CBX Matlab Plotting Scripts User Guide

Christopher Batten
November 28, 2008

The CBX Matlab Plotting Scripts are a collection of matlab functions which simplify formatting and exporting matlab plots for use in LaTeX documents. This user guide is divided into two parts: the first part contains a few simple tutorials, and the second part contains general user reference.

Table of Contents

1. Tutorials

This part includes some simple tutorials to illustrate the common usage for the CBX Matlab Plotting Scripts. You can follow through these tutorials yourself by typing in the commands marked with a > symbol at the matlab prompt. Tasks to try on your own are denoted with (*).

The first thing we need to do is generate some data for plotting. We will be plotting a sine function with different amplitudes and phases. Enter the following commands and take a look at the default matlab plot formatting before continuing.

```
> x = [-2*pi:0.25:2*pi];
> y = sin(x);
> figure;
> H = plot( x, y, x, 2.*y, x, 1+y, 1+x, y );
> set(gca,'xlim',[-3 3]);
```

1.1. Formatting a Simple Plot

We will begin by first resizing the figure. It's important if we are going to fully script plot generation that we set the size of the figure explicitly so that the generated plots always look the same. Otherwise, our plots might be sized based on the current (arbitrary) figure window size. We can use the `cbxplot_format_fig` function to set the width and height of the figure in pixels. The relative width versus height sets the aspect ratio which can be

important for correctly filling the space available in the LaTeX document. Since the fonts and line widths are set in absolute point units, the absolute dimensions of the figure will impact how large these objects appear in the final figure. We set the figure size to be 600x300 pixels and replot the sine data.

```
> figure;
> cbxplot_format_fig( 600, 300 );
> H = plot( x, y, x, 2.*y, x, 1+y, 1+x, y );
> set(gca,'xlim',[-3 3]);
```

Next we add some formatting to each of the lines with the `cbxplot_format_line` function. This function takes a handle to a line object and various formatting parameters.

```
> cbxplot_format_line( H(1), 4, '-',  [0,1,0]               );
> cbxplot_format_line( H(2), 2, '-.', 'red',        'oF', 8 );
> cbxplot_format_line( H(3), 2, '-',  'dark-orange','sF', 8 );
> cbxplot_format_line( H(4), 2, ':',  'dark-blue',  's',  8 );
```

The second parameter is the line width in points and the third parameter is a standard matlab line spec indicating if the line should be solid, dashed, etc. The fourth parameter is a color and it can either be specified as a three element RGB vector (eg. [0,1,0]), as a predefined matlab color string (eg. `red`), or as a predefined cbxplot color string (eg. `dark-orange`). See the reference section for more information on predefined colors. The final two parameters are optional and they specify the marker symbol and size for the line. The possible marker symbols are the same as the standard matlab symbols except that any symbol can include an optional `F` suffix which indicates that the marker should be filled in with the line color. The default is to fill the marker with white. The marker size is in points.

Now that we have formatted the lines, we can move on to formatting the fonts with the `cbxplot_format_fonts` function.

```
> cbxplot_format_fonts( 'times', 18 );
```

The function takes two parameters specifying the font name (eg. `times`) and the font size in points. To see the available fonts on your system use the `listfonts` command, although the `helvetica`, `times`, and `courier` fonts should always be available. Note that the font formatting applies to the current plot's title, axis labels, tick labels, and legend.

The final formatting we might want to do is to set the margins for the plot with the `cbxplot_format_margins` function. If we are going to use this plot in a LaTeX document, then it is convenient to set the outer margins to zero so that we can control the spacing directly within in LaTeX. The following command sets the outer margins to zero.

```
> cbxplot_format_margins( [ 0, 0, 0, 0 ] );
```

In this example, the outer margin is specified as a four element vector containing the top,

right, bottom, and left margins respectively. Each margin is represented as a fraction of the figure height and width. So the margin must range between 0 and 1. You can also specify the outer margin with a single number indicating that all four margins should be the same.

The final step is to export the figure to a PDF with the `cbxplot_export_pdf` function.

```
> cbxplot_export_pdf('cbxplot-tut-1');
```

The parameter is the name for the PDF file. The function will automatically add the `.pdf` extension. The function will take care of setting the PDF page parameters so that the PDF looks similar to how the figure appears on the screen. Open the PDF and see how it looks. (*) Try experimenting with additional formatting and compare the screen figures with the exported PDF files.

1.2. Formatting a Plot with Multiple Subplots

In this section, we will create a figure with four subplots arranged in two rows and two columns. As before the first step is to the format the figure size and then to plot the data. We use the `cbxplot_subplot` function to first specify how many rows and columns of subplots we want, and then to specify which subplot we are currently plotting.

```
> figure;
> cbxplot_format_fig( 400, 400 );
> cbxplot_subplot( 2, 2 );
>
> cbxplot_subplot(1);
> H = plot( x, y, x, 2.*y, x, 1+y, 1+x, y );
> set(gca,'xlim',[-3 3]);
> title('Example Title');
>
> cbxplot_subplot(2);
> H = plot( x, y, x, 2.*y, x, 1+y, 1+x, y );
> set(gca,'xlim',[-3 3]);
> title('Example Title');
>
> cbxplot_subplot(3);
> H = plot( x, y, x, 2.*y, x, 1+y, 1+x, y );
> set(gca,'xlim',[-3 3]);
> title('Example Title');
>
> cbxplot_subplot(4);
> H = plot( x, y, x, 2.*y, x, 1+y, 1+x, y );
> set(gca,'xlim',[-3 3]);
> title('Example Title');
```

The `cbxplot_subplot` function is very similar to the built-in matlab `subplot` function, except that you only need to specify the number of rows and columns once. This makes it easier to add more subplots or change how they are organized later.

Now that we have four subplots, we might want to change the inner margins between them. We can do this with an optional second argument to the `cbxplot_format_margins` function.

```
> cbxplot_format_margins( 0, [ 0.01, 0.1 ] );
```

In this example, the outer margin is specified as a two element vector with the first element denoting the inner margins in the horizontal direction (ie. between columns) and the second element denoting the inner margins in the vertical direction (ie. between rows). As with the outer margin, each margin is represented as a fraction of the corresponding figure dimension. You can also specify the inner margin with a single number indicating that both margins should be the same.

Often when we are generating large grids of small subplots, the x-axes are the same in any column and the y-axes are the same in any row. We can use the `cbxplot_delete_inner_labels` function to remove the inner axis labels, tick labels, and titles to create more compact figures. Note that you should usually delete the inner labels *before* setting the margins.

```
> cbxplot_delete_inner_labels();
> cbxplot_format_margins( 0, 0.01 );
```

Notice that we keep the titles for the top row of subplots and the y-axis labels for the first column of subplots. We can therefore use these to label the rows and columns. (*) Use the formatting functions to change the line and font format for each subplot. Then export the figure to a PDF and examine the resulting file. Also try changing the number of subplots and the size of the inner margins.

1.3. Adding Annotations to Plots

Once we have formatted our plots, we might want to add some annotations to those plots. We can add lines and text with the `cbxplot_add_line` and the `cbxplot_add_text` functions. The following example adds two horizontal lines at +/-1 and labels them `max` and `min`.

```
> figure;
> cbxplot_format_fig( 600, 300 );
> H = plot( x, y, x, 2.*y, x, 1+y, 1+x, y );
> set(gca,'xlim',[-3 3]);
> cbxplot_add_line( [-3,-1], [3,-1], 2, '--', 'dark-red' );
> cbxplot_add_line( [-3,1],  [3,1],  2, '--', 'dark-red' );
> cbxplot_add_text( [-2.9,1.1], 'max', 'times', 14, 'dark-red', 'tr' );
> cbxplot_add_text( [2.9,-1.1], 'min', 'times', 14, 'dark-red', 'bl' );
```

Note that the annotation location is specified in data coordinates. The final parameter for `cbxplot_add_text` specifies where the text should be positioned relative to the location point. In this example we specify that the `max` text should be positioned to the top-right of the given point, while the `min` text should be positioned to the bottom-left of the given point. See the reference section for more information on valid positions. (*) Try drawing additional annotations possibly on a figure with multiple subplots.

2. Reference

In this section we provide more detailed reference on the CBX Matlab Plotting Scripts. We first describe some of the common formatting parameters used across many different functions, before given per function documentation.

2.1. Common Formatting Parameters

This section describes the common color, font, and marker formatting parameters which are used across various different functions.

Color Formatting

Normally, matlab functions which accept a color can handle either a three element RGB vector or a predefined color string. The predefined matlab color strings are:

- `yellow` (`y`) : [1 1 0]
- `magenta` (`m`) : [1 0 1]
- `cyan` (`c`) : [0 1 1]
- `red` (`r`) : [1 0 0]
- `green` (`g`) : [0 1 0]
- `blue` (`b`) : [0 0 1]
- `white` (`w`) : [1 1 1]
- `black` (`k`) : [0 0 0]

The `cbxplot_colors` function allows us to use the following predefined cbxplot colors:

- `dark-red` (`dr`) : [0.75 0.00 0.00]
- `dark-blue` (`db`) : [0.00 0.00 0.75]
- `dark-green` (`dg`) : [0.00 0.75 0.00]
- `dark-orange` (`do`) : [0.75 0.20 0.00]

Font Formatting

A font is specified with the font name and a font size. The font size is always in points. To see a list of font families you can use the `listfonts` command. Standard fonts which should probably always be available include `times`, `helvetica`, and `courier`.

Marker Formatting

A marker is specified with a marker symbol string and a marker size. The marker size is always in points. The following are the built-in marker symbols:

- `+` : plus sign
- `o` : circle

- `*` : asterisk
- `.` : point
- `x` : cross
- `s` : square
- `d` : diamond
- `^` : upward-pointing triangle
- `v` : downward-pointing triangle
- `>` : right-pointing triangle
- `<` : left-pointing triangle
- `p` : five-pointed star
- `h` : six-pointed star

By default the marker face color is white. For `cbxplot` functions, a user can specify that the marker face color should be the same as the line color by using a `F` suffix at the end of marker symbol. For example, the marker symbol string `o` indicates circles filled with white, while the marker symbol string `oF` indicates solid circles. The marker edge color is always the same as the line color.

2.2. Function Reference

This section contains an alphabetical listing of all the functions included in the CBX Matlab Plotting Scripts.

cbxplot_add_line

```
USAGE:
 cbxplot_add_line( pt1, pt2 )
 cbxplot_add_line( pt1, pt2, width, style, color )
```

Draw a line on the current plot. The points are specified in data coordinates. The width is in points. See Section 2.1 for more information on the style and color marker parameters.

cbxplot_add_text

```
USAGE:
 cbxplot_add_text( pt, str )
 cbxplot_add_text( pt, str, font, size )
 cbxplot_add_text( pt, str, font, size, color )
 cbxplot_add_text( pt, str, font, size, color, pos )
 cbxplot_add_text( pt, str, font, size, color, pos, rotation )
```

Draw the given text string (str) on the current plot. The point is specified in data coordinates. See Section 2.1 for more information on the font and color parameters. The position (pos) parameter specifies where the text should be positioned relative to the given data point.

- `tl` : top left
- `cl` : center left
- `bl` : bottom left
- `tc` : top center
- `cc` : center center
- `bc` : bottom center
- `tr` : top right
- `cr` : center right
- `br` : bottom right

For example, using `tl` will position the text above and to the left of the point while `br` will position the text below and to the right of the point. The rotation parameter is the number of degrees to rotate the text counter-clockwise around the given point.

cbxplot_colors

```
USAGE:
 C = cbxplot_colors( [R,G,B] )
 C = cbxplot_colors( 'color-string' )
```

See if the given color is one the predefined cbxplot color strings. If not then just return the color unchanged. See Section 2.1 for more information on color formatting.

cbxplot_data2fig_point

```
USAGE:
 fig_pt = cbxplot_data2fig_point( data_pt )
```

Convert the given data point (in data coordinates) into a figure point (in figure coordinates). Data coordinates refer to the data space while each figure coordinate ranges from 0.0 to 1.0 and is relative to the entire figure dimensions. Figure coordinates can be useful when using the arrow or textarrow versions of the `annotation` function, but remember that these annotations will not move with the axis. So you would need to position all of the axis first (possibly using `cbxplot_format_margins`) and then draw annotations.

cbxplot_delete_inner_labels

```
USAGE:
 cbxplot_delete_inner_labels()
```

Remove the inner labels from the subplots in the current plot. For the top row of subplots we remove just the x-axis labels, and for the bottom row of subplots we remove just the title, and for all other rows we remove both the x-axis labels and the title. Similarly, for all but the first column of subplots we remove the y-axis labels. This means you can use the plot titles to label the columns and the y-axis labels to label the rows. Note that for the best results delete the inner labels before formatting the margins.

cbxplot_export_pdf

```
USAGE:
 cbxplot_export_pdf( filename )
```

Export the current figure to a PDF file. Usually, exporting a figure to PDF changes the aspect ratio and size of the figure. What we really want is for the pdf to look like the figure on the screen. So this function first determines the size of the screen in pixels and then inches to determine the effective dots per inch. It then changes the paper size to be the same as the figure on the screen converted into inches. It also makes sure the figure fills the page with no extra margins.

cbxplot_format_fig

```
USAGE:
 cbxplot_format_fig( width_px, height_px )
```

Resize the current plot to the given width and height specified in pixels. Since the final output will be a vector pdf, the ration of width to height is the most important since it will set the aspect ratio of the final plot. The absolute numbers are somewhat important, though, since the font size and line width are specified in pixels.

cbxplot_format_fonts

```
USAGE:
 cbxplot_format_fonts( name, size )
```

Set the font family and size for all the text (title, axis labels, tick label, and legend) in the current figure. See Section 2.1 for more information on font formatting.

cbxplot_format_margins

```
USAGE:
 cbxplot_format_margins( outer_margin )
 cbxplot_format_margins( [ top, right, bottom, left ] )
 cbxplot_format_margins( ..., inner_margin )
 cbxplot_format_margins( ..., [ horizontal, vertical ] )
```

Change the outer margins for all plots and optionally change the inner margins for plots with subplots. The outer margin can be specified either with a single number indicating that all four outer margins should be the same, or as a four element vector specifying the top, right, bottom, and left outer margins separately. The inner margin can be specified either as a single number indicating that the horizontal and vertical inner margins should be the same, or as a two element vector specifying the horizontal and vertical inner margins separately.

Outer margins are defined to be the space between the edge of the figure and the text which is closest to that edge. Similarly, the inner margin is the space between subplots including

labels and such. So for a vertical inner margin between two rows of subplots, this might be the space between the x-axis label for upper row and the title for the lower row.

cbxplot_format_line

```
USAGE:
 cbxplot_format_line( handle, width, spec, color )
 cbxplot_format_line( handle, width, spec, color, mstyle, msize )
 cbxplot_format_line( handle, width, spec, color, mstyle, msize, zdepth )
```

Set the format of the line for the given handle. The format is made of many components listed below:

- `width` : line width in points
- `style` : line style string
- `color` : line color (RGB or matlab/cbxplot predefined color)
- `mstyle` : marker char with optional `F` suffix
- `msize` : size of marker in points
- `zdepth` : the depth ordering of the lines

The width is in points. See Section 2.1 for more information on the style, color, and marker parameters. The final zdepth parameter can be used to control the order in which the lines are drawn. So if a user wants one line to be on top they should make sure it has a higher zdepth value.

cbxplot_split_subplots

```
USAGE:
 H = cbxplot_split_subplots()
```

Splits each subplot into its own separate figure. Using subplots is nice because it correctly aligns the plots based on their position boxes, but then it becomes difficult to save the subplots as separate PDF files. It can be useful to have one PDF file per subplot if we want to reference them separately in a LaTeX file using the `subfig` package for example. So this function will split each subplot into its own figure and it will preserve the spacing so that they line up just like in the single monolithic figure. Margins are removed, however, since we probably want to manage those explicitly when we reassemble the figure. The function returns a vector of handles to the new figures so that we can loop over them for further processing (probably using `subplot` to make the figure current and then calling `cbxplot_export_pdf`).

cbxplot_subplot

```
USAGE:
 cbxplot_subplot( num_rows, num_cols )
 cbxplot_subplot( id )
```

Manage subplots in a slightly simpler way than the built-in matlab subplot function. The matlab subplot function requires that you always specify the total number of subplot rows and columns, which means it is cumbersome to change the number of rows or columns. To use the `cbxplot_subplot` function you call it once to specify the number of rows and columns, and then you just need to specify the id number for the subplots. This means you only to change one line to change the number of rows or columns. Essentially, this function includes global state to save the number of row and columns, and then uses this information to call the built-in subplot function.