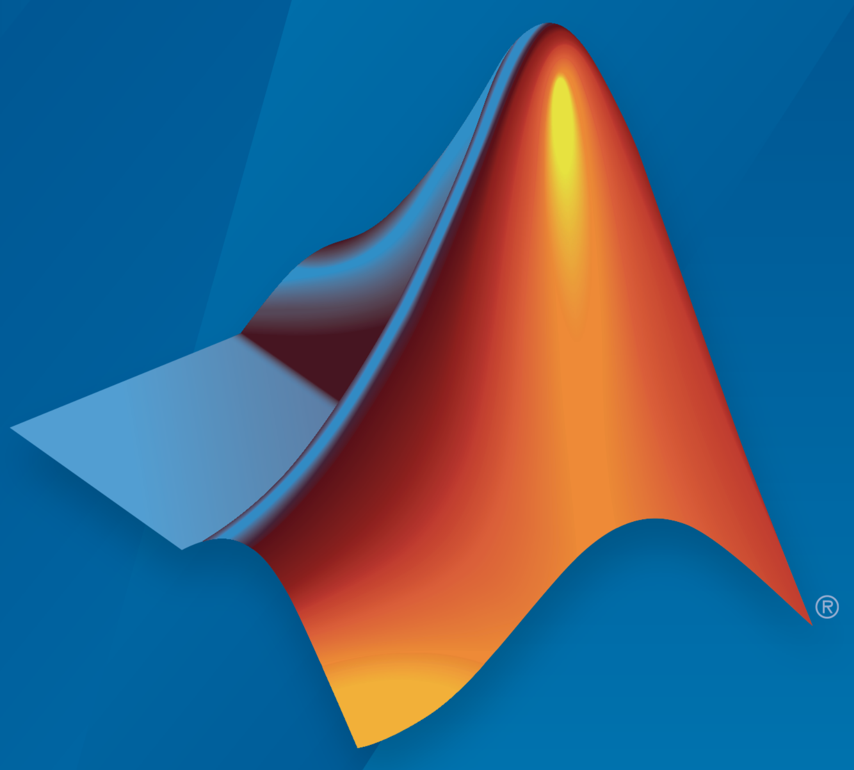
MATLAB®

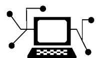
Graphics Changes in R2014b



MATLAB®

R2017a

How to Contact MathWorks



Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

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

Phone: 508-647-7000

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

MATLAB® Graphics Changes in R2014b

© COPYRIGHT 2014–2017 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

Revision History

October 2014	Online only	New for MATLAB 8.4 (Release 2014b)
March 2015	Online only	Rereleased for MATLAB 8.5 (Release 2015a)
September 2015	Online only	Rereleased for MATLAB 8.6 (Release 2015b)
March 2016	Online only	Rereleased for MATLAB 9.0 (Release 2016a)
September 2016	Online only	Rereleased for MATLAB 9.1 (Release 2016b)
March 2017	Online only	Rereleased for MATLAB 9.2 (Release 2017a)

Contents

1 Updating Graphics Code

Major Graphics Changes in R2014b **1-2**
- Graphics Handles Are Object Handles **1-2**
- New Visual Look **1-2**
- New Default Graphics Engine **1-3**
- Compatibility Considerations **1-3**

Graphics Handles Are Now Objects, Not Doubles **1-4**
- Graphics Handles Are Object Handles **1-4**
- Accessing Properties of Graphics Objects **1-5**
- Graphics Handle Arrays **1-5**
- Testing Validity of Graphics Handles **1-5**
- Referring to Figures by Integer Handles **1-6**
- Deleting Multiple Graphics Objects **1-6**
- Logical Expressions with Graphics Handles **1-6**
- Converting Cell Arrays of Graphics Handles **1-7**
- Testing Equality of Graphics Handles **1-8**
- Returning Graphics Objects From cellfun and arrayfun Functions **1-8**
- Saving Graphics Objects **1-8**
- Writing MEX-Files **1-9**

Writing Code That Works in Multiple Releases **1-10**
- Branch on Small Segment of Code **1-10**
- Branch on Large Segment of Code **1-10**

Why Are Plot Lines Different Colors? **1-11**
- New Color Order **1-11**
- The hold Command Cycles Through Colors **1-12**
- Restart Color Order **1-14**

How Do I Make the Graph Title Smaller? **1-16**

Why Do Axis Limits Keep Changing When Using axis Commands? **1-19**

Why Is Part of the Graph Cut Off? **1-23**
Graphics Objects No Longer Extend Beyond Axes Boundaries **1-23**
Disable Clipping **1-25**
Control Style of Clipping **1-27**

Why Are Colorbars and Legends Not Valid Axes Handles? . **1-29**
Use Supported Colorbar and Legend Properties **1-29**
Colorbars and Legends Cannot Be Current Axes **1-29**
Find Objects Using New Type Property Values **1-30**
Colorbars and Legends Have No Children **1-30**

How Do I Replace the EraseMode Property? **1-31**
Create Animations **1-31**
Display Changes to Object Data **1-31**
Produce Overlaid Colors **1-32**
Increase Rendering Speed **1-32**

Why Is the Children Property Empty for Some Objects? . . **1-33**

Why Do Figures Display Simultaneously? **1-35**

Why Does Accessing Tick Label Elements Return Error Message? **1-36**

How Do I Get Exponent Values for Log Axes? **1-37**

Why Are Callbacks and Application Data Not Copied? **1-38**

2 Updating GUI Code

Why Are Some Components Missing or Partially Obscured? **2-2**
Description of the Change **2-2**
Restoring Programmatic Layouts **2-2**
Restoring GUIDE Layouts **2-4**

Why Has the Behavior of ResizeFcn Changed? **2-7**
ResizeFcn Returns Error After Program Launches **2-7**
ResizeFcn Inactive for Invisible Components **2-8**
Unexpected Behavior When Outer Bounds or Drawable Area Changes **2-9**

Why Does handle.listener Return an Error? **2-10**

Updating Graphics Code

- “Major Graphics Changes in R2014b” on page 1-2
- “Graphics Handles Are Now Objects, Not Doubles” on page 1-4
- “Writing Code That Works in Multiple Releases” on page 1-10
- “Why Are Plot Lines Different Colors?” on page 1-11
- “How Do I Make the Graph Title Smaller?” on page 1-16
- “Why Do Axis Limits Keep Changing When Using axis Commands?” on page 1-19
- “Why Is Part of the Graph Cut Off?” on page 1-23
- “Why Are Colorbars and Legends Not Valid Axes Handles?” on page 1-29
- “How Do I Replace the EraseMode Property?” on page 1-31
- “Why Is the Children Property Empty for Some Objects?” on page 1-33
- “Why Do Figures Display Simultaneously?” on page 1-35
- “Why Does Accessing Tick Label Elements Return Error Message?” on page 1-36
- “How Do I Get Exponent Values for Log Axes?” on page 1-37
- “Why Are Callbacks and Application Data Not Copied?” on page 1-38

Major Graphics Changes in R2014b

Starting in R2014b, the MATLAB graphics system is built on an improved infrastructure with a new visual look, a new graphics engine, and many enhancements and added options for customizing charts. Some of the graphics changes introduced are described here.

In this section...
“Graphics Handles Are Object Handles” on page 1-2
“New Visual Look” on page 1-2
“New Default Graphics Engine” on page 1-3
“Compatibility Considerations” on page 1-3

Graphics Handles Are Object Handles

Graphics objects now use object handles of various types instead of the numeric handles used in previous releases. Graphics objects behave like other MATLAB objects and support dot notation for getting and setting properties. See “Graphics Handles Are Now Objects, Not Doubles” on page 1-4 for more information.

New Visual Look

The new visual look of MATLAB graphics has improved clarity and aesthetics with:

- A new default colormap called `parula`. The new colormap is ordered from dark to light and is perceptually uniform. Smooth changes in data appear as smooth changes in color, while sharp changes in data appear as sharp changes in color. The new colormap presents data more accurately making the data easier to interpret.
- New colors when plotting lines. The colors have equal saturation making it easier to differentiate between multiple lines.
- A lighter figure background color and lighter grid lines to emphasize plotted data.
- Anti-aliased fonts and lines for smoother text and graphics. For more information, see the GraphicsSmoothing property of figures and the FontSmoothing property of axes and text objects.
- New axes properties for setting the grid line colors and for controlling the title and axis label font sizes. See the GridColor, TitleFontSizeMultiplier, and LabelFontSizeMultiplier properties for more information.

Other enhancements and new customization options include:

- Rotatable axis tick labels. Use the XTickLabelRotation, YTickLabelRotation, and ZTickLabelRotation properties of the axes.
- Special characters in the axis tick labels, such as superscripts, subscripts, and Greek letters. By default, the axes interprets tick label characters using TeX markup. For more information, see the TickLabelInterpreter property of the axes.
- Different colormaps for each axes in a single figure. To change the colormap for an axes, pass the axes as an input argument to the `colormap` function.
- Automatic update for axis tick labels when using `datetime` and `duration` data types with `plot`.
- Pie charts of `categorical` data with automatic slice labels.
- 3-D graphics that no longer extend beyond the axes boundaries when the `Clipping` property is set to `'on'` (which is the default).

New Default Graphics Engine

Starting in R2014b, MATLAB uses OpenGL® as the default renderer for graphics. Improved OpenGL rendering such as support for transparency in vector output minimizes the need to switch renderers when working with charts.

Compatibility Considerations

The graphics changes introduced in R2014b support most of the functionality of previous releases, although there are some differences. For a list of troubleshooting topics related to the changes you are most likely to encounter, see “Graphics Changes in R2014b”. For a list of removed properties and function syntaxes, see “Save and print functionality being removed or changed” and “Properties and syntaxes being removed or changed” in the Graphics Release Notes.

Some graphics features might not work or might be unreliable because of outdated graphics drivers. Upgrade to the latest graphics drivers provided by your graphics hardware manufacturer. For more information, see “System Requirements for Graphics”.

More About

- “Why Are Plot Lines Different Colors?” on page 1-11

Graphics Handles Are Now Objects, Not Doubles

In this section...

"Graphics Handles Are Object Handles" on page 1-4
"Accessing Properties of Graphics Objects" on page 1-5
"Graphics Handle Arrays" on page 1-5
"Testing Validity of Graphics Handles" on page 1-5
"Referring to Figures by Integer Handles" on page 1-6
"Deleting Multiple Graphics Objects" on page 1-6
"Logical Expressions with Graphics Handles" on page 1-6
"Converting Cell Arrays of Graphics Handles" on page 1-7
"Testing Equality of Graphics Handles" on page 1-8
"Returning Graphics Objects From cellfun and arrayfun Functions" on page 1-8
"Saving Graphics Objects" on page 1-8
"Writing MEX-Files" on page 1-9

Graphics Handles Are Object Handles

In previous releases, graphics handles are numeric handles of type `double`. Starting in R2014b, graphics handles are object handles of various types, depending on the class of the graphics object. Graphics objects now behave like other MATLAB objects.

Most code written for numeric handles still works with object handles. For example, you can access graphics object properties and you can combine graphics objects into arrays, even if the objects belong to different classes. However, you should not perform operations that assume or require graphics handles to be numeric values, such as:

- Perform arithmetic operations on handles
- Use handles directly in logical statements without converting to a logical value
- Rely on the numeric values of the root object (0) or figure handles (integers) in logical statements
- Combine handles with data in numeric arrays
- Use any program logic that depends on handles being numeric
- Converting handles to character vectors or use handles in character vector operations

Accessing Properties of Graphics Objects

There are two ways to access properties of graphics objects that have object handles:

- Use dot notation to refer to a particular object and property. Property names are case sensitive when using dot notation. For example, this code sets the `Color` property of a line to `'red'`.

  ```
  h = plot(1:10);
  h.Color = 'red';
  ```

- Use the `set` and `get` functions to access properties for an array of objects. For example, this code sets the `LineWidth` property for multiple lines.

  ```
  h = plot(rand(4));
  set(h,'LineWidth',2);
  ```

Graphics Handle Arrays

Starting in R2014b, preallocate arrays of graphics handles using the `gobjects` function instead of the `zeros` or `ones` function. Preallocating with `zeros` or `ones` still runs without error, but can be slow.

The syntax for `gobjects` is the same as the syntax for `ones` and `zeros`.

```
h = gobjects(3,1); % preallocate
h(1) = figure;
h(2) = plot(1:10);
h(3) = gca;
```

You can combine graphics handles into arrays even if the handles are different classes. MATLAB casts the array to a common base class.

```
class(h)

ans =

matlab.graphics.Graphics
```

Testing Validity of Graphics Handles

Starting in R2014b, test the validity of graphics handles using the `isgraphics` function instead of `ishghandle`.

```
x = 1:10;
y = sin(x);

p = plot(x,y);
ax = gca;
isgraphics([p,ax])

ans =

     1     1
```

Referring to Figures by Integer Handles

Starting in R2014b, you can refer to a figure by either its object handle or its integer handle. The integer handle is the value in the new Number property of the figure.

```
h = figure; % object handle
fignum = h.Number; % integer handle
```

The integer handle, `fignum`, is a valid figure handle.

```
isgraphics(fignum) % test handle validity

ans =

     1
```

Deleting Multiple Graphics Objects

Starting in R2014b, the `delete` function accepts only one input argument. To delete multiple graphics objects, pass a single handle array to the function, instead of using multiple arguments.

```
h1 = annotation('line');
h2 = annotation('ellipse');
h3 = annotation('rectangle');
delete([h1,h2,h3])
```

Logical Expressions with Graphics Handles

Starting in R2014b, you cannot use graphics handles in logical expressions or rely on MATLAB to return a nonzero value or an empty double `[]`. Use functions such as `isempty`, `isgraphics`, and `isequal` instead.

- To determine if there are existing figures, use `isempty`. The new `groot` command references the root object.

```
if ~isempty(get(groot,'CurrentFigure'))
    disp('There are existing figures.')
else
    disp('There are no existing figures.')
end
```

- To determine if there are graphics objects with a certain tag, use `isempty`.

```
if ~isempty(findobj('Tag','myFigures'))
    disp('There are objects with this tag.')
else
    disp('There are no objects with this tag.')
end
```

- To determine if a handle is a valid figure handle, use `isgraphics` and the object `Type`.

```
if isgraphics(h,'figure')
    disp('h is a valid figure handle.')
else
    disp('h is not a valid figure handle.')
end
```

- To determine if a handle is the root handle, use the new `groot` command.

```
if isequal(h,groot)
    disp('h is the root handle')
else
    disp('h is not the root handle')
end
```

Converting Cell Arrays of Graphics Handles

Starting in R2014b, you cannot use `cell2mat` on a cell array of graphics handles to create a numeric array. Create an object array from the cell array instead.

```
p = plot(magic(3));
par = get(p,'Parent');
objarray = [par{:}]';
whos objarray

  Name          Size              Bytes  Class    Attributes
```

```
  objarray      3x1              128    matlab.graphics.axis.Axes
```

Testing Equality of Graphics Handles

Starting in R2014b, test the equality of graphics handles using `==` or `isequal`.

- To determine if handles reference the same object, therefore, are the same handle, use `==`.

  ```
  p1 = plot(1:10);
  p2 = p1;
  p2 == p1

  ans =

       1
  ```
- To determine if handles refer to objects of the same class with the same property values, but are not necessarily the same object, use `isequal`.

  ```
  l1 = line;
  l2 = line;
  isequal(l1,l2)

  ans =

       1
  ```

Returning Graphics Objects From cellfun and arrayfun Functions

To use the `cellfun` and `arrayfun` functions to return graphics objects, set `UniformOutput` to `false`.

For example:

```
t = num2str(rand);
fh = @(t) text(1,1,t);
th = cellfun(fh,{t},'UniformOutput',false);
```

Saving Graphics Objects

Starting in R2014b, if you save a graphics object in a MAT-file using the `save` function, then the MAT-file contains all the information required to regenerate the object. In

previous releases, the `save` function stores the object as a double and you cannot regenerate the object when you load the MAT-file.

Avoid saving figures with the `save` function. Using `save` to save a figure in R2014b or later makes the MAT-file inaccessible in earlier versions of MATLAB. If you use `save` to save a figure, then the function displays warning message. Save figures using the `savefig` function instead.

Writing MEX-Files

If you write MEX-files or build engine applications, then the `mexGet` and `mexSet` functions do not work on graphic object handles. Use the `mxGetProperty` and `mxSetProperty` functions in the C/C++ or Fortran Matrix Library instead.

Writing Code That Works in Multiple Releases

Most graphics code written in previous releases works with the graphics changes introduced in R2014b. However, there are some cases where code runs in one release and not in the other. If possible, implement an alternative that works across releases. If an alternative does not exist, then you can branch your code to execute different code paths.

In this section...
"Branch on Small Segment of Code" on page 1-10
"Branch on Large Segment of Code" on page 1-10

Branch on Small Segment of Code

To leverage a small feature, such as a property, branch your code based on the existence of a specific feature. For example, `SortMethod` is an axes property introduced in R2014b. This code checks if the property exists before setting its value.

```
ax = gca;
if isprop(ax,'SortMethod')
    set(ax,'SortMethod','childorder')
end
```

Branch on Large Segment of Code

To branch large segments of code when there is no specific feature to test, use the `verLessThan('matlab','8.4.0')` command. This command returns `0` if you are running R2014b or later and returns `1` if you are running earlier releases. For example, use this coding pattern to branch your code.

```
if verLessThan('matlab','8.4.0')
    % execute code for R2014a or earlier
else
    % execute code for R2014b or later
end
```

See Also

`class` | `iscell` | `ischar` | `ishghandle` | `ismethod` | `isprop` | `verLessThan` | `whos`

Why Are Plot Lines Different Colors?

In this section...
"New Color Order" on page 1-11
"The hold Command Cycles Through Colors" on page 1-12
"Restart Color Order" on page 1-14

New Color Order

Starting in R2014b, MATLAB graphics has a new color order that determines the colors used in plots. This table shows the color order introduced in R2014b versus previous releases. It also lists the RGB triplet values that define the colors.

Starting in R2014b			R2014a and Earlier		
0	0.4470	0.7410	0	0	1.0000
0.8500	0.3250	0.0980	0	0.5000	0
0.9290	0.6940	0.1250	1.0000	0	0
0.4940	0.1840	0.5560	0	0.7500	0.7500
0.4660	0.6740	0.1880	0.7500	0	0.7500
0.3010	0.7450	0.9330	0.7500	0.7500	0
0.6350	0.0780	0.1840	0.2500	0.2500	0.2500

The `ColorOrder` property of the axes contains the color order. To change the color order, set a different default value for the `ColorOrder` property. For example, this code sets the default color order to the colors used in previous releases.

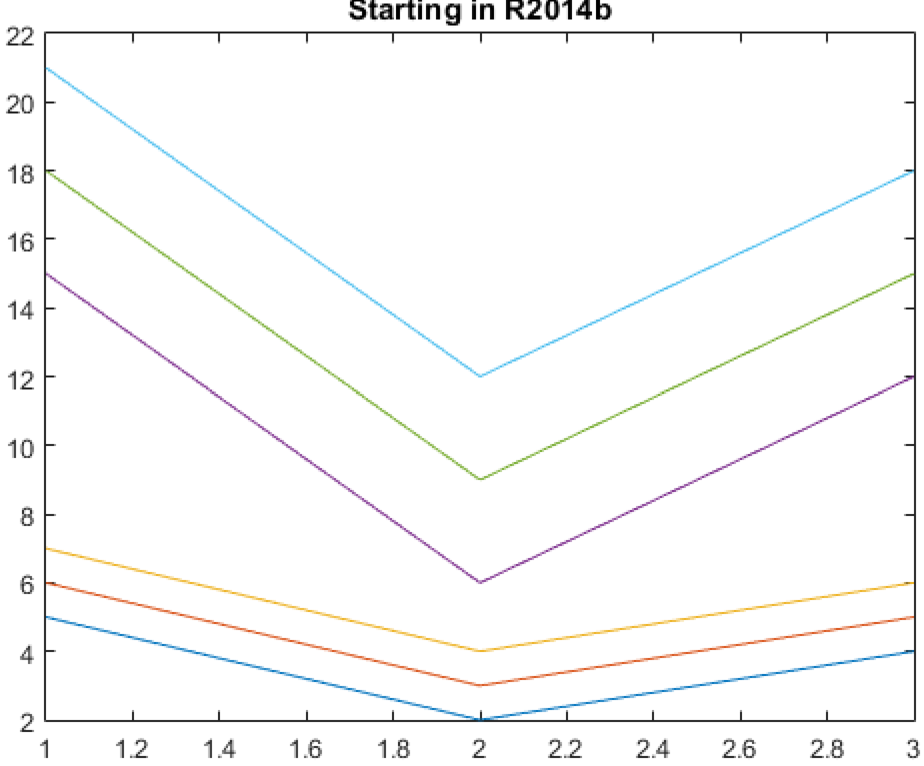
```
co = [0 0 1;
      0 0.5 0;
      1 0 0;
      0 0.75 0.75;
      0.75 0 0.75;
      0.75 0.75 0;
      0.25 0.25 0.25];
set(groot,'defaultAxesColorOrder',co)
```

The hold Command Cycles Through Colors

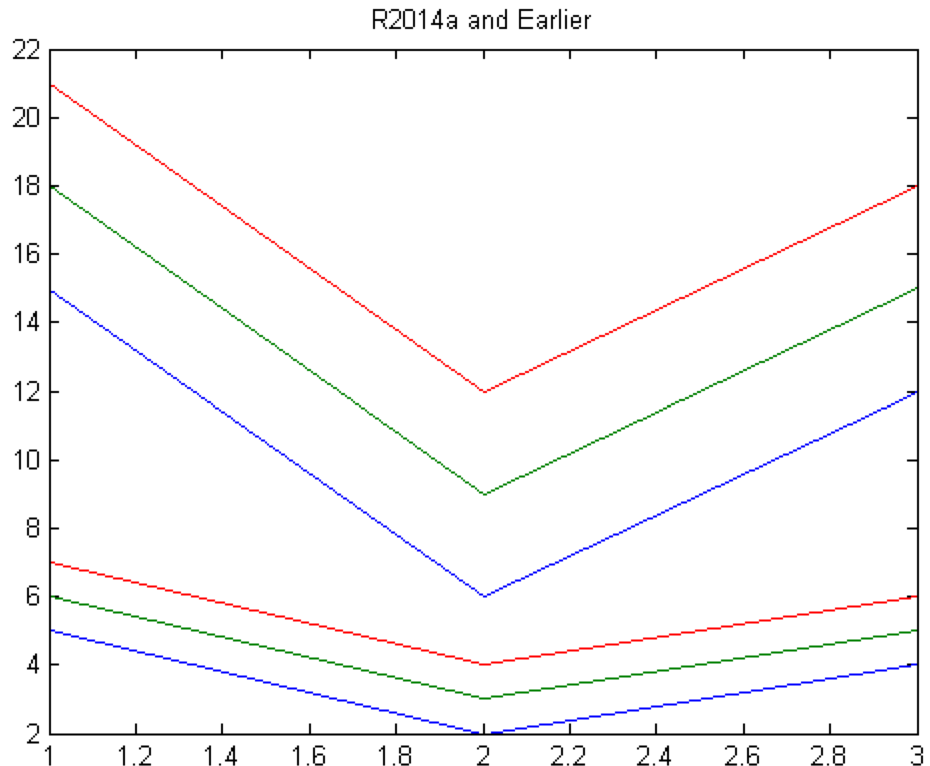
In R2014a and earlier, the `hold on` command does not retain the current color so new plots added to the axes start from the beginning of the color order. Visually, this means that new plots use the same initial color. Starting in R2014b, the `hold on` command retains the current color so that new plots added to the axes use the next colors in the color order.

For example, this code displays six lines using the `hold on` command. Starting in R2014b, the lines cycle through the color order and the resulting plot uses the first six colors of the color order.

```
data = [5 6 7; 2 3 4; 4 5 6];
plot(data);
hold on
plot(3*data);
hold off
```



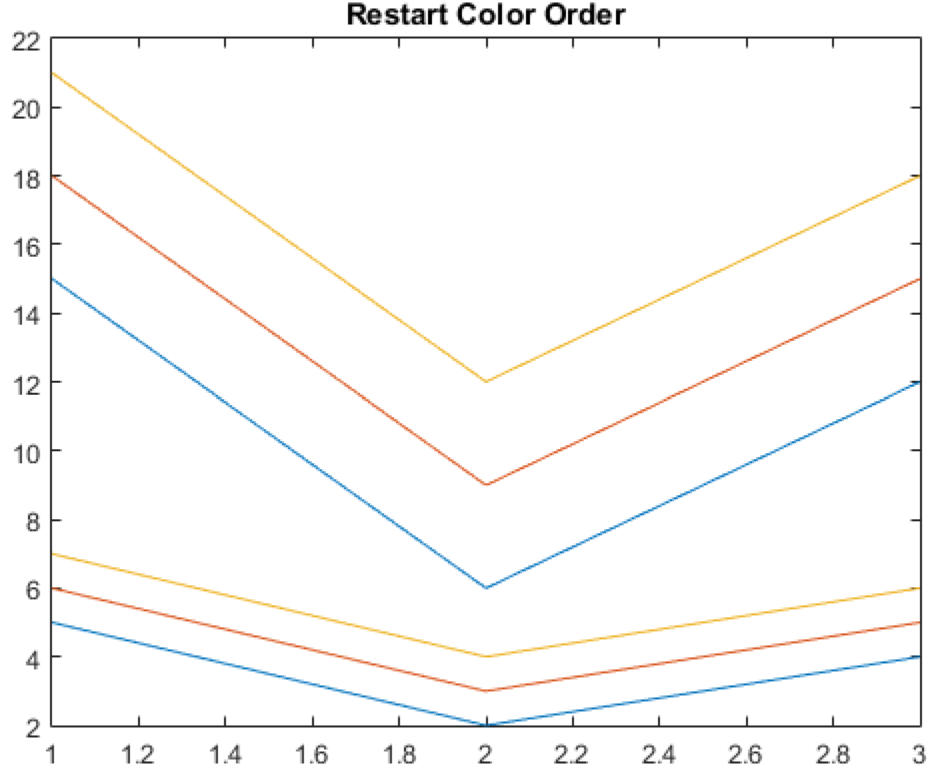
If you run the same code in previous releases, the color order restarts with each plotting command. The resulting plot uses the first three colors of the color order twice.



Restart Color Order

Starting in R2014b, if you want to restart the color order before each plotting command, then set the `ColorOrderIndex` property of the axes to 1.

```
data = [5 6 7; 2 3 4; 4 5 6];
plot(data);
hold on
ax = gca;
ax.ColorOrderIndex = 1;
plot(3*data);
hold off
```



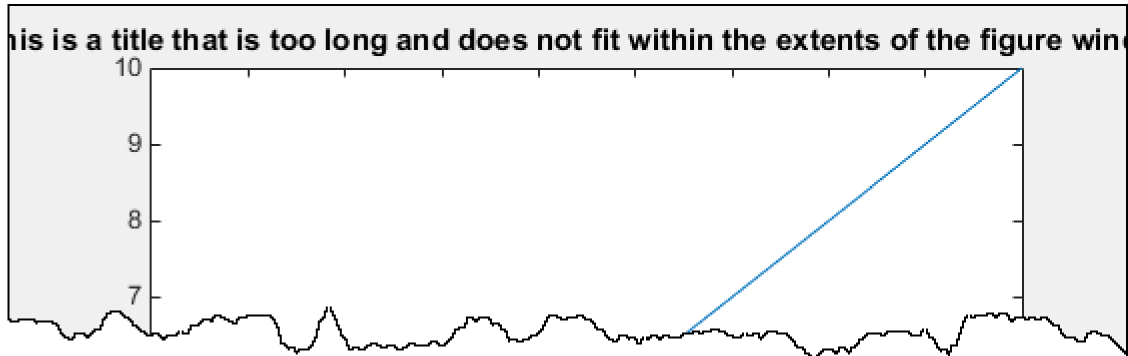
See Also

`hold`

How Do I Make the Graph Title Smaller?

Starting in R2014b, MATLAB graphics titles use a bold and slightly larger font for better visibility. As a result, some text might not fit within the extents of the figure window. For example, this code creates a graph that has a long title that does not fit within the extents of the figure window.

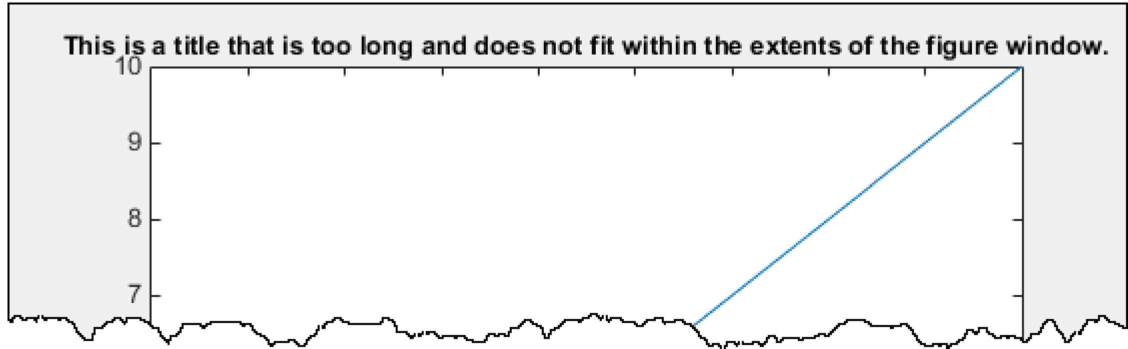
```
plot(1:10);
title(['This is a title that is too long and does not fit',...
    'within the extents of the figure window.'])
```



The title font size is based on the `TitleFontSizeMultiplier` and `FontSize` properties of the axes. By default the `FontSize` property is 10 points and the `TitleFontSizeMultiplier` is `1.100`, which means that the title font size is 11 points.

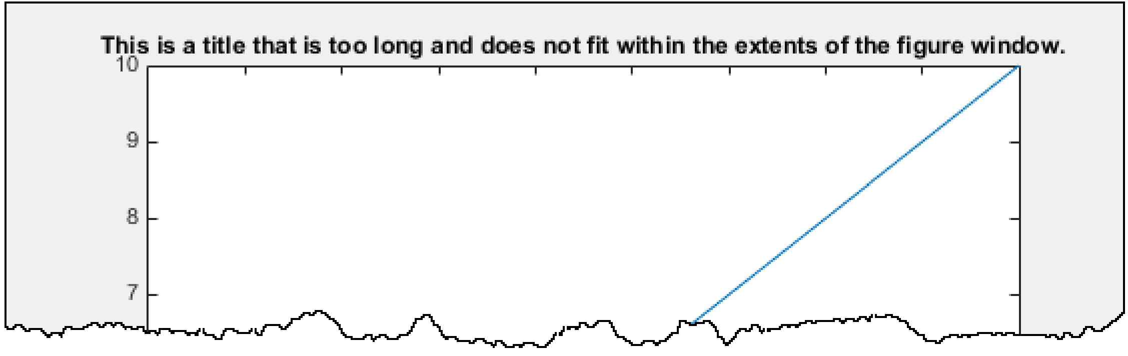
To change the title font size without affecting the rest of the font in the axes, set the `TitleFontSizeMultiplier` property of the axes.

```
plot(1:10);
title(['This is a title that is too long and does not fit',...
    'within the extents of the figure window.'])
ax = gca;
ax.TitleFontSizeMultiplier = 1;
```



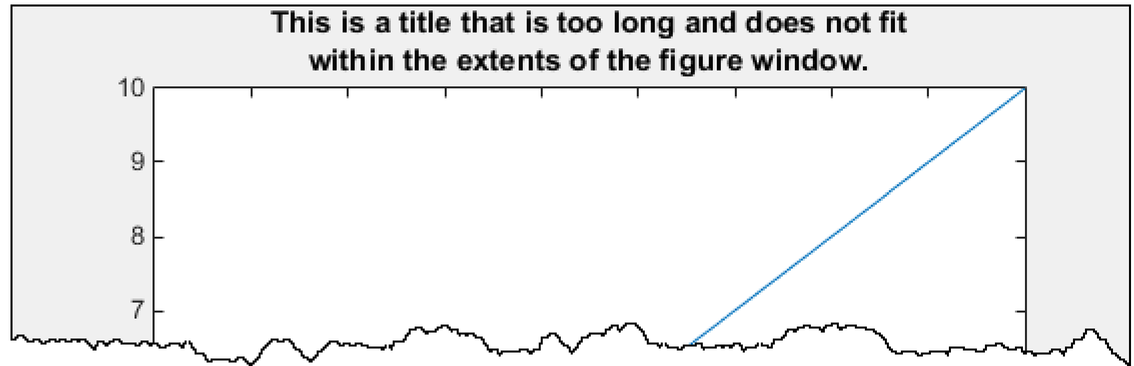
To make the font size smaller for the entire axes, set the `FontSize` property. Changing this property affects the font for the title, tick labels and axis labels, if they exist.

```
plot(1:10);
title(['This is a title that is too long and does not fit',...
    'within the extents of the figure window.'])
ax = gca;
ax.FontSize = 8;
```



To keep the same font size and display the title across two lines, use a cell array with curly brackets `{}` to define a multiline title.

```
plot(1:10);
title({'This is a title that is too long and does not fit',...
    'within the extents of the figure window.'})
```



See Also

Functions
`title`

Properties
Axes Properties

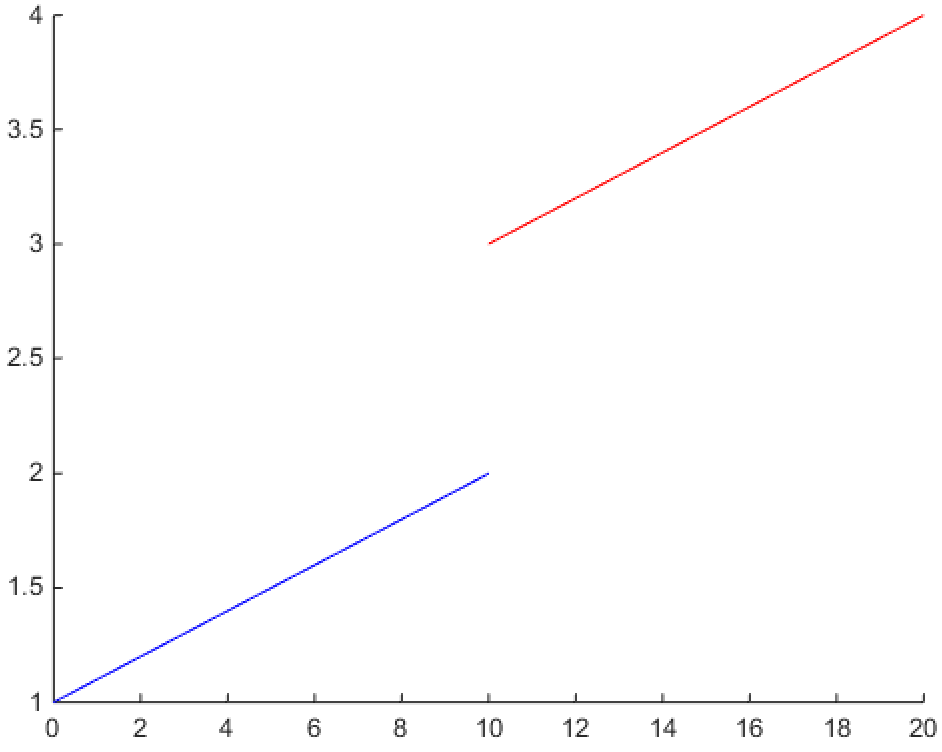
Why Do Axis Limits Keep Changing When Using axis Commands?

In R2014a and earlier, if you set the axis limits using an `axis tight` or an `axis image` command, then the calculated axis limits do not change. If you add new data to the axes outside of the current axis limits, then the limits do not automatically update to encompass the data.

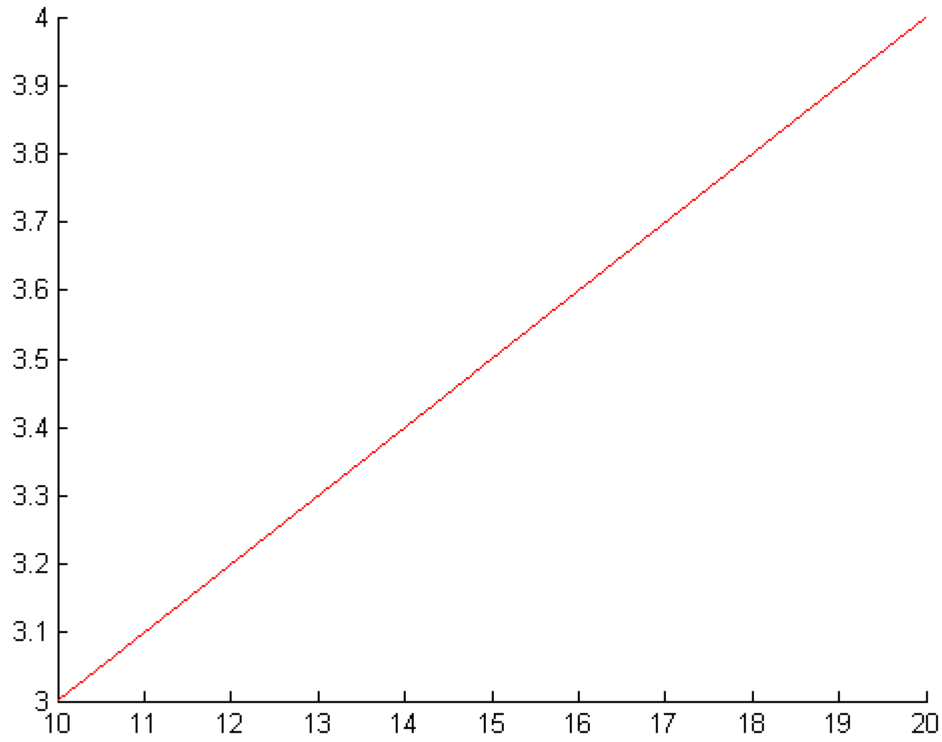
Starting in R2014b, if you use these commands and afterwards add new data to the axes, then the axis limits automatically update to encompass the new data.

For example, this code sets the axis limits using the `axis tight` command, and then adds new data to the graph. Starting in R2014b, the limits update to encompass both lines.

```
line([10 20],[3 4],'Color','red')
axis tight
line([0 10],[1 2],'Color','blue')
```



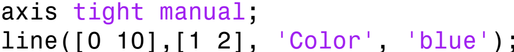
If you run the same code in previous releases, the limits do not update so the blue line is not visible in the axes.



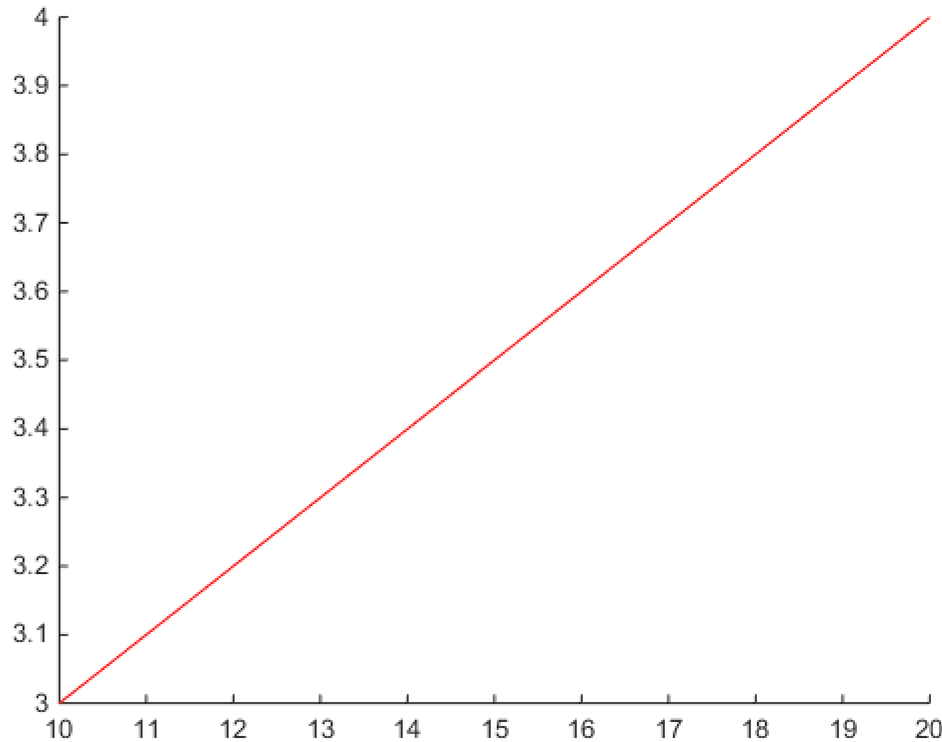
In R2014a and earlier, these `axis` commands set the axis limit modes (`XLimMode`, `YLimMode`, and `ZLimMode`) to `'manual'`. When the limit modes are manual, the limits do not update to reflect changes in the data. Starting in R2014b, these `axis` commands set the axis limit modes to `'auto'`. When the limit modes are auto, the limits automatically update to reflect changes in the data.

Starting in R2014b, to keep the axis limits from automatically updating, append `manual` to the end of the `axis` command. For example, `axis tight manual`. The `manual` option sets the limit modes to manual, as in previous releases, so that the limits do not automatically update. For example, this code does not update the limits to encompass the second blue line.

```
line([10 20],[3 4], 'Color', 'red');
```



```
axis tight manual;
line([0 10],[1 2], 'Color', 'blue');
```



See Also

Functions
`axis`

Properties
Axes Properties

Why Is Part of the Graph Cut Off?

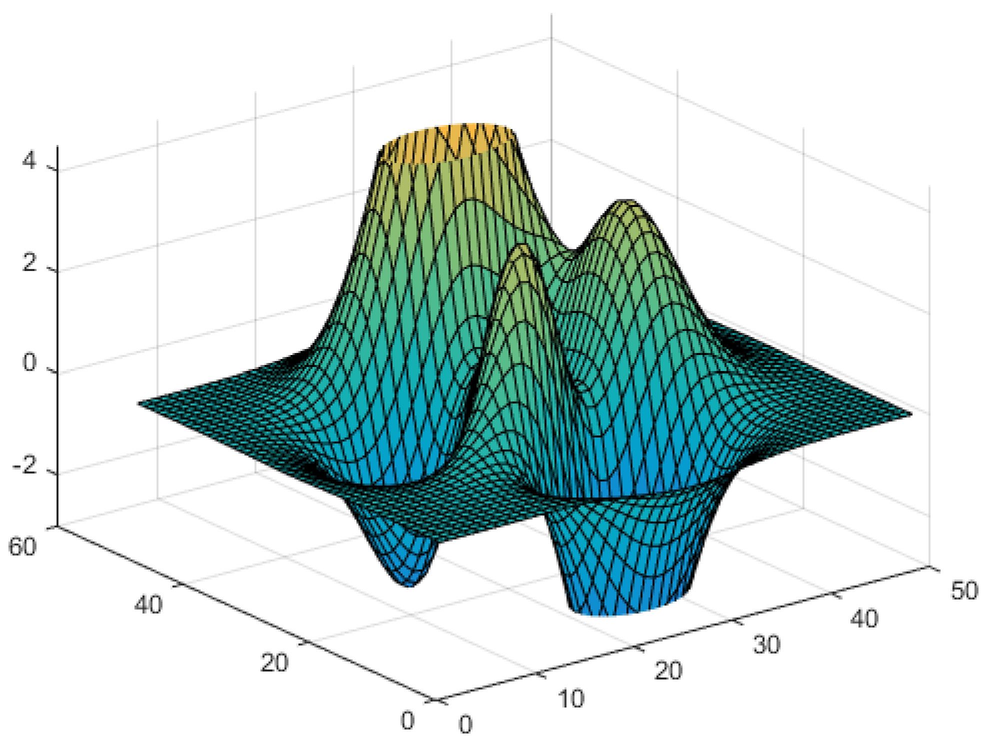
In this section...

"Graphics Objects No Longer Extend Beyond Axes Boundaries" on page 1-23
"Disable Clipping" on page 1-25
"Control Style of Clipping" on page 1-27

Graphics Objects No Longer Extend Beyond Axes Boundaries

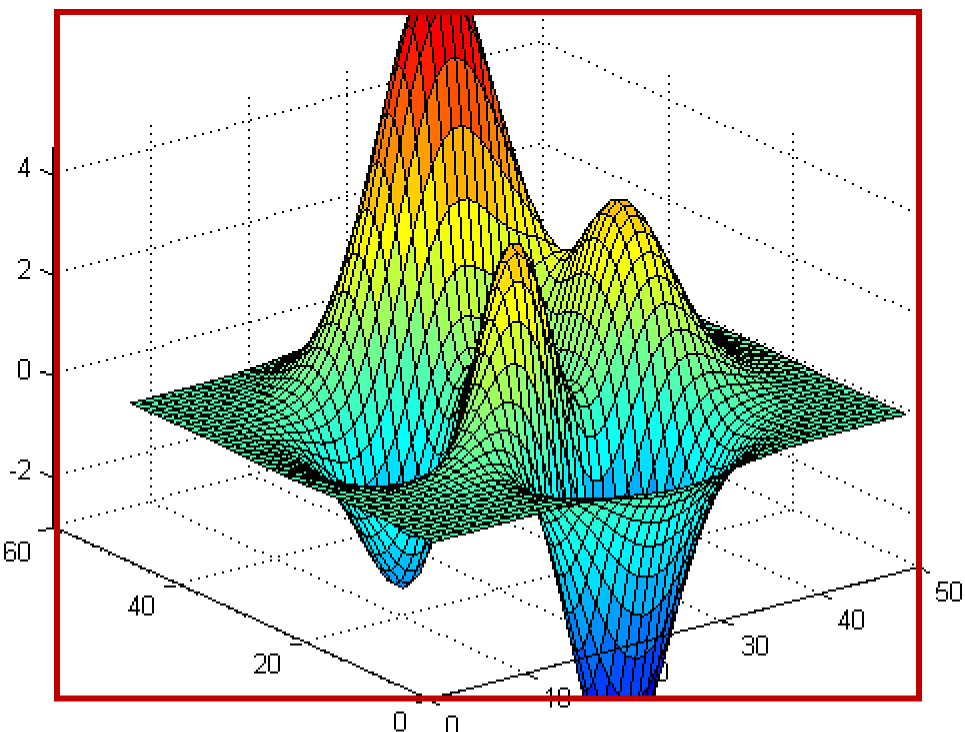
Starting in R2014b, graphics objects do not extend beyond the axes boundaries. Objects are clipped to the six sides of the axes box defined by the axes limits. For example, MATLAB does not display the peaks of this surface that extend beyond the specified *z*-limits.

```
surf(peaks);
zlim([-3,4.5]);
```



In R2014a and earlier, MATLAB uses a different technique to clip objects. Instead of clipping to the axes limits, MATLAB clips to the smallest 2-D rectangle that encloses the axes. For example, in previous releases, the same surface plot extends beyond the specified *z*-limits. The red rectangle indicates the boundaries used for clipping.

```
surf(peaks);
zlim([-3,4.5]);
```



Disable Clipping

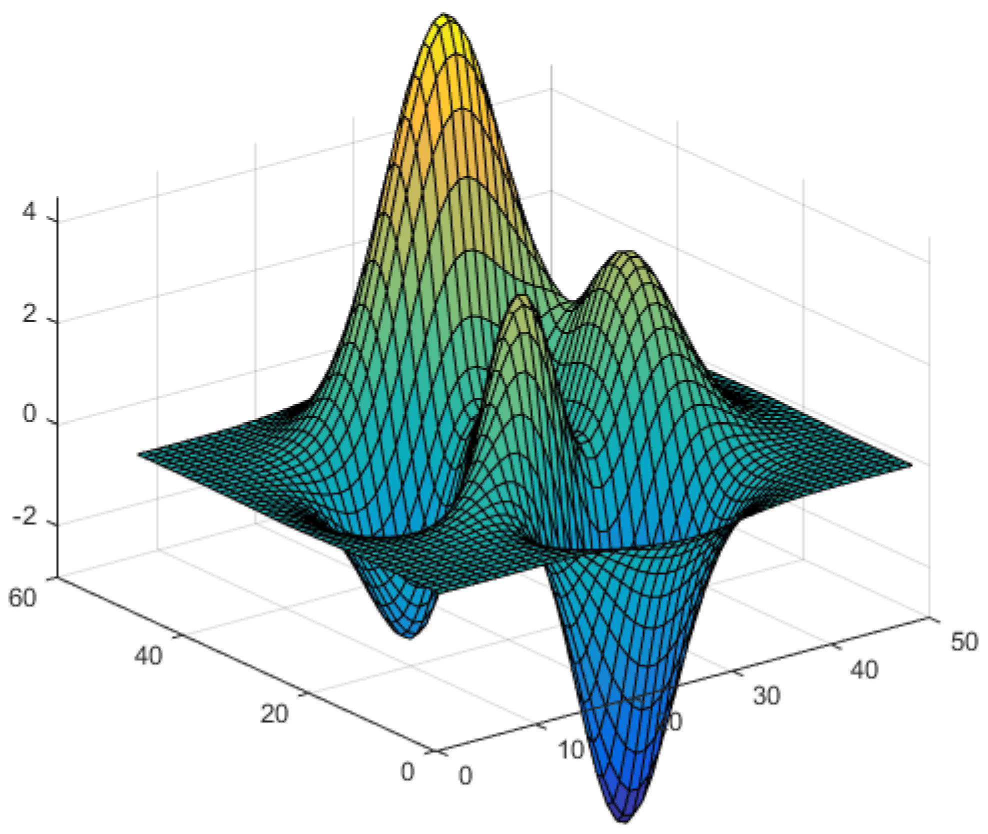
The axes and individual objects in the axes have a `Clipping` property that controls the clipping behavior. By default, this property is set to `'on'`. To disable clipping, set the `Clipping` property to `'off'`.

If the `Clipping` property for the axes is `'on'`, then each individual object in the axes controls its own clipping behavior. To disable clipping for all objects in the axes, set the `Clipping` property for the axes to `'off'`. This table lists the results for different combinations of `Clipping` property values.

Clipping Property for Axes	Clipping Property for Individual Object	Result
'on'	'on'	Individual object is clipped (default)
'on'	'off'	Individual object is not clipped
'off'	'on'	No objects in axes are clipped
'off'	'off'	No objects in axes are clipped

For example, disable the clipping for all objects in the axes by setting the `Clipping` property of the axes to `'off'`.

```
surf(peaks);
zlim([-3,4.5]);
ax = gca;
ax.Clipping = 'off';
```



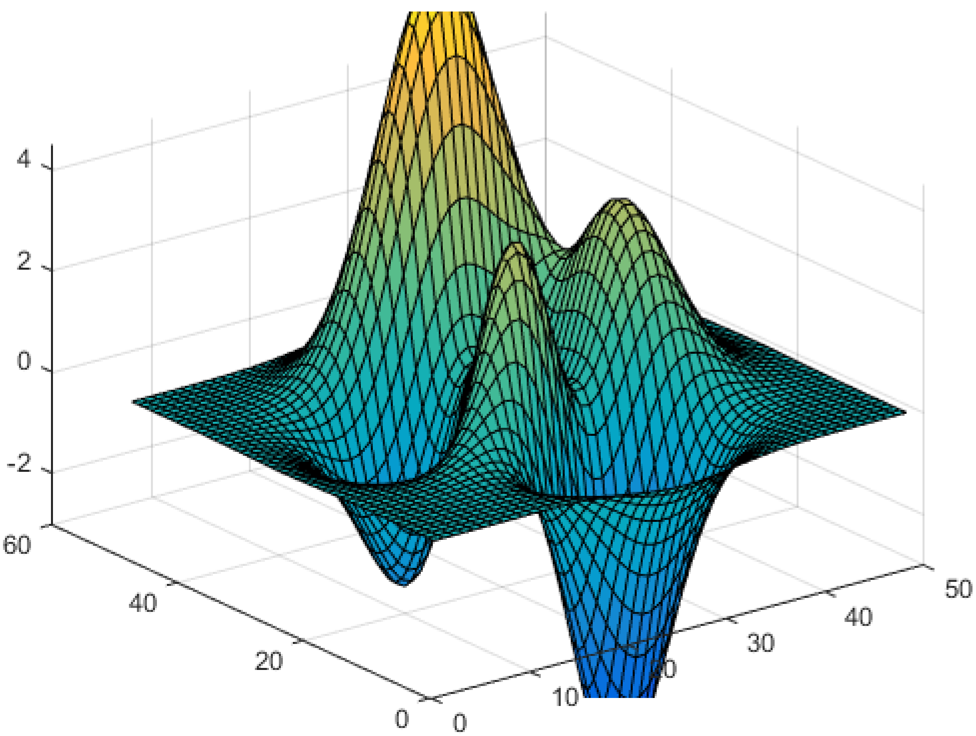
Control Style of Clipping

The new `ClippingStyle` axes property controls the technique used to clip objects. Set this property to one of these values:

- `'3dbox'` — Clips objects to the six sides of the axes box defined by the axes limits. This is the default value.
- `'rectangle'` — Clips objects to the smallest 2-D rectangle that encloses the axes in any given view.

To get the same style of clipping as in R2014a and earlier, set the `ClippingStyle` to `'rectangle'`.

```
surf(peaks)
zlim([-3,4.5]);
ax = gca;
ax.ClippingStyle = 'rectangle';
```



See Also

Axes Properties

Why Are Colorbars and Legends Not Valid Axes Handles?

Starting in R2014b, colorbars and legends are no longer axes objects. They are new types of objects that have their own sets of supported properties. In previous releases, they are axes objects that you can modify using axes properties. However, many axes properties are not relevant to colorbars and legends.

You should not perform operations that assume or require colorbars and legends to be axes objects.

Use Supported Colorbar and Legend Properties

Do not use axes properties to modify colorbars or legends. Use their supported properties. For a list, see Colorbar Properties or Legend Properties.

For example, to reverse the direction of the color scale along a colorbar use the new Direction property for the colorbar instead of setting the `XDir` or `YDir` axes property.

```
c = colorbar;
c.Direction = 'reverse';
```

Colorbars and Legends Cannot Be Current Axes

Do not pass a colorbar object or a legend object to a function that expects an axes object as an input argument.

For example, passing a legend object to the `axes` function to make it the current axes returns an error message:

```
plot(1:10)
l = legend('line plot');
axes(l)
```

```
Error using axes
Handles of type Legend cannot be made the current Axes.
```

Similarly, passing a colorbar object to the `axes` function returns an error message:

```
c = colorbar;
axes(c)
```

```
Error using axes
```

```
Handles of type ColorBar cannot be made the current Axes.
```

In previous releases, you might make a colorbar the current axes before giving it a title, Now, use the new `Label` property of the colorbar instead.

```
c = colorbar;
c.Label.String = 'Colorbar Label';
```

Find Objects Using New Type Property Values

Colorbars and legends no longer have a `Type` property of `'axes'`. Do not use `findall` or `findobj` to find objects with a `Type` property of `'axes'` and expect it to return colorbars and legends.

To find legends, search for objects with a `Type` property of `'legend'`.

```
findall(groot,'Type','legend')
```

To find colorbars, search for objects with a `Type` property of `'colorbar'`.

```
findall(groot,'Type','colorbar')
```

Colorbars and Legends Have No Children

Colorbars and legends no longer contain handles to underlying objects in their `Children` property. Their `Children` properties contain an empty graphics placeholder array. For a legend, access these underlying objects using the output arguments from the `legend` function instead.

See Also

Functions
`colorbar` | `legend`

Properties
Colorbar Properties | Legend Properties

How Do I Replace the EraseMode Property?

Starting in R2014b, the `EraseMode` property has been removed from all graphics objects. You can still achieve most of the effects produced by `EraseMode`, such as creating animations or producing overlaid colors, using the techniques described here.

In this section...
"Create Animations" on page 1-31
"Display Changes to Object Data" on page 1-31
"Produce Overlaid Colors" on page 1-32
"Increase Rendering Speed" on page 1-32

Create Animations

To accumulate a picture by adding data to each frame, use one of these approaches instead of setting the `EraseMode` property to `'none'`:

- Use `hold on` to retain the current data and add new data to the graph.
- Use the new `animatedline` function to create line animations.
- Use the `movie` function to play recorded movie frames.

For example, create a line animation using the new `animatedline` function.

```
theta = linspace(0,2*pi,1000);
h = animatedline();
axis([0,2*pi,-1,1])

for t = theta
    addpoints(h,t,sin(t));
    drawnow;
end
```

For more information on creating line animations, see the `animatedline` reference page and the `drawnow` function.

Display Changes to Object Data

To immediately display changes to object data, call the `drawnow` function instead of setting `EraseMode` to `'xor'`.

For example, change the `YData` for a line and display the updates.

```
t = linspace(0,2*pi,10000);
y = exp(sin(t));
h = plot(t,y);
for k = 1:0.01:10
   y = exp(sin(t.*k));
   h.YData = y;
   drawnow
end
```

Produce Overlaid Colors

To produce overlaid colors, use transparency instead of setting `Erasemode` to `'xor'`.

```
p1 = patch([0,2,2,0],[0,0,2,2],[1,1,1,1]);
p2 = patch([1,3,3,1],[1,1,3,3],[2,2,2,2]);
p2.FaceAlpha = 0.5;
```

Increase Rendering Speed

In previous releases, setting the `EraseMode` property to `'xor'` increases the rendering speed. Remove code that sets the `EraseMode` property to get similar rendering speeds.

See Also

Functions

`animatedline` | `drawnow` | `hold` | `movie`

Why Is the Children Property Empty for Some Objects?

In R2014a and earlier, chart objects, legends, and colorbars contain handles to underlying objects in their `Children` properties. For example, a scatter series contains patch objects in its `Children` property. Changing property values of the patch object changes the appearance of the scatter series.

Starting in R2014b, these objects do not contain handles to underlying objects in their `Children` properties. To customize the graph, use properties of the actual object. This table lists the affected objects.

Object	Children in R2014a and Earlier	Children Starting in R2014b	Alternate Options
Colorbar	image objects	`0x0 empty GraphicsPlacehold array`	Use colorbar properties to modify the colorbar.
Legend	line, patch, and text objects	`0x0 empty GraphicsPlacehold array`	Get the handles to these objects using the output arguments from the `legend` function.
Area	patch objects	`0x0 empty GraphicsPlacehold array`	Use area properties to modify the area.
Bar series	patch objects	`0x0 empty GraphicsPlacehold array`	Use bar series properties to modify the bars.
Contour	patch objects	`0x0 empty GraphicsPlacehold array`	Use contour properties to modify the contour lines.
Errorbar series	line objects	`0x0 empty GraphicsPlacehold array`	Use errorbar series properties to modify the errorbars.
Scatter series	patch objects	`0x0 empty GraphicsPlacehold array`	Use scatter series properties to modify the markers.

Object	Children in R2014a and Earlier	Children Starting in R2014b	Alternate Options
Stair	line objects	`0x0 empty GraphicsPlaceholder array`	Use stair properties to modify the stairs.
Stem series	line objects	`0x0 empty GraphicsPlaceholder array`	Use stem series properties to modify the stems and markers.
Quiver series	line objects	`0x0 empty GraphicsPlaceholder array`	Use quiver series properties to modify the arrows.

See Also

`area` | `bar` | `colorbar` | `contour` | `errorbar` | `legend` | `quiver` | `scatter` | `stairs` | `stem`

Why Do Figures Display Simultaneously?

If you create multiple figures in a script in R2014a and earlier, then MATLAB attempts to wait for each figure to display on the screen before continuing to execute the script. Starting in R2014b, MATLAB does not wait for a figure to display before continuing to execute the script. Thus, the script might run to completion before the figures are displayed. This change is most noticeable for scripts that create multiple figures and perform long computations.

To force figures to display as they are created, use `drawnow`.

```
figure
plot(1:10);
drawnow
```

See Also

`drawnow`

Why Does Accessing Tick Label Elements Return Error Message?

In R2014a and earlier, the `XTickLabel`, `YTickLabel`, and `ZTickLabel` properties of the axes contained the tick label values in either a character array or a cell array. Starting in R2014b, these properties always contain the tick labels in a cell array. Access elements of the array using cell array indexing with curly braces `{}`.

Using matrix indexing to access the elements returns an error message:

```
plot(0:10,0:10);
ax = gca;
xticks = get(ax,'XTickLabel');
xticks(1) = 'start';
```

```
Conversion to cell from char is not possible.
```

Use cell array indexing with curly braces `{}` to access tick label elements.

```
plot(0:10,0:10);
ax = gca;
xticks = get(ax,'XTickLabel');
xticks{1} = 'start';
set(ax,'XTickLabel',xticks) % set tick labels to updated values
```

See Also

Axes Properties

How Do I Get Exponent Values for Log Axes?

Starting in R2014b, the `XTickLabel`, `YTickLabel`, or `ZTickLabel` properties for a log axis contain cell arrays with the full TeX markup used for the tick labels. In R2014a and earlier, these properties contain a character array with only the exponent values for the tick marks.

<table>
<tr><th>Starting in R2014b</th><th>R2014a and Earlier</th></tr>
<tr><td><pre>semilogx(1:10000);
ax = gca;
ticks = ax.XTickLabel
class(ticks)</pre></td><td><pre>semilogx(1:10000);
ax = gca;
ticks = get(ax,'XTickLabel')
class(ticks)</pre></td></tr>
<tr><td><pre>ticks =

    '10^{0}'
    '10^{1}'
    '10^{2}'
    '10^{3}'
    '10^{4}'

ans =

cell</pre></td><td><pre>ticks =

0
1
2
3
4

ans =

char</pre></td></tr>
</table>

To extract just the exponent values from the tick label property, use the `regexprep` function.

```
expression = '\d*\^\{(\-?\d*)\}';
replace = '$1';
exponents = regexprep(ticks,expression,replace)
```

```
exponents =

    '0'
    '1'
    '2'
    '3'
    '4'
```

See Also

Axes Properties

Why Are Callbacks and Application Data Not Copied?

Starting in R2014b, `copyobj` does not copy callback properties or application data associated with graphics objects. The copied object has callbacks set to empty character arrays and application data set to empty structure arrays. Copies of objects might not behave as expected. For example, clicking a push button on the copy of a uicontrol has no effect.

If you want to create a copy of an object that has callbacks, then rerun the code used to create the first object to create a second object.

If you have existing code that uses `copyobj` to copy callbacks, then you can use `copyobj` with the `'legacy'` option, for example, `c = copyobj(h,p,'legacy')`. The behavior of the `'legacy'` option is consistent with versions of MATLAB before R2014b.

See Also

`copyobj`

Updating GUI Code

- "Why Are Some Components Missing or Partially Obscured?" on page 2-2
- "Why Has the Behavior of ResizeFcn Changed?" on page 2-7
- "Why Does handle.listener Return an Error?" on page 2-10

Why Are Some Components Missing or Partially Obscured?

In this section...
"Description of the Change" on page 2-2
"Restoring Programmatic Layouts" on page 2-2
"Restoring GUIDE Layouts" on page 2-4

Description of the Change

Axes, uicontrols or uitables might appear to be missing in the new graphics system because they are obscured by other components.

In previous releases, the order of components listed in the `Children` property matches the order in which they are created. However, this order does not necessarily match the front-to-back positioning (or the stacking order) of the components on the screen.

In previous releases, uicontrols always display on top of uipanels and uibuttongroups. Previous releases also allow an axes to display on top of a uipanel without being a child of the uipanel.

Starting in R2014b, the order of components listed in the `Children` property matches the stacking order of child components on the screen. You might need to update your code if your UI contains a uipanel or uibuttongroup:

- Relative positioning is not sufficient to display an axes, uicontrol, or uitable on top of a uipanel or uibuttongroup. To place a component on top of another, set its `Parent` property to be the component you want to appear beneath it.
- Uipanels and uibuttongroups have the same stacking order behavior on the screen as uicontrols and uitables.

The new behavior reflects changes to MATLAB that provide more consistent behavior.

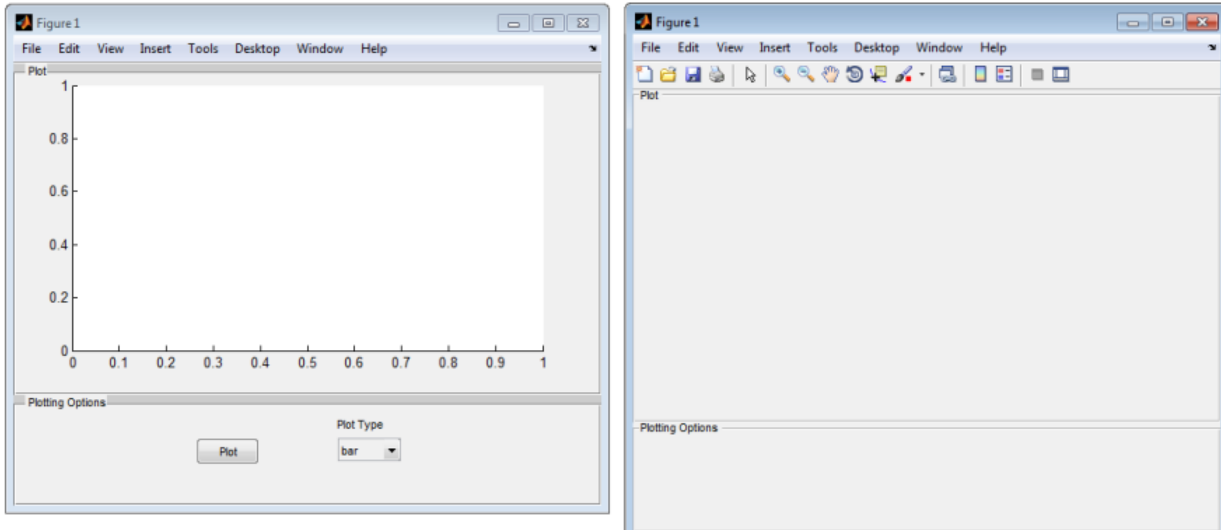
Restoring Programmatic Layouts

This code creates a figure with a top panel containing an axes and a bottom panel containing a push button and pop-up menu.

```
hf = figure;
hb = uicontrol('Style','PushButton',...
               'String','Plot',...
```

```
                'Position',[175, 40, 60, 25]);
hpulabel = uicontrol('Style','text',...
                     'String', 'Plot Type',...
                     'Position', [300, 65, 60, 20]);
hpu = uicontrol('Style', 'popupmenu',...
                'String', {'bar', 'plot', 'stem'},...
                'Position',[310, 40, 60, 25]);
topp = uipanel('Title', 'Plot',...
               'Position',[0 .25 1 .75]);
ah = axes('Position', [.10, .35 .80 .60]);
bottomp = uipanel('Title','Plotting Options',...
                  'Position',[0 0 1 .25]);
```

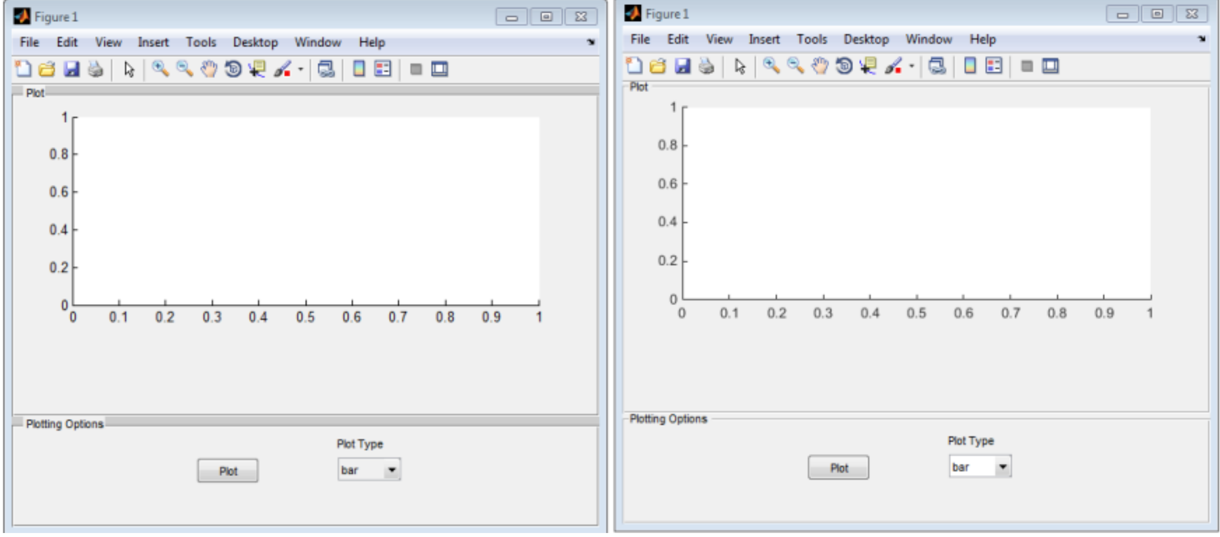
Running the code in previous releases produces the figure on the left. However, running this function in the new graphics system produces the figure on the right.



To ensure there are no components hidden behind containers, set the `Parent` property of each component to make it a child of the container. For example, the following code restores the original UI.

```
hf = figure;
topph = uipanel('Parent', hf,'Title', 'Plot',...
                'Position',[0 .25 1 .75]);
axes('Parent', topph, 'Position', [.10, .35 .80 .60]);
bottomph = uipanel('Parent', hf,'Title','Plotting Options',...
```

```
                   'Position',[0 0 1 .25])
hpulabel = uicontrol('Parent', bottomph, 'Style','text',...
                     'String', 'Plot Type',...
                     'Position', [300, 65, 60, 20]);
hb = uicontrol('Parent', bottomph, 'Style','PushButton',...
               'String','Plot',...
               'Position',[175, 40, 60, 25]);
hpu = uicontrol('Parent', bottomph, 'Style', 'popupmenu',...
                'String', {'bar', 'plot', 'stem'},...
                'Position',[310, 40, 60, 25]);
```

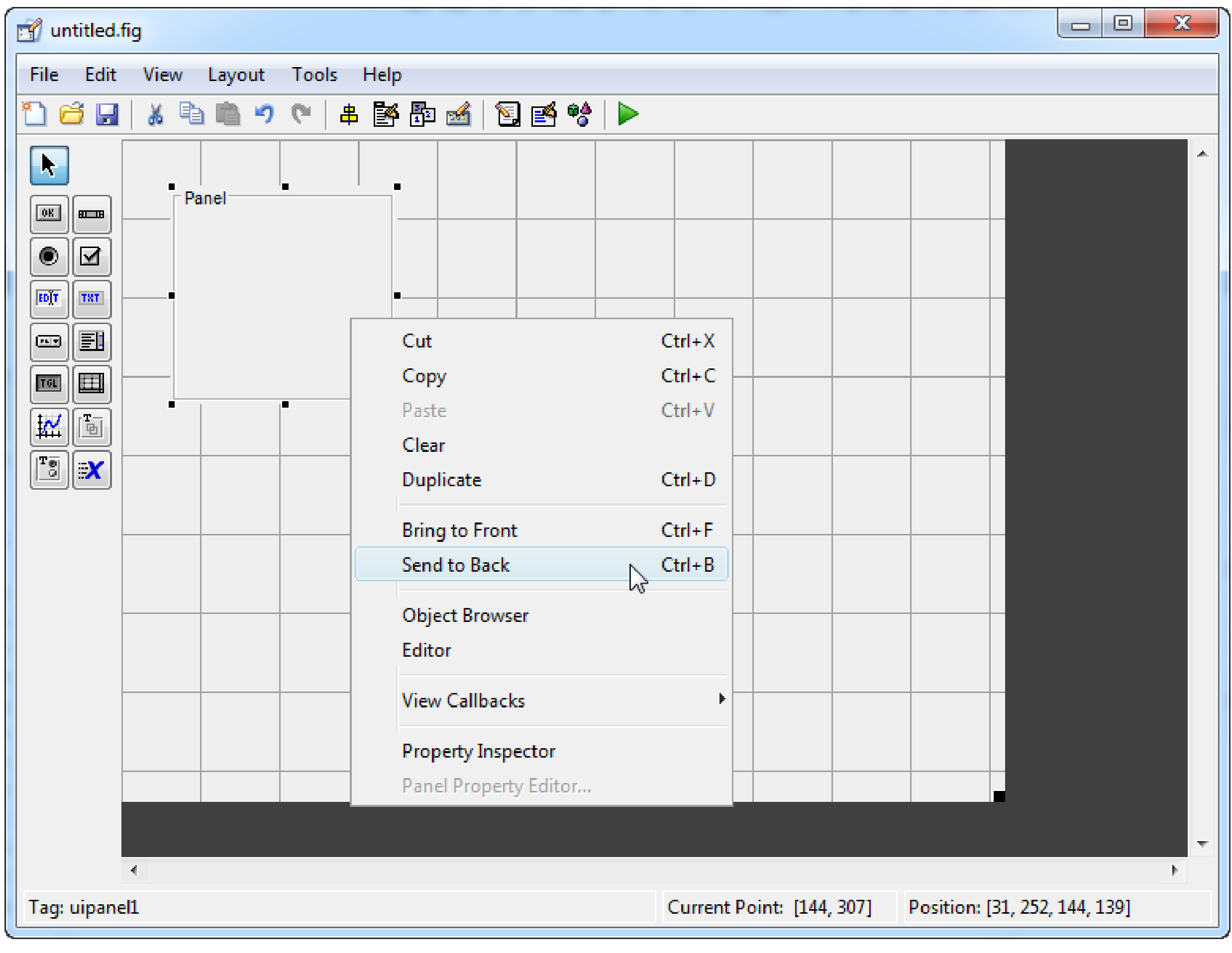


Restoring GUIDE Layouts

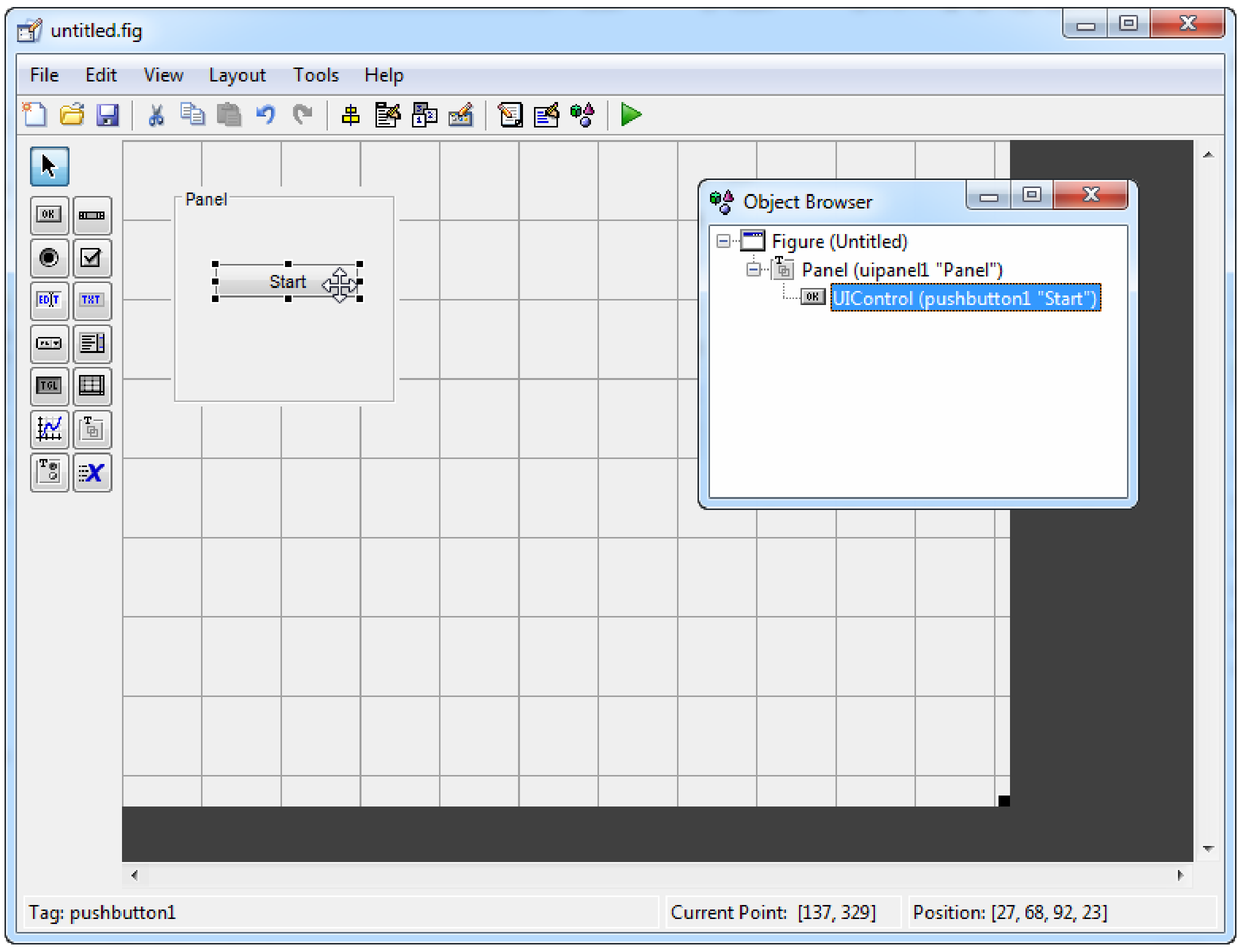
Restoring the layout of a GUIDE UI requires two separate steps:

- Fix the layout in GUIDE.
- Fix the child order of the components.

To fix the layout in GUIDE, open the fig-file in GUIDE and use the **Send to back** option to rearrange the stacking. For example, to move the panel in this layout behind all other components, right-click the panel and select **Send to back**.



To fix the child order of the components, so that they display the same as in GUIDE, select **View > Object Browser**. Then, select a component and move it slightly within the panel or button group. For example, selecting and moving this **Start** button slightly while it is on the panel makes it a child of the panel.



Child components display on top of their parent, so this **Start** button displays on top of the panel when the program runs.

Related Examples

- "Lay Out a UI Programmatically"

Why Has the Behavior of ResizeFcn Changed?

Changes in the behavior of the `ResizeFcn` callback reflect changes to the positioning and layout of figures and containers to make their behavior more consistent with one another and across all platforms.

In this section...
"ResizeFcn Returns Error After Program Launches" on page 2-7
"ResizeFcn Inactive for Invisible Components" on page 2-8
"Unexpected Behavior When Outer Bounds or Drawable Area Changes" on page 2-9

ResizeFcn Returns Error After Program Launches

Starting in R2014b, the `ResizeFcn` callback might execute before all variables in the program file are assigned. When this happens, the `ResizeFcn` callback returns an error.

For example, this program has a `ResizeFcn` callback that uses a variable returned by the `createGUI` function.

```
function mygui

    hs = createGUI;

    function handles = createGUI
        % Create figure and its children
        f = figure('Tag','fig',...
                   'ResizeFcn',@doResizeFcn,...
                   'Visible','off');
        u = uicontrol('Parent',f,'Tag','ctrl');
        handles = guihandles(f);

        % make figure visible
        set(f,'Visible','on');
    end

    function doResizeFcn(varargin)
        length(hs)
    end
end
```

Running this program in the new graphics system results in an error because `hs` does not exist when the `doResizeFcn` callback executes for the first time (when the figure becomes visible).

```
mygui

Undefined function or variable "hs".
Error in mygui/doResizeFcn (line 18)
        length(hs)

Error using mygui/createGUI (line 14)
Error while evaluating Figure SizeChangedFcn
```

To correct the problem, make the figure visible after all the variables that the callback references are assigned. In this case, make the figure visible after calling the `createGUI` function.

```
function mygui

    hs = createGUI;
    % Make the figure visible
    set(hs.fig, 'Visible','on');

     function handles = createGUI
        % Create figure and its children
        f = figure('Tag','fig',...
                   'ResizeFcn',@doResizeFcn,...
                   'Visible','off');
        u = uicontrol('Parent',f,'Tag','ctrl');
        handles = guihandles(f);

    end

    function doResizeFcn(varargin)
        length(hs)
    end
end
```

ResizeFcn Inactive for Invisible Components

Starting in R2014b, changing the size of an invisible container, such as a figure, panel, button group, does not trigger the `ResizeFcn` callback until the container becomes visible.

In previous releases of MATLAB, the `ResizeFcn` callback executes when the size of the container changes, regardless of whether it is visible.

You can control the visibility of figures and containers using the `Visible` property:

- A figure is visible if its `Visible` property is set to `'on'`.
- A uipanel or uibuttongroup is visible if its `Visible` property, and that of its ancestors, is set to `'on'`. For example, a uibuttongroup whose parent is a uipanel is visible when the `'Visible'` property of the uibuttongroup, uipanel, and the figure are all set to `'on'`.

Unexpected Behavior When Outer Bounds or Drawable Area Changes

Starting in R2014b, changing the outer bounds of a figure or container does not trigger the `ResizeFcn` callback.

For example, this figure `ResizeFcn` callback does not execute in the new graphics system when you change the `OuterPosition` (by removing the menu bar) on the second line of this code. However, the callback does execute in previous releases of MATLAB.

```
f = figure('ResizeFcn','display resized');
set(f,'Menubar','none');
```

Starting in R2014b, the `ResizeFcn` callback executes only when the container's drawable area (the inner area) changes. Previous releases of MATLAB might not execute the `ResizeFcn` callback when the drawable area changes.

For example, this uipanel `ResizeFcn` callback executes in the new graphics system when you change the uipanel's drawable area by increasing the border width. The callback does not execute when your run this code in previous releases.

```
p = uipanel('ResizeFcn','display resized');
set(p,'BorderWidth',3);
```

Related Examples

- "Lay Out a UI Programmatically"

Why Does handle.listener Return an Error?

The new graphics system is based on MATLAB objects, which do not support the use of `handle.listener` to create event listeners. To create listeners for graphics objects, use `addlistener` .

Note: Release 2010b and later support `addlistener`, so your code can run in multiple releases after you update it.