Physics 326G
Winter 2006

Class 4

Today we'll learn how to make two-dimensional graphs. Matlab has several built-in functions for graphing data, and you can control just about every feature of a plot to make it look exactly the way you want it to look.

Let's start by creating a set of data that we can play with. Make a vector called *time* that runs from 0 to 20 in steps of 0.4. Then make a vector called *data*:

```
data = 30 + 3*time - 0.5* time.^2 + 0.001*time.^4;
```

We can make a simple graph of this function using the PLOT command. Type

```
plot(time,data)
```

This plots the data, joining the points with a blue line. The first argument of PLOT is plotted on the *x*-axis, and the second argument is plotted on the *y*-axis. The two vectors you are plotting must have the same number of elements; if not, you will get an error message. Your plot comes up in a separate graphics window. Unless you give Matlab specific instructions, the program will set the scales of the axes the way is thinks is best. We will see shortly how to change this.

There are several ways to change features of your plot. You can include extra arguments when you call PLOT. You can enter commands after you have called PLOT to change things. Finally, you can edit the plot in the graphics window itself. This last option is often convenient but it has one big drawback, that being that every time you want to do a plot, you have to make all the changes by hand each time. If you are going to make a plot several times, or if you want to reproduce a plot a few days later, it is best to make the changes by running commands from the command window (or in a program, as we will see next class).

Let's change our plot. We would like to plot the data as red circles rather than as a blue line. To do this type

```
plot(time,data,'ro')
```

The third argument here is a "line specifier" that tells Matlab how to indicate the points it is plotting. The 'r' means red here, and 'o' means to draw a circle at each point. There are several different line specifiers we can use, including different styles of connecting lines, different marker types. You can see what they are by looking at the HELP entry for the function PLOT. (The line specifier is a "string" – that is, an array of text characters. Strings are identified by enclosing them in *single quotes*. You can manipulate strings the same way you manipulate arrays of numbers, except that most arithmetic operations don't make sense with strings.)

We can make the circles a different size by adding an additional argument to PLOT. The circles are called *markers*. We can change the marker size like this:

```
plot(time,data,'ro','MarkerSize',12)
```

Here 'MarkerSize' is the name of a property of the plot, and 12 is the value we want to set that property equal to. We can fill in our circles by setting the property 'MarkerFaceColor' to 'r' for red.

```
plot(time,data,'ro','MarkerSize',12,'MarkerFaceColor','r')
```

Note that the property name and some of the property values are strings and have to be entered in single quotes. A plot in Matlab has many, many different properties, and you can set all of them if you want to, either in the PLOT command or afterwards using the functions GET and SET. To see all of the properties of our plot we have set a variable equal to the return value of our plot command, like this:

```
plot_handle = plot(time,data,'ro','MarkerSize',12,'MarkerFaceColor','r')
```

The variable *plot_handle* is what Matlab calls a *handle*, that is, an identifier the program can use to refer to the plot. Now type

```
get(plot_handle)
```

The result is a list of all of the properties of the object (your plot) identified by the handle *plot_handle*. To change one of the plot's properties after you have run the plot command, you can use the SET command. Try

```
set(plot_handle,'Marker', 's')
```

Some useful properties are

- LineWidth
- MarkerSize
- MarkerEdgeColor
- MarkerFaceColor
- LineStyle
- Color
- Marker

Now let's play with the axes of our graph. The first thing we need is labels for both the *x* and *y* axes. The commands to use are XLABEL and YLABEL:

```
xlabel('time (s)')
ylabel('f(t)')
```

You can easily change the properties of these labels. You can specify the font size, font color, and other properties in the command line by adding extra arguments:

```
xlabel('time (s)','FontSize',16,'Color','m')
```

It is also possible to change things using GET and SET as above. For example:

```
label_handle = xlabel('time (s)','FontSize',16,'Color','m');
set(label_handle,'FontAngle','i')
```

is one way to change the font from 'normal' into italics.

You can give your graph a title using TITLE.

You can place text anywhere on the graph using TEXT. You have to specify the position of the text by giving its *x* and *y* coordinates.

(Matlab knows Greek letters: to make a title like 'A plot of α versus Γ' you would type

```
title('A plot of \alpha versus \Gamma')
```

\Gamma gives an upper-case gamma, and \gamma gives a lower-case gamma.

You can change the ranges of the *x* and *y* axes in two ways. The command AXIS takes as an argument a four-element array consisting of the minimum and maximum values of the *x* coordinate and the minimum and maximum values of the *y* coordinate. Try typing

```
axis([5 10 25 35])
```

Typing

```
axis auto
```

will put the axes back to Matlab's automatic setting.

Explore the functions GRID and LEGEND

Normally, when you enter a second PLOT command, Matlab will erase the existing plot. If you want to keep your original plot and draw the new graph in a new window, just type

```
figure
```

This will open a new graphics window. Each graphics window will have a number (1, 2, etc.), and you can switch between them by typing

```
figure(1)
```

```
figure(2)
```

and so on. You can clear an existing figure using the function CLF.

Often you will want to plot more than one set of data on a single graph. You can tell matlab to keep the existing plot rather than erasing it using the HOLD command.

```
hold on
```

tell Matlab to hold the current plot, so that you can plot more data in the same window.

```
hold off
```

turns the hold off, so the next plot in this window will cause the existing graph to be erased.

```
hold
```

by itself toggles the current HOLD setting.

The graphs you have done so far have had linear axes. Often you will want to plot data with one or both of the axes having logarithmic scales. Make an array *x* that runs from 1 to 1,000. Then calculate $y = x^{-2}$. Now plot *y* versus *x*. This works, but the values of *y* get small very quickly and it is very hard to get any information from the graph. In a case like this a log-log graph is much more useful. You do this simply by typing

```
loglog(x,y)
```

The same line specifications and properties that you used with PLOT can be used with LOGLOG. To get a logarithmic *y* axis and a linear *x* axis use

```
semilogy(x,y)
```

This is useful when you are looking at radioactive decays or other things that decay exponentially with time. There is also SEMILOGX which gives you a logarithmic *x* axis.

Matlab has a function called ERRORBAR which allows you to plot error bars along with your data. Define these arrays:

```
xx = 1:6;
yy = [1.1 1.9 2.6 3.5 4.1 5.7];
err = [0.1 0.1 0.2 0.1 0.2 0.1];
```

Here the array *err* contains the uncertainties in the data in the array *yy*. Now do

```
errorbar(xx,yy,err,'ko')
```

There are other plotting functions that produce special kinds of plots. Try

```
bar(xx,yy)
```

The command PIE produces a pie chart. Look it up with the help utility and figure out how it works.

One way to produce several small graphs on one figure is with SUBPLOT. SUBPLOT takes three arguments. Imagine you are producing an *m* by *n* array of plots. The first argument is *m*, the number of rows in your array of plots, and the second is *n*, the number of columns. The third is the number of the graph you want to make current, with 1 being the top left plot and *m*n* being the bottom right. Set up an array *theta* running from zero to 10π in steps of π/20. Then use SUBPLOT to plot cos, sin, cosecant, and secant of *theta* in separate plots on the same figure like this:

```
subplot(2,2,1)
plot(theta, cos(theta));
subplot(2,2,2)
plot(theta, sin(theta));
```

and so on. (You will need to play around a bit to make the graphs of secant and cosecant look nice. Try setting the axes so that the *y* axis runs from -10 to 10 and plotting with symbols instead of a line.)

Now demonstrate your expertise with graphing by doing the following exercises.

1. The position of a particle moving along the *x* axis is given by $x(t) = 0.4t^3 - 2t^2 - 5t + 13$ meters, where *t* is the time in seconds. Derive expressions for the velocity *v* and acceleration *a* of the particle. Then use SUBPLOT to make three plots on the same page, showing the position, velocity, and acceleration as a function of time for *t* from 0 to 7 s, with *x* at the top of the page, *v* in the middle, and *a* at the bottom. Plot *x* as a blue line, *v* as a green line, and *a* as a cyan line. Label all of the axes showing correct units. Give the graph a title.

2. Plot $y1 = x^{1/2}$ as a dashed red line and $y2 = x^{1/4}$ as a dotted magenta line, both on the same graph. Let *x* run from 0 to 2. Include enough points so that your plots look smooth. Put a legend on the graph that identifies each curve.

3. Do the same thing, but on a log-log graph.

4. The energy levels of an electron in a hydrogen atom are given by $E_n = -13.6/n^2$ in units of electron volts, where *n* is the quantum number of the energy level and the energy is measured relative to energy required to remove the electron from the atom (in other words, at $E = 0$ the electron is no longer bound to the nucleus). Plot E_n as a function of *n* for *n* up to 50, using a logarithmic energy axis. Plot the data points as filled-in black triangles. Label your axes and give the plot a title.

5. A student surveyed his physics class to determine who the students' favorite physicist was. He got the following results:

Heisenberg	5
Schroedinger	8
Einstein	17
Hawking	10
Faraday	3
de Bruyn	1

Make a pie graph that illustrates these results. Label each segment of the pie with the physicist's name. (Note that only one student in the above class passed the course!)