Department of Physics

Introduction to MATLAB

2014

GLOBAL AIMS

- Familiarisation with basic MATLAB structure and syntax
- Use MATLAB to analyse experimental data and perform an error analysis
- Basic plotting features

BACKGROUND

Assumed knowledge

- First year physics
- First year mathematics

Name :	
Student Number :	
Demonstrator :	
Date :	

CONTEXT

In second and third year labs students are encouraged to use MATLAB for manipulating and presenting data in their lab reports. MATLAB is a powerful tool for data analysis and visualisation, with many advantages over Excel. The purpose of this introductory session is to introduce you to the key features of MATLAB that will be useful during laboratory classes.

MATLAB presents a bit of a learning curve, especially if you don't have much computing experience. Remember that we are here to help you so just ask if you get stuck.

During the semester you may develop some MATLAB files that will be useful to you in other labs; save your code on a USB and bring it with you to your next lab!

These notes were developed by:

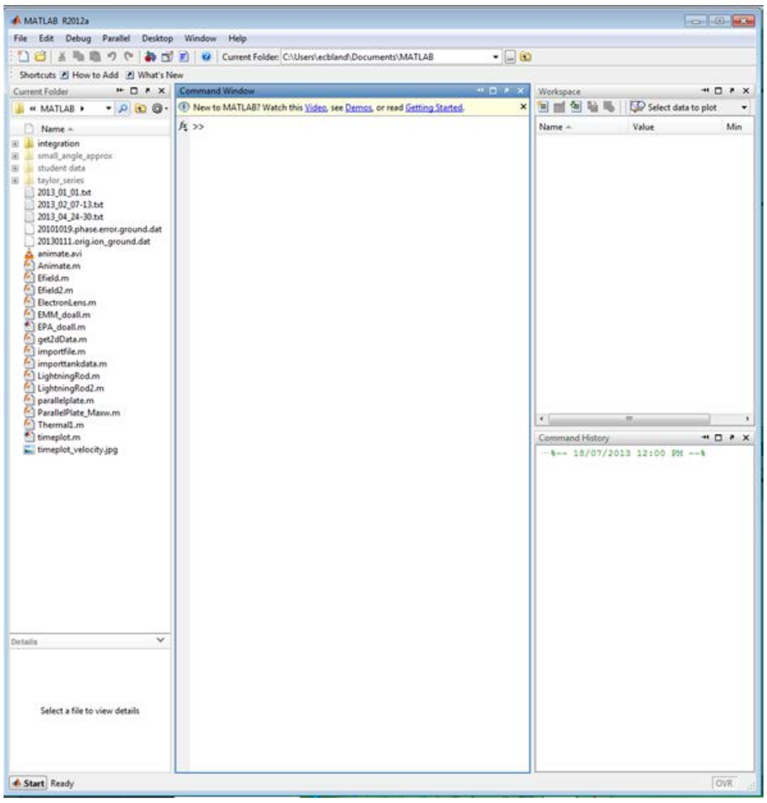
Emma Bland
Department of Physics
La Trobe University
e.bland@latrobe.edu.au

What is MATLAB?

MATLAB is a specialised computer software package for numerical calculations with matrices (MATrix LABoratory). It is a powerful tool for data analysis and visualisation, with many advantages over Excel.

No, really. What is it?

Let's find out! Log on to your computer and open MATLAB. The screen will look something like this:



New to MATLAB? Watch this Video, see Demos, or read Getting Started.

```
>>
```

```
%-- 18/07/2013 12:00 PM --%
```

The panel in the centre is the command window – this is where you can give instructions to MATLAB. The top right panel is the workspace – this tells you the names and properties of all the variables currently stored in the computer's memory. Each time you create a new matrix it will appear in the workspace. Below the workspace is the command history panel, which tells you everything you've already typed into the command window. The panel on the left shows the working directory where MATLAB will look for files.

PART 1: Getting Started

Let's begin by getting a feel for some of the basic features of MATLAB. Type the following into the command window and press ENTER:

```
>> a=5; b=10; a+b
```

This creates **scalars** a=5 and b=10 and then **adds** them together. Notice that `a` and `b` have appeared in the workspace (top right panel). The answer to a+b is stored as the variable `ans`.

Now try some more operations:

```
>> a=1:10
>> a'
>> sum(a)
```

Notice that the first command creates a **row vector** (1d matrix) of integers from 1-10 inclusive. Make a note of what effect the other two commands have on the matrix `a`.

The following code **creates a 1d matrix** of values from 1-10 separated by 0.5:

```
>> a=1:0.5:10
```

Alternatively, you can specify the number of elements in the matrix, rather than the element spacing:

```
>> a=linspace(1,10,19)
```

This creates a matrix with 19 entries equally spaced between 1 and 10.

You can also create **multi-dimensional matrices** using commands such as:

```
>> a=ones(3,4)
>> b=zeros(2,5)
>> I=eye(5)
```

Again, make a note of what each of these commands does.

MATLAB Tip: put a semicolon (`;`) at the end of a statement suppress the output. This is particularly useful for large matrices as it won't fill up the command window with large outputs.

MATLAB Tip: clear the command window by typing `clc`.

More fun with matrices:

Enter the following matrices into MATLAB:

```
>> A=[0 2 4;5 2 0;1 1 1]
>> B=5*ones(3,3)
>> C=[1; 2; 1]
```

Matrix addition/subtraction (matrix dimensions must agree):

```
>> A+B
>> A-B
```

Matrix multiplication (notice that the matrix product is *not* commutative):

```
>> A*B
>> B*A
>> A*C
>> C*A
```

(The last expression will return an error because the inner matrix dimensions do not agree.)

Element-wise multiplication, which multiplies corresponding elements in each matrix ($A_{ij} \times B_{ij}$).

Notice that element-wise multiplication *is* commutative:

```
>> A.*B
>> B.*A
```

Matrix inverse (only for square matrices):

```
>> inv(A)
>> inv(C)
```

Perhaps we want to **extract the second column** of the matrix A:

```
>> A(:,2)
```

Or **extract the third row** of matrix A:

```
>> A(3,:)
```

MATLAB Tip: use the ↑ arrow to display the last command you entered into the command window.

PART 2: Small Angle Approximation

The small angle approximation is used to approximate the trigonometric functions $\sin\theta$, $\cos\theta$ and $\tan\theta$

$$\sin\theta \approx \theta; \quad \cos\theta \approx 1 - \frac{\theta^2}{2}; \quad \tan\theta \approx \theta$$

where θ is measured in *radians*.

In this exercise you will investigate the validity of the approximation, $\sin\theta \approx \theta$.

Rather than typing files line-by-line into the command window, we are going to create a **script** (called an **m-file** in MATLAB). This allows you to execute multiple commands at once, and saves you from typing the same things again and again! Open a new script file in MATLAB (the button just under the File menu) and save it in the [username]/Documents/MATLAB folder. Type the following code into the script:

```
% SMALL ANGLE APPROXIMATION
% Plot the functions y=sin(x) and y=x from x=0 to x=pi/2

x=0:0.05:pi/2;

%PLOT YOUR FUNCTIONS
hold on                  %lets you plot multiple of functions on the
                         same set of axes
p1=plot(x,x,'.-b');      %plot y=x
p2=plot(x,sin(x),'-ro'); %plot y=sin(x)
```

Run the script by clicking ‘save and run’.

Let’s go through the script line-by-line to see what just happened. The first two lines are **comments**; these are for you to annotate your code to remind you what it does. Any text that appears after a `%` is ignored by MATLAB. The next line **creates a 1-d matrix of x-values**, ranging from $x = 0$ to $x = \pi/2$ in increments of 0.05. Type this line into the MATLAB command window if you’re not sure what’s going on. Finally, we get to the interesting part where the plotting happens. The command `hold on` is used to **display multiple plots on the same figure** (the command `hold off` will erase the previous figure before the next is plotted). Not surprisingly, the function `plot(x,y)` creates a **plot of x versus y**. The first plot (called p1) plots the straight line $y = x$ (x versus x), and the second plot (called p2) plots x versus $\sin(x)$. The extra arguments ‘.-b’ and ‘-ro’ are used to specify the colours and style of the lines so we can tell them apart. Type `help plot` into the MATLAB command window to see what each of these options means.

MATLAB Tip: type `help [function]` into the command window for more information about a function or command.

Of course the figure produced by the above script is rather limited; the plot needs a title, the axes aren't labelled, and it would be nice to have a **legend** so we know which function is which. Let's do that now. Add the following code to the end of your script, choosing an appropriate **title** and **axis labels** for your plot:

```
%Title and axis labels
title('[plot title]');
xlabel('[x axis title]');
ylabel('[y axis title]');
```

```
%create a legend
legend([p1,p2],'y=x','y=sin(x)');
```

Modify your script to calculate the error in the small angle approximation over the domain $[0, \pi/2]$. That is, the difference between x and $\sin(x)$. Plot the error as a function of x and modify the legend as appropriate. For which values of x do you think the small angle approximation is valid?

____

____

One more thing: the axis labels and titles will probably be very small. Assist your demonstrator's tired eyes by **enlarging the text** so that it is easier to read:

```
>> set(gca,'FontSize',16)
```

`gca` stands for 'get current axes'. That is, the above command will be applied to the current figure window.

PART 3: Measuring absolute zero

Consider the following experiment: a container of gas is brought into contact with a liquid and they are allowed to come to the same temperature. The temperature and gas pressure are then recorded. This is repeated for several liquids at different temperatures:

Liquid	**Temperature (°C)**	**Pressure of gas (kPa)**
Liquid nitrogen	-196.0	21.1
Ethanol with dry ice (solid CO_2)	-73.5	55.7
Iced water	2.5	74.9
Tap water	21.5	82.5
Boiling water	99.0	100.7
Hot cooking oil	229.5	139.2

We are going to use the results above to estimate absolute zero. For an ideal gas, the pressure P, volume V and temperature T (in Kelvin) are related by:

$$PV = nRT$$

where n is the number of moles of the gas and R=8.31 J $mol^{-1}K^{-1}$ is the gas constant. Since we are working in degrees Celsius, the above equation can be rewritten as:

$$PV = nR(T_c + a)$$

where *a* is the conversion factor between degrees Celsius and degrees Kelvin, and T_c indicates that the temperature is in degrees Celsius. In the above experiment, the volume and amount of gas remain constant, so we have a linear relationship between temperature and pressure:

$$P = m(T_c + a) = \mathrm{m}\mathrm{T_c} + b$$

where we have substituted $m = nR/V$ and $b = ma$. If we set the pressure equal to zero, we obtain the value for absolute zero in degrees Celsius:

$$T_{c,0} = -{}^{b}/_{m}$$

Although we could just type the pressure and temperature values directly into MATLAB, let's have a go at **importing them from Excel**. The measurements are saved in the file `absolutezero.xlsx`. Download this file from the LMS and save it in the C:\Users\[username]\Documents\MATLAB folder. You should now be able to see this file on the left panel in MATLAB; right click on it and select *Import Data*. A dialogue box will appear showing the cells containing the data. Check that the correct cells are selected (just the temperature and pressure values, not the row/column names) and click the *import* button. MATLAB should display a message to tell you that the data have been imported. Close the dialogue box.

The data have been imported into a matrix called `untitled`; it should appear in the workspace. **Rename this matrix** as `data` by right clicking it in the workspace and choosing *rename.* To view the contents of the matrix, type the name of the matrix into command window:

```
>> data
```

Now that you have imported the data the long, tedious way, let's **import the data directly from the command line** in a single step. There is a special function to import data from Excel:

```
>> data=xlsread('absolutezero.xlsx')
```

(Refer to the Quick Reference Guide at the end of this document for instructions on importing data from text/ASCII files into MATLAB).

You should see two columns containing your data. **Extract** the first column into a matrix called `temperature`:

```
>> temperature=data(:,1)
```

Repeat the above process to extract the second column into a matrix called `pressure`.

Let's plot pressure as a function of temperature. Rather than using the `plot()` command, we can **plot the data with error bars** in a single step using the `errorbar(X,Y,Err)` command:

```
>> errorbar(pressure,temperature,0.5*ones(6,1),'.')
```

`0.5*ones(6,1)` is a 6 x 1 matrix containing the error values (in this case, ±0.5 °C) associated with each temperature measurement. If you're not sure what this syntax means, type it into the command window. For more information about error bars, type:

```
>> help errorbar
```

Now we are going to **fit a line** (polynomial of order 1) to these points:

```
>> [p,s]=polyfit(pressure,temperature,1)    %polynomial of order 1
```

The first value of the matrix `p` is the **gradient** and the second is the **y-intercept**. You can access these in MATLAB by calling `p(1)` and `p(2)` respectively. We will use `s` to evaluate R^2 for the linear fit (see below).

To **plot the line of best fit** and **display the equation** on the plot:

```
>> x=0:200;                    %matrix of x-values for plotting
>> yfit=p(1)*x+p(2);           %evaluate the line of best fit at each
                               value of x
>> hold on                     %overlay multiple plots on one figure
>> plot(x,yfit)
```

```
%display equation on plot (choose appropriate coordinates [xpos,ypos])
>> text(xpos,ypos,['y=' num2str(p(1)) 'x+' num2str(p(2))])
```

We should also calculate the **coefficient of determination (R^2 value)** for the linear fit. In MATLAB this requires a little bit of work:

```
>> Rsq=1-(s.normr^2)/(norm(y-mean(y))^2)
>> text(4,4.5,['R^2=' num2str(Rsq)])
```

(For this example, replace y with `temperature`, since these are the y-values)

Using the code from the previous section, create labels for your axes (with units) and an appropriate title for the plot. Adjust the **range of the axes** using:

```
>> axis([xmin xmax ymin ymax])
```

Make sure that you choose the ranges of the axes such that all the data points on the graph and the y-intercept are clearly visible.

What is the value of the temperature (in °C) when the pressure is equal to zero? Is this what you expected?

____

____

____

____

PART 4: Measuring a Spring Constant

Recall Hooke's law: the force F required to extend/compress a spring by a distance x is given by:

$$F = -kx$$

where k is a constant of proportionality (the 'spring constant') which depends on the material and construction of the spring.

In an experiment to measure the spring constant of a particular spring, a group of students attach different masses to the end of the spring, which hangs vertically, and record the length of the spring each time:

Mass (g)	Spring Length (cm)	Error in Spring Length (cm)
50	18.5	0.5
100	20.2	0.5
150	21.7	0.5
200	23.1	0.5
250	24.5	0.5
300	26.2	0.5

Since the mass is not moving when the spring length is measured, the restoring force provided by the spring is equal and opposite to the weight force due to gravity.

Download the file `springconstant.xlsx` from the LMS and save it in C:\Users\[username]\Documents\MATLAB).

Use the template below to create a MATLAB script that does the following:

1. Import the data into MATLAB from the Excel file `springconstant.xlsx`
2. Create 1d matrices of values for the force on the spring, the length of spring, and the error in spring length (all in SI units)
3. Plot force (x) versus spring length (y) with y-error bars
4. Fit a line to the data and display it on the figure
5. Calculate the R^2 value for the fit and display it on the figure
6. Determine the spring constant from the gradient of the line

```
%[Brief description of what the code does]
%[your name here]

% Import data into MATLAB


% Make column matrices of
%    (1) Force exerted by spring (in Newtons)
%    (2) Length of spring (in metres)
%    (3) Error in length of spring (in metres)


%Plot force (x) versus spring extension (y) with error bars
figure;
hold on
errorbar( , , ,'.')


%Fit a line to the data and plot it on the graph (with equation)


%Calculate R^2 value and display on plot


%Calculate the spring constant
k=
```

What is the value of the spring constant k?

What is the length of the spring with zero mass attached?

Finally, we are going to write a function that calculates the period of oscillation of the spring given the attached mass and the spring constant:

$$T = 2\pi\sqrt{\frac{m}{k}}$$

```
function [T]=spring_T(m,k)
%Calculates the period of oscillation of a spring with spring
%constant k and mass m attached

[equation goes here]

end
```

Complete the above function by adding the appropriate equation. Note that $\sqrt{\square}$ in MATLAB is `sqrt()` and π is `pi`. Save the script as `spring_T.m`.

What is the period of oscillation of the spring with a 200g mass attached?

PART 5: Taylor polynomials

Taylor polynomials are often used in physics to approximate equations. Recall that the Taylor expansion of the function $f(x)$ about the point $x = a$ is given by

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \cdots$$

Calculate by hand the first six terms of the Taylor series expansion of the function, $f(x) = \sin(x)$ near $x = 0$ (some of these terms will be zero).

MATLAB has an inbuilt function for calculating Taylor polynomials. Check your working using the following code:

```
>> syms x
>> f=inline('sin(x)')
>> taylor(f(x),x,0,'Order',6)
```

Note that MATLAB calculates the Taylor expansion to order n-1. That is, the code above gives the 5th order approximation (which has 6 terms).

The above code takes advantage of MATLAB's symbolic mathematics capabilities. It treats x as a symbol/pronumeral rather than a matrix with a fixed value. This allows it to perform algebraic manipulations on a mathematical expression (like you do by hand).

Now write a script to plot the Taylor series expansion of $f(x) = \sin(x)$ for n=1,3,5,9 on a single pair of axes. To plot the symbolic function f(x)=sin(x) we will use the MATLAB function `ezplot()` rather than `plot()`. Also plot $y = \sin(x)$ so you can compare it to your Taylor approximations. Here's some code to get you started:

```
% TAYLOR EXPANSIONS OF THE FUNCTION Y=SIN(X) ABOUT THE POINT X=0

%DEFINE FUNCTION
syms x
f=inline('sin(x)');

%TAYLOR POLYNOMIALS
taylor1=taylor(f(x),x,0,'Order',2);     %n=1
taylor3=taylor(f(x),x,0,'Order',4);     %n=3

%GENERATE PLOTS
hold on

a=-pi:0.05:pi;                              % define matrix a
plot(a,sin(a),'black','LineWidth',5);       % plot the function sin(a)

p1=ezplot(taylor1,[-pi,pi]);         % n=1
p3=ezplot(taylor3,[-pi,pi]);         % n=3


%SET LINE COLORS FOR EACH PLOT
set(p1,'Color', 'green');
set(p3,'Color', 'red');


%CREATE A LEGEND
legend([p1,p3],'n=1','n=3','Location','EastOutside')


%AXIS LABELS AND PLOT TITLE
```

Describe what happens as n increases:

____

____

____

____

Finished?
Submit your m-file for the spring constant exercise via the submission page on the LMS.

Save the m-files you created onto a USB (or email them to yourself) and bring them to the lab!

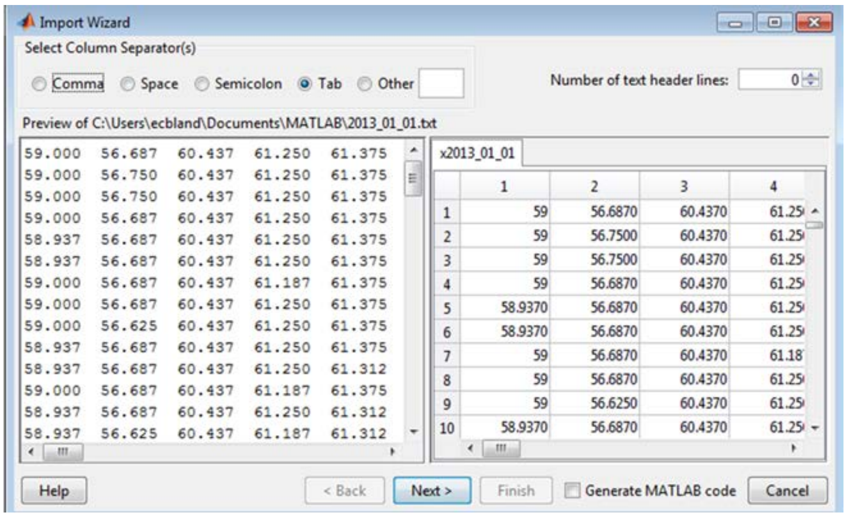
Quick reference guide

Setting the file path

The file path tells MATLAB where to look for m-files you have created. To set to the path go to File>Set Path. Click Add Folder (or Add with Subfolders), navigate to the appropriate directory, click ok and then press Save. The files in the current MATLAB search path will be displayed in the panel on the left of the screen.

Importing text files into MATLAB

If the file containing your data is in the working directory, right-click the file in the left panel and select *Import Data.* If the file is saved elsewhere, go to File > Import Data where you can browse for the file. This will launch the Import Wizard:



Comma   Space   Semicolon   [✓ Tab]   Other

Number of text header lines: 0

Generate MATLAB code

Preview of C:\Users\ecbland\Documents\MATLAB\2013_01_01.txt

```
59.000  56.687  60.437  61.250  61.375
59.000  56.750  60.437  61.250  61.375
59.000  56.750  60.437  61.250  61.375
59.000  56.687  60.437  61.250  61.375
58.937  56.687  60.437  61.250  61.375
58.937  56.687  60.437  61.250  61.375
59.000  56.687  60.437  61.187  61.375
59.000  56.687  60.437  61.250  61.375
59.000  56.625  60.437  61.250  61.375
58.937  56.687  60.437  61.250  61.375
58.937  56.687  60.437  61.250  61.312
59.000  56.687  60.437  61.187  61.375
58.937  56.687  60.437  61.250  61.312
58.937  56.625  60.437  61.187  61.312
```

x2013_01_01

	1	2	3	4
1	59	56.6870	60.4370	61.25
2	59	56.7500	60.4370	61.25
3	59	56.7500	60.4370	61.25
4	59	56.6870	60.4370	61.25
5	58.9370	56.6870	60.4370	61.25
6	58.9370	56.6870	60.4370	61.25
7	59	56.6870	60.4370	61.18
8	59	56.6870	60.4370	61.25
9	59	56.6250	60.4370	61.25
10	58.9370	56.6870	60.4370	61.25

On the left side of the Import Wizard is a display of the data file you are importing. Make sure that the number of header lines (e.g. column titles) in the box in the top-right corner is correct. You will also need to specify how the columns are separated (tabs, spaces, commas etc.).

If you would prefer to import your data directly from the command line or script file:

```
>> data=importdata('[filename]');
```

This will import the data into a matrix called data. Make sure that you check the data have been imported correctly. If the file is not located in the MATLAB file path, you will need to include the full path to the file (e.g. C:\Users\[username]\Documents\experiment_data\mydata.txt).

Importing data from Excel

You can use the Import Wizard for Excel files (it looks slightly different). Alternatively, there is a special MATLAB function for importing data from Excel from the command line:

```
>> data=xlsread('[filename]')
```

Again, if the file is not located in the MATLAB file path, you will need to include the full path to the file.

Plotting

```
figure              %create a new figure window
figure(1)           %go to figure 1 (for switching between figures)
clf                 %clear the figure window
hold on             %holds the current plot and all axis properties so
                     that subsequent graphing commands add to the
                     existing graph

hold off            %returns to the default mode whereby PLOT commands
                     erase the previous plots

p1=plot(x,y);       %plot x vs y, store in p1

p1=errorbar(X,Y,E);      %plot X vs Y with error bars ±E, and store in
                         p1. E must have the same length as X and Y.

title('[plot title]');        %plot title
xlabel('[x axis title]');     %x-axis label
ylabel('[y axis title]');     %y-axis label

legend([p1,p2,…],'label 1','label 2',…);  %create a legend

axis([xmin xmax ymin ymax])   %adjust axis range
```

Remember you can type `help [function]` *into the command window for more information about a function/command.*

Linear regression

```
% Fit a polynomial of order 1 (a line) to the points defined by
matrices x and y
>> [p,s]=polyfit(x,y,1)

% Compute the value of the fitted line for each element in the matrix
x. p(1) is the gradient, p(2) is the y-intercept
>> yfit=p(1)*x+p(2);

% Display equation of line on plot at [xpos,ypos]
>> text(xpos,ypos,['y=' num2str(p(1)) 'x+' num2str(p(2))])

% Calculate R² value and display on plot
>> Rsq=1-(s.normr^2)/(norm(y-mean(y))^2)
>> text(xpos,ypos,['R^2=' num2str(Rsq)])
```