HOW TO PLOT SCIENCE GRAPHS WITH MATLAB

A guide to using Matlab for scientific plots in Physics 359E

INTRODUCTION

The third year lab generates a large amount of scientific data that is best presented in the form of graphs. However, not everyone is familiar with the graphical tools available in the lab and on many computers elsewhere. This short note summarizes how to make simple data plots, and plots of algebraic functions – for example, fits to the data points – using the computing language Matlab. Matlab is available on the computers in the lab. You may also have bought a student edition of Matlab for a previous course in applied maths or physics.

Creating a graph of a set of data points

Suppose you have a file on one of the lab computers that contains science data in the form of an $x - y$ table: two columns of numbers (or perhaps three if you have a column with measurement errors). The following recipe should enable you to produce a graph of these data which you can include in your lab report, either as a figure that you import into the word processor file of your report, or as a stand-alone figure which you can simply print out and staple into your report.

- Start Matlab from the icon on the desktop (or search for Matlab using the Find command). The programme will open with a screen showing three windows: the Command Window, where you enter Matlab commands after the prompt; the Workspace window, which shows the names and sizes of the data variables and arrays you have defined; and the Command History window, which lists the commands you have recently typed into the Command Window. There will also be a number of pull-down menus and buttons.
- Since you want to make a graph, your first job will be to import data that you want to visualize. It will be *very helpful* if your data are in a file with a name ending in the extension ".dat"; otherwise Matlab will really not believe that the file contains data, and will want to treat it as a programme. Let's suppose that your file is called *mydata.dat*, and that it has three columns of numbers, say an x value (perhaps energy), a y value (for example, number of counts with that energy), and an error or uncertainty σ. The x, y, σ data should simply be separated on each row by spaces.
- In the Workspace window, click on the Load Data File button at the top of the window. A pop-up will open. Select Files of Type: All Files. Navigate to the directory (folder) where you have your data. Select the file you want with the left mouse button, and click on Open. The Import Wizard will open, and (hopefully) show the first part of your file. Avoid the temptation to do anything with the Wizard except clicking on Next, then Finish. Your data array should now show in the Workspace window.

- Now the way graphing in Matlab works is that you must provide the plot routine with a list of x values at which you want plotted point, and a corresponding list of y values, one for each x point. You can do this in two convenient ways. The most direct is to tell the plot programme which columns in your data file to use by typing the following command after the prompt in the Command Window:
  ```
  >> plot(mydata(:,1),mydata(:,2))
  ```
  This command invokes the plotting programme, and gives it a list of x points (the first column of your datafile *mydata*; the colon ":" tells Matlab to use all of the points in the column, and the 1 tells it to use points only from the first column) and corresponding y points (the second column in your datafile). When you enter this command, a new little window called Figure No. 1 appears on the screen. This should have your data plotted in a reasonably chosen box. The Figure window has a number of buttons (look at these); one useful one is the Print Figure button, which sends your figure to the printer. Another useful one is the Save Figure button, which stores the figure for later use in your lab report.

- The other way of making a graph, which you may find a little clearer, is to first define two column vectors, one with the x values, and the other with the y values, as follows:
  ```
  >> xdata=mydata(:,1);
  >> ydata=mydata(:,2);
  ```
  where the semicolon ";" after the command simply tells Matlab not to bother listing the whole new column vector *xdata* on the screen. Then
  ```
  >> plot(xdata,ydata)
  ```
  produces the same figure as before.

- The command
  ```
  >> errorbar(mydata(:,1),mydata(:,2),mydata(:,3))
  ```
  produces the same plot as before, but uses the third column of *mydata* to plot errorbars on each point in the figure.

- You can make your graph much fancier, of course. Melissinos & Napolitano describe a number of the useful commands for labelling axes, etc, in Appendix B. One very useful command is
  ```
  >> hold on
  ```
  which allows you to plot a second graph over the first one on the same figure (that is, the first graph is not erased when you repeat the plot command with another dataset). Then
  ```
  >> hold off
  ```
  releases the graph for a fresh start.

Creating a graph of an algebraic function $y = y(x)$

Matlab can also be used to create a graph of an algebraic expression, using a syntax very much like that used in other computing languages such as FORTRAN or C. The following recipe will enable you to graph functions with Matlab. You can use Matlab to plot the values of a function at all the same x values as your experimental data, so that you can put both your data and a fitting function onto the same graph.

- Start Matlab. If you have a data file in it, you can use the x values of the data (the column vector *xdata* from the section above) to compute the values of the fitted function; otherwise, you will need to start by creating a new column vector with a series of x values at which Matlab will compute the y value of your expression. Let's suppose that you don't have a data file to start from, but simply want to plot y at a series of x values increasing from 10 to 20 with steps of 0.5.

- After the prompt in the Command Window, type
  `>> xeqn = 10:0.5:20`
  This creates a column vector at with values $x = 10, 10.5, \ldots, 20$.

- Next you need to create a column vector with the y values corresponding to your x values. Suppose that the equation that you want to plot is

  $$y = 3.0 \exp(-0.2x^2) + 20.$$

  You can compute the y values corresponding to the x values in your array *xeqn* by typing after the prompt in the Command Window the following:
  `>> yeqn = 3.0 .* exp( -0.2 .* xeqn.^2) + 20`
  This equation instructs Matlab to create a column vector of y values called *yeqn*, with one value evaluated for each element of the column vector *xeqn*.

- Matlab, like other computer programming languages, has various symbols for the algebraic operations it needs. Four of these symbols appear in the Matlab equation above: the assignment operator "=" which tells Matlab to create a new variable *yeqn* and to compute values for all of the values of the column vector *xeqn*; the "+" operator, which has the usual meaning regardless of what kind of objects one adds (scalar number, matrices, etc); and the multiplication operator ".*", which tells Matlab to do element-by element multiplication (that is, each numerical value of *xeqn* is taken in turn, and a corresponding value of *yeqn* is computed, rather than doing some kind of matrix multiplication); and the operator ".^" which tells Matlab to raise *xeqn* (element-by-element) to the 2nd power.

- In general, Matlab (like other similar programmes such as the – free – Scilab) uses different symbols for operators which act differently in matrix algebra than in simple scalar or array arithmetic. In particular, *matrix* multiplication, division, and raising to a power are given the symbols "*" , "/" , and "^", while operators for element-by-element array multiplication, division, and raising to a power are ".*", "./", and ".^". Functions you may need mostly will have familiar names such as log, exp, sqrt, cot, or abs. To get a list, type
  `>> help elfun`

- You can now plot a graph of your function as before, or include it in a graph with your data plot by turning hold "on" for the graph that has your data in it, and using `plot` to put the theoretical curve over the data.

- The same technique can be used to create a column of data with is the difference between your actual data and your fitting function – just create a new column vector
  `>> diff = xeqn - xdata`
  and plot it on the same graph.

There area a lot of small tricks that experienced Matlab users will be able to use in this process, and which non-experienced users will gradually discover. Don't be discouraged if your first couple of tries don't quite succeed. Try again. Experimentation will be rewarded.