ECEN203 Circuit Analysis
Graphing with Matlab

June 14, 2016

Matlab includes a plethora of specialised plotting commands that can produce a wide variety of high quality plots. With wise use Matlab can easily make scientific graphs that are far superior to the default outputs of software such as spreadsheets (for example). Of course, if abused, it can also produce a wide range of horrendous crimes against data.

This document will attempt to introduce some of the more critical plotting commands of Matlab. Along the way we will introduce some of the concepts that make a good technical graphs.

Philosophy The conventions for the production of good technical graphs are not the same as those for the types of financial and management graphs to which we are all exposed. In scientific and technical contexts, the aim is to convey information as clearly as possible. Graphs should not be used to deceive or trick the viewer, nor should they be "pretty" for the sake of it. Consequently a good technical graph is generally very simple. Every pixel added to a figure should be there for a reason.

1 Basic plotting commands

Let's create some data to plot throughout this tutorial. We will need a few types of data, so lets make a few different datasets.

```
>> t = 0:0.01:10;
>> x = exp(-t).*sin(2*pi*t);
>> y = exp(-2*t).*sin(2*pi*t);
```

Let's now being some actual plotting. Examine the output following commands (one at a time) to compare the produced graphs.

```
>> plot(x)
>> plot(t, x)
```

You should see that the second form plots `t` vs `x`, in contrast with the first choice.

We can alter the appearance of the plotted line within the `plot` command. Try for example the following examples.

```
>> plot(t, x, 'r--')
>> plot(t, x, 'g+')
```

This extra function argument specifies the color (red or green for the two examples) as well as the form of the symbol or line used to plot the dataset. This options are widely used to distinguish different plots in the same figure, so look at "LineSpec" in the Matlab help to see the options.

Multiple datasets can be plotted simultaneously.

```
>> plot(t, x, 'r', t, y, 'b')
```

However, if you want to add a new plot to an existing plot, then you will need to use `hold on` and `hold off` to prevent the second plot command from overwriting the first. Matlab will continue to add to the plot until the hold off command is received.

```
>> plot(t, x, 'r')
>> hold on
>> plot(t, y, 'b')
>> hold off
```

1.1 Axis limits and labels

The default graph made by Matlab is pretty good. However, there are a few problems with the graph that we have currently produced. For example, in this graph there is not much happening after $t = 8$. Let's restrict the x axis to a smaller range to make things a little clearer.

```
>> xlim([0,8]))
```

Extension to the y-axis (and z-axis) limits using the `ylim()` and `zlim()` command should be self-evident.

The largest problem with our graph is currently the lack of axis labels. You should always include appropriate axis labels, including units.

```
>> xlabel('Time [s]')
>> ylabel('Amplitude [V]')
```

You can set the title of a graph using the `title` command. This can be useful when using Matlab for design or analysis work. However, technical graphs don't normally use titles, but instead rely on a figure caption to explain what the figure is about.

1.2 Legends

Our graph has two plots, and it is important that the reader be able to distinguish the meanings of the two graphs. Let's use a legend to provide that information.

```
>> legend('\sigma_1=-1','\sigma_2=-2', 'Location', 'NorthWest')
```

Adding `\ldots'Location','NorthWest'` at the end of the command allows the position of the legend to be specified. NorthWest is in fact the default, so if you are happy with the legend being in the top right of the figure, then it can be omitted.

The example shows how greek symbols can be used in the legend (and in fact elsewhere in producing graphs). Simply prepend a `\` character and then spell out the desired letter. Compare `\sigma` and `\Sigma` in the code above to see the effect. An underscore can be used to produce a subscript (`A_{sub}`) and a caret can similarly be used to make a superscript (eg `A^2` or `e^{j \omega t}`).

2 Graph Annotation

We often wish to add information to a graph that is not contained in a dataset. For example, we might wish to add some explanatory text, or lines showing some feature.

2.1 Text

While legend are simple and functional, in production graphs it is often preferable to annotate the plots directly.

```
>> text(1.2, 0.3, '\sigma_1 = -1')
```

2.2 Lines

We can also add lines using the `lines` command. Let's imagine that we wanted the signal in our graph to have an amplitude of less than 0.2 V. We will add some dotted black horizontal lines on the graph at ± 0.2 V.

```
>> line([0 8],[0.2 0.2], 'Color', 'k', 'Linestyle', '--')
>> line([0 8],[-0.2 -0.2], 'Color', 'k', 'Linestyle', '--')
```

The two vectors in these line expressions include the x values and y values of the points that are to be joined by the line. While only two points have been used in each of these example, more points can be included to draw more compelx lines.

3 Other graph features

3.1 Boxes

You may notice that the entire plot is surrounded by a box. Some purists argue that this box serves no purpose and should be removed. The presence of this box can be adjusted with the `box on` and `box off` commands.

Similarly the legend is surrounded by a box. This too can be turned off using

```
h= legend('\sigma_1=-1','\sigma_2=2')
set(h, 'Box', 'off')
```

This example shows the use of a so called *object handle* (`h` in this case). We will not delve deeply into handles in this tutorial, but they do provide a mechanism to fine tune the appearance of your figures.

3.2 Grids

Sometimes it is useful to include a grid on your graphs. You should *not* do this in general. Unless you want a reader to read particular values from your graph, you are better avoiding the additional visual clutter of a grid. Simply use `grid on` and `grid off` to toggle the grid as desired.

3.3 Subplots

It is often useful to include multiple small plots within one larger figure. This can be done with the `subplot` command. The arguments used for this command specify the desired number of rows and columns, followed by the number of the subplot to be next written into.

```
>> subplot(3,2,1)
>> plot(t,x)
>> subplot(3,2,4)
>> plot(t,y)
```

4 Other graph types

4.1 Error bars

Including error bars on a plot is often useful when presenting experimental data. Experiment with the following commands to get a sense of the possibilities.

```
>> t = [0, 1, 2, 4];
>> x = [1, 2, 4, 4];
>> e = [0.5 0.5 0.7 1]
>> e_upper = [0.5 0.7 1 1.5]
>> errorbar(t, x, e)           % Symmetric error bars.
>> errorbar(t, x, e, e_upper)  % Asymmetric error bars.
```

Errorbar can not cope with multiple data series, so you may need to use `hold on` and `hold off`.

4.2 Logarithmic axes

We often need to use logarithmic spacing on one or both axes. An example would be plotting an gain response as a function of frequency.

```
>> w = logspace(0,6,120);
>> G = j.*w ./ (j.*w + 1000);
>> semilogx(w, 20*log10(abs(G)))
```

In this case `semilogx` is used to make the x-axis logarithmic, while the y-axis remains linear. Similarly `semilogy` can be used to make the y-axis logarithmic. Finally `loglog` can be used to make both axes logarithmic.

```
>> w = logspace(0,6,120);
>> G = j.*w ./ (j.*w + 1000);
>> loglog(w, abs(G))
```

4.3 Others

There are numerous other specialised graph types that Matlab can produce. Have a look at the examples or help for `scatter`, `polar`, `surf` and `bar` to get a sense of some other useful possibilities. If you select "Graphics Help" from the Help menu of any matlab figure window you can see a quick overview of the options.