

## *EGR 423 W'01*

### *Scilab PLOT2D Tutorial*

Scilab's plotting facilities are hard to use. This tutorial is intended to guide your way through the `plot2d` range of functions. Note that an easier-to-use alternative exists in the form of a third-party contributed package known as `plotlib`. It must be installed as an add-on to Scilab. You can get `plotlib` from:

`http://www.dma.utc.fr/~mottelet/myplot.html`

The goal behind `plotlib` is to present a MATLAB-like interface to plotting for Scilab. This is, in my opinion, an easier-to-use interface to the same functionality. However, it is still important to know of Scilab's basic plotting functionality. If you do use `plotlib` and it doesn't behave the way you want it to, you will need to revert to Scilab's functions.

#### **1.0 PLOT**

The simplest Scilab function for plotting a vector of data is `plot` since you can simply say:

`plot(y)`

and the data in the vector `y` is plotted on the Y-axis against the numbers 1 through N (where `N=length(y)`) on the X-axis.

For control over the numbers on the X-axis you can write:

`plot(x,y)`

and now the vector `y` is plotted against the vector `x`. Finally, you can add a caption for both the X-axis, Y-axis, and the title of the plot:

`plot(x, y, 'x axis label', 'y axis label', 'title of plot')`

#### **2.0 PLOT2D**

The `plot2d` function must be used when `plot` is insufficient. That is, `plot2d` provides the following additional options for controlling the appearance of a plot:

- Multiple vectors may be plotted at the same time
- Line styles and colors may be specified
- A legend may be displayed
- The axes may have their bounds set manually
- Whether or not axes are drawn and how tick-marks appear on the axes

## 2.1 Plotting multiple vectors

Multiple sets of (x,y) vectors may be plotted by putting all X-axis vectors as *column vectors* into one matrix X and all Y-axis vectors as *column vectors* into another matrix Y. For example:

```
x = (0:0.1:10)';           // X axis for both curves. Note that it
                           // MUST BE A COLUMN VECTOR, hence the '
y1 = sin(x);              // First vector to plot
y2 = cos(x);              // Second vector to plot
plot2d([x, x], [y1, y2]);
```

The first parameter to `plot2d` is an NxP matrix comprising the set of X-axis column vectors (which will most likely all be the same thing), where N is the number of points to plot and P is the number of vectors that you will be plotting at the same time.

The second parameter to `plot2d` is also an NxP matrix comprising the set of Y-axis column vectors. The most common error with the `plot2d` family of functions is forgetting the required column vector format.

Note the following:

- The range construct `0:0.1:10` (for example) constructs *row vectors*. This must be transposed to form a column vector (e.g., `(0:0.1:10)'`).
- Scalar functions that can operate over vectors (e.g., `sin()`, `cos()`, etc.) return a result that is the same shape as their argument. So `sin(x)` is a row vector if `x` is a row vector and `sin(x)` is a column vector if `x` is a column vector.
- Column vectors are pasted into a single matrix using commas:  
`[column1, column2, column3]`

## 2.2 Line styles and colors

The optional third parameter to `plot2d` is a 1xP *row vector* describing how each one of the P curves should be plotted:

- Values less than or equal to 0 draw unconnected points rather than a connected curve. The types of points to draw (known as *marker style*) is given by the number:
  - 0: '.'
  - 1: '+'
  - 2: 'X'
  - 3: '\*'
  - 4: Filled diamonds
  - 5: Unfilled ovals
  - 6: '⊗'
  - 7: '∇'
  - 8: Clubs (known as *trifle* in the documentation)
  - 9: 'o'

Marker styles below -9 are drawn as style -9.

- Values greater than 0 indicate a line color (1: black, 2: blue, 3: green, etc.)

For example, let us plot the sin and cos curves from a previous example using different colors:

```
x = (0:0.1:10)';           // X axis for both curves. Note that it
                           // MUST BE A COLUMN VECTOR, hence the '
y1 = sin(x);              // First vector to plot
y2 = cos(x);              // Second vector to plot
plot2d([x, x], [y1, y2], [2, 3]); // Blue & Green
```

To plot a curve with connected lines and also marking each point:

```
plot2d([x,x], [y1,y1], [-2,4]); // Cyan curve, 'X' markers
```

### 2.3 Legends

To describe each curve, a legend may be associated with the plot. This requires specifying two additional parameters to `plot2d`, and this is where things get ugly.

The first additional parameter (known as *strf* in the documentation) is a three-character string of digits (e.g., '021', the default if not specified). Each character controls a different aspect of the plot:

- The first character is 0 if no legend is to be displayed or 1 to display a legend
- The second character controls the boundaries of the axes (see below)
- The third character controls whether axes are drawn (see below)

To keep the defaults for axes but also display a legend, use '121' instead of the default '021'. Then, specify an additional string parameter where each curve's description in the legend is separated by '@':

```
plot2d([x,x], [y1,y2], [2,3], '121', 'sin@cos');
```

This will plot the same sin and cos curves as before in blue and green. A legend (known as a caption in the documentation) will also appear associating the text 'sin' with the blue curve and 'cos' with the green curve.

There is an alternative method of constructing a legend, which requires more calls to `plot2d` but is much easier to handle. When only curve is plotted, the third parameter (normally a 1xP vector of line styles) may optionally be a 1x2 vector describing the line style and legend position from 1 through 6. Multiple calls to `plot2d` can then be used to build up the legend:

```
plot2d(x, y1, [2, 1], '121', 'sin');           // Legend position 1
plot2d(x, y2, [3, 2], '121', 'cos');         // Legend position 2
plot2d(x, y3, [6, 3], '121', 'tan');         // Legend position 3
```

## 2.4 Axis boundaries

The *strf* parameter (the three-character string of digits) uses the last two characters to specify axis boundaries and display. The second character is most commonly set to one of the following:

- 0 : keep the axis boundaries the same as they are. This is useful when calling `plot2d` several times to superimpose plots.
- 1 : an additional argument to `plot2d` specifies axis boundaries
- 2 : auto-compute axis boundaries based upon data to be plotted

The third character is most commonly set to one of the following:

- 0 : do not draw anything around the actual data to be plotted
- 1 : draw axes (box, tick marks, values) around the data to be plotted
- 2 : draw a box around the data to be plotted but no tick marks or values

For example, plotting the sin or cos function with auto-scaling sets the Y-axis to be [-1,1] so the range of values completely fills the plot. To provide some blank space for improved visibility, we may want the Y-axis to range from [-1.25,1.25] even though the actual data only ranges from [-1,1]. We can set manual axis scaling as follows:

```
plot2d(x, y2, 1, '011', ' ', [x(1),-1.25,x($),1.25])
```

Note that we must still include a (blank) legend string (i.e., ' ') even if we are not plotting a legend. The axis boundaries is set by the 1x4 vector with the format:

```
[x_min, y_min, x_max, y_max]
```

The default *strf* argument is '021'. By setting it to '011' we indicated that we are manually specifying axis boundaries using the additional 1x4 vector argument shown above.

Let us plot both sin and cos vectors using different colors and with a legend on these new axes:

```
plot2d(x, y1, [2, 1], '111', 'sin', [x(1),-1.25,x($),1.25]);  
plot2d(x, y2, [3, 2], '101', 'cos');
```

Note that we have saved some typing by using the '101' *strf* argument: keep the same axes as before (since the middle character is 0 instead of 1).

## 2.5 Tick marks

Sometimes, the axis boundaries and values are such that they are difficult to read. You can control exactly how many tick marks occur on the axes with yet another optional parameter that follows the 1x4 axis boundary vector. This (last) parameter is known as *nax* in the documentation and is formatted as follows:

```
[x_minor_ticks, x_major_ticks, y_minor_ticks, y_major_ticks]
```

The `x_major_ticks` scalar indicates how many labelled tick marks (i.e., tick marks with numbers beside them) will be shown on the X axis (and `y_major_ticks` applies to the Y axis). The `x_minor_ticks` scalar (and `y_minor_ticks` scalar) indicates how many intervals will be in between major tick marks. Each of these minor tick marks has no value shown beside it.

For example, let's plot the sin and cos curves again over the range `[-1.25,1.25]` with fewer tick marks on the Y-axis but more numbered tick marks on the X-axis:

```
plot2d(x, y1, [2, 1], 'o11', ' ', [x(1),-1.25,x($),1.25], [2,20,1,5])
```

For the X-axis, we want 20 numbered tick marks. Since the X-axis range is `[0,10]` we will see the numbers 0.0, 0.5, 1.0, ... 9.5, 10.0. We have also indicated that there are two minor intervals in each major interval so we will see one non-numbered tick mark in between each numbered tick mark.

For the Y-axis, we want only 5 numbered tick marks, thus `-1.25, -0.75, -0.25, 0.25, 0.75, 1.25` are all that we will see on the Y-axis. By specifying only 1 minor interval for each major interval, there will be no non-numbered tick marks.

As before, we cannot specify the 1x4 tick marks vector without specifying all that comes before it (legend, axis boundaries).

### 3.0 PLOT2D1

The good news is that once you have become comfortable with the various options above, the remaining plot functions are quite simple. The `plot2d1` function gives additional control over linear vs. logarithmic axes with an additional new *first* parameter (called *str* in the documentation) which is a 3-character string:

- The first character should be 'g' (more on this later)
- The second character should be 'l' for a logarithmic x-axis or 'n' for a linear x-axis
- The third character should be 'l' for a logarithmic y-axis or 'n' for a linear y-axis

For example, let's plot logarithmic power against logarithmic frequency:

```
f = (0.1:10:10000)'; // Remember: COLUMN VECTOR
pow = 1 ./ (1 + 1 ./ (2*pi*f*100e-6).^2);
plot2d1('gll', f, pow);
```

and against linear frequency:

```
plot2d1('gnl', f, pow);
```

All of the remaining options described for `plot2d` (control over legend, line styles, axis boundaries, tick marks) apply in exactly the same way for `plot2d1`.

## 4.0 PLOT2D2

The `plot2d2` function has exactly the same parameter structure as `plot2d1` but plots stair-step waveforms (i.e., it assumes the functions to be plotted are piecewise constant). Try this:

```
plot2d2('ggn', (0:10)', (0:10)');
```

## 5.0 PLOT2D3

The `plot2d3` function has exactly the same parameter structure as `plot2d1` but plots isolated vertical bars. This is useful for drawing functions that you explicitly want to illustrate as discrete-time sequences. Try this:

```
x = (0:0.5:10)';
y = sin(x);
plot2d(x, y, -3);          // Plot '*' at each point
plot2d3('ggn', x, y);    // Plot stem of each point
```

## 6.0 PLOT2D4

The `plot2d4` function has exactly the same parameter structure as `plot2d1` but plots arrows. This plot style can be useful for plotting vector fields.

## 7.0 Miscellaneous

### 7.1 XSET

The `xset()` function can control various attributes of a plot. Type `xset()` for an interactive dialog. With this function you can control the fonts and font sizes of labels and numbers, foreground and background colors, the sizes of the various marks used, the format for the display of axis values, and lots more (type `'help xset'`). Try this:

```
xset('background', 1);      // Black background
xset('foreground', 8);      // White axes and labels
xset('font', 3, 10);        // Italics and bigger font size
xset('thickness', 2);       // Thicker lines
plot2d(0:10, 0:10, 2);     // Blue line
```

Note that some parameters set with `xset` persist from one plot to the next while others do not. You must enter a world of experimentation! Also note: the Symbol font is font number 1.

### 7.2 Overlays

Plotting multiple plots on the same axes is assumed by all the `plot2d` functions *but not* the simple `plot` function! The latter clears any existing plot prior to drawing the new plot.

You can manually clear the plot window by typing `xbasc()`.

### 7.3 Labels and Title

You can add a title and axis labels after calling a `plot2d` function by calling `xtitle`:

```
xtitle('Title', 'X Axis Label', 'Y Axis Label');
```

## 7.4 Annotations

You can add text and lines to a graph by using `xstring` for text and `xsegs` for line segments. For example, let's plot one period of a sine wave and add lines to show the X-axis and Y-axis:

```
x = (-%pi:0.1:%pi)';
y = sin(x);
plot2d(x, y, 2);
xsegs([0,0], [-1,1], 1); // Draw Y-axis in black
xsegs([-4,4], [0,0], 1); // Draw X-axis in black
xstring(-3,0.6,'Nifty!');
```

Note that the co-ordinates given are those relative to the plot axes, not some absolute screen or window co-ordinates. This makes it easy to figure out where to draw lines or text.

Connected line segments (i.e., a polygon) can be drawn with `xpolys`.

Rectangles can be drawn with `xrects`.

Single numbers can be placed on the plot with `xnumb`. This may be useful, for example, in labeling interesting points on the axes or indicating maximum or minimum values directly at the points at which they occur.

A grid may be added to the plot with `xgrid`.

Circles and ellipses may be drawn with `xarc`, `xarcs`, `xfarc`, etc.

## 7.5 Subplots

Sometimes it is most effective to draw several plots in the same window. This is especially true when preparing plots for publication. The `xsetech` function partitions a single graphics window into multiple plot areas, each with their own axes and properties. A great example is given in the documentation for `xsetech`:

```
xbascc()
xset("font",2,0)
xsetech([0,0,0.5,0.5]); plot3d()
xsetech([0.5,0,0.5,0.5]); plot2d()
xsetech([0.5,0.5,0.5,0.5]); grayplot()
xsetech([0,0.5,0.5,0.5]); histplot()
```

Note that simply typing `plot2d()`, for example, produces a demo.

## 7.6 Clearing and deleting windows

A window may be cleared with `xbascc()` or `xclear()`, with the difference being only in that the latter does not erase the commands used to date. This is only important when these commands are being recorded for later “playback” (i.e., don't worry about it...use `xbascc`).

A window may be deleted with `xdel()`. Both `xdel()` and `xbasc()` take an optional window ID allowing you to manage multiple graphic windows.

For example:

```
plot(1:10);  
x_dialog('Message','Press OK to terminate');  
xdel();           // All done! Delete the window.
```