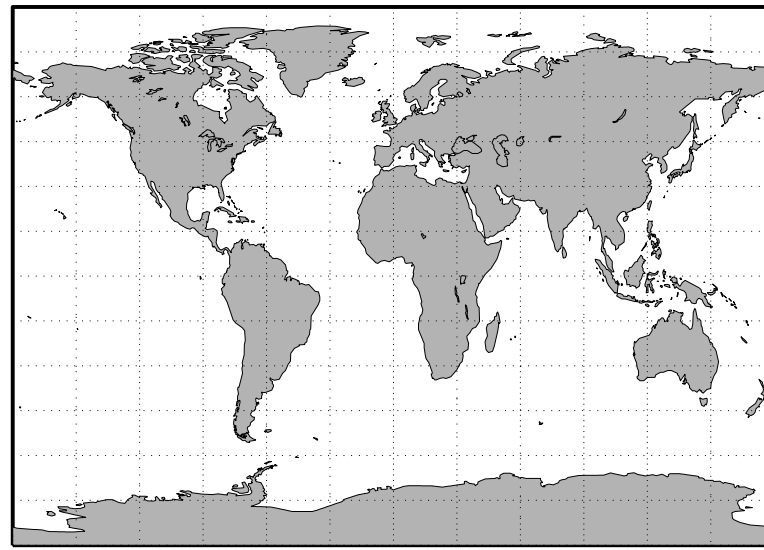
Poles: Straight lines equal in length to the Equator.

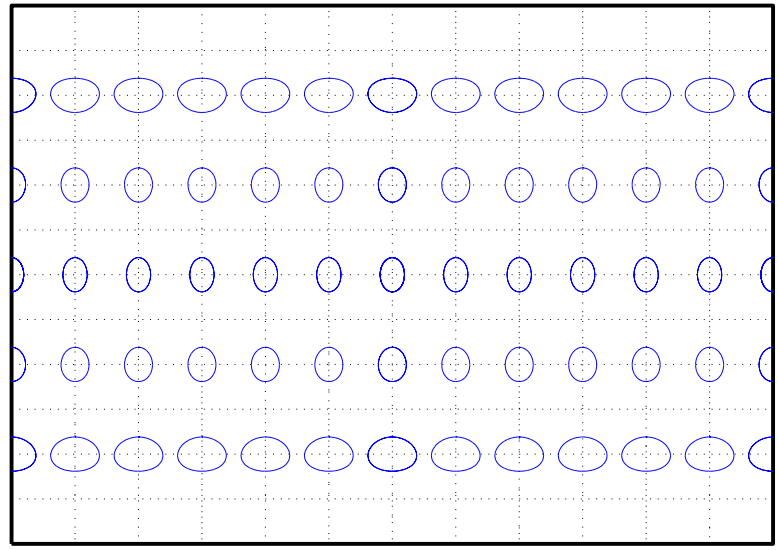
Symmetry: About any meridian or the Equator.

Features This is a projection onto a cylinder secant at the 45° parallels. Distortion of both shape and area increase with distance from the standard parallels. Scale is true along all meridians (i.e., it is equidistant) and the two standard parallels, and is constant along any parallel and along the parallel of opposite sign.

Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, the standard parallel is by definition fixed at 45°.

Remarks This projection is a specific case of the Equidistant Cylindrical projection, with standard parallels at 45°N and S.





Gall Orthographic Projection

Classification Cylindrical

Syntax `gortho`

Graticule Meridians: Equally spaced straight parallel lines.

Parallels: Unequally spaced straight parallel lines, perpendicular to the meridians. Spacing is closest near the poles.

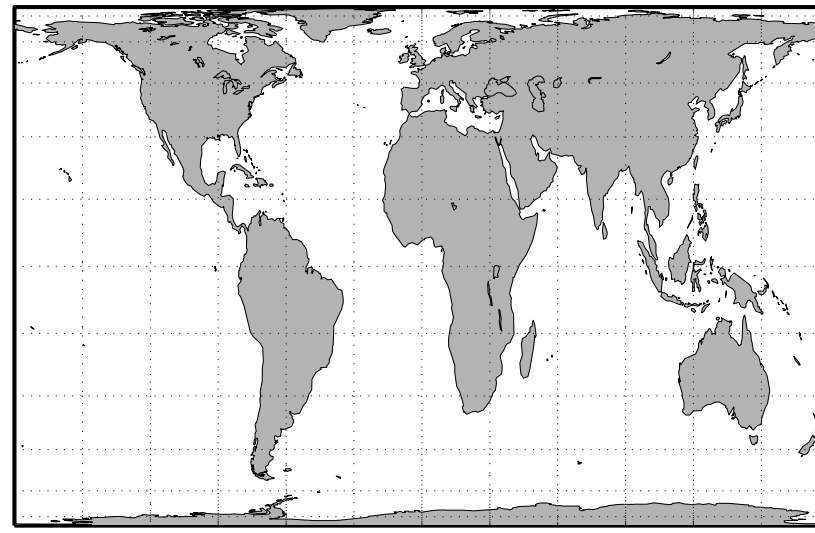
Poles: Straight lines equal in length to the Equator.

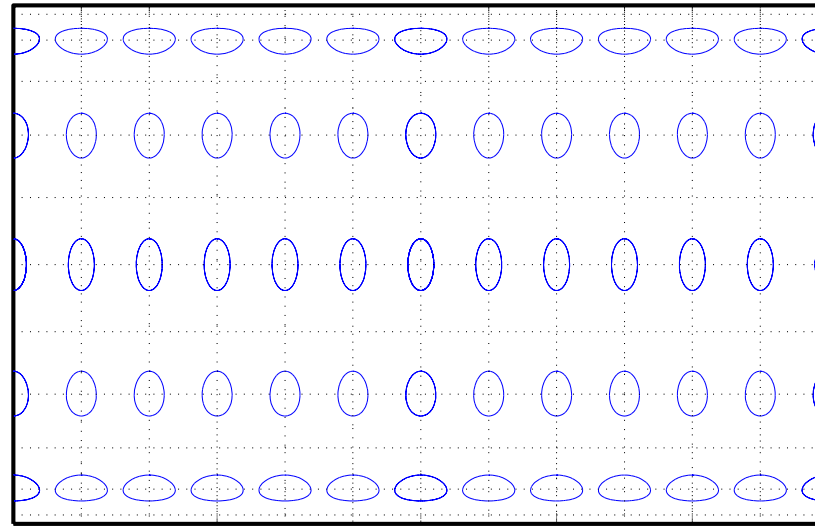
Symmetry: About any meridian or the Equator.

Features This is an orthographic projection onto a cylinder secant at the 45° parallels. It is equal-area, but distortion of shape increases with distance from the standard parallels. Scale is true along the standard parallels and constant between two parallels equidistant from the Equator. This projection is not equidistant.

Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, the standard parallel is by definition fixed at 45°.

Remarks This projection is named for James Gall, who originated it in 1855 and is a special form of the Equal-Area Cylindrical projection secant at 45°N and S. This projection is also known as the Peters projection.





Gall Stereographic Projection

Classification Cylindrical

Syntax `gstereo`

Graticule Meridians: Equally spaced straight parallel lines 0.77 as long as the Equator.

Parallels: Unequally spaced straight parallel lines, perpendicular to the meridians. Spacing increases toward the poles.

Poles: Straight lines equal in length to the Equator.

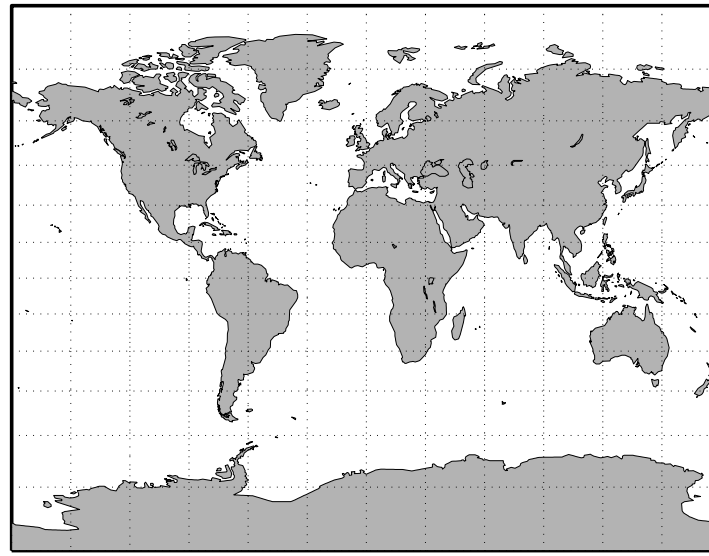
Symmetry: About any meridian or the Equator.

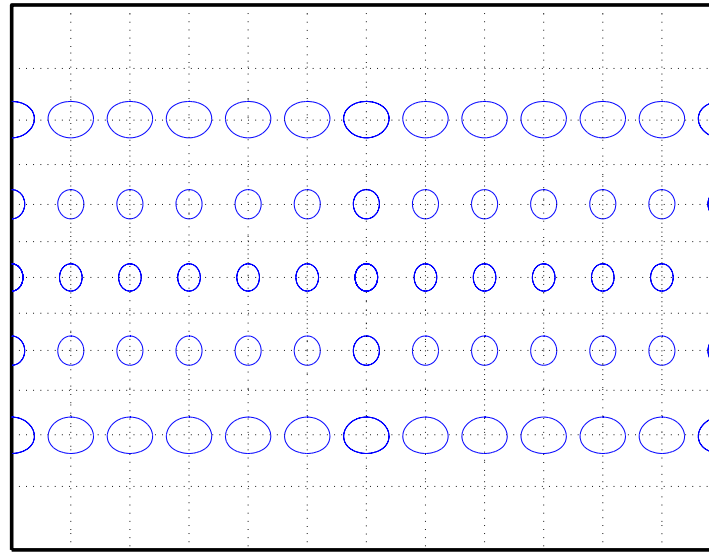
Features This is a perspective projection from a point on the Equator opposite a given meridian onto a cylinder secant at the 45° parallels. It is not equal-area, equidistant, or conformal. Scale is true along the standard parallels and constant between two parallels equidistant from the Equator. There is no distortion along the standard parallels, but it increases moderately away from these parallels, becoming severe at the poles.

Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, the standard parallel is by definition fixed at 45°.

Remarks This projection was presented by James Gall in 1855. It is also known simply as the Gall projection. It is a special form of the Braun Perspective Cylindrical projection secant at 45°N and S.

Limitations This projection is available for the spherical geoid only.





Globe

Classification Spherical

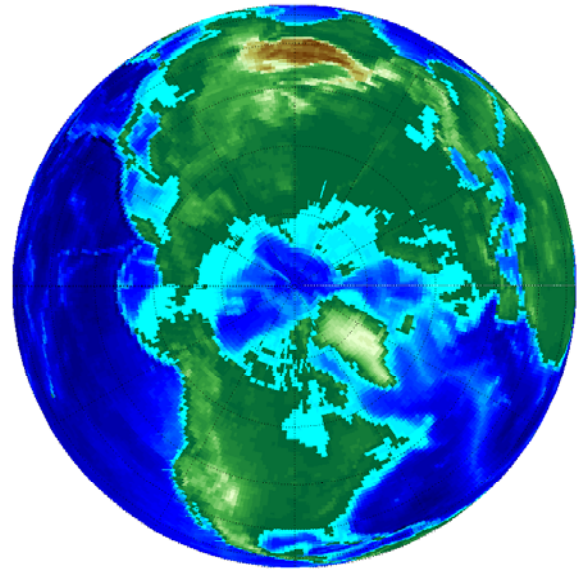
Syntax `globe`

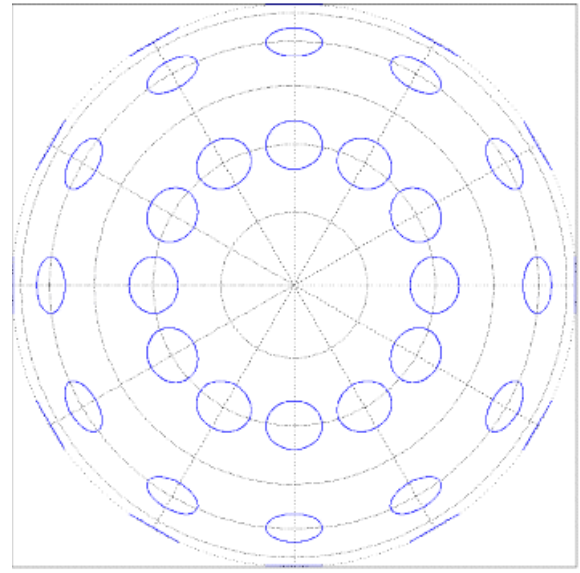
Graticule This *projection* is constructed by calculating a three-dimensional frame and displaying the map objects on the surface of this frame.

Features In the three-dimensional sense, this projection is true in scale, equal-area, conformal, minimum error, and equidistant everywhere. When displayed, however, this projection looks like an Orthographic azimuthal projection, provided that the MATLAB Axes `Projection` property is set to `'orthographic'`.

Parallels The globe requires no standard parallels.

Remarks This is the only three-dimensional representation provided for display. Unless some other display purpose requires three dimensions, the Orthographic projection's display is equivalent.





Gnomonic Projection

Classification Azimuthal

Syntax `gnomonic`

Graticule The graticule described is for a polar aspect.

Meridians: Equally spaced straight lines intersecting at the central pole. The angles displayed are the true angles between meridians.

Parallels: Unequally spaced circles centered on the central pole. Spacing increases rapidly away from this pole. The Equator and the opposite hemisphere cannot be shown

Pole: The central pole is a point; the other pole is not shown.

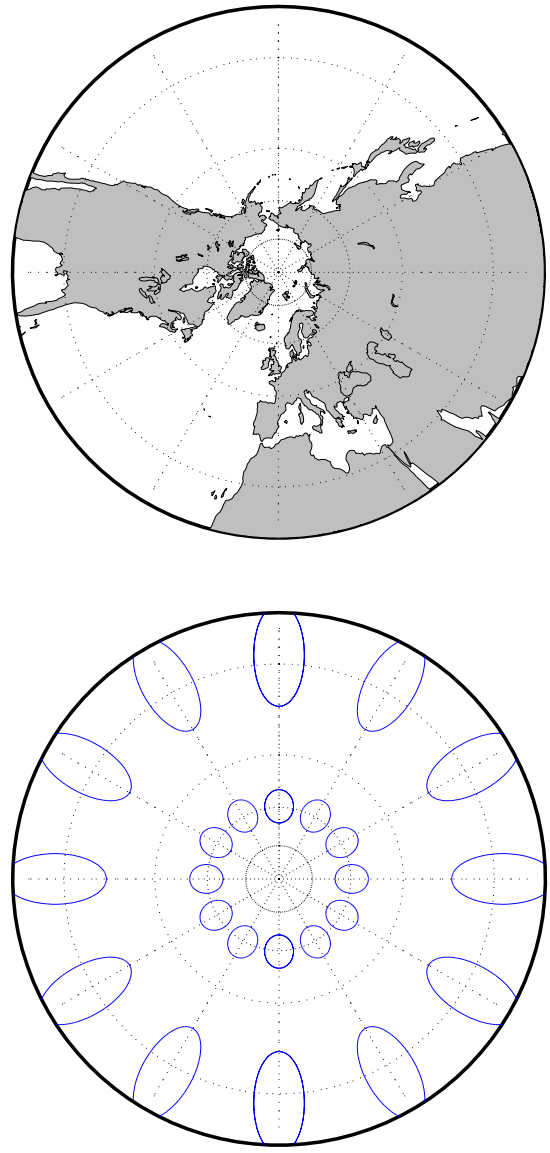
Symmetry: About any meridian.

Features This is a perspective projection from the center of the globe on a plane tangent at the center point, which is a pole in the common polar aspect, but can be any point. Less than one hemisphere can be shown with this projection, regardless of its center point. The significant property of this projection is that all great circles are straight lines. This is useful in navigation, as a great circle is the shortest path between two points on the globe. Only the center point enjoys true scale and zero distortion. This projection is neither conformal nor equal-area.

Parallels There are no standard parallels for azimuthal projections.

Remarks This projection may have been first developed by Thales around 580 B.C. Its name is derived from the gnomon, the face of a sundial, since the meridians radiate like hour markings. This projection is also known as a Gnomic or Central projection.

Limitations This projection is available for the spherical geoid only. Data greater than 65° distant from the center point is trimmed.



Goode Homolosine Projection

Classification Pseudocylindrical

Syntax `goode`

Graticule Central Meridian: Straight line 0.44 as long as the Equator.

Other Meridians: Equally spaced sinusoidal curves between the 40°44'11.8" parallels and elliptical arcs elsewhere, all concave toward the central meridian. The result is a slight, visible bend in the meridians at 40°44'11.8" N and S.

Parallels: Straight parallel lines, perpendicular to the central meridian. Equally spaced between the 40°44'11.8" parallels, with gradually decreasing spacing outside these parallels.

Poles: Points.

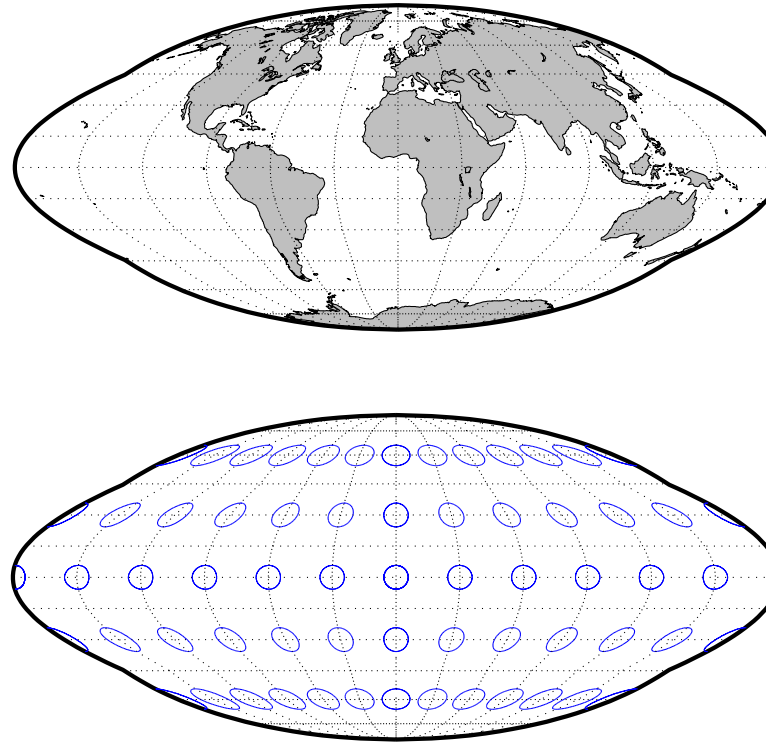
Symmetry: About the central meridian or the Equator.

Features This is an equal-area projection. Scale is true along all parallels and the central meridian between 40°44'11.8" N and S, and is constant along any parallel and between any pair of parallels equidistant from the Equator for all latitudes. Its distortion is identical to that of the Sinusoidal projection between 40°44'11.8" N and S, and to that of the Mollweide projection elsewhere. This projection is not conformal or equidistant.

Parallels This projection has one standard parallel, which is by definition fixed at 0°.

Remarks This projection was developed by J. Paul Goode in 1916. It is sometimes called simply the Homolosine projection, and it is usually used in an interrupted form. It is a merging of the Sinusoidal and Mollweide projections.

Limitations This projection is available in an uninterrupted form only.



Hammer Projection

Classification Modified Azimuthal

Syntax `hammer`

Graticule Meridians: Central meridian is a straight line half the length of the Equator. Other meridians are complex curves, equally spaced along the Equator, and concave towards the central meridian.

Parallels: Equator is straight. Other parallels are complex curves, equally spaced along the central meridian, and concave towards the nearest pole.

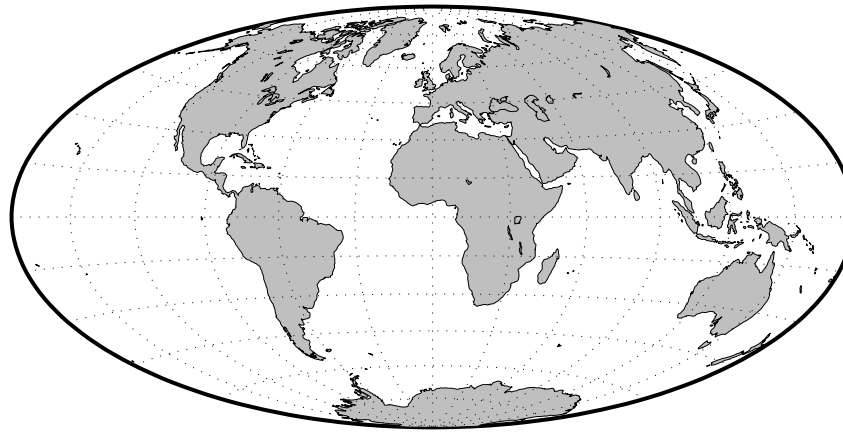
Poles: Points.

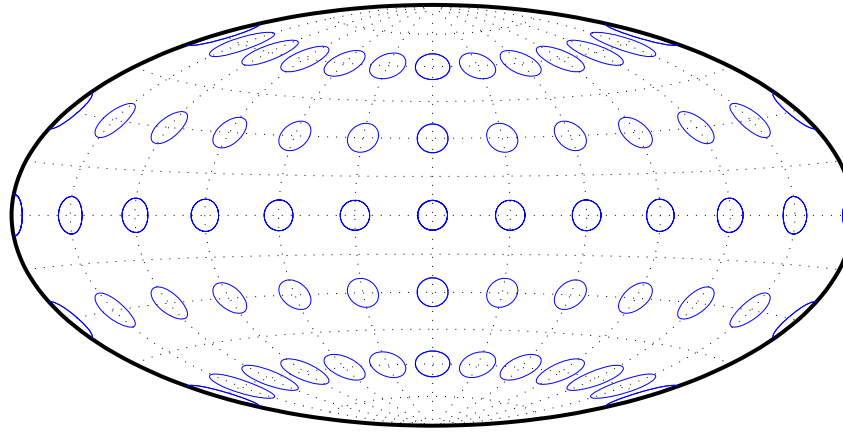
Symmetry: About the Equator and central meridian.

Features This projection is equal-area. The only point free of distortion is the center point. Distortion of shape is moderate throughout. This projection has less angular distortion on the outer meridians near the poles than pseudoazimuthal projections

Parallels There is no standard parallel for this projection.

Remarks This projection was presented by H. H. Ernst von Hammer in 1892. It is a modification of the Lambert Azimuthal Equal Area projection. Inspired by Aitoff projection, it is also known as the Hammer-Aitoff. It in turn inspired the Briesemeister, a modified oblique Hammer projection. John Bartholomew's Nordic projection is an oblique Hammer centered on 45 degrees north and the Greenwich meridian. The Hammer projection is used in whole-world maps and astronomical maps in galactic coordinates.





Hatano Asymmetrical Equal-Area Projection

Classification Pseudocylindrical

Syntax `hatano`

Graticule Central Meridian: Straight line 0.48 as long as the Equator.

Other Meridians: Equally spaced elliptical arcs concave toward the central meridian. The eccentricity of each ellipse changes at the Equator.

Parallels: Unequally spaced straight parallel lines, perpendicular to the central meridian. Spacing is not symmetrical about the Equator.

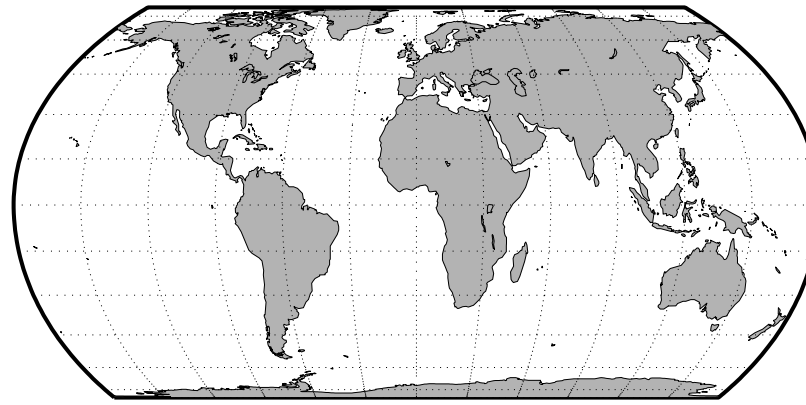
Poles: The North Pole is a line two-thirds the length of the Equator; the South Pole is a line three-fourths the length of the Equator.

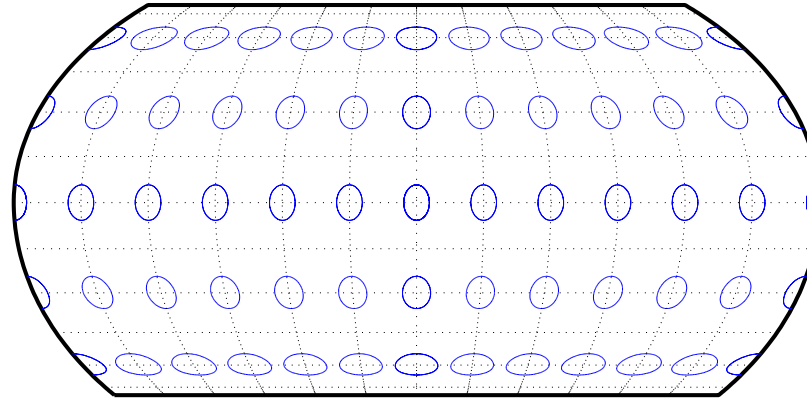
Symmetry: About the central meridian but *not* the Equator.

Features This is an equal-area projection. Scale is true along 40°42'N and 38°27'S, and is constant along any parallel but generally *not* between pairs of parallels equidistant from the Equator. It is free of distortion only along the central meridian at 40°42'N and 38°27'S. This projection is not conformal or equidistant.

Parallels Because of the asymmetrical nature of this projection, two standard parallels must be specified. The standard parallels are by definition fixed at 40°42'N and 38°27'S.

Remarks This projection was presented by Masataka Hatano in 1972.





Kavraisky V Projection

Classification Pseudocylindrical

Syntax `kavrsky5`

Graticule Meridians: Complex curves converging at the poles. A sine function is used for y, but the meridians are not sine curves.

Parallels: Unequally spaced straight lines.

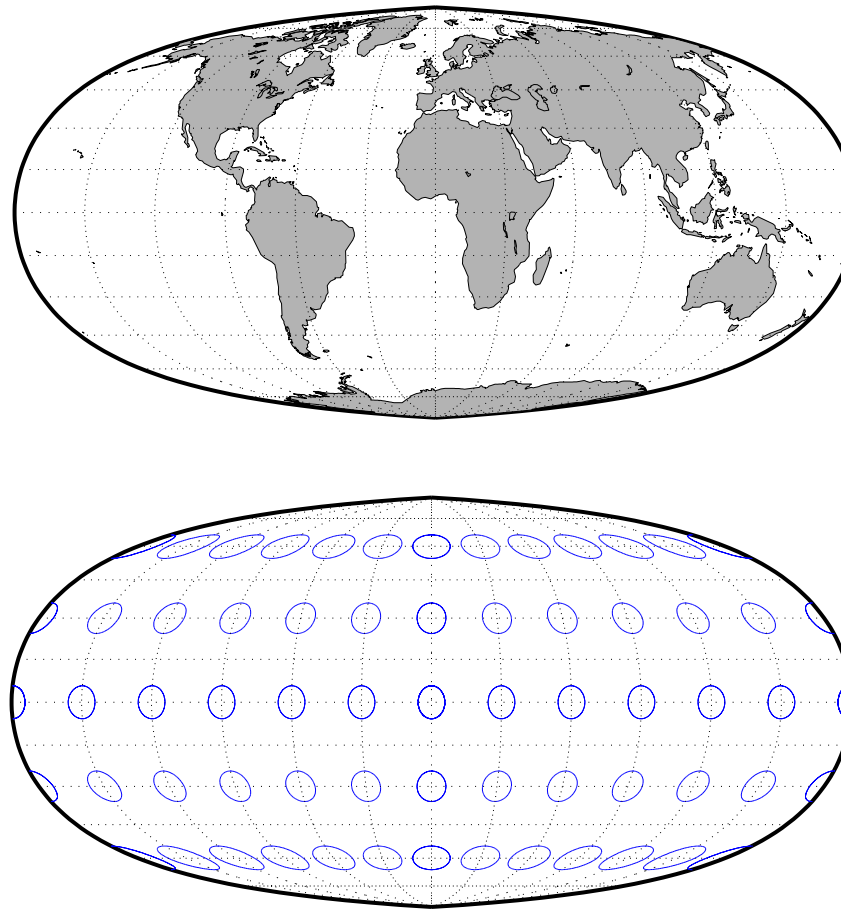
Poles: Points.

Symmetry: About the Equator and the central meridian.

Features This is an equal-area projection. Scale is true along the fixed standard parallels at 35°, and 0.9 true along the Equator. This projection is neither conformal nor equidistant.

Parallels The fixed standard parallels are at 35°.

Remarks This projection was described by V. V. Kavraisky in 1933.



Kavraisky VI Projection

Classification Pseudocylindrical

Syntax `kavrsky6`

Graticule Central Meridian: Straight line half the length of the Equator.

Meridians: Sine curves (60° segments).

Parallels: Unequally spaced straight lines.

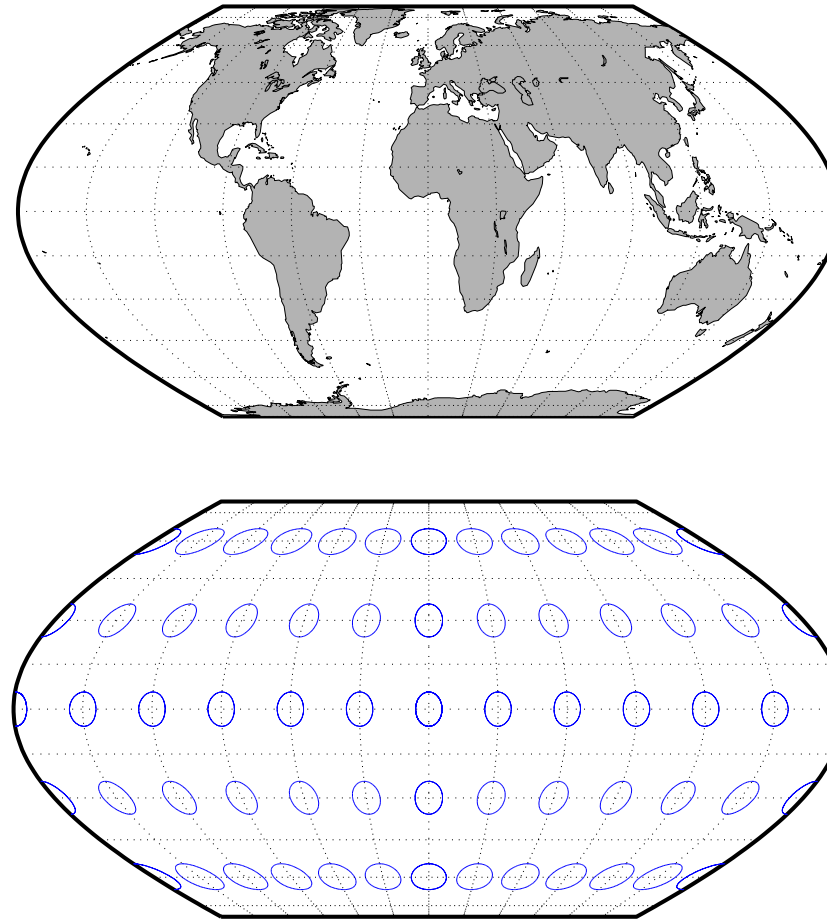
Poles: Straight lines half the length of the Equator.

Symmetry: About the Equator and the central meridian.

Features This is an equal-area projection. Scale is constant along any parallel or pair of equidistant parallels. This projection is neither conformal nor equidistant.

Parallels There are no standard parallels for this projection.

Remarks This projection was described by V. V. Kavraisky in 1936. It is also called the Wagner I, for Karlheinz Wagner, who described it in 1932.



Lambert Azimuthal Equal-Area Projection

Classification Azimuthal

Syntax `eqaazim`

Graticule The graticule described is for a polar aspect.

Meridians: Equally spaced straight lines intersecting at the central pole. The angles displayed are the true angles between meridians.

Parallels: Unequally spaced circles centered on the central pole. The entire Earth can be shown. Spacing decreases away from the central pole.

Pole: The central pole is a point; the other pole is a bounding circle with 1.41 the radius of the Equator.

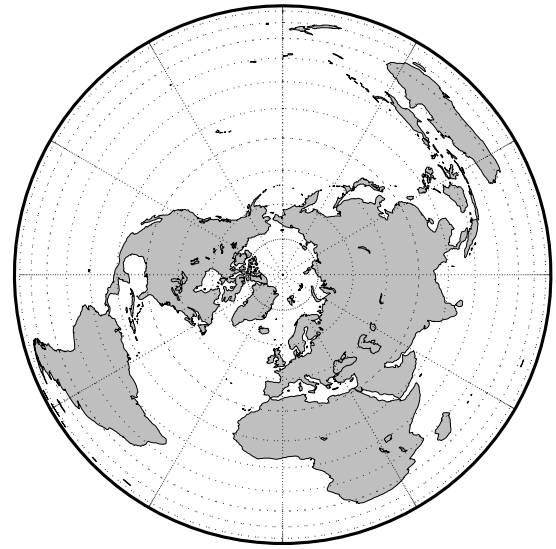
Symmetry: About any meridian.

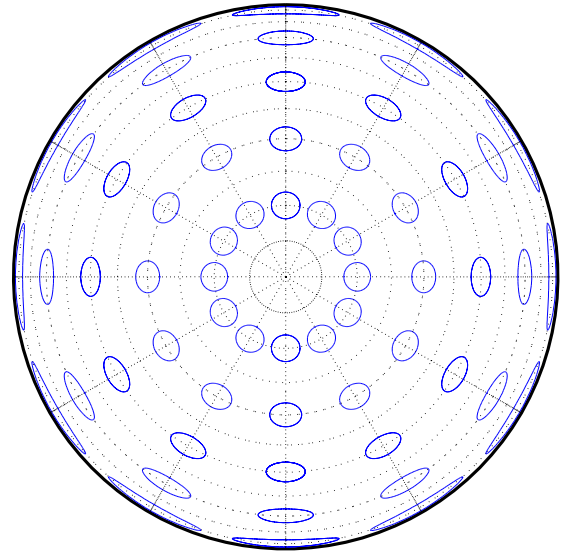
Features This nonperspective projection is equal-area. Only the center point is free of distortion, but distortion is moderate within 90° of this point. Scale is true only at the center point, increasing tangentially and decreasing radially with distance from the center point. This projection is neither conformal nor equidistant.

Parallels There are no standard parallels for azimuthal projections.

Remarks This projection was presented by Johann Heinrich Lambert in 1772. It is also known as the Zenithal Equal-Area and the Zenithal Equivalent projection, and the Lorgna projection in its polar aspect.

Limitations Data greater than 160° distant from the center point is trimmed.





Lambert Conformal Conic Projection

Classification Conic

Syntax `lambert`

Graticule Meridians: Equally spaced straight lines converging at one of the poles. The angles between the meridians are less than the true angles.

Parallels: Unequally spaced concentric circular arcs centered on the pole of convergence. Spacing of parallels increases away from the central latitudes.

Poles: The pole nearest a standard parallel is a point, the other cannot be shown.

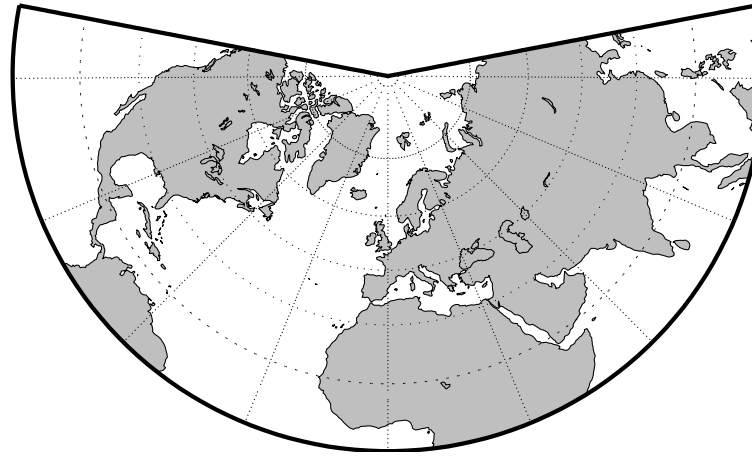
Symmetry: About any meridian.

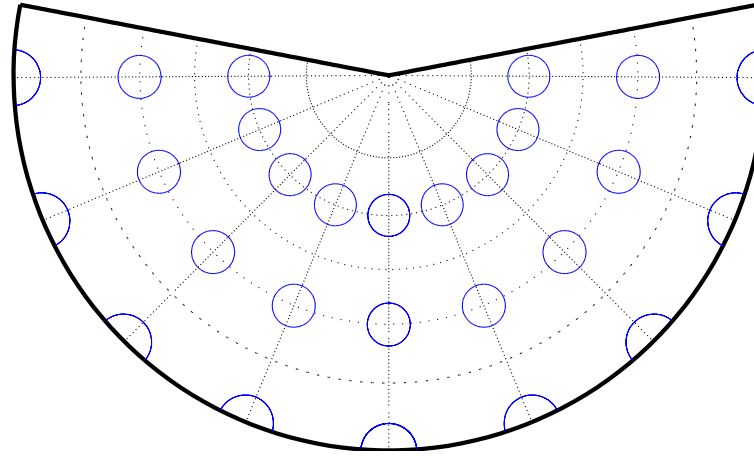
Features Scale is true along the one or two selected standard parallels. Scale is constant along any parallel and is the same in every direction at any point. This projection is free of distortion along the standard parallels. Distortion is constant along any other parallel. This projection is conformal everywhere but the poles; it is neither equal-area nor equidistant.

Parallels The cone of projection has interesting limiting forms. If a pole is selected as a single standard parallel, the cone is a plane, and a Stereographic Azimuthal projection results. If two parallels are chosen, not symmetric about the Equator, then a Lambert Conformal Conic projection results. If a pole is selected as one of the standard parallels, then the projected pole is a point, otherwise the projected pole is an arc. If the Equator or two parallels equidistant from the Equator are chosen as the standard parallels, the cone becomes a cylinder, and a Mercator projection results. The default parallels are [15 75].

Remarks This projection was presented by Johann Heinrich Lambert in 1772 and is also known as a Conical Orthomorphic projection.

Limitations Longitude data greater than 135° east or west of the central meridian is trimmed. The default map limits are [0 90] to avoid extreme area distortion.





Lambert Equal-Area Cylindrical Projection

Classification Cylindrical

Syntax `lambcyln`

Graticule Meridians: Equally spaced straight parallel lines 0.32 as long as the Equator.

Parallels: Unequally spaced straight parallel lines, perpendicular to the meridians. Spacing is closest near the poles.

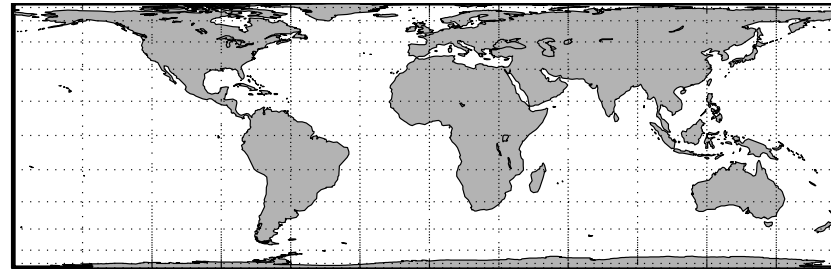
Poles: Straight lines equal in length to the Equator.

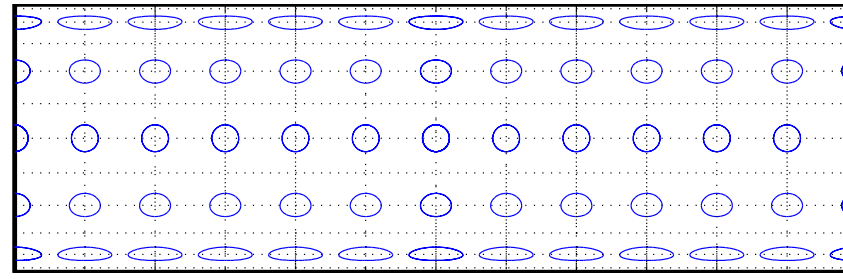
Symmetry: About any meridian or the Equator.

Features This is an orthographic projection onto a cylinder tangent at the Equator. It is equal-area, but distortion of shape increases with distance from the Equator. Scale is true along the Equator and constant between two parallels equidistant from the Equator. This projection is not equidistant.

Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, the standard parallel is by definition fixed at 0°.

Remarks This projection is named for Johann Heinrich Lambert and is a special form of the Equal-Area Cylindrical projection tangent at the Equator.





Loximuthal Projection

Classification Pseudocylindrical

Syntax `loximuth`

Graticule Central Meridian: Straight line at least half as long as the Equator. Actual length depends on the choice of central latitude. Length is 0.5 when the central latitude is the Equator, for example, and 0.65 for central latitudes of 40°.

Other Meridians: Complex curves intersecting at the poles and concave toward the central meridian.

Parallels: Equally spaced straight parallel lines, perpendicular to the central meridian.

Poles: Points.

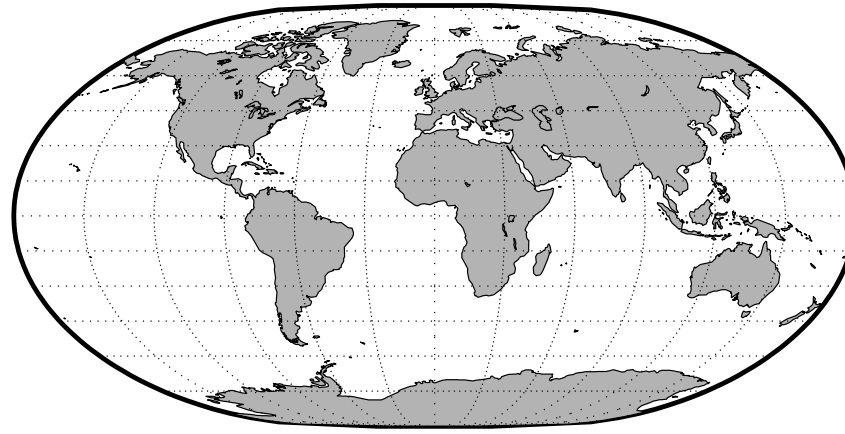
Symmetry: About the central meridian. Symmetry about the Equator only when it is the central latitude.

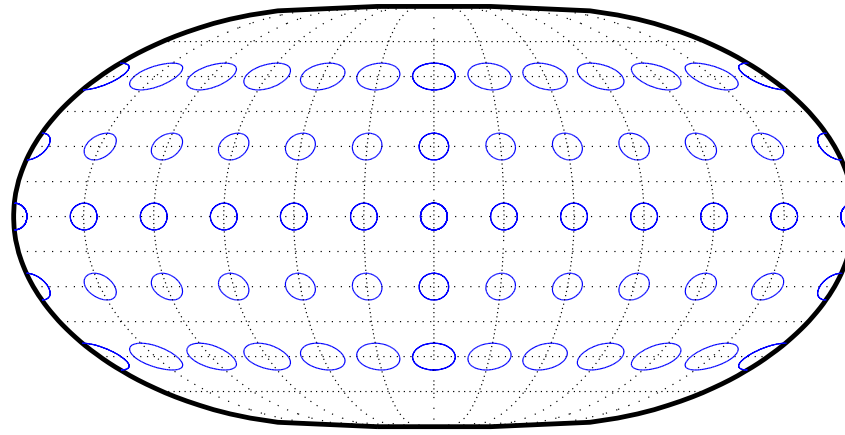
Features This projection has the special property that from the central point (the intersection of the central latitude with the central meridian), rhumb lines (loxodromes) are shown as straight, true to scale, and correct in azimuth from the center. This differs from the Mercator projection, in that rhumb lines are here shown in true scale and that unlike the Mercator, this projection does not maintain true azimuth for all points along the rhumb lines. Scale is true along the central meridian and is constant along any parallel, but not, generally, between parallels. It is free of distortion only at the central point and can be severely distorted in places. However, this projection is designed for its specific special property, in which distortion is not a concern.

Parallels For this projection, only one standard parallel is specified: the central latitude described above. Specification of this central latitude defines the center of the Loximuthal projection. The default value is 0°.

Remarks This projection was presented by Karl Siemon in 1935 and independently by Waldo R. Tobler in 1966. The Bordone Oval projection of 1520 was very similar to the Equator-centered Loximuthal.

Limitations This projection is available for the spherical geoid only.





McBryde-Thomas Flat-Polar Parabolic Projection

Classification Pseudocylindrical

Syntax `flatplrp`

Graticule Central Meridian: Straight line 0.48 as long as the Equator.

Other Meridians: Equally spaced parabolic curves concave toward the central meridian.

Parallels: Unequally spaced straight parallel lines, perpendicular to the central meridian. Spacing is greatest near the Equator.

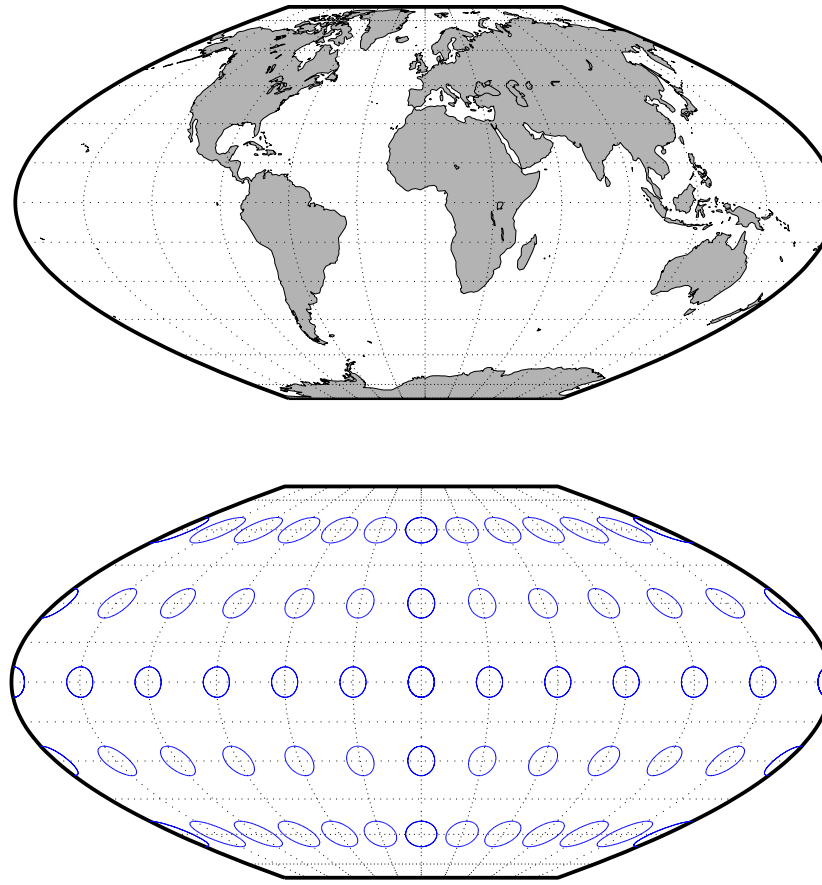
Poles: Lines one-third as long as the Equator.

Symmetry: About the central meridian or the Equator.

Features This is an equal-area projection. Scale is true along the 45°30' parallels and is constant along any parallel and between any pair of parallels equidistant from the Equator. Distortion is severe near the outer meridians at high latitudes, but less so than on the pointed-polar projections. It is free of distortion only at the two points where the central meridian intersects the 45°30' parallels. This projection is not conformal or equidistant.

Parallels For this projection, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. The standard parallel is by definition fixed at 45°30'.

Remarks This projection was presented by F. Webster McBryde and Paul D. Thomas in 1949.



McBryde-Thomas Flat-Polar Quartic Projection

Classification Pseudocylindrical

Syntax `flatplrq`

Graticule Central Meridian: Straight line 0.45 as long as the Equator.

Other Meridians: Equally spaced quartic (fourth-order equation) curves concave toward the central meridian.

Parallels: Unequally spaced straight parallel lines, perpendicular to the central meridian. Spacing is greatest near the Equator.

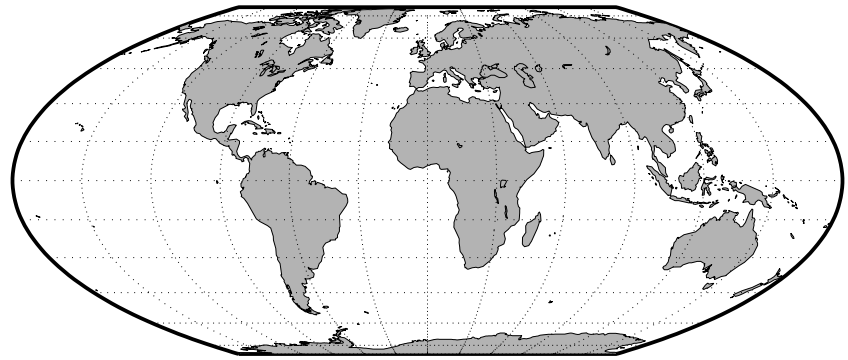
Poles: Lines one-third as long as the Equator.

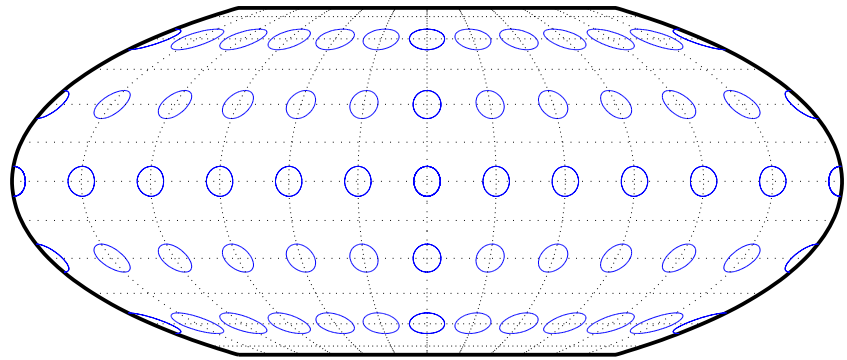
Symmetry: About the central meridian or the Equator.

Features This is an equal-area projection. Scale is true along the 33°45' parallels and is constant along any parallel and between any pair of parallels equidistant from the Equator. Distortion is severe near the outer meridians at high latitudes, but less so than on the pointed-polar projections. It is free of distortion only at the two points where the central meridian intersects the 33°45' parallels. This projection is not conformal or equidistant.

Parallels For this projection, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. The standard parallel is by definition fixed at 33°45'.

Remarks This projection was presented by F. Webster McBryde and Paul D. Thomas in 1949, and is also known simply as the Flat-Polar Quartic projection.





McBryde-Thomas Flat-Polar Sinusoidal Projection

Classification Pseudocylindrical

Syntax `flatplrs`

Graticule Central Meridian: Straight line half as long as the Equator.

Other Meridians: Equally spaced sinusoidal curves intersecting at the poles and concave toward the central meridian.

Parallels: Unequally spaced straight parallel lines, perpendicular to the central meridian. Spacing is widest near the Equator.

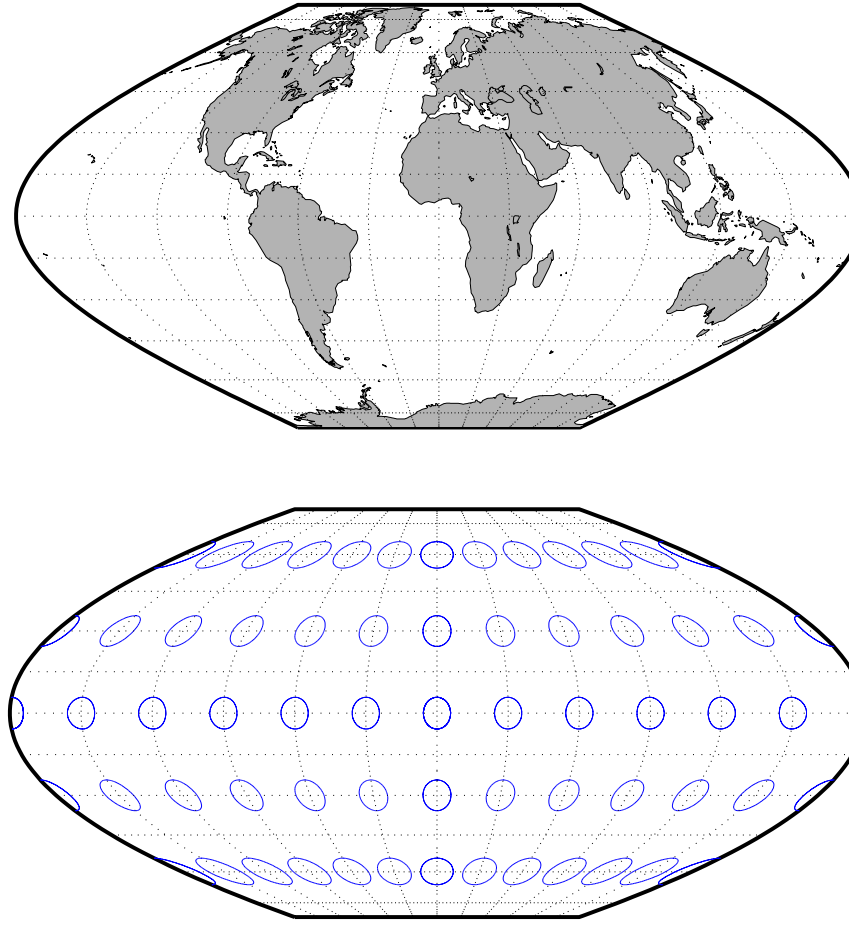
Poles: Lines one-third as long as the Equator.

Symmetry: About the central meridian or the Equator.

Features This projection is equal-area. Scale is true along the 55°51' parallels and is constant along any parallel and between any pair of parallels equidistant from the Equator. It is free of distortion only at the two points where the central meridian intersects the 55°51' parallels. This projection is not conformal or equidistant.

Parallels For this projection, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. The standard parallel is by definition fixed at 55°51'.

Remarks This projection was presented by F. Webster McBryde and Paul D. Thomas in 1949.



Mercator Projection

Classification Cylindrical

Syntax `mercator`

Graticule Meridians: Equally spaced straight parallel lines.

Parallels: Unequally spaced straight parallel lines, perpendicular to the meridians. Spacing increases toward the poles.

Poles: Cannot be shown.

Symmetry: About any meridian or the Equator.

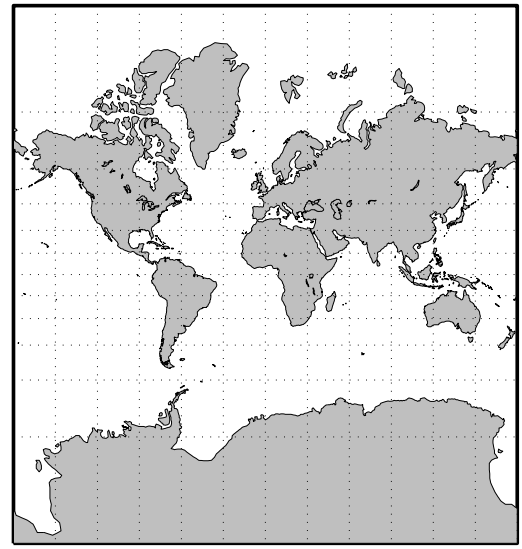
Features This is a projection with parallel spacing calculated to maintain conformality. It is not equal-area, equidistant, or perspective. Scale is true along the standard parallels and constant between two parallels equidistant from the Equator. It is also constant in all directions near any given point. Scale becomes infinite at the poles. The appearance of the Mercator projection is unaffected by the selection of standard parallels; they serve only to define the latitude of true scale.

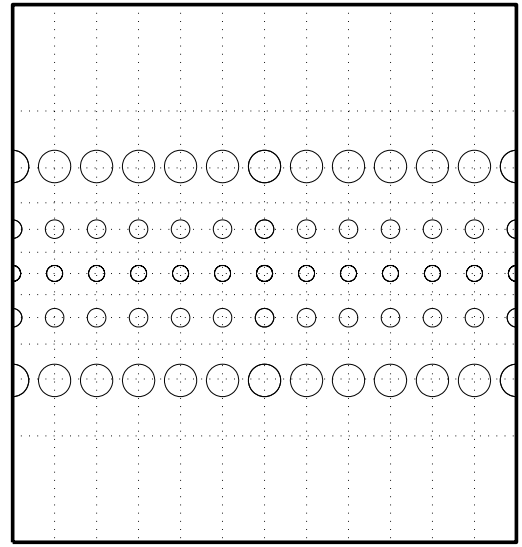
The Mercator, which may be the most famous of all projections, has the special feature that all rhumb lines, or loxodromes (lines that make equal angles with all meridians, i.e., lines of constant heading), are straight lines. This makes it an excellent projection for navigational purposes. However, the extreme area distortion makes it unsuitable for general maps of large areas.

Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, any latitude less than 86° may be chosen; the default is arbitrarily set to 0°.

Remarks The Mercator projection is named for Gerardus Mercator, who presented it *for navigation* in 1569. It is now known to have been used for the Tunhuang star chart as early as 940 by Ch'ien Lo-Chih. It was first used in Europe by Erhard Etzlaub in 1511. It is also, but rarely, called the Wright projection, after Edward Wright, who developed the mathematics behind the projection in 1599.

Limitations Data at latitudes greater than 86° is trimmed to prevent large *y*-values from dominating the display.





Miller Cylindrical Projection

Classification Cylindrical

Syntax `miller`

Graticule Meridians: Equally spaced straight parallel lines 0.73 as long as the Equator.

Parallels: Unequally spaced straight parallel lines, perpendicular to the meridians. Spacing increases toward the poles, less rapidly than that of the Mercator projection.

Poles: Straight lines equal in length to the Equator.

Symmetry: About any meridian or the Equator.

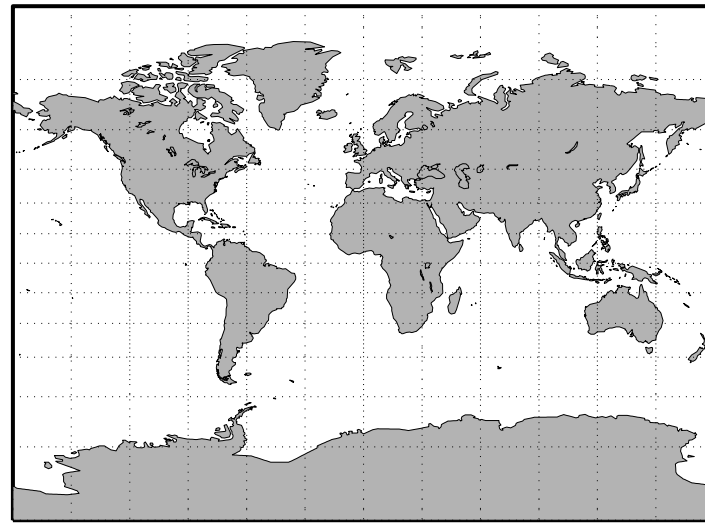
Features This is a projection with parallel spacing calculated to maintain a look similar to the Mercator projection while reducing the distortion near the poles and allowing the poles to be displayed. It is not equal-area, equidistant, conformal, or perspective. Scale is true along the Equator and constant between two parallels equidistant from the Equator. There is no distortion near the Equator, and it increases moderately away from the Equator, but it becomes severe at the poles.

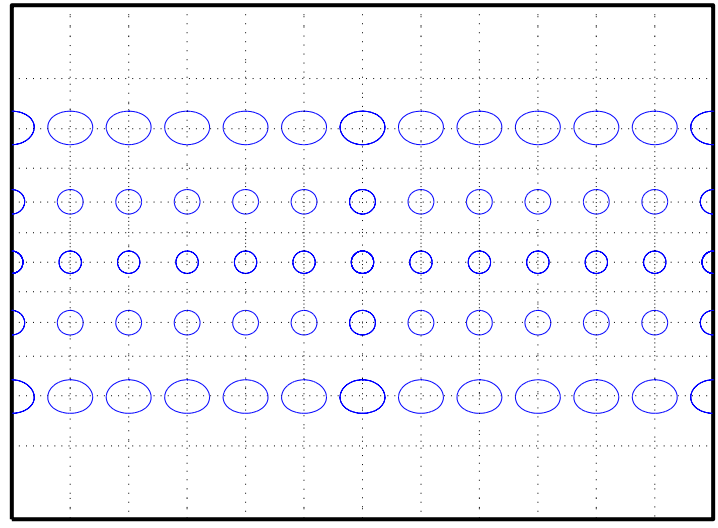
The Miller Cylindrical projection is derived from the Mercator projection; parallels are spaced from the Equator by calculating the distance on the Mercator for a parallel at 80% of the true latitude and dividing the result by 0.8. The result is that the two projections are almost identical near the Equator.

Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, the standard parallel is by definition fixed at 0°.

Remarks This projection was presented by Osborn Maitland Miller of the American Geographical Society in 1942. It is often used in place of the Mercator projection for atlas maps of the world, for which it is somewhat more appropriate.

Limitations This projection is available for the spherical geoid only.





Mollweide Projection

Classification Pseudocylindrical

Syntax `mollweid`

Graticule Central Meridian: Straight line half as long as the Equator.

Other Meridians: Meridians 90° east and west of the central meridian form a circle. The others are equally spaced semiellipses intersecting at the poles and concave toward the central meridian.

Parallels: Unequally spaced straight parallel lines, perpendicular to the central meridian. Spacing is greatest toward the Equator, but the spacing changes gradually.

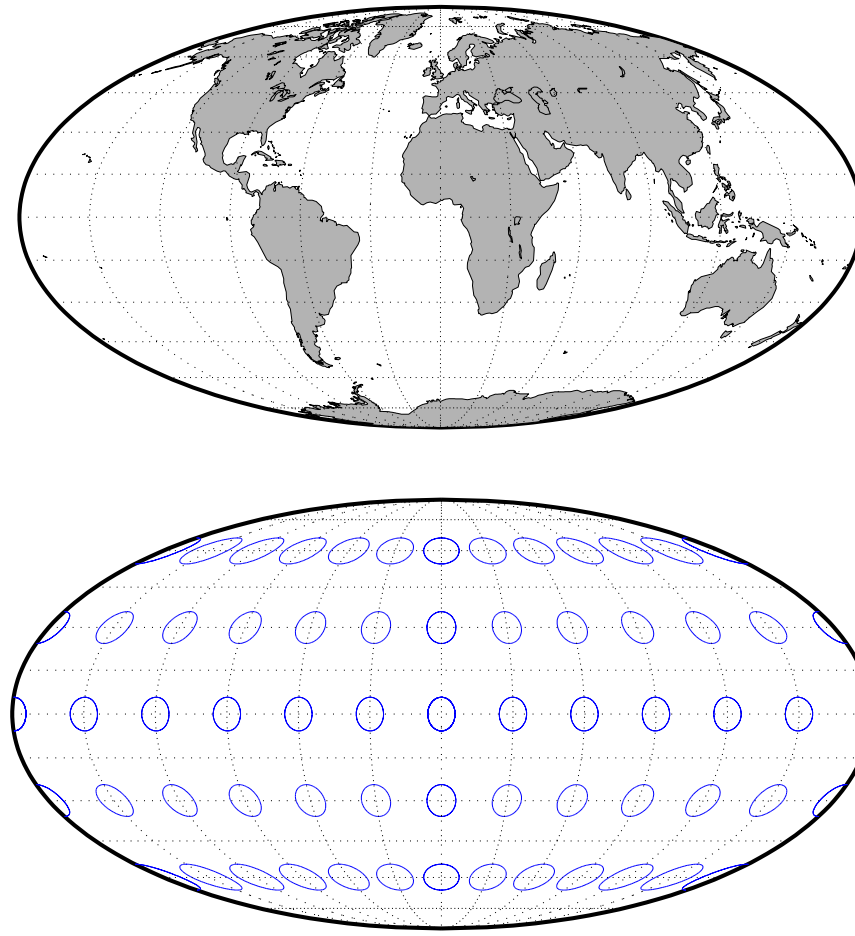
Poles: Points.

Symmetry: About the central meridian or the Equator.

Features This is an equal-area projection. Scale is true along the 40°44' parallels and is constant along any parallel and between any pair of parallels equidistant from the Equator. It is free of distortion only at the two points where the 40°44' parallels intersect the central meridian. This projection is not conformal or equidistant.

Parallels For this projection, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. The standard parallel is by definition fixed at 40°44'.

Remarks This projection was presented by Carl B. Mollweide in 1805. Its other names include the Homolographic, the Homalographic, the Babinet, and the Elliptical projections. It is occasionally used for thematic world maps, and it is combined with the Sinusoidal to produce the Goode Homolosine projection.



Murdoch I Conic Projection

Classification Conic

Syntax `murdoch1`

Graticule Meridians: Equally spaced straight lines converging at one of the poles.

Parallels: Equally spaced concentric circular arcs.

Poles: Arcs, one of which might become a point in the limit.

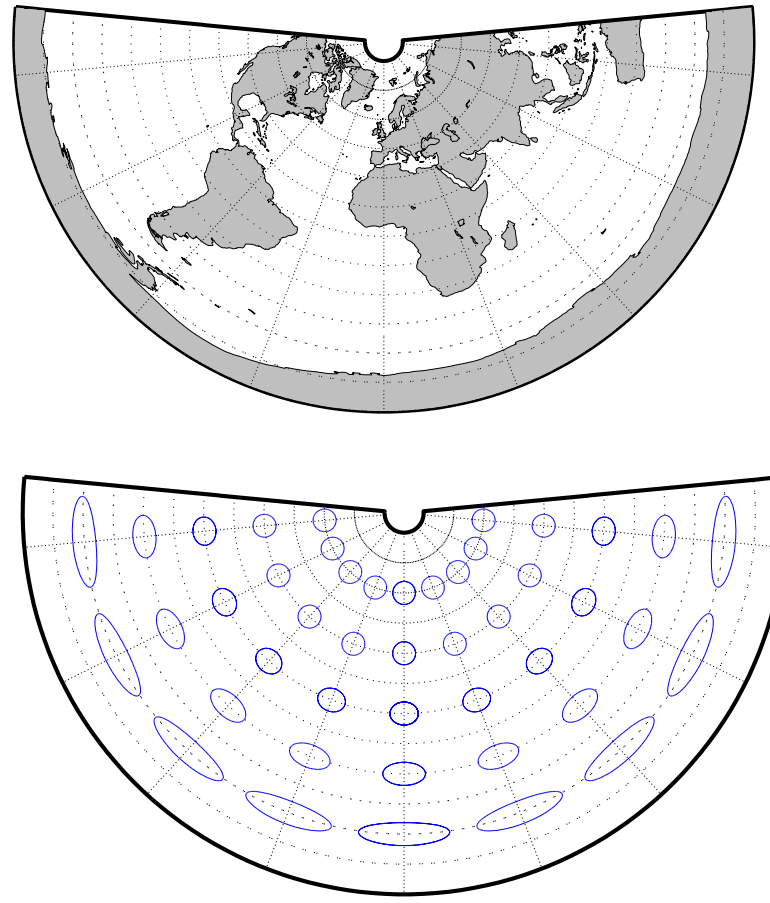
Symmetry: About any meridian.

Features This is an equidistant projection that is nearly minimum-error. Scale is true along any meridian and is constant along any parallel. Scale is also true along two standard parallels. These must be calculated, however (see below). The total area of the mapped area is correct, but it is not equal-area everywhere.

Parallels The parallels for this projection are not standard parallels, but rather limiting parallels. The special feature of this map, correct total area, holds between these parallels. The default parallels are [15 75].

Remarks Described by Patrick Murdoch in 1758.

Limitations This projection is available for the spherical geoid only. Longitude data greater than 135° east or west of the central meridian is trimmed.



Murdoch III Minimum Error Conic Projection

Classification Conic

Syntax `murdoch3`

Graticule Meridians: Equally spaced straight lines converging at one of the poles.

Parallels: Equally spaced concentric circular arcs.

Poles: Arcs, one of which might become a point in the limit.

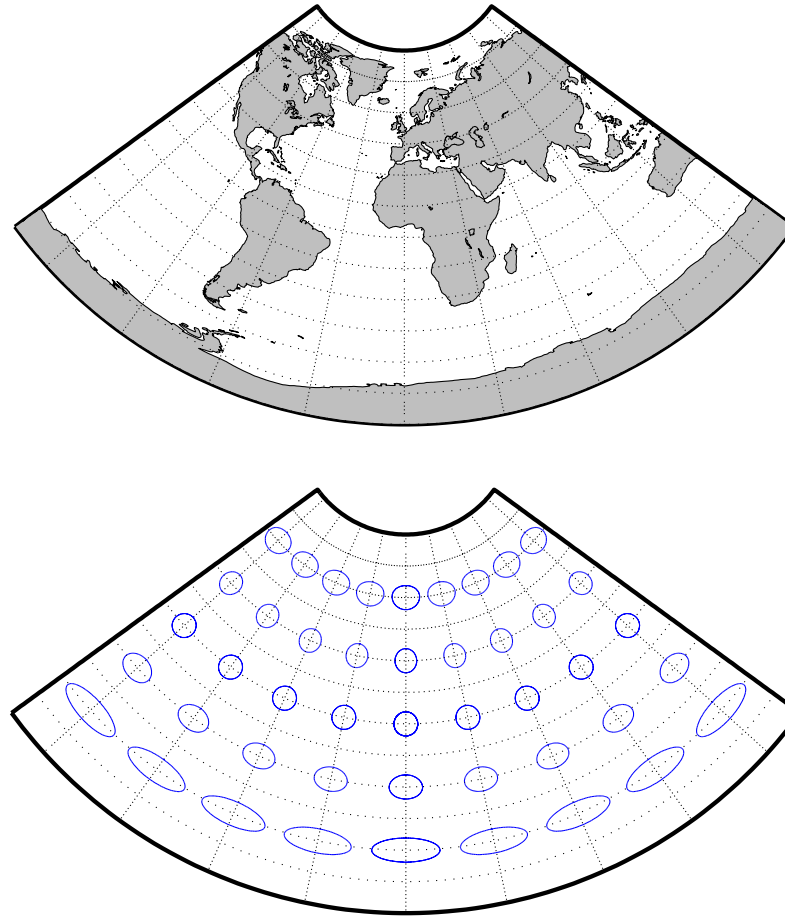
Symmetry: About any meridian.

Features This is an equidistant projection that is minimum-error. Scale is true along any meridian and is constant along any parallel. Scale is also true along two standard parallels. These must be calculated, however (see below). The total area of the mapped area is correct, but it is not equal-area everywhere.

Parallels The parallels for this projection are not standard parallels, but rather limiting parallels. The special feature of this map, correct total area, holds between these parallels. The default parallels are [15 75].

Remarks Described by Patrick Murdoch in 1758, with errors corrected by Everett in 1904.

Limitations This projection is available for the spherical geoid only. Longitude data greater than 135° east or west of the central meridian is trimmed.



Orthographic Projection

Classification Azimuthal

Syntax `ortho`

Graticule The graticule described is for a polar aspect.

Meridians: Equally spaced straight lines intersecting at the central pole. The angles displayed are the true angles between meridians.

Parallels: Unequally spaced circles centered on the central pole. Spacing decreases away from this pole. The opposite hemisphere cannot be shown.

Pole: The central pole is a point; the other pole is not shown.

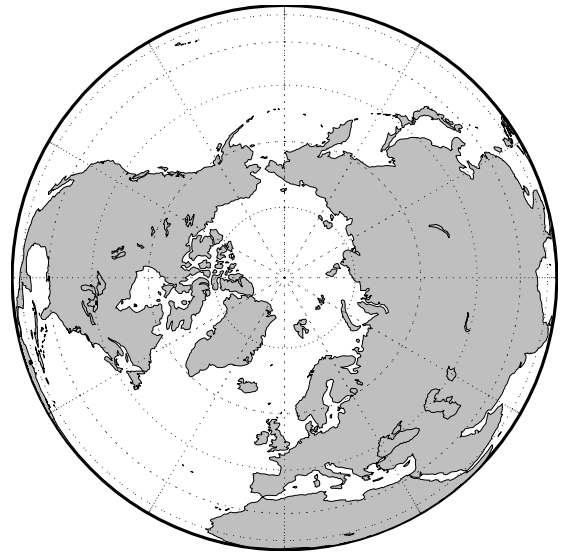
Symmetry: About any meridian.

Features This is a perspective projection on a plane tangent at the center point from an infinite distance (that is, orthogonally). The center point is a pole in the common polar aspect, but can be any point. This projection has two significant properties. It looks like a globe, providing views of the Earth resembling those seen from outer space. Additionally, all great and small circles are either straight lines or elliptical arcs on this projection. Scale is true only at the center point and is constant in the circumferential direction along any circle having the center point as its center. Distortion increases rapidly away from the center point, the only place that is distortion-free. This projection is neither conformal nor equal-area.

Parallels There are no standard parallels for azimuthal projections.

Remarks This projection appears to have been developed by the Egyptians and Greeks by the second century B.C.

Limitations This projection is available for the spherical geoid only. Data greater than 89° distant from the center point is trimmed.



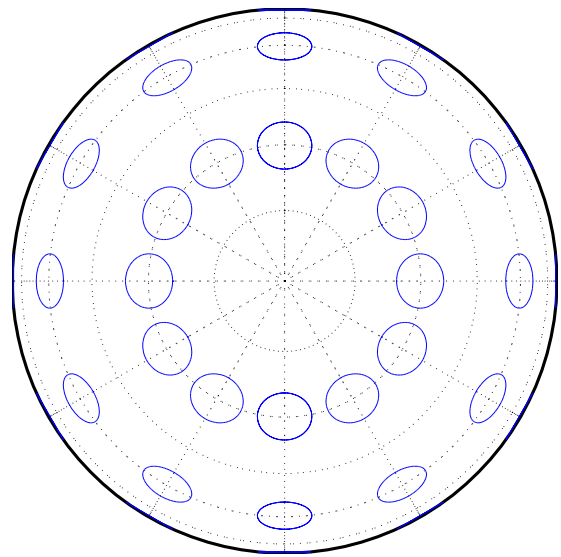


Plate Carrée Projection

Classification Cylindrical

Syntax `pcarree`

Graticule Meridians: Equally spaced straight parallel lines half as long as the Equator.

Parallels: Equally spaced straight parallel lines, perpendicular to and having the same spacing as the meridians.

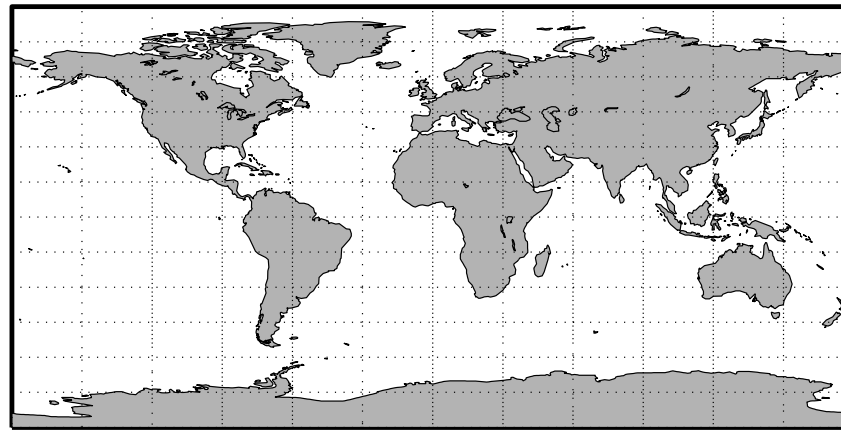
Poles: Straight lines equal in length to the Equator.

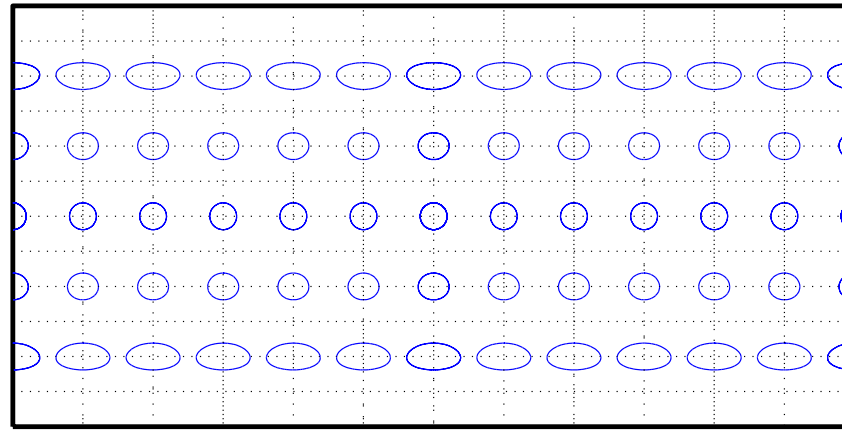
Symmetry: About any meridian or the Equator.

Features This is a projection onto a cylinder tangent at the Equator. Distortion of both shape and area increases with distance from the Equator. Scale is true along all meridians (i.e., it is equidistant) and the Equator and is constant along any parallel and along the parallel of opposite sign.

Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, the standard parallel is by definition fixed at 0°.

Remarks This projection, like the more general Equidistant Cylindrical, is credited to Marinus of Tyre, thought to have invented it about A.D. 100. It may, in fact, have been originated by Erastosthenes, who lived approximately 275-195 B.C. The Plate Carrée has the most simply constructed graticule of any projection. It was used frequently in the 15th and 16th centuries and is quite common today in very simple computer mapping programs. It is the simplest and limiting form of the Equidistant Cylindrical projection. Another name for this projection is the Simple Cylindrical. Its transverse aspect is the Cassini projection.





Polyconic Projection

Classification Polyconic

Syntax `polycon`

Graticule Central Meridian: A straight line.

Meridians: Complex curves spaced equally along the Equator and each parallel, and concave toward the central meridian.

Parallels: The Equator is a straight line. All other parallels are nonconcentric circular arcs spaced at true distances along the central meridian.

Poles: Normally circular arcs, enclosing the same angle as the displayed parallels.

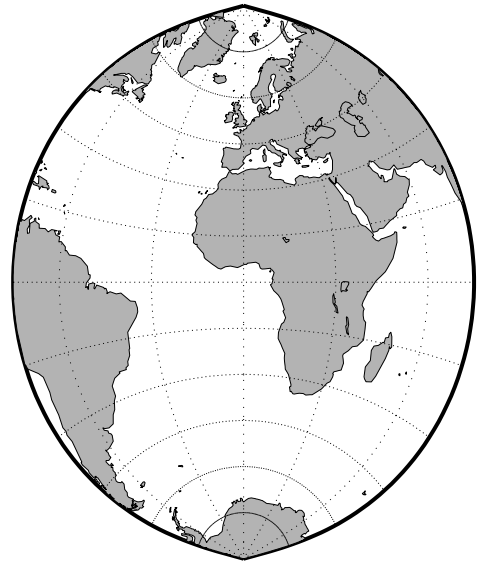
Symmetry: About the Equator or the central meridian.

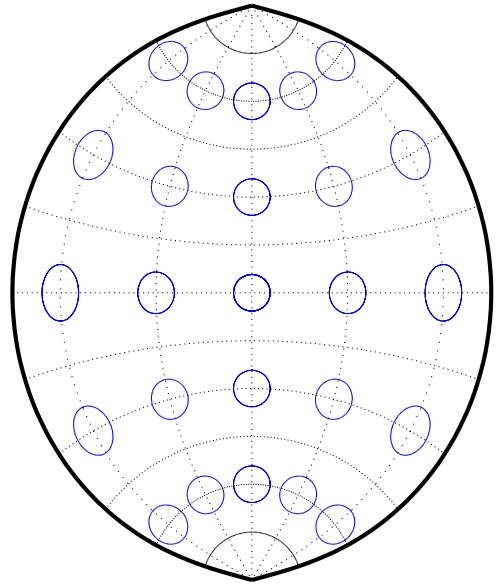
Features For this projection, each parallel has a curvature identical to its curvature on a cone tangent at that latitude. Since each parallel has its own cone, this is a "polyconic" projection. Scale is true along the central meridian and along each parallel. This projection is free of distortion only along the central meridian; distortion can be severe at extreme longitudes. This projection is neither conformal nor equal-area.

Parallels By definition, this projection has no standard parallels, since every parallel is a *standard parallel*.

Remarks This projection was apparently originated about 1820 by Ferdinand Rudolph Hassler. It is also known as the American Polyconic and the Ordinary Polyconic projection.

Limitations Longitude data greater than 75° east or west of the central meridian is trimmed.





Putnins P5 Projection

Classification Pseudocylindrical

Syntax `putnins5`

Graticule Central Meridian: Straight line half as long as the Equator.

Other Meridians: Equally spaced portions of hyperbolas intersecting at the poles and concave toward the central meridian.

Parallels: Equally spaced straight parallel lines, perpendicular to the central meridian.

Poles: Points.

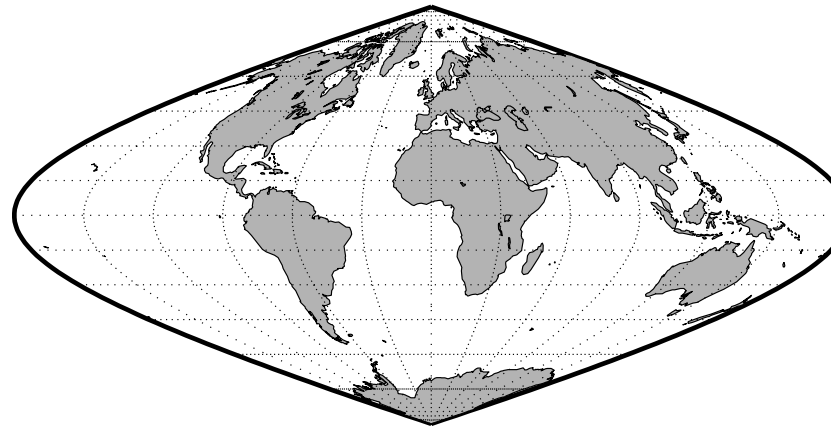
Symmetry: About the central meridian or the Equator.

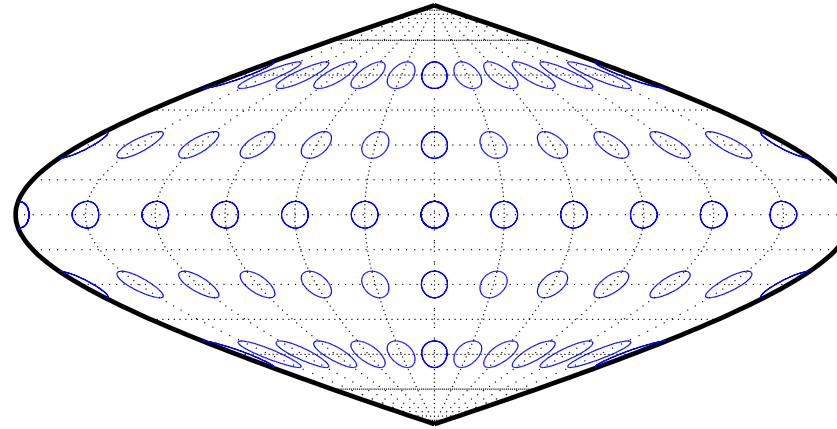
Features Scale is true along the 21°14' parallels and is constant along any parallel, between any pair of parallels equidistant from the Equator, and along the central meridian. It is not free of distortion at any point. This projection is not equal-area, conformal, or equidistant.

Parallels For this projection, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. The standard parallel is by definition fixed at 21°14'.

Remarks This projection was presented by Reinholds V. Putnins in 1934.

Limitations This projection is available for the spherical geoid only.





Quartic Authalic Projection

Classification Pseudocylindrical

Syntax `quartic`

Graticule Central Meridian: Straight line 0.45 as long as the Equator.

Other Meridians: Equally spaced quartic (fourth-order equation) curves concave toward the central meridian.

Parallels: Unequally spaced straight parallel lines, perpendicular to the central meridian. Spacing changes gradually and is greatest near the Equator.

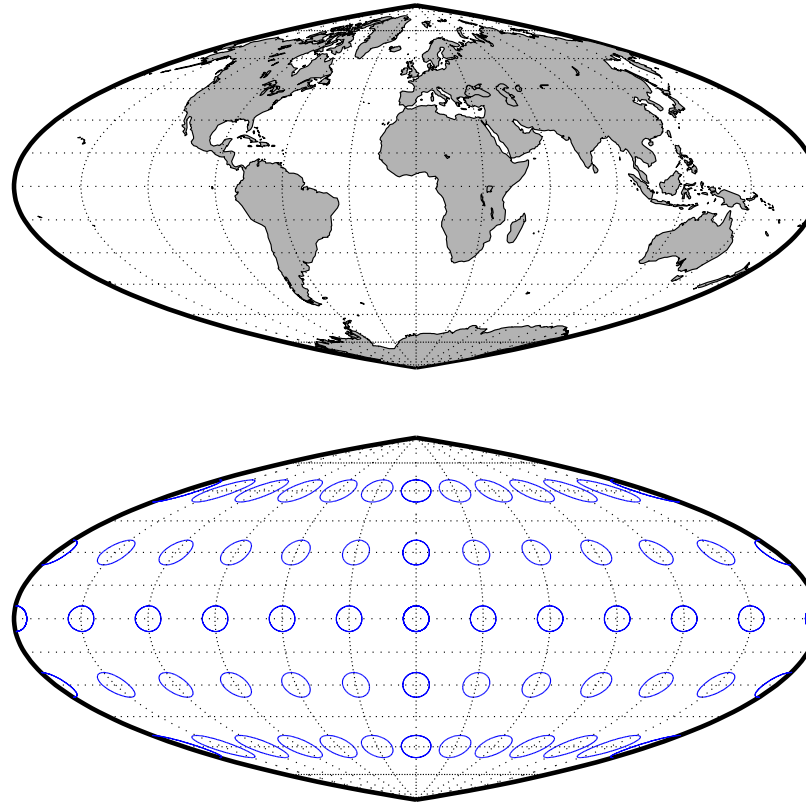
Poles: Points.

Symmetry: About the central meridian or the Equator.

Features This is an equal-area projection. Scale is true along the Equator and is constant along any parallel and between any pair of parallels equidistant from the Equator. Distortion is severe near the outer meridians at high latitudes, but less so than on the Sinusoidal projection. It is free of distortion along the Equator. This projection is not conformal or equidistant.

Parallels This projection has one standard parallel, which is by definition fixed at 0°.

Remarks This projection was presented by Karl Siemon in 1937 and independently by Oscar Sherman Adams in 1945.



Robinson Projection

Classification Pseudocylindrical

Syntax `robinson`

Graticule Central Meridian: Straight line 0.51 as long as the Equator.

Other Meridians: Equally spaced, resemble elliptical arcs and are concave toward the central meridian.

Parallels: Straight parallel lines, perpendicular to the central meridian. Spacing is equal between the 38° parallels, decreasing outside these limits.

Poles: Lines 0.53 as long as the Equator.

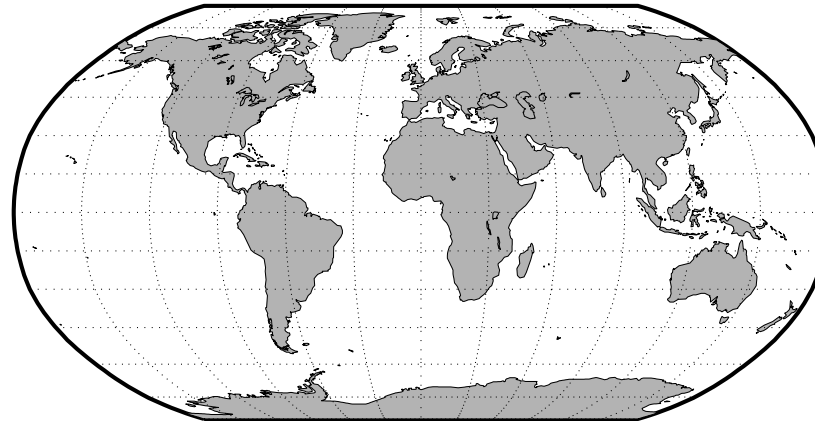
Symmetry: About the central meridian or the Equator.

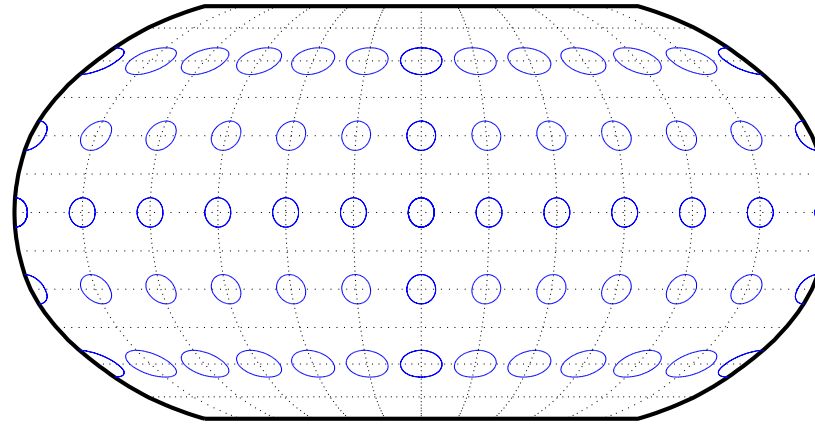
Features Scale is true along the 38° parallels and is constant along any parallel or between any pair of parallels equidistant from the Equator. It is not free of distortion at any point, but distortion is very low within about 45° of the center and along the Equator. This projection is not equal-area, conformal, or equidistant; however, it is considered to *look right* for world maps, and hence is widely used by Rand McNally, the National Geographic Society, and others. This feature is achieved through the use of tabular coordinates rather than mathematical formulae for the graticules.

Parallels For this projection, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. The standard parallel is by definition fixed at 38°.

Remarks This projection was presented by Arthur H. Robinson in 1963, and is also called the Orthophanic projection, which means *right appearing*.

Limitations This projection is available for the spherical geoid only.





Sinusoidal Projection

Classification Pseudocylindrical

Syntax `sinusoid`

Graticule Central Meridian: Straight line half as long as the Equator.

Other Meridians: Equally spaced sinusoidal curves intersecting at the poles and concave toward the central meridian.

Parallels: Equally spaced straight parallel lines, perpendicular to the central meridian.

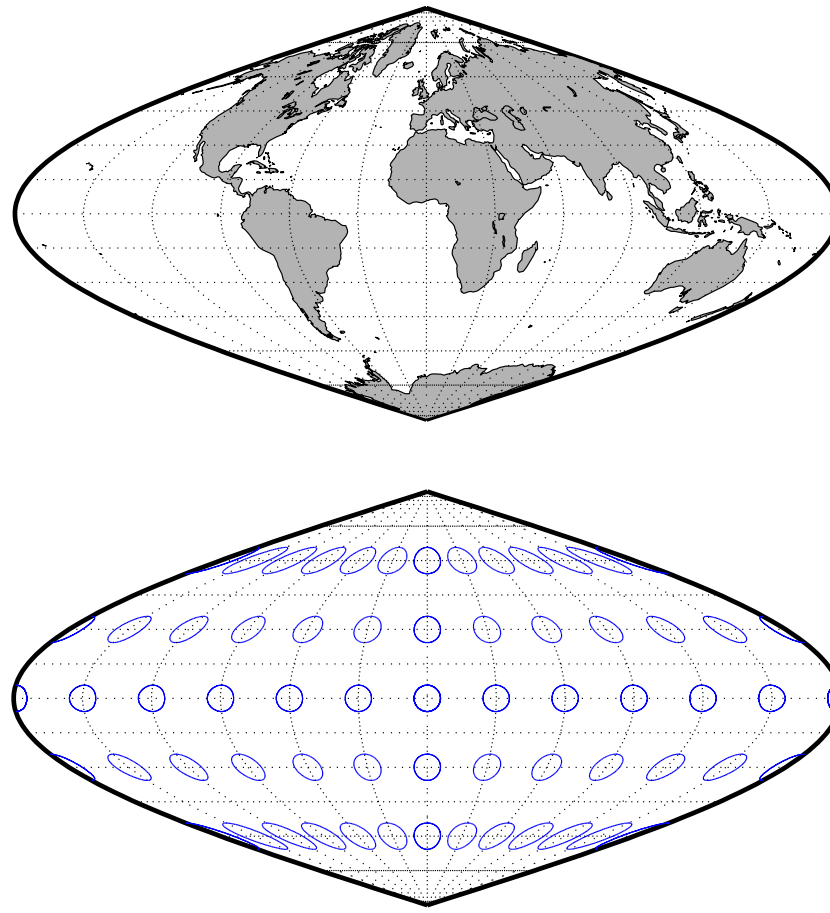
Poles: Points.

Symmetry: About the central meridian or the Equator.

Features This projection is equal-area. Scale is true along every parallel and along the central meridian. There is no distortion along the Equator or along the central meridian, but it becomes severe near the outer meridians at high latitudes.

Parallels This projection has one standard parallel, which is by definition fixed at 0°.

Remarks This projection was developed in the 16th century. It was used by Jean Cossin in 1570 and by Jodocus Hondius in Mercator atlases of the early 17th century. It is the oldest pseudocylindrical projection currently in use, and is sometimes called the Sanson-Flamsteed or the Mercator Equal-Area projection.



Stereographic Projection

Classification Azimuthal

Syntax `stereo`

Graticule The graticule described is for a polar aspect.

Meridians: Equally spaced straight lines intersecting at the central pole. The angles displayed are the true angles between meridians.

Parallels: Unequally spaced circles centered on the central pole. Spacing increases gradually away from this pole. The opposite hemisphere cannot be shown

Pole: The central pole is a point; the other pole is not shown.

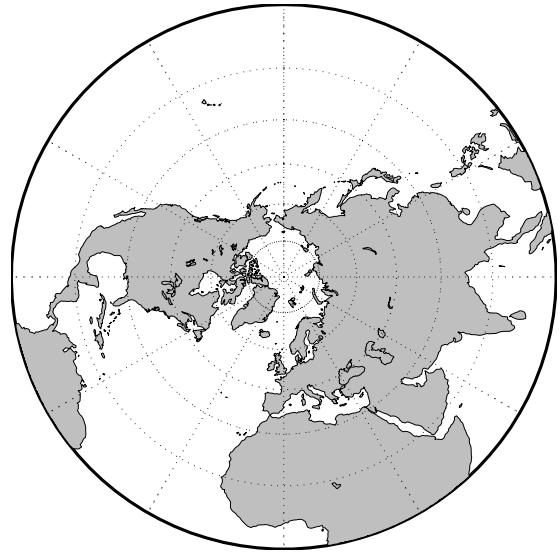
Symmetry: About any meridian.

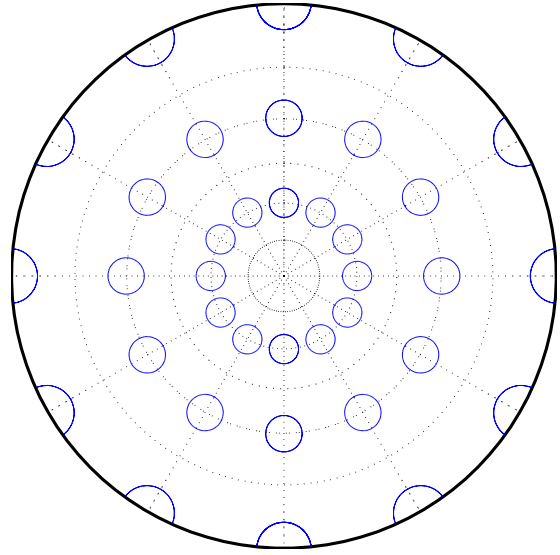
Features This is a perspective projection on a plane tangent at the center point from the point antipodal to the center point. The center point is a pole in the common polar aspect, but can be any point. This projection has two significant properties. It is conformal, being free from angular distortion. Additionally, all great and small circles are either straight lines or circular arcs on this projection. Scale is true only at the center point and is constant along any circle having the center point as its center. This projection is not equal-area.

Parallels There are no standard parallels for azimuthal projections.

Remarks The polar aspect of this projection appears to have been developed by the Egyptians and Greeks by the second century B.C.

Limitations Data greater than 90° distant from the center point is trimmed.





Tissot Modified Sinusoidal Projection

Classification Pseudocylindrical

Syntax `modsine`

Graticule Meridians: Sine curves converging at the Poles.

Parallels: Equally spaced straight lines.

Poles: Points.

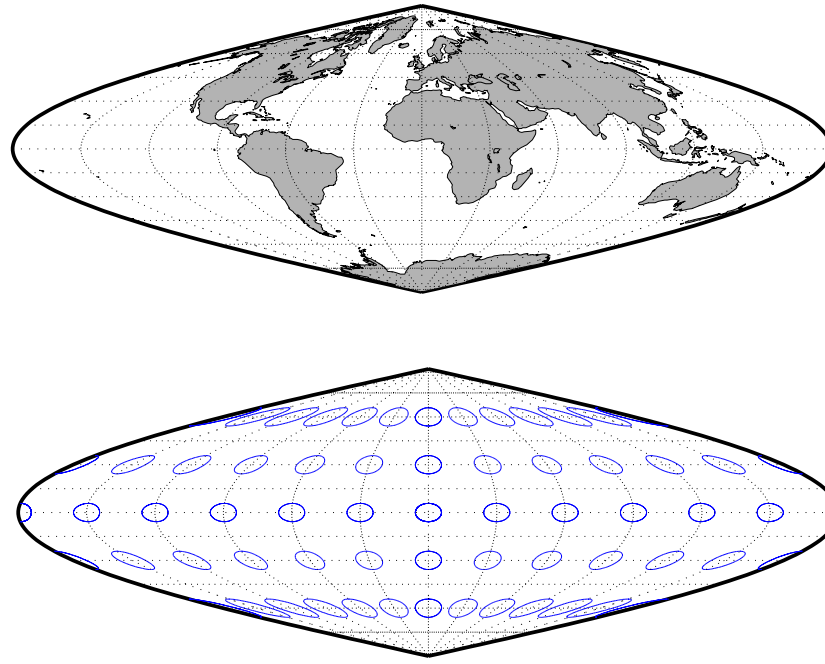
Symmetry: About the Equator and the central meridian

Features This is an equal-area projection. Scale is constant along any parallel or any pair of equidistant parallels, and along the central meridian. It is not equidistant or conformal.

Parallels There are no standard parallels for this projection

Remarks This projection was first described by N. A. Tissot in 1881

Limitations This projection is available for the spherical geoid only.



Transverse Mercator Projection

Classification Cylindrical

Syntax `tranmerc`

Features This conformal projection is the transverse form of the Mercator projection and is also known as the Gauss-Krueger pojection. It is not equal area, equidistant, or perspective.

The scale is constant along the central meridian, and increases to the east and west. The scale at the entral meridian can be set true to scale, or reduced slightly to render the mean scale of the overall map more nearly correct.

Remarks The uniformity of scale along its centeral meridian makes Transverse Mercator an excellent choice for mapping areas that are elongated north-to-south. Its best known application is the definition of Universal Transverse Mercator (UTM) coordinates. Each UTM zone spans only 6 degrees of longitude, but the northern half extends from the equator all the way to 84 degrees north and the southern half extends from 80 degrees south to the equator. Other map grids based on Transverse Mercator include many of the state plane zones in the U.S., and the U.K. National Grid.

Trystan Edwards Cylindrical Projection

Classification Cylindrical

Syntax `trystan`

Graticule Meridians: Equally spaced straight parallel lines.

Parallels: Unequally spaced straight parallel lines, perpendicular to the meridians. Spacing is closest near the poles.

Poles: Straight lines equal in length to the Equator.

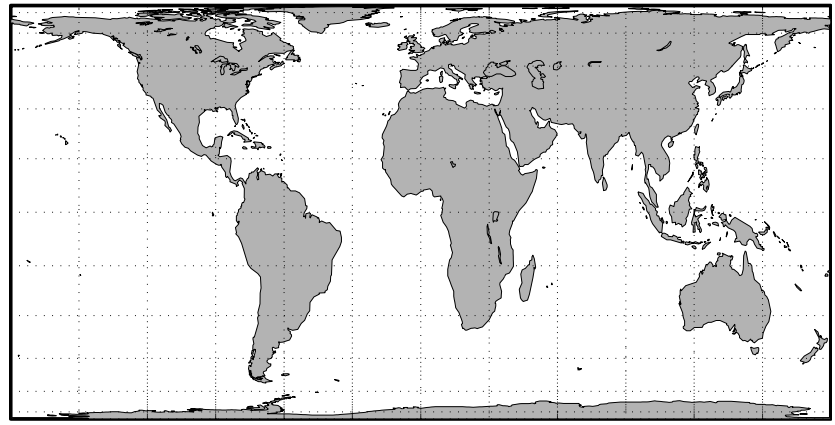
Symmetry: About any meridian or the Equator.

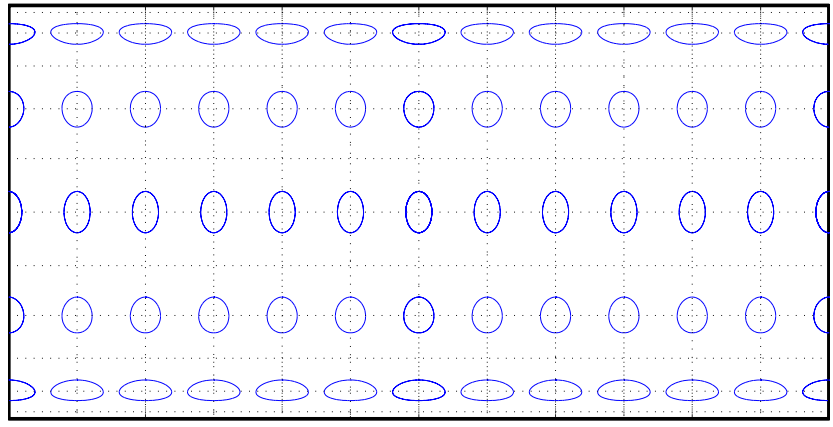
Features This is an orthographic projection onto a cylinder secant at the 37°24' parallels. It is equal-area, but distortion of shape increases with distance from the standard parallels. Scale is true along the standard parallels and constant between two parallels equidistant from the Equator. This projection is not equidistant.

Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, the standard parallel is by definition fixed at 37°24'.

Remarks This projection is named for Trystan Edwards, who presented it in 1953. It is a special form of the Equal-Area Cylindrical projection secant at 37°24'N and S.

Trystan Edwards Cylindrical Projection





Universal Polar Stereographic Projection

Classification Azimuthal

Syntax `ups`

Graticule The graticule described is for the southern zone.

Meridians: Equally spaced straight lines centered on the South Pole. The angles displayed are the true angles between meridians.

Parallels: Unequally spaced circles centered on the South Pole. Spacing increases gradually away from the circle of true scale along latitude 87 degrees, 7 minutes N. The opposite pole cannot be shown.

Poles: The South Pole is a point. The North Pole is not shown.

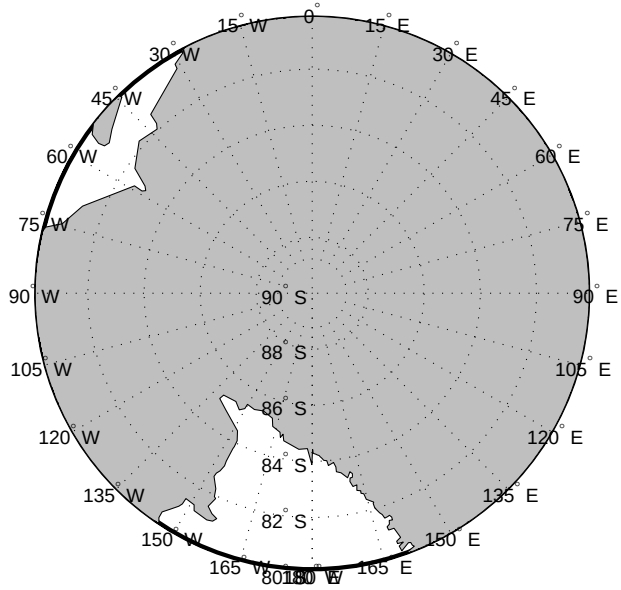
Symmetry: About any meridian.

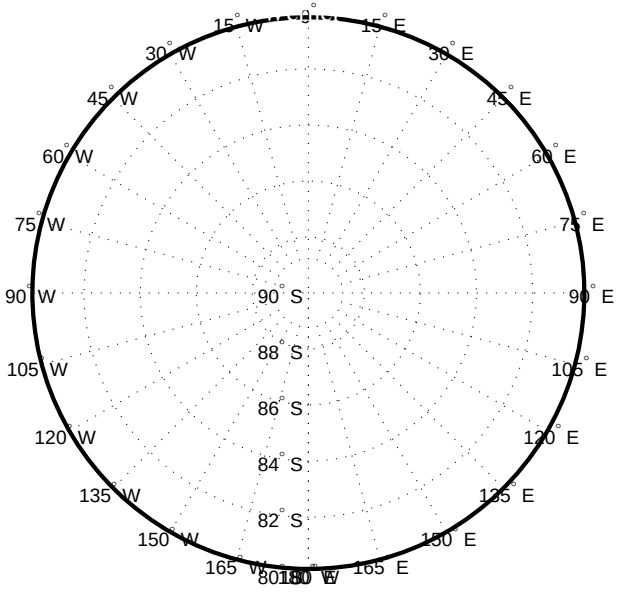
Features This is a perspective projection on a plane tangent to either the North or South Pole. It is conformal, being free from angular distortion. Additionally, all great and small circles are either straight lines or circular arcs on this projection. Scale is true along latitudes 87 degrees, 7 minutes N or S, and is constant along any other parallel. This projection is not equal area.

Parallels The parallels 87 degrees, 7 minutes N and S are lines of true scale by virtue of the scale factor. There are no standard parallels for azimuthal projections.

Remarks This projection is a special case of the stereographic projection in the polar aspect. It is used as part of the Universal Transverse Mercator (UTM) system to extend coverage to the poles. This projection has two zones: 'North' for latitudes 84° N to 90° N, and 'South' for latitudes 80° S to 90° S. The defaults for this projection are: scale factor is 0.994, false easting and northing are 2,000,000 meters. The international ellipsoid in units of meters is used as the geoid model.

Universal Polar Stereographic Projection





Universal Transverse Mercator Projection

Classification Cylindrical

Syntax `utm`

Graticule Meridians: Complex curves concave towards the central meridian.

Parallels: Complex curves concave towards the nearest pole.

Poles: Not shown.

Symmetry: About the central meridian or the Equator.

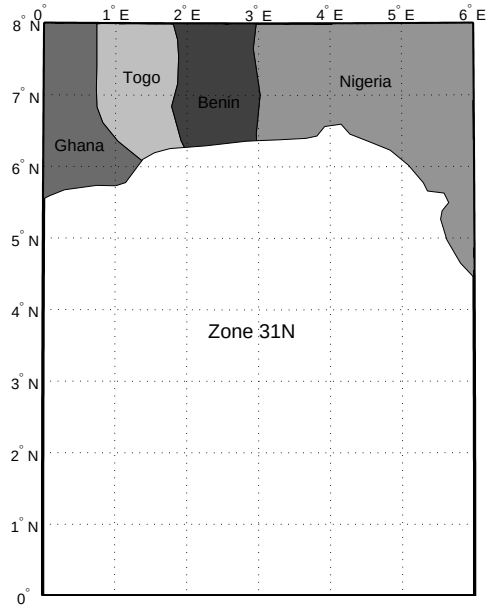
Features This is a conformal projection with parameters chosen to minimize distortion over a defined set of small areas. It is not equal area, equidistant, or perspective. Scale is true along two straight lines on the map approximately 180 kilometers east and west of the central meridian, and is constant along other straight lines equidistant from the central meridian. Scale is less than true between, and greater than true outside the lines of true scale.

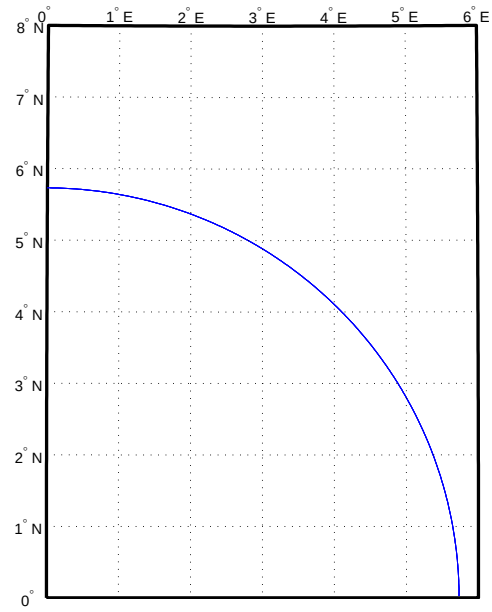
Parallels There are no standard parallels for this projection. There are two lines of zero distortion by virtue of the scale factor.

Remarks The UTM system divides the world between 80° S and 84° degrees N into a set of quadrangles called zones. Zones generally cover 6 degrees of longitude and 8 degrees of latitude. Each zone has a set of defined projection parameters, including central meridian, false eastings and northings and the reference ellipsoid. The projection equations are the Gauss-Krüger versions of the Transverse Mercator. The projected coordinates form a grid system, in which a location is specified by the zone, easting and northing.

The UTM system was introduced in the 1940's by the U.S. Army. It is widely used in topographic and military mapping.

Universal Transverse Mercator Projection





Van der Grinten I Projection

Classification Polyconic

Syntax `vgrint1`

Graticule Central Meridian: A straight line.

Meridians: Circular curves spaced equally along the equator and concave toward the central meridian.

Parallels: The Equator is a straight line. All other parallels are circular arcs concave toward the nearest pole.

Poles: Points.

Symmetry: About the Equator or the central meridian.

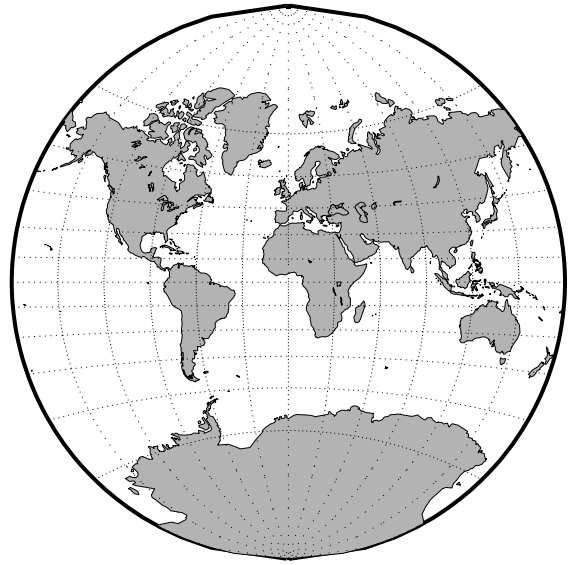
Features In this projection, the world is enclosed in a circle. Scale is true along the Equator and increases rapidly away from the Equator. Area distortion is extreme near the poles. This projection is neither conformal nor equal-area.

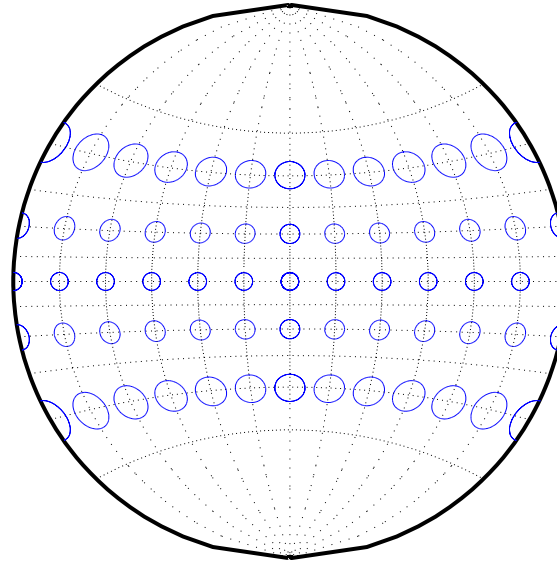
Parallels There are no standard parallels for this projection.

Remarks This projection was presented by Alphons J. Van der Grinten in 1898. He obtained a U.S. patent for it in 1904. It is also known simply as the Van der Grinten projection (without the “I”).

Limitations This projection is available for the spherical geoid only.

Van der Grinten I Projection





Vertical Perspective Azimuthal Projection

Classification Azimuthal

Syntax `vperspec`

Graticule The graticule described is for a polar aspect.

Meridians: Equally spaced straight lines intersecting at the central pole. The angles displayed are true angles between meridians.

Parallels: Unequally spaced circles centered on the central pole. Spacing decreases away from this pole. The opposite hemisphere cannot be shown, nor can distant parts of the closer hemisphere. The limit of visibility depends on the observation altitude.

Poles: The central pole is a point. The other pole is not shown.

Symmetry: About any meridian.

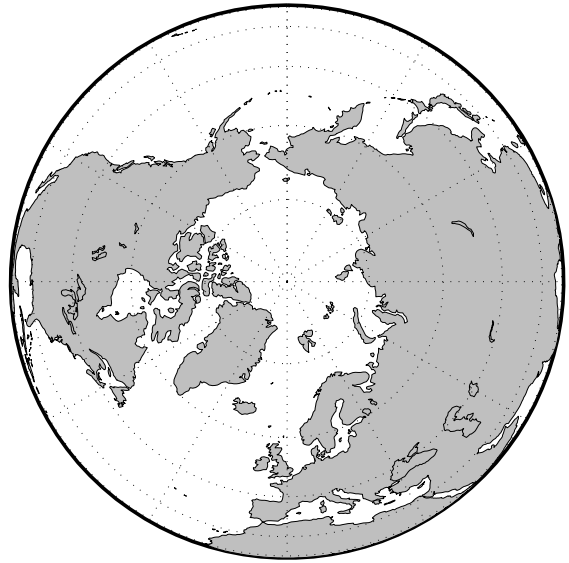
Features This is a perspective projection on a plane tangent at the center point from a finite distance. Scale is true only at the center point, and is constant in the circumferential direction along any circle having the center point as its center. Distortion increases rapidly away from the center point, the only point which is distortion free. This projection is neither conformal nor equal area.

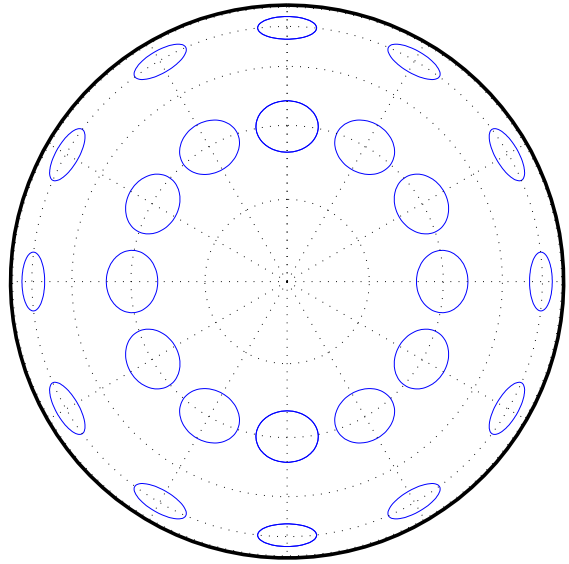
Parallels The standard parallel contains the observation altitude above the surface in the same units as the geoid semi-major axis.

Remarks This projection provides views of the globe resembling those seen from a spacecraft in orbit. The Orthographic projection is a limiting form with the observer at an infinite distance.

Limitations This projection is available for the spherical geoid only. Data more distant than the limit of visibility is trimmed.

Vertical Perspective Azimuthal Projection





Classification Pseudocylindrical

Syntax

```
wagner4
```

Graticule Central Meridian: Straight line half as long as the Equator.

Other Meridians: Equally spaced portions of ellipses concave toward the central meridian. The meridians 103°55' east and west of the central meridian are circular arcs.

Parallels: Unequally spaced straight parallel lines, perpendicular to the central meridian. Spacing is greatest toward the Equator.

Poles: Lines half as long as the Equator.

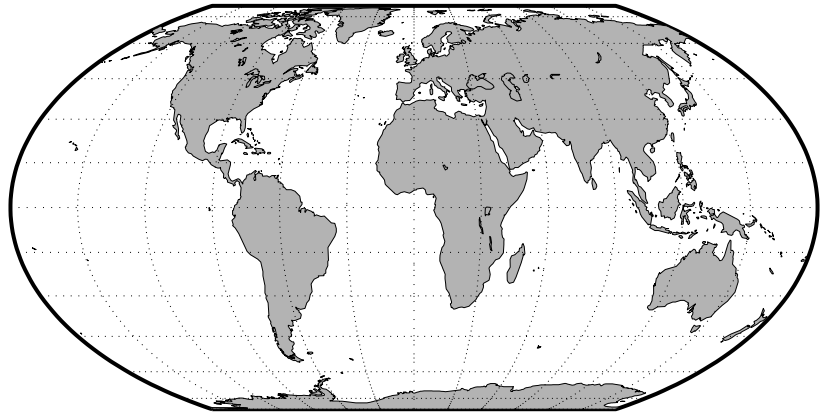
Symmetry: About the central meridian or the Equator.

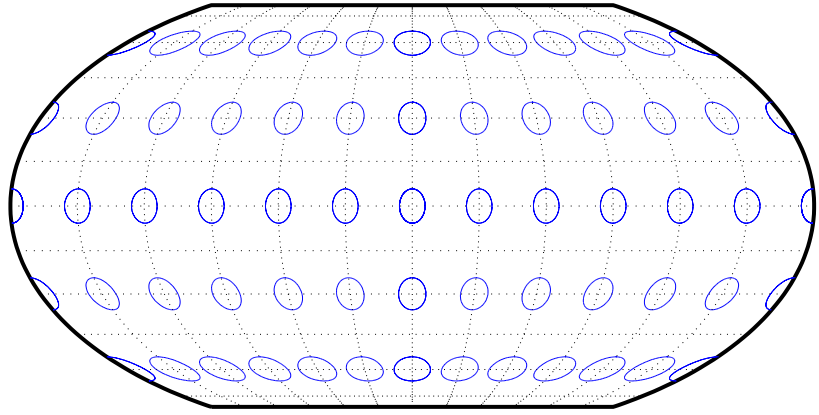
Features This is an equal-area projection. Scale is true along the 42°59' parallels and is constant along any parallel and between any pair of parallels equidistant from the Equator. Distortion is not as extreme near the outer meridians at high latitudes as for pointed-polar pseudocylindrical projections, but there is considerable distortion throughout the polar regions. It is free of distortion only at the two points where the 42°59' parallels intersect the central meridian. This projection is not conformal or equidistant.

Parallels For this projection, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. The standard parallel is by definition fixed at 42°59'.

Remarks This projection was presented by Karlheinz Wagner in 1932.

Wagner IV Projection





Classification Pseudoconic

Syntax `werner`

Graticule Central Meridian: A straight line.

Meridians: Complex curves connecting points equally spaced along each parallel and concave toward the central meridian.

Parallels: Concentric circular arcs spaced at true distances along the central meridian, centered on one of the poles.

Poles: Points.

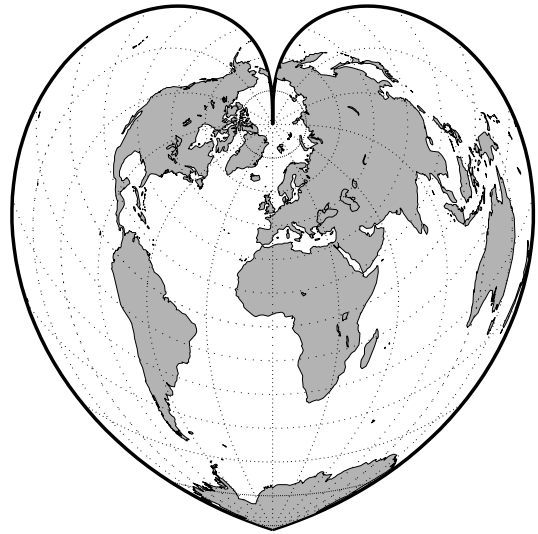
Symmetry: About the central meridian.

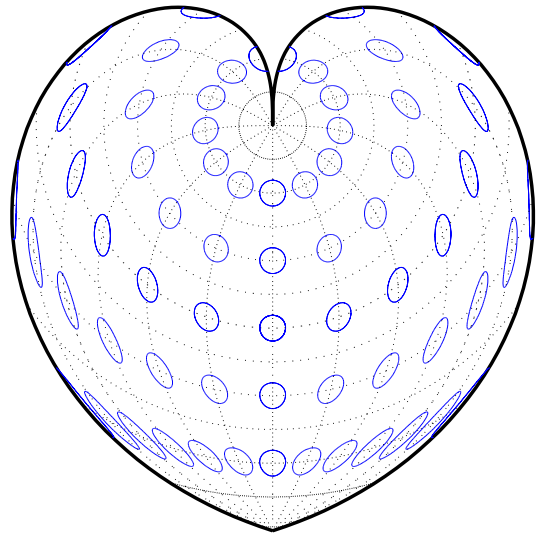
Features This is an equal-area projection. It is a Bonne projection with one of the poles as its standard parallel. The central meridian is free of distortion. This projection is not conformal. Its heart shape gives it the additional descriptor *cordiform*.

Parallels The standard parallel for this projection is set to 90° N.

Remarks This projection was developed by Johannes Stabius (Stab) about 1500 and was promoted by Johannes Werner in 1514. It is also called the Stab-Werner projection.

Werner Projection





Classification Cylindrical

Syntax

```
wetch
```

Graticule Central Meridian: Straight line (includes meridian opposite the central meridian in one continuous line).

Other Meridians: Straight lines if 90° from central meridian, complex curves concave toward the central meridian otherwise.

Parallels: Complex curves concave toward the nearest pole.

Poles: Points along the central meridian.

Symmetry: About any straight meridian or the Equator.

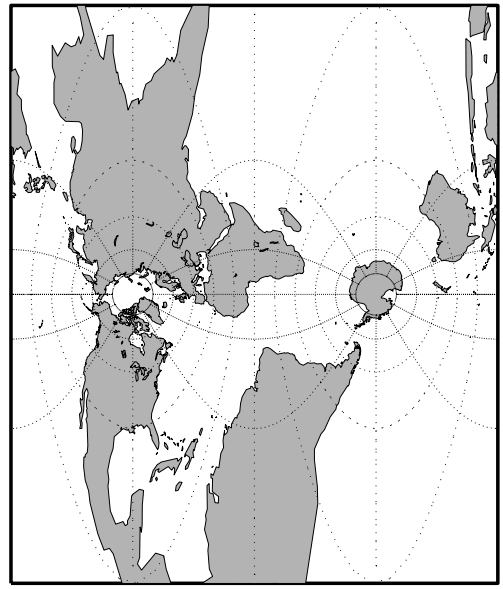
Features This is a perspective projection from the center of the Earth onto a cylinder tangent to the central meridian. It is not equal-area, equidistant, or conformal. Scale is true along the central meridian and constant between two points equidistant in x and y from the central meridian. There is no distortion along the central meridian, but it increases rapidly away from the central meridian in the y-direction.

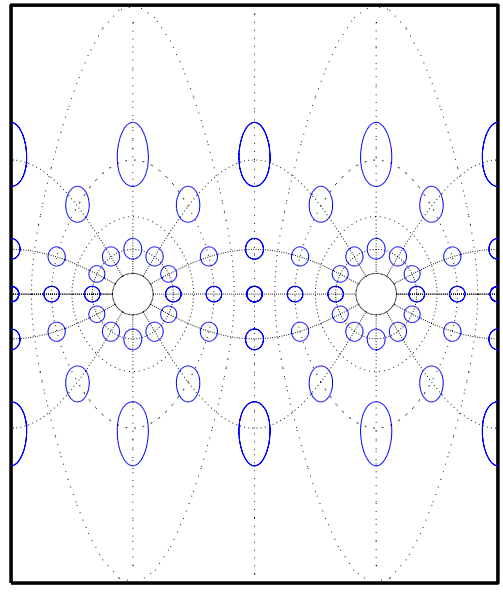
Parallels For cylindrical projections, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. For this projection, which is the transverse aspect of the Central Cylindrical, the standard parallel *of the base projection* is by definition fixed at 0°.

Remarks This is the transverse aspect of the Central Cylindrical projection discussed by J. Wetch in the early 19th century.

Limitations This projection is available for the spherical geoid only. To prevent large y-values from dominating the display, data at y-values that would correspond to latitudes of greater than 75° in the normal aspect of the Central Cylindrical projection is trimmed.

Wetch Cylindrical Projection





Wiechel Projection

Classification Pseudoazimuthal

Syntax `wiechel`

Graticule The graticule described is for a polar aspect.

Meridians: Equally spaced semicircles from pole to pole, concave toward the west.

Parallels: Concentric circles.

Pole: The central pole is a point; the other pole is a bounding circle.

Symmetry: Radially about the center point.

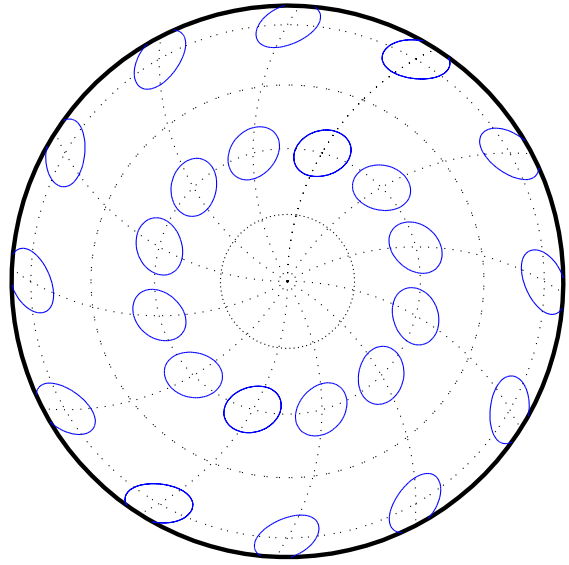
Features This equal-area projection is a novelty map, usually centered at a pole, showing semicircular meridians in a pinwheel arrangement. Scale is correct along the meridians. This projection is not conformal.

Parallels There are no standard parallels for azimuthal projections.

Remarks This projection was presented by H. Wiechel in 1879.

Limitations Data greater than 65° distant from the center point is trimmed.

Wiechel Projection



Classification Pseudocylindrical

Syntax `winkel`

Graticule Central Meridian: Straight line at least half as long as the Equator.

Other Meridians: Equally spaced sinusoidal curves concave toward the central meridian.

Parallels: Equally spaced straight parallel lines, perpendicular to the central meridian.

Poles: Lines at least half as long as the Equator.

Symmetry: About the central meridian or the Equator.

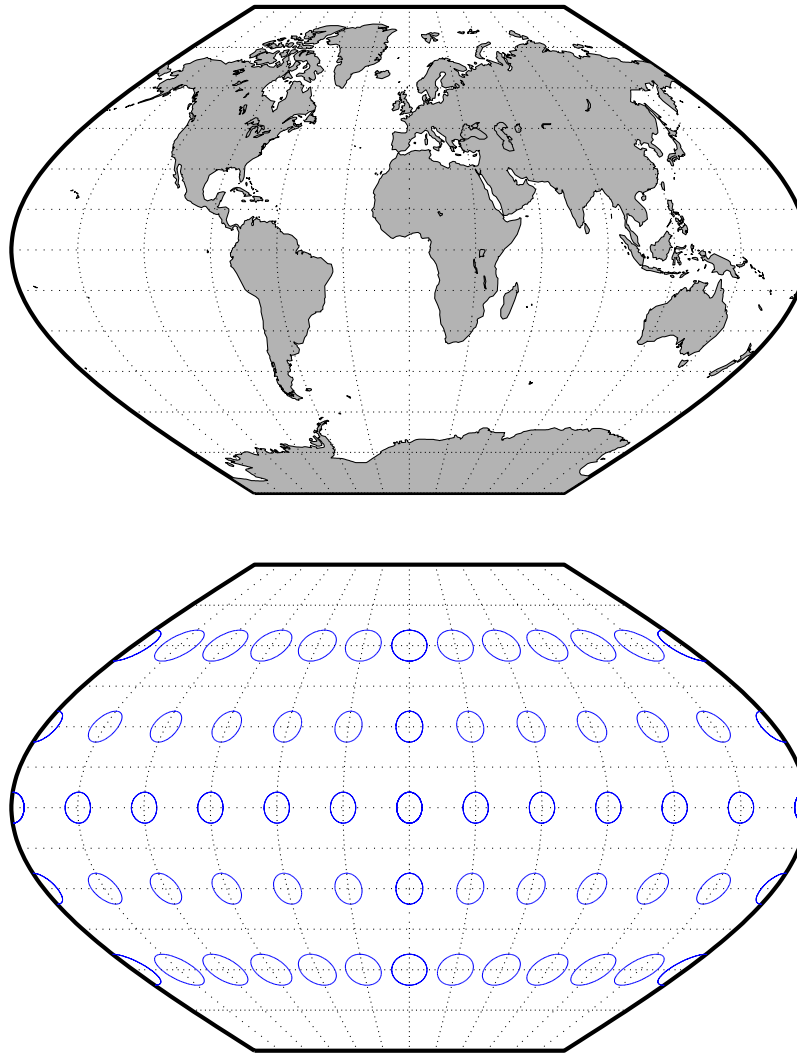
Features This projection is an arimethical average of the x and y coordinates of the Sinusoidal and Equidistant Cylindrical projections. Scale is true along the standard parallels and is constant along any parallel and between any pair of parallels equidistant from the Equator. There is no point free of distortion. This projection is not equal-area, conformal, or equidistant.

Parallels For this projection, only one standard parallel is specified. The other standard parallel is the same latitude with the opposite sign. Any latitude may be chosen; the default is set to 50°28'.

Remarks This projection was developed by Oswald Winkel in 1914. Its limiting form is the Eckert V when a standard parallel of 0° is chosen.

Limitations This projection is available for the spherical geoid only.

Winkel I Projection



12

GUI Reference

Graphical User Interface Functions — Categorical List

Map Definition Tools

`axesm, axesmui`	Define map axes and display property setting.
`originui`	Allow interactive modification of map origin.
`parallelui`	Tool for interactively modifying map parallels

Mapping Tools

`maptool`	Create figure window with menu-activated mapping tools.
`mapviewer`	Interactive map viewer for projected geodata
`mlayers`	Manipulate layers of a map.
`mobjects`	Manipulate mapped object sets.
`qrydata`	Allow interactive queries of map data.

Object Projection Tools

`linem`, `plotm`, `plot3m`	Project line objects onto map axes.
`fillm`, `fill3m`, `patchm`, `patchesm`	Project patch objects onto map axes.
`patchesm`	Project patches as individual objects onto map axes.
`meshm`	Warp regular matrix map onto projected graticule mesh.
`pcolorm`, `surfacem`, `surfm`	Warp general matrix map onto projected graticule mesh.
`lightm`	Project light objects onto map axes.
`surflm`	Project 3-D shaded surface with lighting onto map axes.
`meshlsrm` `surflsrm`	Project 3-D lighted shaded surface onto map axes.
`textm`	Project text objects onto map axes.

Display Manipulation Tools

`clrmenu`	Add colormap menu to figure.
`panzoom`	Pan and zoom on 2-D map display.

Thematic Map Tools

`cometm, comet3m`	Project comet plot onto map axes.
`contourm, contour3m`	Project contour plot onto map axes.
`quiverm`	Project 2-D quiver plot onto map axes.
`quiver3m`	Project 3-D quiver plot onto map axes.
`stem3m`	Project stem plot onto map axes.
`scatterm`	Project proportional symbol map onto map axes.

Object Property Tools

`clmo`	Clear specified map objects from map axes.
`cmapui`	Create custom colormap.
`colorui`	Set custom RGB color triples.
`handlem`	Return handles of specified map objects.
`hidem`	Hide specified map objects.
`showm`	Show specified map objects.
`tagm`	Edit tag of specified map objects.
`zdatam`	Edit z-plane of specified map objects.
`property editors`	Edit properties of specified map objects.

Track Tools

`scircleg`	Manipulate small circles on map axes.
`scirclui`	Display small circles on map axes.
`sectorg`	Manipulate sectors of small circles on map axes.
`surfdist`	Calculate distance, azimuth, and reckoning.
`trackg`	Manipulate great circle and rhumb line tracks on map axes.
`trackui`	Display great circle and rhumb line tracks on map axes.

Map Data Construction Tools

`colorm`	Create colormaps for regular matrix map.
`limitm`	Compute latitude and longitude limits of regular matrix map.
`maptrim`	Allow interactive trimming of map data.
`seedm`	Encode regular matrix map.

Graphical User Interface Functions — Alphabetical List

The following GUI Reference pages are organized alphabetically by the name of the function or tool. Most of the GUI tools in the Mapping Toolbox are activated by command-line functions without any input arguments. Users should consult the corresponding pages in Chapter 10, "Reference" as well. The entries in this chapter contain the following:

Purpose	Provides a short, concise description of the tool.
Activation	Provides methods of activation from the MATLAB command window, by mouse interaction with the map display, and/or from within `maptool`.
Description	Describes what the tool does, how each activation method works, and any rules or restrictions that apply.
Controls	Describes how to use the interface associated with the tool, including dialog boxes, menu bars, etc.
Examples	Provides examples of how the tool can be used.
See Also	Refers users to other related command, functions, or tools.

Purpose

Define map axes and modify map projection and display properties

Activation

Command Line	Maptool	Map Display

`axesm`
`axesm(h)`
`axesmui`
`axesmui(h)`
`c = axesmui(...)`
	Display⇒Projection	extend-click on map display

Description

`axesm` activates a **Projection Control** dialog box, which allows map projection definition and property modification. If no map is currently defined, `axesm` creates a map axes with the Robinson projection as the default.

`axesm(h)` activates the **Projection Control** box for the axes specified by the handle `h`.

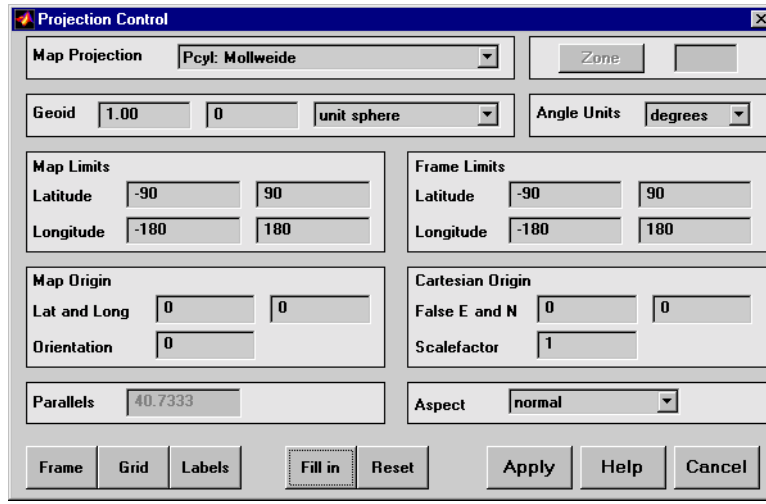
`axesmui` activates the **Projection Control** box for the current map axes.

`axesmui(h)` activates the **Projection Control** box for the map axes specified by the handle `h`.

`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 on 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 the UTM or UPS zone. For non-UTM and UPS projections, the two are disabled.

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 Greenwich 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 Greenwich 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'`.

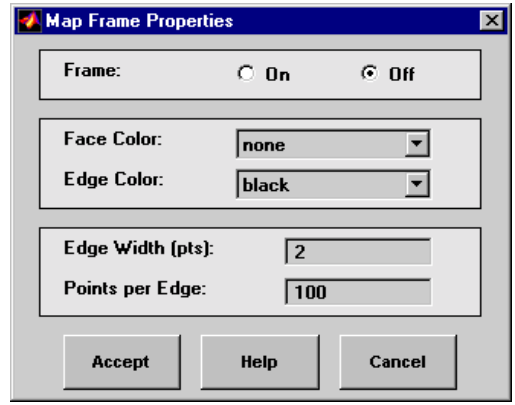
The **Apply** button is used to apply the projection and display settings to the current map, which results in the map being reprojected.

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.

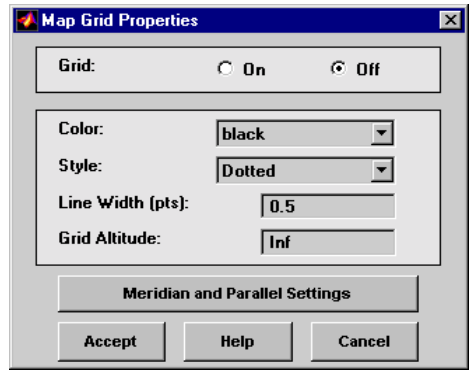
The **Points per Edge** edit box is used to enter the number of points used to display each edge of the map frame.

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.

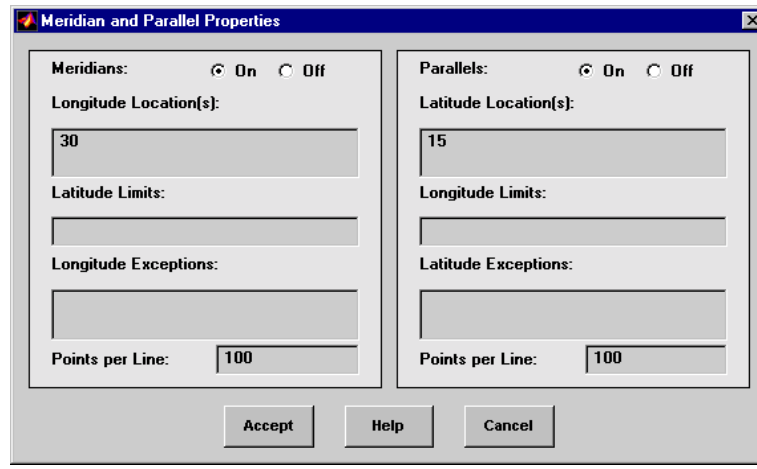
The **Meridian and Parallel Settings** button brings up the **Meridian and Parallel Properties** dialog box, which allows the properties of the meridian and parallel grid lines to be modified.

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 bedisplayed 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.

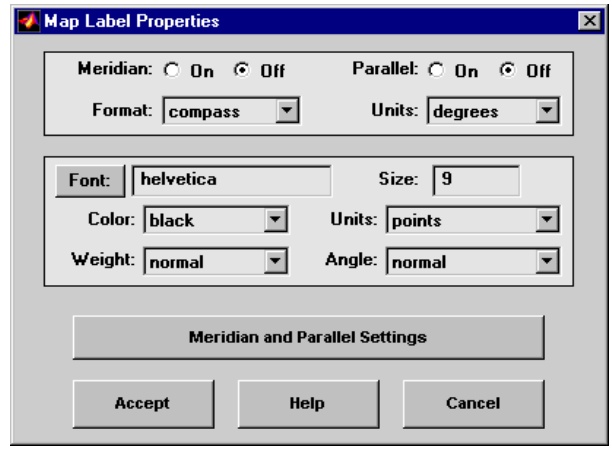
The **Points per Line** edit boxes are used to enter the number of points used to plot each meridian and each parallel grid line. The default value is 100 points.

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 non-italic font. `italic` and `oblique` specify italic font.

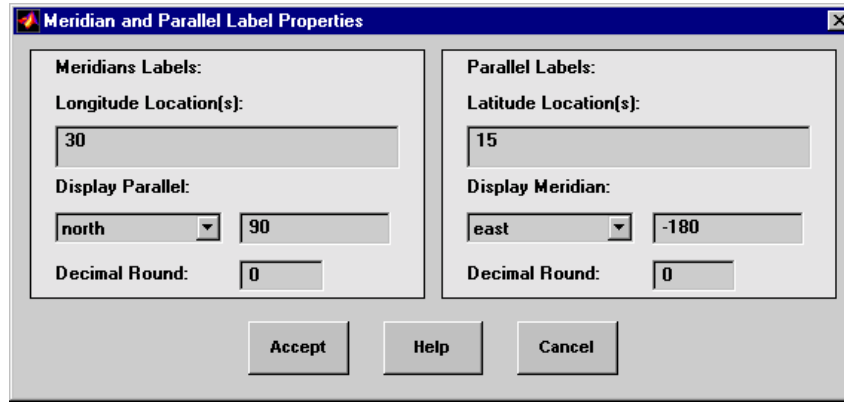
The **Meridian and Parallel Settings** button brings up the **Meridian and Parallel Label Properties** dialog box, which allows modification of properties specific to the meridian and parallel grid labels.

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 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`

Purpose

Clear mapped objects

Activation

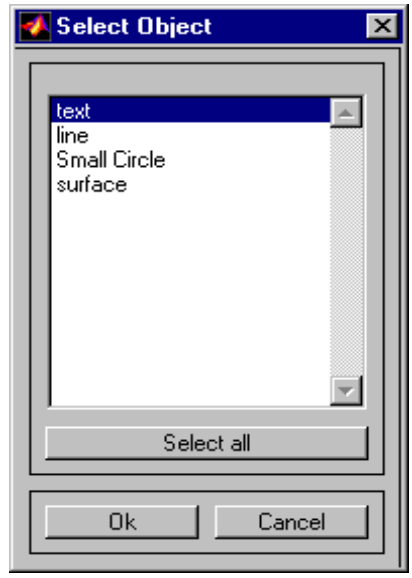
Command Line	Maptool
`clmo`	**Tools⇒Delete⇒Object**

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`

clrmenu

Purpose

Add a colormap menu to a figure

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:

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.

Spin spins the colormap.

Define allows a workspace variable to be specified for the colormap.

Digital Elevation activates the DEM Color Map Input dialog box associated with the `demcmap` tool. This tool is used to create a colormap for a digital elevation map.

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.

See Also

`colorm`

Purpose

Create colormaps for an indexed regular data grid

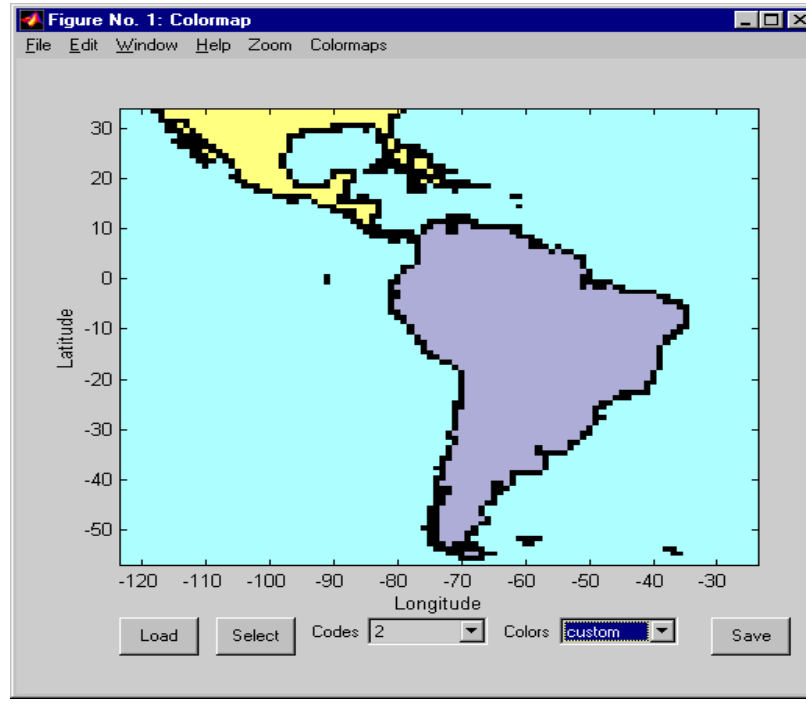
Activation

Command Line
`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 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 the `panzoom` box on and off. The box can be moved by clicking on 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.

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` `panzoom` `seedm`

cometm, comet3m

Purpose

Project animated 2-D and 3-D comet plots on the current map axes

Activation

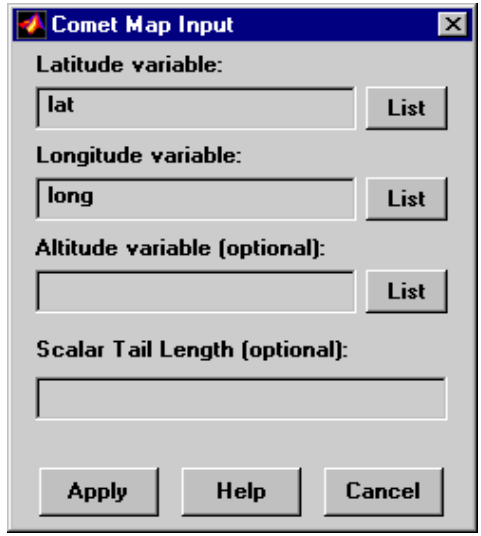
Command Line	Maptool
`cometm` `comet3m`	**Map⇒Comet**

Description

`cometm` and `comet3m` activate a **Comet Map Input** dialog box for projecting comet plots onto the current map axes.

If no map axes are current, a **No Map Axes** dialog box appears. Choose **Yes** to activate the **Projection Control** dialog box for defining map axes properties. Upon creation of the map axes, the **Comet Map Input** dialog box appears.

Controls



The **Latitude variable** edit box is used to specify the workspace variable containing the latitude data for the comet plot.

The **Longitude variable** edit box is used to specify the workspace variable containing the longitude data for the comet plot.

The **Altitude variable** edit box is used to specify the workspace variable containing the altitude data for the comet plot.

Pressing the **List** button produces a list of all current workspace variables, from which the latitude, longitude, and altitude variables can be selected.

The **Scalar Tail Length** edit box is used to enter a scalar value between 0 and 1 for the length of the comet tail. The default value is 0.1.

Pressing the **Apply** button accepts the input data and projects the comet plot onto the current map axes.

Pressing the **Cancel** button disregards any input data and closes the **Comet Map Input** dialog box.

See Also

`cometm, comet3m`

Purpose

Project 2-D and 3-D contour plots onto the current map axes

Activation

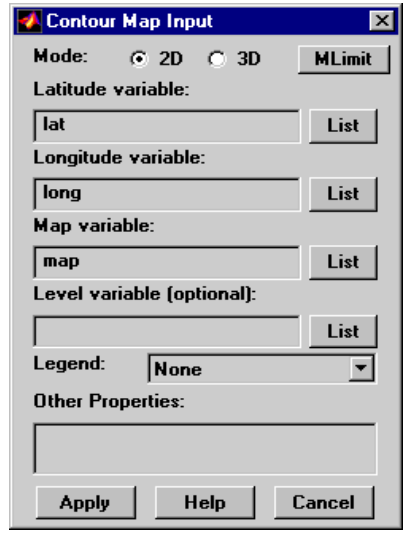
Command Line	Maptool
`contourm` `contour3m`	**Map⇒Contours**

Description

`contourm` and `contour3m` activate a **Contour Map Input** dialog box to project contour lines onto the current map axes.

If no map axes are current, a **No Map Axes** dialog box appears. Choose **Yes** to activate the **Projection Control** dialog box for defining map axes properties. Upon creation of the map axes, the **Contour Map Input** dialog box appears.

Controls



The **Mode** selection buttons are used to indicate a two- or three-dimensional contour plot.

The **MLimit** button brings up a **Map Limit Input** dialog box that computes the limits of a regular data grid and stores them as variables that can be used as the latitude and longitude inputs for the contour plot. This enables the creation

of contour plots for regular data grids. See `limitm` in this guide for more information about the **Map Limit Input** dialog box.

The **Latitude variable** edit box is used to specify the workspace variable containing the latitude vector or matrix for the contour plot. If a vector, it should be monotonically increasing and describe the latitude of each row of the data grid. If a matrix, it should be the size of the map matrix and give the latitude associated with each map matrix element.

The **Longitude variable** edit box is used to specify the workspace variable containing the longitude vector or matrix for the contour plot. If a vector, it should be monotonically increasing and describe the longitude of each column of the data grid. If a matrix, it should be the size of the map matrix and give the longitude associated with each map matrix element.

The **Map variable** edit box is used to specify the workspace variable containing the data grid.

The **Level variable** edit box is used to specify the workspace variable containing the values of the contours to be plotted. A vector of contour level values, enclosed in brackets, can be entered instead of a variable name. If omitted, the contour values are chosen automatically.

Pressing the **List** button produces a list of all current workspace variables, from which the latitude, longitude, map, and level variables can be selected.

The **Legend** pull-down menu is used to select the type of contour labeling or legend to be added to the plot. If the `Plot Legend` option is selected, any existing legend is deleted.

The **Other Properties** edit box is used to specify additional properties of the contour lines, such as `'Color','b'`. String entries must be enclosed in quotes. Linespec strings, such as `'c-'`, are also valid entries.

Pressing the **Apply** button accepts the input data and projects the contour plot onto the current map axes.

Pressing the **Cancel** button disregards any input data and closes the **Contour Map Input** dialog box.

See Also

`contourm, contour3m`

demcmap

Purpose

Create and assign a colormap to a digital elevation data grid

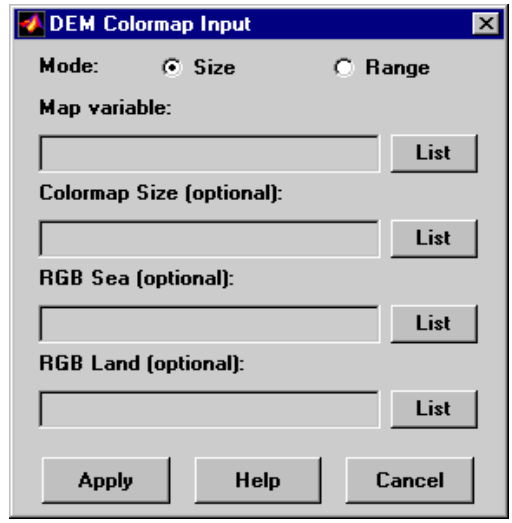
Activation

Command Line	Maptool
`demcmap`	**Colormaps⇒Digital Elevation**

Description

`demcmap` activates the **DEM Color Map Input** dialog box, which accepts inputs used to create a colormap for a digital elevation data grid, and then applies 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.

Controls



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 **Color Map 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 the string `'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 the string `'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 Color Map Input** dialog box.

See Also

`demcmap`

fillm, fill3m, patchm, patchesm

Purpose

Project patch objects on the current map axes

Activation

Command Line	Maptool

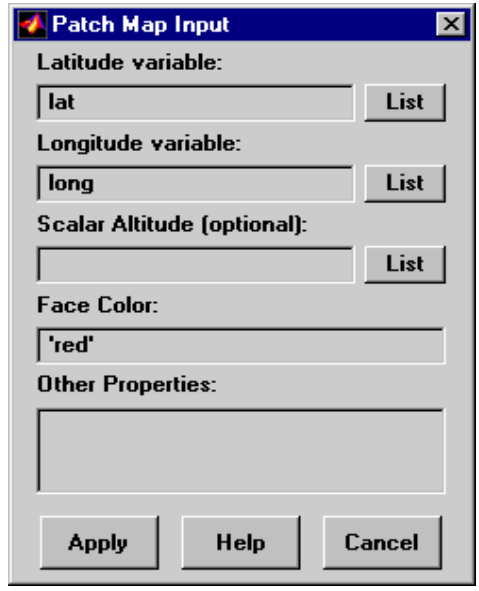
`fillm`
`fill3m`
`patchm`
`patchesm`
	Map⇒Patch

Description

`fillm`, `fill3m`, `patchm`, and `patchesm` all activate a **Patch Map Input** dialog box that accepts input data to project a patch object onto the current map axes.

If no map axes are current, a **No Map Axes** dialog box appears. Choose **Yes** to activate the **Projection Control** dialog box for defining map axes properties. Upon creation of the map axes, the **Patch Map Input** dialog box appears.

Controls



The **Latitude variable** edit box is used to specify the workspace variable containing the latitude data of the patch object to be projected.

The **Longitude variable** edit box is used to specify the workspace variable containing the longitude data of the patch object to be projected.

The **Scalar Altitude** edit box is used to specify a scalar value or scalar workspace variable that determines the plane in which the mapped patch object is to be displayed.

Pressing the **List** button produces a list of all current workspace variables, from which the latitude, longitude, and altitude variables can be selected.

The **Face Color** edit box is used to specify the color of the patch face. A valid color string, enclosed in quotes, or an RGB triple enclosed in brackets, can be entered. A workspace variable can also be entered, provided it is a color string or an RGB triple.

The **Other Properties** edit box is used to specify additional properties of the patch object to be projected, such as `'EdgeColor','none'`. String entries must be enclosed in quotes.

Pressing the **Apply** button accepts the input data and projects the patch object onto the current map axes.

Pressing the **Cancel** button disregards any input data and closes the **Patch Map Input** dialog box.

See Also

`fillm` `fill3m` `patchm` `patchesm`

handlem

Purpose

Return handles of mapped objects

Activation

Command Line:h = handlem
`h = handlem('prompt')`

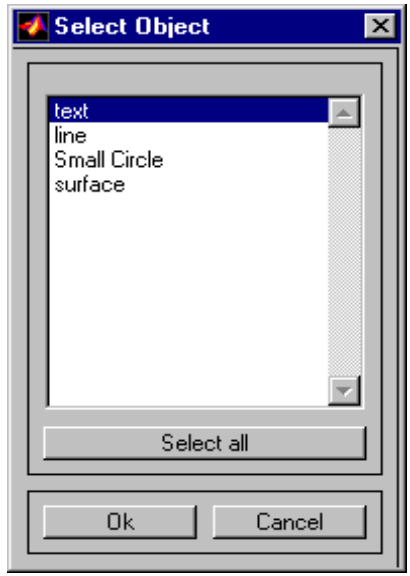
Description

`h = handlem` brings up a **Select Object** dialog box, which lists all currently displayed objects. Objects can be selected and their handles returned.

`h = handlem('prompt')` brings up a **Specify Object** dialog box, which allows greater control of object selection.

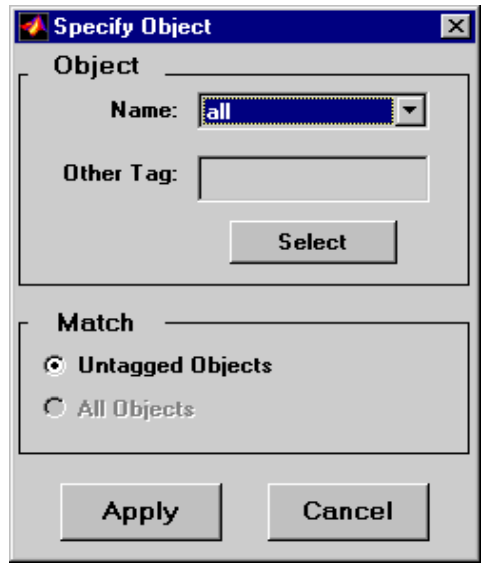
Controls

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 handles 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 object strings. 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 Handle Graphics object type (image, line, surface, patch, or text) is specified. The **Untagged Objects** selection button is used to return the handles of only those objects with empty tag properties. The **All Objects** selection button is used to return all object handles of the specified type, regardless of whether they are tagged.

Pushing the **Apply** button returns the handles of the specified objects. Pushing the **Cancel** button aborts the operation.

See Also

`handlem`

hidem

Purpose

Hide mapped objects

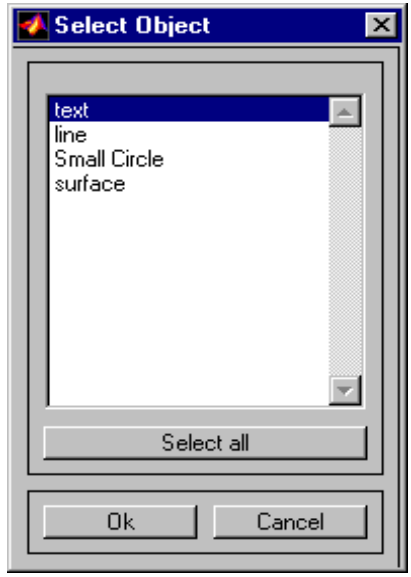
Activation

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`

Purpose

Project light objects on the current map axes

Activation

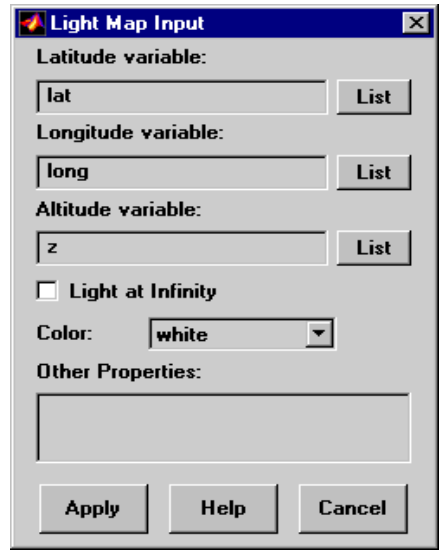
Command Line	Maptool
`lightm`	**Map⇒Light**

Description

`lightm` activates a **Light Map Input** dialog box for projecting a light object onto the current map axes.

If no map axes are current, a **No Map Axes** dialog box appears. Choose **Yes** to activate the **Projection Control** dialog box for defining map axes properties. Upon creation of the map axes, the **Light Map Input** dialog box appears.

Controls



The **Latitude variable** edit box is used to specify the workspace variable containing the latitude data of the light object to be projected.

The **Longitude variable** edit box is used to specify the workspace variable containing the longitude data of the light object to be projected.

The **Altitude variable** edit box is used to specify the workspace variable containing the altitude data of the light object to be projected. A scalar value can be entered to indicate at which height above the map the light object is to be displayed. This entry has no effect if an infinite light source is specified by the **Light at Infinity** check box.

Pressing the **List** button produces a list of all current workspace variables, from which the latitude, longitude, and altitude variables can be selected.

The **Light At Infinity** check box is used to specify a parallel or divergent light source. If the box is checked, the light source is at infinity, in which case the light rays are parallel. If the box is not checked, the altitude of the light source is specified by the altitude variable, and the light rays diverge in all directions. If this box is checked, the altitude variable has no effect.

The **Color** pull-down menu is used to specify the color of the light coming from the light object. Selecting `custom` allows a custom RGB triple to be defined.

The **Other Properties** edit box is used to specify additional properties of the light object to be projected, such as `'Tag','Blue Light'`. String entries must be enclosed in quotes.

Pressing the **Apply** button accepts the input data and projects the lighted object onto the current map axes.

Pressing the **Cancel** button disregards any input data and closes the **Light Map Input** dialog box.

See Also

`lightm`

Purpose

Compute latitude and longitude limits for a regular data grid

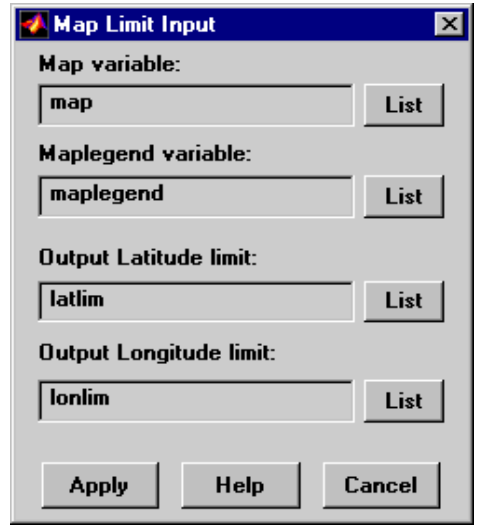
Activation

Command Line	Maptool
`limitm`	**Map⇒Contours**

Description

`limitm` activates the **Map Limit Input** dialog box, which allows the limits of a regular data grid to be computed. These limits are then stored in the workspace as variables.

Controls



The **Map variable** edit box is used to specify the workspace variable containing the regular data grid.

The **Maplegend variable** is used to specify the workspace variable containing the referencing vector. A three-element referencing vector, enclosed in brackets, can be entered instead of a variable name.

The **Output Latitude limit** edit box is used to specify the name of the variable that stores the computed latitude limits of the data grid. If this variable name already exists in the workspace, it is overwritten.

The **Output Longitude limit** edit box is used to specify the name of the variable that stores the computed longitude limits of the data grid. If this variable already exists in the workspace, it is overwritten.

Pressing the **List** button produces a list of all current workspace variables, from which the map, referencing vector, output latitude, and output longitude variables can be selected.

Pressing the **Apply** button calculates the limits of the data grid and stores the results in the specified output variables.

Pressing the **Cancel** button disregards any input data and closes the **Map Limit Input** dialog box.

See Also

`limitm`

Purpose

Project 2-D and 3-D line objects on the current map axes

Activation

Command Line	Maptool

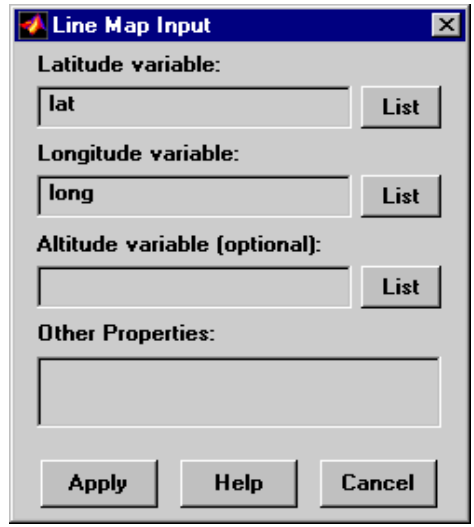
`linem`
`plotm`
`plot3m`
	Map⇒Lines

Description

`linem`, `plotm` and `plot3m` activate a **Line Map Input** dialog box that accepts input data to project a line object onto the current map axes.

If no map axes are current, a **No Map Axes** dialog box appears. Choose **Yes** to activate the **Projection Control** dialog box for defining map axes properties. Upon creation of the map axes, the **Line Map Input** dialog box appears.

Controls



The **Latitude variable** edit box is used to specify the workspace variable containing the latitude data of the line object to be projected.

The **Longitude variable** edit box is used to specify the workspace variable containing the longitude data of the line object to be projected.

The **Altitude variable** edit box is used to specify the workspace variable containing the altitude data of the line object to be projected. A scalar value can be entered to indicate the plane in which to display the object.

Pressing the **List** button produces a list of all current workspace variables, from which the latitude, longitude, and altitude variables can be selected.

The **Other Properties** edit box is used to specify additional properties of the line object to be projected, such as `'LineWidth',2`. String entries must be enclosed in quotes. Linespec strings, such as `'b:'`, are also valid.

Pressing the **Apply** button accepts the input data and projects the line object onto the current map axes.

Pressing the **Cancel** button disregards any input data and closes the **Line Map Input** dialog box.

See Also

`linem` `plotm` `plot3m`

Purpose

Create a figure window with a map axes and associated mapping tools

Activation

Command Line

`maptool(`*`PropertyName`*`,PropertyValue)`
`maptool(`*`ProjectionFile`*`,...)`
`h = maptool(...)`

Description

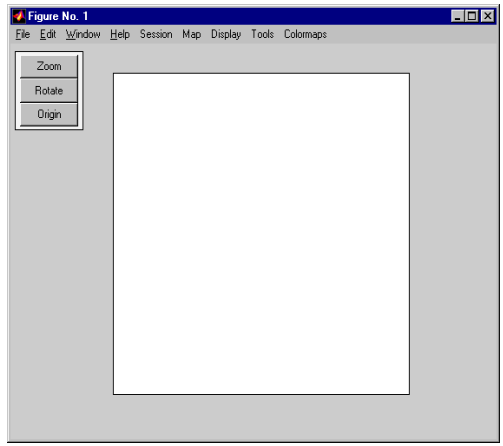
`maptool` creates a figure window with a map axes and activates the **Projection Control** dialog box for defining map projection and display properties. The figure window features a special menu bar that provides access to most of the Mapping Toolbox GUIs.

`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 identifying string 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



Session Menu

The **Load** option is used to load workspace data. Select from the workspace names provided, or use the **Specify Workspace** option to enter a different workspace.

The **Layers** option is used to load a map layers workspace and activate the `mlayers` tool. Select from the workspace names provided, or use the **Other** option to enter a different workspace. Choosing **Workspace** loads all structure variables in the current workspace.

The **Renderer** option is used to set the renderer for the maptool figure window. The **Figure Renderer** dialog box is activated when this option is selected.

The **Variables** option is used to view the current workspace variables.

The **Command** option brings up the **Workspace Commands** dialog box for entering commands to operate on the current workspace.

The **Clear** option is used to clear variables and functions from memory.

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 **Comet** option activates the **Comet Map Input** dialog box for a projecting two- or three-dimensional comet plot 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.

The **Light** option activates the **Light Map Input** dialog box for projecting light 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 **Zoom Tool** option is used to toggle Panzoom (`panzoom`) mode on and off. It is used for zooming in on a two-dimensional map display.

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 **Rotate** option is used to toggle Rotate 3-D (`rotate3d`) mode on and off. Rotate 3-D mode is used to interactively rotate the view of a three-dimensional plot.

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 Guide Property Editor to manipulate properties of a plotted object. Choose from the **Current Object** or **Last Object** options, or choose the **Object** option to activate the **Select Object** dialog box.

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.

Zoom Button

The Zoom button toggles Zoom mode on and off. Zoom mode is used for zooming in on a two-dimensional map display.

Rotate Button

The Rotate button toggles Rotate 3-D mode on and off. Rotate 3-D mode is used to interactively rotate the view of a three-dimensional plot.

Origin Button

The **Origin** button toggles Origin mode on and off. Origin mode is used to interactively modify the map origin.

See Also `maptool` `tools`

Purpose

Interactively trim and convert map data from vector to matrix format

Activation

Command Line

`maptrim(lat,lon)`
`maptrim(lat,lon,`*`linespec`*`)`
`maptrim(datagrid,refvec)`
`maptrim(datagrid,refvec,`
`  `*`PropertyName`*`,PropertyValue,...)`

Description

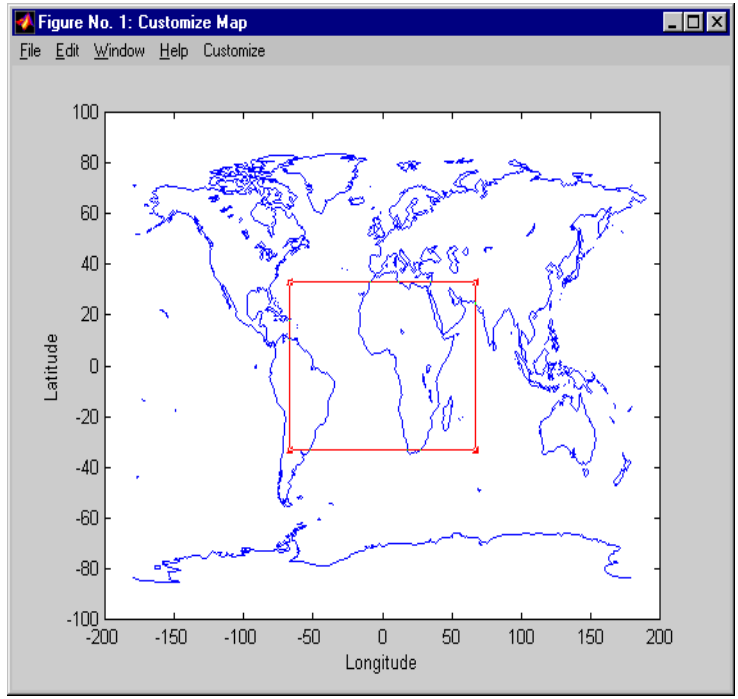
`maptrim(lat,lon)` displays the supplied map data in a new figure window and allows a region of the map to be selected and saved in the workspace. `lat` and `lon` must be vector map data. The output can be line, patch, or regular surface (matrix) data. If patch map output is selected, the inputs `lat` and `lon` must originally be patch map data.

`maptrim(lat,lon,`*`linespec`*`)` displays the supplied map data using the *`linespec`* string.

`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.

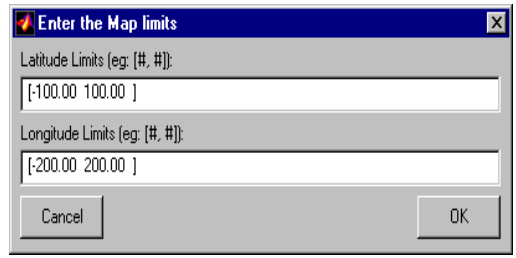
Controls



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 option toggles the `panzoom` box on and off. The box can be moved by clicking on 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.

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.

See Also

`maptriml` `maptrimp` `maptrims` `panzoom`

Typing `mapview` starts an instance of the Map Viewer, a self-contained GUI for viewing geospatial data in map (*x*-*y*) coordinates. For information on using `mapview` see “mapview” on page 10-322, and the Map Viewer tutorial “Tour Boston with the Map Viewer” on page 1-9.

Purpose

Display a regular data grid warped to a projected graticule

Activation

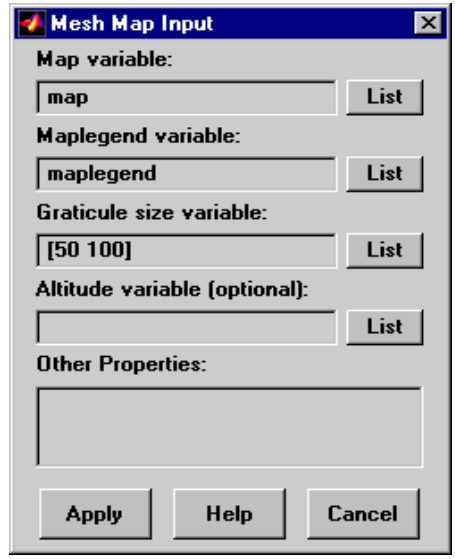
Command Line	Maptool
`meshm`	**Map⇒Regular Surfaces**

Description

`meshm` activates a **Mesh Map Input** dialog box that accepts input data to project a regular surface onto the current map axes.

If no map axes are current, a **No Map Axes** dialog box appears. Choose **Yes** to activate the **Projection Control** dialog box for defining map axes properties. Upon creation of the map axes, the **Mesh Map Input** dialog box appears.

Controls



The **Map variable** edit box is used to specify the workspace variable containing the data grid.

The **Maplegend variable** edit box is used to specify the workspace variable containing the referencing vector. Alternatively, a three-element referencing vector enclosed in brackets can be entered in place of a workspace variable.

The **Graticule size variable** edit box is used to specify the workspace variable containing the graticule resolution. A two-element vector of the form `[latitude-points longitude-points]` can be entered in place of a workspace variable. The default graticule resolution is `[50 100]`.

The **Altitude variable** edit box is used to specify the workspace variable containing the altitude data. A scalar value can be entered to specify the *z*-axis plane in which the graticule mesh is plotted.

Pressing the **List** button produces a list of all current workspace variables, from which the latitude, longitude, graticule size, and altitude variables can be selected.

The **Other Properties** edit box is used to specify additional properties of the surface object to be projected, such as `'EdgeColor',[1 1 0]`. String entries must be enclosed in quotes. The `CData` property contains the data grid and therefore cannot be set by users.

Pressing the **Apply** button accepts the input data and projects the surface object onto the current map axes.

Pressing the **Cancel** button disregards any input data and closes the **Mesh Map Input** dialog box.

See Also

`meshm`

Purpose

Interactively display and control objects in a geographic data structure workspace

Activation

Command Line	Maptool

`mlayers(`*`workspace`*`)`
`mlayers(`*`workspace`*`,h)`
`mlayers(cellarray)`
`mlayers(cellarray,h)`
	Session⇒Layers

Description

The `mlayers` tool activates a dialog box for the specified geographic data structure *`workspace`*, which enables display and manipulation of the map objects that it comprises.

`mlayers(`*`workspace`*`)` associates the geographic data structures, which in this context are also called map layers, in the *`workspace`* MAT-file with the current map axes. The geographic data structure variables are accessible only through the `mlayers` tool, and not through the base workspace. *`workspace`* must be a string.

`mlayers(`*`workspace`*`,h)` assigns the layers in *`workspace`* 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)`.

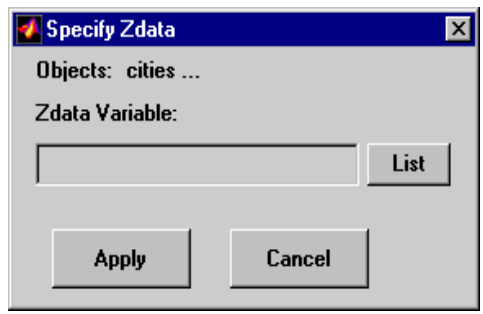
`mlayers(cellarray,h)` assigns the layers specified by `cellarray` to the map axes specified by the handle `h`.

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.

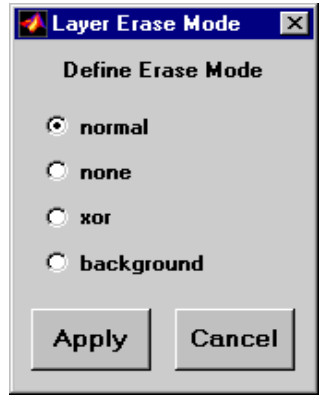


The **Highlight** button is used to toggle the selected map layer between highlighted and normal display.

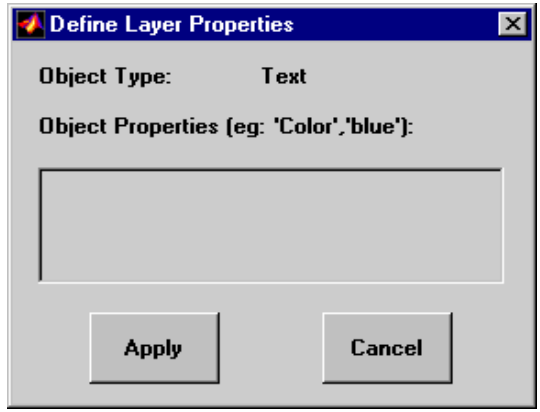
The **Members** button brings up a list of members of the selected map layer. Members of a layer are defined by their `Tag` property.

The **Delete** button deletes the selected map layer from the map.

The **Emode** button activates the **Layer Erase Mode** dialog box, which is used to specify the erase mode of the selected map layer.



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. String 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`

Purpose

Manipulate object sets plotted on a map axes

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`.

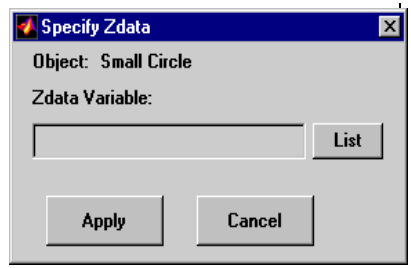
Controls

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.

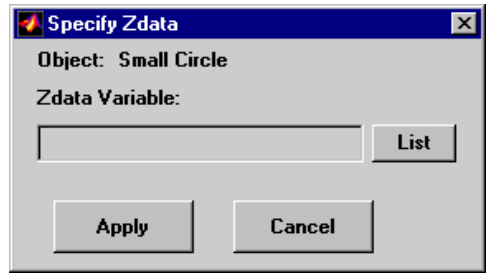


The **Highlight** button highlights all objects belonging to the selected object set.

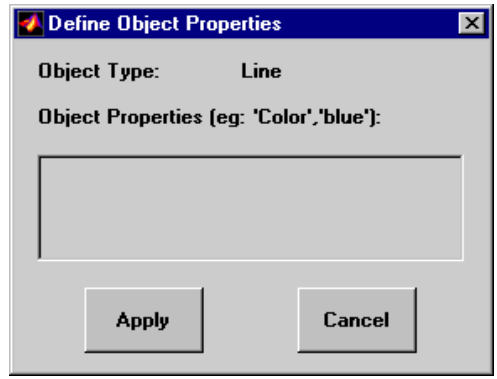
The **Tag** button brings up an **Edit Tag** dialog box, which allows the tag of all members of the selected object set to be modified.

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 **Emode** button activates the **Object Erase Mode** dialog box, which is used to specify the erase mode of the selected object set.



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. String 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`

originui

Purpose

Modify a map origin

Activation

Command Line	Maptool

`originui`
`originui on`
`originui off`
	Tools⇒Origin (menu)
	Origin (button)

Description

`originui` toggles the origin tool on and off.

`originui on` activates the origin tool.

`originui off` deactivates the origin tool.

The `originui` tool allows for interactive modification of the origin of a displayed map projection. A dot marker appears on the map at the location of the current origin. This dot can be moved using the mouse or keyboard, and the map can be reprojected with the location of the dot as the new origin.

Controls

Mouse Interaction

A single-click and drag moves the dot marker. A double-click on the dot marker reprojects the map with the dot location as the new origin. An extend-click moves the dot marker in the x or y direction only, depending on the direction of greatest mouse movement. Alternate-click exits the origin tool.

Keyboard Interaction

The following keyboard interaction is enabled if the figure containing the map axes is made the active window.

The `n`, `s`, `e`, and `w` keys move the dot marker north (up), south (down), east (right), and west (left) by one degree per keystroke. `N`, `S`, `E`, `W` keys move the dot marker ten degrees per keystroke in the same directions. Pressing the `Return` key reprojects the map with the dot location as the new origin. Pressing the **Enter** key reprojects the map with the new origin and exits the origin tool. Pressing the `Esc` or `Delete` keys exits the origin tool.

See Also

`axesm` `setm`

Purpose

Pan and zoom on a 2-D map display

Activation

Command Line	Maptool

`panzoom`
`panzoom on`
`panzoom off`
`panzoom setlimits`
`panzoom out`
`panzoom fullview`
	Tools⇒Zoom Tool (menu)
	Zoom (button)

Description

`panzoom` toggles the pan and zoom tool on and off.

`panzoom on` activates the pan and zoom tool.

`panzoom off` deactivates the pan and zoom tool.

`panzoom setlimits` sets the zoom out limits to the current settings on the map axes.

`panzoom out` zooms out to the current map axes limit settings.

`panzoom fullview` resets the axes to their full view range and resets the pan and zoom tool with these settings.

The pan and zoom tool provides an interactive means of defining zoom limits on a two-dimensional map display. A box that can be resized and moved appears on the map display and is used to define the zoom area. The box cannot be moved beyond the current axes limits.

Controls

Mouse Interaction

With the cursor inside the zoom box, a single-click and drag moves the box. The zoom box can be resized by dragging the corners of the box. A double-click in the center of the box zooms in to the current boundaries of the box. A single-click outside the zoom box moves the box to that location. An extend-click inside or outside of the zoom box zooms out by a factor of two. Alternate-click exits the pan and zoom tool.

Keyboard Interaction

The following keyboard interaction is enabled if the figure containing the map axes is made the active window.

Pressing the `Return` key sets the axes to the current zoom box and remains in pan and zoom mode. The `Enter` key sets the axes to the current zoom box and exits pan and zoom mode. Pressing the `Esc` or `Delete` keys exits pan and zoom mode.

See Also

`zoom`

Purpose

Modifying 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`

pcolorm, surfacem, surfm

Purpose

Project a geolocated data grid onto the current map axes

Activation

Command Line	Maptool

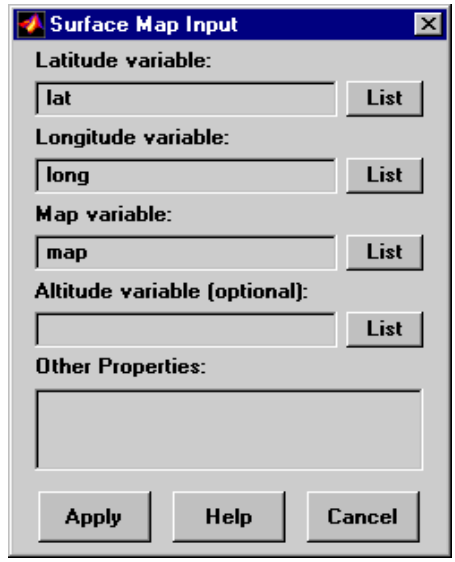
`pcolorm`
`surfacem`
`surfm`
	Map⇒General Surfaces

Description

`pcolorm`, `surfacem`, and `surfm` activate a **Surface Map Input** dialog box for projecting general surfaces onto the current map axes.

If no map axes are current, a **No Map Axes** dialog box appears. Choose **Yes** to activate the **Projection Control** dialog box for defining map axes properties. Upon creation of the map axes, the **Surface Map Input** dialog box appears.

Controls



The **Latitude variable** edit box is used to specify the workspace variable containing the latitude data of the surface to be projected.

The **Longitude variable** edit box is used to specify the workspace variable containing the longitude data of the surface to be projected.

The **Map variable** edit box is used to specify the workspace variable containing the data grid.

The **Altitude variable** edit box is used to specify the workspace variable containing the altitude data of the surface to be projected. A scalar value can be entered to indicate the plane in which to display the object.

Pressing the **List** button produces a list of all current workspace variables, from which the latitude, longitude, map, and altitude variables can be selected.

The **Other Properties** edit box is used to specify additional properties of the surface object to be projected, such as `'EdgeColor',[1 1 0]`. String entries must be enclosed in single quotes.

Pressing the **Apply** button accepts the input data and projects the surface object onto the current map axes.

Pressing the **Cancel** button disregards any input data and closes the **Surface Map Input** dialog box.

See Also

`pcolorm` `surfacem` `surfm`

property editors

Purpose

Edit properties of mapped objects using display-activated property editors

Activation

map display:	alternate-click on mapped object (for **Click-and-Drag Property Editor**)
	double-click on mapped object (for MATLAB **Guide Property Editor**)
maptool:	**Tools, Edit** menu item (for MATLAB **Guide Property Editor**)

Description

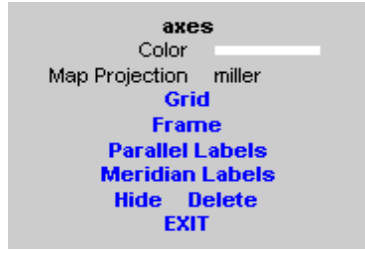
Alternate-clicking on a mapped object activates a property editor, which allows modification of some basic properties of the object through simple mouse clicks and drags. The objects supported by this editor are map axes, lines, text, patches, and surfaces, and the properties supported for each object type are shown below.

Double-clicking on a mapped object activates the MATLAB Guide Property Editor for that object. The Guide Property Editor provides a complete list of the properties and values of the selected object, allowing for modifications of the object.

Controls

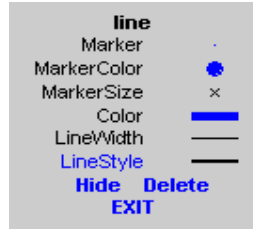
Click-and-Drag Property Editor

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 string. 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.

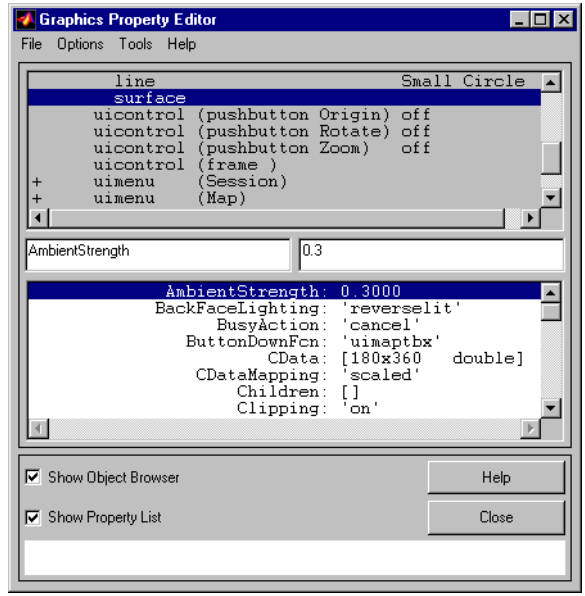
Click-and-Drag Editor for a patch object

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 on the background of the property editor closes the **Click-and-Drag** editing session.

Guide Property Editor

The MATLAB **Guide Property Editor** allows for modification of property values for all properties applicable to the selected object. The **Object Browser** is used to expand and collapse the hierarchy of objects, showing an object's parents and children. A plus sign (+) before an object indicates that the object can be expanded to show its children. A minus sign (-) before an object indicates an object can be collapsed to hide its children. 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.



MATLAB Guide Property Editor for a surface object

See Also

`propedit` `guide` `uimaptbx`

qrydata

Purpose

Interactively perform data queries

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 the string *`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 on the axes.

The input `cellarray` is used to define the data set and the query. The first cell must contain the string used to label the data display line. The second cell must contain the type of query operation, either a pre-defined operation or a valid user-defined function name. This input must be a string. The pre-defined 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* strings are 'nearest', 'linear', or 'cubic'. The default is 'nearest'.

The 'vector' query uses the MATLAB 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.

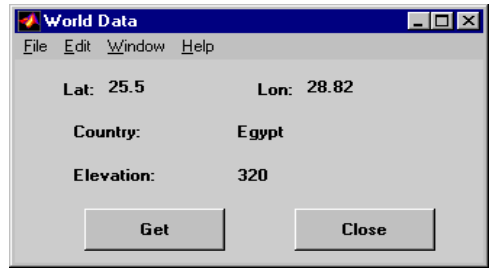
The 'mapmatrix' query interpolates to find the value of the map at the input (lat,lon) point. The format of cellarray for this query is: {'label','mapmatrix',datagrid,refvec,*method*}. datagrid and refvec are the data grid and the corresponding referencing vector. *method* is an optional argument that specifies the interpolation method. Possible *method* strings 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 pre-defined operations above. *function* is a user-created function and must refer to an M-file of the form 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 on the axes. Once a point is selected, the queries are performed and the results are displayed.

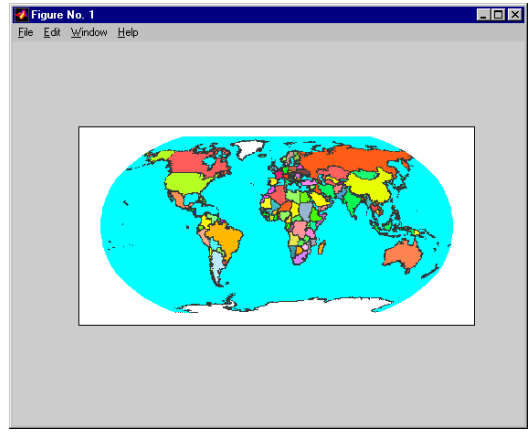
The **Process** button appears if the handle `h` is not provided. In this case, the `(x,y)` coordinates of the desired point are entered into the edit boxes. Pressing the **Process** button performs the data queries and displays the results.

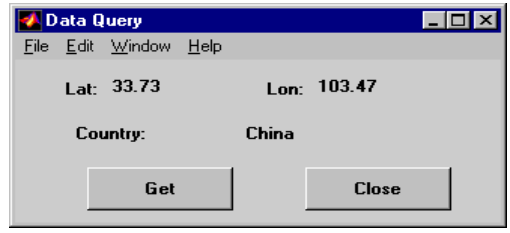
Pressing the **Close** button closes the query dialog box.

Examples

The following example can be found in the `wrlddemo` query demo provided with the Mapping Toolbox. The call to `qrydata` creates a query dialog box that enables the user to click on the map display and retrieve the name of the country corresponding to that point.

```
load worldmtx
axesm robinson
colormap(clrmap)
meshm(datagrid,refvec)
qrydata(gca,'Data Query',{'Country:','mapvector',...
        datagrid,refvec,strvcat(nations(:).name)})
```





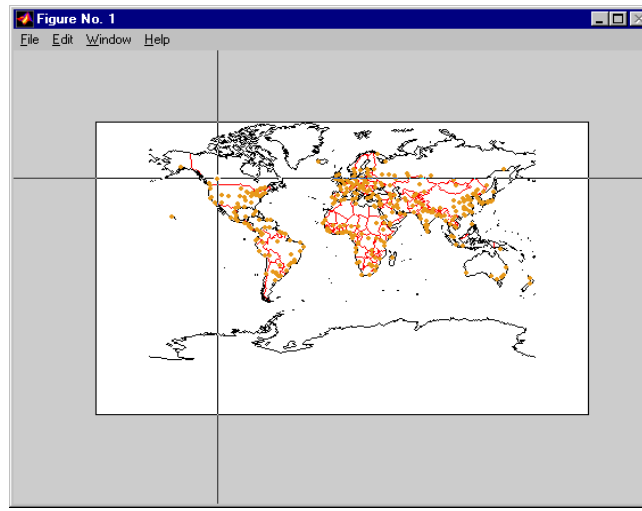
The next example makes use of a user-defined query to display city names for map points specified by a mouse click.

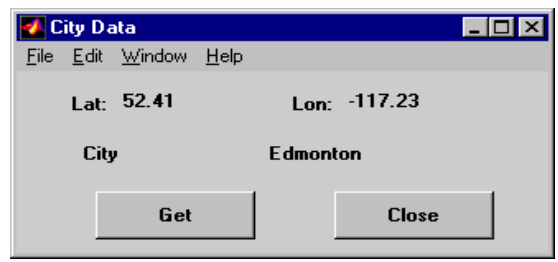
```
axesm miller
load worldlo
lat = [PPtext.lat]';
lon = [PPtext.long]';
mat = strvcat(PPtext.string);
displaym(POline)
displaym(PPpoint)
qrydata(gca,'City Data',{'City','qrytest',lat,lon,mat})
```

The following code would be contained in the M-file `qrytest`, created by the user.

```
% 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 the string `'No city located near click'`.

See Also

`interp2` `qrydemo` `wrlddemo`

quiver3m

Purpose

Project a 3-D quiver plot onto the current map axes

Activation

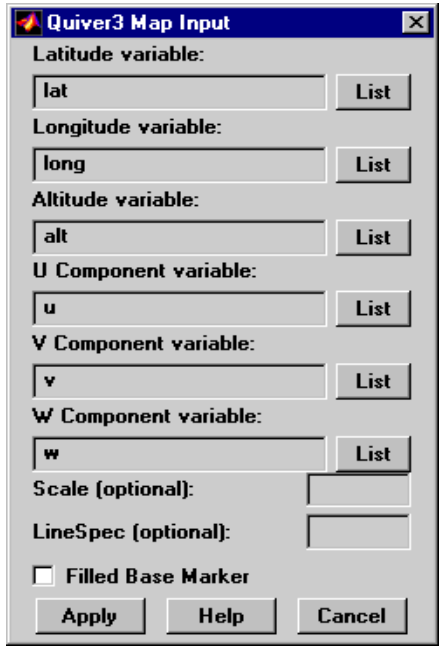
Command Line	Maptool
quiver3m	**Map⇒Quiver 3D**

Description

`quiver3m` activates a **Quiver3 Map Input** dialog box to project a three-dimensional quiver plot onto the current map axes. The vectors `(u,v,w)` are displayed at the points `(latitude,longitude,altitude)` on the map.

If no map axes are current, a **No Map Axes** dialog box appears. Choose **Yes** to activate the **Projection Control** dialog box for defining map axes properties. Upon creation of the map axes, the **Quiver3 Map Input** dialog box appears.

Controls



The **Latitude variable** edit box is used to specify the workspace variable containing the latitude data for the quiver plot.

The **Longitude variable** edit box is used to specify the workspace variable containing the longitude data for the quiver plot.

The **Altitude variable** edit box is used to specify the workspace variable containing the altitude data for the quiver plot.

The **U Component variable** edit box is used to specify the workspace variable containing the u vector component data.

The **V Component variable** edit box is used to specify the workspace variable containing the v vector component data.

The **W Component variable** edit box is used to specify the workspace variable containing the w vector component data.

Pressing the **List** button produces a list of all current workspace variables, from which the latitude, longitude, altitude, u, v, and w variables can be selected.

The **Scale** edit box is used to enter the workspace variable containing the scale factor applied to the projected vectors. The vector lengths are automatically determined to make them as long as possible without overlapping. The vector lengths are then multiplied by `scale`. A `scale` of 0.5 results in vectors half as long as they would be with the default `scale` of 1. A `scale` of 0 suppresses automatic scaling, and the vector lengths are determined from the inputs. In this case, the vectors are plotted from `(latitude,longitude,altitude)` to `(latitude+u,longitude+v,altitude+w)`. A scalar value for `scale` can be entered instead of a variable name.

The **Linespec** edit box is used to enter a line specification, such as `'-r*'`, for the quiver plot. If a symbol is given in the linespec string, it is plotted at the beginning of the vectors. If no symbol is given in the linespec string, arrows are plotted at the end of the vectors.

The **Filled Base Marker** check box is used to specify a filled-in symbol at the beginning of each vector.

Pressing the **Apply** button accepts the input data and projects the quiver plot onto the current map axes.

Pressing the **Cancel** button disregards any input data and closes the **Quiver3 Map Input** dialog box.

See Also

`quiver3m`

Purpose

Project a 2-D quiver plot onto the current map axes

Activation

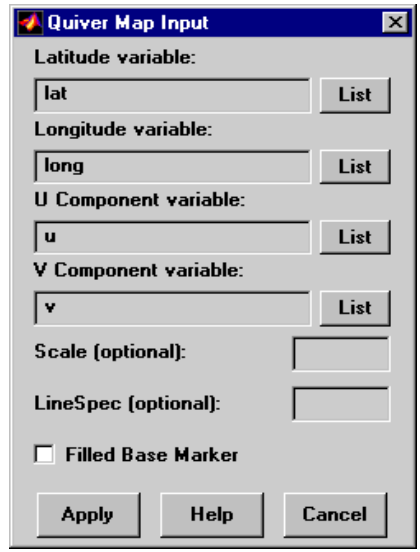
Command Line	Maptool
quiverm	**Map⇒Quiver 2D**

Description

`quiverm` activates a **Quiver Map Input** dialog box to project a two-dimensional quiver plot onto the current map axes. Vectors with components `(u,v)` are displayed at the points `(latitude,longitude)` on the map.

If no map axes are current, a **No Map Axes** dialog box appears. Choose **Yes** to activate the **Projection Control** dialog box for defining map axes properties. Upon creation of the map axes, the **Quiver Map Input** dialog box appears.

Controls



The **Latitude variable** edit box is used to specify the workspace variable containing the latitude data for the quiver plot.

The **Longitude variable** edit box is used to specify the workspace variable containing the longitude data for the quiver plot.

The **U Component variable** edit box is used to specify the workspace variable containing the u vector component data.

The **V Component variable** edit box is used to specify the workspace variable containing the v vector component data.

Pressing the **List** button produces a list of all current workspace variables, from which the latitude, longitude, u, and v variables can be selected.

The **Scale** edit box is used to enter the workspace variable containing the scale factor applied to the projected vectors. The vector lengths are automatically determined to make them as long as possible without overlapping. The vector lengths are then multiplied by `scale`. For example, a `scale` value of 0.5 results in vectors half as long as they would be with the default `scale` of 1. A `scale` of 0 suppresses automatic scaling, and the vector lengths are determined from the inputs. In this case, the vectors are plotted from `(latitude,longitude)` to `(latitude+u,longitude+v)`. A scalar value for `scale` can be entered instead of a variable name.

The **Linespec** edit box is used to enter a line specification, such as `'-r*'`, for the quiver plot. If a symbol is given in the linespec string, it is plotted at the beginning of the vectors. If no symbol is given in the linespec string, arrows are plotted at the end of the vectors.

The **Filled Base Marker** check box is used to specify a filled-in symbol at the beginning of each vector.

Pressing the **Apply** button accepts the input data and projects the quiver plot onto the current map axes.

Pressing the **Cancel** button disregards any input data and closes the **Quiver Map Input** dialog box.

See Also

`quiverm`

Purpose

Project a symbol map on the current map axes

Activation

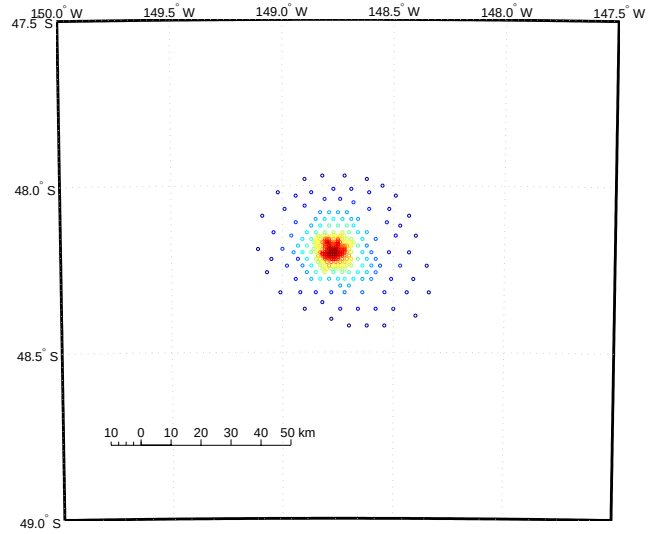
Command Line	Maptool
scatterm	**Map⇒Scatter**

Description

`scatterm` activates a **Scatter Map Input** dialog box to project a symbol plot onto the current map axes. A symbol map displays symbols proportionally sized to the data.

If no map axes are current, a **No Map Axes** dialog box appears. Choose **Yes** to activate the **Projection Control** dialog box for defining map axes properties. Upon creation of the map axes, the **Scatter Map Input** dialog box appears.

Controls



The **Latitude variable** edit box is used to specify the workspace variable containing the latitude coordinates for the scatter plot.

The **Longitude variable** edit box is used to specify the workspace variable containing the longitude coordinates for the scatter plot.

The **Marker size variable** edit box is used to specify the workspace variable containing the marker weights. The markers areas proportionally sized based on these weights. The marker size can also be a scalar, which is applied to all markers.

The **Marker Color Variable** edit box is used to specify the workspace variable containing the marker color data. The marker color data is linearly mapped to the colors in the colormap. The marker color data can also be a vector of RGB values or a color string.

Pressing the **List** button produces a list of all current workspace variables, from which the latitude, longitude marker size, and color variables can be selected.

The **Marker Style** popup menu is used to select the marker type..

The **Filled** check box is used to select unfilled (the default) or filled markers.

Pressing the **Apply** button accepts the input data and projects the scatter plot onto the current map axes.

Pressing the **Cancel** button disregards any input data and closes the **Scatter Map Input** dialog box.

See Also

`scatterm`

Purpose

Display small circles on a map axes

Activation

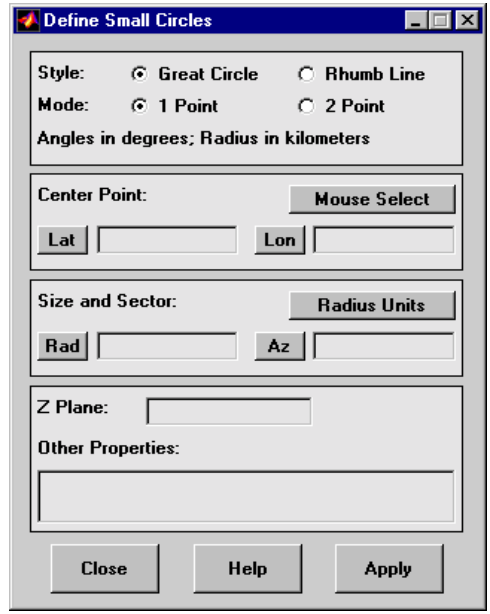
Command Line	Maptool
`scirclui` `scirclui(h)`	**Display⇒Small Circles**

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 on 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 on 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 Radius 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.

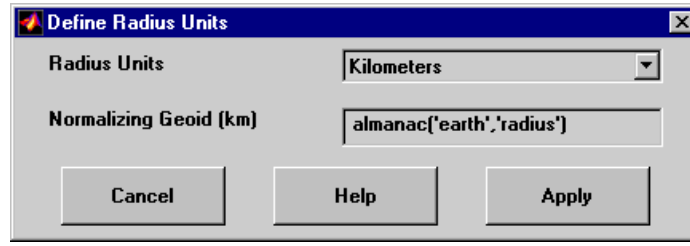
The **Other Properties** edit box is used to specify additional properties of the small circles to be projected, such as `'Color','b'`. String entries must be enclosed in quotes.

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.

Define Radius Units 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` `point`

Purpose

Encode a regular surface map

Activation

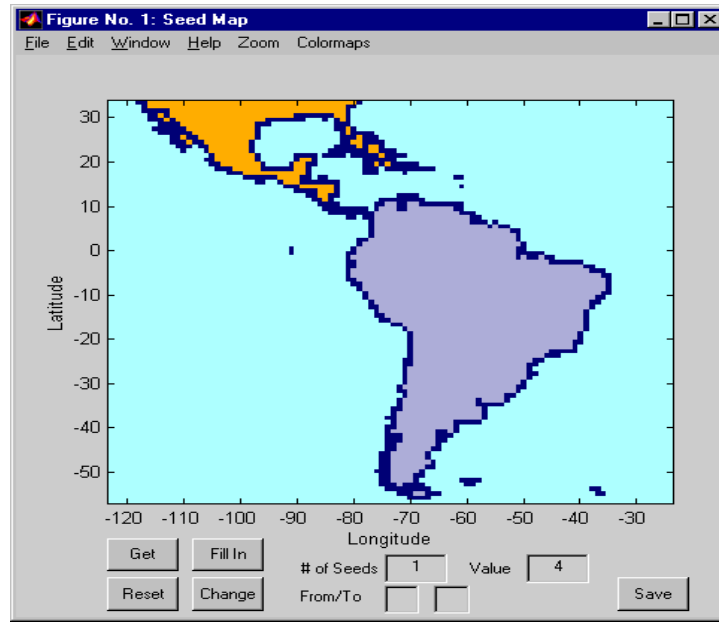
Command Line
seedm(datagrid,refvec)

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 on 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`

Purpose

Show mapped objects

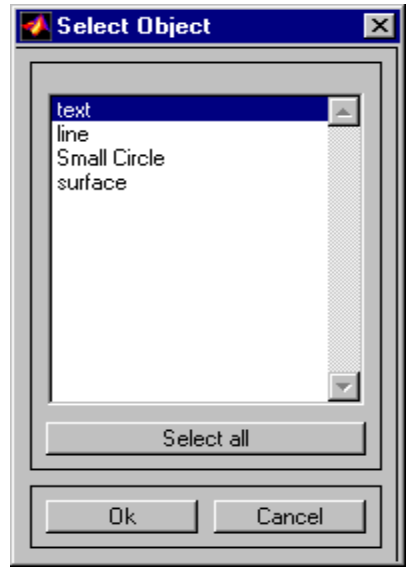
Activation

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`

stem3m

Purpose

Project a stem plot onto the current map axes

Activation

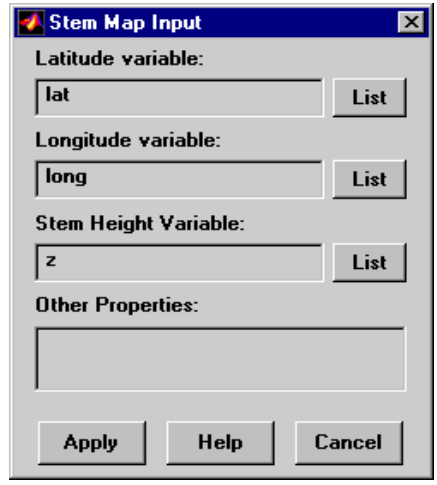
Command Line	Maptool
stem3m	**Map⇒Stem**

Description

`stem3m` activates a **Stem Map Input** dialog box for projecting a stem plot onto the current map axes. A stem plot displays data as lines extending perpendicular to the *xy*-plane on the map.

If no map axes are current, a **No Map Axes** dialog box appears. Choose **Yes** to activate the **Projection Control** dialog box for defining map axes properties. Upon creation of the map axes, the **Stem Map Input** dialog box appears.

Controls



The **Latitude variable** edit box is used to specify the workspace variable containing the latitude coordinates for the stem plot.

The **Longitude variable** edit box is used to specify the workspace variable containing the longitude coordinates for the stem plot.

The **Stem Height variable** edit box is used to specify the workspace variable containing the stem height data.

Pressing the **List** button produces a list of all current workspace variables, from which the latitude, longitude, and stem height variables can be selected.

The **Other Properties** edit box is used to specify additional properties of the stem lines to be projected, such as `'Color','r'`. String entries must be enclosed in quotes.

Pressing the **Apply** button accepts the input data and projects the stem plot onto the current map axes.

Pressing the **Cancel** button disregards any input data and closes the **Stem Map Input** dialog box.

See Also

`stem3m`

surfdist

Purpose

Interactively calculate distance, azimuth, and reckoning

Activation

Command Line	Maptool

`surfdist`
`surfdist(h)`
`surfdist([])`
	Display⇒Surface⇒Distances

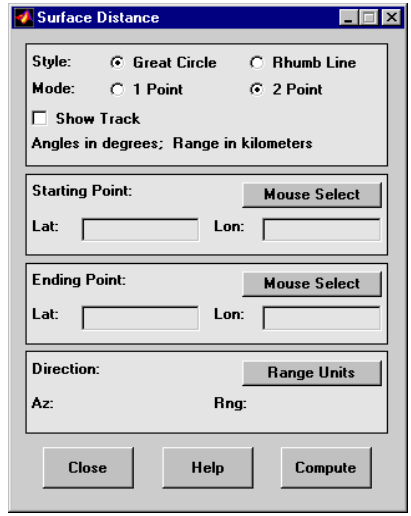
Description

`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** 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 on 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 on 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

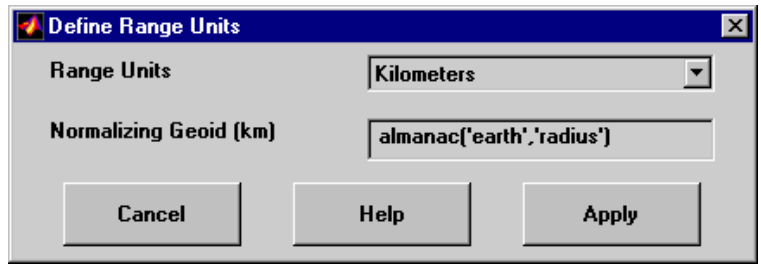
used to compute the ending point of the track in one-point mode. During two-point mode, the **Direction** controls are disabled, but the azimuth and range values resulting from the surface distance calculation are displayed.

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.

surflm

Purpose

Display a lighted data grid warped to a projected graticule

Activation

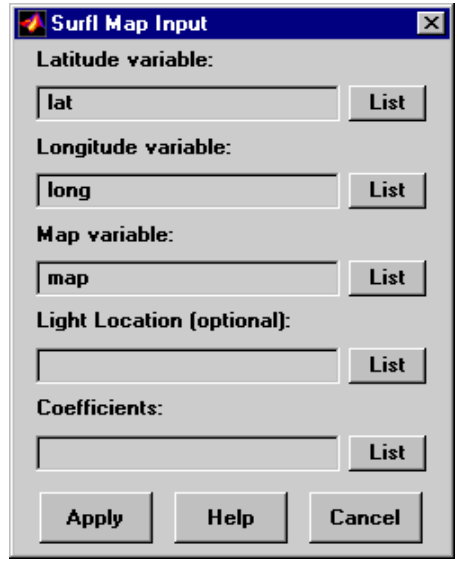
Command Line
surflm

Description

`surflm` activates a **Surflm Map Input** dialog box to project a lighted map surface onto the current map axes.

If no map axes are current, a **No Map Axes** dialog box appears. Choose **Yes** to activate the **Projection Control** dialog box for defining map axes properties. Upon creation of the map axes, the **Surflm Map Input** dialog box appears.

Controls



The **Latitude variable** edit box is used to specify the workspace variable containing the latitude data of the surface to be projected.

The **Longitude variable** edit box is used to specify the workspace variable containing the longitude data of the surface to be projected.

The **Map variable** edit box is used to specify the workspace variable containing the data grid.

The **Light Location** edit box is used to specify the workspace variable containing the direction of the light source. This can be a three-element vector of the form `[x y z]` or a two-element vector of the form `[azimuth elevation]`. If omitted, the default is 45 degrees counterclockwise from the current view direction.

The **Coefficients** edit box is used to specify the workspace variable containing the relative contributions of ambient light, diffuse reflection, specular reflection, and the specular shine coefficient. This is a four-element vector of the form `[ka kd ks shine]`. If the entry is omitted, the default is `[.55 .6 .4 10]`.

Pressing the **List** button produces a list of all current workspace variables, from which the latitude, longitude, map, light location, and coefficient variables can be selected.

Pressing the **Apply** button accepts the input data and projects the lighted surface object onto the current map axes.

Pressing the **Cancel** button disregards any input data and closes the **Surflm Map Input** dialog box.

See Also

`surflm`

tagm

Purpose

Edit the tag of mapped objects

Activation

Command Line
`tagm` `tagm(h)`

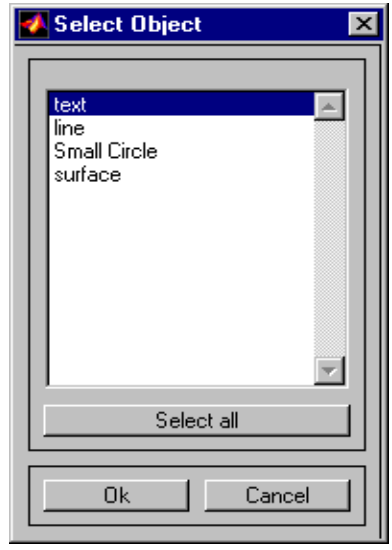
Description

`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

The new tag string is entered in the edit box. Pressing the **Apply** button changes the `Tag` property of all selected objected to the new tag string. Pressing the **Cancel** button closes the **Edit Tag** dialog box without changing the `Tag` property of the selected objects.

See Also

`tagm`

textm

Purpose

Project text on the current map axes

Activation

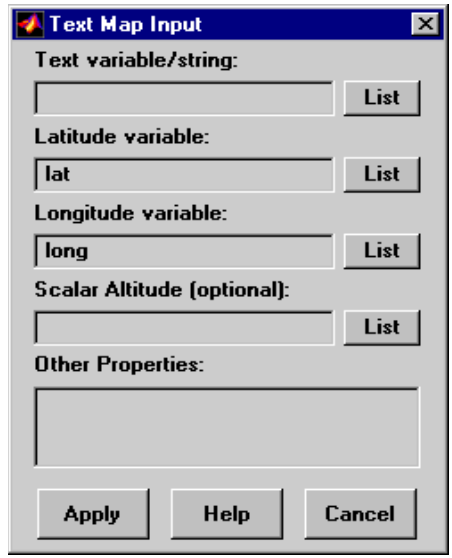
Command Line	Maptool
textm	**Map⇒Text**

Description

`textm` activates a **Text Map Input** dialog box, which accepts input data to project a text object onto the current map axes.

If no map axes are current, a **No Map Axes** dialog box appears. Choose **Yes** to activate the **Projection Control** dialog box for defining map axes properties. Upon creation of the map axes, the **Text Map Input** dialog box appears.

Control



The **Text variable/string** edit box is used to specify the workspace variable containing the text strings to be projected. A single text string can also be entered, provided it is enclosed in single quotes. Multiple lines of text can be entered using a cell array.

The **Latitude variable** edit box is used to specify the workspace variable containing the latitude data for the text string(s) to be projected. If a single text string is to be plotted, a scalar latitude value can be entered.

The **Longitude variable** edit box is used to specify the workspace variable containing the longitude data of the text object(s) to be projected. If a single text string is to be plotted, a scalar longitude value can be entered.

The **Scalar Altitude** edit box is used to specify the workspace variable containing the *z*-axis altitudes of the text object(s) to be projected. If a single text string is to be plotted, a scalar altitude value can be entered.

Pressing the **List** button produces a list of all current workspace variables, from which the latitude, longitude, and altitude variables can be selected.

The **Other Properties** edit box is used to specify additional properties of the text object(s) to be projected, such as `'FontSize',12`. String entries must be enclosed in quotes.

Pressing the **Apply** button accepts the input data and projects the text object(s) onto the current map axes.

Pressing the **Cancel** button disregards any input data and closes the **Text Map Input** dialog box.

See Also

`textm`

trackui

Purpose

Display great circles and rhumb lines on a map

Activation

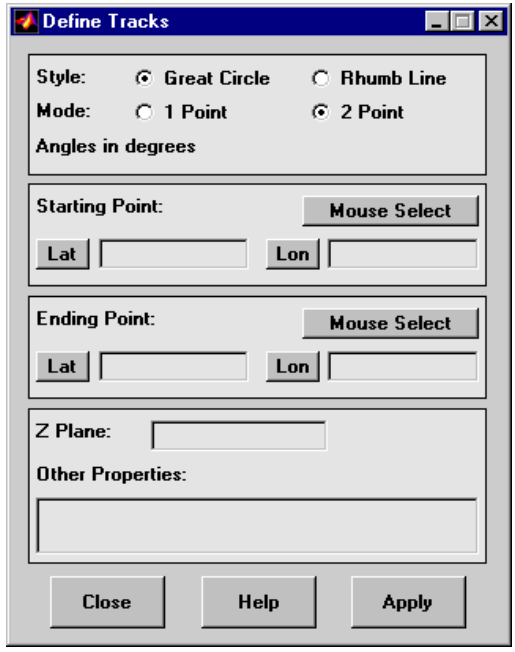
Command Line	Maptool
`trackui` `trackui(h)`	**Display⇒Tracks**

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 on 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 on 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.

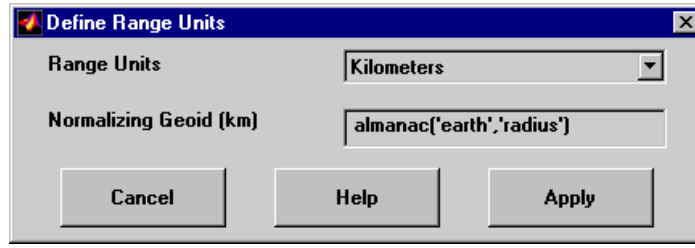
The **Other Properties** edit box is used to specify additional properties of the tracks to be projected, such as `'Color','b'`. String entries must be enclosed in quotes.

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`

uimaptbx

Purpose

Process mouse button down 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 the Mapping Toolbox.

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 Guide 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 – Control click left button or right button
`For X-Windows:`	Extend-click – Shift click left button or middle button Alternate-click – Control click left button or right button

See Also

`axesm  axesmui property editors`

Purpose

Adjust the *z*-plane of mapped objects

Activation

Command Line

`zdatam`
`zdatam(h)`
`zdatam(str)`

Description

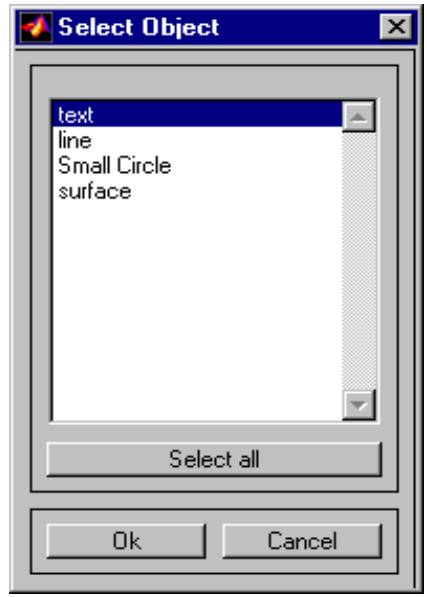
`zdatam` brings up a **Select Object** dialog box for selecting mapped objects and adjusting their `ZData` property. Upon selecting the objects, the **Specify Zdata** dialog box is activated, in which the new `ZData` variable is entered. Note that not all mapped objects have the `ZData` property (for 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 string recognized by `handlem`.

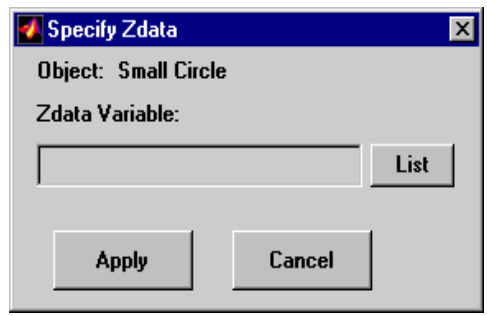
Controls

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`

A

Atlas Data

Types of Data

The Mapping Toolbox includes a number of data sets for global and regional displays. Map data is available in both vector and matrix format, covering the world and the United States. The Mapping Toolbox also provides astronomical data for simple stellar cartography.

More detailed data is available over the Internet or on CD-ROM and can be imported to MATLAB using the data interface functions listed in “Geospatial Data Import and Access” on page 10-6.

World Vector Data

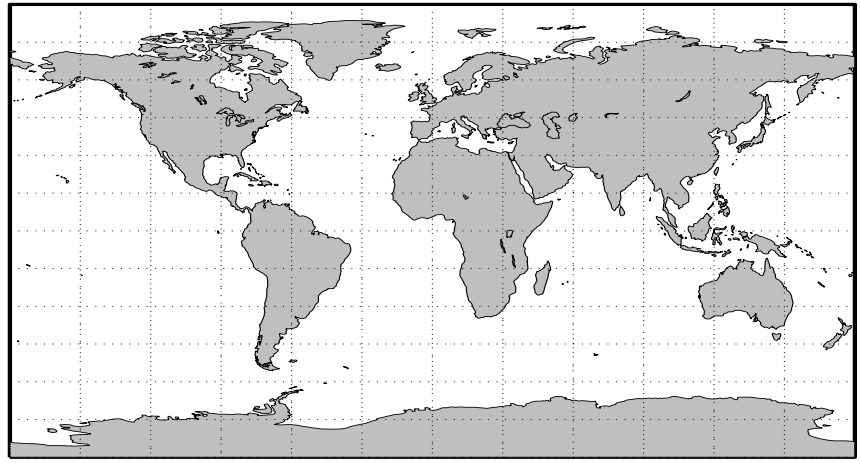
Coastlines

The `coast` MAT-file contains a set of vector shorelines intended for global displays in which political boundaries are not required. The data is stored in vectors of latitude and longitude, consisting of nearly 10,000 points. While this a considerable quantity of data, it is of very low resolution by cartographic standards.

```
load coast
whos
Name       Size                    Bytes  Class

  lat     9589x1                    76712  double array
  long    9589x1                    76712  double array

Grand total is 19178 elements using 153424 bytes
```

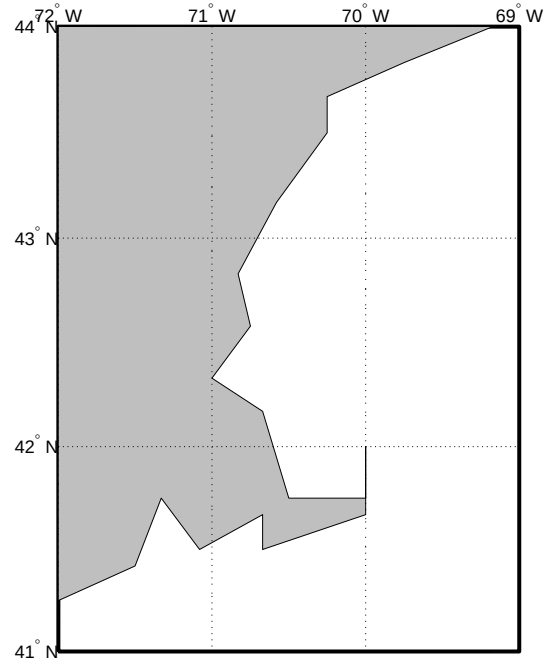
The data set is in patch format, meaning it has polygon segments that return to their starting points and can therefore be usefully displayed using patch functions. The polygon segments are separated by `NaN`s into about 240 faces. The data can also be displayed as lines or points.

```
axesm giso; framem; gridm
patchesm(lat,long,'FaceColor',0.75*[1 1 1])
```



The resolution for this data set makes it ideal for global-scale displays but inappropriate for significantly smaller regions, as can be seen when the region around Cape Cod in the northeastern United States is displayed. This is, incidentally, the area covered by the `cape` MAT-file provided with MATLAB.

```
figure
axesm('MapProjection','mercator',...
   'MapLatLimit',[41 44],'MapLonLimit',[-72 -69])
framem
gridm('MLineLocation',1,'PLineLocation',1);
mlabel('MLabelLocation',1)
plabel('PLabelLocation',1)
patchesm(lat,long,'FaceColor',.75*[1 1 1])
```



Low-Resolution World Atlas Data

The `worldlo` global atlas data contains a set of very small-scale (approximately 1:30,000,000) data for use in global displays. The data is provided in the form of Mapping Toolbox geographic data structures containing national boundaries, rivers, lakes, and cities. Here is a list of the structures in the `worldlo` MAT-file:

```
load worldlo
whos
Name              Size                    Bytes  Class

  DNline            1x2                   112226  struct array
  DNpatch           1x39                   50092  struct array
  POline            1x2                   456216  struct array
  POpatch           1x208                 645030  struct array
  POtext            1x197                 131670  struct array
```

```
  PPpoint          1x319                278154  struct array
  PPtext           1x318                205628  struct array
  description      4x75                    600  char array
  gazette          1x513                244476  struct array
  source           2x63                    252  char array

Grand total is 215687 elements using 2124344 bytes
```

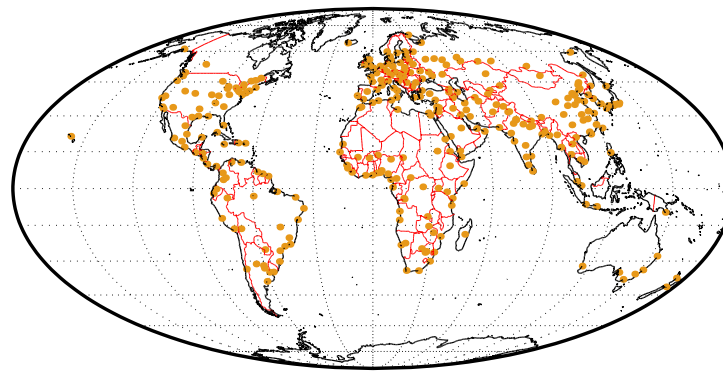
The data has been classified into Drainage (DN), Political/Ocean boundaries (PO), and Populated Places (PP). The data is further separated into structures containing patches, lines, points, or text.

In addition to containing more kinds of data than `coast`, this data set is also more detailed. The drainage data consists of about 7,000 points, while the political lines and patches each have about 30,000 points. There are more than 200 political units in the political data and more than 300 major cities in populated places. Because of the quantity of data, complicated displays may take longer to project and render. Displays of patches require the greatest memory and computing time.

The contents of the structures can be displayed from the command line using `displaym`. For example, type the following:

```
axesm mollweid; framem; gridm
displaym(POline)
displaym(PPpoint)
```

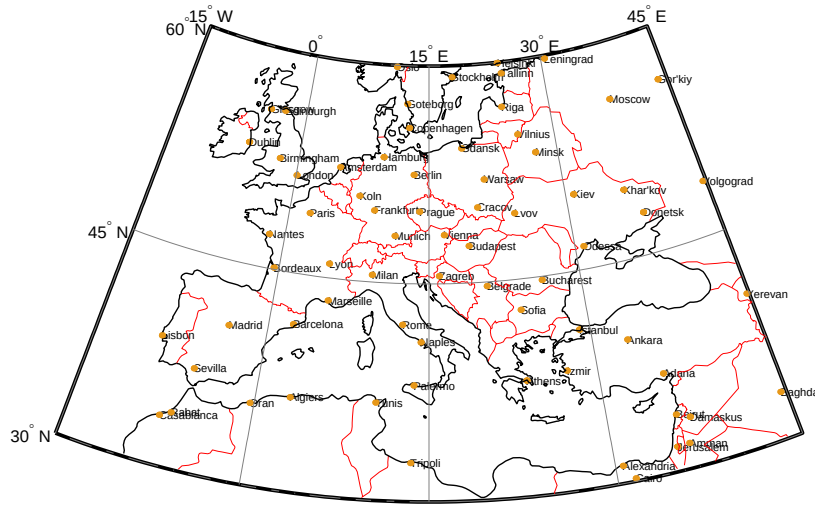
Here is the result:



The map displays the coastlines, international borders, and major cities of the world. We could also label the cities with data from the `PPtext` structure, but at this scale, the result would be unreadable. When you restrict yourself to a smaller geographic region, more information can be shown:

```
figure
axesm('Mapprojection','eqaconic','MapParallels',[],...
    'MapLatLimit',[30 60],'MapLonLimit',[-15 45],...
    'MLabelLocation',15,'MLineLocation',15,...
    'PLabelLocation',15,'PLineLocation',15,...
    'GColor',.5*[1 1 1],'GLineStyle','-')
framem; gridm; mlabel; plabel
displaym(POline)
displaym(PPpoint)
h = displaym(PPtext); trimcart(h)
```

The zoomed-in region of Europe allows the names of the major cities to be read more easily:

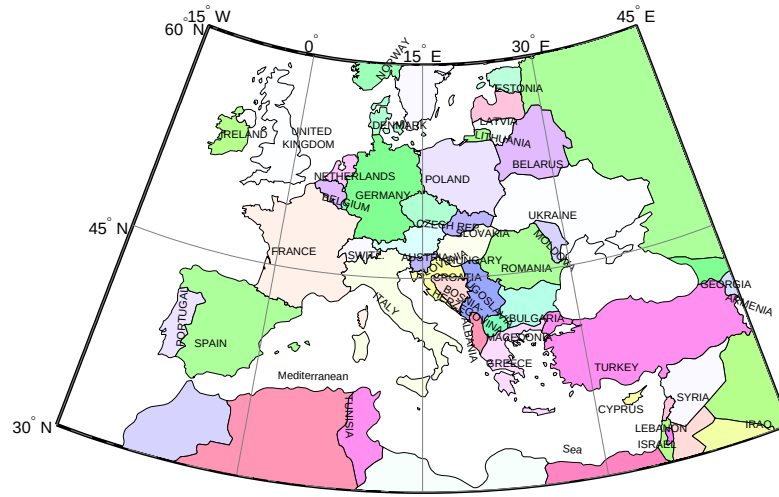


Note Unlike vector and matrix data, text objects are not automatically trimmed when they fall outside the map frame limits. You can remove them using the `trimcart` function or some other method (e.g., click-on-text and `deletem(gco)`, or use click-and-drag tools).

There are other ways to represent political data. We can display the countries as patches, label them with strings from the `POtext` structure, and pick random colors for the patch faces. Clear the previous line map, and redraw the patch map:

```
clma
displaym(POpatch); polcmap
h = displaym(POtext); trimcart(h); zdatam(h,1)
```

Here is the result:

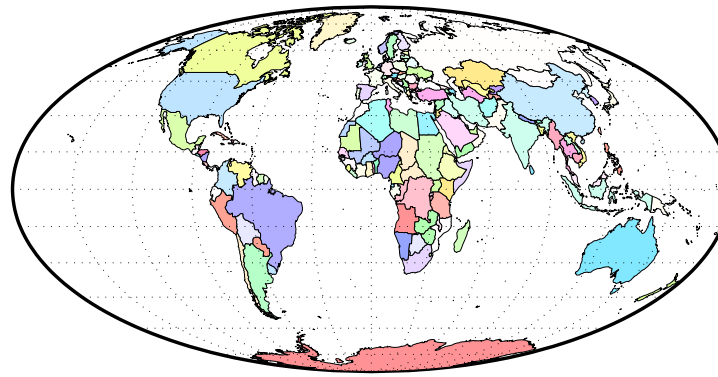


Note how the separated parts of countries are filled with the same colors. Sardinia and Sicily, for example, are shaded the same color as the boot of Italy. This is accomplished automatically because each of the components is tagged as “Italy.”

Every element in a `worldlo` structure has a tag field. Drainage data has been tagged to distinguish between rivers and lakes. The political/ocean lines are tagged as coastlines or international boundaries, and the political patches are tagged with the names of the countries. Both of the functions `displaym` and `mlayers` use the same colors for similarly tagged objects if no `otherproperty` is specified in the geographic data structure.

In the following example, it is evident that the many islands of Indonesia and Canada are similarly colored, as are all parts of the United States:

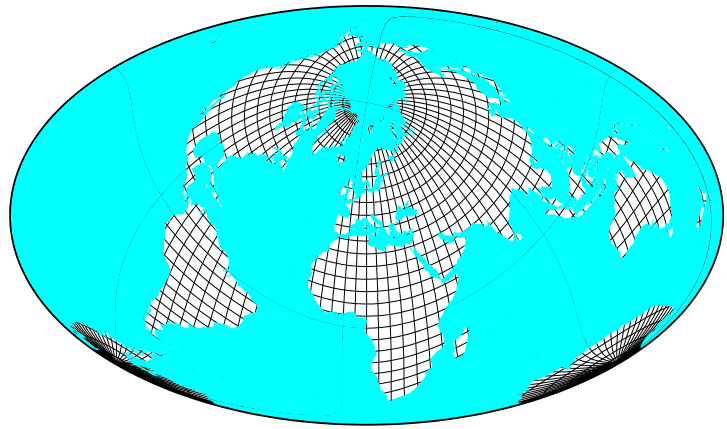
```
figure
axesm mollweid; framem; gridm
displaym(POpatch)
```



There is also a corresponding set of patches for the oceans. These can be used to *blank out* objects in ocean areas. The data is stored separately in the `oceanlo.mat` file, but it can be accessed using the `worldlo` atlas function. The patches are in large tiles with few points along the edges, which may result in visible seams for noncylindrical projections. You can use `interpm` to fill in points at a sufficiently fine spacing.

```
[lat,lon] = extractm(worldlo('oceanmask'));
[lat,lon] = interpm(lat,lon,5);

axesm('bries','GAlt',-1,'MLineLocation',5,'PLineLocation',5,...
      'GLineStyle','-')
framem; gridm
```

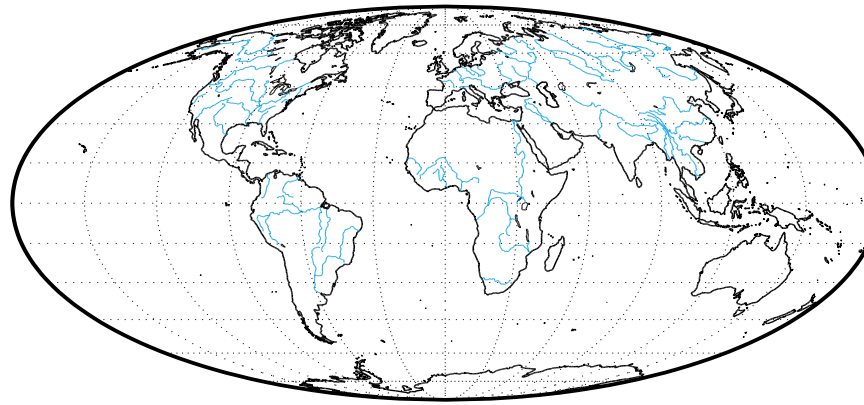
```
patchesm(lat,lon,'c','EdgeColor','none')
```



You can use the tags to reduce the amount of data in the display by selectively deleting data. Youdo this by displaying all the data in a structure and then using the tags to get the handles of the displayed objects you want to remove. The first European example made use of this technique, as does the next one. It shows rivers, lakes, and coastlines. International boundaries have been removed using their tags:

```
clma
axesm mollweid; framem; gridm
displaym(POline)
delete(handlem('International Boundary'))
displaym(DNline)
```

Here is the map of world coastlines and drainage:



There are a number of other ways to work with the tags associated with this data. You can view the tags in the workspace by working directly with the geographic data structure.

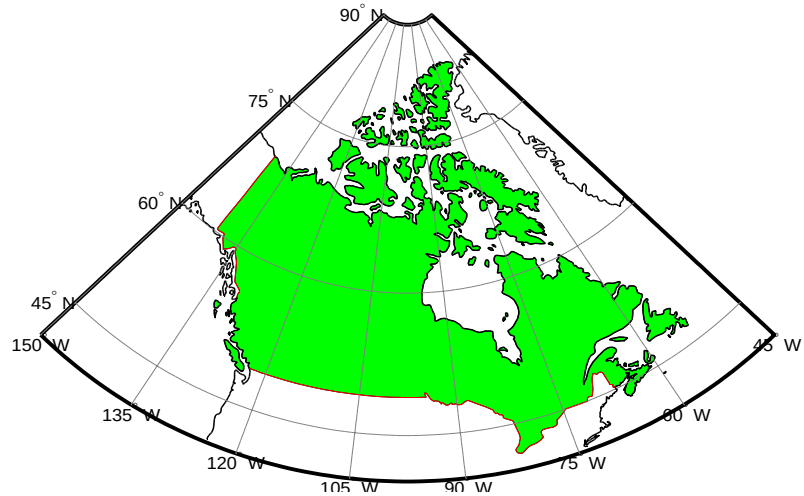
```
unique(strvcat(DNline.tag),'rows')
ans =
Inland shorelines
Streams,rivers,channelized rivers
```

Or you can use `mlayers` and `mobjects` to plot the structures, view the tags, and manipulate similarly tagged objects as a group. See theGUI reference pages for `mlayers` and `mobjects` for some examples.

You can also use the tags to extract data from the structures. The following commands extract the patch data for Canada and plot it with the political line data:

```
figure
axesm('MapProjection','eqaconic','MapParallels',[],...
    'MapLatLimit',[40 90],'MapLonLimit',[-150 -45],...
    'MLabelLocation',15,'MLineLocation',15,...
    'PLabelLocation',15,'PLineLocation',15,...
    'GColor',.5*[1 1 1],'GLinestyle','-',...
    'MLabelParallel','south')
framem; gridm; mlabel; plabel
displaym(POline)
```

```
[clat,clon] = extractm(POpatch,'canada');
patchesm(clat,clon,'g')
```



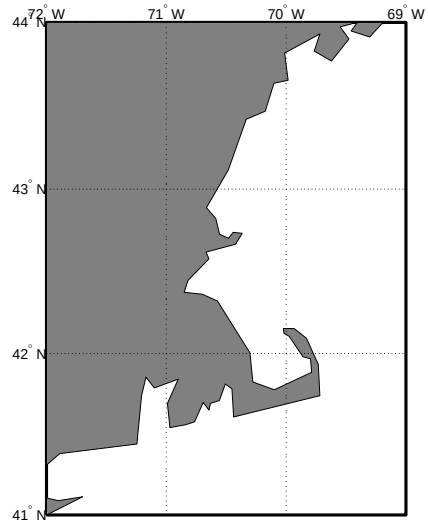
Note Extracting and displaying only the desired data can reduce the time and memory required to display a map.

The Mapping Toolbox automatically trims and saves data outside the current map latitude and longitude limits. If you use `displaym` to show a small part of the world, the rest of the world data is retained within the map axes structure. Every time the projection parameters are changed, all of the world data is restored, trimmed, and projected. This overhead can be significant for a data set as large as `worldlo`. It is best to finalize the projection parameters before plotting a lot of data.

Apply the extraction technique to generate a display of Cape Cod. As was shown in the earlier examples, this data contains somewhat generalized shapes for the countries and rivers. The level of detail is great enough for global and regional maps, but the distinct data points become noticeable when displayed for smaller regions. This data should be considered accurate to no better than 10 or 20 kilometers. For more accurate data, see the `usahi` atlas

data or the high-resolution data accessible through the external interfaces listed in “Geospatial Data Import and Access” on page 10-6.

```
figure
axesm('MapProjection','mercator','Frame','on',...
   'MapLatLimit',[41 44],'MapLonLimit',[-72 -69])
gridm('MLineLocation',1,'PLineLocation',1)
mlabel('MLabelLocation',1); plabel('PLabelLocation',1)
h = displaym(POpatch,'united states');
set(h,'FaceColor',.5*[1 1 1])
```

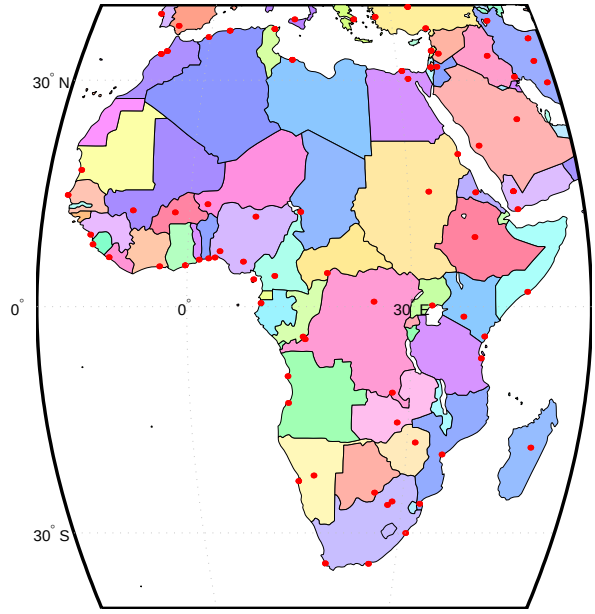


There are some more convenient ways of working with the `worldlo` atlas data. The `worldmap` function makes it easy to create map displays. Just specify the region or country you want to map. This function handles the cartographic details like selecting a projection, setting the map limits, and setting the origin and the grid and label spacings. You can also use the `worldlo` interface function to load one of the atlas data structures as an argument to a function, or load just that structure into the workspace.

Make a map of Africa with filled patches, and add the locations of major cities to the display.

```
worldmap('africa','patch'); polcmap(256)
```

```
h = displaym(worldlo('PPpoint')); setm(h,'Color','r')
```



You can also use the `worldlo` atlas data to look up names and locations. All the names mentioned in the `POtext` and `PPtext` structures are collected in a gazette structure, which you can query using `extractm`. For example,you can find the location of the city of Antananarivo on the island of Madagascar by using the following commands:

```
[lat,long,indx] = extractm(gazette,'antan');
gazette(indx)
ans =
             type: 'line'
    otherproperty: []
              tag: 'Antananarivo'
           string: 'Populated Place Name'
         altitude: []
              lat: -18.9141
             long: 47.5258
```

A much more extensive gazette feature can be found in the Digital Chart of the World (DCW), containing more than 10,000 names, compared to about 500 in the `worldlo` MAT-file. See “Vector Map Products” on page 10-7, for more information on the DCW and related exchange formats.

High-Resolution World Atlas Data

The `worldhi` global atlas data contains a set of very large-scale (approximately 1:1,000,000) data for use in regional displays. The data is provided in the form of Mapping Toolbox geographic data structures containing national boundaries, ocean names, and some cities.

You can request the outline of a particular country by providing the name. For a complete list of names, type `worldhi` with no arguments.

```
worldhi

WORLDLO atlas data:


  Geographic Data Structures:

    POpatch       - Countries as patches
    POtext        - Names of water bodies as text
    PPpoint       - Major cities as points
    PPtext        - Major cities as text labels

  Countries in the WORLDHI database:

    Afghanistan
    Agalega Island (MAU)
    Albania
    Alderney (UK)
    Algeria
    American Samoa (USA)
    Andorra
    ...
```

When you provide a country name, `worldhi` returns a geographic data structure containing the NaN-clipped vectors with the outline of the mainland and islands. The result also contains some additional fields over those required

by the definition of the geographic data structure. The `latlim` and `lonlim` fields contain the limits of a bounding box for the country in its entirety. The area field contains the area (in units of square kilometers) and the latitude and longitude limits the bounding box for each of the mainland and islands.

```
worldhi('Haiti')

ans =

             type: 'patch'
    otherproperty: {}
         altitude: []
              lat: [2011x1 double]
             long: [2011x1 double]
              tag: 'Haiti'
           latlim: [18.021 20.088]
          longlim: [-74.479 -71.613]
             area: [5x1 double]
          latlims: [5x2 double]
         longlims: [5x2 double]
            sname: 'Haiti'
```

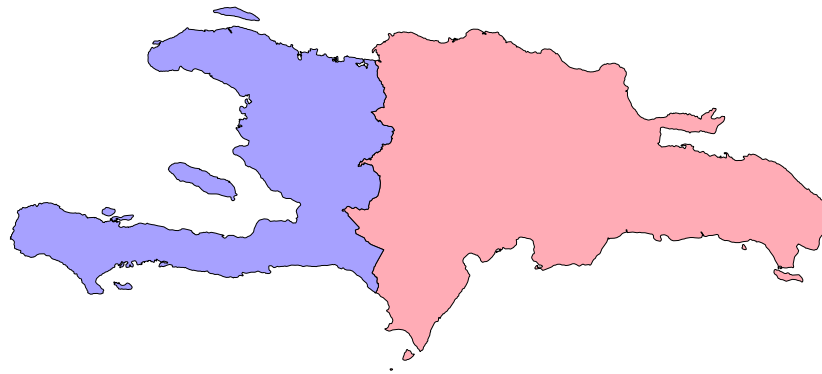
You can request more than one country at a time by providing a string matrix or a cell array of strings.

```
worldhi({'Haiti','Dominican Republic'})

ans =

1x2 struct array with fields:
    type
    otherproperty
    altitude
    lat
    long
    tag
    latlim
    longlim
    area
    latlims
    longlims
```

```
    sname
```

The geographic data structures can be displayed on a map axes with `displaym`, or extracted to vectors of latitude and longitude with `extractm`.

```
axesm miller
displaym(worldhi({'Haiti','Dominican Republic'}))
polcmap
```



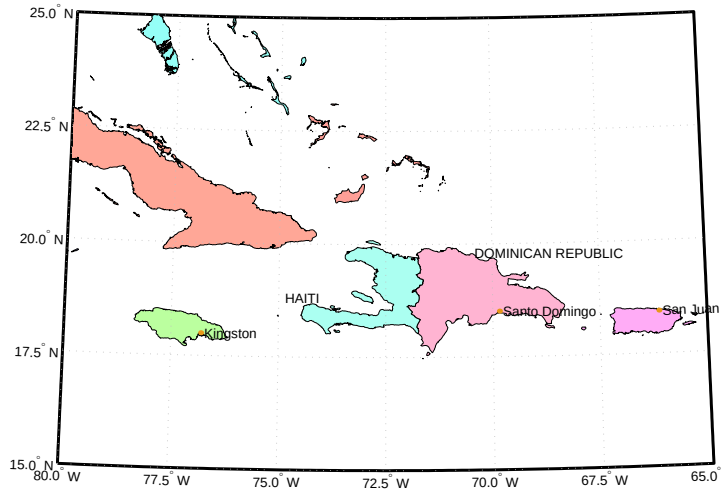
You can also have `worldhi` return all country outlines present within a geographic quadrangle. Here are the outlines for parts of countries that fall within the requested latitude and longitude limits. Note that `worldhi` omits islands or the mainland of a country with bounding boxes that fall outside the requested limits.

```
latlim = [15 25]; lonlim = [-80 -65];
s = worldhi(latlim,lonlim);
strvcat(s.tag)

ans =

Bahamas, The
Cayman Islands (UK)
Cuba
Dominican Republic
Haiti
Jamaica
Navassa Island (USA)
```

```
Puerto Rico (USA)
Turks and Caicas Islands (UK)
Virgin Islands, U.S. (USA)
```

While you can build up your own map displays using low-level functions like `axesm` and `displaym`, the easy way to make a base map from this data is with the `worldmap` function. The `worldhi` function takes care of cartographic details like selecting a projection, setting standard parallels, and grid and label increments. Here is a base map for the same area.

```
worldmap('allhi',latlim,lonlim,'patch')
hidem(gca)
```



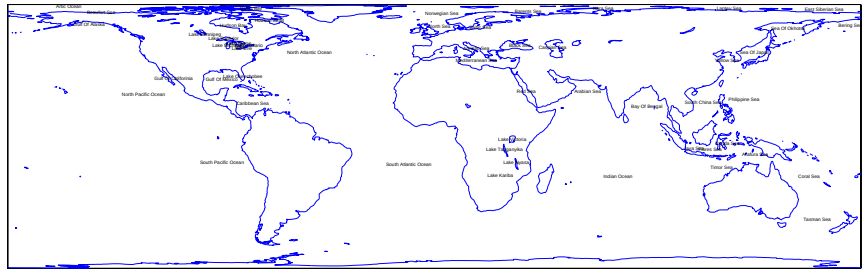
The optional arguments to `worldhi` can be used to omit islands smaller than some threshold, or to control how names are matched. You may have noticed the political affiliations in the names of some of the islands in the previous example. Here is an example that extracts all the entries in the `worldhi` database showing affiliation with the United Kingdom.

```
s = worldhi('(UK)','findstr');
strvcat(s.tag)

ans =
```

```
Alderney (UK)
Bermuda (UK)
British Indian Ocean Territory (UK)
Cayman Islands (UK)
Falkland Islands [Islas Malvinas] (UK)
Gibraltar (UK)
Gough Island (UK)
Guernsey (UK)
Jersey (UK)
Montserrat (UK)
Pitcairn Islands (UK)
Saint Helena (UK)
South Georgia and the South Sandwich Islands (UK)
Tristan da Cunha Group (UK)
Turks and Caicas Islands (UK)
United Kingdom (UK)
Virgin Islands, British (UK)
```

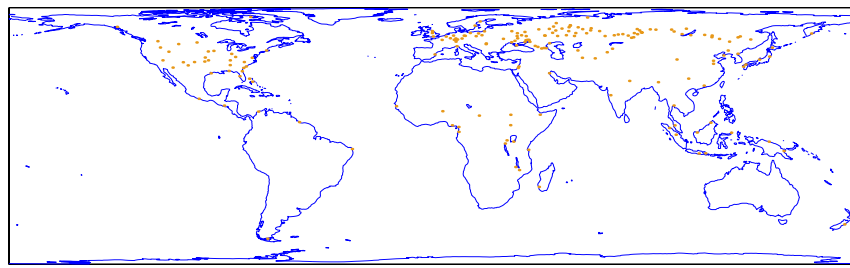
In addition to the country outline, the `worldhi` database also contains some information that complements the `worldlo` data. The `POtext` structure contains a list of ocean names that you can use to annotate map displays.

```
axesm lambcyln
load coast
plotm(lat, long)
tightmap
displaym(worldhi('POtext'))
set(handlem('alltext'),'fontsize',6)
```


Arctic Ocean
Greenland Sea
Norwegian Sea
Barents Sea
Kara Sea
Laptev Sea
East Siberian Sea
Chukchi Sea
Bering Sea
Sea Of Okhotsk
Baffin Bay
Labrador Sea
Hudson Bay
Gulf Of Alaska
Lake Winnipeg
Lake Of The Woods
Lake Superior
Lake Michigan
Lake Huron
Lake Erie
Lake Ontario
Great Salt Lake
Great Bear Lake
Great Slave Lake
Lake Athabasca
Gulf Of Saint Lawrence
North Atlantic Ocean
North Pacific Ocean
Caribbean Sea
Gulf Of Mexico
Sea Of Japan
East China Sea
Philippine Sea
South China Sea
Bay Of Bengal
Arabian Sea
Red Sea
Lake Nicaragua
Lake Titicaca
Lake Poopo
Lake Maracaibo
Lake Kariba
Lake Victoria
Lake Tanganyika
Lake Turkana
Lake Volta
Lake Nasser
Andaman Sea
Timor Sea
Arafura Sea
Coral Sea
Tasman Sea
South Pacific Ocean
South Atlantic Ocean
Indian Ocean

The `worldhi` database also includes a set of city location markers and text labels. The `worldhi` populated place structures contain cities not in `worldlo`'s similar `PPpoint` and `PPtext` structures. Here the city markers in the `worldhi` data are shown in red, while the cities in `worldlo` are orange.

```
clmo all
h = displaym(worldhi('PPpoint'));
set(h,'color','r')
displaym(worldlo('PPpoint'))
load coast
plotm(lat, long)
```



World Matrix Data

Political

The Mapping Toolbox includes a set of matrix data coded with political information in the MAT-file `worldmtxmed`.

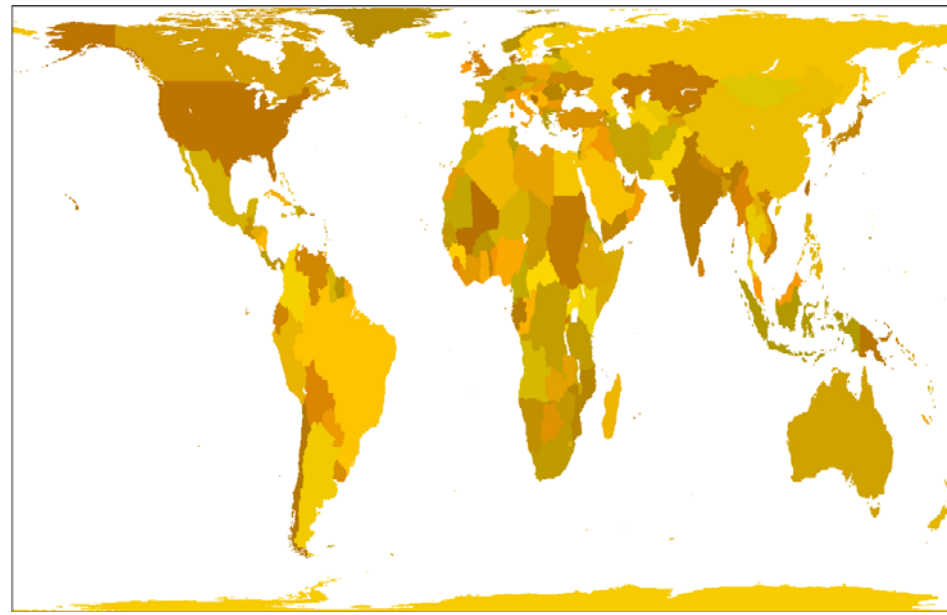
```
load worldmtxmed
whos
Name              Size                    Bytes  Class

cmap             207x3                    4968  double array
description        4x71                    568  char array
map              720x1440              1036800  uint8 array
maplegend          1x3                     24  double array
names            207x1                   16516  cell array
refvec             1x3                     24  double array
source             1x64                   128  char array
```

The variable `map` is a regular data grid containing indices of political entities. The `names` cell array relates the indices in the map to the names of countries, and the variable `cmap` is a colormap that provides a good political display of the world.

Display the political regular data grid using the `meshm` function and the provided colormap:

```
axesm gortho
meshm(map,maplegend);
colormap(cmap)
tightmap
```

The map is shown in a Gall Orthographic projection, more recently made familiar as the Peters projection. This projection is equal-area, a property that is often desirable in representing competing political units.



The resolution of this data makes it suitable for global displays. Larger scale, regional maps are better made with vector data such as that in the `worldlo` or `worldhi` databases. You can create more detailed political data grids from vector data using `vec2mtx`, `country2mtx`, and other approaches described in "GUI Reference" on page 12-1.

The primary use of this data set is to determine which country contains a point known by its latitude and longitude. What country claims sovereignty over the point at 5° E and 13° N?

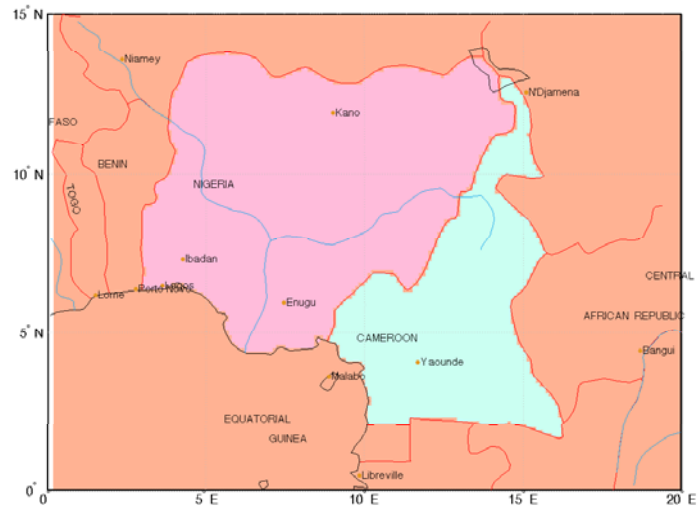
```
code = ltln2val(map,maplegend,5,13)
code =
   34

names{code}
ans =
Cameroon
```

Work through this example to see how the result was obtained. The data grid contains integer country codes. Calling the function `ltln2val` extracted the

country code of the geographic point. The country code is an index into the `names` cell array, which returns the name of the country.

You can create your own data grids at higher resolutions. If the `worldlo` atlas data is good enough for your purposes, use `country2mtx`. If you have higher resolution data, use `vec2mtx`. Here is a data grid of Cameroon and Nigeria at a resolution of 10 cells per degree, or about 10 kilometers.

```
latlim = [0 15]; lonlim = [0 20];

[cmap,cmaplegend] = country2mtx('cameroon',10,latlim,lonlim);
[nmap,nmaplegend] = country2mtx('nigeria',10,latlim,lonlim);

newmap = zeros(size(cmap)); newmaplegend = cmaplegend;

newmap(cmap==0) = 34;  % find cells in the interior
newmap(nmap==0) = 130; % use codes from nations structure

worldmap(latlim,lonlim)
meshm(newmap,newmaplegend,size(newmap))
polcmap
```



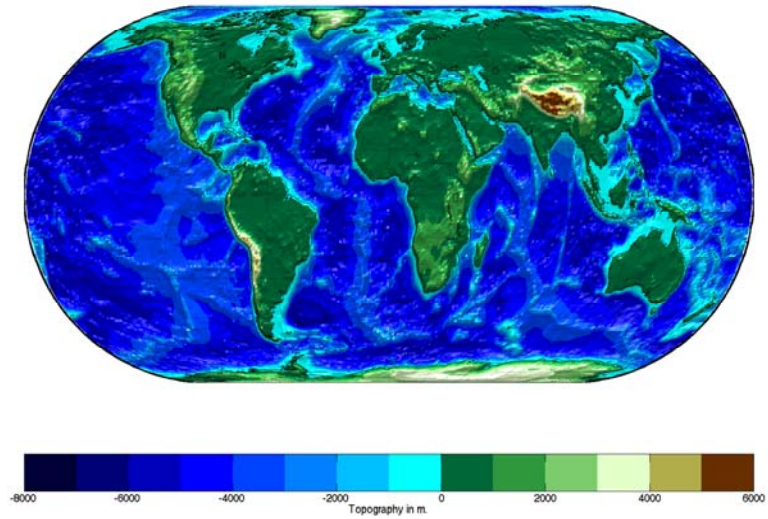
Terrain

A low resolution set of global elevation data is provided with MATLAB in the `topo` MAT-file. It is a regular matrix of average elevations and depths (in meters) for a constant grid spacing of one degree by one degree.

```
load topo
whos
Name             Size                    Bytes  Class

  topo           180x360                518400  double array
  topolegend       1x3                      24  double array
  topomap1        64x3                    1536  double array
  topomap2       128x3                    3072  double array

Grand total is 65379 elements using 523032 bytes
```

Here it is displayed in an equal-area Eckert IV projection as a shaded-relief surface, which allows both large and small features to be seen:

```
axesm eckert4; framem; gridm
load coast
plotm(lat, long,'k'); zdatam(handlem('line'),max(topo(:)))

meshm(topo,topolegend,size(topo),topo,'FaceColor','interp')
demcmap('inc',topo,1000)

camlight(0,80);; material([0.7 0.8 .4]); daspectm('m',300)
hcb = colorbar('horiz')
set(get(hcb,'XLabel'),'String','Topography in m.')
```



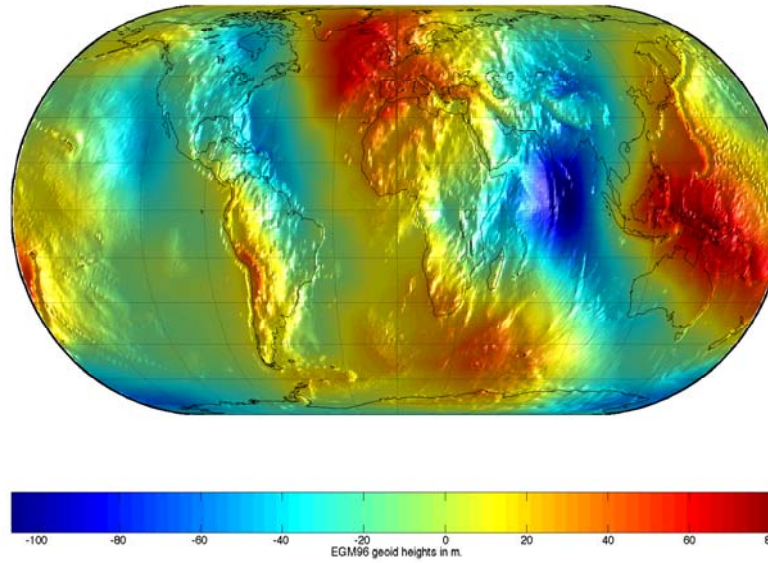
Geoid

While the `topo` matrix contains the heights and depths of the Earth's crust, the `geoid` matrix gives the shape of its gravitational field. The geoid can be regarded as the ocean surface formed in the absence of waves, currents, tides, and land obstructions. In technical terms, it is a gravitational equipotential surface. Elsewhere, this document refers to the ellipsoidal shape of the Earth as the geoid. Although this is sufficient for cartography, the ellipsoidal approximation is not accurate enough for applications like surveying and geodesy, and more detailed models are used. The EGM96 geoid model is a widely used spherical harmonic model of the geoid complete to degree and order 360. A 1-degree grid of geoid heights is provided in the `geoid.mat` atlas file. You can read the full resolution version of the matrix of geoid heights with a grid spacing of 15 minutes using the `egm96geoid` external interface function.

```
load geoid
figure; axesm eckert4; framem; gridm

load coast
plotm(lat, long,'k')
zdatam(handlem('line'),max(geoid(:)))
meshm(geoid,geoidlegend,size(geoid),geoid,'Facecolor','interp')
```

```
light; material(0.6*[ 1 1 1])

set(gca,'dataaspectratio',[ 1 1 200])
hcb = colorbar('horiz')
set(get(hcb,'Xlabel'),'String','EGM96 geoid heights in m.')
```

Here is the result:




```
z = ltln2val(geoid,geoidlegend,-11.1,130.22,'bicubic')
z =
   53.0055
```

Interpolating the full-resolution grid gives a result that is about half a meter higher.

United States Vector Data

Low-Resolution Data

Vector political and coastline data for the United States is provided in a number of useful formats in the `usalo` MAT-file.

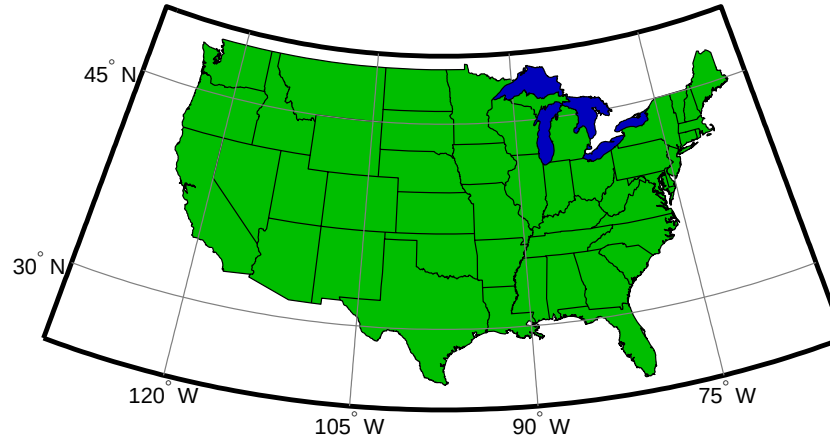
```
load usalo
whos
Name              Size                    Bytes  Class

  conus             1x1                   70224  struct array
  description       2x59                    236  char array
  greatlakes        1x3                   21612  struct array
  gtlakelat      1229x1                    9832  double array
  gtlakelon      1229x1                    9832  double array
  source            1x45                     90  char array
  state             1x51                  246970  struct array
  stateborder       1x1                   38358  struct array
  statelat       2345x1                   18760  double array
  statelon       2345x1                   18760  double array
  uslat          4339x1                   34712  double array
  uslon          4339x1                   34712  double array

Grand total is 61684 elements using 504098 bytes
```

There are several types of data here. The double arrays are vectors of latitude and longitude points. The pair `uslat` and `uslon` form the outline of the coast and political borders of the continental United States. The polygon closes on itself, so it can be filled as a patch. Next, the `statelat` and `statelon` form the borders between the states for display as lines. Finally, `gtlakelat` and `gtlakelon` are patchable outlines of the Great Lakes. The functions suitable for displaying vector data of this type are `plotm`, `linem`, `fillm`, `patchm`, and `patchesm`.

Display the data vectors as patches and lines:

```
axesm('MapProjection','eqaconic','MapParallels',[29.5 49.5],...
   'MapLatLimit',[24 50],'MapLonLimit',[-130 -65],...
   'MLabelLocation',15,'MLineLocation',15,...
   'PLabelLocation',15,'PLineLocation',15,...
   'GColor',.5*[1 1 1],'GLineStyle','-',...
```

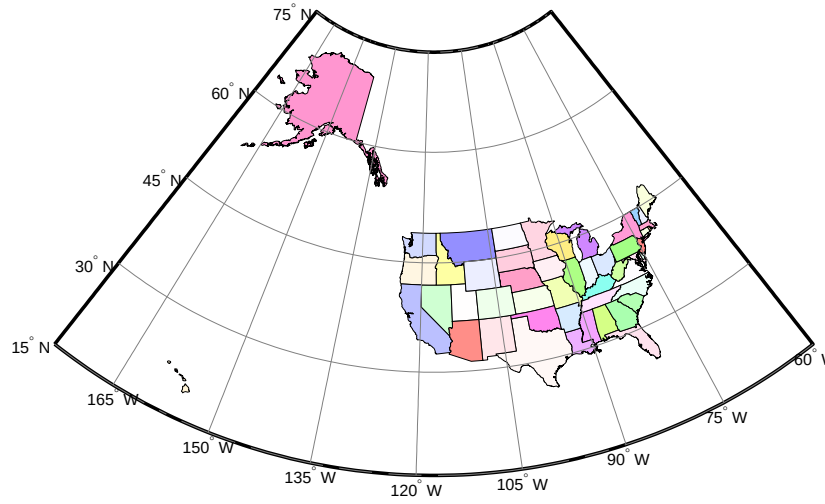
```
    'MLabelparallel','south')
framem; gridm; mlabel; plabel
patchm(uslat,uslon,[0 .75 0])
patchm(gtlakelat,gtlakelon,1,[0 0 .75])
plotm(statelat,statelon,'k')
```



The structure arrays are Mapping Toolbox geographic data structures and can be displayed using `mlayers` or `displaym`. The data in `stateborder` is identical to that in `statelat` and `statelon`. Similarly, `conus` contains the same outline of the continental United States as `uslat` and `uslon` but is defined to be displayed as a patch. The 50 state and District of Columbia patches are located in the `state` structure.

Display the `state` structure using `displaym`. Notice that only one command is needed to plot 51 different patches.

```
figure
axesm('MapProjection','eqaconic','MapParallels',[],...
    'MapLatLimit',[15 75],'MapLonLimit',[-175 -60],...
    'MLabelLocation',15,'MLineLocation',15,...
    'PLabelLocation',15,'PLineLocation',15,...
    'GColor',.5*[1 1 1],'GLineStyle','-',...
    'MLabelParallel','south')
framem; gridm; mlabel; plabel
displaym(state); polcmap
```

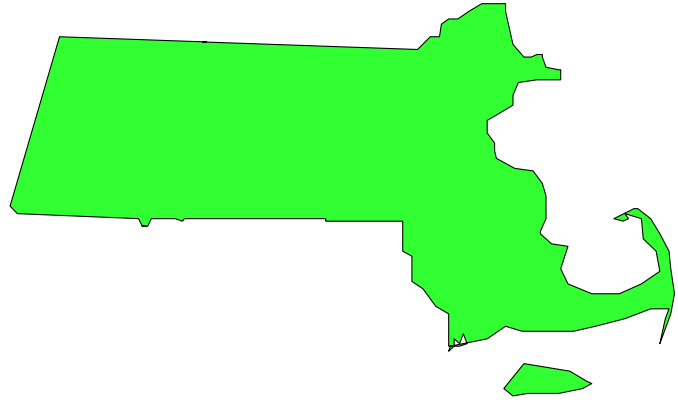


The states are all tagged, allowing them to be manipulated or to be extracted by name. You can see the available state names by entering the following:

```
strvcat(state.tag)
```

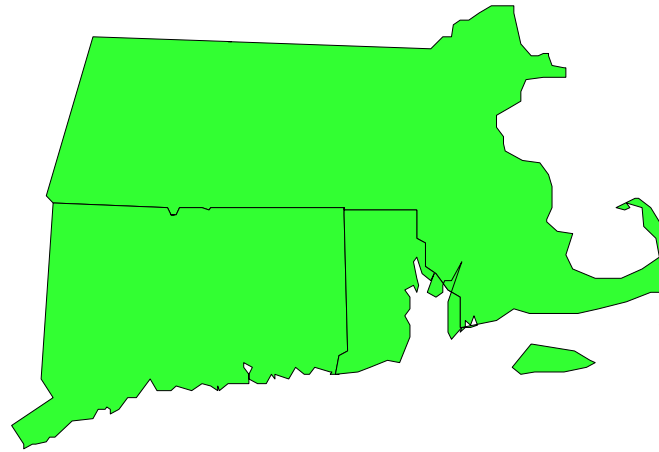
The state of Massachusetts can be displayed by itself:

```
displaym(state,'mass');
```



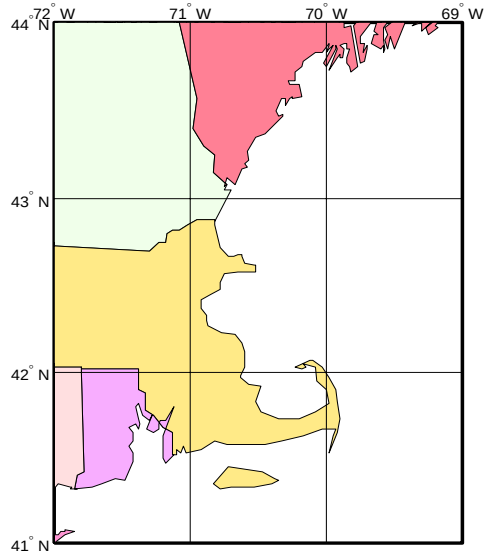
Or with its neighboring southern states, Connecticut and Rhode Island:

```
displaym(state,strvcat('mass','conn','rhode'));
```



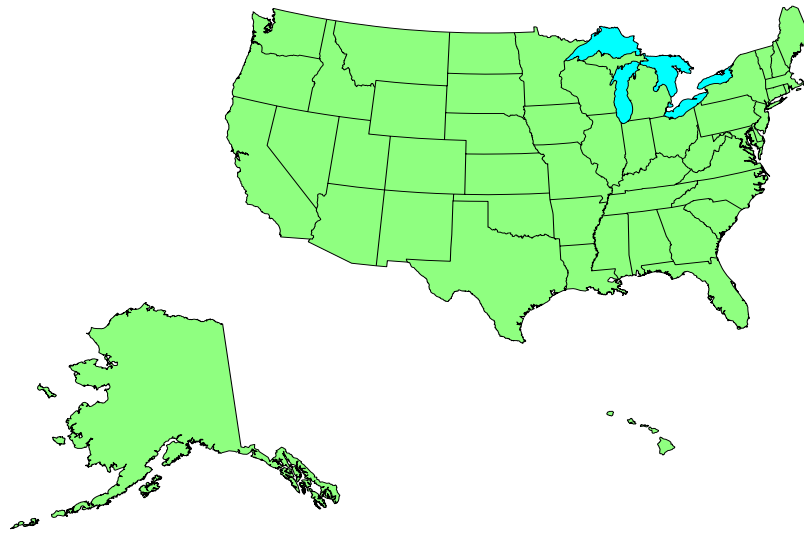
Look at the region around Cape Cod again, to get an impression of the detail of the data:

```
figure
axesm('MapProjection','mercator','Frame','on',...
   'MapLatLimit',[41 44],'MapLonLimit',[-72 -69])
gridm('MLineLocation',1,'PLineLocation',1,...
   'GLineStyle','-')
mlabel('MLabelLocation',1)
plabel('PLabelLocation',1)
displaym(state); polcmap
```



There is an obvious compromise between the size of a data set and its accuracy and detail. This data can be displayed relatively quickly for displays of all the United States, but it is not suitable for mapping smaller regions. Note the generalized character of the coastline and the absence of small islands. More detailed and accurate data is available in the `usahi` MAT-file and through the Mapping Toolbox External Data Interface functions.

An easy way to display the `usalo` data is with the `usamap` function. The `usamap` function takes care of the cartographic details. The inset maps of the entire United States are made with the `usalo` data. Here are the fifty states displayed at the same scale.



Medium Resolution State Outlines

Another more detailed set of state outlines is available in the Mapping Toolbox. The data in the `usahi` MAT-file is similar in format to the `state` structure found in `usalo` but contains more detailed coastlines and islands. The resolution of `stateline` in the `usahi` MAT-file is about three times greater than the corresponding `state` in the `usalo` MAT-file.

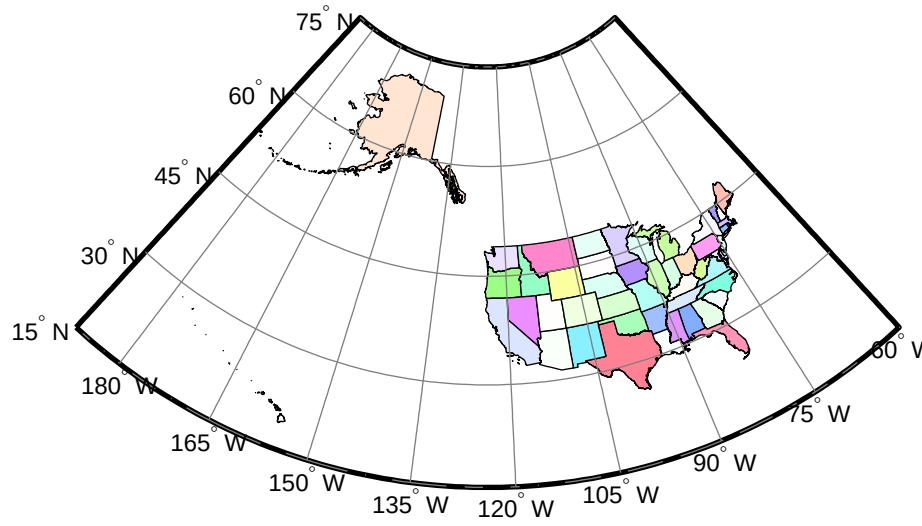
```
load usahi
whos
Name              Size                    Bytes  Class

  ans               1x60                      120  char array
  description       2x62                      248  char array
  source            1x58                      116  char array
  stateline         1x51                   826456  struct array
  statepatch        1x51                   826558  struct array
  statetext         1x51                    40704  struct array
```

The listed variables are all formatted geographic data structures containing lines, patches, or text. The data in the `stateline` structure is the same as that in `statepatch` but has been defined to be displayed as lines rather than patches. The structure `statetext` contains the corresponding names of the states. Note that because of the higher resolution of this data, it might require more than the default memory size to display and will take longer to project and render. This difference is particularly marked for patches.

Display the map in an Equal-Area Conic projection:

```
axesm('MapProjection','eqaconic','mapparallels',[],...
   'MapLatLimit',[15 75],'MapLonLimit',[-188 -60],...
   'MLabelLocation',15,'MLineLocation',15,...
   'PLabelLocation',15,'PLineLocation',15,...
   'GColor',.5*[1 1 1],'GLineStyle','-',...
   'MLabelParallel','south')
framem; gridm; mlabel; plabel
displaym(statepatch); polcmap
```

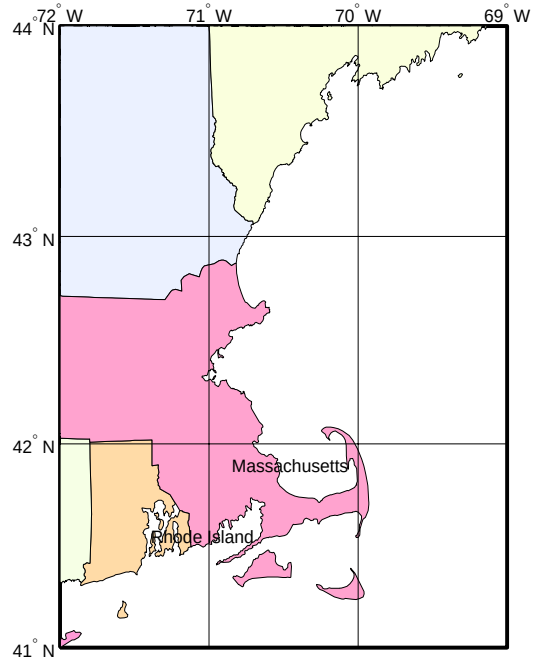
Here is the result:



At this scale, it is difficult to see the difference between this data and the much smaller `usalo` MAT-file. If you focus on the Cape Cod region, the higher resolution becomes more apparent:

```
figure
axesm('MapProjection','mercator','Frame','on',...
    'MapLatLimit',[41 44],'MapLonLimit',[-72 -69])
gridm('MLineLocation',1,'PLineLocation',1,'GLineStyle','-')
mlabel('MLabelLocation',1)
plabel('PLabelLocation',1)
displaym(statepatch)
h = displaym(statetext); trimcart(h)
polcmap
```

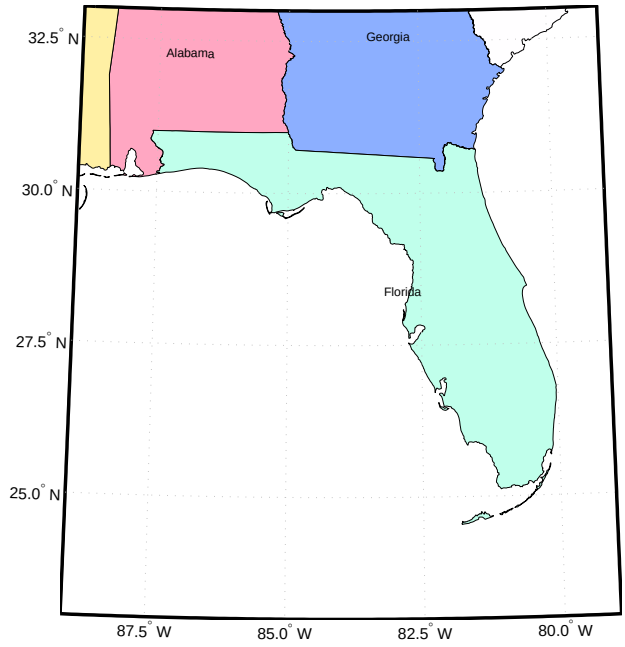
Here is the zoomed-in region of Cape Cod. The level of detail is noticeably greater than that for the data in the `usalo` MAT-file.



You can use `usamap` to quickly create base maps with this data. Just provide the state name or geographic limits. This function handles the cartographic

details like selecting a projection, setting the map limits, and setting the origin and the grid and label spacings. You can also use the `usahi` interface function to load one of the atlas data structures as an argument to a function, or load just that structure into the workspace.

```
figure
usamap florida
```



United States Matrix Data

Political

The Mapping Toolbox also provides a set of political matrix data for the continental United States, located in the `usamtx` MAT-file.

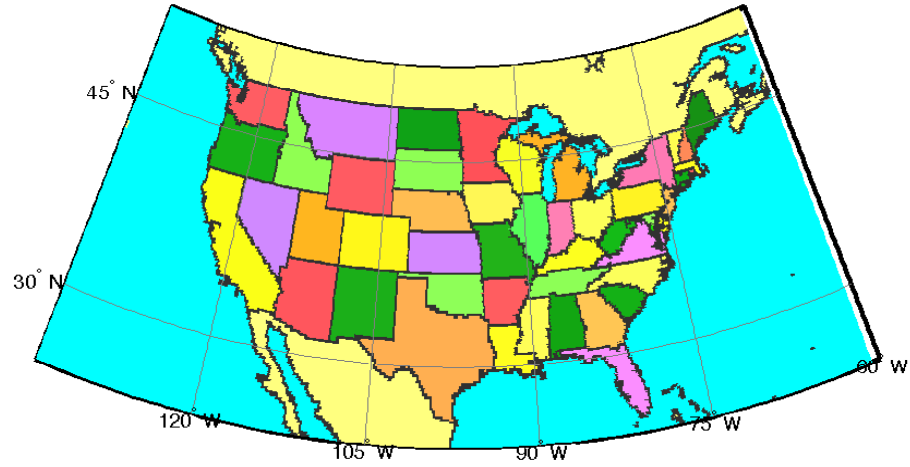
```
load usamtx
whos
Name              Size                    Bytes  Class

  clrmap           54x3                    1296  double array
  description       4x74                    592  char array
  map             140x370                 414400  double array
  maplegend         1x3                      24  double array
  refvec            1x3                      24  double array
  source            1x68                    136  char array
  states           54x15                   1620  char array
```

The data is in a regular data grid format, similar to that discussed in the "World Matrix Data" section. The variable `map` is at a resolution of 5 cells per degree, or roughly 20 kilometers or better. Recall that the resolution of the `worldmtx` data is 1 cell per degree; hence, this data set is five times finer. A colormap has been provided, along with a vector of state names, both corresponding to the political code, or *index*, used in the `map` matrix.

Display the data grid in an Albers Equal-Area Conic projection:

```
axesm('MapProjection','eqaconic','MapParallels',[],...
   'MapLatLimit',[24 52],'MapLonLimit',[-134 -60],...
   'MLabelLocation',15,'MLineLocation',15,...
   'PLabelLocation',15,'PLineLocation',15,...
   'GColor',.5*[1 1 1],'GLinestyle','-',...
   'MLabelParallel','south')
framem; gridm; mlabel; plabel
meshm(map,maplegend); colormap(clrmap)
```

Here is the result:



As with the world matrix data, the low resolution of this data makes it suitable for only large area displays. Vector maps are more appropriate where more detail is required. The functions itemized under “Georeferenced Images and Data Grids” on page 10-11 inlcude tools for creating higher resolution data grids from vector maps.

The primary use of this data set is to determine which state contains a particular geographic point. In what state is Alamogordo (32.8990° N and 105.9570° W)?

```
code = ltln2val(map,maplegend,32.8990,-105.9570)
code =
   34

states(code,:)
ans =
New Mexico
```

You can create your own higher resolution political data grids using `vec2mtx`.

Terrain

As an example of a higher resolution digital elevation map, MATLAB provides the `cape` MAT-file, containing an image of elevation data for the northeastern

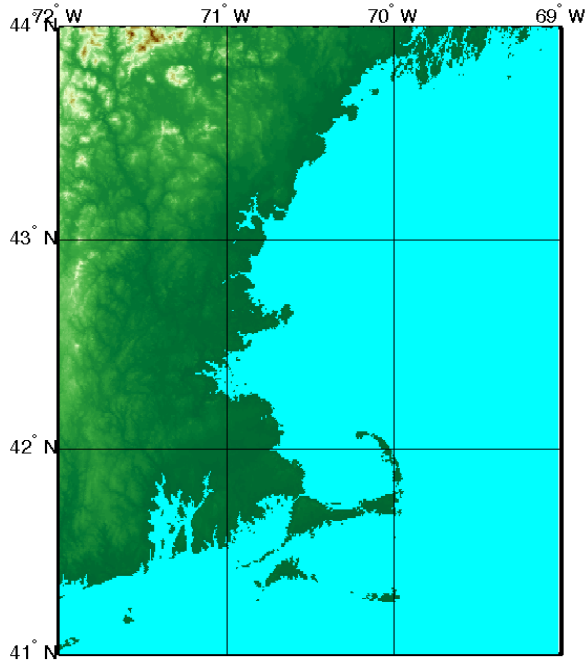
United States on a 30 arc-second grid (resolution of about one kilometer or better on the ground). The data can be defined as a regular data grid using the `loadcape` function script, which rearranges the data and provides the necessary map legend:

```
loadcape
whos
Name              Size                    Bytes  Class

  clrmap           54x3                    1296  double array
  description       4x74                    592  char array
  map             140x370                414400  double array
  maplegend         1x3                      24  double array
  refvec            1x3                      24  double array
  source            1x68                    136  char array
  states           54x15                   1620  char array

Grand total is 53142 elements using 418092 bytes
```

Here the elevation data is displayed using a conformal Mercator projection, so shapes of small regions suffer little distortion, while the distortion in relative areas is scarcely noticeable for such a small region.

```
axesm('MapProjection','mercator',...
   'MapLatLimit',[41 44],'MapLonLimit',[-72 -69])
framem
gridm('MLineLocation',1,'Plinelocation',1,'GLineStyle','-')
mlabel('MLabelLocation',1)
plabel('PLabelLocation',1)
meshm(map,maplegend); demcmap(map)
```

Here is the result:



Global coverage of digital terrain and bathymetry at this resolution is provided through the Mapping Toolbox External Data Interface. A variety of freely available digital elevation maps is available over the Internet for import into MATLAB. These maps range in resolution from about 10 km to 30 meters. See “Geospatial Data Import and Access” on page 10-6 for more information on the data and import functions.

Astronomical Data

Although the Earth may be the most commonly mapped object, the same cartographic techniques are used to map the stars and planets. As an example of such astronomical data, the Mapping Toolbox includes map data for the Earth's moon and the stars.

The moon data is low-resolution topography and albedo (reflectance) from the Clementine spacecraft. Topography, found in the `moontopo.mat` file, is a 1-degree regular data grid in units of meters. Albedo is stored as a 3-cells-per-degree integer image in the `moonalb.mat` file as double-precision numbers. Here is the albedo image texture mapped onto the topography, vertically exaggerated by a factor of 10.

```
load moontopo; load moonalb
axesm('globe','geoid',almanac('moon','radius','m'))

h = meshm(moontopo,moontopolegend,size(moontopo),10*moontopo);
set(h,'CData',moonalb,'FaceColor','texturemap')
colormap(gray); view(20,20)
camlight; material dull; lighting phong
```

Star positions and magnitudes are found in the `stars` MAT-file.

```
load stars
whos
Name               Size                    Bytes  Class

  description        1x74                      148  char array
  glat            4652x1                     37216  double array
  glong           4652x1                     37216  double array
  lat             4652x1                     37216  double array
  long            4652x1                     37216  double array
  relintensity    4652x1                     37216  double array
  source             1x82                      164  char array
  vmag            4652x1                     37216  double array

Grand total is 28068 elements using 223608 bytes
```

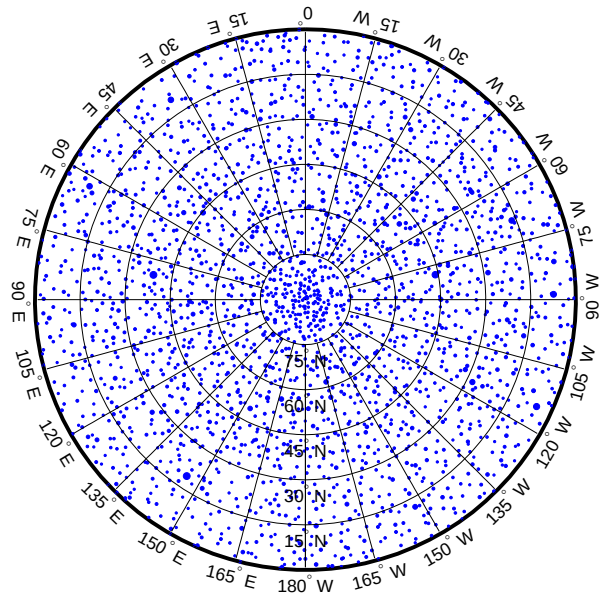
This is a set of more than 4500 star locations and visual magnitudes derived from the Fifth Fundamental Catalog of Stars parts I and II (FK5 and FK5e). Other star data such as mean errors, proper motions, spectral types, parallaxes, radial velocities, and cross identifications to other catalogues can be obtained from the catalog using the `readfk5` interface function.

The positions of stars in equatorial coordinates are given in the vectors `lat` and `long`. These are latitudes and longitudes in degrees for the stars as seen from within the celestial sphere. The visual magnitude of each star is given in the vector `vmag`. Stars are typically plotted with diameters proportional to the relative intensity. The vector `relintensity` contains modified `vmag` data suitable for display with `scatterm` to exhibit this proportional diameter. All intensities are positive, and brighter stars have a larger `relintensity` number.

Display the stars of the northern sky in the Equidistant azimuthal projection. Scale the relative intensities to

```
axesm eqdazim
framem; gridm; mlabel; plabel
setm(gca,'Origin',[90 180 0],'FLatLimit',[-inf 90],...
   'MLabelLocation',-180:15:165,'MLineLocation',15,...
   'MLineLimit',[-75 75],'MLabelParallel','equator',...
   'PLabelMeridian',180,'PLabelLocation',15:15:75,...
   'LabelRotation','on''GLineWidth',.01,...
```

```
    'GLineStyle','-')
set(handlem('alltext'),...
    'VerticalAlignment','top','HorizontalAlignment','center')
h = scatterm(lat,long,sqrt(relintensity)*30,'filled');
```

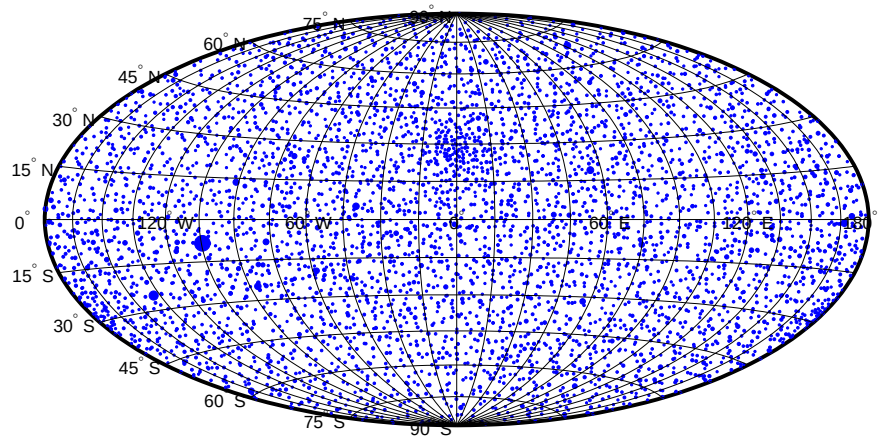


You may be able to identify the Big Dipper, or the constellation of Ursa Major, located at 180° E, 60° N.

Another system used to describe the positions of heavenly bodies is galactic coordinates. This coordinate system puts the center of our galaxy at the origin and places the North Pole so that the Milky Way is aligned with the galactic equator. The positions of the stars in this coordinate system are given in the vectors `glat` and `glong`. Stars in galactic coordinates are typically plotted in a pseudocylindrical projection like the Hammer:

```
figure
axesm hammer
framem; gridm; mlabel; plabel
setm(gca,'GLineWidth',.01,'GLineStyle','-',...
    'MLineLocation',15,'MLabelParallel','equator',...
    'MLabelLocation',-120:60:180)
```

```
scatterm(glat,glong,sqrt(relintensity)*30,'filled')
```



You can also render point symbols by value using `geoshow` and `mapshow` with symbolspecs.

B

Bibliography

1 Snyder, J. P., *Map Projections - A Working Manual*, U.S. Geological Survey Professional Paper 1395, Washington, D.C., 1987.

2 Maling, D. H., *Coordinate Systems and Map Projections*, 2nd Edition, Pergamon Press, New York, NY, 1992.

3 Snyder, J. P. and Voxland, P. M., *An Album of Map Projections*, U.S. Geological Survey Professional Paper 1453, Washington, D.C., 1994.

4 Snyder, J. P., *Flattening the Earth - 2000 Years of Map Projections*, University of Chicago Press, Chicago, IL, 1993.

Glossary

This glossary of geographical terms is drawn extensively from *An Album of Map Projection, U.S. Geological Survey Professional Paper 1453*, by John P. Snyder and Philip M. Voxland.

Because the purpose of this glossary is to assist in understanding and using the Mapping Toolbox, it includes some terms are specific to the toolbox, and gives some other terms shades of meaning beyond their general definitions.

Antipodes Two points on opposite sides of a planet.

Arc-second 1/3600th of a degree (1 second) of latitude or longitude.

Aspect The conceptual placement of a projection system in relation to the Earth's axis (direct, normal, polar, equatorial, oblique, and so on).

Attribute In vector geodata, a quantitative or qualitative descriptor of a spatial entity. An attribute can describe a real-world quality (such as population or land area), or a graphic quality (such as patch color or line weight). Attributes are frequently coded as numbers or strings in character-coded or binary tabular data files., with one or more attribute per map feature.

Authalic projection *See* Equal-area projection.

Axes *See* Map axes.

Azimuth The angle a line makes with a meridian, taken clockwise from north.

Azimuthal projection A projection on which the azimuth or direction from a given central point to any other point is shown correctly. When a pole is the central point, all meridians are spaced at their true angles and are straight radii of concentric circles that represent the parallels. Also called a zenithal projection.

Bathymetry The measurement of water depths of oceans, seas, lakes, and other bodies of water.

Bowditch, Nathaniel A late 18th/early 19th century mathematician, astronomer, and sailor who "wrote the book" on navigation. John Hamilton Moore's *The Practical Navigator* was the leading navigational text when Bowditch first went out to sea, and had been for many years. Early in his first voyage, however, Bowditch began noticing errors in Moore's book, which he recorded and later used in preparing an American edition of Moore's work. The revisions were to such an extent that Bowditch was named the principal author, and the title was changed to *The New American Practical Navigator*, published in 1802. In 1868 the U.S. Navy bought the copyright to the book, which is still commonly referred to as "Bowditch" and considered the "bible" of navigation.

Buffer zone The locus of points that lie within a specified distance from a map feature.

Cartography The art or practice of making charts or maps. *See* map.

Categorical geodata Geospatial data in which raster pixel values (or vector data attributes) are categorical indices, usually coded as integers. The meanings of the categories are usually stored in a separate table. Examples are geocodes, landuse categories, and indexed color images. The `worldmtx` dataset contains an example of categorical data. Each entry in the raster dataset is an index into a data structure containing the names of the world countries. *See* Numerical geodata.

Central meridian The meridian passing through the center of a projection, often a straight line about which the projection is symmetrical.

Central projection A projection in which the Earth is projected geometrically from the center of the Earth onto a plane or other surface. The Gnomonic and Central Cylindrical projections are examples.

Choropleth A map portraying regions of of homogeneous classified attribute values, changing abruptly at region boundaries, and colored or shaded according to their attribute values. Thematic political maps are usually choropleth maps.

Complex curves Curves that are not elementary forms such as circles, ellipses, hyperbolas, parabolas, and sine curves, such as rivers, coastlines, and administrative boundaries.

Composite projection A projection formed by connecting two or more projections along common lines such as parallels of latitude, necessary adjustments being made to achieve fit. The Goode Homolosine projection is an example.

Conformal projection A projection on which all angles at each point are preserved, except at a finite number of singular points (e.g., the poles in a Mercator projection). Also called an orthomorphic projection.

Conic projection A projection resulting from the conceptual projection of the Earth onto a tangent or secant cone, which is then cut lengthwise and laid flat. When the axis of the cone coincides with the polar axis of the Earth, all meridians are straight equidistant radii of concentric circular arcs representing the parallels, but the meridians are spaced at less than their true angles. Mathematically, the projection is often only partially geometric.

Constant scale A linear scale that remains the same along a particular line on a map, although that scale may not be the same as the stated or nominal scale of the map.

Contour All points that are at the same height above or below a reference datum; generally applied to continuous, single-valued surfaces only, such as elevation, temperature, or magnetic field strength.

Conventional aspect *See* Normal aspect.

Correct scale A linear scale having exactly the same value as the stated or nominal scale of the map, or a scale factor of 1.0. Also called true scale.

Cylindrical projection A projection resulting from the conceptual projection of the Earth onto a tangent or secant cylinder, which is then cut lengthwise and laid flat. When the axis of the cylinder coincides with the axis of the Earth, the meridians are straight, parallel, and equidistant, while the parallels of latitude are straight, parallel, and perpendicular to the meridians. Mathematically, the projection is often only partially geometric.

Data Grid A raster dataset consisting of an array of values posted or sampled at specific geographic points. In the Mapping Toolbox, data grids can be implicit (regular) or explicit (irregular), depending on the uniformity of the grid. *See* Regular data grid, Geolocated data grid.

Datum (vertical) A base reference level for establishing the vertical dimension of elevation for the earth's surface. A datum defines sea level and incorporates an ellipsoid; thus one can reference a coordinate system to a datum or to a specified ellipsoid, but not both at the same time.

Datum (horizontal) A base measuring point ("0.0 point") used as the origin of rectangular coordinate systems for mapping or for maintaining excavation provenience. Two examples are the North American Datum of 1927 (NAD27) and the North American Datum of 1983 (NAD83). Earth-centered coordinate systems, such as WGS84, combine horizotal and vertical datums.

Dead reckoning From “deduced reckoning,” the estimation of geographic position based on course, speed and time.

DEM (Digital Elevation Map/Model) Elevation data in the form of a data grid, generally a regular (implicit) one. DEM also refers to the five primary types of digital elevation models produced by the U.S. Geological Survey; the Mapping Toolbox can read 30-meter and 10-meter DEMs as well as 3-second DEMs.

Departure The arc length distance along a parallel of a point from a given meridian.

Developable surface A simple geometric form capable of being flattened without stretching. Many map projections can be grouped by a particular developable surface: cylinder, cone, or plane.

Direct aspect *See* Normal aspect.

Distortion A variation of the area or linear scale on a map from that indicated by the stated map scale, or the variation of a shape or angle on a map from the corresponding shape or angle on the Earth.

DMS Degrees-minutes-seconds angle notation of the form ddd° mm' ss". There are 60 seconds in a minute, and 60 minutes in a degree. In the Mapping Toolbox, when “dms” angles are represented by a single number, the format is dddmm.ss.

Easting The distance of a point eastward from the origin in the units of the coordinate system for the defined projection. Paired with *Northings*.

Ellipsoid When used to represent the Earth, a solid geometric figure formed by rotating an ellipse about its minor (shorter) axis. Also called spheroid.

Ellipsoid vector A vector describing a specific ellipsoid, model. The ellipsoid vector has the form:

```
ellipsvec = [semimajor-axis eccentricity]
```

Ellipsoidal height Elevation of a point above a reference ellipsoid, as measured along a normal to the ellipsoid.

Equal-area projection A projection on which the areas of all regions are shown in the same proportion to their true areas. Shapes may be greatly distorted. Also called an equivalent or authalic projection.

Equator The great circle straddling a planet at a latitude of 0°, perpendicular to its polar axis and midway along it, dividing the northern and southern hemispheres.

Equatorial aspect An aspect of an azimuthal projection on which the center of projection or origin is some point along the Equator. For cylindrical and pseudocylindrical projections, this aspect is usually called conventional, direct, normal, or regular rather than equatorial.

Equidistant projection A projection that maintains constant scale along all great circles from one or two points. When the projection is centered on a pole, the parallels are spaced in proportion to their true distances along each meridian.

Equireal projection *See* Equal-area projection.

Equivalent projection — *See* Equal-area projection.

False Easting — The value of the easting assigned to the projection origin. Easting values increase to the east.

False Northing — The value of the northing assigned to the projection origin. Northing values increase to the north.

Flat-polar projection — A cylindrical projection on which, in normal aspect, the pole is shown as a line rather than as a point. For example, the Miller projection is flat-polar

Frame — *See* Map frame.

Free of distortion — Having no distortion of shape, area, or linear scale. On a flat map, this condition can exist only at certain points or along certain lines.

Geodesic — A minimum-distance curve on a curved surface, independent of the choice of a coordinate system. On a sphere a geodesic is equivalent to a great circle arc.

Geolocated data grid — A data grid defined with separate latitude, longitude, and value matrices, allowing irregular sampling, non-rectangular shapes, and non-cardinal orientations. Satellite imagery swaths are often represented as geolocated data grids. *See* Data grid, Regular data grid

Geodata — Geospatial data. *See* Geospatial

Geoid — The figure of the earth less its topography, defined as an equipotential surface with respect to gravity, more or less corresponding to mean sea level. It is approximately an oblate ellipsoid, but not exactly so because local variations in gravity create minor hills and dales. Empirically determined geoids are used to define *datums* and to compute orbital mechanics.

Geometric projection — *See* Perspective projection.

Geographic coordinates — Spherical 2D coordinate tuples (latitudes, longitudes) that specify point locations for unprojected geodata. The analogous term for geodata projected to a rectangular coordinate system is *Map Coordinates*.

Geographic data structure — In the Mapping Toolbox, a MATLAB structure array with one element per vector geographic feature. It includes a `Geometry` or `type` field, at least two coordinate array fields (`X` and `Y`, `Lat` and `Lon`, or `lat` and `long`), and optional attribute fields.

Georeferencing Identifying objects and locations by name, identifier, or coordinates to describe where they are located on the Earth's surface.

Geospatial Spatial data, concepts, and techniques that specifically refer to geographic space or phemonena, and not just to arbitrary coordinate systems or abstract space frames.

GeoTIFF An extension of the TIFF image file format with additional tags containing parameters for image georeferencing and projected map coordinate system definition.

GIS (Geographic Information System) A system, usually computer based, for the input, storage, retrieval, analysis, and display of interpreted geographic data.

Globular projection Generally, a nonazimuthal projection developed before 1700 on which a hemisphere is enclosed in a circle and meridians and parallels are simple curves or straight lines.

Graticule A network of lines representing a subset of the Earth's parallels and meridians (or plane coordinates) used as a reference grid on globes and maps. Generally synonymous with *map grid*, except that many map grids are rulings at regular intervals in projected coordinates. *See* Map grid, National grid (U.S.), National grid (U.K.). The vertices of the graticule grid are precisely projected, and the map data contained in any grid cell is warped to fit the resulting quadralateral. A finer graticule grid results in a higher projection fidelity at the expense of greater computational requirements.

Great circle Any circle on the surface of a sphere, especially when the sphere represents the Earth, formed by the intersection of the surface with a plane passing through the center of the sphere. It is the shortest path between any two points along the circle and therefore important for navigation. All meridians and the Equator are great circles on the Earth taken as a sphere.

Grid *See* Map grid, Data grid.

HMS Hours-minutes-seconds time notation of the form hh° mm' ss". In the Mapping Toolbox, when "hms" times are represented by a single number, the format is hhmm.ss.

Homalographic /homolographic projection *See* Equal-area projection.

Hydrography The science of measurement, description, and mapping of the surface waters of the Earth, especially with reference to their use in navigation. The term also refers to those parts of a map collectively that represent surface waters and drainage.

Hydrology The scientific study of the waters of the Earth, especially with relation to the effects of precipitation and evaporation upon the occurrence and character of ground water.

Hypsographic tints A graphic means of representing terrain or other scalar attributes using a sequence of colors or tints indexed to elevation.

Hypsography The scientific study of the Earth's topologic configuration above sea level, especially the measurement and mapping of land elevation.

Index Map A small-scale map used to help locate a map containing a region or feature of interest in a tiled geospatial database, map series, plat book or atlas.

Indicatrix A circle or ellipse useful in illustrating the distortions of a given map projection. Indicatrices are constructed by projecting infinitesimally small circles on the Earth onto a map, and giving them visible dimensions. Their axes lie in the directions of and are proportional to the maximum and minimum scales at their point locations. Often called a Tissot indicatrix after the originator of the concept. In the Mapping Toolbox, Tissot indicatrices may be displayed using the `tissot` command, and indicatrices for all supported projections are provided in the “Projections Reference” chapter of the online Mapping Toolbox reference documentation.

Interrupted projection A projection designed to reduce peripheral distortion by making use of separate sections joined at certain points or along certain lines, usually the Equator in the normal aspect, and split along lines that are usually meridians. There is normally a central meridian for each section. The Mapping Toolbox does not include interrupted projections, but the user can separate data into sections and project these independently to achieve this effect.

Large-scale mapping Mapping at a scale larger than about 1:75,000, although this limit is somewhat flexible. Includes cadastral, utility, and some topographic maps.

Latitude (astronomical) The complement of the elevation angle of the celestial North Pole, which depends on normal to the Earth's equipotential surface (geoid) at a given point (positive if the point is north of Equator, negative if it is south). It can be thought of as the angle that a plumb line makes with the equatorial plane.

Latitude (auxiliary) Intermediate forms of latitude that are mathematically constructed (normally by transferring latitudes first from an ellipsoid to a sphere, and then to a plane) in order to achieve desired map projection properties. Types include *conformal* (for constructing conformal maps), *authalic* (for constructing equal-area maps), and *rectifying* (for constructing equidistant maps).

Latitude (geocentric) The angle at which a line connecting the surface of a sphere or reference ellipsoid to its center intersects the equatorial plane (positive if the point is north of Equator, negative if it is south). One of the two common geographic coordinates of a point on the Earth.

Latitude (geodetic) The angle made by a perpendicular to a given point on the surface of a sphere or ellipsoid representing the Earth and the plane of the Equator (positive if the point is north of Equator, negative if it is south). Also called *geographic latitude*. One of the two common geographic coordinates of a point on the Earth.

Latitude of opposite sign *See* Parallel of opposite sign.

Legs Line segments connecting waypoints.

Legend *See* Map legend.

Limiting forms The form taken by a system of projection when the parameters of the formulas defining that projection are allowed to reach limits that cause it to be identical with another separately defined projection.

Logical data grid A binary data grid consisting entirely of 1s and 0s. An example of a logical data grid can be created with the `topo` map by performing a logical test for positive elevations (`topo>0`). Each entry in the data grid contains a 1 if it is above sea level, or a 0 if it is at or below sea level.

Longitude The angle made by the plane of a meridian passing through a given point on the Earth's surface and the plane of the (prime) meridian passing through Greenwich, England, east or west to 180 (positive if the point is east, negative if it is west). One of the two common geographic coordinates of a point on the Earth. Paired with *Latitude*.

Loxodrome *See* Rhumb line.

Map A diagrammatic or pictorial representation of a planet's surface or part of it, showing the geographical distributions, positions, etc., of natural or artificial features such as roads, towns, relief, land cover, rainfall, populations, etc. Maps represent geospatial data visually.

Map axes A handle graphics axes object for which the `UserData` property is set to a scalar structure defining a projection type, projection parameters, and setting related properties such as map latitude and longitude limits. Many display functions in the Mapping Toolbox require that a map axes first be defined. Others create a map axes if necessary (e.g., `worldmap` and `usamap`) or assume that your data are in a projected map coordinate system (`mapshow` and `mapview`).

Map coordinates Orthogonal planar 2D coordinate tuples that specify point locations for projected geodata. The analogous term for unprojected geodata is geographic coordinates. Also called grid coordinates and plane coordinates.

Map frame In the Mapping Toolbox, a projected rectangle or quadrangle enclosing a geographic data displayed on map axes.

Map grid A symbolized network of lines, or graticule, representing parallels and meridians or plane coordinates. Plane coordinate grids are almost always rectangular with uniofrm spacing. Azimuthal map grids are organized as polar coordinates. *See* Graticule.

Map layer A vector or raster geographic dataset read into the Map Viewer, for example roads, rivers, municipal boundaries, topographic grids or orthophoto images. Map layers are "stacked" from top to bottom, and may be reordered and hidden by the user.

Map legend A key to symbolism used on a map, usually containing swatches of symbols with descriptions, and may include notes on projection, provenance, scale, units of distance, etc.

Matrix map *See* Data grid

Meridian A reference line on the Earth's surface formed by the intersection of the surface with a plane passing through both poles and some third point on the surface. This line is identified by its longitude. When the Earth is regarded as a sphere, this line is half a great circle; on the Earth regarded as an ellipsoid, it is half an ellipse.

Minimum-error projection A projection having the least possible total error of any projection in the designated classification, according to a given mathematical criterion. Usually, this criterion calls for the minimum sum of squares of deviations of linear scale from true scale throughout the map ("least squares").

National grid (U.K.) A metric grid based on the Tranverse Mercator Projection developed by Ordnance Survey in 1936 for use in Great Britain. Sometimes abbreviated "OSGB36," it is the de facto standard projection for display of UK based mapping.

National grid (U.S.) A metric grid based on the Tranverse Mercator Projection, adopted by the Federal Geographic Data Committee (FGDC) in 2001 for use in the United States. It is an evolving standard intended to unify georeferencing across the U.S., but is not yet as widely used as other countries' national grids.

Nominal scale The stated scale at which a map projection is constructed. Scale is never completely constant across the extent of a map, although in some maps (especially at large scales) it may vary by miniscule amounts.

Normal aspect A form of a projection that provides the simplest graticule and calculations. It is the polar aspect for azimuthal projections, the aspect having a straight Equator for cylindrical and pseudocylindrical projections, and the aspect showing straight meridians for conic projections. Also called conventional, direct, or regular aspect.

Northing The distance of a point northward from the origin, in the units of the coordinate system for the defined projection. Paired with *Eastings*.

Numerical geodata Geospatial data in which raster pixel values (or vector data attributes) are cardinal, ratio, or ordinal numeric measurements or computed values. For example, the `topo` datset contains numerical geodata. Each value in its data grid is an average elevation in meters for the geographic area covered by that cell. *See* Categorical geodata.

Oblique aspect An aspect of a projection on which the axis of the Earth is rotated so it is neither aligned with nor perpendicular to the conceptual axis of the map projection.

Orthoapsidal projection A projection on which the surface of the Earth taken as a sphere is transformed onto a solid other than the sphere and then projected orthographically and obliquely onto a plane for the map.

Orthographic projection A specific azimuthal projection or a type of projection in which the Earth is projected geometrically onto a surface by means of parallel projection lines.

Orthometric Height Elevation above a datum defined by a geoid representing mean sea level.

Orthomorphic projection *See* Conformal projection.

Parallel A small circle on the surface of the Earth formed by the intersection of the surface of the reference sphere or ellipsoid with a plane parallel to the plane of the Equator. This line is identified by its latitude, which may be defined in several ways. The Equator (a great circle) is usually also treated as a parallel. *See* entries for Latitude.

Parallel of opposite sign A parallel that is equally distant from but on the opposite side of the Equator. For example, for lat 30(°N (or +30°), the parallel of opposite sign is lat 30° S (or -30°). Also called latitude of opposite sign.

Perspective projection A projection produced by projecting straight lines radiating from a selected point (or from infinity) through points on the surface of a sphere or ellipsoid and then onto a tangent or secant plane. Other perspective maps are projected onto a tangent or secant cylinder or cone by using straight lines passing through a single axis of the sphere or ellipsoid. Also called geometric projection.

Planar projection A projection resulting from the conceptual projection of the Earth onto a tangent or secant plane. Usually, a planar projection is the same as an azimuthal projection. Mathematically, the projection is often only partially geometric.

Planimetric map A map representing only the horizontal positions of features (without their elevations).

Polar aspect An aspect of a projection, especially an azimuthal one, on which the Earth is viewed from directly above a pole. This aspect is called *transverse* for cylindrical or pseudocylindrical projections.

Pole An extremity of a planet's axis of rotation. The North Pole is a singular point at 90°N for which longitude is ambiguous. The South Pole has the same characteristics and is located at 90°S.

Polyconic projection A specific projection or member of a class of projections that are constructed like conic projections but with different cones for each parallel. In the normal aspect, all the parallels of latitude are nonconcentric circular arcs, except for a straight Equator, and the centers of these circles lie along a central axis.

Projected coordinate system A coordinate system defined for a particular map projection and associated parameters, which normally is planar with well-defined coordinate origin, handedness, nominal scale, and units of distance. While map scale may vary at different coordinate locations, a linear projected coordinate system has constant units of distance.

Projection A systematic representation of a curved 3-D surface such as the Earth onto a flat 2-D plane. Each map projection has specific properties that make it useful for specific purposes. For a list of projections supported by the Mapping Toolbox, type `maps`.

Projection Parameters The values of constants as applied to a map projection for a specific map; examples are the values of the scale, the latitudes of the standard parallels, and the central meridian. The required parameters vary with the projection.

Pseudoconic projection A projection that, in the normal aspect, has concentric circular arcs for parallels and on which the meridians are equally spaced along the parallels, like those on a conic projection, but on which meridians are curved.

Pseudocylindrical projection A projection that, in the normal aspect, has straight parallel lines for parallels and on which the meridians are (usually) equally spaced along parallels, as they are on a cylindrical projection, but on which the meridians are curved.

Quadrangle A region bounded by parallels north and south, and meridians east and west.

Raster geodata A georeferenced array or grid of values corresponding to specific geographic points, usually regularly and rectangularly sampled in either geographic or map space. Values may be continuous or categorical. In the case of georeferenced multiband images, raster geodata can take the form of 3- and higher-dimensional arrays.

Reckoning The determination of geographic position by calculation.

Referencing matrix A 3-by-2 matrix defining the the scaling, orientation, and placement of raster map data on the globe or in planar map coordinates. The matrix specifies an affine transformation that ties (geolocates) 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). *See* Referencing vector.

Referencing vector A three-component vector defining the geographic placement and unit cell size for raster map data. A referencing vector has the form:

```
refvec = [cells/angleunit north-latitude west-longitude]
```

A referencing vector specifies an affine transformation with rows and columns aligned to latitude and longitude, respectively, and the same data spacing in both latitude and longitude. As such, it is more specific than a referencing matrix. Note that a referencing vector may always be transformed to a referencing matrix, but only certain referencing matrices may be transformed to referencing vectors. *See* Referencing matrix.

Regional map A small-scale map of an area covering at least 5 or 10 degrees of latitude and longitude but less than a hemisphere.

Regular aspect *See* Normal aspect.

Regular data grid A data grid with equally spaced grid points in either latitude-longitude or map coordinates, defined with a referencing matrix or vector, and limited to a rectangular shape and cardinal orientation. *See* Data grid, Geolocated data grid, Referencing matrix.

Retroazimuthal projection A projection on which the direction or azimuth from every point on the map to a given central point is shown correctly with respect to a vertical line parallel to the central meridian. The reverse of an azimuthal projection.

Rhumb line A complex curve (a spherical helix) on a planet's surface that crosses every meridian at the same oblique angle; a navigator can proceed between any two points along a rhumb line by maintaining a constant heading. A rhumb line is a straight line on the Mercator projection. Also called a loxodrome.

Scale The ratio of the distance on a map or globe to the corresponding distance on the Earth; usually stated in the form 1:5,000,000 for example.A given region will appear smaller on a small scale map than on a large scale map.

.

Scale factor The ratio of the scale at a particular location and direction on a map to the nominal scale of the map. At a standard parallel, or other standard line, the scale factor is 1.0.

Secant cone, cylinder, or plane A secant cone or cylinder intersects the sphere or ellipsoid along two separate lines; these lines are parallels of latitude if the axes of the geometric figures coincide. A secant plane intersects the sphere or ellipsoid along a line that is a parallel of latitude if the plane is at right angles to the axis.

Shaded relief Shading added to a map or image that makes it appear to have three-dimensional aspects. This type of enhancement is commonly done to satellite images and thematic maps utilizing digital topographic data to provide the appearance of terrain relief.

Shapefile A widely-used file format for vector geodata designed by Environmental Systems Research Institute. Shapefiles encode coordinates for points, multipoints, lines (polylines), or polygons along with tabular attributes.

Singular points Certain points on most but not all conformal projections at which conformality fails, such as the poles on the normal aspect of the Mercator projection.

Skew-oblique aspect An aspect of a projection on which the axis of the Earth is rotated, so it is neither aligned with nor perpendicular to the conceptual axis of the map projection, and tilted, so the poles are at an angle to the conceptual axis of the map projection.

Small circle A circle on the surface of a sphere formed by the intersection with a plane. Parallels of latitude are small circles on the Earth taken as a sphere. In the Mapping Toolbox, great circles, including the Equator and all meridians, are treated as special, limiting cases of small circles. The Mapping Toolbox generalizes the concept of small circle with computations for two other types of curve: the locus of points on an ellipsoid at a given distance (as measured along a geodesic) from a central point, or the locus of points on a sphere or ellipsoid at a given distance from a central point, as measured along a rhumb line.

Small-scale mapping Mapping at a scale smaller than about 1:1,000,000, although the limiting scale sometimes has been made as large as 1:250,000.

Spatial Data Transfer Standard (SDTS) A self-documenting geospatial file formating standard adopted by the U.S. government and others. SDTS can encode locations, attributes, topological relationships, data quality, and other metadata. Note that the Mapping Toolbox can read the SDTS Raster Profile, but does not currently support SDTS vector data.

Spheroid *See* Ellipsoid.

Standard parallel In the normal aspect of a projection, a parallel of latitude along which the scale is as stated for that map. There are one or two standard parallels on most cylindrical and conic map projections and one on many polar stereographic projections.

State Plane A set of commensurate coordinate systems commonly used for utility and surveying applications in the lower 48 United States. Each state contains one or more zones. Coordinates for zones elongated north-to-south use are based on Transverse Mercator projections, while zones elongated east-to-west use Lambert Conformal Conic.

Stereographic projection A specific azimuthal projection or type of projection in which the Earth is projected geometrically onto a surface from a fixed (or moving) point on the opposite face of the Earth.

Symbolization In cartography, a mapping between geospatial objects or numerical or categorical values and cartographic symbols. The choice of graphic symbols, their size,density, shape, contrast, color and pattern are principal aspects of symbolization.

Symbolspec (Symbol specification) A cell array structure that defines symbolism characteristics for points, lines and polygons with respect to attributes and their values, or as a default symbolization irrespective of attributes.

Tangent cone or cylinder A cone or cylinder that just touches the sphere or ellipsoid along a single line. This line is a parallel of latitude if the axes of the geometric figures coincide.

Thematic map A map designed to portray primarily a particular subject, such as population, railroads, or croplands.

Tissot indicatrix *See* Indicatrix.

Topographic map A map that usually represents the vertical positions or elevations of features as well as their horizontal positions.

Transformed latitudes, longitudes, or poles Graticule of meridians and parallels on a projection after the Earth has been turned with respect to the projection so that the Earth's axis no longer coincides with the conceptual axis of the projection. Used for oblique and transverse aspects of many projections.

Transverse aspect An aspect of a map projection on which the axis of the Earth is rotated so that it is at right angles to the conceptual axis of the map projection. For azimuthal projections, this aspect is usually called *equatorial* rather than transverse.

True scale *See* Correct scale.

Vector data set Data representing geospatial objects as sequences of geographic or projected coordinate points which are implicitly connected if they represent linear or areal shapes. In the Mapping Toolbox, such geodata is often represented by two vectors, one with latitudes, another with longitudes. Segments can be demarcated by inserting `NaN`'s in both vectors. Often the pair of coordinate vectors constitute field values in a geographic data structure array.

Viewshed The portion of a surface that is visible from a given point on or above it; derived from the concept of a watershed.

Waypoints Points through which a trip, track, or transit passes, usually corresponding to course or speed changes.

WGS 72 (World Geodetic System 1972) An Earth-centered datum, used as a definition of DMA (now NIMA) DEMs. The WGS 72 datum was the result of an extensive effort extending over approximately three years to collect selected satellite, surface gravity, and astrogeodetic data available throughout 1972. These data were combined using a unified WGS solution (a large-scale least squares adjustment).

WGS 84 (World Geodetic System 1984) The WGS 84 was developed as a replacement for the WGS 72 by the military mapping community as a result of new and more accurate instrumentation and a more comprehensive control network of ground stations. The newly developed satellite radar altimeter was used to deduce geoid heights from oceanic regions between 70° north and south latitude. Geoid heights were also deduced from ground-based Doppler and ground-based laser satellite-tracking data, as well as surface gravity data. The ellipsoid associated with WGS 84 is GRS 80.

World file A small text file used to georeference different raster image formats, developed to incorporate imagery into ESRI's ArcView software.

Zenithal projection *See* Azimuthal projection.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Z