IA Scientific Computing 2017-18

Session Two
"Publication Quality Graphics"

October 2017

The second session of the course focusses on producing high quality or "publication quality" scientific graphics. You should work through the tasks given below, in conjunction with the notes from the introductory lecture, the course reference book, and by using the MATLAB online help.

The worksheet is split into three sections:

(a) The first section teaches you how to load scientific data into MATLAB from a typical plain text data file, then to use it to generate and export a high quality figure.

(b) In the second section you will use these skills to build a figure for a publication quality report, which you will submit as a PDF file on the course Moodle site.

(c) Finally, the last section includes some more advanced plotting methods which will give you even more control over your scientific figures. It doesn't matter if you don't fully complete this section.

We actively encourage you to work with your colleagues and discuss the material - however **your submission must be your own work**. If you get stuck or don't understand something, ask the demonstrators for help, but you should make sure you have thought about the problem yourself, first.

Microsoft Windows Users:
Most versions of Microsoft Windows hide the extension part of filenames (e.g. showing only "image" instead of "image.png"), which can make scientific work quite confusing. Before starting this session you might find it helpful to adjust this setting to show full filenames:

(a) Open the Windows File Explorer from the Start Menu, or by pressing Windows-E

(b) Select File > Options from the menu

(c) Go to the "View" tab and untick "Hide extensions for known file types"

(d) Press OK.

1 Generating Publication Quality Figures

1.1 Reading data from a comma separated value (CSV) data file

In Scientific Computing, we very often end up with data from an experiment or simulation in a data-file. The simplest data files are plain text files, which just contain a series of characters, usually arranged into rows and columns, with column entries separated by a particular character. Comma separated value (CSV) files are such files where numbers are separated by a comma character, ",".

There are many ways that MATLAB can load in, or 'import' data files, and many different types of files are supported[1]. However, the simplest way to load in columns of data from a plain text file is to use the **dlmread** command - which we will focus on. The command requires four parameters which are (in order):

(a) the filename to read from (remember text must be given between single quotes)

(b) the character separating the columns

(c) the row to start reading from (the first row is counted as row zero)

(d) the column to start reading from (the first column is counted as column zero)

For example to read in a typical data file with one header line, and put the results in a matrix called **data**, you would use the command:

```
>> data = dlmread('myfile.dat', ',', 1, 0);
```

Remember, these arguments were also explained in the lecture - it is important to understand what these do.

Now try reading in a file for yourself:

(a) Locate the file **example.dat** which is available on Moodle and put a copy of it in your MATLAB working directory.

(b) Open Notepad (or a similar text editor) and drag-and-drop the file **example.dat** into Notepad so you can examine its contents. (You could also go into the MATLAB script editor and open the file using the "Open as text..." option from the menu.) Which are the header lines, and which lines contain data? Think about what rules you could use to tell someone else how to distinguish header lines from the data (there are several ways).

(c) By following the example above, use the **dlmread** command to import the contents of the data file into MATLAB.

(d) Open a new script in the MATLAB editor and put the **dlmread** command you just used into it. Save the script and check you can run it.

[1] For more information about loading in different types of files, see pages 17 to 22 of the course-book.

(e) Extend your script to separate the three columns of data that you have just imported into three vectors, called **a**, **b**, and **c**, respectively. You will need to use the matrix manipulation techniques that we introduced last week (you may want to refer back to your notes).

(f) Check that the contents of your variables **a**, **b**, and **c** are consistent with the original data file (i.e. compare a little of their contents with the contents of the file in Notepad).

1.2 Plotting a High Quality Figure

After loading a set of data, we very often wish to visualise the information.[2] Within MATLAB you can both use interactive "point and click" tools, or command line functions, to draw and refine plots. We will focus on the command line functions as these are more powerful, and they can be combined together into a script that you can use again later.

The general process is to generate a line using the **plot** command. To add further lines to the figure (rather than replace the current line), you need to use the **hold on** and **hold off** commands. You can then adjust the range of the figure, add axis labelling and a legend, if required.

Some examples are as follows:

```
>> plot(x1,y1)

>> hold on
>> plot(x2,y2,'r')
>> plot(x3,y3,'bx')
>> plot(x4,y4,'--k')
>> hold off

>> xlim([0 10])
>> ylim([-1 1])

>> xlabel('Name of the x-axis')
>> ylabel('Name of the x-axis')
>> title('My graph')

>> legend('First line', 'Second line', 'Third line', 'Fourth line')

>> grid on
```

Now try these tasks:

(a) Take the script you wrote in the last section and add commands to plot the second column of data (y-axis) against the first column (x-axis). You should see a series of oscillations which gradually decay away.

[2] For more information about plotting and refining graphics, see pages 27 to 40 of the course-book.

(b) Change your script to plot the third column of data against the first column of data and re-run it. You should see an 'average' of the decaying oscillations.

(c) Add another **plot** command to your script, along with **hold on** and **hold off** commands, to plot both curves on the same figure.

(d) The graph does not change much for x-value greater than about 40, so add an **xlim** command to your script to refine the x-range. You may also wish to adjust the y-range if you think it would look better that way.

(e) Add labels to your axes and add a title to your graph.

(f) Add a legend to describe the lines using the **legend** command.

1.3 Exporting your figure to use elsewhere

Once you have produced a figure that you are happy with, you often want to export that figure to use in a report or paper. Both vector graphic files and bitmap files were discussed in the lecture. Vector files give the highest quality graphics, but are sometimes tricky to import into wordprocessors. In this course we suggest using PNG bitmap files with a 600 dpi resolution (dpi = dots per inch). These give good quality results, and are easy to use. Please DO NOT use use JPEG images for scientific graphics; the lossy compression algorithm is good for photographs, but handles scientific images poorly.

To export a high quality PNG image, you typically use a command like:

```
>> print('myfigure.png','-dpng','-r600')
```

Tasks:

(a) Export your figure as a PNG file and check that you can open it correctly.

2 Producing a Publication Quality Report for Submission

Each week you need to prepare and submit a short piece of work during the session. This week you need to submit a single PDF file containing a figure along with a suitable caption.

2.1 Plotting a comparison of share prices

The share prices for the three high-technology companies Apple, IBM and Microsoft have been put together in the data file **`shares.dat`** which is available on the course Moodle site. The first column of data in the file contains the date, given as a year value that is expressed as a decimal (for example, July 2003 would be expressed as 2003.5). The second, third and fourth columns contain the share prices in US$, for Apple, IBM and Microsoft, respectively.

You should make a labelled plot of the share prices of these companies between the years 2000 and 2015.

The main steps you should follow are:

(a) Locate the file **`shares.dat`** on Moodle (in the resource section for Session 2), and save a copy of it in your working directory.

(b) Make a copy of your script from the previous section, and modify it to load and plot the shares data.

(c) Amend your script to adjust the axis ranges and labelling appropriately.

(d) Add an appropriate legend to distinguish the three lines.

(e) Export a high resolution PNG file of the new plot.

2.2 Incorporating your figure into a report

You now need to incorporate your figure into a wordprocessor document. Normally such a document would contain the text of a report, but for this exercise we only require you to include a title, figure and figure caption.
You will need to:

(a) Start your wordprocessor and create a new A4 document.

(b) Add the PNG file that you generated, using the "Insert Picture" function (or similar) and position it roughly in the middle of the page.

(c) Add a figure caption just below the image as indicated in the lectures. The figure should be numbered and start with something like "Figure 1: Plot of...", even though there is only one figure. The caption should then contain a statement of what the figure shows, and what you think the main message is that the reader should take away from looking at the figure.

(d) Add a title to the page, along with your name, CRSID and college.

(e) Save the file for your convenience, then export it as a **single page PDF file** ready for submission.

2.3 Checking your submission

Before you submit your PDF file, you should check that it:

- contains a neat, high-resolution plot, showing the share price of the three companies between 2000 and 2015
- is correctly labelled and contains a suitable legend
- includes a figure caption describing the figure and telling the reader "what to look at"
- includes a title, your name, CRSID and college
- is presented clearly, so it is easy for someone else to read.

2.4 Submitting to Moodle

Once you are satisfied with your PDF file, you are ready to upload your submission to Moodle. You should follow the following steps:

(a) Go to the course Moodle site, at **`https://vle.cam.ac.uk`**

(b) Follow the "Upload PDF..." link for Session 2. The link will bring up a page with more information, from which you can add your files.

(c) Find and select your PDF file, and upload it to the web-page.

(d) When you are sure you have uploaded the file, accept the submission statement then press the "Save changes" button on the assignment page. Until you do so, you will not have actually submitted your work.

It is essential to check your work has been submitted correctly. On submitting, you will be sent a confirmation email - if you do not receive this email, it is likely that you have not actually submitted your work. You can also check that your submision status has changed to "Submitted for Grading" on Moodle. If you need to replace your submission (before the deadline), you can simply upload a new version.

3 Further Exercises

Now that you have submitted your work, you have time to learn some more about the more advanced plotting features in MATLAB. You do not have to tackle these sections in any particular order.

3.1 Logarithmic graphs

Logarithmic plots are widely used in scientific computing. In MATLAB they can be made easily by using the **semilogx**, **semilogy** or **loglog** commands instead of the usual **plot** command, producing figures with logarithmic x-axes, y-axes, or both, respectively.

Tasks:

(a) Make a plot of the exponential function $y = e^{-x}$ in the x range between 0 and 100.

(b) Repeat the plot using logarithmic axes, in the x direction, y direction, and on both axes. How does the exponential curve show in in each case? How does each plot change if you plot $y = e^{-ax}$, with different values of a?

3.2 Fine grained control over graphics with `get` and `set`

As mentioned in the lectures, you can control the fine detail of every graphical object within MATLAB using the **get** and **set** commands. It can be particularly useful to adjust the properties of the axes such as the font-size, and the thickness of a line in a plot.

The first step is to obtain the 'handle' which refers to a particular graphical item on the screen. You can request the handle to the current set of axes using the **gca** command (which stands for "get current axes"). Alternatively, you can keep record of the handle of an object when you first create it, such as when you use the **plot** command. For example:

```
>> h = gca;        % stores handle to the current axes in h
>> p = plot(x,y)   % p stores the handle to the line
```

You can then use the **get** command to find all the possible parameters associated with a handle that can be adjusted, and their current values. Similarly, the **set** command can be used to adjust the value of any particular parameter, as shown in these examples:

```
>> get(h)                    % lists the parameters
>> get(h,'fontsize')         % gets fontsize value
>> get(p,'linewidth')        % gets linewidth value

>> set(h,'fontsize',16)      % changes a parameter
>> set(p,'linewidth',2)      % changes a the line-weight
```

The combination of a parameter and its value are known as "parameter-pairs". You can also adjust parameter pairs when you first run many commands. For example:

```
>> plot(x,y,'linewidth',2) % plots and adjusts parameters
```

Tasks:

(a) Use one of your scripts to replot one of the figures from earlier. Use the **gca** command to obtain the axes handle, followed by the **get** command to find out what parameters can be adjusted.

(b) Change the fontsize in your figure to something considerably larger.

(c) How would you adjust the thickness of the axes frame?

3.3 More Unusual Graphs

MATLAB can produce a range of more unusual figures, which are well suited to particular scientific applications.

Try the following:

(a) The **pie** command can be used to make pie-charts. Use the command **doc pie** to bring up the online help for the **pie** command, and use the examples to find out how to use the command. Check you understand by making a pie chart to represent the elements in the vector $x = [5\ 3\ 1]$.

(b) Polar plots can be very useful when plotting angular functions. Use the **polar** command to plot the function $r = \sin(\theta)$ over the range $\theta = 0 \ldots 2\pi$.

(c) Plots with error-bars can be made using the **errorbar** command rather than the **plot** command, although the two are very similar. Use **doc errorbar** to read the built-in help information about making errorbar plots. Try making up some data and errorbars to check that you can use the command correctly.