EEEN203 Analogue Circuits and Systems
Lab 5
Graphing with Matlab

Due: Not assessed

1 Graphing with Matlab

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.

1.1 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
```

1.2 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.

1.3 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}`).

1.4 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.

1.4.1 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')
```

1.5 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.

1.6 Other graph features

1.6.1 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.

1.7 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.

1.8 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)
```

1.9 Other graph types

1.9.1 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`.

1.9.2 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))
```

1.10 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.

2 Symbolic Computation with Matlab

Matlab has a powerful toolbox for symbolic computation. Symbolic computation enables describes the capacity to deal directly with mathematical expressions, and lets us do things like differentiate or laplace transform general expressions.

There are times when this is a useful alternative to "normal" numerical computation of Matlab (or other languages), so it is useful to have at least a basic understanding of the language's capabilities. In particular, the symbolic toolbox provides a great tool for checking your algebra when working through problems by hand. We encourage you to use it freely for that purpose, but please don't use it as a crutch to avoid working through things yourself, as that will only lead to trouble later.

This document merely serves as a quick introduction to the most pertinent capabilities for us. It is by no means intended as an exhaustive exploration of the symbolic toolbox. Work through the examples and feel free to explore further options and commands by consulting the matlab help.

2.1 Expression manipulation

Before symbolic computation can be used, we must first tell Matlab which symbols it should treat as normal variables, and which it should treat as abstract symbols. In matlab everything is assumed to be a variable unless we say otherwise using the `syms` or `sym` command.

```
>> syms w R C
```

At any time you can use the `whos` command, which will report all of the defined variables and tell which of them are symbols.

Once you have defined some symbols, you can use them to create more complex expressions. For example, let's find the impedance of a parallel resistor and capacitor. We can then use the toolbox to rearrange the expression into a nicer form.

```
>> Z = 1/((1/R)+w*C)
>> Z2 = simplify(Z)
```

The `simplifyFraction` command is generally quicker than `simplify` when dealing with fractions like this, though you won't notice the difference with this simple a fraction. When dealing with more complex expressions you might like to try both, as they sometimes return answers that have slightly different forms.

In any case, we can use the new expression to perform other calculations. For example, let's find the current that would flow through our parallel RC combination if we were to apply a 3 V ac signal.

```
>> i = 3/Z
```

Notice that if you `whos` to examine the new variable `i`, that it has automatically been created as a symbolic expression.

The expressions produced by the symbolic toolbox are sometimes hard to understand, due to the deep nesting of parentheses or divisions. Try using the `pretty` command can help understand such expressions. This command attempts to use the matlab command line to present an expression in form similar to a hand written representation.

```
>> pretty(i)
>> pretty(Z)
>> pretty(Z2)
```

2.2 Substituting for variables

If you know particular numerical values for some symbol in an expression, then the `sub` value let's you do the appropriate substitution. For example, the following examples find the current if $R = 10\,\mathrm{k\Omega}$, or $f = 1$ kHz respectively.

```
>> subs(i, R, 10e3)
>> subs(i, w, 2*pi*1000)
```

Exercise Use the `sub` command to find the current flowing through the RC combination discussed above if $R = 10$ kOhm, $C = 100$ nF and $f = 1000$ Hz.
Hint: You can use `subs(f, [var1 var2 ...], [val1 val2 ...])` to substitute values for multiple variables simulataneously.

2.3 Partial fractions expansion and related matters

A particularly useful command for us is `partfrac`, which performs partial fractions expansion.

```
>> clear
>> syms s
>> Y = (3 * s + 13) / (s^2 + 4 * s + 3)
>> Yp = partfrac(Y)
```

One of the quirks of the toolbox is apparent if we compare the two forms for Y in the code above. One would certainly hope that `Y-Yp` would be equal to zero. Try it! The use of `simplify` can help with this situation. Try `simplify(Y-Yp)`

You can use the `collect` to move in the other direction. In this case the command has the effect of gathering the expression over a common denominator.

```
>> Y2 = collect(Yp)
```

In general `collect` collects the coefficients of various powers of a variable. If necessary you can specify which variable should be used as the variable of interest. Contrast the outputs of the following two uses of `collect`.

```
>> syms s a b
>> Y= 2*a*s^2 + a^2*s^2 + 3
>> collect(Y)
>> collect(Y, a)
```

2.4 Plotting symbolic expressions

The `ezplot` command allows you to quickly plot algebraic expressions.

```
>> y = x^2 + 9*x + 7
>> ezplot(y)
```

You can then modify the resulting figure as you wish, using commands like `xlabel`, `title`, `xlim` and so forth. When you want to constrain the x-axis limits there is a shortcut that let's you include the desired axis limits within the `ezplot` call.

```
>> y = x^2 + 9*x + 7
>> ezplot(y, [0, 10])
```

Exercise Recreate the various mode shapes included in the notes using the symbolic toolbox. That is, plot the modes that correspond to signals $s = e^{\sigma t}$, $s = e^{j\omega t}$ and $s = e^{(\sigma + j\omega)t}$. You will need to substitute appropriate values into the general expression to produce the plots. PLot the responses only for $t > 0$.

2.5 Solving equations

Matlab can solve a symbolic equation for you. Notice that you need to use `==` to express the "equals" sign in such expressions. If you omit the equals part, then Matlab will assume that you wish to solve for the specified expression being zero. Sadly it is not always able to solve complicated problems (like cubic polynomial problems) as shown by this example.

```
>> solve(x^2 + 4*x + 20)
>> solve(x^3 + 4*x + 20)
>> solve(x^3 + 6*x^2 +11*x + 6 ==  0)
```

Both the `solve` command and the related `factor` command are useful for when dealing with identifying poles of a transfer function of higher order polyomials in preparation for partial fractions expansion.

```
>> factor(x^3 + 2*x + 20)
```

The symbolic toolbox can also can deal with the soution of systems of equations:

```
>> [x,y]=solve (x + y == 8, x - y == 2)
```

2.6 Symbolic Laplace and Inverse Laplace transforms

You can quickly find the Laplace and inverse Laplace transforms using Matlab's `laplace` and `ilaplace` commands. Use `heaviside(t)` to define a unit step and `dirac` to specify a dirac delta function.

For example, let's apply a signal $y(t) = 3u(t-2) + \sin(\omega t)$ to a system having transfer function $G(s) = \frac{3}{s+3}$ and find the output signal.

```
>> clear
>> syms w t s
>> X = laplace(3 * heaviside(t-2) + sin(w * t))
>> G = 3/(s+3)
>> x= ilaplace(X*G)
```

2.7 Symbolic calculus

If you have a suitably defined symbols, then you can perform symbolic differentiation and integration. Try the following to see how this works:

```
>> clear
>> syms a x
>> diff(a*x^2)
>> int(a*x^2)
```