1 Starting Matlab

If you attended the first Matlab session then you should be able to start Matlab by typing the following in a command window.

```
h:\mybin\p
cd h:\ma2915
ma
```

This will change to your ma2915 directory and then start Matlab. To start Matlab you can alternatively type the following at the command prompt

```
l:\cc\matlab61\bin\win32\matlab.exe
```

Observe that your current working directory is displayed in a bar at the top of the Matlab window.

2 Remarks about working with Matlab

The normal mode of working is to create your Matlab instructions with the Matlab editor, to save what you have created as a file with a name which has a name ending in `.m` and then to test the instructions by typing the name of the m-file at the >> prompt. You should be used to working in this way from when you used Matlab in level one and and we suggested that you did this in the instructions for your first Matlab session for MA2915. If you work in this way for the assignment then you can prepare `lde.m` and `astest1.m` with the editor and when you have something ready to test then you type `astest1` at the >> prompt. Matlab then attempts to execute the set of instructions. If Matlab reports errors on certain lines or if the script is not creating the results that you are expecting then you can then go to the editor window and attempt to correct the mistakes, save the 'corrected' version and then type again `astest1` at the >> prompt. (If the last thing that you typed at the >> prompt was `astest1` then you will get this up just pressing the up-arrow key.) You can continually edit and test your function file `lde.m` and your script `astest1.m` until everything is working satisfactorily.

3 A few tasks to try

3.1 Vectorised operations

Type the following and note what happens.

```
x=linspace(0,0.5*pi,5)
y=sin(x)
whos x y
```

You should get row vectors of length 5.

Next try the following.

```
x=2:6
y=3:7
x.^2
z=x.*y
```

The `.^` and `.*` work entry-wise on the row vectors.

3.2 Working with matrices and vectors

Matlab has an extensive list of commands for working with vectors and matrices. Try the following which involves computing eigenvalues and eigenvectors.

```
A=[2 1;1 2]
eig(A)
[V D]=eig(A)
```

The columns of `V` are eigenvectors of `A` (normalised to have length 1) with the diagonal entries of `D` being the corresponding eigenvalues. You can verify this by typing

```
A*V
V*D
A*V-V*D
```

In the above if you only want the diagonal entries of `D` and not the entire matrix then you can type

```
d=diag(D)
```

The character \ is the Matlab operation for solving a system of linear equations. If you have a square matrix `A` and a column vector `b` with the same number of rows then `x=A\b` is the Matlab way of solving the system to get the solution of "`Ax=b`". Try the following

```
A=[3 1 1; 1 3 1; 1 1 3]
b=[8;10;12]
x=A\b
y=A*x
```

Here Matlab selects the "appropriate" routine to solve the linear equations. The statement `y=A*x` is the matrix multiplication of the matrix `A` and the column vector `x` giving a vector `y` which agrees with `b` to machine precision accuracy. You can also get the solution by typing `inv(A)*b` but this is not very efficient as it first works out the inverse matrix (which is not needed) and then forms the matrix vector product.

Standard matrices are available as built-in functions in Matlab. Try the following.

```
O=ones(3)
I=eye(3)
Z=zeros(3)
```

From smaller matrices you can create larger matrices block by block. Try the following

```
A=[I O;-O I]
eig(A)
```

This generates a 6×6 non-symmetric matrix which has some complex eigenvalues.

In addition to being able to build up matrices block by block you can extract parts of matrix using the : operation. To illustrate this try the following.

```
b=1:5
A=[b;2*b;4*b;8*b;16*b]
c2=A(:,2)
r3=A(3,:)
mid=A(2:4,2:4)
```

Note that `c2` is the 2nd column of `A`, `r3` is the 3rd row of `A` and `mid` is the 3×3 matrix obtained from the entries in row positions 2, 3 and 4 and column positions 2, 3 and 4.