

version 01-

Jan-17 Copyright © 2017, Paul Mennen



# Getting started

- Introduction
- Installation instructions
- Experiment with plt5
- <u>Default colors</u>
- <u>Workspace plotting</u>
- <u>Release notes</u>

# Using the plt window

- <u>Preliminaries</u>
- Adding traces
- <u>Selecting traces</u>
- Zooming and panning
- <u>Cursoring</u>
- <u>Right hand axis</u>
- Menu box and menu bar
- Data editing

# **Programming with plt**

- Calling sequence and line styles
- <u>GUI building with plt</u>
- <u>Trace properties</u>
- <u>Axis properties</u>
- Labels and figure properties
- <u>Colors</u>
- <u>Options</u>
- <u>Cursor commands</u>
- <u>Pseudo objects</u>
- Auxiliary plt functions and .m files
- <u>Programming examples</u>

plt **Introduction** version 01Jan17



Copyright © 2017, Paul Mennen

Matlab users may be interested in this toolbox for its focus in one or more of these three areas:

# 1. A plotting interface. An alternative to Matlab's plot and plotyy routines

- Like plot, plt commands can be typed at the command prompt to display your workspace arrays. For simple commands the interface is the same.
- Optimized for data exploration.
- Improved zooming, panning, linear/log toggling, & auto-scaling controls.
- You can interactively select which variables to plot (workspace plotting).
- Automatically generates a legend that also provides trace selection controls.
- Up to 999 traces on a single axis. (Limited to 99 traces if a legend is required).
- Fast and easy cursor movement with delta, rms, mean, y/x, and magnitude readouts.
- Support for dual y-axes and sub-plots, each with individual cursor control & readout.
- Peak/Valley finder, display expansion history, and support for metric prefixes.
- Better looking grid lines with selectable color and style.
- Interactive editing of trace properties, figure colors, and annotations.
- Data editing (mouse or keyboard driven).
- Regular updates based suggestions from users.
- A major advantage of plt is the consistency and flexibility of the command line interface, all explained in a single help file with includes example code for every important option. You no longer will

you have to hunt for the many obscure handle graphics commands used by the native Matlab commands that are scattered throughout the Matlab documentation.

## 2. A GUI building framework. An alternative to Matlab's guide

- Provides a framework for developing graphical user interfaces, usually involving 2D or 3D plotting
- Provides these *pseudo objects* (collections of Matlab graphical objects with a single purpose):
  - cursorimageslider
  - grid
    color picker
    popup
  - plthelp text
- edit object
- Provides these *auxiliary functions* which perform tasks commonly needed in plot oriented GUIs:
  - plttmetricpPvbar
  - prin figpos Pebar
  - datestr
     pltwater
     Pquiv
- The capability to move and resize the pseudo objects and native Matlab objects while recording the positions so that they can be made permanent.
- A methodology for combining these elements presented with a series of examples & demo programs. These examples are designed to demonstrate the use of the various pseudo objects.
- 3. **Signal Processing.** Fourteen of the example programs, in addition to their role in demonstrating various plt features, were also designed to have an educational value in the signal processing field:
  - bounce.m (random walks)
  - curves.m (classic plane curves)
  - dice.m (Monte Carlo simulation)
  - editz.m (z-plane analysis)
  - gauss.m (summation of random variables)
  - gui2.m (classical analog filters)

- julia.m (Mandelbrot & Julia set fractals)
- pltquiv.m (Hermite polynomial interpolation)
- pltmap.m (2-dimensional cubic interpolation)
- square.m (synthesis of harmonic functions)
- tas.m (aircraft performance modeling)
- weight.m (classic sound level weighting curves)
- wfall.m (clipping distortion effects)
- winplt.m (fft windowing)

This toolbox has been extensively tested and verified to run under all Matlab releases from 12.1 (ver 6.1) to R2016a under Windows 10, Windows 8, Windows 7, Windows Vista, and Windows XP. Brief testing has also been done under the Mac and other Unix based platforms.

I hope using plt enhances your Matlab experience. I'm interested in hearing about your problems and suggestions. You can reach me at <u>paul@mennen.org</u>.



# **Installation instructions**

- If you are using Matlab R2014b or later, you may select "Download Toolbox" from the file exchange and all the installation details will be taken care of for you (and you can ignore all the instructions below). If you have an older version of Matlab select "Download Zip" from the file exchange and then follow the instructions below. (Don't be intimidated however it is not difficult at all.)
- Create a new folder called **plt** in any convenient place on your disk.
- Add this new folder to your Matlab path.
- Extract the downloaded archive file (plt.zip) into this new folder. It is important that your unzipper preserves the directory structure. (Most unzippers will do this properly by default.) After the unzip operation the new folder will contain the files and folders shown below.

plt.m	Matlab code for plt
pltt.m Pybar m	Auxiliary function: for adding traces to an existing plt figure Auxiliary function: for displaying functions as
Pebar.m Pquiv.m pltwater.m	Auxiliary function: for displaying error bar plots Auxiliary function: for displaying vector fields (arrows) Auxiliary function: for displaying 3D waterfall plots
figpos.m screencfg.m TaskbarSZ.m	Auxiliary function: for positioning figure windows (called by plt) Auxiliary function: called from figpos to determine screen layout (size, taskbar, etc). Auxiliary function: used by screencfg.m if

pltColor1.mat	automatic method fails Color specification file: Rename this file to pltColor.mat to use Matlab's default colors.
prin.m prin.pdf Pftoa.m	A powerful alternative to sprintf & fprintf. (Called by plt and its demo programs.) A complete description of the prin function. Called by prin. (Implements the additional floating point conversion formats.)
contents.m	brief help text
plt.htm pltfiles∖*.*	Top level html help file A folder containing all lower level html files and images.
plt.chm	full plt help documentation (compiled from above plt.htm and pltfiles\)
<pre>demo\plt5.m demo\bounce.m demo\circles12.m demo\curves.m demo\dice.m demo\gauss.m demo\gauss.m demo\gui1.m demo\gui2.m demo\julia.m demo\plt50.m demo\plt50.m demo\plt50.m demo\pltn.m demo\pltvar.m demo\pltvar.m demo\pltvar.m demo\pltvbar.m demo\pub2.m demo\pub3.m demo\subplt.m demo\subplt8.m</pre>	Example programs. For descriptions, click <u>here</u>

<pre>demo\subplt16.m demo\subplt20.m demo\tasplt.m demo\trigplt.m demo\weight.m demo\wfall.m demo\wfalltst.m demo\winplt.m</pre>	
demo\demoplt.m	Auto sequences through all 31 examples (in the order of the files shown above)
demo\gui1v6.m demo\gui2v6.m demo\qui2ALT.m	Alternative versions of some of the example programs listed above. These alternative programs are also described <u>here</u>

- Note that demoplt.m and the 31 example programs that it runs appear in a subfolder called demo. It's usually convenient to also add this demo directory to your Matlab path. If you prefer not to do that, you can still run the demo programs by using the Matlab Cd command to make the demo folder active.
- You may delete the downloaded plt.zip if you prefer.

## Notes for PC based systems:

If you type plt help or if you click on the "Help" tag in the plt menu box, then plt.chm is opened with the Windows help system hh.exe. (which usually resides in the C:\Windows folder) If hh.exe is missing or if you prefer not to use the chm compiled help file, just delete the chm file and plt will instead open plt.htm using your browser.

## Notes for Unix based systems:

If you type plt help or if you click on the "Help" tag in the plt menu box, then plt.htm is opened in your browser. This file points to many other html files and images inside the pltfiles\ folder. If your browser supports chm help files and you prefer to use it, you can do that by deleting or renaming the plt.htm file.



# **Experiment with plt5.m**

The easiest way to start learning about plt is to start the sample script plt5.m. Just type **plt5** at the command prompt. This simple script plots five traces. Note that he last trace (Line 5) is plotted on the right hand axis. Experiment by trying the following:

- Click on all the objects in the plt window including the trace IDs.
- Try that again with the right mouse button (which usually does something different).
- Click and drag on the x and y axes tick labels (again ... try both left and right buttons)
- Left or right click and drag in the plot area. (On and off the traces do different things).
- Hold down both mouse buttons in the plot area and drag (creates an expansion box).

Expand on what you have learned playing with plt5 by:

- Using plt with some data you are currently working with. Type plt(x, y) just as you would with Matlab's plot routine.
- Use the plt <u>workspace plotter</u> by typing plt with no arguments. Then you can interactively select the variables you wish to plot.
- Run the other <u>programming examples</u>. (demoplt.m sequences through all the examples.)
- Explore the topics in "**Using the plt window**" from the plt help <u>home</u> page.



# **Default colors**

The first thing most users notice when running plt for the first time is that the traces are plotted on a black background. In fact this can be shocking at first because it is so different from the traditional Windows and Matlab standard color scheme that you may have grown used to. Rest assured however that you are not forced into any color scheme. The next section explains why you may not want to change the defaults, and the section after that explains how this is easily done if you prefer to ignore that advice.

## The advantage of the default plt color scheme



The primary virtue of plt's default black plotting background is that you can distinguish far more traces based on color alone when compared with Matlab's default white plotting background.

Why is this? Consider the green trace for example. The plt default uses [0 1 0] for the green trace as you

would expect. However with the standard Matlab color scheme, the green trace is not  $\begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$  because that is too bright and yields low contrast against the white background. So instead they use  $\begin{bmatrix} 0 & .5 & 0 \end{bmatrix}$ . But this means that the green is less saturated making it more difficult to distinguish the green from say a black or dark grey. Similar problems happen with some other colors. This is why the Matlab default trace color order only includes seven different colors. Once you define an 8th trace it cycles back around and uses the same color as the first trace. Especially with the thin traces commonly used, seven is about the maximum number of colors most people can distinguish and even that is not easy. However with a black background and when using plt's carefully chosen default trace colors it is not difficult to distinguish at least three times as many traces based on color alone.

To see that this is true, open the pltn example with 20 traces enabled (i.e. type pltn(20) at the command prompt). Now see if you can match up all 20 traces with the respective trace labels in the TraceID box. If you aren't seriously color blind you probably will find this task easy. Now use the edit box below the TraceID box to change the number of lines to 40. (The 40 trace colors that will be used are shown to the left.) If you have sharp color vision you still probably can identify all 40 traces just by the trace color. You will find that if you switch

to a white plotting background, you will be able to identify far fewer traces based on the trace color no matter what trace color sequence you choose. Matlab's standard plot routine doesn't have features that encourage the use of so many traces, and so you probably haven't noticed this problem with the white plot background. However, plt was designed to work well with many dozens of traces and you will likely take advantage of this capability soon ... and in the process you will come to appreciate plt's default color scheme.

# **Configuring plt to use Matlab default colors (***color specification files***)**

When you type a command into the command window such as plt(x,y) the data specified will be plotted using plt's default colors (i.e. dark background). However if even after reading the previous section you would rather it plot the data using Matlab's default colors, the easiest way to do this is to rename the file pltColor1.mat in the plt folder to pltColor.mat. This is a "color specification file" whose contents are described in the section below. Every time you enter a plt command from the command window, plt will look for this file (pltColor.mat in the plt folder) and will use the specified colors if the file exists. This particular file (until you change it with the methods described below) specifies colors that are the same as Matlab's default color selections.

If you call plt from a Matlab script or function file then plt will not use the pltColor.mat file; however, it will look for a different color specification file. Suppose you write a Matlab script called FooPlot.m that contains a call to plt. Then plt will look for a color specification file called FooPlotColor.mat. This file must be located in the same folder that contains FooPlot.m. So to make FooPlot use the default colors, you could copy pltColor1.mat to FooPlot.m. This can also be accomplished by including the 'ColorDef', 0 parameter in the plt argument list, however the color specification file method is better if you want to allow the user to easily modify the program's colors from the its graphical interface.

There is one other special color specification file that you can use named <pltColorAll.mat. If this file exists in the plt folder it will be used by plt no matter which script or function it is called from and even when plt is entered from the command line. However the colors specified by pltColorAll.mat may be overridden by several methods:

- A color file whose name is derived from the name of the script as explained above will take precedence over the pltColorAll.mat file
- If a color specification file is included in the plt command line (via the 'ColorFile' parameter, of course that is the file that will be used. If the 'ColorFile' parameter includes a null argument (i.e. [] or '', then plt will

ignore all color specification files, thereby reverting to the usual plt defaults. Equivalent to the 'ColorFile' parameter with the null argument is to include the string 'IgnoreColorfile' in the argument to the 'Options' parameter.

• Even when a color specification file is being used, any particular color characteristic may be overridden by the specific plt parameter that controls that feature. All these parameters are defined in the <u>Colors</u> section.

## Creating or modifying a color specification file

Start by opening any plt figure. It will be easier if you choose a plt figure that is already using colors that is close to what you want. To edit one of the trace colors:

- Frist click on the trace that you want to change color.
- Then right-click on the y-cursor edit box
- Select "Properties" from the popup menu by left-clicking on it.
- The edit box in the lower left corner of the small figure that appears contains the color triple for the selected trace. Simply edit the color triple with the value you have in mind. As soon as you hit enter, the color of the selected trace and its associated TraceID label will be changed to the color you entered.
- Or if you don't know the color triple that you want, right-click on the color triple edit box and a "Color Pick" palette will appear allowing you to choose the color you want by changing the sliders and then clicking on one of the 100 colors visible in the palette. Learn more about this palette in the description of the Color Pick pseudo object in the pseudo object section.
- Repeat step one and two, but instead of left-clicking on "Properties", right click instead. A small figure will appear that will allow you to edit the figure colors.
- Select the figure element you want to change with the popup menu.
- As before you can change the color by entering the color triple or by using the color palette.
- After all the colors have been adjusted to your satisfaction, enable the top menu bar by clicking on the "Menu" tag in the MenuBox, then click on the last menu (plt). Select "Save figure colors". You will see a message box that tells you the name and location of the file that was saved. (If your script was called FooPlot, the file will be FooPlotColor.mat. If you called plt from the command line the file will be called pltColor.mat.) If the color file had already existed, it will be overwritten without warning.

Once the color file is created, you may rename it to pltColorAll.mat if you want it to apply to all your script and function files. Although it is usually easier to edit the colors by the method described above, you can also manually edit the colors in the Matlab command window. For example, if you type:

#### clear; cd plt; load pltColor; who;

you will see the following list of variables from the file:

cTRACE - Trace colors (an N x 3 matrix, where N is the number of traces in the TraceID box) CFIGbk - Figure background color CPLTbk - Plot background color CXYax - Axis border color CXY1b1 - Axis label color CDELTA - Delta cursor color

After editing the variables you want to modify, type **save pltColor** to make the changes permanent.

The menus used to edit these colors are also described in the <u>Menu box and menu bar</u> section. There you will also find that there are other ways of accessing those menus which perhaps you will find more convenient.



# **Workspace** plotting



### Starting the workspace plotter

The workspace plotter is a fast way to plot the variables in your current workspace. No more errors from mistyping those long variable names. Just type **plt** (with no arguments) at the command prompt and a window will appear such as the one shown to the left. (This window was actually generated with the included **pltvar**.m script example which creates the variables listed and then calls plt with no arguments.) All the variables in your workspace (except for scalars and strings) will be listed in the workspace plotter figure. The size of the variable name. If your workspace contains many variables, the variable

list may appear in several columns. If your workspace includes 1x1 structures with vector fields (such as the bottom two variables in this figure), then these fields will also appear in the workspace plotter figure using the usual structure notation (struct.field)

#### Choosing an x-vector

The first thing you should do is select the x-vector that you want to plot along the x-axis. As you can see from the instructions (in green) at the top of the figure, you should do this by clicking on the desired x-vector using the right<sup>1</sup> mouse button. You may click on any of the variables shown in white. Note that in this example, the variable called long\_variable\_name is grayed out. This is because only row or column vectors can be x-vectors. Since long\_variable\_name has three rows of 400 elements it can't be selected as

an x-vector.

<sup>1</sup> Actually in this initial situation you may also select the x-vector with a left click. Usually a left click is used for selecting y-vectors, but since you can't do that without at least on selected x-vector it assumes that you are choosing an x-vector in either case. For consistency, you still might want to stick with the right click for selecting the x-vector - a habit that probably will make workspace plotting easier.



Suppose you choose to plot the 400 element row vector SeC along the x axis. After you click on that vector, it turns red for identification and visibility and the tag  $\leftarrow X$  is placed after the variable name indicating this is the chosen x vector as shown to the left.

#### **Choosing a y-vector**

Next you must choose the array (or arrays) to plot along the y axis. The plot routine requires that one of the dimensions of the y array match the length of the x vector. So in this example, the y arrays must have either 400 rows or 400 columns. In this example five of the variables do not meet this condition, so they are grayed out and you will not be allowed to select them.



Suppose you then click on these 3 array names: long\_variable\_name, psvb1, and s.psvb4. As you click on them, the names will turn yellow and the tag ←y is placed after the variable names to indicate that these arrays are to be plotted along the y axis.

📣 plt
longve1
longve2
longve3
psvb1
s.psvb4

If you then click on the "Plot" button plt will create a plot containing five traces. The first 3 traces will be the 3 rows of the array long\_variable\_name. The fourth trace will contain the data

from **psvb3** and the last trace will contain the data from **s.psvb4**. (Note that the order of the traces will be according to the order that the variables appear in the list and not on the

order that you clicked on them.) By default, a maximum of 7 characters are used in the TraceID box which means that some characters may be removed to make it fit. Note that the last character is always included and underscores are removed to save space. Also note that for arrays with more than one row or column a row or column index is attached to the end of the name.

Since a single x vector is being used, the x vector name (**sec**) is used as the x-axis label. In this example, the y-axis is labeled with Y axis for lack of anything better. If you had selected just a single y-axis variable for plotting, that variable name would be used as the y-axis label.



Starting from the previous situation, suppose the numbers in psvb1 were far bigger than the other selected y variables. Then the other traces would be too small when plotted on the same y-axis scale. One way to solve this problem is to plot psvb1 on the right hand axis and leave the other variables on the left side. To do this, double click on psvb1. Note that its color changes to orange and the  $\leftarrow$ y changes to  $\leftarrow$ yR to indicate that this variable will be plotted on the right hand axis. You may select as many traces as you want for either the left or right hand axes. To change a  $\leftarrow$ yR (orange) to a  $\leftarrow$ y (yellow) single click a few times on the variable name. (The exact number of clicks depends somewhat on the situation.)



So far we have only selected a single x vector. Although this will be sufficient most of the time, the workspace plotter allows multiple x vectors to be selected. Suppose you now right-click on seconds and vb2rep. (Since they are graved out, they won't accept a left click, but in this situation they will accept a right click). These two variables will then turn red to indicate that they are to be used as x vectors. They will also be marked with  $\leftarrow x2$  and  $\leftarrow x3$  respectively and the  $\leftarrow x$  marking on **sec** will change to  $\leftarrow x1$ . The digits after the "X" make it easier to know which x vector you have selected for each y vector. Also the first two variables (b1catb3 and b2catb4), which were grayed out before, now turn white because these 800 element row vectors now can be plotted versus either x2 or x3.

Suppose you then click once on both b1catb3 and b2catb4. They will both turn yellow and be marked with  $\leftarrow$ y2 indicating that they should be plotted with respect to x2 (seconds). In this case there is more than one choice of x vectors, so if you click on b2catb4 again, its mark changes from  $\leftarrow$ y2 to  $\leftarrow$ y3 as shown in the figure to the left. Since there are no other possible x vectors to choose from, if you click on b2catb4 one more time the  $\leftarrow$ y3 mark will disappear and the variable name will change back to white, indicating that it is no longer selected.

Every time you click the "Plot" button, the workspace plotter will create a new figure window containing the plot you specified by the various  $\leftarrow X$  and  $\leftarrow Y$  tags. When you have created many plots, pressing the "CloseAll" button is a convenient way to close all of these figure windows (although the workspace browser window itself remains open). If you have many long variable names, you may not be happy with TraceIDs of only 7 characters. In that case, you can specify longer TraceID's tags by typing a command such as **TraceIDlen=17**; before calling the workspace plotter. (Try this before typing **pltvar**). However you will notice one problem. The longer trace names will not fit in the space reserved for the TraceID box and the characters will run into the main plot axis. You could solve this problem by using a plt option variable, which is any variable containing the characters **pltvar** in the variable name. So for example the variables pltopt, pltopt2, another pltoption would all be recognized as plt option variables where as variables pltOpt2 and another\_plt\_option would not be so recognized because it must contain the string "pltopt" exactly (including case).

So before starting pltvar, try typing the following two lines at the command prompt:

# TraceIDlen=17; pltopt = {'xy', [0 .24 .12 .74 .86; -1 .01 .83 .2 .15]};

The first row of the xy parameter gives new coordinate locations for the plot (both left & right axes) and the second row gives new coordinates for the TraceID box. These coordinates can be generated easily by moving/resizing the objects with the mouse. (To see how, look at the manual section <u>GUI building</u> <u>with plt</u>).

If you wanted to make the two commands above permanent for all workspace plotting, create a file on your Matlab path named pltdef.m which contain those two lines. Below are more details about how to use pltdef.m

As an alternative to the 'xy' parameter used above you could use the 'AxisPos' parameter as follows:

```
pltopt = {'AxisPos',[1.8 1 .86 1 2.8]};
```

This would increase the width of the TraceID box by a factor of 2.8 while letting plt choose the height of the TraceID box appropriately (an advantage over the 'xy' command). Unfortunately the AxisPos parameters can't be determined automatically with the GUI building tools. (The "1.8" tells plt to make the blank space to the left of the axis 80% bigger to make more room for the TraceIDs and the ".86" tells plt to make the plot width 86% of the former size so that the plot doesn't run off the right edge.)

To see a list of all the possible commands you can insert into pltopt variables, see these sections of the manual

- <u>Trace properties.htm</u>
- <u>Axis properties.htm</u>
- Labels and figure properties.htm
- <u>Colors</u>
- <u>Options</u>

### pltdef.m

If a file named **pltdef**. **m** exists on your Matlab path then that file will be run before the workspace plotter is opened. This file may contain any Matlab commands, and are usually used to defining workspace plotter defaults and variables.

The workspace plotter looks for any variable that contains the characters

pltopt anywhere in its name. If it finds any such variables it will use the
parameters they contain as arguments to plt when you press the "Plot" button.
For example suppose pltdef.m contains these two lines:

```
pltopt1 = {'Options','Menu','Title','This is a plot
title'};
pltopt2 = {'FigName','Workspace plotter'};
```

Then any plot created by the workspace plotter will have its figure menu bar enabled (from Options), its axis label will be set to 'Frame data' and its figure name will be set to 'Workspace plotter'.

pltdef.m may also include other variables or commands unrelated to plotting
options. For example suppose pltdef.m included these three lines:

```
circleY = exp((0:.04:2)*pi*1j); % 50 point unit circle
circleX = real(circleY);
circleY = imag(circleY);
```

Then whenever you started the workspace plotter you would see circleX and circleY in the workspace variables list. This would allow you to add a unit circle to your plot by selecting these variables from the list (x vs. y) This would be quite useful if you were often plotting z-plane poles and zeros, Nyquist data, or other data that lives in the complex plane.



## **Release notes**

#### Version

- New features have been added to the image pseudo object which also has been generalized so that it now allows the use of more than more than one image object in a single figure window. The default positioning of the optional components have been improved. The number of colormap selections has been increased (from 7 to 10). A new meaning has been added to the Edge parameter. (Specifying "Edge=0" indicates that the complete data range of the input should be used.) A new demo program called julia.m was added to demonstrate some of these new features and also demonstrates several mouse driven gui programming techniques.
- plt.m was a mere 1440 lines of code for its first release back in 2004, but now it has grown to about 4900 lines which is pretty large for a single function. To make this function more manageable it has been split this into two parts. The first part (pltinit.m) includes the code which creates a new plot (a new plt pseudo object ... if you will). The second part (plt.m) contains everything else, i.e. the code to create, access, or modify all the other pseudo objects. This change also makes plt applications clearer, although you can continue to call plt the same way you used to with previous releases if you prefer. This is explained more clearly in the

Calling sequence and line styles section.

• A simpler and more natural method of using the mouse to adjust the size of an expansion box was added. (The older method is still available). See "*Adjusting the expansion box*" in the <u>Zooming and panning</u> section of the help file.

- All eight of the demo programs that display moving traces now also show the number of updates per second. Also programs that include automatic sequencing (curves.m and demoplt.m) display the elapsed time required for the sequence to complete. These measures are useful for comparing GUI speeds between different computers and different Matlab versions. demoplt.m was enhanced, to allow all the demos to be run in sequence without having to click "continue" for each program. This is useful for the above mentioned speed measurements, as well as providing a way to quickly check that no errors where introduced by changes to plt.m or pltinit.m.
- Many of the cursor commands have been simplified
  - 1. Commands of the form:

plt('cursor',cid,'set','param')
have been shortened to:
plt('cursor',cid,'param').
The only exceptions to that are that
'set','position' changes to 'setObjPos'
and 'set', 'activeLine' has been changed to
'setActive'
and 'set', 'expHist' has been changed to
'exRestore'

- 2. Commands of the form: plt('cursor',cid,'get','param') have been shortened to: plt('cursor',cid,'param'). The only exceptions to that are that: 'get', 'position' changes to just 'get' and 'get', 'activeLine' was changed to: 'getActive'.
- 3. The 'axisCBaux' cursor command was changed to 'axisCBr'
- A new option was added to the 'xlim', 'ylim', and 'xylim' commands to allow bypassing the axis change callback.
- 5. All the cursor commands used to be case sensitive, however now they are case insensitive.
- 6. The use of the MotionEdit and MotionZoom

#### 01Jan17

parameters has been slightly simplified. Also a similar parameter called MotionZup has been added. The julia.m demo program demonstrates the use of this new parameter.

These new parameters and the simplified cursor command structure are all described in the <u>Cursor commands</u> section

- The slider pseudo object commands have been simplified by allowing the 'set' argument to be optional.
- A new optional modifier (j or k) is now allowed with the %w and %v formats in prin and Pftoa. Also cell array arguments are now allowed in the input parameter list. This is described fully in the prin help file (prin.pdf) which you can view by simply typing "prin" (i.e. no arguments) at the Matlab command prompt.
- Added a new item to the plt menu in the menu bar (*Hide/Show cursor controls*)
- The enable parameter of the edit and pop pseudo objects was changed from a two way switch to a three way switch (hide/disabled/enabled).
- When you click on the menubox Help tag, plt used to search for just two possible help files (plt.chm and plt.htm). This list has been expanded to search for three other possible help files all having the same name as the application followed with the extension .chm, .htm, or .pdf. Also when a chm help file is specified, you now have the ability to specify that the chm file be opened starting at a particular topic. Details are found in the description of the HelpFile parameter in the Options section.
- Several improvements were made to the HelpText pseudo object. The 'text' command was added to the pseudo object. Also a new option was added for how the text properties are applied. HelpText objects sometimes interfered with the axis callback function, but this problem was solved by assigning the axis callback to the helptext as well.
- A bug was fixed that prevented you from moving some axes in the mouse driven repositioning mode.
- Other minor bug fixes.
- Improved the formatting of the help file, and corrected several documentation errors.

- A compiled version of demoplt.m is now available (for Windows computers only) which allows you to run all the demo programs without having to install plt (and in fact Matlab doesn't even need to be installed). This may be useful if you are unsure if you want to install plt, but still want to investigate its features. It's also useful so that you can run the demos with signal processing educational value on computers that don't have Matlab. This is not included in the file exchange submission since the file exchange does not permit executable files, but you can download it from my web site (www.mennen.org). It's the first entry in the "Other Stuff" category. Amazingly enough it's only a 5 MByte zip file. Noting to install. Simply unzip the file to the folder of your choice and start it by clicking on the "demopltC.exe" file.
- The editz.m example program was improved by combining the two figures into one, both to demonstrate the use of the 'Fig' parameter and to make the application easier to use.
- The pltsq.m example program was rewritten to make it clearer and more concise. A new checkbox was added (called "Live cursor") which when checked allows the cursor to follow its trace as it changes amplitude. Also a smoothing function was applied to the update rate readout to make it easier to read.
- The **bounce** . **m** example program was enhanced so that it also can display arrows representing the velocity of each marker. This was to demonstrate the use of the Pquiv . **m** auxiliary function as well as the use of the super button mode of the popup pseudo object (used to control the length of the velocity arrows).
- Enhanced the gauss.m example program to alternatively display the error terms instead of the convolution functions. A checkbox was added to select whether the error terms should be displayed. These error terms are stored in the lines' Zdata property and shows the value of this technique for storing alternative plot values.
- Added comments and made other minor changes to several of the other example programs.

- Added a new OID code of -3. These OID (Object ID) codes are used by the 'xy' parameter and this new code allows control of the position of the main axes as well as its associated cursor controls. Unlike the other OID codes, the position argument may be complex. The complex component adjusts the scaling of the cursor controls. The editz, plt50 and pltvar examples were modified to take advantage of the flexibility provided by the complex position parameter. A full description of the new OID code can be found under the 'xy' parameter in the Axis properties section. Actually this OID code was added for the October release, but I forgot to document it.
- Some of plt's components and pseudo objects operate repeatedly when the mouse button is held down. The repeat rate of these objects were controlled globally, but now each of these objects may be controlled individually, so that different objects can use different rates. Also a means of disabling the repeat feature is now available. This is explained fully in the description of the edit pseudo object in the <u>Pseudo objects</u> section. (Look for the description of the **repeat** application data property.)

02Nov16

- Added an option to the cursor ZoomOut command to control the zoom amount (formerly fixed at 40%).
- Fixed bugs in pltwater.m which could cause a crash when stopping an updating plot or when closing a figure while the plot is updating.
- Fixed a bug which created a small memory leak when closing a window that was created using the 'Fig' parameter.
- Clicking on the "x" or "y" labels in front of the cursor readouts is used allow the use of the edit readout boxes for modifying the axis limits. There was a bug preventing that from working, however that bug has been fixed in this release.
- Fixed a bug related to using HelpText along with the 'Fig' parameter.
- Fixed a bug related to using the 'xstr' or 'ystr' parameters along with the 'Fig' parameter.
- Fixed a bug in pltwater.m relating to its close request function. (This could have caused a crash when exiting a program while the water fall plot was running.)
- Added help text to the top of the **pltmap**. **m** example program.
- Added the use of the Link to the circles12.m example program so that closing any one of its four figure windows will close all of them.
- Substantially rewrote the <u>Zooming and panning</u> section to describe these features more clearly and to update it to agree with some minor recent changes in the way these features work.
- In earlier releases, the 'Fig' parameter existed, but plt could not be called more than once using the same figure. That restriction has now been lifting making this parameter far more useful. The expanded use of the 'Fig' parameter is described at the end of the <u>Labels and figure properties</u> section. That section describes how the 'Fig' parameter may be used to create multiple plots in a single figure and contrasts that with the 'Subplot' parameter which is used for the same purpose. (Each method has different advantages.)
- The plt50.m example program was rewritten to demonstrate the new use of the 'Fig' parameter. Since that example is focused on data exploration, both of its plots support the full generality of the plt cursoring system with none of restrictions imposed by the use of the 'Subplot' parameter. A new example program called pub3.m was also added to demonstrate the use of the 'Fig' parameter for plots focused on publication quality. The ability to create multiple plots each containing both left and right hand axes made the 'Fig' parameter the right choice for that example. Both examples are described in more detail in the <u>Programming examples</u> section.
- The use of color selection files has been generalized. A new section of the help file (<u>Default colors</u>) has been added to describe the default color scheme and the use of the color selection files.
- The default trace color order has been modified somewhat (although the first 13 colors remain the same as before).
- The cursor behavior when a number is entered into the Ycursor edit box has been changed to be more useful and also more consistent with the behavior of the Xcursor edit box. This behavior is described under "Typing in a cursor location" in the

#### 190ct16

Cursoring section.

- As before, the dual cursor may be enabled using the "Set dual cursor" submenu of plt menu in the menu bar. However now the "Set dual cursor" submenu acts as a toggle, i.e. if the dual cursor was already enabled, selecting this submenu again will disable it.
- Enhanced the pltn.m' example program by adding a pseudo popup menu to control the trace thickness.
- Minor bug fixes and documentation corrections.
- The reach of the plt toolbox has been dramatically extended from its historical 2D roots into the domain of 3D function plotting by including a new pseudo object called **image** (primarily for displaying intensity maps) as well as a new auxiliary function called **pltwater**. **m** (for waterfall plots). Matlab already has many tools for 3D plotting (mesh, meshc, meshz, waterfall, surf, shading, surfc, surfl, surfnorm, ezsurf, contour, contour3, contourf, plot3, slice, isosurface, smooth3, isocaps, isonormals, ribbon, quiver, quiver3, fill3, stem3, sphere, cylinder), yet when you need to scroll thru slices of a massive 3D data set in detail and use the cursors to identify or mark particular values or features, plt now offers something that those other plotting tools can't provide. For now plt only supports 3D plots of single valued functions, i.e. z = f(x,y). The image pseudo object is fully described in the <u>Pseudo objects</u> section. pltwater.m is fully described at the end of the Auxiliary functions section.
- A new example program pltmap.m was added to the demo folder to demonstrate the use of the new image pseudo object. Because this example and its underlying pseudo object have such a wide variety of features, a tutorial of sorts is available to help you learn about these features. For the other examples, this information is included as comments in the program header, but this was not done for pltmap.m because of its length. So to find this tutorial, please see the description of pltmap.m in the <u>Programming examples</u> section.
- A new example program wfalltst.m was added to the demo

folder to demonstrate how to use the pltwater.m function mentioned above. Note that all of the controls on the screen were added by plt & pltwater, so wfalltst only had to supply the data to plot. Some help text is added to the plot to get you started, but eventually you will want to read the full pltwater.m documentation at the location mentioned above. The previous plt release included a demo program called wfall.m which demonstrates the basic ideas involved with creating a waterfall plot, and this demo is still included in the demo folder. However, unlike pltwater.m, wfall.m doesn't have the control, cursoring, scaling, scrolling, and annotating options needed to serve as a general-purpose plotting tool.

- A new auxiliary function pltt.m (which stands for "plt trace") has been added to the plt folder. This function can be used to add traces to an existing plt figure, which makes it the moral equivalent of the hold on and hold off commands that are used with Matlab's native plot command. pltt.m spares you from having to type these extra hold on/off commands and alleviates the confusion that can result from forgetting the hold status. The pltt function is described fully in the new Adding traces section. Related to this new auxiliary function is a new plt parameter, concisely named '+' which allocates space in the TraceID box for traces that are added later with the pltt function. The '+' is the last parameter described in the Trace properties section.
- A new auxiliary function figpos.m has been added to the plt folder to aid in figure window positioning. The main advantage of figpos is that one can position figures relative to the free space available on the screen. The traditional Matlab positioning coordinates don't take into account the taskbar position, the figure borders, and the title bar size. This makes it difficult to avoid overlapping figure windows with the taskbar when you are trying to take advantage of the entire screen real estate. When you supply a position vector to plt (with the 'pos' or 'position' parameter), plt automatically runs these coordinates through figpos. figpos.m calls on screencfg.m (also in the plt folder) to determine the screen coordinates of the area that is available for the figure windows. Both of these new functions

are fully described in the <u>Auxiliary functions</u> section. Many of the demo routines were simplified by taking advantage of this new feature.

 Quite often you will want to add annotations to the figure (permanent, temporary, or toggleable) to identify certain elements of the plot, list equations used, or to provide the user some help in determining how to control or interpret the plot. To avoid having to repeat similar code for this purpose in many of your GUIs, a new pseudo object was added called 'HelpText' which provides these functions. This pseudo object is summarized in the <u>GUI building with plt</u> section and described more fully in the <u>Pseudo objects</u> section. Many of the demo programs (including curves, editz, gauss, gui2, pltquiv, subplt, tasplt, trigplt, and wfalltst) make good use of this feature.

With previous versions, once you drew an expansion box you only had two choices - either left-click to accept the result (i.e. expand the axis to the limits indicated by the zoom box) or right-click to cancel the zoom box, probably to begin again in an attempt to draw the desired zoom box. With this release, you may also choose to make fine adjustments to either the size or the position of the zoom box ... which is usually far easier than simply starting over again. To see exactly how to make these adjustments, consult the paragraph titled "Adjusting the expansion box" in the Zooming and panning section.

24May16

- A few alternative methods of specifying the position and shape of the slider pseudo object were added which provide more flexibility as well as a more compact form for the pseudo slider (in both vertical and horizontal formats). Details can be found in the <u>Pseudo objects</u> section. The <u>bounce . m</u> example program was substantially rewritten to show the use of the new pseudo slider forms (both vertical and horizontal) as well as to demonstrate more sophisticated plotting techniques. Also the new pltmap.m uses many of the vertically oriented pseudo sliders. So many in fact that there would not have been room to fit them all using the full original horizontal style.
- A simplified method of using the 'TIDcolumn' parameter was added. For full details, see the description of this parameter

in the <u>Trace properties</u> section. The pltn.m example program was simplified by taking advantage of this new method.

- In the prin command, two optional modifiers (+/-) for the W format were added that allow the output to be padded with blanks. Also an easier way to view the prin help file (prin.pdf) was added simply type prin (i.e. without arguments) at the Matlab command prompt, and the help file will appear.
- The ability to change the name of the trace (as it appears in the TraceID box) after the initial plt call that specified these trace names. See "*Right-clicking on the Cursor ID Tag*" near the end of the <u>Cursoring</u> section.
- A new cursor command ('updateN') was added which has the same effect as the 'updateH' command except that the cursor callback (defined by 'MoveCB') is not executed. Details in the <u>Cursor commands</u> section
- The default figure name (if none was specified with the 'FigName' parameter) used to be plt but now the name of the function that is calling plt is used as the default name.
- plt('pop', H, 'get') which is already an abbreviation for plt('pop', H, 'get', 'index') may now be further abbreviated as plt('pop', H).
- plt('edit', H, 'get') which is already an abbreviation for plt('edit', H, 'get', 'value') may now be further abbreviated as plt('edit', H).
- plt('slider', H, 'get') which is already an abbreviation for plt('slider', H, 'get', 'value') may now be further abbreviated as plt('slider', H).
- The argument for the 'HelpFile' and 'HelpFileR' parameters normally specifies document to display, however, now the parameter may also be a Matlab command to be executed.
- Many improvements to the **demoplt**.m program including:
  - Added a string (lower right corner) that shows the number of (uncommented) lines of code used in the currentlyselected example program. At a glance, this number gives you an idea of the complexity of the example relative to the other demos.

- Added a modal dialog on exit to control whether changes to the setup should be saved in the demoplt.mat configuration file. (If no changes were made to the configuration, the modal dialog is skipped.)
- A history of the changes to the color configuration is displayed at the top of the listbox. This is partly done just to demonstrate the use of the ColorPick callback function.
- The size of the figure window is preserved in the configuration file, but the figure window is always placed in the far lower right corner of the screen to minimize the overlap between the demoplt figure and the example programs being displayed.
- Many improvements to the Curves.m example program including:
  - Added four additional interesting curves to the collection.
  - Greatly enhanced the program's educational value by adding annotations to the plot area describing many details of the currently displayed set of curves.
  - Added a way to reset the curve parameters to the default settings. (Default button)
  - Improved the way functions may be defined in complex or parametric form.
- Fixed a bug introduced a few years ago causing plt to crash when improperly editing an *edit pseudo object*. Also fixed a dozen or so other minor bugs in plt and the demo programs.
- Major enhancements to the subplot capabilities:
  - Previously you were limited to two columns of plots. As of this release any number of columns are allowed. A new example program called subplt16.m was added to demonstrate this new flexibility.
  - A way of altering the default vertical spacing between plots is now provided. (Previously only the horizontal spacing was adjustable.)
  - Previously the cursors and x-axis limits for all the plots within a column were synchronized. With this release, this synchronization is optional and is referred to as "linked" mode. The new alternative mode is called "independent"

mode, and as the name implies, all plots may have different x-axis limits and different cursor locations. A new example program called subplt20.m was added to demonstrate this new mode. Note that the independent mode allows you to pack even more plots into a given area.

- Since subplots were added late in the development of plt, many plt features did not work appropriately when subplots were enabled. For example the peak/valley finder and marker select button did not work. Also 5 of the tags in the menu box did not work for subplots (LinX, LinY, Mark, Zout, XYrotate). I'm happy to report that all those problems are now fixed. The delta cursors now also work when subplots are enabled, however you can only use them on the (lower left) main plot since there isn't enough room to display the delta cursor readouts for all the subplots.
- A few enhancements to the subplt8.m example program, which now uses the default Matlab color scheme with a modified trace color order (accomplished using plt's 'ColorDef' argument).
- The order of the y-axis cursor readout edit boxes was swapped (left to right) as that felt more natural. In any case, the color of the cursor readout edit boxes are linked to the color of the trace that is being cursored making confusion unlikely.
- Trace data may now be included in the plt argument list using cell arrays. Full details on how to use this are included in the <u>Calling sequence</u> section. The <u>gauss.m</u> demo program was modified (and simplified) by taking advantage of this new option.
- When the multiCursor is enabled, plt previously added a cursor marker to every trace. Now however, cursor markers are not added to disabled traces or to traces whose Xdata differs from the currently active trace. The full details are in the <u>Cursoring</u> section.
- The trace color order was modified slightly. To see what the new trace color order looks like for the first 99 traces, see the section called <u>Selecting traces</u>. To see a list of the trace colors in terms of the RGB triples, see the description of the <u>TRACEc</u>

#### 08Nov15

parameter.

- The 'Position' or 'Pos' argument specifies the [xleft ybottom width height] position of the plt figure window as before, however now you may specify just the size by using a 2 element vector in which case the xleft and ybottom values will be set to their default values. For example 'Pos', [500 600] would be equivalent to 'Pos', [9 55 500 600] since the default values of xleft & ybottom are 9 & 55 respectively.
- A new option (replace vs. append) was added to the **'ColorDef'** argument. Also the default changed from append to replace. See the details in the <u>Colors</u> section.
- The FigBKc, PltBKc, TraceC, xyAXc, xyLBLc, GRIDc, and TraceC parameters continue to accept standard Matlab color triples as before (i.e. a row vector of 3 numbers all between zero and one). But now there are two additional color specification formats allowed. The first is "percent", meaning that [15 35 92] is equivalent to the matlab color triple [.15 .35 .92]. The 2nd alternative method is to use a single number to represent all three colors. Using that method the color above would be represented by the number 153592. Of course if you prefer the standard Matlab style, you can ignore all this, but otherwise, check out the full description in the <u>Colors</u> section.
- Various improvements to the **CUTVES**. **m** demo program. These improvements don't demonstrate any additional plt features, however the new curves that have been added rounds out this collection of classic and unusual plane curves ... perhaps providing some educational or recreational value.
- Fixed several problems relating to the screen positioning of objects when using the new graphics engine of R2014b or later.
- Fixed errors in the documentation describing the Cursor commands, the xy parameter, and the Options parameter. Also made other minor documentation corrections and improvements.
- Enhanced the **pltquiv**. **m** example program to demonstrate data editing on quiver plots, the use of the **MotionEdit**

parameter, and real time Hermite polynomial interpolation.

- Added a default for the Cursor ID, simplifying many of the demo programs.
- Simplified data editing routines by adding the cursor index to the 'Dedit' figure application data. Added a full description of the Dedit array in a table at the end of the Data editing section.
- Minor improvements to the grid pseudo object.
- Multi-line titles are now allowed. See the **Title** parameter in the <u>Labels and figure properties</u> section.
- Added the title property prefix (^) which is described in <u>Axis properties</u> section.

# 03Jul15

- Added right click actions to the "pub" button of the pub2.m example and to the "Delete P/Z" button of the editz.m example
- Improvements in the plt('datestr') function
- Fixed bugs relating to the use of the 'Fig' parameter.
- Fixed a bug which caused the right click action to be executed when double clicking on the LinX/LogX or LinY/LogY menubox tags.
- Fixed a bug in the xView slider feature that caused it to interfere with the graph title.
- Fixed a bug in the 'Nocursor' option that unintentionally left the cursor marker visible.
- Fixed bugs causing error messages when cycling quickly thru demoplt.
- Many corrections and improvements to the help file especially in the <u>Pseudo objects</u> section which was substantially rewritten to include the motivation behind these objects.
- Added the "multiCursor" feature which allows you to cursor all plot traces simultaneously. Usually this cursoring mode is enabled from the Yedit menu, but sometimes you may want to have this mode enabled at startup. This can be done using the new "multiCursor" Options item. (As with all the other option items, only capital letters are significant so if you want to be concise just the letter C is sufficient). The trigplt.m demo

program was modified to use this new options item to enable the multiCursor feature. The multiCursor feature is more completely described in the <u>Cursoring</u> section.

- Added the "xView slider" feature that offers yet another way to pan and zoom the display horizontally. It is especially useful with long time records because of how easy it is to use and how well it gives you feedback about where and how much of the data you are viewing. Usually this feature is enabled from the Yedit menu, but sometimes you may want to have the xView slider visible at startup. This can be done using the new "xView" Options item. (As with all the other option items, only capital letters are significant so if you want to be concise just the letter V is sufficient). The pub2 .m demo program was modified to use this new options item to enable the xView slider. The xView slider feature is more completely described in
- Modified the plt50.m demo program to demonstrate how to modify the cursor size and shape.
- Fixed a bug in the implementation of the Link parameter (with pre-R2014b versions)
- Fixed a bug in the initialization of the cursor data window (with R2014b only)
- Fixed a bug which caused plt to behave improperly when given an empty 'Right' parameter.
- Various documentation improvements

the **Zooming and panning** section.

13Mar15

Although plt retains all its previous capabilities plt has been expanded and refocused as a GUI building tool, primarily for applications which include one or more 2D plots. This was accomplished by enhancing the pseudo objects and by adding the plt move command to allow you to move or resize all the graphical objects using the mouse. When you reposition or resize an object, its new coordinates are displayed in the command window so that you can make the new locations permanent by embedding these coordinates into your program. These operations are described in a new section of the manual: GUI building with plt

- In addition to the **plt move** command, you may also enter the repositioning mode by right-clicking on the cursor delta button. When you do this, the delta symbol on the button changes to a double right arrow to indicate that you are now in repositioning mode. Also the uicontrols will be grayed out to indicate that these controls may be repositioned/resized. Right-clicking on this button a second time cancels the repositioning mode, changes the button symbol back to delta, and restores all the controls to their original functions.
- Two new example programs (**guil.m** and **gui2.m**) where added to help you learn how to use the new plt GUI building features. An alternate version of gui2.m called gui2ALT.m is also included in the demo folder to demonstrate an additional approach to the problem that may be instructive. Both of these new examples use Matlab features not found in Matlab v6.1 so alternate versions (called gui1v6 and gui2v6) are available for running on that version. demoplt queries the Matlab version and automatically runs the appropriate version of gui1 and gui2.)
- Support for the latest version of Matlab (R2014b) has been added. The plt options related to the line erasemode are ignored when running R2014b since The Mathworks has abandoned support for the erasemode line property. The erasemode options continue to work as before for all older versions of Matlab (dating back to Matlab 6.1). The defaults for the grid style and color have changed somewhat because of this, and this is described in the default section of the <u>GRIDc</u> parameter. As before right-clicking on the Grid tag in the menu box toggles the grid style between solid and dashed lines.
- Reorganized the plt item in the menu bar. (Also plt Hardcopy was moved from the file menu to the plt menu.) The new plt menu now shows the mouse shortcuts (in blue) for most of the submenus. These shortcuts are often more convenient since the menubar is hidden by default.
- The AxisPos parameter which provides a way to reposition/resize the main plotting axis and/or the TraceID box is still supported for backwards compatibility, but for new programs a more versatile xy parameter was added. It serves similar functions but also allows you to reposition the menu box

as well as any plotting axis inside the plt window. The Xy method of specifying the axis size and positions is simpler and easier to use and is also compatible with the new repositioning mode mentioned above. (All the objects that can be moved and resized via the Xy parameter may also be moved and resized using the mouse.) The demo programs that used the AxisPos parameter were modified to use the xy parameter instead. A complete description of the new Xy parameter may be found at Axis properties

- Several enhancements to the Edit pseudo object:
  - Edit objects may be incremented or decremented by a fixed value as before, but now may also be incremented or decremented by a percentage of the current value.
  - The value property was split into two different properties (value and val). They differ only in that when you set the latter one (val) the edit object's callback does not get called.
  - An option was added to allow a label to be associated with the edit pseudo object.
  - A more concise form for creating the edit pseudo object was added for convenience.
  - Several **get** commands where added allowing you to query the current setting for any of the edit pseudo object parameters.
  - Edit objects now come in two types, type 1/2 using figure/axis coordinates respectively. Type 1 is usually the most convenient. (Previously only type 2 was supported). See details at <u>Pseudo objects</u>
- Several enhancements to the Popup pseudo object:

19Jan15

- The right click operation of Popup pseudo object (which advances the popup without opening it) was enhanced so that holding down the right mouse button continues to advance to the next selection for as long as you hold down the button. The repeat rate may be altered by the same method used to control the edit pseudo object repeat rate.
- An option was added to allow a label to be associated with the popup pseudo object.
- An option was added for swapping the role of the left &

right buttons (super-button mode).

- A method for opening or closing the popup from a program was added.
- Setting the index of the popup using the 'index' parameter used to call the popup's callback function after the index was set, just as if you had clicked on that selection from an opened popup. Now the callback will not be called. However if you want the callback to be called in this circumstance, specify the negative of the index as the index parameter.
- A more concise form for creating the popup pseudo object was added for convenience. See details at <u>Pseudo objects</u>
- Several enhancements to the Slider pseudo object mostly related to the log increment mode. Also the calling sequence was altered to allow easier use of the optional parameters.
- A new mode was added to the **DualCur** parameter that allows specifying the dual cursor trace as an offset from the primary cursor. See the demo program **gui2**. **m** for an example showing the utility of using this mode. Also the ability to select the DualCur trace interactively was added to the plt menu. See the <u>Dual Cursor</u> section for more information.
- A new parameter (CloseReq) was added which allows you to specify a function to be executed when the plt window is closed. Details about this new parameter may be found at <u>Labels and figure properties</u>. The plt50.m, wfall.m, and gui2.m example programs where modified to demonstrate the use of this new parameter.
- A new parameter (**TIDC**) was added which allows you to choose the background color for the TraceID box. Details about this new parameter may be found at <u>Colors</u> Also the solid patch used in the TraceID box to indicate that the trace is on the right hand axis is now optional. To see how to do that, look at the details of the TraceID parameter here: <u>Trace properties</u>
- The demo program pltsq.m was rewritten to use pseudo popups in place of the uicontrol popups of the previous version. Also the two buttons (start & stop) were replaced with a single pseudo popup in the "super button" mode. The older version was renamed to pltsqALT.m and is also included in the demo

folder so you can compare the relative merits of these control types.

- A new example program (pub2.m) was added which demonstrates how to create a plot optimized for publication by first setting up the desired viewing window using the cursors on a data exploration mode plot. Another goal of this particular example was to demonstrate as many unusual plt parameters and programming techniques as possible.
- The cursor mainCur command was renamed to update and the cursor updateH (update hold) command was added (similar to the update command except that y-axis limits are not permitted to change.
- The showTrace command was renamed to show. Also new is that this command may now be used for both setting and reading which traces are currently visible. The example program gui2.m demonstrates both uses of the show command. See details at <u>Calling sequence</u>
- All the pseudo objects that accept a 'position' argument will also now accept the more concise 'pos' form of this parameter.
- Several enhancements and bug fixes to the various example programs.
- Many improvements and corrections to the plt help file.
- Enhancements to edit pseudo object: Values can now be continually incremented or decremented by holding down the mouse button. Absolute incr/decr values may be specified (as before) or now they may be specified as a percentage of the current value. incr/decr values may now be used with floating point edit pseudo objects as well. (Zero increment disables this feature.) Position may be specified in normalized units (as before) and now data units may be used as well. The slider in the circles12.m demo program was changed to an edit pseudo object to illustrated its use.
- Improved log grid lines. Now includes subdecade grids for 6 decades or less
- HardCopy figure window now defaults properly for bmp files

• Improved appearance of the menu box (background and border colors)

#### 13Mar14

- Added right click action to "LinY/LogY" tag. It now brings up the hard copy dialog.
- Added 'Link' parameter to force groups of plt figures to close together. The demo programs editz.m and tasplt.m were modified to take advantage of this new parameter
- Enhanced a few of the functions plotted in curves.m
- Added images & thumbnails of all the example programs to the help file
- Fixed various bugs related to log axis scaling and panning
- Fixed LinX/LogX switching bug for subplots
- Fixed bug which prevented data editing of subplots
- Fixed bugs with the subplot parameter and in the weight.m demo program
- Fixed bug occurring when using the 'MoveCB' parameter with subplots
- Fixed bug occurring when right-clicking on the grid tag using default colors
- Other minor bug fixes.
- Renamed vbar.m, ebar.m, quiv.m, and ftoa.m to Pvbar, Pebar, Pquiv, and Pftoa to reduce the probability of name collisions (since these files typically reside in the plt folder on the Matlab path).
- Added Linesmoothing to the list of strings that may appear in the Options parameter. When this option is specified, the Linesmoothing property of the traces is set which tells Matlab to use anti-aliasing techniques to make the traces look smoother. You can also toggle the Linesmoothing property by rightclicking on the marker button (labeled with "o"). See important details about this property in the <u>Cursoring</u> section
- Changed the default characteristics of the grid lines. The new default is solid lines instead of dashed lines and the default color has changed from dim grey (30%) to very dim gray (13%). Also the default erase mode for the grid lines has changed from 'norm' to 'xor', The grid linestyle may now be selected by using the new GridStyle parameter. The

Linestyle parameter was also added to the 'init' action of the grid pseudo object. The grid lines used to update only when a zoom or pan operation was complete, but this has been changed so that the grid lines are now updated during the pan and zoom operations. The color file format was updated to include the grid lines erase mode as well as the new GridStyle parameter

- Left-clicking on the Grid tag in the menu box has the same meaning as before (i.e. toggling between grid lines and ticks) however now you can also right click on the Grid tag. The effect of this right click is to toggle back and forth between the current default grid style and something similar to the older grid style (i.e. slightly brighter grey dashed lines in normal erase mode).
- Left-clicking on the Menu tag in the menu box has the same meaning as before (i.e. toggling the menu bar visibility) but there is a new meaning to right clicking on the Menu tag. The effect of this right click is to open a new figure window containing a textual view of the trace data. See *Cursor Data Window* in the Menu box section

### 03Jan14

- Replaced the "Color/Lines" heading in the menu bar (which included 4 submenus) with the "plt" heading containing the original 4 submenus and 4 additional ones. Three of the new menus mirror right click actions of the GUI objects, but the menu bar makes the actions easier to discover and remember. The 4th action (Delete Cursor Annotations) is a new feature which allows you to quickly delete all annotations entered using the Mark tag. (Previously these annotations could only be deleted one at a time.)
  - Added the swap x/y action which swaps the x and y data of all traces (i.e. displays inverse functions). This action is available from the menu bar (plt menu) as well as by right clicking on the LinX tag
  - The installation has been simplified by always including both the compiled help file (plt.chm) as well as the all individual html files used to create it. By default PC systems will open the plt.chm compiled help file and Unix based systems will open plt.htm (the top level html file). However you can change these defaults. See the Installation instructions for details.
  - As before the HelpFile parameter modifies the help tag left

click action. In a similar manner, the new HelpFileR parameter modifies help tag right click action. For a full description of these parameters see <u>Options</u>.

- Enhancements to the ColorPick pseudo object.
- Improvements to the example programs include:
  - *Fontsize* popup added to demoplt.m
  - # of lines control added to pltn.m
  - *Cycle* button added to **curves**.m
  - The upper plot in editz.m now demonstrates the use of Dual Cursors.
  - Other more minor enhancements.
- Y-axis metric prefixes are now disabled when using the SubTrace parameter
- Fixed a bug which sometimes prevented you from being able to close the plt window.
- Many other bug fixes and documentation improvements.
- Removed the plt('ftoa') and plt('vtoa') commands and replaced and expanded on this functionality with the separate functions prin.m and ftoa.m. (ftoa is called by prin and is not usually called directly) See the new file prin.pdf for a description. Modified demo\pub.m as well as several of the other example programs to take advantage of the new features provided by the prin() function.
- The cursor control buttons (a group of buttons in lower left corner) previously consisted of 3 buttons (peak finder, valley finder, and delta cursor). As of this release a 4th button is included in this group (called the marker button) and is labeled with the letter "o". The function of this new button is described in the <u>Cursoring</u> section but if you click on this button 3 times you will already know all it does.
- Left/right-clicking on the x-axis label still moves the cursor left/right one data element as it did before, but now this has been enhanced so that the cursor continues to move as long as you hold down the mouse button. The default repeat rate is 33 times a second, but this is adjustable. See the full description in the <u>Cursoring</u> section. This gives the x-axis label nearly all the

features of the optional x-axis slider without taking up any extra space.

- Including the letters H or Hidden in the options string now instructs plt to leave the plt figure hidden until the figure visibility property is latter set to 'on'.
- Fixed a bug that appeared only on MACs that caused incorrect displays of some floating point numbers in edit boxes.
- Other minor bug fixes.
- Changed the focus of plt somewhat (in the <u>Introduction</u>) from plotting to GUI tools although many users will still think of plt primarily as a substitute for plot and plotyy.
- Added the ColorPick pseudo object. (See <u>Pseudo objects</u>). The demoplt.m function was modified to demonstrate how to use this new pseudo object.
- The data editing capabilities of plt (See <u>Data editing</u>) were completely revamped, both to make it easier to use and to allow its use with subplots. The <u>demo\editz.m</u> demo routine was substantially rewritten to take advantage of the new data editing interface.

#### 06Jun12

- Modified the demo\trigplt.m demo routine to show how to create a "clipboard" button as well as how to modify the position and appearance of the traceID box.
- Modified the demo\pltvbar.m demo routine to show how to use the Grid pseudo object to display some tabular character data.
- The plt('ftoa') function was enhanced to include an additional formatting code ('C') which is useful for displaying color triples.
- Many minor bug fixes.
- Added the 'SubTrace' parameter to allow more flexibility on how traces are assigned to the various subplot axes. (See <u>SubTrace Parameter</u>)
- Added a new interpretation of the 'SubPlot' parameter which allows you to adjust the spacing between two columns of

subplots. (See <u>Subplot Parameter</u>)

- The demo program demo\pub.m was expanded to include yet one more plot to show how subplots can be used when formatting plots for publication and to show how the SubTrace parameter is used.
- Fixed bugs which were causing the delta cursor and peak/valley finder features to work incorrectly when using Matlab 2009b or newer.
- Fixed bugs which caused the AxisPos parameter to be interpreted incorrectly when used with subplots.

02Dec11

11Mar10

Added the 'MotionEdit' parameter which allows you to specify a function to be executed while the mouse is being moved during a data edit operation. (See <u>Mouse Motion Functions</u>.) The demo\editz.m program was updated to take advantage of this feature. If you have used previous versions of editz (which updated the plots only after you have selected the final root location), you will recognize the benefit of seeing the transfer function plot update in real time as you drag a root to a new location.

- In a similar vain to 'MotionEdit' the 'MotionZoom' parameter was added which allows you to specify a function to be executed while the mouse is being moved during a display zoom operation (i.e. while the zoom box is being dragged). (See Mouse Motion Functions.) The demo\gauss.m function was updated to take show how to use this feature.
- Fixed a bug which caused the display to flicker while a plt window was being initialized.
- Added more useful information to the "Calling sequence" section of the documentation.
- The demo program demo\pub.m was expanded to include an additional plot (bar chart).
- A bug was fixed that caused an error when null strings were specified in the TraceID parameter.
  - The cursor ID (cid) is now saved in the axis user data even for subplot axes.
  - Changes for compatibility with Matlab ver 2009b

- Added a cursor mode to show distance from cursor to the origin or to the marked location, i.e. abs(x+iy). See *Average*, *RMS*, *Slope*, & *Distance readout* section in <u>Cursoring</u>.
- Improvements and bug fixes to the "plt hardcopy" function (from the file menu or optionally in the menu box). Defaults are now saved in pltHcpy.mat in the folder containing plt.m.
- Fixed a bug that caused the cursor to misbehave when cursoring non-monotonic traces on the right hand axis.

#### 25Feb10

- Fixed a bug that caused annotations added with the "mark" feature to sometimes be associated with the incorrect axis.
- Fixed a bug that sometimes caused the grid lines to not initialize properly until the first time you clicked on the figure window.
- Fixed some bugs related to grid lines on subplots (turning on/off via the menu box tag or via the 'Ticks' options argument).
- Fixed bug in setting x-axis limits for right hand column of subplots ('Xlim' parameter).
- Improved cursor hiding feature (right click of y-axis label) to work with subplots as well.
- Added a new example (pub.m) to the demo folder to show how to optimize a plot for publication (instead of the typical data exploration uses of plt). Also demonstrates the use of the new property prefix characters described below.

Previously there were two property prefix characters ('+', '-') which refer to the left and right axis properties. (Without the prefix characters plt assumes properties are to be applied to the line objects.) Now three additional prefix characters have been added ('<', '>', '.') which refer the properties to the left y-label, right y-label and x-label respectively. A property may be preceded by more than one prefix character. For example '>. - FontSize', 14 will change the font size of the right y-label, the x-label and the tick marks of the right hand axis. See Axis properties for details.

- Improved demoplt.m so that it is easier to tell which demo is currently running.
- Fixed a bug with the workspace plotter that occurred when the

#### 03Dec09

workspace contained a large number of variables.

- Fixed a bug in plt's Edit line selection from the color menu.
- Fixed a bug in demoplt.m that would sometimes cause an error if you clicked in the listbox window.
- Fixed a bug in the plt('open') command for non Windows systems
- Significant enhancements to the winplt.m demo program including the use of an application specific help file as well as the use of a popup control to edit a vector (the window kernel in this example).
- Added the **plt hideCur** command. Equivalent to rightclicking on the y-axis label. See <u>Cursor commands</u>.
- Added the plt('showTrace', e) command. For details see the "DIStrace" section at <u>Trace properties</u>.
- Documentation updates
- All the example routines in the demo folder (except the scripts) were substantially rewritten for improved clarity.
- The winplt demo now demonstrates how to include two help tags in the menu box (one for plotting help and the other for application specific help). The files winplt.chm and winplt.pdf were added both to demonstrate this feature and to provide more extensive information about the winplt demo program.
- The demoplt.m program was rewritten to make it easier to run the individual examples as well as auto-sequence thru them. A listbox was added to allow easy viewing of the demo programs help text.
- In addition to pure strings to be evaluated, the plt callback functions now allows you to specifying a function handle of the form @func as well as the cell array form
   {@func, arg1, ar2, ... argn} which lets you
   provide arguments to the callback function. This applies to the
   plt parameters moveCB, axisCB, TIDcback, Xstring,
   and Ystring as well as the callbacks for the auxiliary plt

250ct09

07Nov09

functions edit, pop, and slider. Note that you can't take advantage of the string substitution features of these callbacks when using the function handle forms. This form is now used in many of the <u>Programming examples</u>.

- Added the ability to modify properties of the left or right axis using plt parameters. (See <u>+AxisProp</u>, <u>-AxisProp</u>). This feature is demonstrated in the <u>demo\pltvbar</u>.m example program.
- Enhanced the data editing features to include the ability to add or remove data points from a trace. Also added a more natural way of entering the data edit mode. See <u>data editing</u> for details.
- Added a new set command ('val') to the plt('slider') pseudo object. See <u>Pseudo objects</u>
- Added the plt('save', file) command. See <u>menu box</u> for details.
- Fixed a bug that caused the right hand axis to be missing from getappdata(gcf, 'axis'), as well as a minor bug fix in the data editing mode and a bug fix relating to a rare crash when saving a file from the menu bar.
- This is the first release which includes the full . **m** source code. The pcode or dll versions are no longer needed.
- Added the TraceMK parameter to allow you to include the line types in the trace selection box. See the demo programs demo\trigplt.m and demo\subplt.m for examples of using this parameter. See <u>Trace properties</u> for a complete description of the TraceMX parameter.
- Added a new example to the demo folder (trigplt.m) to demonstrate the use of the TraceMK parameter as well as the use of the slider pseudo object.
  - Added the Fig parameter to allow plt to use an already opened figure window instead of creating a new one. See <u>Labels and figure properties</u> for a complete description of the Fig parameter.
  - Fixed a bug which caused plt to crash when called from a guide application.

#### 11Aug09

# 11Mav09

- Fixed a bug which sometimes caused the wrong foreground color in the cursor edit boxes.
- Fixed a bug which sometimes caused the Xlim or Ylim parameter to be ignored.
- Improved the operation of the x-axis slider.
- The SubPlot parameter was added, a major enhancement in plt's ability to place multiple plots in a figure window. This is more flexible and far easier to use than the subplot method introduced in version 19Aug08. The demo\subplt.m and subplt8.m examples were rewritten to use the new SubPlot parameter. The weight.m example (which was also simplified by using the new parameter) also demonstrates a trick for allowing multiple traces in each subplot. See Axis properties for a complete description of the SubPlot parameter.
- The Xlim and LabelX parameters now allow cell array inputs so that they may specify values for the right hand column of subplots. The LabelY parameter now allows a cell array input so that you can specify labels for all the subplots as well as for the right hand axis of the main plot, and Ylim parameter can take a cell array input to specify limits for both the left and right hand axis of the main plot.
- Added an easier way to remove cursor objects for production plots. See the -All and the Nocursor items under the Options parameter. (The last figure of the circles12.m example demonstrates this usage.)

# 18Jan09

- right-clicking on the Menu tag in the menu box will now toggle the line style used for of all traces. (See Menu box)
- Added the Lhandles, axis, and cid cid figure application properties. (See <u>Calling sequence</u>)
- Added the 'tag', 'MenuBox' property to the menu box axis to make it easier to find.
- Added a new parameter to the plt('grid', 'init', ... command (the erase mode). (See <u>Auxiliary plt functions</u>). Also the default erase mode for the grids has changed from xor to normal. You may select the exclusive-or erase mode for the grids by specifying a grid color containing a negative value (e.g.

'GRIDC', [0 - .2 .4]). The actual grid color used in this example is [0 .2 .4]).

- Added a special case to the plt('cursor', cid, 'set', 'activeLine', a, k) command for when a is zero. (See <u>Cursor commands</u>)
- Added a special case for null trace ID strings. (See <u>Trace properties</u>)
- Fixed bug that could cause a crash when selecting logarithmic y-axis.
- Fixed bug that caused a crash when using the color selection box to edit plot colors.
- A zero is now allowed for the 3rd or 4th element of the 'position' parameter to indicated that the width or height of the window should be chosen so that circles will look symmetric. However after stretching the figure size with the mouse, the circle will then look like an ellipse. If you want the circle to look symmetric even after the figure window is resized, you should follow the plt command with the command axis('equal').

#### 030ct08

- The example programs circle12.m and editz.m were enhanced to demonstrate the feature mentioned above.
- The TraceID box has been made wider in situations where it will not run into the y-axis label.
- The file **plt.dll** is now compressed into a zip file called pltdll.zip. This replaces a different type of compression used in earlier releases that caused some errant virus protection programs to flag this file as a virus. (For a description of this file, see the <u>installation instructions</u> or see the last item in the 12-Mar-08 release notes).
- Added two new example programs (demo\subplt.m and subplt8.m) which show how to put several plots (each with its own cursor) in a single figure window. The cursor move callback (moveCB) and the axis callback (axisCB) are used to keep the plots synchronized.

- The way that plt.chm is opened when you click on the help tag has been improved. Previously plt and any application calling plt would hang until the help window was closed. Now you can continue to use plt while the help window is open.
- The axis change (axisCB) callback is now called (as it should) when you click on the Zout or XY<-> tags in the menu box.
- Now when you right-click on the y-axis label, the menu box and all cursor objects disappear. Right-clicking again causes the objects to reappear. This is useful for making screen captures of the plots since these objects are used for data exploration and are normally just a distraction in a hardcopy. See <u>cursoring</u>.
  - The cursor('get', 'obj') call was added so that one can easily get the handles of the cursor objects.
  - The cursor('set', 'VISon') and VISoff calls were improved so that they now include control of the xstring and ystring objects, the x-cursor slider and the cursor id string.
  - A bug was fixed which caused plt to render some text improperly when the default text interpreter was changed.
  - A bug was fixed which sometimes caused the plt cursor to change shape after the data editing feature was used.
  - Added a new example program (demo\movbar.m) which shows a simple way to start and stop a moving plot as well as demonstrating the use of the xstring parameter.
  - Added the ability to set properties of the main axis from within the plt call. (Just add a "+" in front of the axis property name.

# 13Jun08

- For example **+FontSize** will set the axis font size property. For an example of how you might use this feature, see the **demo\pltvbar.m** demo program.
- Added a cursor function to call the user defined cursor callbacks. (See the <u>mainCur</u> function and <u>demo\movbar.m</u>)
- Added the ability to link the left and right axes for panning and zooming. This option is enabled by default. Use the new AxisLink parameter to turn off linking by default. Regardless

of the status of the default you may toggle the axis linking by clicking on the right-hand y-axis label. The divide sign indicates the axes are unlinked (i.e. divided).

• Improved the demo\editz.m demo program, which now displays phase as well as magnitude. It also demonstrates the use of the new AxisLink parameter

#### 11Apr08

- Previously although a plt figure could be saved (.fig), when the figure was reopened the usual plt cursor controls would not be available. In this release, plt save and plt open items have been added to the file menu. Figures saved via the plt save menu are saved with a .plt extension (although like .fig files they are actually ordinary .mat files). You can reopen the figure from a file browser dialog box by selecting the plt open menu or by typing plt open at the command prompt. Or at the command prompt you can type plt open filename.
- Fixed a bug relating to metric prefixes on the y-axis. (The scaling was improperly applied to the right hand axis where as the intention was that it should be applied only to the left hand axis.)
- Fixed a bug relating to the modification and saving of color schemes when a right hand axis was being used.
- Changed the method of rendering the right-hand axes to avoid the problem when plotting overlapping or very dense data sets.
- Changes for compatibility with Matlab ver 7.5
- Previously data editing only worked on traces on the left axis. Now it works on the right axis as well.
- Renamed the "COLORc" parameter to "COLORdef" for clarity.
- Eliminated the flicker and window movement you used to see when starting plt.
- Fixed bug in plt('datestr') and other minor bug fixes.
- Added the plt.d file to this release. This dll is only useful for compiled applications using the older Matlab compiler (ver 2.2) that shipped with Matlab 6.1. Feel free to delete this file if you are not using that compiler. (*To use this file, rename it to plt.dll*)
- Many of the example programs where modified to improve

#### 12Mar08

clarity.

- A new example program (demo\dice.m) was added that shows how to plot a function as it is as it is being computed (i.e. the plot shows the function growing in length).
- A new feature was added to the x-axis label. Now when you left/right click on this label the cursor x-axis position increments/decrements by one data index. See the <u>cursoring</u> section for details.
  - The x-axis slider now provides a more natural functionality. See the <u>cursoring</u> section for details.
  - The cursor 'maincur' call now changes the axis limits if necessary to keep the cursor in view.
  - Modified the default trace color ordering. The most significant change is that yellow is now used as the color of the 13th trace instead of the 7th. This means that yellow will be used less often (which is good since yellow doesn't normally print well).
  - Expanded the pltsq.m example to show how all the traces can be cursored at once
  - Several of the demo examples were edited to improve readability
  - Removed the 'noshift' option that was added in the previous version. (This proved to be misguided).
  - The plt('datestr') function has been modified slightly. For dates between 2000 and 2099, it now uses two digits for the year instead of four. For example "12-Jan-2007" becomes "12-Jan-07". This only applies to date string formats 0 and 1.
- 17May07
- A new method of drawing a zoom expansion box is provided ... hold both mouse buttons down, and drag. (Actually this method was previously available as an undocumented feature.)
- The demo\tasplt.m example program was updated and simplified. It now shows how to use a right hand or top axis to plot in more than one unit.

08Jul07

- The plt('slider') function has been reworked to be easier to use. Several arguments were reordered with improved defaults. See winplt.m and the new wfall.m for examples of how to use the slider object.
- A new example program (demo\wfall.m) was added to demonstrate waterfall plotting as well as extensive use of the plt('slider') function.
- Now the callback function defined by 'moveCB' is not called by events initiated from outside the figure window containing the cursor. (For example a button push that moves the cursor in

### 04Mar07

- the cursor. (For example a button push that moves the cursor in another figure window would not activate the callback.) The reason for this change was to prevent infinite loops when figure A modifies figure B's cursor and visa versa. If you do want to enable the callback for external events, insert an extra semicolon as the first character of the moveCB callback string.
- Fixed bug causing the cursor to move to the wrong y-axis position for traces on the right hand axis (only a problem when clicking in the axis area). Also improved the ability to select the desired line when several right hand traces are close together.
- Removed the 'NewLimit' parameter. (It had the same function as 'axisCB'.)
- Removed the 99 trace limitation for cases where the TraceID box is not used.
- Added the 'moveCB' and 'axisCB' parameters to plt as an alternate to the equivalent plt('cursor', ...) commands. The editz.m example was modified to take advantage of this.
- Added a new example (demo\bounce.m) showing how to use matrix arguments to create many line objects at once. Also shows how to avoid plt's 99 trace limit by disabling the TraceIDs.

# 28Jan07

- Modified the pltn example to allow displaying an unlimited number of traces.
- The colors specified by **TRACEC** are now used in a cyclical fashion to allow specifying fewer colors than traces.
- The ENAcur and DIStrace parameters are now extended if they contain fewer elements than the number of traces.

- Added **TIDcolumn** to the documentation (inadvertent omission).
- Fixed a bug causing the "Mark" function to crash when TraceIDs are disabled.
- Fixed a bug causing incorrect colors when choosing COLORC, 'default'.
- The 'Xstring' and 'Ystring' parameters where added to provide additional textual information related to the plotted data including customized cursor readouts. The ptln.m, tasplt.m, editz.m examples now use this feature to good advantage.
- The callback functions defined by 'TIDcback', 'Xstring', 'Ystring', 'moveCB', and 'axisCB' can now contain strings within strings defined using a single double quote character on either side of the string, instead of the usual and more cumbersome method using two single quote characters on each side.
- The 'pltquiv.m' example was updated to demonstrate the use of the 'TIDcback' parameter.
- Added a right click action to the XY ↔ (XYrotate) tag in the menu box.
- Improved the avg and rms cursor readout so that NaN values are ignored.
  - Added an additional plot to the circles12.m example program.
  - Added the @CID tag to the 'moveCB' cursor callback function. The weight.m and tasplt.m examples now make use of this tag.
  - The xy output from plt('cursor', cid, 'get', 'position') is now complex.
  - Improved the look of the cursor peak/valley buttons and delta cursor button by removing the rectangular dotted line around the string.
  - Fixed bug in Y/X cursor value
  - Added "HardCopy" as the fourth item of the File menu. (See "print" under "Menu box").

- Many documentation improvements and updates
- Added the capability of assigning different line property values for each trace
- A new parameter (**TIDcolumn**) was added to allow you to arrange the trace IDs in multiple columns.
- Added a new demo program (circles12.m)
- Fixed a bug that prevented assigning a color to two different traces.
- Fixed a bug that caused incorrect positioning for the "Mark" function (right hand axis).

# 26Nov06

- Added a default to the color property of the 'pop' and 'edit' objects ([1 1 .4]). Fixed the default color for the background of an open 'pop' menu to agree with the documentation.
  - Added the Fresnel function to curves.m demo so that the symbolic toolbox is not called.
  - Modified the editz.m example so that the signal processing toolbox is not required.
  - Modified the winplt.m example program so that it can run without the signal processing toolbox, however in that case it will not be able to display all the window types.
  - The way plt handles complex arguments was changed. Now including w in the argument list (where w is complex) is equivalent to including the two arguments
     "real(w), imag(w)". In earlier revisions this was only true sometimes.
  - The delta button is now highlighted whenever the delta cursor mode has been selected.
  - Left-clicking on the up/down arrow buttons moves to the next peak/valley respectively as before. Also as before, clicking on the trace resets the peak or valley finder so that the next click of the up/down arrow brings you to the largest peak or smallest valley. Now, the resetting of the peak or valley finder may also be accomplished by *right* clicking on the up/down arrow button.
  - Added files:

#### vbar.m - for displaying functions as vertical bars

ebar.m - for displaying error bars

## 04Sep06

quiv.m - for displaying vector fields (arrows)

- The functionality of quiv.m is similar to Matlab's quiver.m although unlike quiver, quiv doesn't call plot. Therefore you must call plot or plt with the data generated by quiv. In earlier versions of plt, the code in quiv.m was inside plt.p, but I extracted it to a separate function to provide more flexibility and a cleaner user interface.
- In the previous release, all the files were in the same folder. In this release a new folder named "demo" was added to contain all the example programs.
- A new program named demoplt.m was added (in the demo folder) which runs all 12 example programs in succession. Every time you press <Enter> demoplt will close the current example and start up the next one.
- A new example program named pltvbar.m was added to demonstrate the use of vbar.m and ebar.m.
- The pltquiv.m example program was modified to use the new quiv.m function.
- A new parameter ('Right') was added to specify which traces should be on the right hand axis. In earlier plt versions, only one trace was allowed on the right hand axis and it had to be the last trace. Now you may place any or all of the traces on the right axis. For example, in the argument list you could specify 'Right', [1 4:2:10 17]. Then plt would put trace numbers 1,4,6,8,10, and 17 on the right axis and all other traces on the left axis. A slight shading is used behind the Trace IDs associated with the right hand axis so you can tell at a glance which traces belong to which axis. You can also tell which axis a trace is on by the shape of its cursor (+/o for left/right axis). As with earlier revisions, a right axis is created if you include the 'LabelYR', or 'YlimR' parameters in the argument list. For clarity, you should also include the 'Right' parameter in that case, however if omitted, plt will put the last trace on the right axis.

A new parameter ('DualCur') was added to control a feature I now call a dual cursor. The usual cursor can only show the value of one of the visible traces on the plot. With a dual cursor, two such values can be shown at the same time - especially useful when displaying two tightly linked values (for example, the magnitude and phase of a complex quantity). In earlier versions of plt, a dual cursor was provided, however it was forced to be on the last trace only which was also forced to be on the right hand axis. (Moreover the dual cursor was not optional. If you wanted to use a right hand axis, you automatically got a dual cursor). As of this version, you specify a dual cursor for the Nth trace by including 'DualCur', N in the parameter list. This means that trace N will never be the active trace and so its value will never appear in the usual y value readout location (immediately to the right of the "y:" cursor label). However when you cursor any other trace, a cursor will also appear on trace N at the same x-axis location and the value of trace N will appear just to the right of the value for the active trace. The dual cursor is easily distinguished from the usual cursor by its shape - an asterisk if the dual cursored trace is on the left axis or a square if it is on the right axis.

17Aug06

- A new parameter ('TIDcback', fcn) was added to that one can define a callback function to be executed whenever the user clicks on a Trace ID (i.e. a trace is enabled or disabled).
- The winplt example program was updated to show comparisons between previous window time shapes as well as the previous frequency shapes. (In the earlier revision, only the previous frequency shapes were saved). This takes advantage of plt's new ability to put multiple traces on the right hand axis.
- Added the **Color** menu to the menu bar providing the ability to change any of the colors used by plt and optionally to make those changes permanent.
- Enhanced the ability to edit strings created with the plt('edit') command by using the left and right arrow keys.
- Enhanced the functionality of the **winplt** example program.

- Changed the default alignment of plt('edit') strings to 'center'. (Required to make the increment/decrement feature to work.)
- Set foreground (text) color of x/y cursor readout objects to white or black, whichever enhances contrast.
- Swapped execution order of MoveCB and MoveCB2 callbacks. This appears to fix the problem of mouse button up events getting lost (which was only a problem with Matlab ver 2006a or later).
- Swapped viewing order of the menu and TraceID boxes. This fixed the problem where the top menu box item was inaccessible.
- Fixed bug with '@VAL' substitution in plt('slider').
- Fixed bug in **plt('slider')** relating to the **'%d'** format.
- Fixed bug relating to accidentally redefining the exp() function
- Fixed bug relating to finding the help file.
- Added a new option plt('Options', 'Slider', ... to create a slider used for changing the cursor x position
- Added a dialog box (via a right click on the Mark tag) that allows the editing of line and cursor properties
- Fixed delta cursors so they can work between different traces
- Extended plt('ftoa', ...) to work with vectors as well as scalars and added a delimiter option. The plt('vtoa', ...) function was also enhanced.
- Added the [TexOff] specifier in the 'title' option.
- The slider now replaces '@VAL' in the slider callback with the current slider value.
- Biased 'metricp' call to favor no prefix between .1 and 9999
  - Changed order of ancillary cursor readout functions (now Avg, RMS, Y/X)
  - Added the plt('rename', s) function
  - Added the plt('cursor', 'set', 'activeLine') function
  - Enhanced the plt('cursor', 'Maincur') call by

# 11Apr06

# 25Jun06

allowing optional parameters

- Added the plt('cursor', 'init' ...) call to the documentation
- Added a new example program called weight.m

# • Changed default color order for better hardcopy results

• Fixed minor bug in the workspace plotter

# • Added the **Zout** and **Print** tags to the menu box

- Minor bug fixes and documentation updates
- Minor improvements to the pseudo slider object
  - Added support for plotting vector arrows (quiver)
  - Added the option to use cell arrays of strings in place of character arrays in argument lists
  - Extended the use of the 'AxisPos' parameter (position of TraceID)

# 18Sep05

- The workspace plotter recognizes the new TraceIDlen
   parameter
- Added plt close as an alternative to plt closefigs
- Added a new programming example (pltquiv.m) to demonstrate quiver and several other plt features
- Minor bug fixes and documentation updates
- Added the auxiliary function plt('edit') for creating edit text objects
- Added the auxiliary function plt('pop') for creating popup text objects

 Added a new programming example (curves.m) to demonstrate the use of the plt('edit') and plt('pop') commands.

• Fixed a bug in the data editing capability that caused it to work

only with large fonts selected in windows.

- Two new programming examples have been added (tasplt.m and winplt.m).
- Added the slider component plt('slider', ...). See winplt.m for an example of its use.
- Added auxiliary functions plt('ftoa'), plt('vtoa'), plt('closefigs'), plt('help'), and plt('version').
- The workspace plotter now can plot vectors contained inside structures.
- The default figure window position was changed so it doesn't overlap the start menu.
- Minor bug fixes and documentation updates
- A new programming example (editz.m) has been included to demonstrate the utility of the data editing capability.
  - Eliminated the commercial version of plt and integrated the enhanced features into the free version. The public domain version of plt now supports delta cursors and up to 99 traces (an increase from five traces).
  - The workspace plotter has been added, which allows you to interactively select the vectors in your Matlab workspace that you want to plot. This is activated by calling plt with no arguments. A new sample script (pltvar.m) was added to demonstrate this feature. It simply creates several vectors in the workspace and calls plt with no arguments.
- 14Jul04
  The cursor ID tag is now always to the left of the cursor y label. Changing the cursor ID tag to Avg or RMS is now accomplished by clicking on the cursor ID tag itself to rotate among the options. Also an additional option (Y/X) has been added to this selection, which displays the ratio of the y and x cursor values. (In delta cursor mode, this can be used to display the slope of a trace segment.)

29Mar05

- Data editing has been added. The data editing mode is entered by right-clicking on the x or y cursor edit boxes. See the documentation for additional details.
- Modified the plt5 and pltn examples so they didn't use interp (interp requires the signal processing toolbox)
- Renamed PLT.p to plt.p in the zip file (unix is case sensitive)
- Added the release notes section to the documentation
- Added a more obvious way to restore zoomed axis limits
- Fixed minor bug with the Styles argument
- Improved the Options argument and made it less cryptic
  - Supply documentation as html in addition to chm
  - Provided an easy way to use Matlab's default colors
  - Improved the organization of the help file
- First public release of plt

#### 24May04

16May04


## **Adding traces**

From the command window, the most common way to start plt is with a command such as:

### plt(x,y)

where x and y contain the data that you want to plot. Of course this is the same as you would have done before being exposed to plt, except that you would have spelled it "**plot**". This command opens a new axis in a new figure window containing just the single trace defined by the x,y parameter pair. There are many ways to call plt so that multiple traces are defined, and most of these methods are also shared with the native Matlab **plot** command. These methods are reviewed in the <u>Calling sequence</u> section.

Where **plt** and **plot** diverge more noticeably is how you add traces to a figure that has already been created. With **plot** this is done using the **hold on** and **hold off** commands that I think you will recognize as cumbersome after learning how it is done with **plt**. The more traces you add, the more you will recognize the advantage of the active legend automatically provided by plt, both for identifying the traces and controlling which traces are visible. (Soon you may begin to wonder how you managed without plt.)

Assuming a single trace was defined using the plt command mentioned above, the command:

## pltt(x1,y1,'TraceName')

will cause a new trace defined by the x1,y1 pair to appear on the plot. Also a second entry will appear in the TraceID box with the trace name specified in the call to pltt. If you are feeling lazy, you could omit the trace name in the argument list, in which case a default (incrementing) trace name will be used. The more traces you add, the more you will realize that you should have specified the trace

names all along.

When you type the plt command in the command window, space will be reserved in the TraceID box for eight additional traces (beyond those defined in the plt command itself) to be added using the pltt command. This is usually enough, but if you go beyond this limit, pltt will overwrite the trace data and traceID of the previously added trace to make room for the new trace being added. If you anticipate needing to add more than eight traces, you can allocate more space in the TraceID box using the + parameter in the plt argument list. (The + is the last parameter described in the Trace properties section.)

Usually you will probably find it most convenient to add just one trace at a time, however pltt also provides a few ways to add more than one trace at a time. (This is especially useful inside script files). For example, both of these commands:

```
pltt(x, {y1 y2 y3}, {'new1' 'new2' 'new3'});
pltt(x, [y1; y2; y3], {'new1' 'new2' 'new3'});
```

are equivalent to the three separate commands:

```
pltt(x,y1, 'new1');
pltt(x,y2, 'new2');
pltt(x,y3, 'new3');
```

Note that in the 2nd form above, it was assumed that y1,y2,y3 were row vectors. (If they were column vectors they would need to be delimited with spaces or commas in place of the semicolons.)

And one last form to consider for when you are adding traces with differing x data. The command:

```
pltt({x1 x2 x3}, {y1 y2 y3}, {'new1' 'new2' 'new3'});
```

is equivalent to these three commands:

pltt(x1,y1, 'new1');
pltt(x2,y2, 'new2');

pltt(x3,y3,'new3');



## **Selecting traces**

📣 plt		
Line 1		
Line 2		
Line 3		
Line 4		
Line 5		

If more than one y-vector argument was given to plt, a trace selection box will appear in the upper left corner of the plt window. Note: when using sub-plots, only the main plot (lower left) includes a *trace selection box*. (Note: sometimes I call this the *TraceID box*).

In the example shown to the left, plt has been given five y-vectors to plot. The name of each trace and the colors used to plot them are

also given. Usually you should give each trace a more informative name, but in this case no trace names were specified on the command line, so plt just named each trace with a number. The trace IDs associated with traces 2 and 3 are shown in an italic normal weight font, indicating that they are currently disabled (not visible). The other three trace IDs are shown in a bold upright font, indicating that those traces are enabled in the plot area.

Below is a list of the ways you can control which traces are displayed. If you are plotting just a few traces, the first bullet should cover all that you need to know. However since plt can plot up to 99 traces, the remaining tricks may come in handy:

- When you left-click on one of the trace names in the trace selection box, the associated trace is toggled on or off.
- To view a single trace all by itself, right-click on the trace name of interest. That trace will be enabled and all the others will be disabled.
- Double click on any of the trace names to enable all traces at once.

The default trace color order used by plt (shown in this figure) is quite different from Matlab's usual default in that it allows you to distinguish many traces based on color alone. Since the color used for the trace IDs match the color of the corresponding trace, you can easily identify each trace by name for plots with a

ID01	ID34	ID67
1002	035	ID68
1003	1036	1069
1004		ID70
ID05	ID38	ID71
ID06	1039	ID72
1007	ID40	D73
1008	ID41	D74
1009	ID42	D75
1010	ID/13	1076
ID11	D44	
1012	ID45	ID78
ID13	ID46	1079
1014	ID47	1080
ID15	ID48	ID81
ID16	ID49	1082
1017	1050	10.02
1018	ID51	084
1019	052	ID85
ID20	1053	ID86
1021	D54	D87
1022	1055	1081
ID23	1056	1089
1024	1057	ID90
1025	1058	ID91
1026	1059	D92
027	ID60	ID93
1028	ID61	ID94
1029	ID62	
1030	D63	1096
1031	ID64	10.97
ID32	ID65	
1033	ID66	
1000	15,000	

dozen or so traces and perhaps even many times that amount depending on the acuity of your color perception. Read more about this in the Default colors section If you have any doubt about the name of a particular trace, simply click on the trace, and its trace ID will appear in the "cursor ID" tag just to the left of the y-axis cursor readout. (See "Cursoring" on the next page). Another way to verify the

name of a trace is to click on the name of the trace in the trace ID box. As mentioned above, the corresponding trace will become invisible, and then restored when you click on the trace ID a second time. To see the list of RGB triples used to generate this default color order, see the description of the <u>TRACEc</u> parameter. Note that only the colors for traces 1 thru 40 are defined by this array. The defaults for lines 41 to 80 are the same as the colors listed above for lines 1 to 40 except that they are 26% dimmer. The defaults for the lines 81 to 99 are again 26% dimmer than the trace colors for lines 41 to 59.

If the plot uses a right hand axis, plt indicates which traces are plotted on the right hand axis by adding some shading to the background of the trace ID. For instance, in this example, traces 1,3,4, and 5 are plotted on the right hand axis. The use of italics to indicate inactive traces, and the methods for enabling/disabling traces are the same no matter which axes are used. (Traces 2 and 4 are disabled in this example.)





If the **TraceMK** argument was used, then the line types are also shown in the trace selection box. An example of this is shown here which used **'TraceMK'**, . 6 in the plt argument list. The .6 indicates that the first 60% of the width is used for the trace names and the remaining width is used to show the line types. See <u>Trace Properties</u> for more detail on the TraceMK parameter.



When the line types are included in the trace selection box, you may enable/disable traces by clicking on the trace name as described above, or by clicking on the line type to the right of the trace name. In this example traces 3,4 and 5 have been disabled. The trace names for the disabled traces are shown in italics as usual and the line types are also grayed out (actually blue/green) to make it clear that these traces are not currently

visible in the plot area.

🥠р	lt	
×		×
<b>G</b>		-8
<u>ا</u>	-0	

Since left or right clicking on the line types serves the same purpose as left/right clicking on the trace names, you can dispense with the trace names altogether if you like. (This mode is selected by specifying assigning less than 25% of the width to the trace names, i.e. a TraceMK parameter of less than .25). In the example to the left, the TraceMK parameter was 0.1 and note that the first trace has been disabled.



## **Zooming and panning**

### Panning

The simplest way to pan the x or y axis is to click on one of the axis tick labels (actually anywhere outside the plot area will work) and drag it until the part of the display you wish to view is visible. Sometimes you may want to pan both the x and y axes at the same time. Instead of doing separate pans on each axis you can do both at the same time by clicking anywhere in the plot area (but NOT on any of the traces) and dragging that point until the desired view is achieved. Yet another way to pan the x axis is to use the optional x-axis cursor slider that is described in the next section (cursoring) One panning issue you should be aware of is that its performance will suffer when plotting very large arrays. For example, try the following command:

#### plt(humps((1:1e6)/1e6)'\*(1:4));

This will plot 4 traces each of which contains a million points. (These vectors are far bigger than what you will usually want to plot.) The display update rate while panning this plot on my 2011 era desktop computer is about 3 times per second, which is noticeably jerky but certainly useable. However if you want to plot even more data than this it may make sense to decimate it before plotting. Your eye won't know the difference by the time you are plotting more than a couple of hundred points per trace, so you won't really be missing anything unless you zoom in dramatically.

### Zooming

Try the same thing as with panning, except drag with the right mouse button instead of the left. You will find that dragging towards the origin compresses the axis (for zooming out) and dragging away from the origin expands the axis (for zooming in). As with panning you can zoom both axes at once by a right click and drag in the plot area. (Unlike panning you don't have to worry about whether you are on a trace or not. The same thing will happen in either case.)

Often to get the desired view requires two mouse movements. The first, with a right click and drag to expand or contract the axis (or axes) and the second, with a left click, to re-center the display. You may find that this is the most convenient method, or perhaps you will like one of the seven other methods described below.

## The expansion box

If the portion of the graph that you want to zoom in on is completely visible on the graph, the fastest way to display the desired area is to draw an expansion box. This is the way you will likely use most often, so you should try all four ways to draw the expansion box that are listed below to see which ones you like best.

1. Position the mouse in the plot area over one corner of the area you wish to zoom in on. Then click both mouse buttons at the same time, holding them both down while dragging the mouse towards the

opposite corner of the desired zoom area. A yellow box will be drawn, which will be stretched or contracted as you drag the mouse around. When you let go of the mouse, the display will look similar to the picture to the left.

- 2. An alternate method which you may find easier is double click the left mouse button, but don't release the button after the second click. Hold the mouse button down while you drag to create the expansion box.
- 3. A method which requires less coordination is to press and hold the keyboard shift key, then click and drag with the mouse until the expansion box is the size you want.
- 4. And finally you can left-click the grey "x" label in front of the x-cursor edit box. (Actually using the grey "y" label in front of the y-cursor edit box does exactly the same thing.) This draws the expansion box covering the exact same area as the current axis limits, and then zooms the display out by about 20% so that you can see the expansion box. This method makes more sense when you which to make small changes to the x or y axis limits or when you are planning to type in the new limits numerically.



If you are happy with the expansion box you have drawn, left-click the mouse anywhere in the plot area (or even outside the plot area if you avoid the edit boxes) and the display will expand to show only the data inside the expansion box. If you are not happy with the expansion box, you can modify it using one of the methods mentioned below ... or simply right-click anywhere to remove the expansion box and start again.

If plt was called with the MotionZoom or the MotionZup parameter, the function specified with that parameter can cause additional text, plots or other visual effects to appear and be modified as you adjust the size of the expansion box. (See Mouse Motion Functions)

## Accepting or cancelling the limits indicated by the expansion box

After an expansion box is drawn using any of the four methods described above, both the x and y-cursor edit boxes double up and contain the limits of the expansion box (as shown in this figure). To accept the current limits shown, simply LEFT click



anywhere inside the plot area. To remove the expansion box keeping the axis limits the way they were, simply right-click anywhere inside the plot area.

You may want to modify the expansion box size or position before accepting it. Several methods of doing this are described below.

## Adjusting the expansion box

The most precise way of setting the expansion box limits is to simply type them in. For example, suppose you want to change the x-limits shown in the figure above (7.1866 to 7.9650) to the values 7.1 to 7.9. Simply highlight the lower limit x limit (7.1866) by dragging the mouse over it and then type in the desired value (7.1). Then press "tab" which will accept that value and automatically highlight the next value (upper x-limit). Then after entering the upper limit, press tab again to highlight the y-axis lower limit ... or if you don't want to edit the y limits as well, hit enter instead of tab. As soon as you hit enter or tab each time you will see the edit box change in the plot area to reflect the entered values. Note that the limits are shown in increasing order, however you are not restricted by that convention. (i.e. entering the limits 4,3 draws the same expansion box as if you entered 3,4). Although this method is by far the most precise way to adjust the expansion box, it is usually more convenient to do this using the mouse as described below:

### 1. Adjusting the expansion box size

Simply click and drag on any of the four corners of the expansion box. The corner you clicked on will follow the mouse while the diagonally opposite corner will remain fixed. Note that the mouse behaves in the same way as when the expansion box was first drawn.

2. Adjusting the expansion box position (preserving its size) Click on the midpoint of any edge of the expansion box and simply drag the expansion box to its desired location. The expansion box size does not change during this operation.

When adjusting the expansion box size or position in this way, you should click reasonably close to the corners or the edge midpoints. If you click too far away from these points then as mentioned above, the click indicates that the limits indicated by the expansion box should be accepted as the new axis limits, and of course the expansion box is cleared after the new limits are set.

#### Alternate method

In earlier versions of plt (before the above method was devised) a different method was used to adjust the expansion box with the mouse. Although it allows you to adjust a single edge at a time (as well as moving the position while preserving size), you will probably find this older method less natural. Neverthe-less I did not remove this method in case you became used to it with older versions of plt. Newer users will probably stick to the method described above, but for completeness the older method is described below:

• Right-click anywhere near the middle of the lower x limit (the 7.1866 number in this figure) but don't release the mouse button. As you are holding the mouse button down, drag the mouse to the left or the right. (You don't have to remain inside the edit box, or even inside the figure window for that matter). As you drag to the left you will see the 7.1866 number decreasing and also the left edge of the expansion box will move to the left.

As you drag the mouse to the right of center, number will start increasing and the expansion box moves accordingly. The farther you drag the mouse from the center of the edit box, the faster the left edge of the zoom box moves. Any vertical movement of the mouse in this situation is ignored.

- Right-click anywhere near the middle of the upper x limit (the 7.9650 number in this figure), hold the mouse down and drag left or right. This time the right side of the edit box moves in a similar manner to that described above.
- Right-click near the middle of the lower y limit (the 2.2735 number) to adjust the lower edge of the expansion box in a similar manner except now only the vertical movement of the mouse matters. Any horizontal movement is ignored.
- Right-click near the middle of upper y limit (the 5.4595 number) to adjust the upper edge of the expansion box. Again, any horizontal movement is ignored.
- Note that the mouse methods for adjusting the four edges described so far alter the size of the expansion box. There is one final method to describe that doesn't change the size of the expansion box, but only its position. Right-click on the left or right edge of any of the four edit boxes, leaving the mouse button down while dragging the mouse as before. (You might have to be pretty close to one of these edges for this to work.) This time, as you drag, the position of the expansion box moves in the same direction as the mouse offset from the original clicked position. The farther the mouse is moved from this original position the faster the expansion box position changes. Unlike the previous four situations, here both the horizontal and vertical movements of the mouse have an effect, so you can even move the box's position diagonally if you choose. Note that the behavior is the same no matter which of the four edit boxes you use to initiate the action. The size of the expansion box will not change during this movement with one exception. If you bump the expansion box into one of the edges the box becomes smaller because the trailing edge will continue to move even after the leading edge has hit the wall.

When using the alternate mouse methods, sometimes the edit box may be too close to a screen edge to allow a reasonably fast movement of the expansion box because the mouse travel is limited. In that situation you may want to move the figure slightly farther away from the edge of the screen.

## **Auto scaling**

If plt is called without any 'xlim' or 'ylim' arguments, both axes are initially auto-scaled to show the entire data range. At any later time you can auto-scale the x-axis by right-clicking on the grey "x" label in front of the xcursor edit box. Right clicking on the grey "y" label is similar for auto-scaling the y-axis, although there is one difference. The difference is that the y-axis is scaled to insure that the data associated with the active trace is visible. There is an alternate way to auto-scale that picks display limits to insure that all the traces are visible instead of just the active trace. (See "Expansion history" below).

## **Expansion history**

Whenever you change the x or y axis limits by any of the above methods, the previous limits are stored in a expansion history list. You can cycle through these stored limits by left-clicking on the  $XY \leftrightarrow$  tag in the menu box. (See "Menu box" below). This list is 4 elements deep, so when you zoom or pan the fifth time the oldest display limits fall off the bottom of the stack. Assuming the expansion history list is full (which is usual) clicking on the  $XY \leftrightarrow$  tag four times in a row will show you the last four display limits. On the fifth click, plt will auto-scale both the x and y axes in a way that insures that *all* the data for *all* traces falls inside the display area. On the sixth click, plt goes back to using the axis limits stored in the expansion history list. Although you can auto-scale by clicking on the  $XY \leftrightarrow$  tag a suitable number of times that can be cumbersome since you usually don't know where you are in the rotation. For this reason a faster way to auto-scale is provided ... simply *right* click once on the  $XY \leftrightarrow$  tag.

## Doubling or halving the display area

Left-clicking the **Zout** tag in the menu box (see "Menu box" below) expands each axis by 40% which increases the display area by 1.4<sup>2</sup> (1.96) i.e. approximately doubling the display area. Right-clicking zooms in, halving the display area. In both directions the center point of the display remains in the center after the zoom operation.

## xView slider

The black horizontal bar with the short gray segment that appears above the plot is called the xView slider. It provides yet another way of panning and zooming



particularly useful when you want to view a small segment of a long data set. The whole bar represents the entire data set and the gray segment represents the portion of the data currently in view. If 10% of the data is currently in view, then the length of the gray segment will be 1/10 the length of the whole bar. Similarly the position of the gray segment within the bar represents the position of the data relative to the whole data set.

Properties
multiCursor
xView slider
Cancel
Range
Range↔
Rangeî↓
Insert
Insert↔
Insert1↓
Modify
Modify↔
Modifv1↓

To bring up the xView slider, first right-click on the Ycursor edit box. This will bring up the Yedit popup menu shown here. Then select the third item in this menu (xView slider) and the slider will appear. This is a toggle, so selecting it again will make the xView slider disappear.

If you wanted to the xView slider to appear when your program starts up, you can include the string xView in the 'Options' parameter. Also you can enable or disable the xView slider from the command line or in a program with the command

plt click Yedit 3; or its functional form
plt('click', 'Yedit', 3);

Moving the gray segment left or right is as easy and natural as you would expect. Simply click on the gray segment and drag it left or right. The plot underneath will update as you are sliding allowing you to easily search for the data portion that interests you. You can also make the gray segment larger so that a larger portion of the data is displayed. To do this simply click in the black area to the left of the gray segment and the left edge of the gray segment will immediately be extended to the point where you clicked. (Similar for the right edge of course.) But notice that this method won't work if you want to make the gray segment smaller. So how do we do it? Simple, just click in the black area, hold down the mouse and drag. The edge that you selected will follow the mouse allowing you to place it wherever you want. (An alternate method of making the gray segment smaller is to right-click inside the gray segment, but the first method I mentioned is usually easier.) And finally there is one more trick you can do with the gray segment. Double clicking on it expands the gray segment to fill the entire black bar (i.e. it resets the x-axis limits to cover the full extent of the x data). This is somewhat similar to right-clicking on the menubox  $XY \leftrightarrow$  tag except that the XYrotate tag effects both the x and y axis where as the xView slider never effects the y axis. (Double clicking on the gray segment a second time undoes the effect of the first double click.)

Notice that when the x-axis is zoomed or panned by any of the other methods provided, the xView slider will automatically be updated so that the gray segment properly represents the visible portion of the data.

The appearance of the xView cursor is probably suitable for most situations, but you can modify its appearance by using the xvProps figure application data. This is best illustrated with an example. Suppose we follow the call to plt with the expression:

# setappdata(gcf,'xvProps', ... {'color' 'red' '+color' 'blue' '+' [0 -.01 0 .02]});

The cell array consists of property name/value pairs. If the property name does not have the "+" prefix the property is applied to the short gray segment, so the first property pair above changes the gray segment into a red segment. If the property name does include a "+" prefix then the property is applied to the long horizontal black bar (which actually is an axis). So the second property pair changes the black bar into a blue one. Any axis property name may be used. The last property pair is a special case since it has the prefix without a property name. The meaning of this special case is that the value specified is to be added to current position value for the horizontal bar (axis). So what this example does is to move the (blue) horizontal bar down by 1% of the figure height and to make the bar thicker by 2% of the figure height. (Note that you could also specify the position in absolute terms be replacing the '+' with '+pos')



## **Right-hand axis**

## Enabling

You specify which traces should appear on the right-hand axis with the 'Right' parameter. For example if you included 'Right', [1 4:2:10 17] in the parameter list, then plt would put trace numbers 1,4,6,8,10, and 17 on the right axis and all other traces on the left axis. A slight shading is used behind the Trace IDs associated with the right hand axis so you can tell at a glance which traces belong to which axis. (You may disable that shading if you prefer. To see how to do that, look at the details of the TraceID parameter at Trace properties.) You can also tell which axis a trace is on by the shape of its cursor ('+' for left axis and 'o' for the right axis). You can optionally specify the label or the limits for the right hand axis by using the 'LabelYR' or 'YlimR' parameters respectively or by using cell array inputs with the 'LabelY' or 'Ylim' parameters. Note that if you enabled metric prefixes on the y-axis, this applies only to the left hand axis. The right hand axis uses standard scaling.

## Cursoring

Cursoring the traces on the right or left hand axes is identical except for the shape of the cursor - a '+' for traces on the left axis and a 'o' for traces on the right axis. Different cursor shapes are used for the <u>dual cursor</u>.

## Panning and zooming

The following controls affect both the main and auxiliary axes simultaneously:

- LinX/LogX (menu box)
- LinY/LogY (menu box)
- Panning the x-axis

• Zooming the x-axis



Panning and zooming the y-axis (which includes zooming with an expansion box and left/right clicking on the "zout" tag) is also normally done simultaneously on the right and left hand axes. This is called the "linked" mode. Sometimes however it is more convenient to adjust the left and right axes separately (i.e "unlinked"). To unlink the axes, simply click on the right hand axis label. The label will then appear between two divide signs as shown in this picture (i.e. the axes are "divided"). Click on the label again and the divide signs will disappear indicating that the axes are again linked.

Normally the axes are linked when plt initializes. However if you want plt to start in unlinked mode, include the parameter 'AxisLink', 0 in the plt argument list. Including 'AxisLink', 1 tells plt to start in linked mode, although you will rarely do that since linked mode is the default anyway.

Regardless of the linked/unlinked status you can pan or zoom the right hand axis by right or left clicking on or near one of the right-hand axis tick labels (i.e. the 20,40,60,80, or 100 in this picture) and dragging them to the desired position. (As before left click/drag is for panning and right click/drag is for zooming).



## Menu box

Help LinX LinY Grid Menu Mark Zout XY↔

By default, the menu box contains the 8 items shown in this figure. Some or all of these items may be missing if they were specifically excluded by using the 'Options' argument in the calling sequence. The name of each menu box items are chosen to identify the action when you left click on the item. The action corresponding to a right click are less obvious, but a quick look at the plt menu (see below) will remind you what these actions are.

See **pltquiv.m** for an example of how you can change one of the menu box items to perform an alternate function.

Details about each menu box item follows:

## Help

On Windows systems when you left or right click on the the Help tag, plt will display the file plt.chm. On Unix and other systems the browser will be opened to display the file plt.htm (since these systems don't support the chm file format). The left click behavior may be modified by including the 'HelpFile' parameter on the command line and the right click behavior may be modified by including the 'HelpFileR' parameter. Those parameters are described here.

## LinX

Left-clicking this tag changes the x-axis scale from linear to logarithmic. The name of the tag itself also toggles between LinX and LogX so that it the tag name always matches the current x-axis scaling type. Left-clicking again toggles it back to LinX.

Right-clicking this tag swaps the x and y data for all the traces which has the effect of displaying the inverse function of the original display. This swapping works best when only a single axis is being used. For multiple axes (i.e. with a right hand axis or with subplots) the effect might not quite what you expect.

## LinY

The y-axis scale changes from linear to logarithmic. This tag also changes to LogY so that it always matches the current scaling of the y-axis. Clicking again toggles it back to LinY. Right-clicking on this tag opens the plt HardCopy dialog box. See the description of the HardCopy dialog box below (under the Print menu item).

## Grid

Left-clicking on the Grid tag alternates between no grids (tick marks only) and full grids on both x and y axes. By default, the grids are solid dim grey lines. (This may be modified by the GRIDc and GridStyle parameters.) Grid lines with high contrast colors and brightness (such as what you get with the native Matlab plot command) makes it more difficult to observe the main data traces.

Assuming the grid style has not been modified with the GRIDc or GridStyle parameters, right-clicking on the Grid tag will alternate between the following two grid modes:

Color = [.13 .13 .13], LineStyle = '-'
Color = [.26 .26 .26], LineStyle = ':'

## Zout

Each left click of the Zout (Zoom out) tag expands the x and y axes by 40% (20% at each end). This approximately doubles the area of the Cartesian plane displayed within the axis limits. ( $1.4^2 = 1.96$ ). Right-clicking on the Zout reverses the effect of a left click (i.e. zooms in).

## XY ↔

Each click on the  $XY \leftrightarrow (XYrotate)$  tag cycles the x and y axis limits to the next

display expansion stored in a history list. (The axis limits from the last four zooms or pans are saved in this list). After all the display limits in the history list are used, the next click autoscales both axes so that all the data is displayed. The next click again uses the first display expansion in the history list. Right-clicking on this tag skips the rotation through the history list and directly autoscales the axes to show the full x and y extent of the trace data.

## Print

Note that the **Print** tag does not appear in the menu box by default. It will only appear if enabled by an '**Options**' argument in the calling sequence. This is done to reduce clutter and is justified since this is not a commonly used dialog and because you may also access this by right-clicking on the LinY tag (mentioned above) as well a from the Hardcopy selection in the plt menu of the menu bar. (If you don't see the menu bar, click on the Menu tag in the menu box to make the menu bar visible.)



This is what the hardcopy dialog looks like when opened. The primary use of this dialog box is to create BMP bitmapped images of the plt window. You will see from the popup menu, that you

can select other graphics formats as well, although not all of them are well tested. (This image shows the dialog setup for making a windows meta file format.) For all file types except BMP, the colors are inverted if the background is dark. On most printers this makes the copy far more readable and saves large amounts of toner.

You can also print directly to the default windows printing device from the HardCopy dialog, although an easier (and possibly more reliable) way of doing this is to select "print" from the File menu. And for a bitmapped image, yet another method is to use the screen capture facility (via the PrintScreen key on Windows based systems). Before capturing your bit map you may want to right-click on the y-axis label to hide the menu box and cursor objects so these objects won't distract from the basic plot data. Right-clicking on the y-axis label a second time re-enables the cursor objects.

## Mark



When you left-click on the Mark tag, a square marker is placed at the current cursor location and a marker string is added to the plot containing the x and y coordinates of the cursor location. The text string will be the same color as the active trace. If the text label is not positioned where you want it, click on

the label and drag it to the location you want.

📣 Edit Marker	_ 🗆 🗙	
Marker properties:	String properties:	
Marker	String	
square	(7.675, 2.95902)	

If you right-click on the marker string (which is (7.675, 2.95902) in this example), a new dialog box will appear which allows you to change any or all of the properties of both the trace

marker (with the left popup and edit box) and the marker string (with the right popup and edit box). The two pictures below show the Marker and String properties respectively that you can edit.

Note that you can have many of these Edit Marker dialog boxes open at same time - as many as one per text string (or even more, although there is probably little benefit to that). All these dialog boxes will be deleted automatically if the main plot window is deleted.

📣 Edit Marker	_ 🗆 🗙
Marker properties:	String properties:
Marker	String
Delete	(7.675.2.95902)
Color	(1.013, 2.03002)
LineStyle	
LineWidth	
Marker	
MarkerSize	
Xdata	
Ydata	
Zdata	

Once you select one of the 8 marker properties, or one of the 10 string properties, the current value of the property appears in the respective edit box. To change the property value, click in the edit box, and type in a new value.

Note that both the Marker and the String popups have **delete** as one of the options. This is useful if you

want to add a marker without a string or a string without a marker. You may delete all the markers you have added this way by selecting **delete** and then typing **all** into the edit box below the popup. The same trick works for deleting all the strings added to the figure via the Mark tag. If you want to delete all the markers and all the strings at once choose the "Delete cursor annotations" selection in the plt menu of the menu bar. (See below, and also note the mouse shortcut for this function.)

📣 Edit Marker		
Marker properties:	String properties:	
Marker	String	
square	Delete Color	
,	FontAngle	
	FontName	
	FontSize	
	FontWeight	
	HorizontalAlign	
	Position	
	Rotation	
	String	
	VerticalAlign	

Note that when you select the **color** property, the property value is a set of three numbers corresponding to red, green, and blue respectively. Each number is an intensity value and must range from zero (off) to 1.0 (full intensity). You may change the color by entering the desired rgb values. When you press <Enter> the new value will be accepted and you will see the marker or string change to the new color. However since it is often difficult to predict exactly what these

colors look like, plt provides an easier way to select new colors. Instead of leftclicking on the rgb triple, use a right click. A new color selection box will appear. The use of the color selection box (also called a ColorPick pseudo object) appears below.

### Menu

Left-clicking on this tag toggles the menu bar on or off. (See the description of the menu bar below). Note that the initial state of the menu bar is off unless the (unless 'options', 'Menu' appears in the argument list).

📣 plt	5			_ 🗆 🗙	<
sav	e			fontsize: 9 💌	
185	5.400000	4.875644	3.941501	1.916746	•
186	5.425000	4.903466	3.891/63	1.940884	
107	5.450000	4.951254	3,042033	1.900294	
189	5 500000	4.956959	3 748815	2 020721	
190	5.525000	5.013723	3,704393	2.049631	
191	5.550000	5.040690	3,662101	2.079600	
192	5.575000	5.067286	3.622209	2.110574	
193	5.600000	5.093448	3.584957	2.142502	
194	5.625000	5.119115	3.550548	2.175329	
195	5.650000	5.144228	3.519149	2.209004	
196	5.675000	5.168728	3.490889	2.243473	
197	5.700000	5.192561	3.465857	2.278687	
198	5.725000	5.215673	3.444102	2.314593	
199	5.750000	5.238014	3.425632	2.351143	
index	X	Line 1	Line 2	Line 3	
700		VVVVVV	3 449 445	2 200205	
200	5.775000	5.259534	3.410415	2.388286	
201	5.800000	5.280188	3.398117	2.425975	
202	5.025000	5.299955	2,200291	2.404102	
203	5.875000	5 336535	3.300002	2.502000	
204	5 900000	5 353320	3 372214	2.541045	
205	5.925000	5.369050	3.370770	2.620975	
207	5,950000	5.383697	3,370983	2,660977	
208	5.975000	5.397234	3.372661	2.701213	
209	6.000000	5.409639	3.375615	2.741645	
210	6.025000	5.420891	3.379655	2.782234	
211	6.050000	5.430974	3.384595	2.822943	
212	6.075000	5.439874	3.390257	2.863735	•

Right-clicking on the Menu tag opens up a new window similar to this one shown called the Cursor Data Window . This window shows the x and y values of all visible plot traces. The down arrows (VVVVV) highlighted in the middle of this window indicate that the cursor is currently pointing to the 200th data element of the trace whose trace ID is "Line 1". As you move the cursor around (by any of the many methods) the cursor data window will automatically be updated so that arrows always point to the current cursor location. The cursor index will always be shown near the middle of the

window but you can use the scrollbar on the right side of the figure to view any of the data values that appear on the plot.

The first column heading is always "index" and the index column will contain all the integers between 1 and n, where n is the data length of the longest visible trace. The second column heading is always "X" which indicates that this column contains the Xdata values of the first visible trace. The 3rd column will contain the y values of the first visible trace and the column heading will be the traceID. The 4th column will again have a heading of "X" and will contain the x data for the 2nd visible trace except (as in this example) when the x values are the same as for the previous trace in which case the X column is omitted. In this example all 3 traces had the same x vector so only one column is needed for it. However if the 3 traces each had different x vectors then the column headings would have been *"index,X,Line1,X,Line2,X,Line3"*.

If the characters are two big or small for your taste you can adjust the size with the fontsize popup. Click the save button to create a text file that contains the

exact text that appears in the list box (from index 1 all the way to the end of each plotted array). The column headings will appear at the very top of the text file.

Note that if you are using subplots, some of the column headings may be blank since the subplot traces do not use TraceIDs. However if a subplot has a y-axis label then that label will be used as column heading for the trace inside the subplot.

## Menu Bar



Usually you will select the desired menu with the mouse. However you may also use the keyboard. The underlines shown in this figure and in the drop down menus below only appear when you press the ALT key. When you press the ALT key followed by one of the underlined characters, the respective dropdown menu will appear. You can then select one of the dropdown menu items with the mouse or by pressing one of the underlined characters in the dropdown menu.

One of the most useful functions of the menu bar is **Print** (the last item under the **File** menu). This is probably the easiest and most reliable way to make a hardcopy of the plt window.

As you can see from the figure to the right, plt adds the following two items to the File menu:

 plt save saves the current figure so that that it can be opened later (see plt open below). A dialog box opens allowing you to choose the name of the file. A .plt extension is used for these files although in fact they are ordinary .mat files. This menu item is equivalent to typing plt save at the Matlab command prompt. Also at the command



prompt, you may type plt save filename which avoids the file dialog box by specifying the file directly and of course the functional form plt('save', 'filename') works also.

plt open opens a dialog box that allows you to select a .plt figure file that was saved with the plt save menu. The new window should look and behave the same as the original plt window. (Note that if the original plt window was created with a function that provided additional plotting features to the plt window, those features will not be available after opening the figure.) This menu item is equivalent to typing plt open at the Matlab command prompt. Also at the command prompt, you may type plt open filename which avoids the file dialog box by specifying the file directly. A new window is opened containing the data that was saved.

N	Help	plt
		Edit line (e) Rclick Mark
		Edit all lines (a) Delta+Rclick Mark
		Edit figure colors (f) Rclick Properties in Ypopup
		Save figure colors ( <u>s</u> )
		Cursor Data Window (c) Relick Menu
		Swap X/Y axes ( <u>w</u> ) <i>Rclick LinX</i>
		Hardcopy ( <u>h</u> ) Relick LinY
		Toggle line smoothing ( <u>t</u> ) <i>Rclick</i> "o"
		Delete cursor annotations ( <u>d</u> ) Delta+Rclick "o"
		Set dual cursor ( <u>u</u> )
		Toggle Reposition mode (r) Rclick Delta
1	$\frown$	Reposition Grid size (g) Delta+Rclick Delta
X	$\langle \rangle$	

The last menu item (plt) is unique to plt. When you click on this menu item you will see these twelve submenus.

The accelerator keys for selecting one of these submenus are shown in parentheses. For example to select the "Save figure colors" submenu using the keyboard, you would first press ALT key followed by the p key (to select the plt menu) and finally press the s key to select the "Save figure colors" submenu. (You don't

have to release the ALT key before the p key is pressed if you prefer.)

Note that all but three of these submenus have some blue text after them. These are directions for selecting the submenu action without using the menu bar (which may be easier, especially when the menu bar is hidden). For example, the first submenu (Edit line) contains the string *Rclick Mark*. This means that you can invoke the Edit line function by right-clicking on the Mark tag inside the menubox. The next submenu (Edit all lines) contains the string *Delta+Rclick Mark*. What this means is that you can invoke this menu by first left-clicking on

the Delta ( $\Delta$ ) button and then right-clicking on the Mark button. The third submenu (Edit figure colors) contains the string *Rclick Properties in Ypopup*. This means that you should right-click on the Properties selection that appears in the Yedit popup (that opens when you right-click on the Ycursor edit box). Of course you won't remember these shortcuts unless you use them often, but you can always use this menu as a reminder.

These twelve submenus are described in order below:

## 1.) Edit line Rclick Mark or Lclick Properties in Yedit popup



To change the color or other property of a trace or of its associated cursor, first click on the trace that you want to modify (i.e. make the trace active) and then click on the Edit line

submenu. You will see a new figure similar to the this one.

The left side of the Edit Line dialog box controls the properties of the active trace and the right side controls the properties of the cursor attached to that trace. The properties that appear in both these popups are the same as those under "Marker properties" in the Edit Marker dialog box shown above. Note that one can edit the data plotted by selecting and editing the Xdata or Ydata line properties. This works well for simple traces with less than a few dozen data elements. For longer sequences you will be better off using the data editing technique described in the <u>data editing</u> section. Note that (for example) if you modify the Ydata property by removing or adding data elements that the line will disappear until you also modify the Xdata property by removing or adding the same number of points. This is because the line object can't be rendered unless the lengths of the Xdata and Ydata properties are the same.

## 2.) Edit all lines *Delta+Rclick Mark*



When you select the Edit all lines submenu a dialog box such as this will appear. When you change the popup menu to select a new property, the edit box will be updated to show the current value of that property for the active trace just as before. However if you then change the property value in the edit box, this property will get changed for all the traces on the plot, not just the active trace. This is probably not appropriate for the color property, but it may be useful for some of the other trace properties, such as linewidth. You can also use this dialog box to make all the cursors larger or a different shape for instance.

## 3.) Edit figure colors Rclick Properties in Yedit popup

🛃 Edit figu	re colors	_ 🗆 🗙
	Figure background	
	.25 .15 .15	

lines dialog boxes described above.

Edit figure colors
Figure background
Figure background
Plot background
Plot background
Axis color
Axis color (right)
Axis labels
Delta cursor
Grid color
Grid style

dialog box will appear which allows you to change all of the figure colors which are not accessible from the edit

When you select this submenu this

This shows the eight items that you can modify from this dialog box. After selecting one of these items, the current color of that item appears in the edit box below the popup as a set of three intensity numbers corresponding to red, green, and blue respectively. You may change the color by left-clicking the edit box and

entering the desired rgb values or by right-clicking the edit box which will bring up the color selection box described at the bottom of this page.

Note that the three property editing windows shown above for the first three submenus may also be opened from the "Properties" selection that appears when you right-click on the Yedit cursor. (See <u>Data editing</u>.)

## 4. Save figure colors

Changing colors inside the plt figure using the color selection box is not permanent (i.e. those colors will be forgotten once the application is closed). However you can make the changes permanent by selecting this submenu which will save the current colors to the file (which we call a "color selection file"). You will find the rules that plt uses to determine the color file name in the description of the 'ColorFile' parameter here.

When plt starts, the color for each screen element is determined as follows:

- If a color file is found, the color for all screen elements will be determined by the file contents.
- If no color file exists, the colors for particular screen elements are determined by the color parameters included in the plt function call. These parameters are described in the <u>Colors</u> section.
- If such a parameter is not included in the function call, then the color for that particular screen element will be specified by the plt default color scheme.

These color selection files are in a consistent format so a color file generated in one application can be used in another application by renaming the color file, or by using the name of the desired color file explicitly with the 'ColorFile' parameter. More detailed instructions about how to modify a color selection file are given in the <u>Default colors</u> section.

## 5.) Cursor Data Window Relick Menu

This submenu opens a cursor data window which is described above under the right click action of the Menubox Menu tag.

## 6.) Swap X/Y axes *Rclick LinX*

This submenu performs the action described above under the right click action of the Menubox LinX tag.

## 7.) Hardcopy Relick LinY

This submenu menu opens a dialog box used for printing and creating screen captures of plt figures. The use of this dialog box is described above in the description of the menubox Print tag.

## 8.) Toggle line smoothing *Rclick* "o"

This submenu toggles the line smoothing property of all plot traces from off to on or visa versa. This is described in more detail here: <u>The Cursor button group</u>

## 9.) Delete cursor annotations *Delta+Rclick* "o"

If you have added many plot annotations (with the menubox Mark tag) you can delete them one by one by selecting delete for the string and/or marker from the

Edit Marker window. However this would be tedious if you have had many annotations. This submenu provides a way to delete all of them with one simple action.

## 10.) Set dual cursor

The dual cursor mode allows you to simultaneously cursor two traces on the same plot. Usually this is set up using the 'DualCur' parameter. (see Dual Cursor). However you may also use this menu to set the dual cursor interactively. Simply put the cursor on the trace that you want as the dual cursor (by clicking on it) and then select this submenu. After that the Dual Cursor will become active on the selected trace. This submenu acts as a toggle, which means that if the dual cursor was already enabled, it will be disabled.

## 11.) Toggle Reposition mode Relick Delta

The reposition mode a key feature of plt's GUI building tool set which allows you to move and resize screen objects with the mouse. This submenu toggles between the normal GUI mode and the reposition mode and back. The reposition mode is described in more detail here: <u>GUI building with plt</u>

## 12.) Reposition Grid size *Delta+Rclick Delta*

This submenu brings up a small auxiliary figure titled **SnapTo** containing two sliders, one for controlling the x step size and the other controlling the y step size of the GUI object repositioning mode. An example of what this figure looks like and what the values mean can be found here: <u>GUI building with plt</u>

## The Color Selection box



When using the "Edit Marker", "Edit line", "Edit all lines", or "Edit figure colors" dialog boxes described above, if you right-click on an edit box containing an rgb triple, the color selection box will appear.

As with the rgb triples, the three sliders represent the intensity values (except in percent) and will initially be set to the same values that were in the edit box. You can move the sliders (or type in new values) to give the proportion of each color that you want. Only integer values between 0 and 100 are accepted giving you more than a million different colors (101 cubed). As you move the sliders, the color of the marker, line, or figure element selected is continually adjusted to reflect the slider settings. The color of the large rectangular patch to the right of the red slider (called the "current color patch") is also adjusted at the same time which makes it easier especially for the smaller screen elements.

To make it easier to find the most pleasing color, the 11 by 11 grid of colors is also updated every time a slider is moved. What this grid shows you are the colors that result when the intensity value of the active slider is mixed with 1 of 11 different intensity values of the two inactive sliders. The active slider (i.e. the slider that was last moved) is shown highlighted in yellow.

So for example, in the figure above, the bottom slider (blue) is active and happens to be set to 0%. This means that all 121 squares of the grid are made up of colors containing no blue. Each row of the grid contains a different intensity percentage of red (0% for the bottom row, 10% for the next row, and 100% for the top row). Likewise each column of the grid contains a different intensity percentage of green (0% for the left column and 100% for the right column). Thus, the upper right square in this example then would be yellow (rgb =  $[1 \ 1 \ 0]$ ).

When you see a color in the grid that you like, just click on it. The screen element selected as well as the current color patch will instantly change to be the same as the color you clicked on. Of course the two inactive sliders move to show the intensity values of the color you just selected. If you can't find a better color, you can revert to the color in effect when you right-clicked on the rgb triple by clicking on the current color patch.

All this may sound somewhat complicated, however it is far easier to do than to explain. Generally you can pick any of the three sliders, move it around a bit, and you will quickly see the color you want in the grid. By the way, clicking on the slider trough area moves the slider by 10%, so if you want to limit yourself to the 1,331 colors formed with the intensities 0,10,20,30,...100%, then you can see all such colors after just 10 clicks of the mouse.


# **Data editing**

plt has the capability to modify the plotted data, either one point at a time or over a specified range. Data entry is accomplished by entering the coordinates via the keyboard or by using the mouse when keyboard accuracy is not required. (An alternate method of data editing appropriate for short data sequences is described in the <u>Menu box</u> section.)

Properties
multiCursor
xView slider
Cancel
Range
Range↔
Rangeî↓
Insert
Insert↔
Insert1↓
Modify
Modify↔
Modify↑↓

The first step is to click on the trace containing the data that you wish to edit. The second step is to right-click on the Ycursor edit box. (This is the edit box closest to the lower right corner of the figure.) When you do that, a popup menu will appear with the list of 13 choices shown here. The first three items in this menu, don't really have anything to do with data editing, but this was a convenient place to put them. When you select the first menu item (Properties) one of three different property editing windows will appear depending on how it is selected. When you left-click on "Properties" a window appears which allows you to modify the color and other properties of the currently selected line and its

cursor. If before clicking on "Properties" you enable delta cursor mode ( $\Delta$  button) then a property windows appears which allows you to edit all the lines at once. This is often appropriate for the color property, but it also may be useful for some of the other trace properties, such as linewidth. And finally, if instead of left-clicking you right-click on "Properties" then a window appears that allows you to edit the figure colors. This is the easiest method of accessing the three property editing windows, although for historical reasons there are two additional ways of opening these windows. One is via the "Colors/Lines" selection of the menu bar and the other is via right-clicking the "Mark" tag in the menu box. Both of these methods are mentioned in the Menu box section which also has a full description of the three different property editing windows and how to use them.

The second menu choice (multiCursor) toggles the multiCursor mode which is described here in the <u>Cursoring</u> section and third menu choice (XView slider) toggles the xView slider which is described here in the <u>Zooming and</u> panning section

The fourth menu choice (**Cancel**) is useful if you opened this popup menu accidentally or when you want to abort an already initiated data editing operation.

The remaining nine selections in this popup contain the actual data editing commands. The descriptions below are written for completeness rather than brevity so don't be scared off. The commands are intuitive, so you may be better off skipping the descriptions at first in favor of experimentation. The nine editing commands are divided into these three types:

Modify	The modify commands change the x or y (or both) coordinates of a single data point. When using these commands, the length of the x and y vectors do not change.
Insert	The insert commands add a new xy data pair to the data at the current cursor point. When using these data editing commands, the length of the x and y vectors increase by one. The one exception to that is that when you attempt to add a new data pair with its y value less than the current minimum y-axis limit, then the data at the current cursor point is removed from the data set (i.e. the length of the x and y vectors decrease by one).
Range	With the range commands (as with modify) the length of the x and y vectors remain the same. However in this case more than one data point is changed. All the data points between the current cursor location and the location modified during the previous data editing command are modified so that all these points lie on a straight line connecting the two end locations. For this to work, the previously edited point and the current cursor location must both lie on the same line. If this is not true, then the Range commands behave just like the Modify commands described above, and you will know happened because the cursor shape will be consistent with a Modify command. (See cursor shapes table below).

The nine commands are also categorized into three modes identified by the

arrows next to each command as follows:

These are the three most commonly used data editing modes. As soon as you select the editing mode, the regular data cursor disappears and is replaced by an editing cursor with a different shape. (See data editing cursor shapes below). Then you can grab the edit cursor with the mouse and drag it to the desired location. However you will only be able to move the cursor up and down (i.e. only the y coordinate is allowed to change). Normally as soon as you release the mouse button (after the edit cursor has been dragged to its new location) the edit command will take effect, the edit cursor will disappear, and the normal data cursor will reappear. However if instead of using the normal left mouse button, you drag the edit cursor using the right mouse button, then when you release the mouse the edit command will take effect but the edit cursor will remain active. This makes it easier to see the effect of the edit **Modify**↑↓ and easily re-adjust if necessary. Once you are satisfied with the Insert↑↓ edit command you can regain the normal data cursor mode by **Range**↑↓ selecting "Cancel" (the second popup option mentioned above) or more quickly just by clicking anywhere in the plot area other than the edit cursor itself. In addition to moving the edit cursor with the mouse, you can also move the edit cursor more precisely by typing a new y value into the Ycursor edit box. (You can also modify the x coordinate by typing a new x value into the Xcursor edit box despite the fact that you can't change the x value when using the mouse.) After you type a value into either the x or y cursor edit box, the edit command takes effect and the data edit mode is immediately canceled and the original data cursor is restored (i.e. there is no provision for a delayed exit from edit mode like you get using the right mouse button). If you would rather stay in edit mode until cancelled explicitly, just use one of the three commands without any arrows after it (described below). These three commands behave identically to the three commands **Modify** ↔ described above except for the fact that with the mouse you can Insert ↔ only drag the cursor left or right (i.e. only the x coordinate is **Range** ↔ allowed to change). When using these three commands you can use the mouse to drag the edit cursor anywhere. As before the editing mode is cancelled

## Modify Insert Range

as soon as you release the mouse button (unless you used the right mouse button). As mentioned above, the use of the X and Y cursor edit boxes is slightly different in that it stays in edit mode until you cancel it explicitly (usually by clicking anywhere in the plot area).

The currently enabled cursor can be put into data editing mode from a program as well. For example the command plt click Yedit 12; or its functional form plt('click', 'Yedit', 12); puts the cursor into the "modify left/right" mode, since that is the 12th selection in the menu shown above.

<u>♦</u>
Þ
Δ
$\diamond$
$\triangleright$
Δ
$\diamond$
$\triangleright$
$\wedge$

The usual data cursor is a plus sign or a small circle. Once you select one of the nine editing modes, the data cursor changes to one of the nine edit cursors shown in this figure. Although it wouldn't really be necessary to have a different cursor for each mode, it does help you remember what mode you are currently in. If you don't like the default size that plt chooses for the data edit cursors, you can change this with a command such as: setappdata(gcf, 'CurEdit', 14). The new size will be used the next time a cursor edit mode is selected.

If plt was called using the MotionEdit parameter, the function specified with that parameter will be called continuously as you drag the edit cursor around. This function may be used to create or modify text, plots or other gui objects on the screen. Both the editz.m and pltquiv.m examples demonstrate the use of the MotionEdit parameter. (Also see Mouse Motion Functions)

If you want to save the altered data (to a file for example) you have to get the data from the 'xdata' and 'ydata' properties of the line handle. (Remember the line handles are returned by the plt call.) When the user modifies any data using these data editing functions, plt executes the user specified move cursor callback. (See the description of the 'set', 'moveCB' function here.) The callback routine or any other part of your application can use the 'NewData' application property of the current figure window to determine if data has been modified by one of the data edit commands. For example:

index = getappdata(gcf, 'NewData'); % returns the index where

```
xdata/ydata was modified
if index
   DataWasModifiedAction(index);
   setappdata(gcf, 'NewData', 0);
already processed
end;
```

% is there any new data? % yes, process the new data % indicate that the data was

Note that the NewData property only gets set after a data edit operation is complete, unlike the MotionEdit function which gets called as you are dragging a cursored value to its new location. There is one additional figure application data variable called Dedit that may also be useful for an application using the data editing feature. The command getappdata(gcf, 'Dedit') will return the following 9 element row vector:

## 'Dedit' figure application data (row vector)

1	The CursorID associated with the edited trace
2	A number from 1 to 9 which identifies which data edit command is being used. (This is starting from the top of the popup, so 1 indicates the "Range" selection and 9 indicates the "Modify↑↓" selection).
3	The handle of the cursor object associated with the modified data.
4,5,6	For internal use. (This saves the cursor marker shape, size, and linewidth so that the normal cursor can be restored when the data edit operation is complete.)
7,8	For internal use. (This saves the position index and trace number of the previously edited trace - information is needed for the modify range operations.)
9	The position index of the cursor into the edited trace

After performing a data edit operation it is likely that you will want to perform another data edit using the same data edit command. Thus it would be nice if you could initiate another data edit operation without having to again select one of the nine data editing operations. In fact there is a way to do this - simply rightclick on the Xcursor edit box. This behaves similarly to what happens when you right-click on the Ycursor edit box except you don't get the menu of data editing choices, since it will use your previous selection. There has to be a default for this operation in case the user right-clicks on the Xcursor edit box before doing any other data edit operation. The default is "Modify" since this is the most commonly used operation. It is rare to want to change this default, although it is not difficult to do so. In fact the demo\editz.m demo program changes this default, so refer to that example to see how it is done.



# **Calling sequence and line styles**

This section (not including the two large tables at the end) is a good introduction to how to use plt and the differences between plt and the native Matlab plot.

Usually you will call plt with at least two arguments:

#### plt(x,y);

This plots the data in vector x along the horizontal axis and the data in vector y along the vertical axis. x and y may be row or column vectors. plt will transpose one of the arguments if needed to line things up, so x could be a row vector while y was a column vector. x and y must be the same length however. If not you will get an error message saying that the vectors must be the same length.

If y is a real vector, plt(y) is equivalent to plt(1:length(y), y).

To plot more than one trace, include the x and y vectors for each trace in the argument list. For example this command plots three traces:

#### plt(x1,y1,x2,y2,x3,y3)

Quite often several traces share the same x vector. In this case we can simply repeat the x vector in the argument list, as in:

#### plt(x,y1,x,y2,x,y3)

#### or

plt(x, [y1;y2;y3]). (a shorthand way of writing the above).

That would work only if the y1,y2,y3 were row vectors. If they were column vectors you would need to write:

### plt(x,[y1 y2 y3])

You can call **plt** using an output argument, which will return a column vector of trace handles. For example:

## h = plt(x,[y1 y2 y3])

will return a 3 by 1 column vector **h** of handles. **h(1)** of course would be the line handle associated with the y1 trace. Most often when you type the plt command at the command prompt you don't need to save plt's return value. However when plt is called from a program sometimes the line handles are needed to allow further manipulations of the plot.

If x and y are both matrices of the same size, plt(x, y) will create one trace per column.

## None of this so far should come as a surprise since it is identical to Matlab's plot command. Some of the ways that plt and plot differ will become clear from what follows.

With **plot** the data to be plotted must be passed in via the argument list. However, you may call **plt** without any arguments, allowing you to choose the data to plot interactively.

Find out about this method here: <u>The Workspace Plotter</u>.

Unlike plot, plt will accept data passed in cell arrays. For example the following two commands do the same thing:

plt(x,[y1; y2; y3])
plt(x,{y1; y2; y3})

Although in the example above, y1, y2, and y3 must be the same length so there really isn't a big advantage for the cell array input. However, now consider these two commands (again, these two lines are equivalent to each other):

plt(x1,y1,x2,y2,x3,y3)

## plt({x1 x2 x3}, {y1 y2 y3})

With plot you must use the first form because cell arrays are not allowed. You can't combine the arguments into vectors because they may be different lengths. When typing in the command window the first form is probably easier anyway, but inside a program the second form is far more convenient, especially when the data is be read from files.

If y is a complex vector, plt(y) is equivalent to plt(real(y), imag(y)). Matlab's native plot works that way too. Unlike plot however, plt treats complex arguments this way no matter where they appear in the argument list. For instance if a and b are both complex, plt(a, b) is equivalent to plt(real(a), imag(a), real(b), imag(b)). (Why this doesn't work with plot has sometimes been a mystery and an annoyance to me.)

Also like the plot command you can include any line property in the argument list. For example:

```
plt(x,y, 'LineWidth', 2) is equivalent to
set(plt(x,y), 'LineWidth', 2)
```

However the behaviors of plt and plot differs in that with plot these line properties must appear after all the data vectors in the argument list. (plot gives an error otherwise). With plt the line properties may occur in the middle of the argument list. In that case, the line property is applied only to the lines defined earlier in the argument list. For example:

```
plt(x,[y1;y2],'Marker','Diamond',x,[y3;y4]);
```

only sets the Marker property for the first two traces. An equivalent to the above is:

```
a=plt(x,[y1;y2;y3;y4]);
set(a(1:2),'Marker','Diamond');
```

By using cell arrays, you can set properties differently on each trace. For example:

## plt(x,[y1;y2;y3;y4],'LineWidth',{2 2 4 2});

This would set the LineWidth of the trace associated with y3 to 4 and the other three traces to 2. A column ({2;2;4;2}) would have worked equally as well. The number of elements in the row or column vector must identical to the number of traces defined so far in the argument list. (so as above, traces defined after the LineWidth parameter will just be assigned to the default LineWidth. Note that this is not possible with plot, unless you collect the various trace handles and use set commands to set the LineWidths as desired. (plt tries to insulate you from this need to become familiar with handle graphics).

Two more examples:

```
plt(x,[y1;y2;y3;y4],'LineStyle',{'-' ':' '-.'
'none'});
```

```
plt(x,[y1;y2;y3],'Marker',{'square','none','+'});
```

This method of assigned properties works with any line property. In the two particular line properties used above, you could have replaced 'LineStyle' with 'Styles' and 'Marker' with 'Markers' and the results would be the same. Styles and Markers are not really line properties, however plt allows you to use those alternate forms to allow some additional flexibility in how you write the parameter that follows it. (For example a character array may be used in place of the cell array.) The details of the additional flexibility provided by using these two alternate parameters are described in the <u>Trace properties</u> section.

The special plot types vertical bars, error bars, and vector fields (arrows) are plotted with the help of auxiliary functions Pvbar, Pebar and Pquiv. The use of these functions is described in the <u>Auxiliary functions</u> section.

#### plt vs. pltinit

Most of the code for this toolbox is broken up into these two routines:

Contains the code used to create a new plt pseudo object which means creating a new plot axis or set of axes. This also normally

pltinit.m	means creating a new figure window as well. (The only exception to that is when the 'Fig' parameter is included in the parameter list.)
plt.m	Contains the code used to create or modify any of the remaining pseudo objects (including cursor, grid, edit, pop, slider, image, ColorPick, and HelpText).

From this description you might expect that since all the command examples shown above create a new plot, that they should really be calling pltinit() instead of plt(). While this is technically true, plt recognizes from the syntax when a new plot is being created and simply passes all of its arguments on to pltinit. The advantage of this is that "plt" is shorter and faster to type which is especially important when used from the command window. To create a new plot from a script or function, it is more a matter of taste which function you use. For a complicated gui, pltinit would be a better choice because your gui will likely have many calls to plt as well and it will be a lot easier to see where the plots are created if a different function call is used.

This completes the introduction. What remains in this section and in fact all the remaining help file sections might be too long and detailed to serve as an ideal way to learn about these parameters and commands. Perhaps an easier way to learn how to program with plt is to run thru all the demo programs (conveniently done with demoplt.m) while reading the comments at the top of each example program. The program comments may also be found next to a screen capture of each demo program in the

<u>Programming examples section</u>. You will learn about nearly every plt parameter and option this way. Then you can use what follows merely as reference material.

#### Figure application data:

After a call to plt, the following information is available from the figure application data: (The quoted strings are case sensitive.)

Returns a row vector of handles of the plotted data. The first handle in the vector main plot. This is followed by the su the bottom up. Finally, the last element the handle of the main plot right han
Returns the cursor ID for the axis wi be any axis that appears in the vector command above.
Returns the cursor IDs for each axis, (lower) axis and working upwards to axes. (There is not a cursor object as hand axis since the main axis cursor the right hand axis.)
Returns a list of all handles of all dat Note that this is identical to the plt re
Each axis (including the right hand a 'Lhandles' application data value. Th lines contained in that axis. The main exception since its Lhandles list incl the left and right hand axis.
If the plt call includes a parameter su <b>'FigName', 'Abc'</b> then this com handle of the figure window that plt useful in programs that create multip
Various plt arguments may be used t appearance, or contents of the Trace occasionally you may want to make after the plt call and this command w by returning the handle of the Tracel
This command will return the handle Assuming the menu box is in it's def command such as get(findobj(gcf, 'tag', 'Me will return a list of text objects with 'Help', 'LinX', 'LinY', 'Gri and 'XY<->'.

findobj(gcf,'user','grid')	This command returns a column vec handles of the plot grids for all the a is just a single axis (i.e. one grid obje equivalent to plt('grid',0,'g
getappdata(gcf,'params')	Returns a cell array list of the param command line. (All plt arguments ar arguments specifying the data arrays
getappdata(ax,'xstr')	Returns the value that was specified parameter when plt was called. $a \times r_{0}$ hand axis.
getappdata(ax,'ystr')	Returns the value that was specified parameter when plt was called. $a \times r_{0}$ hand axis.
getappdata(gcf,'multi')	Returns a column vector of handles to render the Multi-cursor (the text object markers, followed by the dotted vert cursor is not currently enabled, then empty.

## Single argument actions:

*The command strings here (as with most plt commands) are not case sensitive. So for example "plt help" and "plt HELP" are equivalent.* 

plt help	Displays the plt help file. You could also use the functional form of this command: plt('help'). Alternatively, if you just want a one page list of the plt parameters type type help plt.
plt version	Returns the plt version. Same as: <pre>plt('version')</pre>
plt save	Opens a dialog box allowing you to select a .plt figure file that can be opened later using the <b>plt open</b> item in the file menu. If you want to avoid the file dialog box add the file name as a 3rd argument (i.e. <b>plt save filename</b> ). The use of these plt figure files are described in more detail in the <u>Menu</u> <u>box</u> section. Note that this command is also available

	from the file menu.
plt open	Opens a dialog box allowing you to select a .plt figure file that was saved using the plt save item in the file menu. If you want to avoid the file dialog box add the file name as a 3rd argument (i.e. plt open filename). Note that this command is also available from the file menu.
plt close	If a programming error causes plt to crash, you may find it difficult to close the plt figure windows (because they use the close request function). This command solves the problem by closing all currently open plt figure windows. Figure windows not created by plt are not closed. (And of course you may also use the functional form.)
plt show	If the current figure was created by plt, then this command, or the equivalent functional form plt('show'), will return a list of trace numbers that are currently being displayed. For example if you run the demo program "plt5.m" (which has five traces) and then turn off traces 3 and 4 (by clicking on their trace IDs) then this command will return [1 2 5] showing that those three traces are currently active. You can also use this command with an argument (the functional form only) from a program or the command window to set the traces you want active for the current figure. For example after running plt5.m, the command plt('show', 2:5) will turn off the first trace while leaving the remaining four traces on. Note that the TraceIDs will change their appearance to indicate which traces are enabled just as if you had done the same operation by clicking on the trace names in the TraceID box. To disable all traces, use plt('show', 'all') or plt('show', 1:n) where n is the number of traces defined. The first two forms to the left (off or with no
	argument) deletes the help text and the last form (on)

plt plt plt	HelpText HelpText HelpText	off on	<pre>recreates that help text again (which it can do by retrieving the help text information using getappdata(gcf, 'helptext')). Help text is usually created by using the 'HelpText' parameter when the plt window is created (this is described in the Labels and figure properties section.) (And of course you may also use the functional form, i.e. plt('HelpText', 'on');).</pre>
plt	move		This command (which has the same effect as right- clicking on the delta cursor button) sets the current plt figure into its repositioning mode. This allows all gui objects to be resized and/or repositioned using the mouse. The new positions are displayed in the command window. Typing plt move a second time cancels the repositioning mode and returns the controls to their prior functions. Details may be found here: <u>GUI building with plt</u>



# **GUI building with plt**

Over the years I have created dozens of GUIs using Matlab, nearly all of which involved collecting and/or viewing data and interacting with the data or data collection process in some way, and I suspect the same is true for the GUIs that you need to create. The first GUIs I created were quite difficult, but as I built up my bag of tools each new program became easier and quicker to write. The key I found was to avoid re-inventing the wheel each time and the best way to do that was to create a series of "pseudo objects". A pseudo object is a collection of Matlab graphics objects embedded with features commonly needed in Matlab GUI applications. (I chose to call them pseudo objects to distinguish them from the graphics objects supplied in the standard Matlab environment.) These pseudo objects are combined into one file exchange submission called (for historical reasons) plt. My primary goal for plt is to make building GUI applications in Matlab easier, faster, and more fun while enabling you to create clearer, more concise code that is compatible across all Matlab platforms and versions.

The two main tasks in creating a GUI application are:

- 1. Choosing the graphical elements and configuring the sizes and positions of these elements.
- 2. Writing the code that enables these graphical elements to serve their intended purpose.

Matlab's GUI building tool (called Guide) helps a lot with the first task but contributes little to the second. However I've found that for all but the most trivial applications, the second task accounts for most of the frustrations and time spent. My strategy is to aid the second task by providing a rich set of the pseudo objects mentioned above. It may seem like this is a tall order for these new objects, but I hope the examples that follow will convince you that they can impressively reduce the amount of code you need to write. The current set of pseudo objects is merely a start. I plan to continue implementing new pseudo objects ... hopefully many of them conceived by Matlab users such as yourself.

A parallel goal is to make it easier to learn Matlab GUI programming by

providing many well commented examples that demonstrate as many of the pseudo object features as possible. It's easier to begin a new GUI application by starting with an example that has at least some of the graphical features that you need. To this end, plt includes 26 example programs covering a wide variety of GUI features and programming techniques. New example programs have been steadily added to the list over the years since the first version of plt, often initiated by questions and requests sent to me by plt users. Although the standard Matlab plotting and graphical elements are thoroughly documented, a common complaint is that this information is spread out over thousands of pages of Matlab documentation making it difficult to find what you are looking for. Also it is difficult to find examples for most features. This inspired me to design the plt help file to avoid these pitfalls by organizing plt's many features into one coherent help file including many examples. Every question from a plt user leads me to reexamine the documentation to see if I have described each feature and example as completely and clearly as possible.

Although the first task mentioned above (configuring sizes and positions of graphical elements) is not where most of the time is spent, without an appropriate tool this could be a painstaking task. Matlab's Guide tool does provide a reasonable solution to this problem however I found several annoyances with this tool:

- 1. Guide forces me to adopt a particular programming methodology and style. Although plt offers unique pseudo objects you are free to use all or none of them to suit your purpose and no demands are made on your programming style. My preferences lean towards conciseness and clarity, as you can see from my programming examples.
- 2. Guide has evolved over the years and works somewhat differently in different Matlab versions which can cause compatibility issues. I often support my Matlab applications for use with older Matlab versions, some of which were released even before Guide was invented.
- 3. I sometimes find it inconvenient that the program definition is split between the .m and .fig files. The .fig file format can also lead to Matlab version dependencies. (GUIs designed with plt don't depend on .fig files.)
- 4. And most importantly Guide was not compatible with the powerful pseudo objects that I had created.

In addition, to the Guide compatibility issues mentioned in point 2 above, the graphics objects themselves have changed between versions. For example the

latest Matlab version (R2014b) finally allows you to set the grid line color independently allowing you to create far more pleasing plots. However this won't help when you try to run your program on previous Matlab versions (which in fact represent the majority of the user base). On the other hand, plt is designed to work the same across all Matlab versions, so you can share your code with colleagues running older Matlab versions. I have tested plt on most Matlab releases dating as far back as June 2001 (Matlab 6.1) and of course it works with Matlab's most recent R2014b version as well. You might think that supporting older versions would limit plt's flexibility, however as you will see plt generally meets or exceeds the flexibility available with even the latest native Matlab plot commands (including grid lines of course).

After abandoning Guide for the above reasons, at first I simply entered position coordinates by hand, often iterating many times to adjust controls to achieve the desired look. But for complicated GUIs with many objects, this is too time consuming. This led me to develop a more automated method of positioning pseudo objects as well as native Matlab objects without using Guide. This method is not quite as automated as Guide, but I hope you will see from the following examples that it strikes a good balance between automation, power, and flexibility. The basic idea was to give plt the the ability to reposition and resize the graphical objects and to display the results in a way that allows you to copy the positions of all your objects into your program with a single cut and paste.

Since the pseudo objects are the key innovation that simplifies Matlab GUI programming, let's briefly summarize the nine pseudo objects that have been implemented so far:

- **Cursor** My goal for the cursor pseudo object was to tailor plt for its role in data exploration. The ease and responsiveness of the cursoring, panning, and zooming operations is unmatched by any other Matlab plotting package (at least according to reports from some plt users). Although you can create cursor pseudo objects independently, by default a cursor object is created for you when you define a plt pseudo object. This relieves you of the need to know about the many details of the cursor object.
- **Grid** In the latest Matlab release (R2014b) you can change the grid color using the axis 'gridcolor' property. However as I mentioned earlier, using this property will give an error if you run the program on any earlier Matlab

release. If you are using any release earlier than R2014b, you can see the problem by typing plot(rand(1,100)); grid on;. and note how the grid lines are so overpowering it is hard to see the trace underneath. Now try changing the grid line color to something less overwhelming by typing set(gca, 'xcolor', [.7 .7 .7], 'ycolor', [.7 .7 .7]);. Now the grid lines look pretty nice but the tick labels become so faint you can barely see them. (This insanity was an annoyance and embarrassment to Matlab users for at least two decades and in fact was one of my motivations for creating the first version of plt a decade ago). The plt grid pseudo object gives you even more flexibility than the latest Matlab release by allowing you to select the grid color, thickness, line style, and erase mode without affecting any other graphic element. As with the cursor pseudo object, you will rarely need to define a grid pseudo object explicitly since it is also created by the plt pseudo object. The defaults are to most people's liking so you may never need to adjust them.

- plt This can be thought of a super axis (or collection of axes). In that
  respect it is similar to Matlab's plot and data is passed to plt in the same
  way making simple calls to plt look the same as the call to plot (except for
  the missing "o"). However unlike plot, plt also (by default anyway) creates:
  - a cursor pseudo object.
  - a grid pseudo object.
  - a menu box containing various plotting controls.
  - a TraceID box which serves both as a legend and as a way to enable and disable individual traces.
  - a menu bar containing the traditional Matlab menus (File,Edit,View,Tools,etc) as well as a plt menu containing items unique to plt.

The plt pseudo object integrates all these elements together in a consistent logical manner with the defaults oriented towards the typical data exploration needs of the most common graphical interfaces. This allows you to take advantage of the plt features in your design even before learning about the many ways to tailor plt to your needs. It might seem strange that I use "plt" both as the name of this pseudo object as well as the name of the entire toolbox, however I think you will find that you can tell which one I'm referring to by context.

• **Slider** - Matlab's slider is the most versatile uicontrol because of the many ways you can change the value (dragging the slider bar, clicking the left or right arrow, clicking the trough). Plus it is the only uicontrol whose action

can repeat continuously as you hold down the mouse button. This allows you to smoothly vary a parameter over a range while observing its effect. However the slider control rarely can stand alone. At a minimum you need a label to identify what the control is for and usually you also need a more precise representation of the slider value than the slider bar itself provides. A text box is often used for this purpose, although an edit box is better since it allows a way to set the slider value precisely. It is also common to want labels to identify the minimum and maximum values associated with the left and right slider bar positions. Furthermore sometimes the slider value must be restricted to be an integer or a multiple of some other factor or other condition (such as a power of 2 for example). Also you may want to adjust the step size for the arrows or trough or to make the steps logarithmically spaced. These requirements mean that you usually need to write a lot of code to make a slider useful for your application. The pseudo slider solves this problem by integrating all the elements and options just mentioned into one object that you can simply drop into your application and move around as a unit. All the code you need to make it useful is already done for you!

- **Edit** As mentioned above, the ability of the slider and the pseudo slider to respond continuously as the mouse button is held down is a powerful feature. The one drawback of these objects is that they take up quite a bit of space. For a GUI with many controls, you may not have room to use sliders for many of them. The edit pseudo object is the answer to such a problem. It takes up even less space than a uicontrol edit box. It doesn't have all the features of the pseudo slider, but you can continuously increment its value by holding down the mouse on the right side of the object, and likewise decrement its value on the left side. Like the pseudo slider, you can select the increment amount as well as the min/max limits. The edit object may also be used to contain vectors or strings. (The auto-increment features do not apply in that case.) Usually edit objects require a label to identify the purpose of the control. The edit pseudo object includes a label (optionally) as an integrated feature. When you reposition the edit control, the label moves right along with it. One less graphical object to define, size, and position. Other advantages over the uicontrol edit box include:
  - Auto evaluations
  - More flexible formatting
  - More powerful callbacks
  - Tex interpreter support

• Choice of figure or axis coordinates

As an example of an Auto evaluation, consider that typing "cos(pi/6)" into the pseudo object would set its value to 0.866025 (with fewer or more digits depending on the format code). The last point also deserves elaboration. Normally you will want to define the edit pseudo object in figure coordinates (what I call a type I edit pseudo object) just as you do with a uicontrol. However if you want to associate the object with a plot (so it moves with the plot if it is repositioned, or if you want to create an array of edit objects it is more convenient to use axis coordinates (i.e. a type II edit pseudo object).

- **Popup** The popup pseudo object closely mimics the function of Matlab's popup uicontrol but has these advantages:
  - You can fit twice as many pseudo popups into a given space as uicontrol popups.
  - You can cycle through the popup's options without opening the popup by right clicking on the object. When you want to see the effect of all the possible selections, it is far faster with the pseudo popup than with the uicontrol.
  - Optionally swap the role of left & right clicks (super-button mode).
  - Integrated label (optional).
  - More powerful callbacks.
  - Independent control of the location/appearance of the opened and closed view of the popup.
  - More flexible formatting.
  - Tex interpreter support.
  - Ability to open or close the popup from a program.

Of course uicontrol popups and uicontrol edit boxes may still be used and may be preferred when you don't need any of the advantages listed above.

- **Image** Most of plt's features are tailored toward 2D plotting (functions of a single variable). However plt provides two methods to plot functions of two variables. One is to use a waterfall plot which makes use of the auxiliary function pltwater.m. The other method is to use the Image pseudo object. The Image pseudo object provides cursoring methods appropriate for this object type and also includes several optional components including:
  - A color bar which serves as a legend for the z-axis values as well as providing a method of changing the colormap used to represent the z data.

- A slider (labeled '**edge**') that allows you to control how wide a range around a midpoint is used when determining the color used to represent each array element.
- A slider (labeled '**'mid'**') that allows you to control the center value of the range of values used to determine the color for each array element.
- A checkbox that allows you to control the visibility of the axis gridlines.
- A 'view all' button, that when clicked on resets the axis limits so that all the data is visible. A secondary feature of this button is activated by right-clicking on it instead which zooms in on the center of the region currently in view.
- **ColorPick** Nothing is quite as individual as the colors we prefer in our applications, and allowing the user to choose the application colors is a true sign that the programmer cares about the user. However choosing several colors that have to blend together in a pleasing way is not a simple matter and providing a substandard interface for color choice can be more of a curse than a blessing. The ColorPick pseudo object was carefully designed to make it as easy as possible to give your application this flexibility. I've found it is important to present the user with palettes of colors to choose from and have the selected objects change instantly when a new choice is made. A very simple example of the use of the ColorPick object is given in gui2.m. A more elaborate example is the demoplt.m program which also includes the code to save the selected colors in a setup file so that the chosen colors remain permanent until changed again.
- **HelpText** To make our GUIs as easy to use as possible, it is nice to show help messages right on the main GUI figure. Even if the GUI is complicated enough to need a manual, most people won't read it and even if they do, some reminders of the basics when the program starts up can be useful. Of course these reminders will quickly become annoying if they get in the way or take up valuable screen area that could be better used by the application. The HelpText pseudo object was designed to solve this need by making it easy to format and position the help messages as well as to make them disappear as soon as you start using the GUI. When this pseudo object is created along with the plt pseudo object (by using the 'HelpText' parameter), right-clicking on the menu box help tag will make the help text reappear, although it is easy to create a dedicated button for that purpose if you prefer.

A more complete description of these pseudo objects from a programming perspective can be found here: <u>Pseudo objects</u>.

Now you are prepared to dive into the examples that follow:

## A first example

Our first GUI example doesn't do any plotting or anything else useful for that matter, so it may not present a compelling case for the GUI tools provided by plt. However since it consists of just a few dozen lines of code it is simple enough that you can quickly see how to use plt to arrange graphic elements inside a figure window.



I find that it is best to start working on a new GUI with pencil and paper. Imagine the control types and arrangement for the application and then sketch a mock up such as this. Your finished application rarely will look much like your first sketch, but with the rapid prototyping possible with Matlab and plt you can quickly iterate improvements in form, concept and implementation.

Here I have decided on 3

pseudo sliders across the top followed below by a uitable on the left and a frame on the right containing 4 uicontrols (a popup, slider, button, and checkbox).

The bottom part of the GUI consists of two large objects for displaying lists of numbers. The right most one is a simple text box with room to show about 10 lines of text. The left most one is a listbox which by virtue of its scroll bar can display a far larger data set (80 lines of text in this example).

First we create the figure window. I usually start by typing "figure" in the command window and adjust the figure size to get a first guess. In this example I decide on 430 by 350 pixels. The menu bar is not needed for such a simple GUI so the menu property is set to 'none'. I chose a dark blue-green for the figure background:

```
function gui1(in1)
figure('name','gui1','menu','none','pos',[60 60 430 350],'color',[0 .1
```

The next line defines the choices for the popup menu. Then we create the array which contains the positions for the three pseudo sliders followed by a single position which is (initially) used for all the uicontrols as well as the uitable. Note that all three slider positions are exactly the same (in the middle of the figure). This is easy and convenient at the moment, but means that all three sliders will appear on top of each other. Not a problem however, as it will be easy to use the mouse to move and resize the sliders to an appropriate position. We will use the fourth (and last) position in this array for all the uicontrols. So of course all these controls will also appear on top of each other, but again we will use the mouse to move them to the desired locations.

```
cho = {'choice A' 'choice B' 'choice C'}; % choices for popup control
p = {[.5 .5]; [.5 .5]; [.1 .1 .1 .1]}; % initial positions: Slider1; Sli
```

Next, the three pseudo sliders and the uitable are created. @CBsli, the last parameter of the slider call, specifies the callback - a function that will be called when the slider value is changed. Note that we save the handles of these four objects even though we don't really use them. (In a real GUI we would almost always need them.)

```
h1 = plt('slider',p{1}, 10, 'PseudoSlider 1',@CBsli); % create the pseudo slid
h2 = plt('slider',p{2}, 60, 'PseudoSlider 2',@CBsli);
h3 = plt('slider',p{3},800, 'PseudoSlider 3',@CBsli);
h4 = uitable('units', 'norm', 'pos',p{4}); % create the uitable
```

Since the uitable hadn't been invented yet for Matlab 6, there is an alternate version of gui1.m in the demo folder called gui1v6.m where this uitable is replaced by a radio button. Of course these two objects don't serve the same function, but since we aren't worried yet about functionality with this example, this is not a problem.

Next, let's create all seven uicontrols in a single line while collecting the handles in a variable named "h". (A long variable name I know ...) The spaces after the "h = " are there so that the property values (style, string, and callback) in the following three lines line up under the respective uicontrol command. This makes it easier to follow what is going on.

```
h = [uicontrol uicontrol uicont
```

Finally we save all 11 handles in the figure user data so the callbacks can easily find them (h1,h2,h3,h4 followed by the seven uicontrols we just created). This method works fine for such a simple program, but in the next example we will see the advantages of using a structure for this purpose instead of an array. We also execute the callback function to initialize the random data tables:

```
set(gcf, 'user', [h1 h2 h3 h4 h]); CBsli; % save the handles and execute the slider call
```

That's it for the main function, just 14 lines of code! (Although we will add a few more lines later.) We finish up by writing the control callbacks. The first three are just stubs to remind us to eventually put some useful action there. The last one (the slider callback) is the only one that does anything, which is to update the data tables with new random data. The reason the random numbers are in the exponent is to create numbers with a widely varying magnitude so that the table looks more interesting. (The same random numbers get put into both the textbox and the listbox). Note that the random numbers are converted to strings using "prin" a substitute for "sprintf" that includes features commonly needed in GUI programming. (prin.m and its documentation prin.pdf are included with plt.)

```
function CBpop(a,b)
                         % popup callback -----
  disp('popup callback');
function CBcheck(a,b) % checkbox callback ------
  disp('checkbox callback');
function CBpush(a,b) % button callback ------
  disp('pushbutton callback');
function CBsli(a,b)
                                              % slider callback ------
  h = get(gcf, 'user');
                                              % get the object handles
  t = 1e20.^{(rand(3, 80))/1e6};
                                              % generate the random data
  set(h(10:11), 'fontname', 'courier', ... % convert random data to a cell array
     'string',prin('3{%6V }~, ',t));
                                              % of strings for the listbox and the textbox
  set(h(4), 'data', 100*rand(3,2));
                                              % more random data for the uitable
```



Now that we are done with the coding, start the GUI by typing guil in the command window. The figure window on the left will appear. As we mentioned above, the uitable and all seven uicontrols are on top of each other near the lower left corner, so we only see the last one. Likewise all three pseudo sliders are also on top of each other in the middle of the figure. Now its time to fix this problem. Type plt move in the

command window to enable the mouse driven repositioning mode.



The graphical objects inside the current figure are grayed out to indicate that the repositioning mode is active. Now we can:

• left-click, hold, and drag to move an object.

• right-click, hold and drag to resizes an object.

• Double click to open an object's property

inspector window.

At the stage shown here we at least have moved and resized all the objects so we can see all the individual items, none of them overlap, and they are at least close to the positions we outlined in our sketch.

I notice that the frame is too bright since I intended it to be a subtle grouping. One way to adjust this is by typing commands into the command window. After clicking on an object (in repositioning mode) a variable called "hhh" is added to the base workspace containing the object's handle. So now I can experiment with colors or other properties by typing commands such as:

#### set(hhh, 'backgr', [1 1 2]/6, 'foregr', [1 1 1]/2);.

Once I get the look I want I can copy and paste the command from the command

window into the program (right before the line that saves the handles to the figure userdata):

#### set(h(1), 'backgr', [1 1 2]/6, 'foregr', [1 1 1]/2);.

Note that I changed the "hhh" to h(1) before inserting the line into the program. A second method for doing this (instead of typing the commands in the command window) is to double click on the object which will bring up the property inspector for that object. Then as you change the property values in the inspector you will immediately see the effect on the GUI. (The second method is certainly easier if you don't know the exact property names for the objects you are working with.)



With a little more rearranging we finally get the look we are aiming for as shown here. Note that the controls are still grayed out since the repositioning mode is still active. One thing that you will notice while repositioning objects is that moving the frame also moves all the objects inside the frame. Unless you have a very old version of Matlab you may also use a uitable to allow this grouping effect. The main advantage

of the uitable is that you can optionally specify a label for the grouping that will appear along the top edge. (The second example demonstrates this.)

Before exiting the program or canceling the repositioning mode click on each of the graphical objects once in the order that you created them in the program (left to right and top to bottom in the GUI). As you are clicking each item, plt will be displaying the positions of the objects in the command window. When you are done the command window will contain something similar to this:

uic:	207	.540	.500	.440	.280;	%	frame1		
uic:	206	.680	.710	.170	.050;	%	choice A		
uic:	205	.570	.610	.380	.060;	%	slider		
uic:	204	.570	.520	.170	.060;	%	button1		
uic:	203	.780	.520	.170	.060;	%	check001		
uic:	202	.020	.050	.480	.400;	%	1.79e9	8.4e-6	2.5e12
uic:	201	.540	.050	.440	.400;	%	1.79e9	8.4e-6	2.5e12
sli:	401	.020	.920	.300;		%	PseudoS1:	ider 1	

sli:	406	.350	.920	.300;	%	PseudoSlider	2
sli:	411	.680	.920	.300;	%	PseudoSlider	3
uit:	208	.020	.500	.480 .280;	%	uitable	

The first column is a three letter identifier for the object type. The native Matlab types - uicontrol, uitable, uipanel, axis, and text are identified as uic, uit, uip, axi, txt respectively and plt's pseudo objects are identified as sli, edi, pop, and xy. (xy refers to elements created by the cursor and plot pseudo objects.) The next column is a unique integer associated with the object. We won't need that now, but example 2 will show how that is used. The next four columns (or three columns for the pseudo sliders) specify the size and position of each object in normalized units. At the end of the line, the object type or string property of the object is included as a comment to make it clear which object the line refers to.

4	$p = \{[.5,.5]; [.5,.5]; [.5,.5]; [.1,.1,.1]\};$					
5						
6	sli: 401 .020 .920 .300 ; % PseudoSlider 1					
7	sli: 406 .350 .920 .300 ; % PseudoSlider 2					
8	sli: 411 .680 .920 .300 ; % PseudoSlider 3 ハルナ メテ いっくナク					
9	uit: 208 .020 .500 .480 .280; % uitable					
10	uic: 207 .540 .500 .440 .280; % framel					
11	uic: 206 .680 .710 .170 .050; & choice A					
12	uic: 205 .570 .610 .380 .060; % slider					
13	uic: 204 .570 .520 .170 .060; % button1					
14	uic: 203 .780 .520 .170 .060; % check001					
15	uic: 202 .020 .050 .480 .400; % 1.02e6 5.0e-5					
16	uic: 201 .540 .050 .440 .400; % 1.02e6 5.0e-5					
17						
18	h1 = plt('slider',p[1], 10, 'PseudoSlider 1',@CBsli);					
19	h2 = plt('slider', pl2', 60, 'PseudoSlider 2', QCBs(1);					
20	n3 = plt('slider',pl3;800, 'Pseudoslider'3',@CBsl1);					
21	<pre>n4'= ultable('units', 'norm', 'pos',pi47); h</pre>					
22	at the flat when the supervised in the supervised and the supervised in the supervis					
23	set(n, 'style'; 'Trame'; 'popup'; 'slider'; 'pusho'; 'checkoox'; 'lstbox'; 'text';;,					
24	f icalbil					
20	the start [5, 1, 1] (united in part in					
20	act (act (1, 1, 1, 1), atts, 100 m, pos, p(4));					
28	Send function auil					

Now we are going to make the hard work we did to reposition the controls permanent. First cut and paste the 11 lines of coordinates from the command window (shown above in blue) directly into the gui1.m source code.

<pre>p = {[.020 .920 .300]; % PseudoSlider 1</pre>
[.350 .920 .300 ]; % PseudoSlider 2
[.680 .920 .300]; % PseudoSlider 3
[.020 .500 .480 .280]; % uitable
[.540 .500 .440 .280]; % frame
[.680 .710 .170 .050]; % popup
[.570 .610 .380 .060]; % slider
[.570 .520 .170 .060]; % button
[.780 .520 .170 .060]; % checkbox
[.020 .050 .480 .400]; % listbox (80 lines)
[.540 .050 .440 .400]}; % text (10 lines)

Then we fix up the 11 imported lines by deleting everything except the coordinates and comments, and adding brackets as appropriate. I've also made slight changes to the comments

for clarity. I find it handy to turn on my editor's keystroke recording while fixing up the first line. Then I can replay this record to fix up the remaining ten lines by hitting the "play macro" button ten times. Most editors have this feature ... but even if you do each line manually, it's not a big deal.



Finally, in this line where we are setting the position of all seven

uicontrols to the same value ... we have to change that (as shown here) so that the last seven entries of the position array are used in sequence.

🛃 Figure 1: gui 1 📃 🔍 🚽								
0 C	eudoSlider 1	Pse 10 0	eudoSlider 2 30	Pseudo 60 0 40	Slider 3 0 800			
1 2 3	1 73.8091 72.8905 55.6243	2 18.2352 59.1239 29.6938		choice A	check001			
2.55e9 210.46 0.1345 65281. 2.2987 8.15e6 78401. 2.64e8 .02363 1.9e-5	9 10.101 5 6.9e10 5 1.86e6 2.5e11 5 2.4e-4 5 3883.9 8.84e6 8 28178. 8 1.3e-4 5 123.48 7 1 9669	1.74e7 4.6e10 441553 0.1975 6.3e12 6.11e6 75460. 7.5e13 1.6e-4 927.81 2.6106	<ul> <li>▲ 2.55;</li> <li>210</li> <li>0.13;</li> <li>6528;</li> <li>2.29;</li> <li>8.15;</li> <li>7840;</li> <li>2.64;</li> <li>.023;</li> <li>1.9e;</li> </ul>	e9 10.101 46 6.9e10 45 1.86e6 1. 2.5e11 87 2.4e-4 e6 3883.9 1. 8.84e6 e8 28178. 63 1.3e-4 -5 123.48	1.74e7 4.6e10 441553 0.1975 6.3e12 6.11e6 75460. 7.5e13 1.6e-4 927.81			

Hit save on your editor, close the gui1 figure window, type "gui1" in the command window to restart it ... and as we hoped the GUI appears just as we had organized it. Try expanding the size of the figure window and note how all the objects grow in proportion to the figure size. This is because we used normalized coordinates throughout. If you want to convert this GUI to pixel coordinates, enter repositioning mode, type

set(findobj(gcf), 'units', 'pix'), and then click on every graphics object, again in the same order as they are defined. Again, cut and paste the coordinates (which now are integer pixels) into the program as we did before. Also remove the two instances of 'units', 'norm' (since pixels is the default when the units aren't specified). When you run it, the GUI will at first look the same, but when you stretch the figure size, all the objects will stay exactly the same size in the same position, thus creating empty space inside the figure. Generally you will stick with either pixels or normalized coordinates, although you can mix them if it suits your purposes.

This concludes our example, although if this was a real GUI you would not likely be satisfied yet. But using the methods we just demonstrated you will be able to iterate until you are satisfied with the control types and positions.

## A second example

Now that we have covered most of the basic concepts and techniques, its time to explore the true power of plt by reviewing the design of a real GUI in one of the application areas that Matlab was designed for. The application I have chosen is the display and analysis of the classical analog filters. Granted this is not a particularly novel idea as it probably has been done before in Matlab and other languages, but nonetheless it serves various educational and practical needs and there is always room to apply our own slant to the project. I'll start out with a relatively modest set of goals:

- Display the magnitude frequency response for the five "classical" analog filters (Butterworth, Bessel, Chebyshev type 1 & 2, and Elliptic). The user should be able to easily select which of these filters to display, as well as allowing all of them (or any subset) to be plotted at the same time.
- Interactive selection of the filter order and type (lowpass, highpass, bandpass, stopband).
- Interactive selection of the number of decades to plot as well as the frequency resolution.
- Both numerical entry and slider control of cutoff frequencies and pass/stop band ripple.
- Cursors should be provided which allow for the easy readout of the frequency response at any point as well as delta readouts to verify stop band and pass band ripples. Peak finding should be provided as well as the ability to annotate the plot with text and markers to document features of interest.

As is my habit, I start with a sketch to clarify my thoughts. I decide to use an array of four pseudo sliders along the top to control the continuously adjustable parameters (edge frequencies & ripple) The four remaining filter and display parameters are grouped to the left of the sliders inside a uipanel unimaginatively labeled "Parameters". The Trace IDs to the left of that will be named after the classical filter types and be used to select which filters to display. The plot and the cursor controls and readouts along the bottom edge are the standard ones created by the plt pseudo object.



Ok ... it's time to start writing code. First I have to come up with some first guess for the object positions, and then I define the choices for filter type and number of points to display:

```
function gui2()
  p = \{[.4 .3 .5 .5];
                           % plot position
        [.2.5.1.2];
                           % uipanel position: Parameters
        [.2.5.1.2];
                         % edit position: filter order
                           % popup position: filter type
        [.2 .5 .1 .2];
                           % popup position: # of decades
        [.2.5.1.2];
        [.2.5.1.2];
                           % popup; position: # of points
        [.2.2]
                           % slider position: Passband ripple
                      ];
        [.2.2]
                           % slider position: Stopband ripple
                       1;
                           % slider position: Cutoff frequency
        [.2.2
                       1;
        [.2.2
                       ]}; % slider position: frequency 2
  typ = {'low pass' 'high pass' 'band pass' 'stop band'}; pts = 100*[1 :
```

So how good are my guesses? Look at the first screen shot below to find out. Clearly not so good. The plot is way too small compared to my sketch. All the pseudo sliders are on top of each other ... and they are near the bottom instead of near the top as shown in the sketch. Plus all the pseudo popups and edit objects are also on top of each other. But as you will see, this will not be a problem at all. In fact, it would be a waste of time to spend more than a minute coming up with the initial guess. Ok, now it is time to create the plotting pseudo object (figure, axis, cursor, grid, traces, etc):

```
S.tr = plt(0,zeros(1,5),'Options','LogX','Ylim',[-80 10],...
'TraceID',{'butter' 'bessel' 'cheby1' 'cheby2' 'elliptic'},.
'xy',p{1},'LabelX','radians/sec','LabelY','dB');
```

The 'xy' parameter is used to position the plot within the figure window (although you will soon learn that this parameter can do far more than that). The data to be plotted for all 5 traces is defined in the plt call (as it must), but notice that each trace just contains the single point (0,0). When calling plt from the command line, you almost always include the actual plot data in the argument list, however in a GUI more often than not the data supplied is just a place holder. The real data is loaded later (in the callback in this example) by using the trace handles returned by plt. Note that we save these handles in **s**.tr (a 1x5 array). S is the structure where we will store the handles of all the objects we define in the GUI. The remaining plt parameters should be reasonably self explanatory. Next we create the uipanel and the four pseudo objects that we will put inside it:

```
uipanel('units','norm','title','Parameters','backgr',get(gcf,'color'),
                                 'pos',p{2} ,'high',[.4 .4 .4],'foregr',[.4 .4 .4]);
S.n = plt('edit', p{3} ,[6 1 25],'callbk',@clb,'label',{'Order:' .0!
S.typ = plt('pop', p{4} ,typ,'callbk',@clb,'swap');
S.dec = plt('pop', p{5} ,1:5,'callbk',@clb,'index',3,'label','Decade:
S.pts = plt('pop', p{6} ,pts,'callbk',@clb,'index',2,'label','Points
```

For the uipanel, I set the background color to be the same as the figure color to give it a transparent look. For both the border outline and the text label of the uipanel I used light grey (rgb = .4 .4 .4). The uipanel wasn't invented yet for Matlab 6, so there is an alternate version of gui2.m called gui2v6.m in the demo folder where this uipanel was replaced by an axis. Note that the tag property of the axis was set to 'frame'. This is to tell plt that moving the axis in repositioning mode should also move all the objects inside it (even objects not children of the axis).
The [6 1 25] parameter of the pseudo edit object means that its initial value will be six with min/max limits of 1 and 25. The string 'Order' is used as a label for the edit object, and the ".05" tells it how much space to allocate for the label (in normalized coordinates). The parameters for the three pseudo popup objects are probably more obvious, but if not, consult the <u>Pseudo objects</u> page. Next we create the pseudo sliders:

```
S.Rp = plt('slider',p{7},[ 2 .01 9],'Passband ripple', @clb);
S.Rs = plt('slider',p{8},[ 40 10 120],'Stopband ripple', @clb);
S.Wn = plt('slider',p{9},[.02 .001 1],'Cutoff frequency',@clb,5,'%4
S.Wm = plt('slider',p{10},[.2 .001 1],'frequency 2', @clb,5,'%4
```

The [2 .01 9] on the first slider has the same meaning as the similar pseudo edit parameter mentioned above - i.e. 2 is the initial value with min/max limits of .01 and 9. The @clb specifies the callback function. (Note that same callback function is used for all the controls.) The "5" after the callback function indicates that the slider will move logarithmically (so for example the slider will move the same number of pixels going from .01 to .1 as it does when changing from .1 to 1. The final parameter '%4.3f 6 2' is shorthand for '%4.3f %6v %2v' and specifies the display format for the min value, current value, and max value respectively. Now we have just a few more lines left to complete the gui2.m function:

```
set(gcf,'user',S);
clb;
% end function gui2
```

The first line saves the handle structure in the figure user data where the callback function can easily retrieve it. The next line (the last of the gui2 function) executes the callback function to initialize the display to agree with the initial values of the controls. After just 17 lines of code, we're finished writing the main line function, plus 10 more lines for our initial guess for the control positions. But now the real work begins - the callback function that makes the GUI come alive:

```
function clb() % callback function for all objects;
S = get(gcf,'user'); % get handle structure
ty = plt('pop',S.typ); % get filter type index
```

```
t = {'low' 'high' 'bandpass' 'stop'}; t = t{ty}; % get filter type name
N = plt('edit',S.n); % get filter order
dec = plt('pop',S.dec); % get number of decades
pts = str2num(get(S.pts,'string')); % get # of points to plot
X = logspace(-dec,0,pts); W = X*1i; % X-axis data (radians/se
```

First we pick up the filter parameters that are inside the uipanel (filter type, order, number of points, number of decades). You might wonder why I seem to repeat myself by defining the filter types again since it would seem more logical to simply get the filter type with get(S.typ, 'string'). That command would retrieve one of the following strings:

{'low pass' 'high pass' 'band pass' 'stop band'}, but the strings accepted by the Matlab filter functions are slightly different: {'low' 'high' 'bandpass' 'stop'}. It would have been much easier just to use the strings that Matlab requires for the popup control, but I was too picky about the look of the popup control to use those somewhat inconsistent strings. Finally the logspace command generates the requested number of points logarithmically spaced between .001 and 1 (for the 3 decades example). W is this same vector on the imaginary axis, which is used with polyval to compute the frequency response function.

Next we pick up the filter parameters from the four pseudo sliders. Note that for the last two filter types (bandpass and stopband) we need the second frequency slider ("frequency 2") and so this slider is only visible when one of those filter types is selected.

Then we use the Matlab classical filter functions to compute the numerator and denominator s-plane polynomials (B,A) and compute the frequency response using polyval. Although it would have been slightly shorter to use freqs() instead of polyval(), I didn't do that since freqs is part of a toolbox that some users will not have. If a filter type other than low pass is selected, the last line changes the Bessel transfer function to "NaN" so that the trace will not appear on the plot. (The Bessel filter is only defined for low pass.)

```
for k=1:5 set(S.tr(k), 'x', X, 'y', 20*log10(abs(H{k}))); end; % set trace data
  plt('cursor', -1, 'xlim', X([1 end])); % set Xaxis lim
% end function clb
```

Then we use the absolute value function to compute the magnitude of the frequency response, and convert to dB (20\*log10) before placing the result in the y-axis property of the 5 traces. Finally to set the x-axis limits in case they have changed (which happens when the callback is in response to the "number of



decades" control).

Finally we are done with the initial coding and we can try it out. Typing "gui2" to start the program brings up this figure. Although all the controls are there as promised, they are not anywhere close to being in the right place, but it will take only a few minutes to fix this. Begin by entering

"repositioning mode" by

right-clicking on the delta button (or if you prefer, by typing plt move.) Then as I described in the previous example, use a left click and drag to move the objects around and a right click and drag to resize them.



I still don't have the final positioning, but it's close. The uipanel contains the controls it should and the other objects are also at least in the vicinity of where they should be. Note that once objects are placed inside the uipanel, moving the uipanel will also move all the objects inside it. After a few more tweaks, we will have at least our first cut

positioning. Before closing this figure it is important to remember to left-click once on every object, in the order that they were created in our program. As we are doing this, the text below will appear in the command window:

xy:	1	.130	.105	.840	.760;	%	axes
uip:	213	.100	.885	.240	.110;	%	uipanel
edi:	211	.165	.935	.040	.030;	%	6
pop:	102	.110	.710	.100	.200;	%	band pass
pop:	103	.310	.750	.020	.200;	%	3
pop:	104	.287	.710	.054	.200;	%	200
sli:	401	.350	.946	.150	;	%	Passband ripple
sli:	406	.510	.946	.150	;	%	Stopband ripple
sli:	411	.670	.946	.150	;	%	Cutoff frequency
sli:	416	.830	.946	.150	;	%	frequency 2

<pre>3 function-gui2() 4 · p = {[.4 · .3 · .5 · .5]; · % plot · · · position 5 · · p = · {[.4 · .3 · .5 · .5]; · % plot · · · position; Parameters</pre>	Then
[12.5.1.2]; suparet position: Parameters	_
[.2.5.1.2]; seal position: filter order	as
7	uo
8 [.2 .5 .1 .2]; % popup position: # of decades	
9 [.2 .5 .1 .2]; % popup position: # of points	
10l.2 .2	
11 ·····[.2 .2 ····]; ·% ·slider · position: Stopband ripple	
12 [.2 .2 ]; % slider position: Cutoff frequency	
13	
14	
15 xy: 1 .130 .105 .840 .760; % axes	
16 uip: 213 .100 .885 .240 .110: % uipanel	
17 edi: 211 165 .935 .040 .030: \$ 6	
18 pop: 102 110 710 100 200: % band pass	
21 c) 401 - 250 - 046 - 150	
21 sli. 401 . 530 . 946 . 150	
22 stil 400 .310 .940 .130 , s stopped rippte	
25 511-411-000-940-1130	
24 St1: 416	
25	

before, we copy and paste those lines into the source code as shown here.

3 function-gui2()
4 p = {[]; % plot position
5 ·····[]; % uipanel position: Parameters
6[]; % edit position: filter order
7 ·····[·]; % popup · · position: filter type
<pre>8[.]; % popup position: # of decades</pre>
9 ·····[]; % popup position: # of points
10
11
12 []; % slider position: Cutoff frequency
13 ····· [/]}; % slider position: frequency 2
14
15 xy: 1 .130.105.840.760; % axes
16 uip: 213 .100.885.240.110; % uipanel
17 edi: 211 .165935040030; % 6
18 pop: 102 .110.710.100.200; % band pass
19 pop: 103
20 pop: 104 .287 .710 .054 .200; % 200
21 sli: 401 .350.946.150; % Passband ripple
22 sli: 406 .510.946.150; % Stopband ripple
23 sli: 411 .670.946.150; % Cutoff frequenc
24 <b>sl1:</b> 416 .830 .946 .150; % frequency 2

We could fix up the brackets and comments line by line as we did in the previous example, however since my editor has a column select mode (as pretty much every programmers editor does) I find it easier to block delete the old coordinates (our rough first guess) and then do a block move (as shown by the red arrow) the new coordinates into the blank array.

We're done with the repositioning step, so we hit save in our editor, restart the application and we should see a figure similar to the one below.

Now we can play with all the controls and make sure everything is behaving as we imagined. Not bad considering we've written a non-trivial GUI applications involving non-trivial filter computations by writing only 51 lines of code. (Fewer if you don't count the automatically generated table of numbers that specify the



object positions.) It's the power of the pseudo objects that allows the program to be written so quickly and concisely.

Of course what nearly always happens the first time you get to experiment with your GUI is that you will have some new ideas:

- Perhaps this isn't the most convenient set of controls. Would it be more useful to have fewer controls, more controls, ... or just different controls?
- Even if the controls seem appropriate, perhaps it would be more esthetically pleasing to rearrange and resize them?
- Do we have features that we don't really need? Or can we add useful features without making the GUI too complicated?
- Can we rearrange or refactor the code to make it easier to understand and adapt?

Indeed when I got to this stage of testing the application I did have a few enhancement ideas:

1. I was curious (mostly for the elliptic filter) how the width of the transition band (the space between the passband and the stopband) varied as the filter order changed as well as the four slider parameters. Could I define such a measure, figure out how to compute it and find a place on the GUI to display it?

- 2. My second idea was to allow the user to control at least some aspect of the color choices used in the application. Actually I don't think such a simple application like this really needs this flexibility, but my ulterior motive was to showcase the ColorPick pseudo object and how easy it is to add to your GUI and how easy it is to select the color you find most pleasing for any display element.
- 3. What does the phase response of these filters look like? Could I add a display of the phase response without cluttering up the plot or obscuring the magnitude response (which is still the primary interest).
- 4. It would be nice if whenever we made a change to the figure size/position or the color selection, that these changes would be recorded so that the application looks the same the next time it is restarted. While we are at it, we might as well remember the state of the eight filter parameters (shown above the plot) so that on start up, the figure looks identical to the way it was when it was shut down.
- 5. Finally lets add a very brief set of help messages to allow a new user of the program to get started without having to consult any help files or manuals. The most important consideration should be that the help messages are not distracting in any way to the user who is already familiar with the help information presented.

These enhancements turned out to be fairly easy to implement. You can look at the final code which includes these enhancements (gui2.m in the plt\demo folder) or read on to find out more about the process.

# 1.) Adding a multi-line text string (elliptic transition ratio)



For a low pass filter, I characterized the transition width in terms of the ratio of these two frequencies:

- The frequency where the stop band spec is first achieved
- divided by the last frequency where the passband spec is still achieved

For a high pass filter, the ratio is:

- The frequency where the pass band spec is first achieved
- divided by the last frequency where the stopband spec is still achieved

I added an 11th line to the position array at the beginning of the program, to define the location for the new text object. Initially it was just a wild guess as usual which was refined using the repositioning mode:

[-.09.650]; % text position: elliptic transition ratio

The text object was created with this line (added after the slider definitions):

```
S.etr = text(0,0,'','pos',p{11},'units','norm','horiz','center','color
```

Note that the string to display was set to null, because the actual string to display will be set in the callback function as follows:

```
h = find(get(S.tr(5),'y') < -Rs2); % find indices where
if isempty(h) h = 0; % stopband specifica
elseif (ty-2)*(ty-3) h = X(h(1))/Wn(1); % computation for lo
else h = find(diff([h inf])>1); h = Wn(1)/X(h(1)); % computation for hi
end;
set(S.etr,'string',prin('Elliptic ~, transition ~, ratio: ~, %5v',h));
```

This last line takes advantage of prin's cell array delimiter feature to create the multi-line string used to display the elliptic transition ratio in the small space available on the left side of the plot. To learn more about prin and the %v format used here, check out the <u>Auxiliary functions</u>.

# 2.) Selecting colors

This was one of the simplest of the five enhancements requiring just the three extra lines shown below. I decided to enable color adjustment of just the pseudo sliders (the most prominent controls), although it would be easy to extend this to other graphic elements. In this figure I have changed the background color of the sliders from its default gray to orange. I encourage you to play around with this ColorPick figure. (Just right-click on any of gui2's pseudo sliders to bring up ColorPick.) If you have ever dealt with the frustrations of assigning screen colors, I think you will be pleasantly surprised about how easy it can be. Also you can read about the ColorPick details near the bottom of this page: <u>Pseudo objects</u>



The first line (below) gets puts the handles of all the objects associated with pseudo sliders into "h". Then the 5th element of each slider is removed, since that is the edit box portion which generally is set to a contrasting color. The second line assigns the ColorPick object as the buttondown function (the action

associated with **right** clicking on the pseudo slider). The third line is necessary to tell ColorPick which property of these objects should be adjusted when a new color is chosen (the background color in this example).

```
h = getappdata(gcf,'sli'); h(5:5:end) = [];
set(h,'buttondown','plt ColorPick;');
for k = 1:length(h) setappdata(h(k),'m',{'backgr' h}); end;
```

# 3.) Linking traces (adding the phase display)

To add the phase display, the most important change is in the callback function. Before we set trace data for 5 traces, but now we must set trace data for 10 traces (the first five for magnitude and the last five for phase):

```
for k=1:5 % set trace data
    set(S.tr([k k+5]), 'x', X, {'y'}, {20*log10(abs(H{k})); angle(H{k})*180/|
end;
```

Then we just need to increase the data array defining the traces in the plt call from 5 to 10, specify that the last 5 traces should be on the right hand axis ... and we would be done. However then we would need 10 TraceIDs up there as well. I

didn't want that because then to enable or disable the trace for the cheby1 filter (for example), I would have to click on two TraceID tags. Not so convenient. Also by default, ten different colors would be chosen for the ten traces. This would make it more difficult to tell which phase trace was associated with which magnitude trace. Both these problems are fixable of course:

```
c = [0 1 0; 1 0 1; 0 1 1; 1 0 0; .2 .6 1]; % trace colors
lbl = {'dB' [blanks(70) 'Phase \circ']}; % y-axis labels: {left, right}
```

The first line above defines the trace colors that also happen to be the default colors normally used for the first five traces. Only we are going to use them below for both the first five traces as well as for the last five traces. The next line defines the y-axis labels for both the left axis (magnitude response in dB) and for the right axis (phase response in degrees). Note the Tex command "\circ" in the right axis which inserts a small circle (the degree symbol) into the label. The 70 blanks that are inserted in front of the right hand label is used to push the label up towards the top of the display where the phase information will be plotted. And finally we have to fix up the main plt call:

```
S.tr = plt(0,zeros(1,10),'Right',6:10,'Options','LogX',...
'DualCur',-5,'TraceID',{'butter' 'bessel', 'cheby1' 'cheby2'
'Ylim',{[-90 60] [-1000 200]},'LabelX','radians/sec','LabelY','
'TIDcback','t=plt("show"); t=t(find(t<6)); plt("show",[t t+5])
'xy',p{1},'TraceC',[c;c],'+Ytick',-140:20:0,'-Ytick',[-180 0 1;</pre>
```

Some of the parameter changes in the plt call were already mentioned, but some others merit mention:

- The 'DualCur' parameter with minus five as an argument tells plt that the second trace number that should be cursored is offset from the first one by five. This means for example, if you click on trace 2, not only will a cursor appear on trace 2, but trace 7 will also have a cursor. (And each cursor will have a separate readout edit box as well.)
- The 'Ylim' parameter now includes two sets of y limits. As with the 'LabelY' command, the first entry is for the left axis and the second is for the right axis. The limits have been chosen to position the magnitude response on the lower portion of the graph and the phase response on the upper portion.
- The '+Ytick' and '-Ytick' parameters specify the tick marks to be used on

the left and right hand axis (respectively). We don't technically need these parameters, but it looks better to include tick marks only in the area where the data can be located.

• The addition of the 'TIDcback' parameter is perhaps the most interesting. This defines a callback function that is executed every time you click on any one of the TraceID text strings. Here, the plt('show') function (see "Single argument actions" near the end of this <u>page</u>) is used to enable only those phase traces that correspond to magnitude traces that are also enabled.

# 4.) Saving/restoring the GUI state using a configuration file

Before the call to plt, let's choose a file name and path for saving the configuration data: S.cfg = [which(mfilename) 'at'];

Next let's add a new function, called cfg which saves the current configuration to the file:

```
function cfg()
S = get(gcf,'user'); sli = findobj(gcf,'style','slider');
cf = { plt('edit',S.n); plt('pop',S.typ);
        plt('pop',S.dec); plt('pop',S.pts);
        plt('slider',S.Rp); plt('slider',S.Rs);
        plt('slider',S.Wn); plt('slider',S.Wm);
        get(sli(1),'backgr'); get(gcf,'pos') };
save(S.cfg,'cf');
```

Then right before we initialize the plot, we load the configuration file if it exists and set the GUI elements to agree with the data in the file:

```
if exist(S.cfg) load(S.cfg);
    plt('edit',S.n,'value', cf{1}); plt('pop',S.typ,'index',cf{3}); plt('pop',S.dec,'index',cf{3}); plt('pop',S.dec,'index',cf{3}); plt('slider',S.Re,'set',cf{5}); plt('slider',S.Re,'set',cf{5}); plt('slider',S.We,'set',cf{7}); plt('slider',S.We,'set(h,'background', cf{9}); set(gcf,'position', cf{1});
end;
```

And finally we add this parameter to the **plt** call:

'closeReq',@cfg

This instructs plt to call the function that saves the configuration data when the user closes the figure window to exit the application.

# 5.) Adding temporary user help message



The **HelpText** pseudo object is ideal for this task since it provides a mechanism for removing the messages once you start using the program. To define the help text and make it visible on the screen, these three lines were added to the end of the main gui2 routine:

```
htxt = {'Select the filter order & type' ...
    'in the parameter box above.' '' ...
    'Vary the ripple & frequency' ...
    'parameters using the sliders.' .6+.62i};
plt('HelpText','on',htxt); % show help text
```

Note that we have defined a help message consisting of five lines of text (with the middle line is blank). The complex number at the end specifies the position relative to the main axis where we want the help text to appear. The real part specifies the horizontal position and the imaginary part specifies the vertical position (in normalized units). And lastly, this line was added to the and of the callback function (clb):

```
plt('HelpText','off');
```

That line insures that as soon as the user starts doing anything with the program,

the HelpText will disappear insuring that it does not become a distraction.

This concludes our discussion of the gui2 example. Although it might seem like coding this example was a lot of work, only about 85 lines of code were needed to implement a fairly complex set of display and computational requirements. GUI programming is notorious for its complexity, and I believe that if you tried to implement a this application in other programming languages you would be looking at a far larger effort with source code running into the many hundreds of lines. I would like to be able to report how long the program would be in Matlab using GUIDE (without using plt), so I would be thrilled if one of the guide experts out there would take up this challenge by implementing the original five goals of gui2 as well as the five enhancements. If you manage to do this, I would gladly include your GUI (with credit of course) to contrast the Guide programming style with the one I present here.

To further your education of GUI programming with plt, I especially recommend reviewing the pltsq.m application if you are interested in moving plots (i.e. realtime updating). Also the curves.m, editz.m, pltmap.m, and winplt.m applications are worth reviewing since they each have a fairly rich GUI design with lots of opportunities for using various plt features in interesting ways.

# **SnapTo resolution**

You may have noticed that in repositioning mode, the objects when dragged don't move or resize smoothly, but rather move in steps of a fixed size. This makes it easier to align related objects and generally gives a more pleasing result. The default grid size is 100 by 100 which means that there are 100 useable positions inside the figure in both the x and y directions. This also means that if you are using normalized coordinates the third decimal place for all position vector elements will be zero.

There are three ways to bring up the SnapTo figure shown below:

- 1. Type plt move res in the command window. (This is the only method if you are not using a plot pseudo object).
- 2. First left-click on the delta button, followed by a right click on the same button.
- 3. It's easy to forget the sequence for method two, so you can also go to the plt menu in the menu bar. There you will see the option *"Reposition grid size"* which will bring up the SnapTo figure.



The default resolution is usually enough, but if you want finer control, move one or both of these sliders to the right edge (i.e. 200). This is nearly always enough,

although if you like you can type in a number bigger than 200 into either edit box. Or you can move the slider all the way to the left (i.e. zero) which disables the snap-to feature altogether.



# **Trace properties**

Right

You specify which traces should appear on the right-hand axis with the 'Right' parameter. For example if you included 'Right', [1 4:2:10 17] in the parameter list, then plt would put trace numbers 1,4,6,8,10, and 17 on the right axis and all other traces on the left axis. A slight shading is used behind the Trace IDs associated with the right hand axis so you can tell at a glance which traces belong to which axis. (You can disable this shading if you prefer. To see how, read the description of the TraceID parameter below). You can also tell which axis a trace is on by the shape of its cursor ('+' for left axis and 'o' for the right axis). You can optionally specify a label for the right hand axis (see LabelY) as well as the axis limits (see YlimR). Specifying an empty list, as in 'Right', [] tells plt to use the left axis for all the traces (the same as if you omitted the Right parameter altogether.)

The Markers parameter is a shorthand way of setting a different marker property for each line. For example:

```
plt(x,y,'Markers',s)
```

is equivalent to:

```
a = plt(x,y);
for k=1:length(a)
set(a(k),'Marker',s(k,:)); end;
```

The argument may be an array of characters or a cell array of strings. The latter method is easier when the elements are different sizes because you don't have to pad with blanks as with the character array. (Wherever a character array is allowed in a plt argument list, a cell array of strings is also allowed and visa versa.) For example, these two lines have give the same result:

#### Markers

```
plt(...,'Markers',['square';'+ ';'none
']);
plt(...,'Markers',{'square','+','none'});
```

This sets the marker for the first two lines to a square and a plus sign respectively while the third line will be rendered without any markers.

The following example shows two ways to set the markers of the six traces to x,+,square,o,asterisk,x (respectively). The shorter method used in the 2nd line is possible because every marker may be represented with a single character:

```
plt(...,'Markers',
['x';'+';'s';'o','*','x']);
plt(...,'Markers','x+so*x');
```

The Styles parameter is a shorthand way of setting the LineStyle property in a similar way that the Markers parameter is used to set the Marker property. For example, to set the first trace to normal, the 2nd and 3rd traces to dotted and dashed respectively, and the 4th trace to none (useful when you want the markers with no lines connecting them) you would use the following command:

```
plt(...,'Styles',{'-',':','--','none'});
```

The shorthand for single character styles mentioned above also works. For instance, to alternate between normal and dotted among eight traces one could use:

# plt(...,'Styles','-:-:-:');

One additional trick applies to the Styles parameter. If a single character is given which is not a valid line style, then the linestyle property is set to none and the given character is applied to the marker property. As an example, the following command defines eight traces of which the first four are rendered as continuous lines (i.e. without markers) and the last four are rendered with plus sign markers placed at each x,y location specified by the data arrays but with no lines connecting the markers:

## plt(...,'Styles','---++++');

Since there are no marker property values which can also be linestyle property values, there is never any ambiguity as to which property should be set.

This parameter allows you to select the grid line style. For example:

GridStyle

**Styles** 

#### plt(...,'GridStyle',':');

will select a dotted or dashed line (depending on the graphics renderer). If this parameter is not included the default is usually a solid line ('-') although there is one somewhat complicated exception to this which is described in the default section of the GRIDC parameter which you can find <u>here</u>.

This parameter allows you to assign a name to each trace. This name will appear in the trace selection box (also sometimes called the TraceID box). The number of characters that will fit in the trace selection box depends on the size you choose for the plt window. For the default figure size there is room for about 5 uppercase or 6 lowercase characters. In the example below, both forms are equivalent:

```
plt(...,'TraceID',['Rtemp';'Ltemp';'RV1
']);
plt(...,'TraceID',
{'Rtemp';'Ltemp';'RV1'});
```

```
Default: ['Line 1';'Line 2'; ... 'Line n'];
```

If you want the plot to be created without a TraceID box, call plt with a TraceID parameter of zero or the empty set ([] or "). Since plt can't create a TraceID box containing more than 99 IDs, if you want to plot more than 99 traces, you must include 'TraceID', 0 (or with the equivalent empty set value) in the parameter list.

When specifying traceIDs, you must have one trace ID for every trace on the main and right hand axes. However if you don't want a trace ID for a specific trace to appear, just use the null string ('') for the trace name. If you do that, the trace ID box will be made smaller to account for the fewer number of IDs displayed.

Normally traceIDs associated with the right hand axis will appear in the traceID box with a slight shading so you can identify those traces at a glance. If you want to disable this shading, insert the special character ']' at the beginning of the first TraceID name. The right bracket will be removed from the trace name before it is used. The third plot of the pub.m demo program demonstrates the use of this special character.

You may specify a callback function (fcn) to execute when the user clicks on any of the TraceID tags by including the parameter 'TIDcback', fcn in the argument list. If the string '@TID' occurs anywhere inside the function string then it's replaced with the handle of the trace ID string. Likewise if the string '@LINE' occurs anywhere inside fcn, it is replaced with the handle of the trace itself and occurrences of '@IDX' are replaced with the index of the selected trace. (i.e. 2 for the second trace listed in the TraceID box). See the

#### **TraceID**

demo program pltquiv.m for an example using the TIDcback parameter. In that example, the name and color of a trace is displayed in the command window when you click on a Trace ID tag. (Not particularly useful, but this example was contrived to demonstrate all the possible substitutions.) To define a quote within a quote in Matlab, one uses two single quote characters in a row. Since this can get confusing at times, callbacks defined within plt may use a double quote character instead of two successive single quotes. The pltquiv.m example uses this alternative form. In addition to a string, fcn may also be a function handle of the form @func or {@func, arg1, arg2, ..., argn}. Note that the string substitutions can't be used with the function handle form of this parameter.

You also may change the traceIDs after the plot has been created. For example, if the current figure contains a plot with four traces, these traces can be renamed with a command such as:

```
plt('rename',
{'First' 'Second' '3rd' '4th'});
```

If there are other changes you want to make to the TraceID box from your program (as in the **CURVES.m** example), you can get the handle of the axis that contains all the TraceID objects with the following command:

```
tbox = findobj(gcf, 'user', 'TraceID');
```

Then, for example the following command would make the TraceID box invisible:

```
set([tbox; get(tbox, 'child')], 'vis', 'off')
```

An easier way to make the TraceID box invisible would be to simply move it outside the figure area:

```
set(tbox, 'pos', [-2 0 1 1]).
```

Or in the unlikely event you wanted to reverse the order of the TraceIDs (i.e. bottom to top ordering in the TraceID box), use the command:

```
set(tbox, 'view', [0 270]).
```

TraceMK

This parameter allows you to show the line types in the trace selection box to help identify the traces. This can be visually pleasing and is especially helpful if you are color blind. If the argument is a vector, it specifies the marker positions within the trace selection box. For example

'TraceMK', [.6.7.8.9] would tell plt to place a horizontal line next to each TraceID label beginning and ending at x = .6 and .9 with markers at the four locations specified (assuming the line type in the plot included markers). The area between x = 0 and .6 (i.e. the first 60%) would be used for the text label. If the first element of the vector is less than .25 then plt will not display the text labels since there probably would not be room for them anyway. (Clicking on the lines in the TraceID box have the same effect as clicking on the labels, so the labels can be removed without loss of functionality). If the argument is a scalar, plt will use that value as the first element of a length 3 vector whose last element is .9. Thus 'TraceMK', .6 is shorthand for 'TraceMK', [.6.75.9]. A special case is when the scalar argument is zero, in which case no lines are inserted into the trace selection box (as if the TraceMK parameter was not used at all). See the demo programs trigplt.m and subplt.m for examples of using the TraceMK parameter.

All TraceIDs will appear in the trace selection box (aka TraceID box) in a single column except when the TIDcolumn parameter is included. This is useful when you are using so many traces that the TraceID box becomes too crowded to fit all the trace names in a single column. The simplest way to use the TIDcolumn parameter is to supply an empty argument TIDcolumn

to the parameter (i.e. ' ' or []). When this is done plt will use just a single column for the TraceID box when the number of traces is 24 or less. Two columns will be used when the number of traces is between 25 and 48, and three columns will be used when there are more than 48 traces. (The TraceID box will not appear when more than 99 traces have been defined). This default will probably work in nearly all situations but if you want exact control over how many columns are used and how many traceIDs appear in each column, you can do that by specifying a non-empty argument to the TIDcolumn parameter as follows: If **TIDCOlumn** is a scalar, it specifies the number of TraceIDs to put in the *second* column. If it is a vector, it specifies the number of TraceIDs to put in columns 2,3,etc, with the remaining going into column 1. For example, if 30 traces are displayed, and you use 'TIDcolumn', 8 then the first 22 TraceIDs appear in the first column and the last 8 appear in the second column. 'TIDcolumn', [5 5 5] would tell plt to arrange the 30 IDs in four columns as follows: (1-15, 16-20, 21-25, 26-30).

By default, all the traces defined by plt are visible until you change that from the trace selection box. You can change the default by disabling some traces from the plt call. For example:

```
plt(..., 'DIStrace', [1 1 0 0 0]);
```

# DIStrace

This tells plt to start the display with the first two traces disabled and the remaining 3 traces enabled. Of course you can later enable the first two traces via the trace selection box. If the parameter has fewer elements than the number of traces, it is extended by adding zeros. This means that we could have used [1 1] above to the same effect. After the call to plt has been made, if you want to change which traces are enabled/disabled you can click on the TraceIDs as described in Selecting traces. However if you want to do that from a program you can use the plt ( 'show', ...) command which is described at the very bottom of the

By default you will be allowed to cursor every visible trace in the plot area. You can change this default using this parameter. For example, if we had five traces, but wanted to use cursors only on traces 1,4, and 5 you would use: **ENAcur DIT**(..., 'ENACUR', [1 0 0 1 1]);
If the parameter has fewer elements than the number of traces, it is extended by adding ones. This means that we could have

used 'ENAcur', [1 0 0] above to the same effect.

This parameter allows you to reserve space for additional traces to be added to the figure after the plt window has been started. For example plt(x1,y1,x2,y2,'+',5); opens the plt window with two traces, the first one defined by x1,y1 and the second one by x2,y2. Then room is reserved in the TraceID box for up to 5 more traces that can be added using the pltt.m function. This parameter is normally only used inside script or function files because when you type the plt command in the Matlab command window an automatic '+', 8 is assumed. You could still include the + parameter from the command window in the unlikely event you were planning on adding more than 8 traces. When plt is called from a script or function, you can't add traces after the plt window has opened unless you had included the + parameter in the argument list.

It is unusual to want to add dozens of traces with the pltt function, but it is possible. For example with the command

plt(x,y, '+', 39, 'TIDcolumn', ''); plt will reserve space in the TraceID box for 40 traces. The first is specified in the plt command and the remaining 39 can be added using the pltt function. The TIDcolumn parameter was needed in this case because without it, plt would attempt to cram all 40 TraceIDs into one column which would probably be unreadable.

You may include the **TraceID** parameter in the argument list as well if you like, and you should be aware that there are two ways of doing this. The first (and by far the most common) way of doing this is to put the 'TraceID' parameter before the **'+'** in the argument list. When done in that order, that TraceID argument specifies the trace names only for the traces defined in the argument list. Then when the '+' parameter is encountered, plt expands the TraceID list using default names that will usually be overwritten by the trace names included in the calls to **pltt**. When done in the opposite order, the TraceID argument should include the trace names you want for the traces that will be added later (even though the trace names will be invisible until those traces are added). And if the TraceID argument does not include enough trace names for this, when a trace is added after the list has been exhausted, the new trace will be added without any corresponding entry in the TraceID box (which occasionally might even be what you wanted). Typically the + parameter is placed after all the traces defined inside the plt argument list, however this is not strictly necessary. In fact multiple + parameters may be included and they may be interspersed with the trace definitions in the parameter list. When you do that, the space reserved in the TraceID box for the traces to be added later will be interspersed with the defined traces in the order in which they appeared. This flexibility is rarely needed, but nevertheless it is available if you want it. Note that when traces are added with the pltt function, the reserved slots are used in order (top to bottom, as well as left to right if multiple columns were enabled).

You might expect that when all the free slots in the TraceID box have been used up, you can no longer add a new trace with the pltt function ... but in fact you can. What happens is that in this situation, pltt will overwrite the data and the trace name of the last trace that was added, so effectively you can never run out of free slots (unless you never allocated any in the first place).



# **Axis properties**

	<pre>plt(, 'Xlim', [xmin xmax]); Specifies the x-axis limits. If you are using a 2 column</pre>
Xlim	<pre>subplot, you can specify the x-limits for both both columns by using a cell array. i.e. 'Xlim', {[xminL xmaxL]; [xminR xmaxR]}; If you want to specify just the right column limits,</pre>
	replace the left column limits with the string 'default'.

plt(..., 'Ylim', [ymin ymax]);

Specifies the y-axis limits for the left-hand y axis of the main plot. Alternatively you may specify the limits for both the left and right hand y-axes of the main plot using a cell array as in:

#### Ylim

'Ylim', {[ymin ymax] [yminR ymaxR]}. The 'Right' parameter should also be included in this case, however if you don't, plt will default to placing the last trace on the right hand axis. Note that this parameter only specifies limits for the main plot and never for any of the other subplots. If you need to set the y-axis limits for the other subplots, use the set command with the axis handles obtained from getappdata(gcf, 'axis').

#### plt(..., 'YlimR', [ymin ymax]);

Usually the y-axis limits are specified using the Ylim parameter (above) however if you only need to specify **YlimR** the limits for right-hand y axis use the **YlimR** 

parameter. The 'Right' parameter should also be included, however if you don't, plt will default to placing the last trace on the right hand axis.

# plt(...,'xy',p]);

where **p** specifies new xy position/size coordinates for various graphical objects created by plt. **p** is a 5 column matrix in the following format:



OID1 thru OIDn (Object IDs) are integers that specify the objects (often an axis) to be resized and repositioned. x and y represent the coordinates of the lower left edge of the object and w and h specify the

width and height. (x,y,w,h may be in pixels or in normalized units i.e. as a fraction of the window size). The OIDs are described in the following table:

OID	
1	The main (left) plot axis

This is the only OID that refers to more than one object, i.e. both the main left and right hand axes. If there is no right hand axis, OIDs 0 and 1 are equivalent. Also this is

- 0 the default OID if none is given (which is only allowed if the OID parameter contains only a single row). This means that plt(..., 'xy', [x y w h]) is equivalent to plt(..., 'xy', [0 x y w h]).
- -1 Represents the axis containing the traceIDs.
- -2 Represents the axis containing the menu box items. This is similar to OID 0 except that in addition to adjusting the positions of the left & right axes, it also adjusts the associated cursor object positions and sizes (TraceID box, menubox, cursor readouts, etc.). For small axes, this can sometimes scale the cursor object to small or close to the axis so there is a way to define this

This usuall hand axis. I have been s represents t remaining s numbered s the next nu subplot is a hand axis. ( that the OII it's index ir getappda array since way.)

xy

scaling independently as follows:

-3	<pre>AxisSize = [.3 .3]; AxisPosition = [.2 .4];</pre>
	CursorSize = [.5 .5];
	<pre>plt(,'xy',[-3 AxisPosition</pre>
	AxisSize + CursorSize*1i]);
	The correction for the cursor size (using the imaginary
	component) may be applied in both x & y directions as
	in the above example, or it may be applied to either
	direction alone. Both the plt50.m and editz.m example
	programs demonstrate the use of the imaginary
	component in the y direction only.
	All graphical objects created by plt as well as those later created in the same figure window have a unique OID and therefore may be repositioned using the XV
other	parameter. To determine an object's OID enter the
0 11 101	repositioning mode by right-clicking on the delta cursor
	button. Then clicking on any other object will display its
	OID followed by its current position coordinates.

#### For example:

plt(..., 'xy', [-1 .01 .8 .12 .18; 1 .2 .16 .7 .8]); will set the traceID box to normalized position [.01 .8 .12 .18] and set the main axis to normalized position [.2 .16 .7 .8].

Although you can determine and enter these position coordinates manually, it is usually far easier to use the plt repositioning mode to determine the coordinates. See <u>GUI building with plt</u> to learn how this is done. That section also demonstrates in detail how to use the xy parameter to reposition any of the graphical objects in the plt figure window.

# AxisPos

# plt(...,'AxisPos',p);

Usually the size and position of the plot and TraceID box are modified using the xy parameter described above, however AxisPos provides an alternate method

that is included primarily for backwards compatibility with older programs written before the xy parameter was added. Although on rare occasions the AxisPos parameter may actually be easier to use than the xy parameter. **p** is a 4 element vector that modifies the size and position of the plot axis in the figure window. The first two elements modify the x and y coordinates of the lower left corner of the axis. The last two elements modify the axis width and height respectively. For example if **p** = [1 1 .9 1], the width of the plot will shrink by 10%. If **p** = [1 2 1 .8] then the space between the bottom of the figure window and the bottom of the x-axis will double and the plot height will shrink by 20%. Changing the size and position of the axis is often useful when building applications to make room for additional GUI objects. If **p** is a 5 element vector, the width of the trace ID box is increased by a factor of **p**(5) to allow longer trace names. If **p** is an 8 element vector, the position of the trace ID box (xLeft,yBottom,width,height) is multiplied by last four elements of **p** (i.e. **p**(5:8))

## axisCB

#### plt(...,'axisCB',s);

Evaluate string S when either the x or y-axis limits are changed. This callback function can be also be specified by the cursor command plt('cursor', cid, 'set', 'axisCB', fcn) which is described in more detail in the cursor commands section. At the top of that section there is

more detail in the <u>cursor commands</u> section. At the top of that section there is also a table that describes the string substitutions that plt performs on the string before evaluating it. These substitutions can make the callback more powerful while using less code. In addition to a string, **s** may also be a function handle of the form **@func** or **{@func**, arg1, arg2, ..., argn}. Note that the string substitutions can't be used with the function handle form of this parameter.

Note that if the function is defined as a string argument often consecutive single quote characters are required (quotes within quotes). In that case readability can be improved by replacing all sequences of two consecutive single quotes with a double quote character. For example 'disp(''ABC'');' could be written as 'disp("ABC");'. Note that this trick does not work for Matlab callbacks in general, but it does work for any callback defined within a plt(...) function call.

#### moveCB

#### plt(...,'moveCB',s);

Evaluate string **S** whenever the cursor is moved. This callback function can also be specified by the the cursor command

plt('cursor', cid, 'set', 'moveCB', fcn) which is described in more detail in the <u>cursor commands</u> section. The moveCB is not really an axis property, but is included in this section because of the parallels with the above axisCB parameter. As with the axisCB parameter, the string substitutions are performed before evaluation. You may use function handle forms as well if you don't need the string substitutions.

## ENApre

#### plt(...,'ENApre',[ENAx ENAy]);

ENAx or ENAy = 0 to disable metric prefixes on the x/y axis.

ENAx or ENAy = 1 to enable metric prefixes on the x/y axis (default). When metric prefixes are enabled plt will choose the best unit for the respective axis. As an example, suppose the x-axis label is 'seconds' and the x-axis data is  $[0\ 1\ 2\ 3\ 4\ 5]$ \*1e-8. With metric prefixes disabled, the x-axis tick-labels and cursor readout will be in scientific notation. With metric prefixes enabled, the xaxis label will change to "nano-seconds" and scientific notation will no longer be required making the graph and cursors far more readable. (Note: metric prefixes are not used on the right hand axis).

# AxisLink

#### plt(...,'AxisLink',m);

Tells plt to start with the left/right axes linked if m=1 or unlinked if m=0. For more details about linking the axes, see the <u>right hand axis</u> section.

+AxisProp

# -AxisProp <LabelProp >LabelProp .LabelProp ^TitleProp

If a property name is prefixed with a + or a - character then the property value will be applied to the left or right hand axis respectively.

If a property name is prefixed with a >, <, . or a ^ character then the property value will be applied to the left hand axis label, right hand axis label, x axis label, or the axis title respectively. Some examples:

# plt(..., '+Ycolor', [0 0 1], '-Yscale', 'Log');

In this example plt will assign the value [0 0 1] (blue) to the Ycolor property of the main (left hand) axis, and it will apply the value 'Log' to the Yscale property of the right hand axis. The plus and minus signs are called a property prefix characters and are required so that plt knows which axis you want to modify.

# plt(...,'>FontName','Lucida Handwriting');

In this example the font used for the right hand axis label is changed to Lucida Handwriting.

# plt(..., '+<. ^FontSize', 13);</pre>

This example shows that more than one property prefix character may be included in front of a property name. In this case, the font size for the left hand axis tick labels, the left y-label, the x-label, and the axis title are all increased to 13.

The example program demo\pub.m demonstrates the use of these prefix characters. Note that if a property name appears without one of these six leading prefix characters (+-<>. ^), then property value will be assigned to all the lines that have been defined so far in the argument list.

SubPlot

# plt(...,'SubPlot',v);

Normally plt puts all the defined traces on a single plot (which may have left and right hand y-axes) that fills most of the figure area. However there are two methods (each with their unique advantages) to create more than one plot in a single figure. The first method is by using the 'Fig' parameter which is described at the end of the Labels and figure properties section. The second method is to use the SubPlot parameter which is described here.

When the SubPlot parameter is used, all the plots in the figure will be arranged in either one or multiple columns. All plots in a column usually use the same xaxis which allow all the cursors in the column to move left or right together. (This is called the "Linked" mode). With the alternate mode (called "Independent") however, each plot even within the same column may have different x-axis values. The subplot in the lower left corner has a special designation (the main plot) since that is the only plot that includes a traceID box. Also some of the cursoring features are only available on the main plot (peak/valley finder, delta cursors, expansion history, the Mark/Zout/LinX/LinY tags, the x-axis slider, multi-cursors and the xView slider. (The 'Fig' parameter method doesn't suffer from any of these restrictions since each plot is a "main" plot, although linked cursors are not available with that method.) Each subplot however has its own y-axis cursor readout. These cursor readouts are easy to identify since its background color matches the trace and axis colors. The full panning and zooming features of plt are supported for each subplot. When any subplot is panned or zoomed in the x-axis, all the x-axis limits of all the other subplots in the same column are set to match the newly chosen values.

#### Single column

To create a single column of plots (all using the same x axis), the subplot parameter should consist of n positive numbers, where n is the number of plots desired. Each number specifies the percent of the area to be occupied by each plot (starting from the bottom). Normally the sum of the array should be 100, although if the sum is less than 100, there will be some unused space at the top of the figure. For example, 'SubPlot', [40 30 15 15] tells plt to create four plots. The bottom one (the "main" plot) will use 40% of the available height. The plot above that will use 30% of the height, and the remaining two will take 15% each. Each subplot except the main (lower) plot is normally assigned a single trace, with the last trace defined appearing in the upper most

axis, the second to last trace appearing in the axis below that, etc. For example, the command plt(1:50, rand(7,50), 'SubPlot', [40 30 15 15]) will create seven traces containing random data, with the first four traces displayed on the main (lower) plot (with a traceID box containing four labels) and the last three traces are displayed in the other three subplots. The example script demo\subplt.m demonstrates the use of single column subplots. Usually only the main plot may contain multiple traces, although the SubTrace parameter (see below) allows you to change this behavior.

#### **Dual column**

The example script demo\subplt8.m demonstrates the use of dual column subplots. To create two columns of plots, insert a negative number into the subplot argument. The number of entries to the left of the negative number indicates how many plots will appear in the left column, and similarly, the number of entries to the right of the negative number indicates the number of plots in the right column. The negative number itself specifies the width (in percent) of the left column. Some examples will help clarify this. In all the examples below, assume that  $y = [a \ b \ c \ d \ e \ f]$  where a through f are column vectors of the same length as x.

#### plt(x,y,'SubPlot',[100 -60 100],'Right',[2 3]);

The subplot parameters tells plt to create two plots both of which fill the entire height available in the plotting area of the figure. The left (main) plot fills 60% of the width with the second plot filling the remaining 40%. Since six traces are defined, the first five traces (a through e) appear on the main plot and the last trace (f) appears on the right plot. Since the TraceID parameter was not included, the TraceID box next to the main plot will contain the default trace labels (Line1 thru Line5). To label the traces more informatively, a parameter such as 'TRACEid', {'a' 'b' 'c' 'd' 'e'} could be added to the plt argument list. Since the 'Right' parameter was included, the main plot will include both right and left axes, with the 2nd and 3rd traces (b and c) on the right and the remaining three traces (a, d,e) on the left. The left and right axes will be separated by enough space to leave room for the axis labels, and this space will be increased when the 'Right' parameter is used so that there is room for an axis label on the right side of the main (i.e. left) axis.

#### plt(x,y,'SubPlot',[50 30 20 -55 70 20]);

In this example three plots will be created in the left column which fills 55% of

the width of the plotting area. The main plot on the bottom (containing traces **a** & **b**) fills 50% of the height, the middle plot (trace **c**) fills 30% and the top plot (trace **d**) fills the remaining 20% of the height. Two plots are created in the right column which fills the remaining 45% of the width. The lower of these (trace **e**) fills 70% of the height, and the upper (trace **f**) fills 20%, with the upper 10% remaining blank. Note that both traces in the main plot use the left-hand axis since no 'Right' parameter was given and no limits or labels were specified for the right-hand axis. (With this many subplots it's best not to use a right-hand axis since it makes all the subplots significantly narrower to make room for the right-hand axis ticks and labels.)

plt(x,y,'LabelX',{'meters' 'pascals'},'Ylim'{[0 5] [0 Even though the subplot argument is not included here, plt will split the plot horizontally as if you had included 'SubPlot', [100 50 100] in the argument list. This is because two different x-axis labels are specified with the 'LabelX' parameter and so plt recognizes that a second column is needed. The right column plot will contain trace **f** and the left column (main) plot will contain traces a thru e. Since two y-axis limits are specified, plt will put both left and right axes on the main plot. In this example the 'Right' parameter is not included, plt will default to putting the last trace of the main plot (trace e) on the right axis with the other four traces on the left axis. (Be careful not to confuse the concepts of the right and left axes of the main plot, with the right and left columns of subplots.) Also remember that the 'Ylim' parameter can't specify axis limits for a subplot. To set the y-axis limits for the subplots, use the set command with the axis handles obtained from getappdata(gcf,'axis') or use the plt('cursor', cid, 'set', 'position', p) command described here.

#### More than two columns

As you can see from the example script demo\subplt16.m you may use as many columns as you want. The negative numbers in the subplot parameter are used to separate the plots into columns. For example

'SubPlot', [50 50 - 30 50 50 - 30 50 50 - 30] specifies an array of six plots (2 rows and 3 columns). Each column is split 50/50 between the two plots. Since each of the 3 columns occupies 30% of the available plot width about 10% of the available width to the right of the last column will be blank (possibly to be filled in later with other graphic elements or controls). The width of the last column does not need to be specified. In this example, if the last

number (-30) was omitted, the last column would take 40% of the available width since plt wants to fill the whole plot area unless instructed otherwise.

# **Plot spacing**

By default, plt allows plenty of space between the subplots to allow for axis ticks and labels. Sometimes you may want to decrease the horizontal or vertical spacing so that you can fit more plots into a given space or to allow each plot to have as much area as possible. Or you may want to increase the spacing to allow room to add additional controls or graphic elements. It would be awkward to require an additional array the size of SubPlot to specify the desired row and column spacing, so this information is embedded into the SubPlot argument. This is done by using the integer part to specify the plot heights and widths (as described above) and by using the fractional part to specify the deviations from the default inter-plot spacing. Fractional parts from 0 to .5 indicate the default spacing should be increased. Fractional parts from .5 to .9999 indicate the default spacing should be decreased. This is best shown by example. Consider a slight change from the previous example:

'SubPlot', [50.02 50.97 -30.96 50 50 -30.01 50 50] The first two fractional parts (.02 and .97) tells plt to increasing spacing below the first plot by 2% and to decrease the spacing below the second plot by 3% (of the available plot height). The fractional parts of the two negative numbers (.96 and .01) tells plt to decrease the spacing to the left of the first column by 4% and to increase the spacing to the left of the second column by 1% (of the available plot width). At first this may seem confusing, but with a little practice you will find that the SubPlot parameter gives you complete flexibility of the subplot positioning. In the rare situations where you can't get the subplots positioned as desired, you can always use the 'XY' parameter to move or resize any or all of the subplots.

# Linked vs. Independent mode

The three sample scripts mentioned so far use the default "linked" mode which is intended to be used when all the plots in each column have the same number of elements and the same x-axis limits. When you move a cursor, all the cursors for the remaining subplots in the same column will be moved left or right so all the cursors in the column remain vertically aligned. Likewise, if you change the x-axis limits of any plot (by panning or zooming) then the x-axis limits of the remaining plots in that column will also change so that all the plots in the column share the same x-axis limits. Note that changes in one column will never
affect any of the other columns. When you don't want the cursors and x-axis limits to be linked in this manner, you should specify the "Independent mode" which is done by putting an "i" after the first SubPlot element. The sample script demo\subplt20.m demonstrates the use of the independent mode.

#### SubTrace

When using subplots, it is important to understand that the default behavior is to allow only a single trace on each subplot except for the main axis (lower left). The main reason for this is to allow plt to provide a simple cursoring mechanism which allows every trace to be cursored. However there are two situations where you may want to change this default behavior. The first is where cursoring is disabled (usually because the plot is to be used for publication instead of for data exploration). Since cursoring is not an issue, there is no reason to stick with the default behavior for assigning the traces to the axes. The second plot in the script demo\pub.m is an example of how the SubTrace parameter might by used in this situation. The second situation where you might want to use this parameter is when you plan on modifying the cursor behavior to make sense for the particular trace arrangement you have in mind. This requires a detailed understanding of plt's cursoring scheme are reasonably simple.An example of this second situation can be found in the script demo\weight.m.

There are two ways to use this parameter to assign the traces to the various subplots. For either method you must know how plt numbers the axes. Axis number one is always the main axis (lower left). Then axis two is the one directly above the main axis and axis three is the one above that, continuing to the top of the left column. Then the lowest axis of the second column (if it exists) is assigned to the next number, and continuing upwards as before. Finally after all the subplots have been assigned a number in this manner, the right hand axis of the main plot (if it exists) is assigned to the next higher integer.

Suppose for example, you have 4 axes and 9 traces and that you want to put two traces on each of the first 3 axes and then put the remaining 3 traces on the last axis. The first way to do this is to specify how many traces to put on each axis, i.e. 'SubTrace', [2 2 2 3]. Instead of specifying how many traces are on each axis, an alternate way to do this is to specify which axis each trace goes on.

So an equivalent to the previous parameter you could use

'SubTrace', [1 1 2 2 3 3 4 4 4]. Of course this second method is always going to be longer than the first method, so you would likely only use it if you needed to assign the traces to the axes in a different order, for example 'SubTrace', [1 2 3 1 2 3 4 4 4] (which is not possible to specify using the first method). plt will always be able to figure out with method you are using.



## Labels and figure properties

#### plt(...,'Title',t);

Inserts the title string **t** above the plot area. **t** may be a cell array to specify a multi-line title.

The Tex interpreter is used to render the string allowing entry of C and other special characters. If you don't want the Tex interpreter t used, include the string [TexOff] anywhere in the first line of tł (The [TexOff] string will be deleted from the title before display). Alternatively you could disable the Tex interpreter after the call to using the command:

#### Title

#### set(get(gca,'title'),'interpreter','none');

A set command similar to the one above may be used to change th fontsize or other title properties. The plot height is automatically s by the amount needed to make room for the title assuming the defisize. If you increase the title font size you may need to adjust the I using the XY parameter described in the <u>Axis properties</u> section.

If t is a number is will be converted to a string. For example plt(..., 'Title', {123 7.88}) will create a two line title with '123' as the first line and '7.88' as th second line.

Default: no title

plt(...,'LabelX',s);

#### LabelX

Uses string **S** as the x-axis label. If you are using subplots with tw columns, you may also specify the x-axis label for both the left an columns of plots by using a cell array:

plt(...,'LabelX', {'left x label' 'right x la

Default: 'x axis'

#### plt(...,'LabelY',s);

LabelY

Uses string S as the left hand y-axis label of the main plot. You ca specify both the left and right labels by using a cell array. For exar there are no subplots, 'LabelY', {'ab' 'cd'} is equivalent 'LabelY', 'ab', 'LabelYR', 'cd'. If there are subplots, th hand axis label must come last. For example with 3 subplots: plt(..., 'SubPlot', [50 20 30], 'LabelY', ... {'lower-axis' 'middle-axis' 'upper-axis' 'right-hand-axis'});

Default: 'Y axis (Left)'

#### plt(...,'LabelYR',s);



Uses string **S** as the right hand y-axis label. The 'right' paramshould also be included in this case, however if you don't, plt will to placing the last trace on the right hand axis. Note that using a ce argument to the 'LabelY' parameter (described above) is usuall convenient way to specify the y-axis label, and the 'LabelYR' parameter is primarily used in legacy code.

```
Default: 'Y axis (Right)'
```

FigName	<pre>plt(, 'FigName', f); Uses string f as the name for the plt figure window.</pre>
	Default: 'plt'

plt(..., 'Position', [xLeft yBottom height wid
plt(..., 'Position', [height width]);
Specifies the figure size and position on the screen in pixels.

Since 9 and 55 are the default values for xLeft & yBottom respect the second form above (with xLeft and yBottom omitted) is equivalent plt(..., 'Position', [9 55 height width])

If you prefer conciseness, you may use 'POS' as an abbreviation 'Position'.

If the height is specified as zero, plt will choose a height so that a along the x-axis is the same as a unit along the y-axis (i.e.if you pl circle, it would look like a circle and not an ellipse). If the width is specified as zero, plt chooses the width to meet the same condition can't specify zero for both the height and the width). If you resize figure window with the mouse, then the units along the x and y ax no longer be equal (and a plotted circle may appear to be an ellipsyou wish that the equal units property to be maintained even after figure window is resized, you should follow the plt command with command axis ('equal').

Position Pos

> If you specify the same position vector for more than one plt comm will add a small offset to all the figure window positions (except tl one) so that no two figures are exactly on top of each other. This fer makes it less likely that you will completely loose sight of one of t figures and also makes it much easier to select or move any figure mouse. If a second plt command specifies a position that differs fr first plt command by even one pixel, then this feature will not be  $\epsilon$

> *Default:* [9 55 700 525] (if sublots are not used). With subplyou add more columns of axes the default width increases from 7( maximum of 980. As you add more axes to a column the default h increases slightly from 525 to a maximum of 600.

#### plt(...,'HelpText',v);

HelpText

This parameter creates a HelpText pseudo object at the same time plt pseudo is being created. **v** is a string or cell array specifying th displayed text. See the <u>Pseudo objects</u> section for a description of format of the **v** argument. Also look at the following demo progra which use the HelpText parameter: curves, editz, gauss, pltquiv, su tasplt, trigplt, and wfalltst.

This parameter is used to force a group of plt figures to close when member of the group is closed. Consider the following sequence:

```
plt(x1,y1);
g = gcf;
plt(x2,y2,'Link',g);
plt(x3,y3,'Link',g);
```

This of course will create 3 plotting figures. Closing any one of th figures will also cause the other two to close.



CloseReq

The link parameter is ignored if it is empty. This makes it easier to figures created in a loop. For example, this loop creates five linked figures:

```
g = '';
for k=1:5
    plt(x{k},y{k},'Link',g);
    if isempty(g) g=gcf; end;
end;
```

The demo programs editz, tasplt, pub, and pub2 take advathis parameter.

This parameter specifies a function that will be run when the plt w closed. The argument may be:

- a string (as shown in the plt50.m example).
- a function handle (as shown in the gui2.m example).
- a cell array containing a function handle and its arguments (a in the wfall.m example).

Note that if the function is defined as a string argument often cons single quote characters are required (quotes within quotes). In that readability can be improved by replacing all sequences of two con single quotes with a double quote character. For example 'disp(''ABC'');' could be written as 'disp("ABC");'.
that this trick does not work for Matlab callbacks in general, but it
work for any callback defined within a plt(...) function call.

Normally plt opens a new figure window when it is called. In som situations you may want to tell plt to use a pre-existing figure inste (The most common reason this is done is to put more than one plo single figure.) This parameter tells plt to do this and specifies whic window should be used. For example, to open plt using figure nun you would use plt('Fig', 4, ...);. More often you will pro use plt('Fig', gcf, ...); which will open plt in the current Generally the plt parameters may be placed anywhere in the paran and in fact the Fig parameter is the only exception to this. The Fig parameter must be placed either as the first or the last parameter ir argument list. The Fig parameter is ignored otherwise. This restric due to lazy programming more than any other reason, and the rest might be removed in a later release.

There are two example programs (plt50.m and pub3.m) which described in the <u>Programming examples</u> section that demonstrate to of the Fig parameter to put multiple plots in a single figure. The fit (plt50) is oriented towards data exploration and takes advantage of generality of plt's cursoring system for both plots. The second one is oriented towards creating a figure for publication and so the cur have been disabled to create a clutter free result.

There is also another method to create multiple plots in a single fit this makes use of the subplot parameter. (This is demonstrated by subplt.m, subplt8.m, subplt16.m, subplt20.m, pub. pub2.m, pltmap.m, and weight.m programming examples). might not expect that there would be a need for two different meth achieving the same end, but it turns out that each of these methods their unique advantages. The subplot method is sometimes simple because all the plots are created with a single call to plt. The subpl method imposes significant restrictions on the plots, but in turn thi the cursor controls to be more compact which makes cursoring po a figure with many more plots than would be possible with the Fig method. The subplot method also provides an option for linking th

Fig

cursors of the plots in a single column. Most of the programming examples with multiple plots per figure would have been difficult impossible without the correct choice between the subplot and Fig parameter methods.



## **Options**

**HelpFile** 

plt(..., 'HelpFile', 'filename'); Specifies the left cl
actions of the Help tag:

If the filename is specified with complete path information the helpfile will be read from the specified location. If no path information is included, plt looks for this file on the Matlab path (except for compiled applications in which case plt looks for the fi in the same folder that contains the executable). The file extension must be included in the filename string since the extension determines which application is used to open the help file. (If you don't include an extension plt will assume that it is an executable command, and plt will simply call that executable when you click the Help tag.) The extension may be .html, .pdf, or .chm, or a file type that your operating system knows how to open. (Usually chm files are only supported on PC systems.) Assuming the help f is found, it will be opened when you left-click on the help tag. Als the file specified is a chm file, then it also may be followed by a to specifier which causes the chm file to open pointed at the chosen topic. (The examples plt.m and julia.m demonstrate how to specify chm topic.) If this parameter is not included, left-clicking on the h tag will open the default plt help file (plt.chm on Windows systems and the file **plt**.htm otherwise). If both plt.chm and plt.htm are not found, then one of the files xxx.chm, xxx.htm, **xxx**. pdf will be opened where "xxx" is the current figure name. none of those files are found, a warning message will appear indicating that no help files were found. The demo programs demo\plt5.m and demo\julia.m demonstrate the use of the HelpFile parameter to open a chm file at a specified topic.

plt(..., 'HelpFileR', 'filename'); Specifies the righ
click actions of the Help tag:

HelpFileR

The rules for finding the help file are the same as described above the HelpFile parameter. Assuming the help file is found, it will opened when you right-click on the help tag. If this parameter is n included, right-clicking on the help tag will open the default plt he file. Often the help tag left click will be used for help on the plt plotting package and the right click will be used for help on the currently running program. Or the roles of left and right clicks ma be reversed. The demo program demo\plt50.m gives an examp of using the HelpFileR option.

#### plt(...,'Options',s);

'Ticks'	Use tick marks (i.e. no grid lines)
'Menu'	Enable the figure menu bar
'xView'	Enable the xView slider
'Slider'	Enable the x-axis control slider
'Xlog'	use logarithmic scaling on the x axis
'Ylog'	use logarithmic scaling on the y axis
'multiCur'	Enable the multiCursor
'Nocursor'	Tells plt to hide all cursor objects. They m be re-enabled with the command: <plt('cursor',0,'set','vison'< pre=""></plt('cursor',0,'set','vison'<>
'Hidden'	Tells plt to exit with the plot figure as usubut leave the figure window hidden.
	Tells plt to use Matlab's line smoothing algorithm (anti-aliasing) for all traces. The line smoothing property may also be controlled from the cursor button group which is described in the <u>Cursoring</u> sectio

s is a string specifying one or more options. The options allowed  $\varepsilon$ 

	'Linesmoothing'	Be aware that line smoothing probably wi not work on versions of Matlab older than about 2008. Also the line smoothing property is ignored in version Matlab R2014b or later. I believe this is because t newer graphics engine introduced with R2014b is supposed to smooth the lines al the time. (It doesn't work on my five year old computer however, even with updated graphics drivers. I actually get better look plots using R2014a or earlier although I suspect with newer hardware R2014b will have the advantage.)
	'-Help' '+Help'	removes/adds the Help tag from the menu box
	'-Xlog' '+Xlog'	removes/adds the LinX/LogX tag from the menu box
Options	'-Ylog' '+Ylog'	removes/adds the LinY/LogY tag from the menu box
	'-Print' '+Print'	removes/adds the Print tag from the menu box
	'-Grid' '+Grid'	removes/adds the Grid tag from the menu box
	'-Figmenu' '+Figmenu'	removes/adds the Menu tag from the menu box
	'-Mark '+Mark'	removes/adds the Mark tag from the menu box
	'-Zout '+Zout'	removes/adds the Zout tag from the menu box
	'-Rotate' '+Rotate'	removes/adds the XYrotate (XY $\leftrightarrow$ ) tag from the menu box
	'-All' '+All'	removes /adds all menu box items

These options strings are case sensitive and in fact only the capital letters are significant. You can add whatever lower case letters,

spaces and other delimiters that you want to make the string more readable. For example suppose you wanted the display to initialize with the menu bar and multiCursor enabled and the grid lines off. Any of these commands would achieve that goal:

```
plt(..., 'Options', 'Menu', 'Options', 'multiCur
Ticks');
    plt(..., 'Options', 'Menu multiCursor
Ticks');
    plt(..., 'Options', 'MCT');
    plt(..., 'Options', 'M,C,T');
```

In addition to those options, suppose you wanted to remove the "Grid" tag from the menuBox. Then we would use something like one of the following:

plt(..., 'Options', 'Menu Cur Tick -Grid');
plt(..., 'Options', 'MCT-G');

You can also use a plus sign on the menu box tags if you would ra specify which tags to include instead of which tags to remove. For example, both of the following commands would remove all the menu box items except for the x and y axis lin/log controls:

plt(..., 'Options', '-H-G-F-M-Z-R');
plt(..., 'Options', '+X+Y');

Note that the Print menu box tag is unique in that it is off by defau and will only appear when +P appears in the Options string. If yc remove all menu box items (i.e. 'Options', '-A'), the box outline is not displayed as well.



## Colors

All the arguments below identified by **rgb** refer to a color specified in the usual Matlab way, i.e. as a 3 element row vector where each element is between zero and one. However for convenience, you may also use two alternative formats when specifying colors with plt.

- The first alternative is to specify the color values as percents instead of fractions. For example the Matlab color triple [.23.45.67] may also be written as [23 45 67]. Whenever you include a number bigger than one in a color triple, plt assumes that you are using this entry style.
- The second alternative is to specify the red, green, and blue components as a single number with two digits assigned to each value. For example the color triple shown above could also be written as 234567. For clarity, you may choose to always use six digits by using leading zeros when necessary, although the leading zeros are not required.

Using the second alternative shown above, you might expect that the largest fraction you can enter is 0.99 since only two digits are allowed for each color. However actually you can use 1.00 because the digits "01" (treated as a special case) is interpreted to mean 1.00. This means that you can't specify the fraction 0.01 using this entry method, although that will rarely if ever be a problem. So for example, a TRACEc argument that would specify a 10 color trace sequence using the first 10 colors shown in the default color list below, a concise way of specifying such an argument would be as follows:

```
plt(..., 'TRACEC', [100; 10001; 101; 10000; 206001; 10101; 16020; 303001; 12060; 200160])
```

plt(...,'TRACEc',ct);

Specify trace colors. Usually you will have at least as many row ct as there are traces, however if ct doesn't have enough rows, pl start over from the beginning. For example if you specify: TRACEC, [1 0 0; 1 1 0; 1 1 1] then plt will use red, yellow, and white respectively for traces 1,2, and 3. If you had a fourth trace, plt would use red for that trace and continue cyclica If ct is a single 3 element vector, plt will use that color for all tr If the TRACEc parameter is not included and color files are not being used (see 'ColorFile' below), then the following defa colors are used:

Default (for the first 40 lines):

ß

۰ ו

۲O

#### TRACEc

**CURSOR**c

LO	-	υ,	-	0	т,	0	-	±,	-	0	υ,	• 4	.0	
1	1	1;	1	.6	.2;	.3	.3	1;	1	.2	.6;	.2	1	
.8	1	.5;	.6	.2	1;	1	1	0;	Θ	.5	0;	.5	0	
0	.6	.6;	Θ	0	.9;	0	.3	.3;	.5	0	0;	.3	.3	
.7	.2	.2;	.2	.7	.2;	.3	.3	.3;	.4	.4	.7;	Θ	.6	
.8	.5	.5;	.4	.6	.3;	.5	.5	0;	.7	.7	.2;	.5	.5	
.7	.2	.7;	1	.4	.4;	.4	1	.4;	.4	.9	0;	.5	.1	
.3	0	.8;	Θ	0	.5;	.5	.5	1;	0	.3	.8;	.7	.2	

ß

۰

6

Q

You will find a picture showing what these 40 colors look like in <u>Default colors</u> section.

The defaults for lines 41 to 80 are the same as the colors listed a for lines 1 to 40 except that they are 26% dimmer. The defaults f the lines 81 to 99 are again 26% dimmer than the trace colors for lines 41 to 59. Default colors are defined only for up to 99 traces you have more than 99 traces, plt will start assigning colors cyclically as described above.

#### plt(...,'CURSORc',rgb);

Specify the cursor color. Here, and on the rest of this page, rgb 1 be a 3 element row or column vector.

Defaults:

Only one trace (dark plot backgrounds): [1 1 .5] Only one trace (light plot backgrounds): [0 0 .5] More than one trace: *Cursor color is set to match trace cole* 

	plt(
DELTAc	Specify co
	Default:

plt(..., 'DELTAC', rgb); Specify color indicating the delta cursor. Default: [1 0 0]

PltBKc	<pre>plt(, 'PltBKc', rgb); Specify the plot area background color. Default: [0 0 0]</pre>
--------	---

FigBKc	<pre>plt(, 'FigBKc', rgb); Specify the figure window background color. Default: [.25 .15 .15]</pre>
--------	---

хуАХс	<pre>plt(, 'xyAXc', rgb); Specify the color of the x and y axes. Default: [1 1 1]</pre>
-------	---

xyLBLc	<pre>plt(, 'xyLBLc', rgb); Specify the color of the x and y axis labels. Default: [.64 .78 .94]</pre>
--------	---

#### plt(...,'GRIDc',rgb);

Specify the color of the grid lines. (See also <u>GridStyle</u>.) Normall grid lines are drawn in exclusive-or erase mode, however if any the rgb values are negative (e.g. 'GRIDC', [0 - .2 .4]), the the grids are drawn in normal mode (which is often preferable especially if the right hand axis is not being used). The actual gri color used in the above example is [0 .2 .4]).

#### Default:

If no right hand axis is enabled then the default is [-.13 .13 .13] i.e. a very dim gray line using normal erase mode. (The gi lines will be solid unless the <u>GridStyle</u> parameter is included). If

#### GRIDc

TIDc

right hand axis is enabled the defaults (again assuming that the GridStyle parameter is not specified), depend on the Matlab vers For Matlab versions earlier than R2014b the default GRIDc is [.13 .13 .13], and the default GridStyle is - i.e. a very dir solid gray line drawn using xor erase mode. For Matlab version R2014b or later the default GRIDc is [-.26 .26], and default GridStyle is : i.e. a dim gray dotted line in normal erase mode (which actually is the only erase mode supported in the ne Matlab versions).

The defaults described above apply only if a very dark plot background is being used, including of course the default plot background (black). If a bright background has been selected (by using either the 'PltBKc' or 'COLORdef', 0 parameters) th the defaults above are inverted (i.e. subtracted from one). These defaults may sound complicated but they have been chosen to be pleasing to most people under the various circumstances.

#### plt(...,'TIDc',rgb);

Specify the background color of TraceID box. *Default:* If this parameter is not included, the color specified by PltBKc parameter (or its default) is used.

#### plt(...,'COLORdef',c);

Sets the PltBKc, FigBKc, xyAXc, and xyLBLc colors mentioned above to be consistent with Matlab's current default colors. *Trace colors are set as follows:* 

c=0or'default'	<pre>Matlab's current default trace colors are used. i.e. TRACEC = get(0, 'DefaultAxesColorOrde</pre>
 c is a 3 column array	c is used instead of Matlab's default trace color order. Note that you may use the traditional Matlab color triple entry or or of the alternate styles described above. T example program subplt8.m

COLORdef		demonstrates how to use both the traditic and alternate color entry styles for this argument.
	c is a 3 column array with a special first entry [.99 .99 .99]	The special case for the first entry (whicl 999999 if using the alternate style) mean that the remaining colors in the array wil appended to the current Matlab default tu colors. This is useful if you like Matlab's default colors, but the color sequence is 1 quite long enough and you just want to a few colors to that sequence.

If TRACEc so defined has fewer rows than the number of traces plotted the colors will be used cyclically as described above.

#### plt(..., 'ColorFile', 'filename');

The normal behavior of plt (i.e. when the **ColorFile** argumen not included) is as follows.

- If plt is called from the Matlab command line, when plt initializes it looks for a file called pltcolor.mat in the folder where you installed plt. If this file isn't found, no acti taken. If it is found, the file is loaded causing all the default colors and the colors specified by all the other parameters shown on this page to be overwritten with the data saved in .mat file. When you select the "Save figure colors" selection under the Color menu, plt saves the current colors to this sa file (pltcolor.mat).
- If plt is called from a Matlab function or script file, the behasis similar except that the file name used is fnameColor.1 where fname is the name of the top level Matlab command script. Again, plt looks for this file in the folder containing .m file defining fname (or in the folder containing the fnam file for compiled applications).

When the **'ColorFile'** parameter is included in the plt argun list, the above behavior is modified as follows:

ColorFile	'ColorFile','filename'	If the filename is specified without any path, plt uses the specified file name both on startup (to load the color sche and when "Save figure colors selected. plt looks for this file the folder containing the .m fi defining fname (or in the fold containing the fname.exe file compiled applications). The extension used is always .ma and you may not include an extension or a period in the filename string.
	'ColorFile', 'drive:\path\filename'	As above, plt uses the specific file both on startup (to load th color scheme) and when "Sav figure colors" is selected. However when path informati is supplied, plt will look for th file only in the location given before, you may not include a extension in the filename strir
	'ColorFile',''	plt will use its default colors, any colors specified by the otl parameters in this section. A color file will never supersede them on startup, even if <b>fnameColor.mat</b> exists. I: you select "Save figure colors plt will allow you to save the current colors to any file of yc choosing (by opening a file selection box), but these color will not be loaded unless you change to one of the above 'ColorFile' options, or

remove the 'ColorFile'
parameter altogether.

t



### **Cursor commands**

The cursor pseudo object, the most complex pseudo object supported by plt, consists of one or more markers and several uicontrols with complex interactions. Because of this you normally will not create cursor objects on your own, but will rely on plt to create them automatically when you specify the data to be plotted. However it is possible to create a cursor object manually with the cursor init command described at the end of this section. For many casual plotting applications, the plt cursors will behave as desired out of the box. For more sophisticated applications you may want to modify the cursoring behavior using the plt parameters (DualCur, Xstring, Ystring, Options) or the independent cursor commands shown below.

#### **DualCur parameter:**

Normally the cursor value for only a single trace (referred to as the active trace) is shown in an edit box below the plot. However sometimes it is convenient to show the y-value for two traces simultaneously. This is done by using the **'DualCur'** parameter which specifies a trace which will always have a display of its y-value on the screen in addition to the display of the active trace. Try out the editz.m demo program which uses Dual Cursors to simultaneously display the magnitude and phase of a transfer function. (In fact, transfer function displays were the problem that inspired the Dual Cursor feature.) The alternate method of specifying the dual cursor trace (as an offset from the active trace) is indicated by using a negative number for the DualCur parameter. The demo program **gui2.m** demonstrates the use of that mode. The use of the DualCur mode is covered in more detail near the end of the cursoring guide found here: Cursoring.

#### **Xstring and Ystring parameters:**

The 'Xstring' and 'Ystring' plt arguments allow you to add text strings

just to the right of the cursor X or Y readout values. Since these strings occupy the same screen area as the delta cursor readouts, they get covered up when you are in delta cursor mode (or a if a zoom window is visible). However those are usually temporary modes, so as you will see, these strings still prove useful.

Most of the power of the
Xstring and Ystring
parameters stem from their
string replacement feature
described in this table.
Strings in the first column
of the table are replaced
with the value shown in the
second column. (The
<i>Xstring and Ystring are both</i>
updated every time the
cursor is moved.)

String	Replacement value
@CID	cursor ID
@XVAL	active cursor X position
@YVAL	active cursor Y position
@XY	<pre>same as complex(@XVAL,@YVAL)</pre>
@IDX	active cursor index
@HAND	handle of active trace
@LNUM	line number of active trace
@XU	Xstring user value
@YU	Ystring user value

#### **Xstring/Ystring Examples:**

Suppose it was important to see the cursor index as well as the usual cursor x and y values (i.e. you want to know that your are looking at the sixty fifth data element for instance). You could do this as follows:

```
plt(x,y,'xstring','sprintf("index = %d",@IDX)');
```

A string within a string (such as the 'index = %d' above) is normally written in Matlab using two consecutive single quote characters on both sides of the string. Since this can get verbose and confusing at times, callbacks defined within plt may use a double quote character instead of two successive single quotes. That's why the double quotes appear in the line above.

Although the mean of the active trace y values can be shown using one of the usual cursor features, suppose you wanted to display the mean of the entire data set (independent of the viewing window). Suppose also that you want to continuously display the y/x ratio. (This ratio is also a standard cursor feature, but its not continuously visible.) You could accomplish both of those feats as follows:

```
plt(x,y,'xstring','sprintf("mean:
%f",mean(get(@HAND,"y")))',...
'ystring','prin("Y/X: %5w",@YVAL/@XVAL)');
```

Suppose your x axis is measured in seconds with a zero reference of 5pm, 21-Jan-2007 UTC. The cursor x-axis readout will be in seconds past the reference, but you may want an additional cursor readout that shows the actual time of day. This can be accomplished as follows:

```
plt(t,y,'xstring',...
'sprintf("utc:%s",datestr(datenum("21-Jan-07
17:00")+@XVAL/86400,13))');
```

Note the 86400 (the number of seconds in a day) is needed because date numbers are measured in days. If your x axis unit was "weeks", you would replace /86400 with \*7. If you removed the , 13 near the end of the line (date string format), then the readout would show the complete date and time instead of just the time. Another way to code the statement above is:

```
plt(t,y,'xstring','sprintf("utc:
%s",datestr(@XU+@XVAL/86400,13))');
set(findobj(gcf,'tag','xstr'),'User',datenum(2007,1,21
```

The second statement puts the reference time in the Xstring user value which is used by plt when updating the Xstring. This method is much more convenient when the reference time can change. Note that the reference time is identical to that used above, although it's written in the vector format instead of the character format.

Sometimes the 1 second resolution provided by datestr is not sufficient. You can increase this resolution to 1 millisecond by using the date string function provided by plt as follows:

```
plt(t,y,'xstring','sprintf("utc:
%s",plt("datestr",@XU+@XVAL/86400,13))');
```

Occasionally its useful to use an edit box instead of a string for one or both of

these customized cursor controls. (The pltn.m example does this for the Xstring, although the Ystring is still rendered as a text string.) To do that, simply insert a question mark before the string. The first example above is rewritten below to use an edit box.

#### plt(x,y,'xstring','?sprintf("index = %d",@IDX)');

#### **Cursor commands**

Notes:	The <b>cid</b> (cursor ID) that appears in all the commands shown below is an integer that identifies the cursor the command is to act on. This integer is returned from the cursor initialization command used to create the cursor. If an axis contains a cursor, its cid is saved in the axis user data. (The cid stored in the axis user data is always a scalar since an axis may only contain a single cursor object.) You can specify that the cid should be retrieved from the axis user data by specifying a zero for the cid. So for example the following two commands have the same effect:
	plt('cursor',0,'set','visON') plt('cursor',get(gca,'user'),'set','visON')
	The figure 'cid' application data variable contains a vector with the cursor IDs for all the cursor objects in the figure. You can specify that the cid should be retrieved from this vector by supplying a negative number as the cid (for example -2 specifies the 2nd element of this vector). This means that the following two lines have the same effect: xy=plt('cursor', -2, 'get');
	<pre>c=getappdata(gcf,'cid'); xy=plt('cursor',c(2),'get');</pre>
	All the following commands are case sensitive (unlike all the other plt parameters previously described) and must use the exact case shown below.
	All the cursor commands below may return up to two arguments. If the return arguments are listed for a cursor command, the return values will be as specified. However if the return arguments are not listed for a particular command, the first return value (if requested)

#### [xy k] = plt('cursor',cid,'get',n);

Get x and y coordinates of the cursor location the last time it was on trace #n. The trace number is optional - if it is not specified then the position of the active trace is returned. xy is a complex value. Its real part is the cursor x-coordinate and its imaginary part is the y coordinate. The second return value (if requested) is the index into the x data vector of the cursor position.

#### [n h] = plt('cursor',cid,'getActive');

Returns the line number of the active cursor. The second return value (if requested) is the handle of the active trace.

#### h = plt('cursor',cid,'obj');

Returns an 13 element vector of handles to the following cursor objects:

1: x label	<b>5:</b> y cursor readout	<b>09:</b> marker line-style button	13: cursor marker
2: y label	<b>6:</b> y cursor expansion	<b>10:</b> delta button	
<b>3:</b> x cursor readout	7: peak button	<b>11:</b> expansion box	
<b>4:</b> x cursor expansion	8: valley button	<b>12:</b> delta cursor	

#### u = plt('cursor',cid,'expHis');

Returns an array containing the display expansion history. Each row contains one display expansion as [xmin, xmax, ymin, ymax, code] where:

- code = -1 indicates the row was not used
- code = 0 indicates a valid display expansion
- code = 1 indicates the current display limits

# plt('cursor',cid,'visON'); plt('cursor',cid,'visOFF');

Shows or hides the following objects:

• peak/valley/delta cursor buttons

- active trace cursor
- auxiliary trace cursor (dual cursor)
- x and y axis edit boxes and respective labels
- xstring and ystring objects
- x-cursor slider
- cursor id string

Note that this function is invoked alternately (visOFF/visON) when you rightclick on the plot y-axis label (which also hides/shows the menu box).

```
plt('cursor', cid, 'aux', 'on');
plt('cursor', cid, 'aux', 'off');
Shows or bides the auxiliary (dual) cursor and its of
```

Shows or hides the auxiliary (dual) cursor and its edit box

#### plt('cursor',cid,'setObjPos',p);

Sets the cursor object positions to p, where p is a 9 by 4 element array. Each row contains (x,y,width,height) which represents the position and size of the following objects:

- 1. x-axis edit box label
- 2. y-axis edit box label
- 3. x-axis edit box (cursor readout)
- 4. x-axis cursor expansion edit box
- 5. y-axis edit box (cursor readout)
- 6. y-axis cursor expansion edit box
- 7. peak button
- 8. valley button
- 9. delta cursor button

Note that this command does not set the position of the optional x-axis control slider. However you can set this position using the plt 'xy' parameter, or with a command such as:

```
set(findobj(gcf,'tag','xslider'),'position',p);
```

#### plt('cursor',cid,'xlim',p);

Set new x axis limits and update expansion history, where p=[xmin, xmax]

#### plt('cursor',cid,'ylim',p,pAux);

Set new y axis limits and update expansion history, where p=[ymin, ymax]. and optionally pAux=[ymin, ymax] (for the right hand axis).

#### plt('cursor',cid,'xylim',p,pAux);

Set new x and y axis limits and update expansion history, where p= [xmin, xmax, ymin, ymax]. and optionally pAux=[ymin, ymax] (for the right hand axis).

#### plt('cursor',cid,'exRestore',u);

Restores an expansion history previously saved in u.

#### plt('cursor',cid,'axisCB',fcn);

String **fcn** will be evaluated whenever an axis limit is changed. This cursor command overwrites any axis callback function entered using the **'axisCB'** parameter on the plt command line. The rules for string substitutions and function handles are the same as mentioned below in the **moveCB** command.

#### plt('cursor',cid,'moveCB',fcn);

String **fcn** will be evaluated whenever the cursor is moved. Before the fcn string is evaluated all occurrences of the strings in the 1st column of the table above (@CID, @XVAL, @YVAL, @XY, @IDX, @HAND, @LNUM, @XU, @YU) are replaced with the values in the 2nd column of that table. fcn is not called by events initiated from outside the figure window containing the cursor. (For example a button push that moves the cursor in another figure window would not activate the callback. This prevents infinite loops when figure A modifies figure B's cursor and visa versa.) If you do want to enable the callback for external events, insert an extra semicolon as the first character of the moveCB callback string. This cursor command overwrites any axis callback function entered using the 'moveCB' parameters on the plt command line. In addition to a string, fcn may also be a function handle of the form @func or *{*@func, arg1, arg2, ..., argn*}*. Note that the string substitutions can't be used with the function handle form of this parameter. Also note that a similar callback is provided for the TraceID fields, although the string substitutions allowed are different than the ones mentioned above. See the 'TIDcback' parameter under Trace properties.

#### plt('cursor',cid,'moveCB2',fcn);

This call operates similarly to the set **moveCB** command shown above and the functions specified in both these calls are executed whenever the cursor is moved. However normally you will not want to use this call because the **moveCB2** function is used internally by plt to keep the CursorID tag (just to the

left of the y-axis cursor readout) so that it always identifies the cursored trace name. It's also used by plt in the linked subplot mode to keep the plots in a column synchronized. In rare situations you may wish to modify those behaviors, which you can do with the **moveCB2** function.

#### plt('cursor',cid,'setActive',a,k);

Switches the active cursor to the line specified by **a** (a must be an integer between 1 and the number of lines in the plot). The cursor will be placed at index **k**. If k (optional) is out of bounds or not supplied, then the cursor will be placed in the center of the array. When **a** is zero (a special case), the active line remains the same and only the cursor index is changed - which would have the the same as calling the update command (below).

# plt('cursor',cid,'update',k); plt('cursor',cid,'updateH',k); plt('cursor',cid,'updateN',k);

Moves the active cursor to index k in the data set and calls any user defined cursor callbacks (moveCB, xstring, ystring). If k is out of bounds, the cursor is set to the middle of the array associated with the active trace. If you do not supply the argument k, then the command does not move the cursor, however it does execute the cursor callbacks. If update moves the cursor to an area that is not inside the current axis limits, it will shift the axis limits to make the cursored data element visible. However when the cursor is moved by updateH the axis limits will never be adjusted. (Think of this as "Update, Hold".) Also updating the cursor with the updateN command has the same effect as using the updateH command except that the cursor callback function (defined by 'MoveCB') is not called like it is with the update and updateH commands.

When the index is not needed we can abbreviate the update command by omitting the 'update' string. This means that the following two lines are equivalent:

```
plt('cursor',cid);
plt('cursor',cid,'update');
```

We can abbreviate the update command even more by omitting the cursor ID which defaults to -1. This means that the following two lines are equivalent:

```
plt('cursor');
plt('cursor',-1);
```

And finally there is one more variant of the update and updateH commands:

```
plt('cursor',cid,'update',k,x,y);
plt('cursor',cid,'updateH',k,x,y);
```

The moves the active cursor to the index k as above. Normally x and y would be the position of the kth element of the array associated with the active trace, in which case this command behaves the same as if you didn't include the last two parameters. However x and y can be any position on the axis, and the visible cursor marker will be moved to those coordinates. (It's rare to want to move the cursor off the line, but it may sometimes be useful.)

```
plt('cursor',cid,'peakval',0);
plt('cursor',cid,'peakval',1);
```

Moves the active cursor to the next peak (0) or to the next valley (1)

```
plt('cursor',cid,'clear');
```

All the cursor objects are deleted.

#### plt hideCur;

Has the same effect as right-clicking on the y-axis label. See <u>cursoring</u>. If you also want to hide the TraceID box, use the commands: tbox=findobj('user','TraceID'); set([tbox; get(tbox,'c|

Note: The following cursor commands were designed primarily for plt internal use, although sometimes they may also be useful in your programs. (These commands are case sensitive.) The "0" in the first seven commands below refers to the current cursor. You may replace the "0" with the actual cursor ID number, or "-n" to refer to the nth cursor.

plt	cleft	0	ZoomOut;	Zoom out both x & y axis by 40%.
plt	cright	0	ZoomOut;	Zoom in both x & y axis by 40%. With the functional form (which applies to the command above as well), you may also include an additional argument which specifies the zoom ratio. For example, this command specifies a 20% ratio (half of the default amount): plt('cright', 0, 'ZoomOut', .2);

plt	<pre>cleft 0 peakval 0;</pre>	Move the cursor to the next peak. (The last argument may be omitted in this case.)
plt	cleft 0 peakval 1;	Move the cursor to the next valley
plt	cleft 0 peakval 2;	Reset the peak finder (i.e. move the cursor to the highest peak)
plt	cleft 0 peakval 3;	Reset the valley finder (i.e. move the cursor to the lowest valley)
plt	<pre>cleft 0 TGLlogy;</pre>	Toggle the y-axis between linear/log
plt	<pre>cleft 0 TGLlogx;</pre>	Toggle the x-axis between linear/log
plt	<pre>cright 0 TGLlogy;</pre>	Open Hardcopy menu
plt	<pre>cright 0 TGLlogx;</pre>	Swap x & y axes
plt	<pre>cleft 0 markCB;</pre>	Toggle the delta cursor mode on or off
plt	cleft 0 mlsCB;	3 way toggle of all traces between markers only, lines only, and both lines & markers
plt	cleft 0 mark;	Adds a text label identifying the current cursor location
plt	<pre>xleft TGLgrid;</pre>	Toggle between grid lines and ticks
plt	<pre>xright TGLgrid;</pre>	Toggle between default and alternate grid style
plt	<pre>xleft TGLmenu;</pre>	Toggle the menu bar on/off
plt	<pre>xright TGLmenu;</pre>	Open a cursor data window
plt	<pre>xleft mark 2;</pre>	Open a window allowing editing plt figure colors
plt	xleft mark 3;	Write a file saving the current plt figure colors
plt	xleft EDIT 1;	Enter data editing (using last used editing mode)
plt	<pre>xleft EDIT 2;</pre>	Open up data edit y-popup
plt	<pre>xleft EDIT 5;</pre>	Exit data editing mode
plt	xleft Yedit 1;	Open a window allowing editing the line properties of cursored trace. (The command plt xright mark;) also does the same thing.
plt	xright Yedit 1;	Open a window allowing editing the plt figure properties

```
plt xleft Yedit 2;
plt xleft Yedit 3;
plt xleft Yedit 4;
plt xleft Yedit 4;
plt xleft Yedit 5;
plt xleft Yedit 6;
plt xleft Yedit 7;
plt xleft Yedit 7;
plt xleft Yedit 8;
plt xleft Yedit 9;
plt xleft Yedit 10;
plt click Yedit 10;
plt xleft Yedit 11;
plt xleft Yedit 12;
plt xleft Yedit 13;
plt xleft link;
```

plt xleft RMS;

Toggle multiCursor mode Toggle xView slider Cancel data editing mode Enter data edit mode (Range) Enter data edit mode (Range left/right) Enter data edit mode (Range up/down) Enter data edit mode (Insert) Enter data edit mode (Insert left/right) Enter data edit mode (Insert up/down) Enter data edit mode (Modify) Enter data edit mode (Modify left/right) Enter data edit mode (Modify up/down) Toggle right hand axis link status

Equivalent to clicking on the cursorID tag which rotates between the five cursor modes [normal, Avg, RMS, y/x,  $sqrt(x^2+y^2)$ ]. After five of these commands the cursor mode will be the same as it was before the first of those commands (having rotated thru all the modes).

#### **Mouse motion functions:**

If you create a figure with a plt command that includes the parameter 'MotionZoom', 'funcname', then if you create a zoom box (see The expansion box) while you are adjusting the size of the zoom box the function funcname([x1 x2 y1 y2]) will be continually called as the mouse is moved (i.e. for as long as the mouse button is held down). The coordinate [x1 y1] is the position of the lower left corner of the zoom box and [x2 y2] is the coordinate of the upper right corner. It may require some imagination to see how using such a parameter would enhance your user interface. The example demo\gauss.m shows how to use the 'MotionZoom' function. Although the use of the MotionZoom feature in this program is not inspirational, at least when you create a zoom box inside the gauss figure you will see the effect that the MotionZoom parameter creates. A more practical demonstration of the use of this parameter can be seen in the pltmap.m example.

In place of the character string 'funcname' you may also use @funcname or to insert extra parameters to the function, use {@funcname param1 param2}. (The 4 element vector specifying the zoombox corners will be the 3rd parameter of the function in this example.) These alternate forms also apply to the other mouse motion functions

Including the parameter 'MotionZup', 'funcname' has a similar effect except that the function 'funcname' only is called when the mouse button is released. The MotionZoom and MotionZup functions are called when the zoom box moved or resized as well as when it is first created.

If you create a figure with a plt command that includes the parameter 'MotionEdit', 'funcname', then if you use the data editing feature (see Data Editing) while you are modifying a data value by dragging it with the mouse, the function funcname(a) will be continually called as the mouse is moved (i.e. for as long as the mouse button is held down). The parameter "a" is a nine element cell array containing information related to the edited trace. The first six of these might be useful:

- a{1}: Cursor ID
- a{2}: index of edited point
- a{3}: edit cursor handle
- a{4}: edit cursor shape (index)
- a{5}: edit cursor marker size
- **a{6}:** edit cursor line width

As with the MotionZoom parameter, it may require some imagination to see how to use it to enhance your user interface. The example <u>demo\editz.m</u> shows one way to use the MotionEdit function to enhance the user interface. In this example you can see the advantage of the MotionEdit function and how useful (and impressive) it is when the the plots are updated while the data is being edited, instead of afterwards. The pltquiv.m example also demonstrates the use of the MotionEdit parameter to update the polynomial interpolation of a vector field in real time.

You may also create, modify, or remove these mouse motion functions after the

call to plt by modifying the corresponding application data variable associated an axis. For example these commands will set the mouse motion functions as expected:

```
setappdata(gca, 'MotionZoom', 'funcA');
setappdata(gca, 'MotionZup',@funcB);
setappdata(gca, 'MotionEdit', {@funcC param1});
```

#### **Creating a cursor pseudo object:**

Usually the cursor objects are initialized from the main plt() call that specifies the data arrays to be plotted. However you may also create the cursor objects using this cursor 'init' call after creating a figure on your own (i.e. without using the plt pseudo object).

## Ret1 = plt('cursor',axis,'init',In1,In2,In3,In4,In5,In6,In7,I

where:

- Ret1: the cursor ID (cid) used to control the cursor with additional calls to plt.
- axis: a scalar if the cursor is assigned to a single axis. A two element row vector is used to assign the cursor to a pair of axes. The second axis (right hand axis) is normally overlaid on top of the primary axis and is used to provide a separate y-coordinate axis on the right and side of the graph.
- In1: a 10 by 4 array containing the positions of the cursor controls. Each row contains [x y w h] where x,y is the position of the control and w,h is its size. The units for all the values in the In1 array must be either pixels or normalized (no mixing units). The last row specifies the position of the slider and is the only optional row. If In1 contains only nine rows then the x-axis cursor slider will not be created. The rows of In1 are assigned as described in this

Row	[x y w h]
1	label for x cursor readout
2	label for y cursor readout
3	x cursor readout
4	x cursor expansion
5	y cursor readout
6	y cursor expansion
7	peak find button
8	valley find button
9	delta cursor button
10	x-axis cursor slider (optional)
table.

• In2: a 3 column array [red green blue] specifying the colors for the cursors and cursor readout text. The rows of In2 are defined as shown in this table.

<sup>[\*1]</sup> Set to [0 0 0] to make the color of this element track its associated line color.

Row	[r g b]	
1	x/y cursor label color	
2	cursor readout color <sup>[*1]</sup>	
3	expansion box color	
4	delta cursor color	
5	line #1 cursor color <sup>[*1]</sup>	
6	line #2 cursor color <sup>[*1]</sup>	
	[*1]	
4+n		
(Must have a row for each line object	line #n cursor color <sup>[*1]</sup>	
that has a cursor)		

- In3: Text for the x/y cursor labels. ['xlabel'; 'ylabel'] or {'xlabel', 'ylabel'}
- In4: Cursor markers. Must have one marker for each line that has a cursor.

For example for 3 lines one could use **'+ox'** 

or to use the same symbol for all 3 cursors use '+++'.

- **In5**: Cursor size (in points)
- In6: Format strings for x and y cursor readouts.
   e.g. ['%2.1f'; '%5w '] or {'%2.1f', '%5w'}.
   (Type "help prin" for a description of these format strings.)
- **In7**: Visible flag (first optional argument). 'on' or 'off'.
- **In8**: (optional) Monotonic flag.

Set to 1 if the x data is monotonically increasing, and 0 otherwise (such as with Nyquist plots).

• In9: (optional) Axis limit change callback function. Executed when the axis limits are changed.

When the plt('cursor', 'init' function is called, plt will attempt to add cursors to all lines of the axis created by plt. If you want plt to skip adding cursors to some of the lines, you should tag the line with the string 'SkipCur'. For example, a cursor would

Notes: t

<pre>not be created for a line created with the following command: line(x,y,'tag','SkipCur');</pre>
Another way to restrict which lines are to be cursored is to add the application data key 'Lhandles' to the axis. (For example: setappdata(ax, 'Lhandles', [h1 h3]); would tell the cursor initialization routine to add cursors only to those two handles.)



## **Pseudo objects**

The table below describes the GUI building pseudo objects provided with the plt toolbox. Since this section jumps immediately into the details, it would be best if you first read the overview of these pseudo objects provided <u>here</u>

The plt pseudo object can be thought of as a super axis and in that Matlab's plot and plotyy functions but as you will learn it is much

It may be confusing at first that plt is both the name of a pseudo ol the whole toolbox. In fact all the pseudo objects describe here are queried by using calls to the same plt() function. The plt pseudo remaining pseudo objects in that:

• For the remaining pseudo objects, the first plt argument will a pseudo object (as a string) but this is not true for the plt pseudo pseudo object, one or more numeric arrays are always require plotted and usually we put these arrays at the beginning of the it more obvious that we are creating a plt pseudo object. It is the numeric arrays first and a call to plt will still create a plt p argument is a string, as long as that string is not one of the following sensitive):

click	dateStr	help	metricp	rename	ver
close	edit	helpText	misc	select	
colorPick	grid	hideCur	move	show	
cursor	hcpy	markEdit	рор	slider	

- When a plt pseudo object is created, two other pseudo objects well (namely cursor and grid)
- A new figure window is immediately created to contain the p
- The plt pseudo object may be called from the command line ( command) whereas it only makes sense to use the other pseud
- Nearly every other section of this help file is dedicated to des

**plt** (pseudo plot object) pseudo object.

Cursor<br/>(pseudo<br/>cursor<br/>object)The cursor pseudo object, more than any of the others is what give<br/>advantage for GUIs involving plotting and data exploration. Once<br/>and natural methods provided for cursoring, zooming and panning<br/>lived without them and you will want all your GUI tools to be simCursor<br/>(pseudo<br/>cursor<br/>object)The description of the cursor pseudo object is long to fit comfortal<br/>section of the help file (which you will find here) is dedicated for 1<br/>describes how to query and modify the cursor objects as well as ho<br/>using the 'init' action. (It's not likely worth your time to review

The trace color of the native Matlab axis grid lines can't be indepe lines that over power the display. The grid line pseudo object is de by providing grid lines of whatever color and style you choose. Ac referred to is mostly solved in the latest R2014b version of Matlab capability if you need to remain compatible with older Matlab verproblem and the grid lines will look wonderful on all supported M R2014b).

because normally you will let the plt pseudo object create the curs

Grid lines are positioned at each tick label. Additional (sub-decade for logarithmically scaled axes that span six or fewer decades. (Th changed by adjusting the **logTR** figure application data property

### The grid line functions are:

```
plt('grid',ax,'init',color,erMode,LineStyle,
```

- Initializes grid lines on axis **ax** of color **color** with erase melineStyle.

- color is optional with default [.13.13.13]
- erMode is optional with default 'xor'.
- LineStyle is optional with default '-'
- In7, In8 is an optional parameter/value pair to apply to the
- In9, In10 is an optional parameter/value pair to apply to the *The pltvbar.m demo program uses this call to create a tal list next to the main plot area. Although that demo as we*.

<b>grid</b> (pseudo grid line	three line example below calls this 'init' action, you will 1 if ever do this because the grid pseudo object created au by the plt pseudo object is usually sufficient.		
object)	<pre>plt('grid',ax,'toggle')</pre>	- toggle grids (on/off)	
	<pre>plt('grid',ax,'get')</pre>	- get grid line handle	
	<pre>plt('grid',ax,'off')</pre>	- turn grids off	
	<pre>plt('grid',ax,'on')</pre>	- turn grids on	
	<pre>plt('grid',ax,'update')</pre>	- update grids	
	<pre>plt('grid',ax)</pre>	- same as above	
	plt('grid')	- equivalent to <plt('grid< td=""></plt('grid<>	

All the above calls return the grid line handle. Setting **ax** to zero in equivalent to specifying gca.

To experiment with these functions, try typing this at the command

>> plt('grid',axes,'init',[.7 1 1]); % create a:
>> set(gca,'ylim',[0 6]); % change;
>> plt('grid'); % update §

The slider control is Matlab's most versatile ways to control a num allows us to change a value continuously and repeatedly using sev & hold on an arrow, click & hold in the trough, and dragging the s object (described later in this table) enhances this capability furthe movement options and coupling it with the labels and numeric rea the slider. The only downside to the pseudo slider is that it takes u everywhere in a GUI containing many numeric controls. For this r uicontrol('Style', 'Edit')) is often used to control a nu because the only way to change the value of an edit box is to type difficult to use when many adjustments are needed to arrive at an o want to develop a feel for the effect of small changes in the param was designed primarily to overcome that difficulty, although it also advantages over the uicontrol edit box:

- A label is almost always required for a numeric parameter and string parameters as well. The uicontrol edit box doesn't have another uicontrol for that purpose, complicating your program to reposition. The edit pseudo object includes an optional labobject its label automatically moves along with it.
- The edit pseudo object may be positioned using figure coordi or data coordinates of an axis. Which coordinates to use is us the use of the particular control. (The uicontrol edit box may
- The edit pseudo object takes up even less space in your GUI you can fit more controls into a given area. This often simplif avoiding the need for additional modes to control which para
- The edit pseudo object has the (optional) ability to honor set a values.
- The string substitutions of the edit pseudo object's callback fu the callback routines.

There are two types of edit pseudo objects:

- **Type 1:** Two uicontrol objects are created (text style) inside t (If a label is not specified, only a single uicontrol is created.) This type is created if the 'pos' property has lengtl height]).
- **Type 2:** Two text objects are created inside the current axis. *(If a label is not specified, only a single text object is created.)*

This type is created if the 'pos' property value has

For reference, this first table describes the edit pseudo object prop query. How to use these properties will become clear later when the

Property Name	Property Value	
	v (scalar)	Sets the edit object's nume (default = 1)
	[min max] (length 2 vector)	Sets the edit object's allov (default = [-1e99, 1e99])

value	[v min max] (length 3 vector) Sets both of the above.	
	Except for the length 2 argument case, the callback f been provided earlier in the argument list or in a preplt('edit').	
val	Equivalent to the <b>value</b> property above, except that called.	
minma	<pre>Equivalent to the val property above. (For clarity, u argument has length 2 (i.e. [min max]).</pre>	
callb	<ul> <li>A callback to be executed when the edit text object is If the callback is defined with a string, then:</li> <li>Occurrences of '@VAL' will be replaced with the ct Occurrences of '@OBJ' will be replaced with the eq Also note that if the function is defined as a string ar single quote characters are required (quotes within q readability can be improved by replacing all sequenc single quotes with a double quote character. For exai 'disp(''ABC'');' could be written as 'disp( this trick does not work for Matlab callbacks in gene any callback defined within a plt() function c</li> </ul>	
enable	0=disable, 1=enable (default=1). If disabled, the text may not be modified.	
incr	The increment value for a numeric edit pseudo objec click on the right/left side of the center of the object, edit object is increased/decreased by "incr". A negati to indicate that the increment factor is in percentage For example, if incr = -0.1 then clicking on the right will increase/decrease the edit object's value by 0.1 µ 1001 or 999). Setting incr to zero disables the incr/de the left clicking on the object will have the same effe that the incr/decr feature is also disabled if the length to anything other than one.	
lengtl	The length of the vector allowed as the edit value. U the edit value must be a scalar. If length=4 (for examination value must be a row or column vector of length 4. The length=-1 and length=0. length=-1 is used to indicate	

	any length is a legal value. length=0 is used for strin string is not interpreted as a number or vector. (defau		
format	The format conversion string used to display numeri (default = '%6w'). Type "help prin" for a descriptio		
	<pre>If the argument is a string, that string will become th label. Usually this is sufficient, but if you want more position or appearance the argument may be a cell an {'LabelString', offset, 'Property1', Value1, 'Property2', Valu</pre>		
	Note that the label will be created with the same type (i.e. a uicontrol for type 1 and a text object for type 2 the cell array must be valid properties for that object		
	Note that the label will be created as a text object in to display the popup choices. Property1, Property2, ¢ object property names. The meaning of the offset pa- length as follows:		
		<u>Offset</u>	<u>Meaning for Type 1</u>
		'' or []	plt estimates the best label size position based on the # of chai in the label.
label		q	The label width is set to q. plt estimates the best label hei position.
[q1 q2]The label width is set of The label position is set edit object position + [real(q2) imag(q2)][q1 q2 q3]NA		[q1 q2]	The label width is set to q1. The label position is set to: edit object position + [real(q2) imag(q2)]
		NA	
		[q1 q2 q3 q4]	The label position/size is set to

	[q1 q2 q3 q4]
	For a type 1 edit object, if the estimated width of the character length is too big or too small you may adju parameter as described above. However an alternate itself. You can make the width bigger by padding the can also make the width smaller by using the tilde cl the desired label is "Abcdef", you can make it slightl the label as "Abcde~f" or smaller still by using "Abc (Note the tilde character itself does not appear in the
* * * * *	If a property name is given which isn't in the list abo applied to the main uicontrol or text object itself. (It name for type of object being used.) The position (' object must be set this way. For a type 1 edit pseudo foreground/background colors ('foregr' / 'back usually set this way, however if these properties are background color defaults to 0.8 times the figure bac foreground color defaults to [1 1 .4] or [0 0 .6] (whic contrast). The text color of a type 2 edit pseudo obje well ('color' property) with its default handled ir foreground color for the type 1 object.

The following commands are used to create an edit pseudo object:

**edit** (pseudo edit object)

H = plt('edit', 'Poperty1',Value1, 'Poperty2',Value2,)	The property name interpretation of th shown in the table many or as few pro whatever order you
H = plt('edit', [x y width height],v, 'Poperty3',Value3,)	Usually both the 'p properties are need you may omit thos property values ap Note that V in this 3 vector if you wan values. Property na may not be omitted
H = plt('edit',	Assuming at least

'pos',[x y width height],	values is greater th
'value',v,	units are being use
<pre>'Poperty3',Value3,)</pre>	translated into this
<pre>H = plt('edit','unit','norm',</pre>	If all of the positio
'pos',[x y width height],	(indicating that not
'value',v,	used), a slightly di
<pre>'Poperty3',Value3,)</pre>	(Note the addition

The above calls create an edit pseudo object and returns the handle type 1) or main text object (for type 2) created. (By "main" I mean value, not the label). This handle (H) may be used to modify or qu parameters using the forms below:

plt('edit',H, 'Poperty1',Value1, 'Poperty2',Value2,)	If H is a scalar, the s are applied to the ed by handle H. The pr same ones described If H is a vector then to the kth row of Va have that many rows used. Only one prop case where H is a ve
plt('edit',H,'get','value')	returns the numeric pseudo object. For c may also be written plt('edit',H, the last two argumer
plt('edit',H,'get','minmax')	returns [min max] - value
<pre>plt('edit',H,'get','callbk')</pre>	returns the string or set via the 'callbk' pរ
<pre>plt('edit',H,'get','enable')</pre>	returns 0/1 if the pse disabled/enabled
plt('edit',H,'get','incr')	returns the value tha parameter

<pre>plt('edit',H,'get','length')</pre>	returns the value that parameter
<pre>plt('edit',H,'get','format')</pre>	returns the string that parameter
<pre>plt('edit',H,'get','label')</pre>	returns the label han
plt('edit',H,'get','cell')	returns an 9 element concatenation of the commands: {val min length format label} may be replaced by of the other 8 valid '

## Keyboard and mouse behavior:

Right-clicking on the edit text object always "opens" the object for that the old edit string appears with the cursor (underscore) at the that it is ready to accept keys typed at the keyboard. If you start ty characters typed will be appended to the end of the old string. To i point other than the end, simply move the cursor to the desired poi keys. To remove characters, press <Backspace> or <Delete> to rer after the cursor. Pressing <Delete> when the cursor is at the end of characters. This special case makes it easier to enter a new string t the previous entry. Note that while typing, the text object is shown you that a new value is being entered. When you press <Enter>, th the color returns to the original. If you type an invalid entry, the w again on the "error" string to try the entry again or to recover the p summary of the special keys follow:

<esc></esc>	The edit text object is closed for editing and the restored as if the edit object was never opened for
<backspace></backspace>	Deletes the character on the left side of the curso
<delete></delete>	Deletes the character on the right side of the curs end of the string, all the characters are deleted le cursor.
<right arrow&gt;</right 	Moves the underscore cursor one position to the
<left arrow=""></left>	Moves the underscore cursor one position to the

<enter></enter>	Closes the edit text object, accepting the current underscore cursor) as the new value.
<click></click>	Clicking the mouse on the edit text object while effect as pressing <esc> on the keyboard.</esc>
i	When entering a scalar value, if lower case "i" (i the last character, this indicates that the entered new "incr" parameter for the object. In this case before the object was opened is retained.

Left-clicking on the edit text object also opens the object for editir exception. This exception happens when the edit text object is a sc is equal to one). In fact this is more the rule than the exception sinused (and default) value for the length parameter.

In this (scalar parameter) case, when you left-click on the edit text incremented or decremented by the object's "incr" parameter. Whe or decremented depends on the position of the mouse click. If you object's center, the value will be incremented. Likewise, the value clicks to the left of center. As an example, suppose the current edit 259, and the increment parameter is 1. Left-clicking on the 9 will (to 260 (because the 9 is right of the center of the text string). On the two will decrement the value by one. Remember that if the inc convenient, you can change the increment amount on the fly by us described above.

An important property of the scalar increment/decrement feature is continue to increment as long as you hold down the mouse button. interactive controls and allows the edit objects to take the place of graphic object to have this repeat property). When you hold down be a delay of 0.4 seconds before the auto-incrementing begins. Aft be incremented once every 0.03 seconds. You can alter the repeat object's application data **repeat** property. For example, if you we the speed of the default) use the command:

```
setappdata(H, 'repeat', 0.06);
```

where H is the handle of the pseudo edit object. You can also chan well. For example, the command setappdata(H, 'repeat',

the repeat rate to 0.06 seconds and the repeat delay to 0.25 second repeat feature by setting the repeat delay to a negative number, for setappdata(H, 'repeat', -1);.

The easiest way to reset back to the default repeat and repeat delay property to null ([] or '') or simply remove this property altoget rmappdata(H, 'repeat').

Whenever a number is being typed in, you may also type an expre the following entries are all equivalent:

- 5
- abs(3+4i)
- [2 1] \* [2 1]'
- sqrt(3\*2^3-cos(pi))

Typing sum(get(gca, 'xlim').\*[0 1])'would be equiva upper x axis limit. And to be really perverse, typing log(-1)/(: to typing ".5i" which as mentioned above would change the auto in

See both the **gui2.m** and the **curves.m** <u>example programs</u> for i pseudo edit objects as well as the pseudo popup objects described example also uses an edit text object for controlling the rotation sp the last figure window.

Just as the pseudo edit object described above may replace an edit object described here may be used to replace a popup uicontrol (uicontrol('Style', 'Popup')). The pop pseudo object those listed above for the edit pseudo object.



The popup pseudo object is highly cu typical example. On the left is the pop on the current selection ("normal" in t will open as shown to the right. Note currently selected item is shown in bc simply clicking on the new selection y with the new selection. The following table describes the popup pseudo object properties that although how to use these properties may not become clear until la discussed:

Property Name	Property Value		
choices	A cell array of strings specifying the choices giver selected. If all the choices are numeric you can use of the cell array. For example [3 5 7.5] is equ '7.5'}.		
index	An integer specifying the current choice (default = the choices cell array). After the popup is set to th popup is closed if it had been open. The callback i when the popup choice is set using the index parau specify the negative of the index desired, then the after the index is set.		
callbk	A callback to be executed when the popup value is If the callback is defined with a string, then: Occurrences of '@IDX' will be replaced with the Occurrences of '@STR' will be replaced with the		
	<pre>[x y width height] for the opened popup units.</pre> Two other options may be enabled by making the		
	Therefore these two lines are equivalent:		
	plt('pop','pos',[3 .5 .1 .2], plt('pop','pos',[ .3 .5 .1 .2],'o1		
position	Likewise the following two lines are also equivale		
pos	plt('pop','pos',[.35 .1 .2], plt('pop','pos',[.3 .5 .1 .2],'s\		
	In both these examples the second line is preferred the descriptions of the offset and swap parameters can make both x and y negative to enable both the		

	If the position was specified previously you can us modify just the height component or a two element		
	width and height components.		
offset	<ul> <li>with and height components.</li> <li>y or [x y] which specifies the location of the close opened popup.</li> <li>x is set to 0.08 if it is not specified (where 0/1 replot the left/right edges of the opened popup). The r becomes clear by considering the example where the contained 4 elements. Then:</li> <li>y = 0: represents the vertical position (opened popup)</li> <li>y = 0.5: represents the vertical position (opened popup)</li> <li>y = 1.5: represents the vertical position (opened popup)</li> <li>y = 2.5: represents the vertical position (opened popup)</li> <li>y = 3.5: represents the vertical position (opened popup)</li> <li>y = 4.0: represents the vertical position (opened popup)</li> </ul>		
	Note that negative values for x or y may be used to popup position should be to the left of or above th popup.		
	If the offset parameter is not included, then the clc top of the opened popup, i.e. at position $\begin{bmatrix} 0 & 08 & r \\ 0 & 09 & 08 \end{bmatrix}$ of popup choices). To put the closed popup at the popup simply include 'offset', 0 in the parameters.		
colorbk	The background color used when the popup is ope		
colorfr	The foreground color used when the popup is open		
enable	0=disable, 1=enable (default=1)		
	Normally a left-click on the pseudo popup control the list of choices while a right click merely advar		

swap	<ul> <li>without opening the popup. Sometimes it is useful left and right-click - a mode I refer to as the "supe there is only a single choice, it behaves exactly lik that the only effect from clicking on it is that its ca are two choices in the choices cell array, it behave button text toggles between the two choices with a there are fewer than four choices, the super-button than the regular mode. The super-button mode is s swap parameter in the argument list. Often you wi button look at least somewhat like a button by inclusuper-button text. The color of this box is specifie. For example swap, 'blue' will draw a blue box Or instead of a Matlab color string, you can use ar the parameter to zero to use the current foreground [.1 1.9]). If the parameter is missing (which is onl argument is at the end) or if the parameter is set to then no box is drawn around the button text. Some able to toggle the super-button mode on or off inte with the following quick procedure:</li> <li>1. First open the popup. (A left or a right click v which mode you are in.)</li> <li>2. Then use the mouse to move the figure winde (less than 15 pixels in any direction).</li> <li>3. And finally close the popup by clicking on ar drop down list.</li> <li>The 15 pixel requirement makes it quite unlikely t performed unintentionally, especially since one rai figure while adjusting a control. The text swap the Matlab command window to assure you that thintended.</li> </ul>
hide	A list of objects to hide before opening the popup. closed, these objects are shown again (unless they popup opened). The hide parameter is used to rem objects that overlap or are too close to the popup l list, before the list is used the zero will be replaced handles) of the plot grid lines. (This is convenient most common object to overlap with opened popu

<b>pop</b> (pseudo popup		parameter to '' or [] or not supplying a paramet the 'hide' argument is the last one) is equivalent to zero.
object)	interp	<b>'none'</b> or <b>'tex'</b> (default='none')
	label	<pre>{'LabelString', offset, 'Property1', Value1, 'Property2', Va.' A label for the popup will be created as a text obje used to display the popup choices and will contair (the first element of the cell array argument). The array is a complex number containing the offset fc pixels from the closed popup. (The real part is the imaginary part is the y offset). For example if the array is -10+20i, then the label will be positioned 20 pixels above the closed popup. The remaining cell array elements (if any) must co pairs, and the properties must be valid text object ' specified properties will be applied to the closed p Specifying an offset of '' or [] is equivalent to s for this cell array element. (i.e. the default offset is Instead of specifying a cell array for the label argu specify a string. For example, the argument 'Lab equivalent to {'LabelString', ''}. Since th sufficient, it turns out that this shorter form is usec cell array parameter.</pre>
	labely	This behaves just like the <b>label</b> parameter above the default offset when it isn't specified is 16i (inst <b>label</b> parameter). This is the usual offset needed label directly above the closed popup text.
	* * * * *	If a property name is given which isn't in the list a applied to the popup text object itself. (It must be The text color ('color') is usually set this way, is not specified, the default color is [1 1 .4].

H = plt('pop', 'Poperty1',Value1, 'Poperty2',Value2,)	The property na interpretation of shown in the tal as many or as fe in whatever ord
<pre>H = plt('pop', [x y width height], {'choice1', 'choiceN'}, 'Poperty3',Value3,)</pre>	Both the 'position properties are re- conciseness you names if the pro- and in this order than these two u
<pre>H = plt('pop',     'pos',[x y width height],     'choices',{'choice1',},     'Poperty3',Value3,)</pre>	The above com this line before any of these con either pixel or m aren't required t property since p property from the the position vec

The following commands are used to create a popup pseudo objec

The above calls create a popup pseudo object and returns the hand visible when the popup is closed. This handle (H) may be used to a pseudo object parameters using the forms below:

plt('pop',H, 'Poperty1',Value1, 'Poperty2',Value2,)	<ul> <li>If H is a scalar, the s are applied to the pc identified by handle are the same ones de above.</li> <li>If H is a vector then to the kth row of Va have that many rows used. Only one prop case where H is a vector H is</li></ul>
---	---

plt('pop',H,'get','string')	returns the currently choices cell array. Equivalent to the sh get(H, 'string
plt('pop',H,'get','axis')	returns the handle o: the opened popup.
plt('pop',H,'get','choices')	returns the cell array via the 'choices' para
plt('pop',H,'get','index')	returns the index nu pseudo object (betw the choices cell arra command may also argument: plt('p without the last two plt('pop', H)
plt('pop',H,'get','callbk')	returns the function set via the 'callbk' pដ
<pre>plt('pop',H,'get','offset')</pre>	returns the number ( specified via the 'off
<pre>plt('pop',H,'get','colorfr')</pre>	returns the 3 elemen that was specified v
<pre>plt('pop',H,'get','enable')</pre>	returns 0/1 if the ps& disabled/enabled
plt('pop',H,'get','hide')	returns the vector of the 'hide' parameter
plt('pop',H,'get','interp')	returns the string that parameter
plt('pop',H,'get','label')	returns the label han
plt('pop',H,'get','swap')	returns true if the sw selected.
plt('pop',H,'get','cell')	returns an 11 elemei concatenation of the commands: {axis choices index enable hide interp la

	not include the first listed above. Also 'c replaced by any strii other 12 valid 'get' a
<pre>plt('pop',H,'open')</pre>	opens the popup rev

#### Mouse behavior:

Left-clicking on the popup text object "opens" the popup. What th string is replaced by a list of the popup choices (rendered using co clicks on the desired choice which then becomes the new text strin popup text object changes the text string to the next available choi already selected, then right-clicking will change the text string to t the "choices" array). If you hold down the right mouse button, the advance cyclically. You can alter the rate at which the cycling proc application data **repeat** property. The use of this property is mor above in the description of the edit pseudo object.

See both the gui2.m and the curves.m example programs for i pseudo popup objects as well as the pseudo edit objects described and the wfall.m examples show how to use this pseudo object to to start and stop a moving plot.

When using a uicontrol slider to control a parameter in a GUI, bes one generally also wants an edit box to show the current slider val entry by typing a number. Also a label is usually required to identi labels indicating the minimum and maximum allowed values are a slider object combines those five objects into one and is included i easier to design. The additional movement and quantization modes reduce the amount of code you need to make the control work as d pseudo object, use the command:

## H = plt('slider', In1, In2, In3, In4, In5, In6, In7

This creates a pseudo object which usually looks something like th

Stopband ripple		
10	40	120

The five component uicontrols that are created for identified as:

-----label-----

## min ValueEditbox max -----slider-----

The variables used in the above slider initialization command are (

Η	The return value is the pseudo slider's handle which is the pseudo slider's properties. (See the get and set c
	<pre>[x, y, width] in pixels or normalized units. Values] to be normalized. Mixing units is ok although x &amp; y m [x, y] are the coordinates of the lower left corner of th (which is also the upper left coordinate of the min valu missing 120 is assumed.</pre>
	When the position is specified with a 2 or 3 element as pseudo slider will look similar to the object shown abo subcomponents. The second way to specify the pseudc element vector in the traditional Matlab format (i.e. [xLeft,yBottom,Width,Height]). When usin vector only specifies the position and size of the actual calculate what is hoped to be the optimal position and box components. If the Width value specified is large then the pseudo slider will end up looking similar to th
In1	<b>*</b> points <b>I I I I I I I I I I</b>

	Finally there In1 argumer which specify components r pixel units an vector or an e invisible.	is one last met at is specified b the positions respectively. The d each componempty vector w	hod when complete flexibi by a cell array. This cell arr of the Label, Slider, MinTe hese positions must all be i nent must be either a 4 elen which indicates that the assc
	In1 is the on	ly required pa	rameter for this function.
In2	<b>[value, s</b> value is the ir smin/smax ar emin/emax ar the edit box.	min, smax, nitial value ass re the slider val re the smallest/ If emin and em upplied, [50	<ul> <li>emin, emax]</li> <li>igned to the slider.</li> <li>lues at its leftmost/rightmost/largest values allowed whenax are missing, 1e-99 and</li> <li>100] is assumed.</li> </ul>
In3	Slider label. I	f you don't wa	nt a label, don't supply this
In4	Slider callbac slider control be replaced w must include If you need to callback, sim	ck. This expres or enters a nu- vith the current it if you want o supply this pa ply set <b>In4</b> to	sion will be evaluated whe mber in the edit box. Occur value. This parameter is o to specify any of the three j arameter for that reason, bu
	The following as <b>In5</b> , <b>In6</b> , may be inclue used to identi	g three parame , and <b>In7</b> in fa ded in the argu fy which parai	eters are optional. Although act these three parameters ( ment list in any order you ( meter is being supplied.)
	This paramet	er controls hov	v the slider moves when the
	are clicked of	when clicking	g in the space to the left or 1
	1	Linear	none
TnF	2	Linear	rounded to nearest integer
	3	Linear	rounded to nearest power
1113	[4 q]	Linear	rounded to nearest multipl
	5	Logarithmic	none
	In2 In4 In5	Finally thereIn1 argumerwhich specifycomponents inpixel units andvector or an einvisible.In1 is the orIn2In1 is the orIn2In2In3Slider label. In''In4Slider callbackIn5	Finally there is one last met In1 argument is specified by which specify the positions components respectively. The pixel units and each components respectively. The summissible.In2In1 is the only required parameter control or enters a number of the supply this parameter controls how are callback, simply set In4 to The following three parameter as In5, In6, and In7 in far may be included in the argunated to identify which parameter I Linear 2 Linear 3 Linear [4 q] Linear 5 Logarithmic

## slider

(pseudo slider object)		[6 q] Linear rounded to nearest integer and to nearest multiple of If In5 is not provided then 1 is assumed. For modes 4 & 6, q defaults to 10 if not specified.
	In6	<pre>{fmin fval fmax} fmin/fmax are formatting strings for the min/max label formatting string for the edit box. These strings may cc formatting codes or the W,V,w,v formats. (Type "help ] these formats). You may use a space to delimit the formatting codes. F '%4w %5.2f %2w'. Or use a row or column cell array if you prefer: { '%4\ Since the w format is the most convenient format to us conciseness a single digit may be used as a shorthand 1 example above may also be written as '4 %5.2f 2' {'4' '%5.2f' '2'}. Often it is sufficient to only specify the format for fval for fmin and fmax (which are '%2w' and '%3w' resp by simply specifying a single format code. For exampl to '2 %5.2f 3'. If In6 is not provided, then '6' (or equivalently '2 €</pre>
	In7	<b>[LabelBG; EditBG; LabelFR; EditFR]</b> This is an array containing 3 columns and up to 4 rows respectively represent the proportion (0 to 1.0) of red/g control. The first two rows are the background colors f respectively. The last two rows are optional and contai foreground colors. If the foreground colors are not speassumed (i.e. [0 0 0]).

For the commands below, **H** is the handle returned from the above

command.

## The get commands:

<pre>plt('slider',H,'get','value')</pre>	returns the pse
plt('slider',H,'get')	equivalent to t
plt('slider',H)	also equivalen shortest and m getting the slic
<pre>plt('slider',H,'get','visible')</pre>	returns 1 if the otherwise
plt('slider',H,'get','ena')	returns 1 if the otherwise
<pre>plt('slider',H,'get','position')</pre>	returns the slic [x y width] in originally spec
<pre>plt('slider',H,'get','pos')</pre>	same as above
plt('slider',H,'get','obj')	returns the slic [ Label; Slid EditBox ]

## The set commands:

plt('slider',H,'set',a)	Sets the slider to val Returns a possibly l
plt('slider',H,'value',a)	Equivalent to above extra parameter ' <b>set</b> <b>plt('slider',</b> if you prefer. In fact allowed (immediate in all the remaining table.
plt('slider',H,'val',a)	Same as above exce <b>not</b> executed
	Sets the slider positi

<pre>plt('slider',H,'position',a)</pre>	(See <b>In1</b> in the slid description above)
plt('slider',H,'pos',a)	Same as above.
plt('slider',H,'minmax',a,b)	a contains [smin,sm above) b is a new slider val Returns a possibly l
plt('slider',H,'visON')	Makes slider visible
plt('slider',H,'visOFF)	Makes the slider inv
<pre>plt('slider',H,'enaON')</pre>	Enables the slider.
plt('slider',H,'enaOFF')	Disables the slider
plt('slider',H,'label',a)	Sets the label string label is made invisit
plt('slider',H,'mode',a)	Sets the slider move above). Returns a possibly l

Before getting much farther in this section, you should try running program so that you have a better context for the information that

The Image pseudo object provides cursoring methods appropriate includes several optional components including:

- A color bar which serves as a legend for the z-axis values as v changing the colormap used to represent the z data.
- A slider (labeled 'edge') that allows you to control how wide used when determining the color used to represent each array
- A slider (labeled "**mid**") that allows you to control the center used to determine the color for each array element.
- A checkbox that allows you to control the visibility of the axi
- A 'view all' button, that when clicked on resets the axis limite object so that the entire image data set is visible. A secondary activated by right-clicking on it instead. Each time you right-limits are zoomed in to expose only the middle 36% of the vi 60% expansion of both the x and y axes.).

#### Initialization

#### h = plt('image',axI,x,y,z,opt);

The image object is created in an axis and figure that must be crea pseudo object. For example, suppose we create a figure with the coplt(x1,y1,x2,y2, 'Subplot', [40 60]); These creates (40% of the height) below and the larger one on top. And suppose map in the larger axis, then we would use plt('image', 2, x, to 2 to indicate that the image should be inserted into the upper plc the ordering of the axes). You wouldn't expect it matters what is in covered over by the image object, but in fact x2 should be set to th specified in the image initialization. This insures that the intensity correctly. (The data in y2 on the other hand does not matter as long x2).

opt is an optional cell array that specifies the image object option contain any or all of the following strings in the following table. T insensitive, and actually all characters except the first one are igno 'view', 'ViewAll', and 'V' would all serve the same purpos

If this string is included the color bar image is created key (i.e. for displaying the current color map). You ca to cycle through seven different color maps as follow

- 0: rainbow
- 1: jet
- 2: sometric
- 3: seismic
- 4: gray

'cbar'

- 5: colorcube
- 6: lines

This entry is an exception to the rule that only the fire because you may also include a digit between 0 and 6 the string. This specifies which color map to appear v initialized. For example 'cbar3', 'c3' or 'Cbar initialize the color map to "seismic". If the last digit c then the the rainbow color map is selected as the by c

<b>image</b> (pseudo	'edge'	If this string is included the "edge" slider will appear of the zData that is mapped to the selected color map
intensity map object)	'mid'	If this string is included, the "mid" slider will appear point of the zData range that is mapped to the selecte adjusting this slider, as well as the "edge" slider ment in more detail in the description of pltmap.m in the section.
	'grid'	If this string is included, a "grid" checkbox will appe- turn the grid lines for the image object on or off. This rule that the case is insensitive since 'Grid' and 'grid' ] meanings. They both create the checkbox, but the cap the check box to 'on' (i.e. the grid lines are enabled) a initializes the box to 'off'.
	'view'	If this string is included a "view all" button will appe clicked on will zoom the axis so that the entire image mentioned above, you may also zoom into the middle right-clicking on this button. (The middle 60% of bot become visible.)

You may optionally include a 4 element position vector (in normal these 5 options strings. If the position vector is not included, then for the item. For example:

# opt = {'Cbar' [.5 .4 .02 .24] 'Grid' 'ViewAl .02]};

When this option cell array is used for the image object initializati checkbox, and a view all button will be created, but the edge and r they are not included in the options list. The color bar and view all the coordinates given, whereas the grid checkbox will be positione there is no position specified for it in the options list. You may use delimiter between option elements (i.e. row vector form as in the *z* semicolons if you prefer (i.e. column vector form).

The call that initializes the pseudo image object returns the image then use in any of the image modification commands shown below update commands which come in seven forms depending on whicl update:

```
plt('image',h,'x',x);
plt('image',h,'y',y);
plt('image',h,'z',z);
plt('image',h,'xy',x,y);
plt('image',h,'xz',x,z);
plt('image',h,'yz',y,z);
plt('image',h,'xyz',x,y,z);
```

Then we have commands which can change the values of the mid allowed if they were created by the image object initialization):

```
plt('image',h,'mid',Value);
plt('image',h,'edge',Value);
```

And finally we have commands to change the x and y limits:

```
plt('image',h,'xlim',[xLower xUpper]);
plt('image',h,'ylim',[yLower yUpper]);
```

For convenience you can combine any of the above image modific command. For example to change the y and z data values, adjust tl limits you would use a command such as:

plt('image',h,'yz',y,z,'edge',1.5,'ylim'



Although you will proba

scheme for your GUI applications, the user's satisfaction with the

improved by allowing them to choose the colors used for the majo ColorPick pseudo object you allow the user to efficiently choose the consistent way within and among your applications. Also you will using the ColorPick figure is much easier than using the typical W attempts to show all possible colors in a single palette. I've found area of your potential color choice before you can decide if the col possible to accomplish that using a single palette which is what ma allowing you to fix one of the colors, ColorPick presents an 11x11 remaining two colors which gives you plenty of area for each colo enough, you can resize the ColorPick figure to be as large as you v that the instant you click on one of the 121 colors in the current pa the palette **AND** the objects in the gui that you are adjusting instar color. This instant feedback is really necessary to remove the frust the task. Scrolling through the many possible palettes is also very and click on the left/right arrows for a finely changing palette, or c slider for a more coarsely changing palette (which should be fine  $\epsilon$ color chooser).

Any Matlab object that has either a callback property or a buttond used to bring up the ColorPick window shown here. Before I desc programming perspective I will give a few more details about how this window.

The text above the sliders is generally used to identify the object o the color being selected. The three sliders always indicate the RGI selected color. In the example shown at the left, the RGB values a respectively. In Matlab, this color would be represented by the vec rectangular patch object in the upper right corner always indicates the proportions are set to agree with the values of the three RGB s red one in this example) will always be shown with its text value i the "active slider". A slider will become the active slider whenever into its edit box or when you click on the left or right slider arrows the currently selected color:

- You can simply select the desired color using the RGB slider: it is irrelevant which slider is identified as active.
- You can click on any one of the 121 small square patches that grid. The colors in this grid are entirely determined by the act since the active slider shows RED=35%, every color in the 11

= .35 with varying amounts of the other two colors. So in this the 4 corners of this grid starting at the lower left and moving [.35 0 0], [.35 1 0], [.35 1 1], and [.35 0 1]. and the square exagrid has the color [.35 .5 .5]. Clicking on any one of these 12 inactive sliders to the values associated with the patch that yc slider will be unchanged and will remain active). Also the lar the gui that are associated with this ColorPick object will cha color.

• Clicking on the large patch will cause the sliders to move to t color that was in effect when the ColorPick window was first never changes for as long as the ColorPick window remains c changed to the values shown currently by the sliders by closin re-opening it again.

Next I will describe how the ColorPick object is created from the J will call the object that initiates the creation of the ColorPick figur You must do the following two things to make a ColorPick figure

**ColorPick** (pseudo color select object) You must assign an application data variable named 'm' to tl array of this form:
 {'PROP1', H1, 'PROP2', H2, 'PROP3', H3, .

'label'}

When the user selects a new color using the ColorPick figure object with handle H1 will be set to the 3 element vector [R C values from the ColorPick figure. If H1 is a row vector of har objects will be treated similarly. Then in sequence the objects in a like manner. If any of the 'PROPi' entries are 'str' or 'strin element color vector is converted to an Ascii string before be In place of any property string, you may use a cell array of st will assign the user selected color to all the properties listed i handles listed in the following argument. Finally, the last entr included will appear above the rgb sliders and is used to iden being controlled.

- To the callback or buttondownfunction property (or both) of t assign one of the following strings:
  - 1. 'plt ColorPick' This will cause the ColorPick fig callback or buttondownfunction is called except for one the callback of an 'edit' style uicontrol is called. The reas that if you type the desired colors directly into an edit bc

you didn't need the help of the ColorPick figure. Note th properties listed in the 'm' application data cell array are above even though the ColorPick figure is not created. N window will appear when you right-click on such an edi buttondownfunction of the edit box has been similarly as

- 2. 'plt ColorPick ccf' This has the same effect as user changes a color with the ColorPick figure, in addition the 'm' cell array, the function ccf is called. ccf stands and may be any string corresponding to a function name arguments, such as in 'plt ColorPick changeFunc(3,-1)'. string arguments as well although this is less convenient sets of quotes around each string argument. For example 'plt ColorPick changeFunc("StringArg")'. You should esp functions that required a string argument containing space wanted to do that it would be possible with an obscure le 'plt("ColorPick", "changeFunc(""A string argument")")'
- 3. 'plt ColorPick 0 0' is similar to case 1 above a
  0' is similar to case 2 except that the special case relatine does not apply (i.e. the ColorPick window will be create

Only one detail remains to describe the operation of the ColorPick with how ColorPick determines the starting positions of the RGB first opened. This is a two step process. First ColorPick must decic to determine the initial color. Once the object is chosen, ColorPick of this object to use. Here are the details of these two steps:

- 1. **Picking the object which will determine the initial color.** U object will be among one of the handles included in the 'm' are in this case the main object itself is used to determine the init. However sometimes this is not the case. For example, in the a would like to open the ColorPick window when we click on t which serves as a label for the text color patch. However since entirely appropriate for the color of this text object to change appear in the list of handles in the 'm' cell array. So in this ins the initial color from the object whose handle is the first elem array. (This is the only instance where the order of the handle
- 2. Picking the property of the selected object which determin
  - *If the selected object is a uicontrol*. If the uicontrol is an property is always used to determine the initial color. If

3 numbers, then extra zeros are added to the end of the v any numbers that are greater than one, then these numbe any numbers are less than zero, then these numbers are c uicontrol is not an edit box, then first the 'string' property represents a valid color vector (i.e. it must have 3 eleme 0 and 1). If it is valid, than this vector is used as the initi uicontrol 'backgroundcolor' property is used as the initia

- *If the selected object is a text object*, its 'string' property to a valid color vector. Otherwise the 'color' property of
- *If the selected object is a patch object*, its 'facecolor' pro color.
- *For all other object types*, the 'color' property is used as

This pseudo object can be thought of as a super text object ... i.e. a that can be created and deleted with a single command. The pseud on of the following two commands:

```
plt('HelpText','set',v);
plt('HelpText','on',v);
```

where  $\vee$  is a string or more commonly a cell array of strings and si HelpText object is associated with the current figure window. Unli figure may contain only one HelpText object, which is why an obj you create a new HelpText object for a figure that already has one, before the new one is created.

The second form above ('on') is equivalent to the following two cc plt('HelpText', 'set', v); plt('HelpText', 'on');

where the 'on' command is described below. Thus the two initialization that when the first form is used the text will not become visible un issued, whereas with the second form, the text becomes visible im

As mentioned in the <u>Labels and figure properties</u> section you may pseudo object at the same time the plt pseudo object is created by in the parameter list of the call to plt. (In fact, that is by far the mo object is created.) Creating the HelpText that way is similar to the the help text becomes visible right away (i.e. a separate HelpText '

The structure of the  $\vee$  parameter is described in the table at the bol list the commands used to control the HelpText object. The functic command form is shown below since that is more convenient whe needed:

plt HelpText on	Although the pseudo object is <b>'set'</b> command, the individuate not actually created until given. Most often this commutelp Text after it was turned shown below.
plt HelpText plt HelpText off	Both of these forms are equivitext objects created by the abyou will want the help text to starts using the plot so it does way to do this is to include 'MoveCB', 'plt HelpT(list when you are creating the sets the move cursor callback the cursor or click on the plot removed.
plt('HelpText','get')	This returns the <b>v</b> parameter HelpText object. There aren't command, although one use HelpText object from one fig
plt('HelpText','text')	This returns a list of handles created after a helptext 'on' executed. An empty list is ret object is currently off. Since special user value, an equival the command findobj(gcf, 'user', :

HelpText (pseudo text object)

The easiest way to describe the structure of the  $\vee$  argument used a examples:

'abcdef'	Using a single string for the <b>v</b> paresulting HelpText object much underlying text object it creates useful because it is so easy to cr pseudo object.
{'abcdef'1+.5i}	At a minimum you will most lik location for the text object. The complex number with the real p position and the imaginary part position. In this example the hor left of the main axis (by 10% of position is at the middle of the n not specified (as in the example is .03+.96i
{'abc' 'def' 'line3' .5+.96i 'color' 'white' 'fontsize' 20}	Each text string may be as many in this example). After the posit specify as many property value text properties are allowed. If th specified, the default color [1 .6 will default to 12 if not specified (as well as the following one be (complex) is required. This help row cell array. (A column cell an expected result.)
	With the full generality, you may as you want, with each of these locations with different string pr creates three strings, the first of "2i" (which occurs twice) in this that indicates a new string is abo any complex number will serve habit is to always use "2i" for th easy to recognize.
{'ab' 'cd' .5+.9i 'color' [1 0 0] 2i	In this example, the first two helred because the text is followed 0]. However the next two lines (

```
'line3' .1+.1i 2i
'line4' .9+.9i
'fontweight' 'bold'}
color specifier, so they will be it
0]). However if you change the
before 'line3' to a -2i then
This is because a delimiter with
component indicates that you w
specified for the previous helpte
current one as well. If you chang
as well, then all the help lines de
same property is included both a
the carry over property list (enal
imaginary component) then the
text string takes priority.
```

Many of the demo programs (including curves, editz, gauss, gui2, trigplt, and wfalltst) create a HelpText pseudo object, so you can r practical examples of the use of the HelpText pseudo object.


# Auxiliary plt functions and .m files

The first two functions in this list are part of plt.m and the remaining functions exist as as separate .m files in the main plt folder. The last four functions (Pvbar, Pebar, Pquiv, and pltwater) aid in creating special plot types. The other functions help solve text formatting issues that often arise when writing graphical interfaces.

**pltt** (add plt trace)

**pltt** is so central to the way plt is used that this auxiliary funct refer to the <u>Adding traces</u> section.

s = plt('datestr', datenum, fmt) is similar to s =
that it displays the time with 1 millisecond resolution instead of
datestr function. Let's compare the results from plt's and Matlab'

**datestr** (serial date number to ascii)

```
a = now;
datestr(a,13)
                      03:51:46
plt('datestr',a,13)
                      03:51:46.153
datestr(a,14)
                      03:51:46 AM
plt('datestr',a,14)
                      03:51:46.153 AM
datestr(a,0)
                      31-Mar-2015 03:51:4
plt('datestr',a,0)
                      31-Mar-2015 03:51:4
datestr(a)
                      31-Mar-2015 03:51:4
plt('datestr',a)
                      31-Mar-15 03:51:46.
```

For a description of the allowable formats type help datest arbitrary formatting strings or integers representing 32 standardi example in the table above since the returned date format when between plt's and Matlab's datestr function. (2 vs 4 character yea

#### [Ret1, Ret2] = plt('metricp',x);

**metricp** (metric prefixes) Used to convert a number to a form using standard metric prefix is most appropriate for displaying the value x, and Ret2 is the x scaling plots to avoid awkward scientific notation in the axis lab x = .3456E-6; sprintf('%.3e Volts',x) 3.4 [pfix mult] = plt('metricp',x); sprintf('%.1f %sVolts',x\*mult,pfix) 345

s = prin(fmtstr,OptionalArguments); s = prin(FID,fmtstr,OptionalArguments);

**prin.m** (sprintf & fprintf alternative) Converts the OptionalArguments to a string s using the format s the same thing as sprintf or fprintf (with the same calling sequer additional features including four extra formatting codes. prin ca to implement the new formatting codes. FID is usually a value r or a 2 to direct the result to the Matlab command window. For a see <u>prin.pdf</u> (in the main plt folder). You can view that help file (i.e. with no arguments) at the Matlab command prompt.

s = Pftoa(fmtstr, x) returns in string s an ascii represen according to the formatting string fmtstr.

If fmtstr is of the form '%nW' then swill be the string repres resolution possible while using *at most* n characters.

If **fmtstr** is of the form **'%nV'** then **s**will be the string repres resolution possible while using *exactly* n characters.

If **fmtstr** is of the form **'%nw'** then **s** will be the string representation possible while using *at most* n characters - not counting



If **fmtstr** is of the form **'%nv'** then **s** will be the string repreresolution possible while using *exactly* n characters - not countin *(float to ascii)* The lower case formats (v,w) are typically used to generate strin width. The reason the decimal point is not counted is that with these gui objects, the extra space taken up by the decimal point i

With all four format types, if the field width is too small to allow is returned.

fmtstr may also use any of the numeric formats allowed with

Pftoa('%7.2f', x) is equivalent to sprintf('%7.2f'

Typing **Pftoa(0)** will create a test file which you may find he point formats.

This function is used to plot a series of vertical bars. It doesn't dearray which is then plotted using plt (or even plot). For example bars at x-axis locations 2,3,7,8. Each bar is to start at y=0 and ex following line would meet this objective:

```
plt(Pvbar([2 3 7 8],0,[6 6 5 1]);
```

Normally all three Pvbar arguments are vectors the same length, position of each bar is the same a constant may be used for the 2

Although you don't have to know this to use it, Pvbar returns a c correctly by plt or plot to display the desired sequence of vertica arrays by plotting the real part of the array along the x-axis and 1 the y-axis. The trick that Pvbar uses to display a series of lines v that NaN values (not a number) are not plotted and can be used 1 and Pquiv functions described below use this same trick.)

The general form of the Pvbar function call is:

v = Pvbar(x, y1, y2)

If the inputs are row or column vectors, this would return a complotted with plt or plot would produce a series of vertical bars (c

**Pvbar.m** (vertical bar plots) given by x. y1 and y2 specify the lower and upper limits of the v you list the upper or lower limit first. If y1 is a scalar, Pvbar exp size as y2.

Suppose you wanted to plot 30 red bars (specified by length 30 green bars (specified by length 30 column vectors xg,yg1,yg2). Pvbar as in:

## plt(Pvbar(xr,yr1,yr2),Pvbar(xg,yg1,yg2));

That's probably the first way you would think of, but if xr and  $x_i$  this case) you can accomplish the same thing with a single call t

# plt(Pvbar([xr xg],[yr1 yg1],[yr2 yg2]));

The second form is especially convenient when plotting many b different color). Interestingly, if you use plot instead of plt first f second form.

Note that Pvbar will expand the second argument in either dime example above, if ya1 and yb1 were the same you could just use suppose the base (lower limit) of the first series was always 0 an always -1. Then you could use [0 -1] as the second argument. If was the same value, then the second argument may be a scalar.

To see Pvbar in action, look at the <u>example program</u> pltvbar also shows the use of the Pebar function described below.

This function is used to plot a series of vertical bars similar to th addition of a small horizontal "T" section on the top and bottom depict an error bound of a function, or a range of values that ma Another difference with Pvbar is the way the lower and upper y With Pebar, the first two arguments (x,y) specify a reference pos normally (but not strictly necessary) somewhere in the middle o (l,u) specify the distance between the reference position and the vertical bar. **Pebar.m** (error bar plots) The general form of the Pebar function call is:

```
e = Pebar(x, y, l, u, dx)
```

The position of the bottom of the error bars is **y**-**1** and the top is scalar that specifies the width of the horizontal Ts as a percentag two arguments are optional. If **dx** is not specified it defaults to 3 to **1** (the 3rd argument) in which case the reference coordinates **e**, **x**, **y**, **1**, **u** are generally vectors or matrices of the same size are are allowed to be scalar. Read the description of Pvbar above matrix inputs are interpreted.

To see Pebar in action, look at the <u>example program</u> pltvbar. also shows the use of the Pvbar function described above.

This function is used to plot a vector fields represented by a set of locations. It doesn't do any plotting itself, but returns an array will plot). For example, suppose you wanted to plot 3 arrows (all in t (2,3) and (1,7). Also suppose you wanted each vector to be of le the right respectively. This could be done as follows:

plt(Pquiv([4;2;1],[9;3;7],[0;0;1],[1;-1;0])

This can also be done using Pquiv's complex input form as follo

```
tail = [4+9i;2+3i;1+7i]; head = [1i;-1i;1];
plt(Pquiv(tail,head));
```

Note that row vectors could have been used instead of column v addition to those 3 vectors, you wanted to plot 3 more vectors (i locations but pointing in the opposite direction. Using the previc This could be done as follows:

```
plt(Pquiv(tail,head),Pquiv(tail,-head));
```

Or you could do the same thing with a single call to Pquiv:

plt(Pquiv(tail\*[1 1],head\*[1 -1]));

## Pquiv.m

(vector plots) Of course the equivalent 4 argument (real input) form of Pquiv c

There are 8 possible calling sequences for Pquiv depending on v complex and on whether the optional arrow head size argument figure out which calling sequence you are using.

Calling sequence	Tail coordinates	Arrow width/le
q = Pquiv(A,B)	[real(A), imag(A)]	[real(B), imag(l
q = Pquiv(A,B,h)	[real(A), imag(A)]	[real(B), imag(l
q = Pquiv(x,y,B)	[x,y]	[real(B), imag(l
q = Pquiv(x,y,B,h)	[x, y]	[real(B), imag(l
q = Pquiv(A,u,v)	[real(A), imag(A)]	[u, v]
q = Pquiv(A,u,v,h)	[real(A), imag(A)]	[u, v]
q = Pquiv(x,y,u,v)	[x, y]	[u, v]
q = Pquiv(x,y,u,v,h)	[x, y]	[u, v]

where:

q, A, B are complex vectors or matrices

x, y, u, v are real vectors or matrices

h is a scalar (Arrow head size - relative to vector length)

Read in the Pvbar description above how complex values and N display. To see Pquiv in action, look at the <u>example program</u> pl

Normally the position of a figure window is specified in pixels a [xleft ybottom width height] relative to the monito leftmost position of the monitor. However it usually is more con the useable screen space, which takes into account the space nee needed for the window borders and title bar.

Consider the following two methods of creating a new figure wi

figure('BackgroundColor',[0 0 .1],'Positior
figure('BackgroundColor',[0 0 .1],'Positior

In the first method, the pixel coordinates in p are relative to the 1 multi-window GUI makes it impossible to make good use of the the taskbar is and other desktop variables. In the second line how to a pre-defined clear area of the screen which are converted into (this routine).

To accomplish this, figpos must know the screen area that can a window. It gets this information from the screencfg.m routine w optimal border area automatically, however it may resort to usin a very old version of Matlab. It will warn you if this happens, ar comments in screencfg to see if you want to adjust any of the co figpos it will only optimize the border area the first time it is cal from the saved value (which is stored both in the 'border' are object as well as in the file screencfg.txt. This means that if you type "screencfg" at the command prompt so that the border area recalculate the border area every time it starts up, you could add startup file, or the line delete(which('screencfg.txt' effect.

First I will first explain how figpos computes the figure position may find it easier to understand by skipping ahead to the examp

In rare situations, you may want to specify the screen position u [left bottom width height] referenced to the screen Of course, then you don't need to call figpos in the first place ... automatically by plt, so we need a way to bypass the usual figpc simply to place an "i" after any element in the 1x4 vector. For example, the sector of the screen of

figpos([40 50i 600 500]) returns the vector [40 50

It doesn't matter which element contains the "i", and in fact you like, i.e. figpos([400 50 600 500]\*1i);

Suppose you call figpos([p1 p2 p3 p4]) where all the t positive. This is called "size priority mode" because the getting to over getting the left/bottom position in the specified place. In the [left bottom width height] where:

```
width = the smaller of p3 and the maximum clear w:
height = the smaller of p4 and the maximum clear he
left = p1 + left border width. However if this per
right edge of the figure overflow the cleat
the left edge is moved rightward just far
bottom = p2 + bottom border width. However if this
top edge of the figure overflow the cleat
the bottom edge is moved down just far end
```

Suppose you call figpos([p1 p2 -p3 p4]), i.e. the same that the 3rd element is negative. The height and bottom values a (size priority), but the width and left values are now computed a

```
left = p1 + left border width.
width = p3. However if this width would make the i
overflow the clear space available, then i
just enough so that the figure fits.
```

Suppose you call figpos([p1 p2 p3 -p4]), i.e. the 4th e values are computed exactly as shown in the first all positive (si height values are now computed as follows (position priority):

**figpos.m** (figure positioning)

bottom = p2 + bottom border width. height = p4. However if this height would make the overflow the clear space available, then 1 by just enough so that the figure fits.

If you call figpos([p1 p2 -p3 -p4]), then both horizon position priority method described above.

An optional 5th value in the input vector is allowed to allocate e want to do this if you know that a menu bar or toolbar will be er larger. Since this is not accounted for in the border area set up by could cause the top edge of the figure to fall off the top edge of t

**figpos([p1 p2 p3 p4 48])** would allocate 48 extra pixele enough for the menu bar (about 21 pixels high) and one toolt

The default left/bottom coordinates are [5 5] which are used i

figpos([730 550]) gives the same results as figpos([ figpos([730 550 21]) gives the same results as figpos You also may specify only the figure length or the figure width a parameter based on the most appropriate aspect ration. For exam figpos([0 944]) both give the same results as figpos([ ratio (1.006) was chosen so that if you plot a circle, the resulting than elliptical. For example, this line plots a perfect circle using

#### plt(exp((1:600)\*pi\*2i/599), 'pos', [800 0])

If you move your taskbar to a new location, for figpos to continu screencfg.m file by commenting out the appropriate lines definin those changes type **screencfg** at the Matlab command promp enable the changes.

The following examples may clarify the specification described

The first example creates 5 plots of the same size placed on the s away from each other as possible. The first four plots are placed four corners, except not so close that any of the figure borders d the taskbar no matter where the taskbar is placed. On a small scr overlap. On a large screen, the first four figures would not overly the corners of the other four (unless the screen had an exception)

```
y = rand(1, 100);
                    sz = [700 480];
                                           % data to plo
plt(y,'pos',[ 0
                    0
                                           % bottom left
                        sz]);
plt(y,'pos',[Inf 0
                        sz]);
                                           % top left co
plt(y,'pos',[ 0 Inf sz]);
plt(y,'pos',[Inf Inf sz]);
                                           % bottom righ
                                           % top right (
p = get(findobj('name', 'plt'), 'pos'); % get positic
plt(y, 'pos', mean(cell2mat(p))*1i);
                                           % put 5th pl(
```

The "\*1i" in the above line is strictly necessary to prevent figpose current border information. The raw pixel location is used becau pixel coordinates. With the "\*1i" removed, the last figure would position, but actually the error would probably be too small to ne

The next example also creates four figures at the corners of the s figure is a fixed size and the remaining figures are tiled so as to screen.

```
plt(y,'pos',[ 0 0 600 400]); % figure 1 is pla
plt(y,'pos',[ 0 440 600 -Inf]); % use all the rer
plt(y,'pos',[615 0 -Inf 400]); % use all the rer
```

plt(y,'pos',[615 440 -Inf -Inf]); % use all the rer

Note that in the example above an extra 15 pixels in width and 4 small gap between the four figures.

In addition to the examples above, a good way to appreciate the demoplt.m and cycle thru all the plt demo programs using the the various windows are placed on the screen. You will apprecia even more if you are able to rerun the demos using a different sc location. Without the figpos function, many of the demos would their figures at appropriate screen positions.

If called without an argument, screencfg attempts to determine t automatically. If the automatic procedure fails, then the predefin the file TaskbarSZ.m is assumed (which may be edited if nee or 5 element row vector called the "border vector". This border 'function's return value) is written in text form to screencfg. screencfg.m), and is also saved in the 'border' application pi

If called with a vector argument, then the supplied argument is t

**screencfg.m** (screen configuration) If called with a scalar argument, then screencfg first looks for th this property exists, then its value is returned and nothing furthe property does not exist, then screencfg will look for the screencf values stored there and will also save this vector in the 'border' p does not exist, then screencfg will behave as if it was called with

When a 4 element vector is used for the border vector, its form i where each number represents the number of pixels of clear space must exist at the four edges of the screen indicated. When a 5 elements

[left	bottom	width	height	0]	The largest visible the standard Matl
[left	bottom	width	height	1]	Same as above ex are used.

# A general purpose 3D surface (waterfall) di

#### **Calling sequence:**

## pltwater(z, 'Param1', Value1, 'Param2', Value2)

- All arguments are optional except for z which is a matrix c
- Note that the arguments are arranged in Param/Value pairs.
- However you may omit the Value part of the pair, in which

#### pltwater recognizes the following 12 Param/Value pairs (case

'go',1	The animation begins immediately as if you
'go'	The same as above (since an omitted value
'run'	The same as above. Note that all the sliders may be adjusted even when the display is rι
'invert'	The surface is displayed upside down.
'transpose'	The surface is rotated by 90 degrees (x/y sv
'delay',v	A pause of v milliseconds occurs between c is supplied, it may be changed later using th
'nT',v	Determines how many traces will be visible number of visible traces using the slider. If traces will be used (initially).
'skip',v	Initially v records (rows of z) are skipped b v=1 only every other record is used.) This v slider.
'dx',v	Successive traces of the waterfall display an which adds a visual perspective. (No perspective walue may be modified with the slider.
'dy',v	Successive traces of the waterfall display ar by v percent of the Zaxis limits. This value
'x',v	Specifies the x values corresponding to each not supplied, the value 1:size(z,2) is used.
'y',v	Specifies the y values corresponding to eac supplied, the value 1:size(z,1) is used.

**pltwater.m** (waterfall plots)

	Line smoothing is a line property in most vesupported in R2014b or later). If this param smoothing is enabled when the display is ru otherwise. That behavior may be modified a		
'smooth',v	<pre>• 'smooth', 1 • 'smooth', -1 • 'smooth', 0</pre>	Line smoothing is Line smoothing is The default line s	
	If you are using a version of Matlab that do pltwater will not enable line smoothing mor parameter.		

If a parameter is included in the pltwater argument list that is no parameter along with its corresponding value are passed onto pluused in the pltwater argument list are:

'TraceC'	'CursorC'	'FigBKc'	' L
'Title'	'FigName'	'Linewidth'	' L

Refer to the **wfalltst**. m demo program to see an example **pltwater**.



# **Programming examples**

In the demo folder you will find 31 example programs to help you learn how to take advantage of many of plt's features. Also included in the demo folder is demoplt.m, a script which makes it easy to start any of the example programs by clicking on the appropriate button or to run all of them in sequence (in the order listed below) by clicking the "*All Demos*" button.



I strongly recommend running through the *All Demos* sequence at least once. Many of the questions emailed to me about plt are something like "can plt do xxxxx?", but if they had only clicked through the *All Demos* sequence once, most likely they would quickly discover that the answer to this question was "yes". Running demoplt is also a good way to verify that plt is working as properly on your system. Just type cd plt\demo and then demoplt at the command prompt. The cd command is not necessary if you have added the plt\demo folder to the Matlab path - which is done automatically if you have installed plt as a toolbox. (Installing plt as a toolbox is possible with Matlab R2014b or later.)

plt5.m is first on the list because it is the simplest most basic example. The other demos appear in alphabetical order. As each demo is run, you may peruse the code for the demo program currently being run in the demoplt list box. Also the number of lines of uncommented code appears in the lower right corner of the figure to give you an idea of the complexity of each example. Use the list box scroll bars to view any portion of the code of interest. If the text is to big or small for comfort, adjust the fontsize using the fontsize popup menu in the lower right corner of the demoplt figure. This fontsize is saved (in demoplt.mat) along with the current figure colors and screen location so that the figure will look the same the next time demoplt is started. (Delete demoplt.mat to return to the original default conditions.)

If you are running a version of Matlab older than 7.0 then the guil button is replaced by the guilv6 button because guil.m uses a uitable which is not supported in Matlab 6. (The uitable is replaces with a radio button in quilv6). Similarly guil is replaced by guilv6 if you are running a version of Matlab older than 7.0 or if you are running version 8.4 (R2014b). guil uses a uipanel which isn't supported in Matlab 6, so guilv6 replaces the uipanel with a uicontrol frame which serves pretty much the same function. R2014b supports the uipanel, but the v6 version is run because of a bug in R2014b relating to the stacking order of a uipanel.

In addition to its main role as a demo program launcher, demoplt demonstrates the use the ColorPick pseudo object. (A pseudo object is a collection of more primitive Matlab objects, assembled together to perform a common objective.) The ColorPick pseudo object is useful whenever you want to allow the user to have control over the color of one of the graphic elements. In demoplt there are 4 such elements: The text color, the text background color, the button color, and the figure background color. The ColorPick window is activated when you click on any of the three small color squares (frames) or if you right-click on the figure background edit box. When the ColorPick window appears you can use the sliders or the color patches to change the color of the respective graphic element. For more details, see the <u>Pseudo objects</u> section in the help file.

An optional feature of the ColorPick object is the color change callback function - a function that's called whenever a new color is selected. This feature is demonstrated here by reporting all color changes at the top of the listbox (i.e. before the example code listing).

Although it's unrelated to plt, demoplt also demonstrates the use of the close request function, which in this example is assigned to demoplt(0) and gets called when you close the demoplt figure window. If you have changed the figure size, the fontsize popup, or any color selection this close request function brings up a modal dialog box consisting of these three buttons:

- Save setup changes (will create a "demoplt.mat" file)
- Exit without saving
- Reset to default settings (will delete the demoplt.mat if it exists)

# CLICK on the thumbnails below to see the full size image



This is a simple script which creates a plot containing 5 tr Hopefully you have already been running this script while following through the earlier sections.

- Note how the five y-vectors are combined to form a argument.
- Note the use of the 'Xlim' and 'Ylim' argument control the initial axis limits.
- Note the use of 'LabelX' argument to assign a lal x axis.
- Note the use of the **'LabelY'** argument to add a la both the left and right-hand axes.

Note that plt will use a right-hand axis since two labels w included in the LabelY parameter. Usually the 'Right parameter is included to specify which traces are to use th axis, but in this example the parameter was omitted, so pl to putting just the last trace on the right-hand axis.

This function displays many of markers with random shaj colors starting at the plot origin and then randomly walkin bouncing off the walls. Click on the Walk/Stop button and stop the motion.

- plt creates 513 line objects. All but the last line object displaying the markers (each marker displayed is act line object containing just a single point). You can co many of these markers are visible and in motion. The object is used to display the arrows representing the of each marker.
- The popup control on the left controls the size of the arrows. This popup was created using the "super but



which means you just click on the popup to advance next larger size. After "large" it will wrap around to (which inhibits the display of the velocity arrows). If to actually open the pop menu to observe your choic right-click on the popup. As with the other controls, modify the control even while it is walking.

- The input argument determines the starting number ( markers, i.e. bounce(88) will display 88 markers. If l called without an argument, a default value will be a (128 markers).
- While the display is walking, the number of updates second is computed and displayed in the figure title while the display is walking, you can change the nur markers that are visible and moving. (The slider belo plot).
- The slider on the left controls the walking speed. The update rate (which actually proceeds as fast as possil actually controls how far each marker moves betwee display update.
- Shows how to set line properties using cell arrays.
- Shows how plt can avoid its 99 trace limit by disabli TraceIDs.
- Demonstrates how to create moving displays by char trace x/y data values inside a while loop.

This is a two part script. The first part creates 3 figures ea showing a different solution to the following problem ... **Draw 12 circles in a plane so that every circle is to exactly 5 others.** 

	The second part of the script draws the solution to the fol
	problem
circles12.m	<b>Divide a circle into n congruent parts so that at l</b>
	part does not touch the center.
	(Hint: as far as I know, the only solution uses n = 12)
	An edit pseudo object is also added below the plot which
	rotate the image and control the rotation speed.



- Demonstrates the utility of using complex numbers t x and y positions of the plotted points.
- Demonstrates using prin to create the Trace IDs.
- Demonstrates how to make circles look true by using the 'Pos' argument (width or height). Also two of are placed as far towards the top of the screen as pos which is done by setting the Ybottom value equal t
- Note that even though the calls to plt for solutions 1 specify same screen location ('Pos' parameter) plt do actually plot them on top of each other. Instead a smis added in this situation, a feature that makes it easie create many plt windows so that any of them can be selected with the mouse.
- The last figure (part 2) shows the use of the Nocurs All options to make the cursor objects and menu bc invisible as well as the 'Ticks' option to select ax marks instead of the full grid lines.

This function shows an example where many GUI control fit into a relatively small space.

- The ten controls above the graph (nine edit text objeone popup text object) all are used to control how the parametric curves in the graph are displayed.
- If we used the traditional Matlab/Windows GUI obje would have had to make the graph much smaller to r room for all these controls.
- In addition, the plt('edit') pseudo objects prove much easier way to modify the numeric values, near matching the convenience of a slider object. The plt('edit') and plt('pop') commands are ( in the <u>Pseudo objects</u> section.

After starting curves.m, right-click on the curve name at t

of the figure to cycle through the 42 different cool looking displays. Left-click on the curve name as well to select free complete list of curves. If you start it by typing "**Curves** then after starting it will cycle once through all 42 curves default rate of one second per curve). **demoplt** . **m** calls way which explains why it starts cycling immediately. If the cycle to proceed at a different rate, you may select the rate with the delay popup just below the Cycle button. WI last curve is displayed the cycling stops and the time it to cycle thru all the curves is displayed in the upper left corr figure. (This a useful as a speed performance measure if y delay to zero.)



- The equations in (reddish) orange just below the graj above the curve name, serve as more than just the x-This specific string is evaluated by Matlab to compu points plotted on the graph.
- The vector t, and the constants a, b, and c that appear equations are defined by the controls above the grapl Experiment by both right and left-clicking on these c
- For the cases when more than one trace is plotted, th control on the left (labeled "trace") indicates which t effected by the other nine controls above the graph.
- Note that when you left-click on a control, it will inc decrease depending on whether you click on the left side of the text string.
- Separate values for a, b, and c are saved for each trace multi-trace plot. This explains the variety of curves the appear for a single set of equations (shown below the Left-clicking on the "Default" button will reset all the parameters to their initial settings for only the function currently selected. It will have no effect on the setting remaining 41 curves. However if you right-click on the "Default" button, then the settings for all 42 curves vert to the values they were initialized to when the current program started.
- Note the help text (in purple, center left) tells you jue to get started using the program, even if you haven't of the documentation. This was added using the

'HelpText' parameter. Selecting a different curve popup pseudo object) will erase the help text and rig clicking on the "Help" tag in the MenuBox will mak reappear.

• For most of the curves there is also some text (in gre plot area that describe some technical or historical in related to the curve, hopefully making this program interesting and educational. The text is embedded in table that stores the curve name, equations, and paral the beginning of the text string are some codes that s text position and font size.

This function displays a simulation of Sam Loyd's carniva game. - You bet 1 dollar to play (rolling 3 dice). If one six you get 2 dollars, if two sixes appear, you get 3 dollars, if sixes appear, you get 4 dollars, and otherwise, you get not this a good bet to make?)

- Three traces are created: accumulated winnings, earr bet, and expected earnings per bet.
- The first two traces are displayed as they are comput every time the dice are rolled, a new value is append trace and the plot is updated so you can watch the fu grow in real time.
- A second axis is added near the top of the figure to s dice. For each die, a line with dots as markers is add each of the six faces, with only one of these lines bei at a time. A square patch is also added for each die for visual effect.

There are three ways you can start the program:

- **dice** sets up simulation. No bets occur until a button.
- **dice(n)** sets up simulation & makes n bets.
- dice(0) sets up simulation & makes bets contin until you click stop.



This function demonstrates the usefulness of plt's data edi capability. Two plots are created, the lower one showing t and zeros of a z-plane transfer function and the upper one showing the magnitude and phase of it's frequency respor frequency response plot automatically updates even while dragging a root to a new location. At first the updating in (i.e. while you are dragging) may not seem so important, you use the program its becomes clear that this allows you feel for how the root locations effect the frequency respor This real time motion is accomplished by using the Moti parameter (see line 131). In addition to demonstrating var features, my other goal for this little application was to cr to help engineering students develop a feel for how the m & phase response reacts to a change in the positions of the function poles & zeros. This application won't make muc until you have learned to think in the z-plane. If you have learned this, I highly recommend *Sitler's Notes* - a paper subject which is just about as old as the subject itself, yet nothing else quite as good has been written since. This pa never officially published, but the good news is you can f my web site (<u>www.mennen.org</u>) in the section titled "Sigr processing papers".

- When the program first starts, text appears in the pol plot that tells you how you can use the mouse to mov roots of the transfer function. However it is easy to r important instructions since they disappear as soon a click on anything in that figure widow. (This was nemanage clutter). However you can re-enable the help any time by clicking on the yellow "editz help" tag v centered near the left edge of the figure window. (No right-clicking on the Help tag in the menu box has th effect.)
- In the frequency plot, the x-cursor edit boxes show the location as a fraction of the sample rate. The Xstriparameter is used to show this as an angular measure degrees) just to the right of the x-cursor readout. Sin DualCur parameter is used, there are two y-cursor edit The first one (green) shows the magnitude response



the second one (purple) shows the phase response in The Ystring parameter is used to show the magnin response in linear form (in percent). Note that after t command, the Ystring is moved to the left of the plo by default the Ystring appears in the same place as tl cursor. The alternate location allows room for a mult Ystring which is generated compliments of prin's c output feature. The AxisLink parameter is used so default the mag/phase axes are controlled separately.

- In the pole/zero plot, the x and y-cursor edit boxes sl pole/zero locations in Cartesian form. The Xstring p shows the polar form just to the right of the x-cursor
- Normally plt's data editing is initiated when you righer either the x or the y cursor readouts. However whener editing is being used extensively (as in this program) useful to provide an easier way to enter editing moder program, this is done with the patch object that appered below the traceID box. (The patch object is created control 146 of this file). The 'Dedit' application data variable (see lines 137 to 139) to change the default editing not the usual default (change only the y coordinate) to the alternative (allow changing both the x and y coordinate) to the application data variable 'EditCur' (see line 140) change the default size of the cursors used for editing the default size of the cursors used for editing the default size of the cursors used for editing the default size of the cursors used for editing the default size of the cursors used for editing the default size of the cursors used for editing the default size of the cursors used for editing the default size of the cursors used for editing the default size of the cursors used for editing the default size of the cursors used for editing the default size of the cursors used for editing the default size of the cursors used for editing the
- Notice that while dragging a pole or a zero to a new the pole or zero remains inside the diamond shape ec ... EXCEPT when you get close to the x axis. At that root jumps out of the edit cursor and sticks to the x a long as the edit cursor remains inside the green band this snap to behavior it would be nearly impossible t purely real root. Similarly, when you drag a zero (bu pole) "close" enough to the unit circle, the zero will the circle. Without this feature it would be difficult to transfer function with a symmetric numerator polync
- How "close" is close enough for these snap to operat Well this is determined by the Tolerance slider whicl lower left corner of the pole/zero plot. Notice that as move this slider, the width of the green band surrour x-axis and the unit circle gets bigger. To disable the s

feature, simply move the tolerance slider to 0.

- Shows how you can take advantage of both left and 1 actions on a button. left-clicking on the "Delete P/Z" deletes the root under the cursor as you might expect clicking on this button undoes the previous delete. T multi-level undo, so you could delete all the zeros ar restore them one by one by successive right clicks of Delete P/Z button. These buttons can also be used to collection of N poles to a collection of N zeros at the locations. To do this, deleting the N poles, then click zero, and then right-click on the Delete P/Z button N (Of course you can similarly change zeros to poles.)
- Demonstrates the use of the 'Fig' parameter to put in one figure with each plot possessing all the feature available to any single plot created by plt.

This script plots the results of combining uniform random variables.

- Shows the advantage passing plot data in cell arrays traces contain different number of data points.
- Shows how the line zData can be used to save an alter data set which in this example is the error terms inster actual correlations. A checkbox allows you to tell the show the alternative data set. The label for the check rotated 90 degrees so that it can fit in the small space left of the plot.
- Note the use of the 'FigName' and 'TraceID' ;
- Note the appearance of the Greek letter in the x-axis
- Shows how to use the 'COLORC' argument to selec default plotting colors (typically set to use a white ba for the plotting area)
- The 'Options' argument enables the x-axis curso (which appears just below the peak/valley finder but enables the menu bar at the top of the figure window Print tag to the menu box, and lastly removes the Lin and LinY/LogY selectors from the menu box.
- Shows how to use the 'DIStrace' argument to di





some traces on startup.

- Shows how to use the 'MotionZoom' argument to new plot showing only the zoom window. Admittedl more contrived than useful in this particular script, b this example will clarify the function of the MotionZ parameter.
- The zoom window plot also demonstrates an easy we the trace data from one plot to another (in this case fimain plot to the zoom plot).
- The first trace is displayed using markers only to dis the true Gaussian curve.
- Demonstrates the use of the 'HelpText' parameter initialize a GUI with user help information that is cle when the user begins to use the application. In this cle 'MoveCB' parameter is used to cause the help text removed when you click on the plot. The help text is removed if you click on the checkbox. If you want the text to reappear, simply right-click on the help tag in MenuBox.

Usually plt is used to build gui applications which include however this example doesn't include plots so that it rema making it a good example to start with if you have no preexposure to Matlab GUI programming. The only pseudo ( used in gui1 is the pseudo slider which is a collection of 5 uicontrols designed to work together to control a single pa The remaining controls used in gui1 are standard Matlab



This GUI doesn't actually perform any useful function oth demonstrate how to create various controls and move thei until the GUI appears as desired. The slider callback gene random numbers for the listbox, textbox, and uitable. The remaining callbacks are just stubs that notify you that you on the object.

You can most easily absorb the point of this example (and following example called gui2.m) by reading this section help file: <u>GUI building with plt</u>.

**guil.m** uses a uitable which aren't supported in Matlab you are running a version of Matlab older than 7.0 then yer run an alternate version of this program called **guilv6**. replaces the uitable with a radio button. If you start guil f demoplt, demoplt checks the Matlab version and runs gui guilv6 as appropriate.

Unlike the previous gui building example (gui1.m) this or includes a plot and actually performs a useful function - d the frequency response of the most common traditional ar filters. GUI controls are provided to adjust the most impo parameters (Filter order, Cutoff frequency, & Passband/St ripple). The capabilities of this program were kept modes it a good introduction to GUI programming with plt.

gui2 creates these eleven pseudo objects:

- 1. a plot
- 2. a cursor
- 3. a grid
- 4. an edit object (filter order)
- 5. a popup (filter type)
- 6. a popup (decades to display)
- 7. a popup (number of points to display)
- 8. a slider (passband ripple)
- 9. a slider (stopband ripple)
- 10. a slider (cutoff frequency)
- 11. a slider (frequency 2)

The first three pseudo objects in this list are created by the to plt and the remaining eight pseudo objects are created v additional calls to plt.

#### gui2.m

Although Matlab already has objects with similar names, pseudo objects are different. They provide more utility an The pseudo objects 4 thru 7 listed above are grouped insiduipanel titled "Parameters".



You can most easily absorb the point of this example (and previous one called gui1.m) by reading this section of the <u>GUI building with plt</u>.

There are two alternate versions of this application includ demo folder. The first one, called gui2v6.m uses a uiconti in place of the uipanel. This alternate version should be u are running a version of Matlab older than 7.0 because M does not support the uipanel. Actually the alternate versio also be used if you are running R2014b or newer. The rea this is that although the uipanel is supported, a bug relatir uipanel's stacking order prevents gui2 from working prop those versions. If you start gui2 from demoplt, demoplt cl Matlab version and runs gui2 or gui2v6 as appropriate.

The other alternate version is called gui2ALT.m and is no demoplt. This version differs from gui2.m primarily in the of traces used. gui2 uses 10 traces (5 for magnitude on the and 5 for phase on the right axis) where as gui2ALT uses axis with just 5 traces. The trick to make this work is to u trace to display both the magnitude and the phase informa Although I eventually decided that the 10 trace method in simpler, the alternate version is included because in some this trick can still be useful. Note that the tick marks are r so that they read in degrees in the phase portion of the plc the phase portion is highlighted with a gray patch to bette it visually from the magnitude plot.

The intent of this example is to demonstrate the generality image pseudo object by including two of these objects in figure, and to demonstrate the use of the 'Fig' parameter a several other graphical programming techniques. It's easy dozens of Julia set graphing programs in nearly every lan (including Matlab) so I wouldn't fault you if you were ske the need for yet another application with this purpose. Ho

#### julia.m



goal was to leverage the power of the plt plotting package how fun it is to explore Julia sets and to make this applicat compelling than any similar application out there. I'll let y judge how well I have met this challenge.

Julia set images are traditionally generated with the repea application of the equation  $\mathbf{z} = \mathbf{z}^2 + \mathbf{c}$  (z and c are compl application also allows exponents other than 2 (called the generalized Julia set). The color of the image is determine number of iterations it takes for the magnitude of z to gro than some fixed value (2.0 for this program). The Mandel uses the same equation and the same color assignment me differs in how the equation is initialized.

Some very basic instructions appear in the figure when th application starts but this help text disappears as soon as y anywhere in the plot region. For a complete description o application, see <u>A Julia set explorer</u>

This function plots a series of 40 random bars and display horizontal threshold line which you can move by sliding t along the vertical green line. As you move the threshold l string below the plot reports the number of bars that exce threshold. (This demonstrates the use of the plt xstring parameter.) These two buttons are created:

- *Rand*: Sets the bar heights to a new random data set.
- *Walk*: Clicking this once starts a random walk proce bar heights. Clicking a second time halts this process Walk button user data holds the walk/halt status (1/0 respectively) demonstrating a simple way to start and moving plot.

Note that you can move the threshold or press the Rand b while it is walking. Also, if you click on one of the vertice bars, the horizontal threshold bar will then follow the upp that vertical bar.



If movbar is called with an input argument, the value of tl argument is ignored, but movbar will start as if the walk t been hit.

This script is an expansion of the simple plt5.m example the demonstrate additional features of plt.

Note that two plots appear in this figure. There are two m that you can use with plt to create figures containing mult The first is to use the subplot parameter to create multiple with a single call to plt. (This is demonstrated by the subr subplt8.m, subplt16.m, subplt20.m, pub.m, pub2.m, pltma weight.m programming examples). The second method (v used here as well as in the pub3.m example) is to use a se to plt for each plot. The first plot (upper) is created by a c that is guite similar to the one used in the simple plt5.n plt creates the figure window as usual and then creates the plot inside the new figure. Both a left and right hand axis for this plot. We are free to put as many traces as we want the left or right hand side, although in this example we pu traces of this plot on the left hand axis except for the last Tr40) which is put on the right hand axis (and is also dra thicker trace). The two major differences between this (fin plt and the plt call used in plt5.m are:

- 1. The number of traces has been expanded from 5 to 4 Without additional action, this would create a TraceI (legend) containing 40 trace names in a single colur However this would not work well or look good to c a long list into the small space available. To solve the the the TIDcolumn parameter has been used to create TraceID box with two columns. The 'TIDcolumn included in the plt argument list actually specifies th of items to put in the second column, which in this e means that both columns will contain 20 items.
- 2. The 'xy' parameter is included to specify the locat plot within the figure window. This wasn't needed be



plt5.m) since the plot was automatically sized to fill figure window. But now we want to create the plot in portion of the window to leave room for a second plo created. The object ID (-3) in the first row indicates position is to be used for both the left and right axis that all the cursor object positions should be position relative to these axes. (ID 0 also refers to both left & axes but does not cause the other cursor objects to be repositioned as well). Although plt makes its best gu positions of the TraceID and Menu boxes often you to reposition them with the xy parameter to make be the available space. The 2nd row of the xy matrix rel the TraceID box. The last row repositions the Yaxis which otherwise would have been covered up by the box. Note that you don't need to figure out the numb xy matrix since they will be reported to you as you a positions of the screen objects with the mouse. (See description of the xy parameter in the Axis propertie of the help file.)

Following the first call to plt which displays the first 40 ti second call to plt is used to display the remaining 10 trace below the first one. As before we use the 'xy' parameter the plot to fit in the remaining open space of the figure. A first plot, we also include both the left and right hand axis again free to put as many of these traces as desired on eitl but we choose to put only trace 5 on the right hand axis w remaining traces on the left axis. The most important diffe between the first and second plt calls is that in the second include the 'Fig',gcf in the parameter list. (gcf stands for " current figure handle"). This tells plt not to open a new fig the plot as usual, but rather to put the plot in the specified The 'Fig' parameter must be either at the beginning or at t the plt parameter list. (All other plt parameters may be pla anywhere in the parameter list). You may notice that the *y* parameter for this plot includes an imaginary component element of the axis position. The reason for this is that sir sizes of the cursor objects are relative to the plot size. Thi sometimes makes the cursor objects too small when the p

small fraction of the figure size. To fix this problem, one a minimum width or height in the imaginary component c width and/or height values.

A few other features of the first (upper plot) are worth poi

- With so many traces, the ability to use the legend (i. TraceID box) to selectively enable or disable individ becomes even more compelling. Although the traces legend are color coded, it's difficult to distinguish ev based on color, so clicking on a legend item is often uniquely identify a trace.
- The 'Pos' parameter is used to increase the figure 30% from the default of 700x525 pixels to 830x550. gives room to fit both plots into the figure area with overcrowding.
- The 'HelpFileR'' parameter is used to specify w file will appear when you right-click on the Help tag menu box. Normally the file specified will contain h currently running script. In this case prin.pdf is just u example and in fact has nothing to do with plt50.
- The use of the 'closeReq' parameter is shown, a this case the function specified merely displays a me Look at the gui2.m and wfall.m demos to see examp somewhat more sophisticated close request function:
- In situations like this with so many traces on the plot difficult to find the cursor. The line following the firs plt solves this problem increasing the cursor size from as well as by changing the cursor shape from a plus a sterisk.

The main purpose of this function is to demonstrate the us **image** pseudo object. The subplot parameter is used to p the figure into two parts. The left part displays a conventi plot which includes the following five traces.

1. A circle (green trace) whose radius is controlled by i amplitude slider (the leftmost slider above the plot ir

amplitudes" section).

- 2. A hyperbola (purple trace). Its amplitude slider contu asymptote slope.
- 3. A polygon (cyan trace). Its amplitude slider controls size of the figure and the number of sides (which ran to 7).
- 4. Two lines (red and blue traces). Their amplitude slid the lines' positions as well as determines the lines' or (vertical/horizontal).

The five Z amplitude sliders assign a z coordinate to each five traces, then these 500 points (100 points per trace) ar interpolated using griddata to create the two input functio displayed using an intensity map on the right side of the f

The many features of pltmap and the image pseudo object intertwined so to help you explore these features, conside up pltmap and running through these tasks:

- Adjust some of the "Y amplitudes" (5 sliders near the left corner) and observe how they affect their respect (1-5). Note that the intensity map changes as well sin values are computed from the shapes of these five transmission.
- Disable and enable the various traces by clicking on • names in the TraceID box. Note that the intensity ma determined only from the enabled traces. A strange t happens if you disable all the traces except for trace 5 (showing just a single horizontal or vertical line). V happens is that the intensity map no longer represent associated with the 2D plot. This is because the grid function is used to interpolate the data from the 2D p fails because it doesn't have enough data when provi just a single line of zero or infinite slope. So when the detected, pltmap creates an alternate intensity map sl 2 dimensional sync function with a random center pe (For variety, this sync function also appears for a the seconds of a moving display [see the description of t button below], but note that this happens only the fir Run button is pressed after pltmap is started. Subseq

presses of the Run button simply start the intensity n updating in the usual fashion.

- Adjust the "Z amplitudes" for each of the 5 traces us sliders. The z values of the traces are not plotted in t (left) but note how these amplitudes affect the intens
- Click on the "color bar", the vertical color key strip 1 upper left corner of the intensity map. Note that this color map through seven choices. Some of the color have a particular purpose, but mostly this is simply a preference.
- Note that the intensity map appears somewhat pixela is because it is composed of a relatively small numble pixels (200x200). Try zooming in on an interesting leregion of the intensity map using a zoom box. Hold they down and drag the mouse to create the zoom box click inside the zoom box to expand the display. Eve you still have only 200x200 pixels, the display will lesmoother because all these pixels are focused on a si region of the more quickly changing z data. You can zoom in by right-clicking on the "view all" button in right corner. Then left-click on the view all button to the limits back to their original values to show the er set.
- Also try opening a zoom box in the 2D plot (left). Ye this as before (shift key and mouse drag) or try the " click and drag" mouse technique which avoids havin the keyboard. You may be surprised to see that the ir map zooms to show the region inside the zoom box ( plot even as you are dragging the edge of the zoom t then click inside the zoom box the 2D plot will also show just the region inside the zoom box ... but let's this just yet. First try moving the zoom box around. clicking the mouse near the mid-point of any edge, a the zoom box around while holding the mouse butto: Also note that if you drag one of the corners instead midpoint then the zoom box changes its size instead position. In both cases the intensity map continues to so that it shows only the zoom box region. These mc motions are further described in the paragraph titled



"Adjusting the expansion box" in the <u>Zooming and</u> section.

- Try sliding the resolution slider (just to the left of the bar) all the way to the top of the slider. This will sele resolution of 800x800 (16 times as many pixels as be the display will look much smoother, but the drawba the update rate will be much slower. Try moving the the way to the bottom (50x50 pixels). Now the inten will look very blocky, and the update rate will be ver Note that when you click inside the intensity map, th will center itself on one of the blocks even if you clic the edge of one of the blocks. This makes it easier to the Z value cursor readout (shown below the intensit Also note the x and y cursor readouts are updated as would expect every time you click on the image. Res resolution slider to 200 before continuing.
- Try adjusting the "edge" slider (right below the color default value of the slider is "1" which means that or data range between  $\mu = \sigma$  and  $\mu + \sigma$  is used for assignin of the z values, where  $\mu$  represents the mean of the z means that all the data bigger than 1 standard deviati the mean is represented by the same color. If you mc slider to "2", then two standard deviations of the data so that you can see variations near the extremes that couldn't see before. But the downside is that you wil detail for smaller changes in the z values closer to th You can also adjust the mid point for the range of for the mid slider (just to the left of the edge slider). For if you are more interested in getting a view of the da than the mean you might set mid=.5 and edge=.8. Th mean that the range of data that produces different co the image would be from  $\mu$ -.3 $\sigma$  to  $\mu$ +1.3 $\sigma$ .
- Although it doesn't really demonstrate any more feat the new image pseudo object, if you really want to b mesmerized by the display, press the "Run" button. happens is that a random selection of the 10 sliders a 2D plot are selected to start moving. (The remaining that are held fixed are made invisible so you can easi what is changing). As the sliders are moving up and

both the 2D and 3D plots are continuously updated t the new information in the sliders. As this is happeni will see the small (blue) frame counter below the Ru button counting down from 100 to zero. When zero i a new random selection is made from the set of 10 sl the frame counter begins down counting anew from While all this is happening, you may change the spec to adjust the motion rates and you also may adjust pr everything else mentioned in the above bullet points find that 100 frame count is too long or short for you simply click on the yellow "100" and you will be pre with a popup menu allowing you to vary this frame c as small as five to as large as 1000.

Similar to plt5 and plt50, except that this is a functi of a script. This function takes an argument which specifi many traces to plot. For instance pltn(1) will plot a sir and pltn(99) will plot 99 traces. If you specify more to this, the trace IDs are not displayed (since there will not b for them). pltn with no arguments does the same thing a pltn(99). You can change the number of traces plotted after pltn is already running by entering a new number in lines" edit box (under the TraceID box). Try entering "10 this edit box just to see that plt can actually handle such a number of traces! Going much beyond 1000 traces is a gc performance test, since on slower computers you will star a significant lag on pan operations.

pltn.m



- The **TIDCOlumn** parameter is used to divide the tra into up to three columns if necessary. (Showing 99 tu one column wouldn't be practical.)
- TraceIDs are disabled when more than 99 traces a specified. (Otherwise plt would give an error message)
- Uses the 'Ystring' parameter to show a continuc readout of the cursor index
- Uses the 'Xstring' parameter to show a continuc readout of the date and time corresponding to the cu position. Note the edit box form is selected by placir
question mark character at the beginning of the strin

- A popup menu (pseudo object) is created below the : label which allows you to adjust the line thickness. N you can right-click on the popup to increment the lin thickness (which sometimes is more convenient than the popup menu).
- A callback is written for the Xstring edit box that mc cursor to the index with a corresponding time as clos possible to the entered value. For example, try this:
  - 1. Click on the top trace (which makes it easy to s cursor).
  - 2. Enter dates into the edit box e.g. "**30 dec 2006 jan-07 9:59**", etc.
  - 3. Verify that the cursor moves to the correspondiu

This function demonstrates the plotting of quivers, polynci interpolation, and the use of several of the plt callback fui (moveCB, TIDcback, MotionEdit).

The Pquiv.m function appears three times in the pl argument list to plot three vector fields. The first two fields (named velocity1 & velocity2) both have their locations specified by f (also plotted on the green trahumps/20) and the arrow lengths are specified by v1 respectively. The first of these Pquiv calls is somewl to the Matlab command

quiver(real(f), imag(f), real(v1), ima
The third Pquiv call generates the vector field showr
which includes only six vectors.

- Uses the xy parameter to make room for long Trace
- Uses tex commands (e.g. \uparrow) inside Trace ID
- Reassigns menu box items. In this example, the Lin replaced by a Filter tag. Its buttondown function executed when you click on 'Filter') searches for the (using findobj) and swaps the contents of its user dat axis data.
- The 'HelpText' parameter is used to identify fea the plot and to explain how to modify the Hermite in



function. This help text disappears as soon as you methe yellow arrows (as described in the yellow help te

- Uses NaNs (not a number) to blank out portions of a this case, the NaNs were inserted into the x coordina although using the y or z coordinates for this purpose equally as well.
- Uses the TraceID callback function (**TIDcback**) to an action when you click on a trace ID. For example you click on the forth trace ID (humps+rand) this wi in the command window: "A trace named humps+ra color [1 0 0] was toggled". Although this TraceID ca not particularly useful, it was contrived to demonstra @ substitutions.
- A MotionEdit function is provided which serves thes purposes:
  - The trace data is updated as you drag the edit cu Without the MotionEdit function the trace data updated when you release the mouse button after cursor has been moved to the desired position. ( this on trace 1 will give you a good feel for what means.)
  - 2. For the quiver traces, moving the arrow position not normally move the "v" portion of the arrow you would hope. This MotionEdit function solv problem by calling Pquiv as the arrow is being
  - 3. If you move one of the arrows associated with t (vectorField) then trace 6 is updated based on a polynomial interpolation which is designed to g tails of all six of the trace 5 vectors. The derival this polynomial are also constrained so that it m slopes of these vectors as well. Use the data edi feature to move the head or the tail of any of the vectors and watch how the interpolated data on (blue) is updated in real time to follow the vector

pltsq.m approximates a square wave by adding up the
odd harmonics of a sine wave. The plot displays successiv
these harmonics which approximates a square wave more

more harmonics are added together. The key point howev the reason this demo was created) is that the amplitudes o sine waves and sums are continually varied (periodically plus and minus one) to produce a "real-time" moving disp well suited to creating real-time displays, but there are a f concepts to learn and this demo is an excellent starting pc

- Type pltsq or pltsq(0) to start pltsq in its store state. (i.e. the display is not updating)
- Type pltsq(1) or pltsq('run') to start pl the display dynamically updating.
- Demonstrates how you can add GUI controls to the j window - typically something you will need to do w creating plt based applications.
- Five pseudo popup controls are added to the figure to of the plot including one "super-button" to start and a plotting.
- The main display loop is only 6 lines long (lines 96runs as fast as possible (i.e. with no intentional pause every second an additional 10 lines of code is run (li to check for new user input and to report on the disp rate. This additional code could be run every time th is updated, but that would needlessly slow down the rate.
- A text object appears below the plot which displays "updates/second" - a good measure of computational graphics performance. The color of this text object is every time it is refreshed so that you can tell the speare recomputed even if the result is the same.
- The 'xy' argument is used to make room for the ps popups as well as for the wider than usual TraceIDs.
- The position coordinates for the five popups are grous single array in the code to make it easy to update the coordinates using the plt move' function. For det how this is done, refer to the gui1 & gui2 examples.
- Normalized units are used here for the uicontrols. Th "plt move" function also handles pixel units which is when you don't want the objects to change size wher figure window is resized.



- The cursor callback parameter ('moveCB') and the plt('rename') call are used to provide simultan cursor readouts for all 5 traces in the TraceID box. T unusual use of the TraceID box, but it serves as an al to the "multiCursor" option (described here) when ye less clutter inside the plot axis. Updating the TraceII every display update would slow the display, so norr cursor is not updated after every display update. How you want the cursor to be updated on every display, box labeled "Live cursor".
- The '**Options**' argument is used to turn off grid litto remove the x and y-axis Log selectors from the m
- You can use the Erasemode popup to explore the effererasemode property on drawing speed. (The erasemode property is no longer supported in Matlab version RZ later, so pltsq.m checks the Matlab version and disat popup appropriately.) You can also effect the drawin by varying the number of points per plot from a low points to a high of 51200 points (32 cycles times 160 per cycle).

This script demonstrates the use of Pvbar . m and Pebai plot vertical bars and error bars respectively. Some things about pltvbar are:

- The first Pvbar in the argument list plots two functio single trace (green) with the 1st function (phase1) de position of the bottom of the vertical bars and the 2n (phase2) defining the position of the tops of the bars
- The second Pvbar in the list plots 3 functions (called bell1, and bell2). The 3 columns of the first Pvbar ar define the x coordinates for those three functions. Th argument (0) indicates that the bottom of all the vert at y=0. The last Pvbar argument gives the y coordinates each of the 3 functions (one function per column).
- The next trace definition (the data argument pair after 'linewidth') plots two traces corresponding to t columns of poly23. The 1st column is a 2nd order

#### pltvbar.m



polynomial and the 2nd column is 3rd order

- The next trace definition uses Pebar function to createrror bar traces, the first trace defined by the first coleach of the 3 arguments and the second trace defined second column.
- The 'Linewidth' argument appears in the middle call to change the width of only the traces defined ea the argument list.
- The 'TraceID' argument is used to assign names trace that are appropriate for the data being displayed
- The 'xy' argument is used to widen the TraceID bo room for the longer than usual trace ID names
- The '+FontSize', '+FontWeight', '+Font, '+Xtick', '+Ytick', arguments are used to more main axis properties of the same name (without the properties)
- The Grid pseudo object is used to create a 8x3 table character data. This table really doesn't have anythin with the plot (and indeed is just filled with random g but it was included just to demonstrate an unusual w this pseudo object.

# pltvar.m

To demonstrate the workspace plotter, this script creates s vectors in the workspace (including a structure containing vector fields) and then starts the workspace plotter by call with no arguments. Workspace plotting is described <u>here</u>

All the other plt examples in the demo folder use plotting appropriate for data exploration (the main design goal of However plt can also use formats appropriate for creating publication. This script demonstrates this by creating thre figures windows. Note that all three windows are created pltpub() which simply calls plt() with several para optimized for creating publishable plots.

- The first window (plot 1 appearing near the top of 1 screen) is a bar chart that demonstrates how to embe data inside the script as comments. It also demonstra use of the prin function to display a table of randou numbers in a text box. The vertical position of the pl depends on the screen size.
- The second window (plot 2 lower left portion of the demonstrates how to distribute 15 functions among 5 by using the 'SubTrace' parameter and how to se colors and line styles.
- The third window (plot 3 lower right) contains two with error bars, shows how to use the TraceID box a The special character ']' is used in the first TraceII disable the shading of the trace name that is normall indicate the trace is on the right hand axis. Also the '+XtickLabel' parameter is used in the plt call t

the tick labels on the x axis. Then an array of text ob used to create specially formatted tick labels. The thi window also demonstrates various ways of modifyin lines, and also shows the use of the "+ - < > ." I modify properties of:

- + the left axis
- the right axis
- < the left y-label
- > the right y-label
  - the x-label
- Demonstrates how to define a new plotting function in this example) which has a different set of defaults for a particular purpose. The pltpub function include
  - Uses the 'COLORdef' parameter to select a w background
  - Uses the 'NoCursor' option to remove the c objects
  - Uses the 'LineSmoothing' option to imprc esthetics
  - Uses the 'TraceID', '' parameter to remov

#### TraceID box

In this example, a plt figure is created in its usual data exp mode showing 6 traces of randomly generated data. Each contains over 50 thousand data points, although the displa zoomed to show only a small portion of the data. The 'xV option is used to enable the xView slider which is particul useful in situations like this where you are viewing only a portion of a long data record. (The xView slider appears *z* primary plot.) The idea is to use the xView slider or other controls to pan and/or zoom the display to some area of it then press the "pub" button to generate a figure containing selected data and optimized for publication.



What makes this more interesting is that when you pan to section of the data and again press the "pub" button, the p figure is redrawn using subplots to show both selected po a like manner, successive presses of the pub button furthe subdivide the plotting area with each new data range appe above the previous ones. To reset the pub figure so that on single axis is plotted simply right-click on the pub button.

The x axis of the data exploration window is plotted in ur days past a time reference (1-Jan-2013 in this example), t date ticks are used on the x axis of the publication plot. To clutter, only the day and month are shown for all vertical except the last one (which includes day, month, & year).

The TraceID box is typically placed to the left of the plot, for the publication figure in this demo the TraceID box is right on top of the plot (more like a legend). This means t sometimes the TraceID box will obscure some of the data that you can easily use the mouse to drag the legend aroun spot that does not interfere with the plot.

This example may seem somewhat contrived - and indeec conceived mostly to demonstrate as many unusual plt par and programming techniques as possible. As with the previous two demos (pub & pub2) multiple p created in a single figure, however a different mechanism In pub/pub2 the subplot parameter is used, which has the of creating multiple plots with a single call to plt. This prouses the 'Fig' parameter instead, and each plot is created v separate call to plt. This provides some advantages over t method, such as allowing each plot to include a traceID b as a right hand axis. Also the position of the plots are con general and don't demand fixed column widths as with su (Note that the positions of the four plots in this example v have been difficult to create using subplots.) In this exam cursors were disabled ('Nocursor' option) since the main § uncluttered publication quality result, but if they cursors v enabled, they would have the full generality and all the pl of single plot graphs (unlike the restricted set of subplot c On the other hand, as the number of plots required on the increases, the restrictions of the subplots are advantageou they allow a more compact plot spacing.

#### pub3.m



The traceID box is enabled for each plot in this example, as a legend, but it can also be used to enable or disable an the figure.

Note that the 2nd trace of each plot (with traceID "samp") consists of 12 superimposed traces. (This is done by delin each of the 12 traces with a NaN element so that a line is from the end of each trace to the beginning of the next.) T have been done by using a separate trace for each of the t "samp" traces, each with their own traceID, but that woul made the legend unnecessarily large and cumbersome. Th trace is the average of the 12 superimposed traces and the (markers only) is the standard deviation of those same 12

The xy parameter contains the positions and sizes of each four plots. Note that a -3 is inserted in front of each of the positions. The -3 indicates that this position refers to both and right axes and also indicates that the traceID box (and cursor controls if they were enabled) are to be positioned

the positons given for the left and right axes. This is descu the description of the xy parameter in the <u>Axis properties</u> the help file.

The **'SubPlot'** argument is used to create 3 axes. plt p single trace on each axes except for the main (lower) axis gets all the remaining traces. In this case, since there are 5 defined, the main axis has 3 traces. Note that the traces ar to the axes from the bottom up so that the last trace (serp) on the upper most axis.

- The 'LabelY' argument defines the y-axis labels f three axes, again from the bottom up. You can also d y-axis label for the right hand main axis, by tacking end of the LabelY array (as done here).
- The 'Right', 2 argument is used to specify that t trace of the main axis should be put on the right hand this argument was omitted, plt would still have know right hand axis was desired (because of the extra y-la LabelY array) however it would have put trace 3 on hand axis. (By default, the last trace goes on the right right hand axis.)
- The LineWidth and LineStyle arguments define the characteristics for all 5 traces.
- The **'TraceMK'** parameter enables the trace select show the line characteristics and the **'xy'** paramete the trace selection box to make room for this.
- Note that all three plots have their own cursor suppo almost all the cursor features. The exceptions are del the xview slider, and the multi-cursor mode. These n still be active but they apply only the the main (lowe
- Only a single x-axis edit box is needed since plt keep cursors of all three axes aligned. Also note that if yo pan any of the 3 plots, the other two plots will adjust axis limits to agree.
- A brief description of this example is added to the sc the 'HelpText' parameter. As you will see in the demo programs, the help text is usually removed wh start using the program, but in this case the help text



place since it doesn't interfere with the plot area or c (However you can toggle the help text on or off by r clicking on the Help tag in the menu box.)

This script shows a slight expansion of the ideas found in by increasing the number of axes from 3 to 8. The axes ar arranged in two columns which allows the use of two diff axes (one for each column).

- Note that the four axes on the left are synchronized v other as well as the four on the right, although the ler right halves are independent of each other and have a axis limits and units.
- There are 11 traces defined in the plt argument list be axes are specified. The extra 3 traces go to the main (lower left). This means that the first 4 traces are on plot and the remaining 7 traces are assigned to the ot subplots.
- Although the black background used in most of the e programs makes it easier to distinguish the trace cold people prefer a white background and this script sho do that by using the 'ColorDef' parameter to sele Matlab's default color scheme. Matlab's default trace order only includes six colors and this may not be lo or ordered ideally for a particular graph. The ColorE parameter may be used to set the trace colors as desi in this example) the ColorDef parameter is a color specification (3 columns of numbers between zero a this color spec is used instead of Matlab's current tra order default. The first line of this script defines this order using Matlab's traditional style. The 2nd line d exact same color sequence using an alternate style al plt which you may also use if you find that more cor than the traditional style. There's a special case (not for the first entry in this color array. If it's [.99.99.9 999999 in the alternate style) then the remaining colappended to the Matlab default color trace order. Thi convenient if for example you just want to add a few



the end of the list instead of merely replacing the wh trace sequence.

• One advantage of the white background is that it is e publish a screen capture since the colors will not nee inverted. Remember that for publishing you can redu clutter of the capture by temporarily removing all the and their associated controls and readouts. You do the right-clicking on the y-axis label of the lower left ple ("main"). Right-click a second time to re-enable the

This short script again is a slight complication from the plexample (supblt8). Not only do we double the number of we take advantage of all the features of the subplot argum varying the number of plots in each column as well as adj vertical and horizontal spacings.

- Note that the whole number parts of the subplot argu specifies the plot widths and heights where as the fra parts specifies the horizontal and vertical spacing be plots.
- So for example the "99.04" near the end of the subpl argument (for the rightmost plot) means that this plo occupy 99% of the available height. The fractional p that the space below the graph should be increased b percent of the height of the available plotting area.
- Also remember that the negative numbers in the sub argument are used to break up the plots into columns example, the "-25.96" value tells plt that the first col should contain four plots (because it follows four po numbers). The whole number part (25) means that th column should use up 25% of the available plotting fractional part (.96) means that we want to reduce th spacing to the left of this column by 4% of the plotti (The default spacing results in a comfortable easy-or layout, but sometimes we want a tighter layout so we bigger plots.) For a more complete description of the argument, refer to the <u>Axis properties</u> section as well <u>GUI building with plt</u> section of the help file.



 As in the previous example, the cursors for the vario each column are linked to each other, but are not linl way to the cursors of the other columns. So for exam move the cursor in the "tribell" plot (top of column 2 cursors of the four plots below it will also move so tl all point to the same x position. Also if you pan or zo axis of the tribell plot, the x-axis of the four plots be also be zoomed or panned so that the x limits remain for the entire column. This is what we call the subplo mode. The unlinked (or "independent") mode is dem in the next example program (subplt20).

The default subplot "linked" mode (demonstrated by the I subplot examples) makes sense when the columns share a x-axis. However in this example the plots do not share a c axis, so the "independent" subplot mode is more appropri tell plt to use the independent mode by putting an "i" after number of the subplot argument (Note the "32i" in the sul argument of this example).

- The only thing now shared between the columns is s displaying the cursor values. For example, the x and boxes below the first column display the cursor valu plot that you last clicked on in that column. The colc edit boxes changes to match the color of the trace that clicked on so you can tell at a glance which plot the values refer to.
- One advantage of the independent mode is that we c more plots into a given space. We could probably dis these 20 plots using the linked mode as well, but the window would have to be very large since in the link a separate y-axis cursor edit box is included for ever
- As with the previous subplot examples, there are mo than axes (21 traces and 20 axes). That means the fir (lower left) gets 2 traces and a traceID box is added you to select which one to display (or both).
- In this example all 21 traces contain the same number points (301). However this was just done for the con



of the code generating the fake data to display. Each traces could include a different number of points and would work equally as well.

• As you experiment with these plots, be aware of the of the "current cursor" (or "current plot" if you prefe is important since there are 16 different cursors visib current cursor is the cursor belonging to the last plot clicked on. When you click on one of the five menu (LinX, LinY, Mark, Zout, XY<->) the appropriate m operation will only be applied to the current cursor. I for the up/down arrow buttons (peak/valley finder) a the "circle" button which toggles whether markers ai positioned over the trace data values. The only except Delta button (delta cursor). This always operates on plot (lower left) regardless of which cursor is current.

This script file creates two plots each consisting of 9 trace plt tricks and features are demonstrated:

- Note that these figures plot multiple valued function relations).
- The first plot (efficiency and range chart) creates a tu each column of gph and mpg (9 columns for 9 altitut
- Demonstrates adding an additional axis to show alter on the right hand and/or top axis
- Demonstrates adding text objects to annotate a graph
- Demonstrates how the cursors in two plots can be lir Moving one, moves the other. Also in this example s the active trace in one plot does the same in the othe
- Uses the 'Xstring' and 'Ystring' parameters display alternate units.
- Shows how to close both figures when either plot is using the 'Link' parameter.
- Shows how to use the 'pos' parameter to position figures as far apart as possible given the available sc
- The 'HelpText' parameter is used to annotate the airspeed chart with the equations that are used to ger plotted data.



This example demonstrates:

- showing the line characteristics in the TraceID using TraceMK parameter
- setting the cursor callback with the **moveCB** parame
- setting axis, TraceID box, and MenuBox positons us parameter
- setting trace characteristics with the Linewidth, S and Markers parameters
- setting an initial cursor position
- enabling the multiCursor mode
- modifying the colors and fonts of the Trace IDs.
- The use of the slider pseudo object
- The use of the plt 'HelpText' parameter to displa temporary help information at the top of the plot wir help text disappears when any parameter is changed re-enabled by clicking on the help button or by right on the help tag in the MenuBox.
- Shows how to use **inf** in the **'Pos'** parameter to p the figure in the upper right corner of the screen. In t example an extra 48 pixels is allocated to the title ba the menu bar and one toolbar can be enabled withou the title bar off the top of the screen.
- The clipboard button captures the figure as a bitmap clipboard
- Using zeros(6) in the plt call to define 6 traces. T callback will overwrite these zeros with the actual dark displayed. Note that nan(6) would also have work as well for this purpose.

This script shows another example of putting more than c a single figure. The SubPlot argument is used to create th The lower axis contains four traces showing the magnituc (decibels) of four different weighting functions used in sc meters (as defined by IEC 651). The middle axis shows th four traces except using linear units instead of dB as used lower axis. The top axis shows the inverse of the linear m



weight.m

traces, which isn't particularly useful except that I wanted demonstrate plotting three axes in a single figure.

- Normally plt only puts one trace on each subplot exc the main (lower) axis. So in this case (with 12 traces 10 traces on the lower axis and one on the other two really want 4, 4, and 4, the 'SubTrace' parameter partition the traces between the axes as desired.
- When using the SubTrace parameter the native plt culobjects will not behave consistently, so normally the will be disabled. Alternatively the program can modicursor behavior to make it consistent with the particul SubTrace settings and this is the approach used in time example. The 'moveCB' cursor callback runs the C function which keeps the cursors on all three axes synchronized so that the cursors in the upper two axe automatically move to the same trace and the same x of the cursor in the main (lower) plot.
- The traceID callback ('TIDcback') insures that t box controls the the visibility of the traces in all thre
- Note the 'LineWidth' argument in the plt call. T illustrates how any line property may be included in calling sequence.

This example has been largely superseded by the followir example (wfalltst.m) which uses the general purpose pltwater 3D plotting routine. That's a far easier way to waterfall plot, although this example doesn't do that since written before pltwater was created. However this example included since it may still be a good starting point if you ' develop a special purpose waterfall display that can't be c using pltwater.

- Demonstrates how to do hidden line removal which waterfall plot much easier to interpret.
- Type wfall or wfall(0) to start wfall in its stustate. (i.e. the display is not updating)
- Type wfall(1) or wfall('run') to start wf

wfall.m



the display dynamically updating.

- One trace color (green) is used for all 30 traces ('Tr parameter)
- The 'TraceID' parameter is set to empty to disab TraceID box.
- The figure user data is used to pass the handle struct the callback.
- Extensive use of the slider pseudo object to control t data.
- The 'Linesmoothing' option is selected (which surprisingly speeds up the display dramatically on m systems)
- A pseudo popup in "super-button" mode is used to st stop the display.
- The number of display updates per second is calcula second with the results shown in a large font below t

This program demonstrates the use of pltwater, a gene purpose 3D plotting utility.

A surface consisting of a sequence of sync functions is cr 800 x 200 array (Z) which is then passed to pltwater.

We could have called pltwater with just a single argument containing the data, but in this example we have included additional parameters to tailor the display, including:

# 

- nT
- skip
- X
- y

all of which are described in the pltwater section of the he well as in the comments in pltwater.m. The remaining parameters included in the pltwater command in this exar not unique to pltwater, so they are passed directly to plt al described in the main plt programming section of the helf Those parameters include:

- HelpText
- TraceC
- CursorC
- Title
- ^Fontsize
- LabelY
- xy

Struggling with Matlab's FFT window display tool (wintc found it cumbersome and limited. I wanted a way to quicl change window parameters and see the effect on the time frequency shapes and the most common window measure (scalloping and processing loss, frequency resolution, and equivalent noise bandwidth). I couldn't modify wintool fc since most of the code was hidden (pcode). So I wrote wi create a more useable gui for displaying windows. winplt traces showing the time and frequency domain shapes of different FFT windows and also is a tool for designing yo windows by adjusting the kernel coefficients with a slider also use winplt's command line interface to return the win shapes for use in your Matlab programs.

While working with this application, you may find the IE on Windows for Harmonic Analysis (by Harris) useful. T most cited reference on FFT windows and includes descri most of the windows plotted by winplt. For your convenie can get this paper from my website (<u>www.mennen.org</u>) in section called "Signal processing papers".

#### winplt.m



Most treatments of FFT windows are highly mathematica the Harris paper). But if you want to understand some of 1 ideas without the many pages of mind numbing equations look at the this portion of a signal processing talk I gave r years ago. The file is called *windowsTalk.pdf* and you can my web site, right next to the Harris paper mentioned abo

winplt was designed primarily for its signal processing ed value but it is also a good demonstration of the use of plt's objects and these gui programming techniques:

- Demonstrates how to provide application specific he menu box tag (HelpW in this example) using the we browser to open an html document as well as by ope specific topic inside a windows compiled help file (... format).
- Demonstrates a novel use of the pseudo popup objec a vector from a gui. (See ID30 adjust kernel)
- Shows the power of the prin.m function [creation of window parameter block].
- Demonstrates how to add an application version strin right corner of the figure)

For a complete description of the winplt application, its n its command line interface, and its graphical interface, cl



# **Trace properties**

Right

You specify which traces should appear on the right-hand axis with the 'Right' parameter. For example if you included 'Right', [1 4:2:10 17] in the parameter list, then plt would put trace numbers 1,4,6,8,10, and 17 on the right axis and all other traces on the left axis. A slight shading is used behind the Trace IDs associated with the right hand axis so you can tell at a glance which traces belong to which axis. (You can disable this shading if you prefer. To see how, read the description of the TraceID parameter below). You can also tell which axis a trace is on by the shape of its cursor ('+' for left axis and 'o' for the right axis). You can optionally specify a label for the right hand axis (see LabelY) as well as the axis limits (see YlimR). Specifying an empty list, as in 'Right', [] tells plt to use the left axis for all the traces (the same as if you omitted the Right parameter altogether.)

The Markers parameter is a shorthand way of setting a different marker property for each line. For example:

```
plt(x,y,'Markers',s)
```

is equivalent to:

```
a = plt(x,y);
for k=1:length(a)
set(a(k),'Marker',s(k,:)); end;
```

The argument may be an array of characters or a cell array of strings. The latter method is easier when the elements are different sizes because you don't have to pad with blanks as with the character array. (Wherever a character array is allowed in a plt argument list, a cell array of strings is also allowed and visa versa.) For example, these two lines have give the same result:

#### Markers

```
plt(...,'Markers',['square';'+ ';'none
']);
plt(...,'Markers',{'square','+','none'});
```

This sets the marker for the first two lines to a square and a plus sign respectively while the third line will be rendered without any markers.

The following example shows two ways to set the markers of the six traces to x,+,square,o,asterisk,x (respectively). The shorter method used in the 2nd line is possible because every marker may be represented with a single character:

```
plt(...,'Markers',
['x';'+';'s';'o','*','x']);
plt(...,'Markers','x+so*x');
```

The Styles parameter is a shorthand way of setting the LineStyle property in a similar way that the Markers parameter is used to set the Marker property. For example, to set the first trace to normal, the 2nd and 3rd traces to dotted and dashed respectively, and the 4th trace to none (useful when you want the markers with no lines connecting them) you would use the following command:

```
plt(...,'Styles',{'-',':','--','none'});
```

The shorthand for single character styles mentioned above also works. For instance, to alternate between normal and dotted among eight traces one could use:

## plt(...,'Styles','-:-:-:');

One additional trick applies to the Styles parameter. If a single character is given which is not a valid line style, then the linestyle property is set to none and the given character is applied to the marker property. As an example, the following command defines eight traces of which the first four are rendered as continuous lines (i.e. without markers) and the last four are rendered with plus sign markers placed at each x,y location specified by the data arrays but with no lines connecting the markers:

### plt(...,'Styles','---++++');

Since there are no marker property values which can also be linestyle property values, there is never any ambiguity as to which property should be set.

This parameter allows you to select the grid line style. For example:

GridStyle

**Styles** 

#### plt(...,'GridStyle',':');

will select a dotted or dashed line (depending on the graphics renderer). If this parameter is not included the default is usually a solid line ('-') although there is one somewhat complicated exception to this which is described in the default section of the GRIDC parameter which you can find <u>here</u>.

This parameter allows you to assign a name to each trace. This name will appear in the trace selection box (also sometimes called the TraceID box). The number of characters that will fit in the trace selection box depends on the size you choose for the plt window. For the default figure size there is room for about 5 uppercase or 6 lowercase characters. In the example below, both forms are equivalent:

```
plt(...,'TraceID',['Rtemp';'Ltemp';'RV1
']);
plt(...,'TraceID',
{'Rtemp';'Ltemp';'RV1'});
```

```
Default: ['Line 1';'Line 2'; ... 'Line n'];
```

If you want the plot to be created without a TraceID box, call plt with a TraceID parameter of zero or the empty set ([] or "). Since plt can't create a TraceID box containing more than 99 IDs, if you want to plot more than 99 traces, you must include 'TraceID', 0 (or with the equivalent empty set value) in the parameter list.

When specifying traceIDs, you must have one trace ID for every trace on the main and right hand axes. However if you don't want a trace ID for a specific trace to appear, just use the null string ('') for the trace name. If you do that, the trace ID box will be made smaller to account for the fewer number of IDs displayed.

Normally traceIDs associated with the right hand axis will appear in the traceID box with a slight shading so you can identify those traces at a glance. If you want to disable this shading, insert the special character ']' at the beginning of the first TraceID name. The right bracket will be removed from the trace name before it is used. The third plot of the pub.m demo program demonstrates the use of this special character.

You may specify a callback function (fcn) to execute when the user clicks on any of the TraceID tags by including the parameter 'TIDcback', fcn in the argument list. If the string '@TID' occurs anywhere inside the function string then it's replaced with the handle of the trace ID string. Likewise if the string '@LINE' occurs anywhere inside fcn, it is replaced with the handle of the trace itself and occurrences of '@IDX' are replaced with the index of the selected trace. (i.e. 2 for the second trace listed in the TraceID box). See the

#### **TraceID**

demo program pltquiv.m for an example using the TIDcback parameter. In that example, the name and color of a trace is displayed in the command window when you click on a Trace ID tag. (Not particularly useful, but this example was contrived to demonstrate all the possible substitutions.) To define a quote within a quote in Matlab, one uses two single quote characters in a row. Since this can get confusing at times, callbacks defined within plt may use a double quote character instead of two successive single quotes. The pltquiv.m example uses this alternative form. In addition to a string, fcn may also be a function handle of the form @func or {@func, arg1, arg2, ..., argn}. Note that the string substitutions can't be used with the function handle form of this parameter.

You also may change the traceIDs after the plot has been created. For example, if the current figure contains a plot with four traces, these traces can be renamed with a command such as:

```
plt('rename',
{'First' 'Second' '3rd' '4th'});
```

If there are other changes you want to make to the TraceID box from your program (as in the **CURVES.m** example), you can get the handle of the axis that contains all the TraceID objects with the following command:

```
tbox = findobj(gcf, 'user', 'TraceID');
```

Then, for example the following command would make the TraceID box invisible:

```
set([tbox; get(tbox, 'child')], 'vis', 'off')
```

An easier way to make the TraceID box invisible would be to simply move it outside the figure area:

```
set(tbox, 'pos', [-2 0 1 1]).
```

Or in the unlikely event you wanted to reverse the order of the TraceIDs (i.e. bottom to top ordering in the TraceID box), use the command:

```
set(tbox, 'view', [0 270]).
```

TraceMK

This parameter allows you to show the line types in the trace selection box to help identify the traces. This can be visually pleasing and is especially helpful if you are color blind. If the argument is a vector, it specifies the marker positions within the trace selection box. For example

'TraceMK', [.6.7.8.9] would tell plt to place a horizontal line next to each TraceID label beginning and ending at x = .6 and .9 with markers at the four locations specified (assuming the line type in the plot included markers). The area between x = 0 and .6 (i.e. the first 60%) would be used for the text label. If the first element of the vector is less than .25 then plt will not display the text labels since there probably would not be room for them anyway. (Clicking on the lines in the TraceID box have the same effect as clicking on the labels, so the labels can be removed without loss of functionality). If the argument is a scalar, plt will use that value as the first element of a length 3 vector whose last element is .9. Thus 'TraceMK', .6 is shorthand for 'TraceMK', [.6.75.9]. A special case is when the scalar argument is zero, in which case no lines are inserted into the trace selection box (as if the TraceMK parameter was not used at all). See the demo programs trigplt.m and subplt.m for examples of using the TraceMK parameter.

All TraceIDs will appear in the trace selection box (aka TraceID box) in a single column except when the TIDcolumn parameter is included. This is useful when you are using so many traces that the TraceID box becomes too crowded to fit all the trace names in a single column. The simplest way to use the TIDcolumn parameter is to supply an empty argument TIDcolumn

to the parameter (i.e. ' ' or []). When this is done plt will use just a single column for the TraceID box when the number of traces is 24 or less. Two columns will be used when the number of traces is between 25 and 48, and three columns will be used when there are more than 48 traces. (The TraceID box will not appear when more than 99 traces have been defined). This default will probably work in nearly all situations but if you want exact control over how many columns are used and how many traceIDs appear in each column, you can do that by specifying a non-empty argument to the TIDcolumn parameter as follows: If **TIDCOlumn** is a scalar, it specifies the number of TraceIDs to put in the *second* column. If it is a vector, it specifies the number of TraceIDs to put in columns 2,3,etc, with the remaining going into column 1. For example, if 30 traces are displayed, and you use 'TIDcolumn', 8 then the first 22 TraceIDs appear in the first column and the last 8 appear in the second column. 'TIDcolumn', [5 5 5] would tell plt to arrange the 30 IDs in four columns as follows: (1-15, 16-20, 21-25, 26-30).

By default, all the traces defined by plt are visible until you change that from the trace selection box. You can change the default by disabling some traces from the plt call. For example:

```
plt(..., 'DIStrace', [1 1 0 0 0]);
```

## DIStrace

This tells plt to start the display with the first two traces disabled and the remaining 3 traces enabled. Of course you can later enable the first two traces via the trace selection box. If the parameter has fewer elements than the number of traces, it is extended by adding zeros. This means that we could have used [1 1] above to the same effect. After the call to plt has been made, if you want to change which traces are enabled/disabled you can click on the TraceIDs as described in Selecting traces. However if you want to do that from a program you can use the plt ( 'show', ...) command which is described at the very bottom of the

By default you will be allowed to cursor every visible trace in the plot area. You can change this default using this parameter. For example, if we had five traces, but wanted to use cursors only on traces 1,4, and 5 you would use: **ENAcur DIT**(..., 'ENACUR', [1 0 0 1 1]);
If the parameter has fewer elements than the number of traces, it is extended by adding ones. This means that we could have

used 'ENAcur', [1 0 0] above to the same effect.

This parameter allows you to reserve space for additional traces to be added to the figure after the plt window has been started. For example plt(x1,y1,x2,y2,'+',5); opens the plt window with two traces, the first one defined by x1,y1 and the second one by x2,y2. Then room is reserved in the TraceID box for up to 5 more traces that can be added using the pltt.m function. This parameter is normally only used inside script or function files because when you type the plt command in the Matlab command window an automatic '+', 8 is assumed. You could still include the + parameter from the command window in the unlikely event you were planning on adding more than 8 traces. When plt is called from a script or function, you can't add traces after the plt window has opened unless you had included the + parameter in the argument list.

It is unusual to want to add dozens of traces with the pltt function, but it is possible. For example with the command

plt(x,y, '+', 39, 'TIDcolumn', ''); plt will reserve space in the TraceID box for 40 traces. The first is specified in the plt command and the remaining 39 can be added using the pltt function. The TIDcolumn parameter was needed in this case because without it, plt would attempt to cram all 40 TraceIDs into one column which would probably be unreadable.

You may include the **TraceID** parameter in the argument list as well if you like, and you should be aware that there are two ways of doing this. The first (and by far the most common) way of doing this is to put the 'TraceID' parameter before the **'+'** in the argument list. When done in that order, that TraceID argument specifies the trace names only for the traces defined in the argument list. Then when the '+' parameter is encountered, plt expands the TraceID list using default names that will usually be overwritten by the trace names included in the calls to **pltt**. When done in the opposite order, the TraceID argument should include the trace names you want for the traces that will be added later (even though the trace names will be invisible until those traces are added). And if the TraceID argument does not include enough trace names for this, when a trace is added after the list has been exhausted, the new trace will be added without any corresponding entry in the TraceID box (which occasionally might even be what you wanted). Typically the + parameter is placed after all the traces defined inside the plt argument list, however this is not strictly necessary. In fact multiple + parameters may be included and they may be interspersed with the trace definitions in the parameter list. When you do that, the space reserved in the TraceID box for the traces to be added later will be interspersed with the defined traces in the order in which they appeared. This flexibility is rarely needed, but nevertheless it is available if you want it. Note that when traces are added with the pltt function, the reserved slots are used in order (top to bottom, as well as left to right if multiple columns were enabled).

You might expect that when all the free slots in the TraceID box have been used up, you can no longer add a new trace with the pltt function ... but in fact you can. What happens is that in this situation, pltt will overwrite the data and the trace name of the last trace that was added, so effectively you can never run out of free slots (unless you never allocated any in the first place).