Design and Computing 1

Introduction to Scientific Computing

Daniel Poole
Department of Aerospace Engineering
University of Bristol
`d.j.poole@bristol.ac.uk`

2017

LECTURE 8

Intro to Scientific Computing

- Have learnt the basics of programming including variables, loops, conditionals, files and arrays.

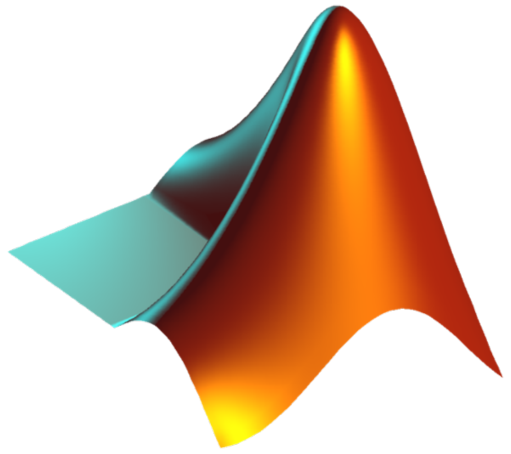
- Have learnt programming through the basic C language

TODAY

- Introduction to MATLAB
  - Where and why it is used
  - Manipulating matrices
  - Plotting

MATLAB

MATLAB (MATrix LABoratory) is a development package produced by Mathworks specifically for numerical scientific and engineering calculations. In the next two lectures, we will introduce what MATLAB is and how to use it. One thing to note, however, is that the full MATLAB installation is very large and has a massive functionality. We really will just scratch the surface of what MATLAB can do (MATLAB has $>$50 toolboxes which add extra functionality to the basic tool we will use).



MATLAB is a tool development package that requires code to be written (with similar syntax to C) which is then interpreted and run directly without the need for a compiler. For certain applications, it is a very effective tool, however, for high performance numerical simulations, it is rare to see MATLAB being run. We will probe some of the useful functions of MATLAB, particularly for what it was first designed for, which is matrix manipulation and analysis.

MATLAB

MATLAB can be used as an interactive, dynamic tool where we can directly type commands and get immediate results. Alternatively it can also be scripted as a program, in a fashion like a C program. Unlike C, errors reported in MATLAB tend to make sense and tell you exactly where the problem lies. If we know what MATLAB is efficient at, we can really take advantage of the power it possesses e.g. matrix multiplication

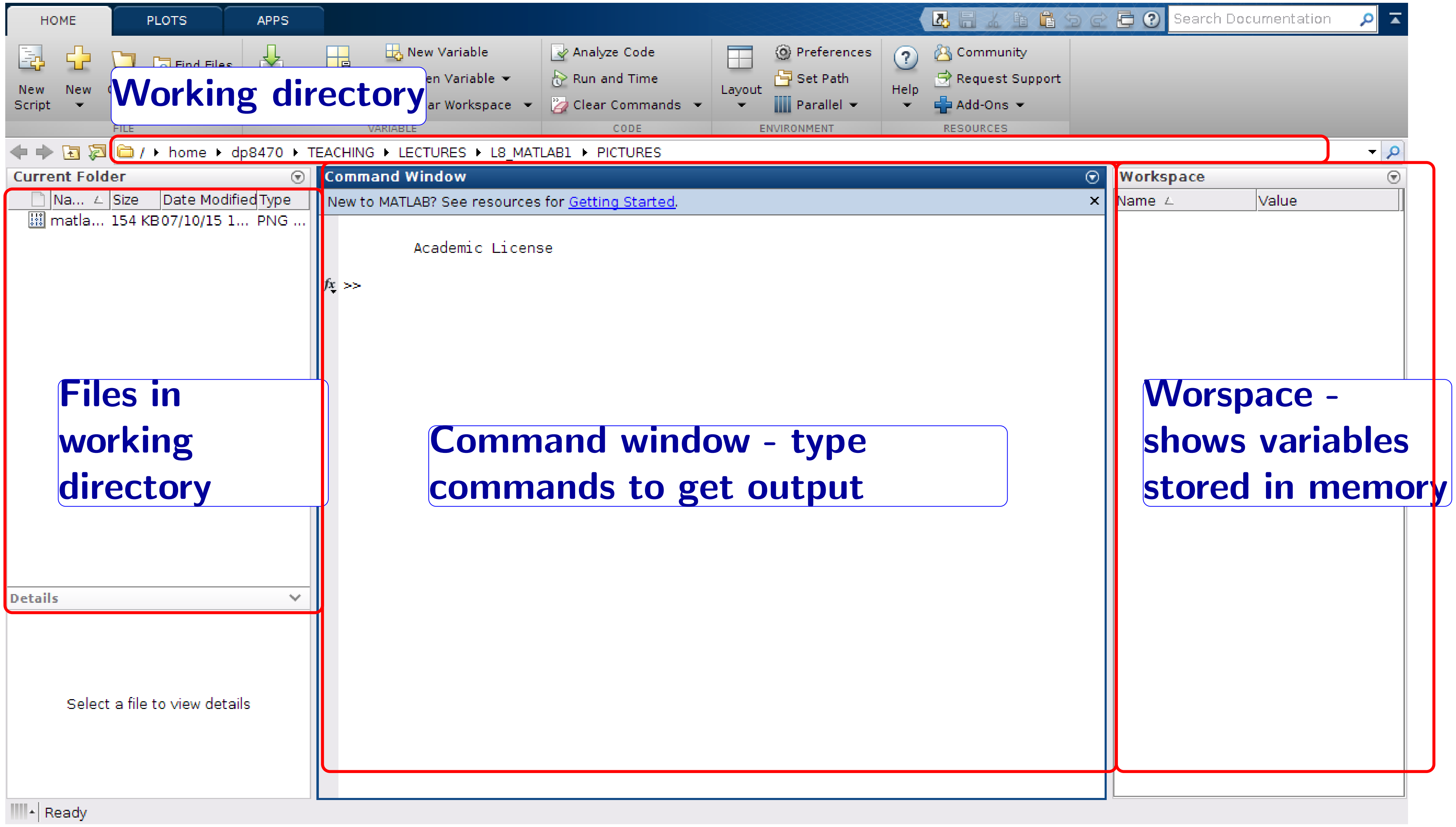
C

```
for (i=0; i<=Arows; i++){
  for (j=0; j<=Bcolumns; j++){
    C[i][j]=0.0;
    for (k=0; k<=Acols; k++){
      C[i][j]+=A[i][k]*B[k][j];
    }
  }
}
```

MATLAB

```
C = A*B;






.
```

MATLAB



```
        Academic License

>>
```

MATLAB commands

Commands can either be typed directly into the command window or written into a script. Anything written by the user in the command window has the command prompt symbol in front of it:

```
>> x=3.2;
>> y=x*(sqrt(x)-2)
y=-0.6757
```

- Similar syntax to C
- Do something to some numbers and store in a variable
- **SEMI-COLON** used to suppress output
  - Unlike C, in which a semi-colon is the end of a line
- MATLAB is case sensitive

MATLAB scripts

A script in MATLAB (similar to a C program) is simply a text file that contains some MATLAB commands. The extension of any MATLAB script is `.m` (e.g. filename.m). A script does not need any header or footer, but just contains some commands:

```
clear all
close all
% This is a comment
a=2;
% pi is a built-in number
b=tan(pi);
% output something by not having a semi-colon
c=a*b
```

- `clear all` clears all variables and `close all` closes any open files or open figures
- Comments are lines beginning with %
- To run, either use the run button or the command window

MATLAB variables and in-built functions

Variables

Variables and variable types are similar to those in C, except in MATLAB the types are decided by MATLAB and **are dynamic**:

```
>> a=1;
>> b=2;
>> c=a+b
c=3
```

```
>> a=1;
>> b=2;
>> c=a/b
c=0.5000
```

Functions

MATLAB also has a massive selection of in-built functions to chose from. Unlike in C where we had to load libraries, this is all done for you in MATLAB. Functions include mathematical operators (`sin`, `atan`, etc. etc.), matrix operations, plotting, surface plots, etc. etc. See the MATLAB pages for a full breakdown:

`http://uk.mathworks.com/help/matlab/`

Your new best friends - help and doc

A powerful command in MATLAB is the help command. In the command window, if you type

```
>> help <somefunction>
```

you will get the internal MATLAB documentation come up which will tell you how to use an in-built function e.g. for the plot command

```
>> help plot
 plot   Linear plot.
    plot(X,Y) plots vector Y versus vector X. If X
    or Y is a matrix, then the vector is plotted ...
```

For even more detail use the doc command in the command window:

```
>> doc <somefunction>
```

Vector and Matrix Definitions

To define a row vector we separate entries by spaces (or commas):

3	1	8

```
a = [3 1 8]
```

To define a column vector we separate entries by semi-colons (which means start a new row):

7.0
2.1

```
a = [7.0;2.1]
```

A matrix is then a number of rows separated by semi-colons:

7	1	8
2	0	5

```
a = [7 1 8; 2 0 5]
```

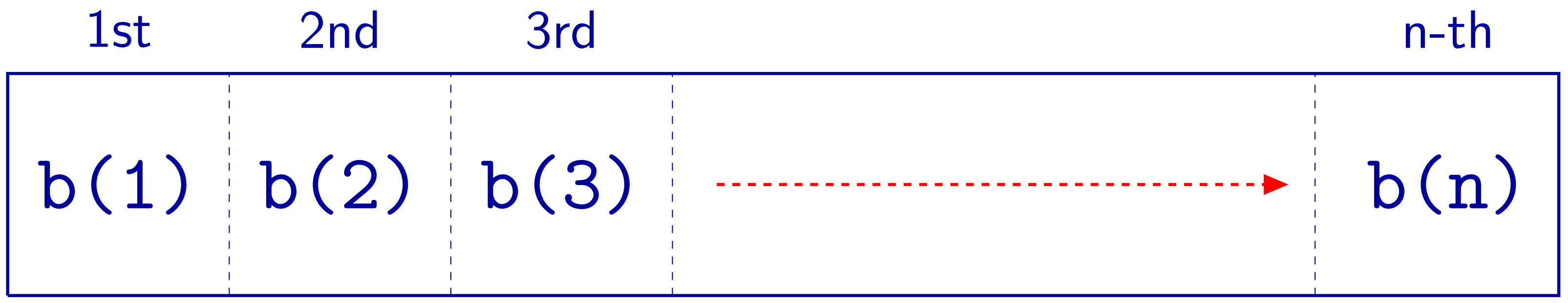
Vector and Matrix Definitions

To assign or access elements of a matrix (or vector), then we can do:

```
a(1,2)=1.2;
x=b(3);
```

which means store the value 1.2 into the **first row, second column** entry of the matrix `a` and store the **third** entry of the vector `b` into the variable `x`, which is a scalar.

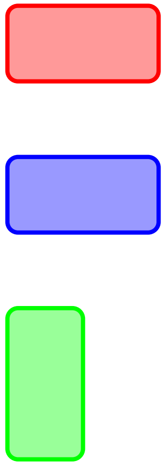
Array indexing in MATLAB starts from 1:



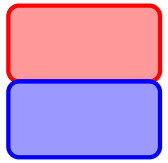
Matrix Concatenation

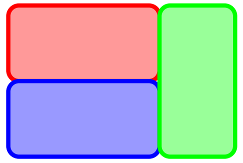
We can also concatenate (combine) matrices and vectors together. However, we must have dimensions that agree for it to work.

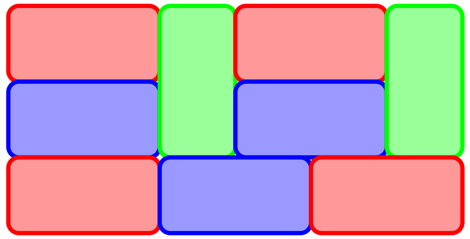
```
a  =  [1  2]


b  =  [4  5]


c  =  [7;8]
```



```
d  =  [a;b]



e  =  [d  c]



f  =  [[e  e];[a  b  a]]
```







Matrix Manipulation

MATLAB has a number of in-built functions to perform basic and complicated matrix manipulation. We will introduce the basic commands that you will need in the labs, in this lecture.

Matrix multiplication	`C = A*B;`
Transpose	`C = A';`
Inverse	`C = inv(A);`
Determinant	`x = det(A);`
Pseudo-inverse	`C = pinv(A);`

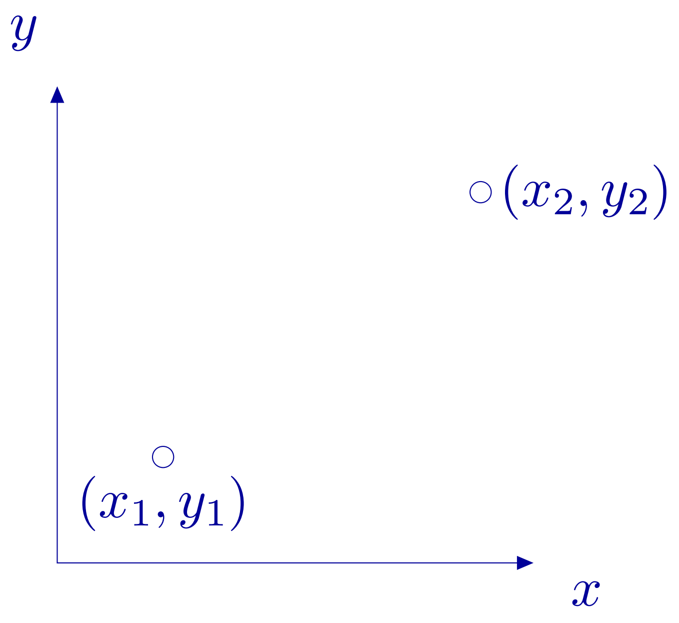
NOTE: to do matrix multiplication, the number of columns of `A` must equal the number of rows of `B`.

ASSIDE: Linear Algebra Refresher

You will have done the basics of linear algebra (solving linear equations using the matrix form) in Engineering Maths. However, we will just have a quick reminder of the important concepts here.

A linear set of equations are any set of simultaneous equations where the variables to be solved for are linear combinations of one another. An example of a set of linear equations is fitting a straight line between two points.

We want to fit a line that has the following equation:



$$y = ax + b$$

ASSIDE: Linear Algebra Refresher

From the two coordinate points we have two equations:

$$y_1 = ax_1 + b \quad \text{and} \quad y_2 = ax_2 + b$$

We can write this in matrix form:

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \end{bmatrix} \begin{pmatrix} a \\ b \end{pmatrix}$$
$$\mathbf{y} = \mathbf{X}\mathbf{a}$$

To find a and b we need to solve this system, which is done by inverting the matrix and pre-multiplying by the inverse:

$$\mathbf{a} = \mathbf{X}^{-1}\mathbf{y}$$

Remember that we can only invert the matrix if it is square (if the number of equations equals number of unknowns), as it is here.

ASSIDE: Linear Algebra Refresher

In MATLAB, we can create the $\mathbf{X}$ and $\mathbf{y}$ matrices:

```
X=[x1 1.0;x2 1.0];
Y=[y1;y2];
```

We can then invert the $\mathbf{X}$ matrix:

```
Xinv=inv(X);
```

And find the unknowns a and b for our straight line equation:

```
A=Xinv*Y;
a=A(1);
b=A(2);
```

ASSIDE: Linear Algebra Refresher

Now consider fitting a straight line through three points. In this case we have three equations (one for each coordinate) but only two unknowns (a and b).

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \end{bmatrix} \begin{pmatrix} a \\ b \end{pmatrix}$$

$$\mathbf{y} = \mathbf{X}\mathbf{a}$$

In this case, the matrix is not square. We can, however, still fit a line through our points, although this line will now no longer go through all of the points. We cannot invert our matrix exactly as it is not square, but we can perform a pseudo-inverse to get an approximation to the inverse of the matrix:

$$\mathbf{a} = \mathbf{X}^{\dagger}\mathbf{y}$$

ASSIDE: Linear Algebra Refresher

If we have three points that we want to fit a curve to, we could just fit a quadratic curve:

$$y = ax^2 + bx + c$$

Which has three unknowns (a, b and c). The system to solve is then:

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_2^2 & x_3 & 1 \end{bmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

$$\mathbf{y} = \mathbf{X}\mathbf{a}$$

$\mathbf{X}$ can then be inverted directly using `inv(X)`.

Elemental Matrix Manipulation

We can also perform operations on elements of matrices (note, matrices must be the same size in all dimensions for this to work). We can add (or subtract) together each element of two matrices:

```
C=A+B;
```

$$\begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} + \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix} = \begin{bmatrix} a_1 + b_1 & a_2 + b_2 \\ a_3 + b_3 & a_4 + b_4 \end{bmatrix}$$

We can also multiple (or divide) each element of two matrices using `.*` and `./`:

```
C=A.*B;
```

$$\begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} + \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix} = \begin{bmatrix} a_1 b_1 & a_2 b_2 \\ a_3 b_3 & a_4 b_4 \end{bmatrix}$$

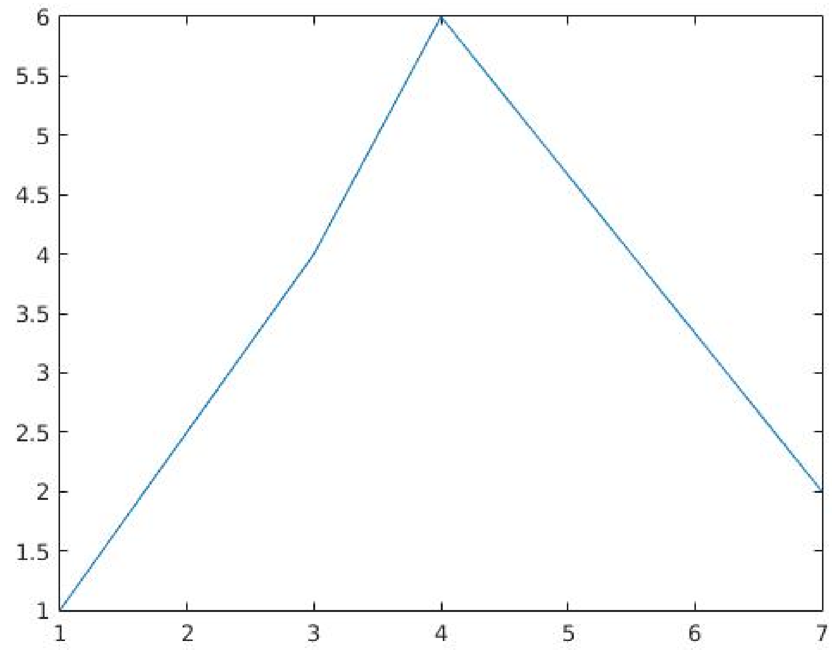
NOTE: matrix multiplication is not the same as multiplying each element of a matrix together.

Plotting

MATLAB has a powerful built-in plotting facility. Again, the `help` and `doc` commands are your best friends here. The library of changes available in the `plot` command are vast. The basics are outlined in these slides.

MATLAB allows the 2D plots of one column vector against another, although the column vectors **MUST HAVE THE SAME LENGTH**. The basic plot command puts dots at the coordinates specified and draws a straight line between the dots e.g.

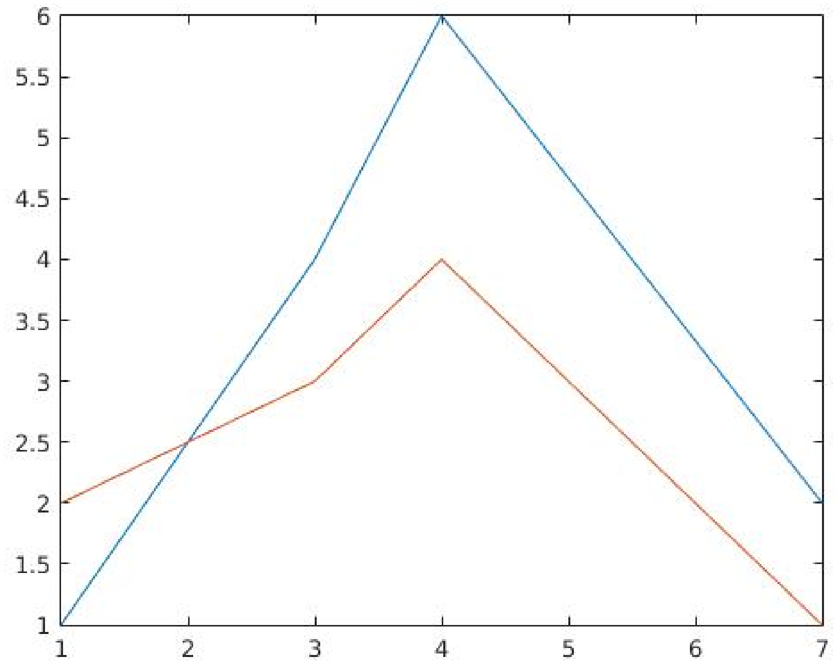
```
x=[1 3 4 7];
y=[1 4 6 2];
plot(x,y);
```



Plotting

To plot multiple lines on the same graph, plot each line and then use the `hold on` command:

```
x=[1 3 4 7];
y=[1 4 6 2];
z=[2 3 4 1];
plot(x,y);
hold on
plot(x,z);
```



Plotting

We can also change various aspects of the plot area and the line styles. To change the line style, add a string of characters to the plot command (see `>>help plot` for the options) e.g.

```
plot(x,y,'g');
plot(x,y,'r--');
plot(x,y,'bx--');
```

- Green line
- Red dashed line
- Blue dashed line with crosses at the points

Other common commands are:

`axis([XMIN XMAX YMIN YMAX])`	Scale axis
`title('text')`	Add the title `text` to the graph
`xlabel('text')`	Add the title `text` to the x axis
`legend('line1','line2',...)`	Add a legend for each of the lines
`grid`	Toggle grid lines on and off
`close all`	Close all opened figure windows

Plotting the Line Fit

Finally, let us revisit a few slides back where we derived a straight line fit using linear algebra (this could of course be a polynomial fit too). We have seen a way, in MATLAB, to get the unknowns (a and b), so we therefore know:

$$y = ax + b$$

Now we need to plot that line on a graph. To do this, we need a range of x against which to plot y. To create a vector with a range of values, we do:

```
xplot=(0:0.1:1)';
```

Which simply means, create a vector `xplot` that contains evenly spaces numbers from 0 to 1, with 0.1 spacing. We need to transpose the matrix (using ') to make it into a column vector for plotting. We then need to create the vector `y` and plot as normal:

```
yplot=a*xplot+b
plot(xplot, yplot)
```

Summary

- Have introduced using MATLAB
- Have considered how to manipulate matrices in MATLAB and plot data

LAB

- Curve fitting data