Part one

Foundations

1
Introduction

1.1 First steps with MATLAB

If you haven't already done so, you should start MATLAB by *double-clicking* the relevant *icon* with your *mouse* or by asking a friend sitting next to you to show you how. Asking a friend is often the quickest way to obtain help and, in what follows, we will encourage you to take this route when all else fails. If 'clicking' and 'icons' mean nothing to you, you may need some extra help in getting started with Windows. It might also be that your system doesn't use *Microsoft Windows* and that simply typing `matlab` will do the trick. For example, if you are using a *Unix* system of some kind this may be the case.† If all goes well you will see a MATLAB prompt

```
>>
```

inviting you to initiate a calculation. In what follows, any line beginning with `>>` indicates `typed input` to MATLAB. You are expected to type what follows but not the `>>` prompt itself. MATLAB supplies that automatically.

1.1.1 Arithmetic with MATLAB

MATLAB understands the basic arithmetic operations: add is `+`, subtract is `-`, multiply is `*` and divide is `/`. Powers are indicated with `^`, thus typing

```
>> 5*5+12^2
```

results in

† We will make some further remarks, where appropriate, about editing and file handling in a non-Windows environment.

```
ans =

    169
```

If you typed the above line and nothing happened, perhaps you omitted to press the <Enter> key at the end of the line. The laws of precedence are built in but, if in doubt, you should put in the brackets. For example

```
>> 8*(1/(5-3)-1/(5+3))

ans =

    3
```

The sort of elementary functions familiar on hand calculators are also available. Try

```
>> sqrt(5^2+12^2)
```

and

```
>> exp(log(1.7))
```

What do think `sin(pi/2)` will produce? Try it.

In fact MATLAB has the value of $\pi = 3.1415926\ldots$ built in. Simply type in `pi` whenever you need it. Try the following:

```
>> pi
>> format long
>> pi
>> format short
```

MATLAB retains considerably more significant figures of accuracy than suggested by the default setting which is given by `format short`.

1.1.2 Using variables

You can assign numerical values to '*variables*' for use in subsequent calculations. Typing

```
>> x=3
```

produces

```
x =

     3
```

or you might want something more useful like

```
>> rad=2; ht=3;
>> vol=pi*ht*rad^2

vol =

       75.3982
```

Note that the first line had two 'commands' on it, neither of which seemed to produce a result! When MATLAB encounters an instruction followed by a semi-colon ; it suppresses any visual confirmation. It really does obey the instruction but keeps quiet, as you can check with

```
>> rad=4;
>> rad

rad =

        4
```

This is useful if you want to avoid cluttering up the screen with intermediate results. Watch out for semi-colons in what follows.

Remember that each variable must somehow be assigned a value before you can make use of it in further calculations. For example if you have followed the above examples and now type

```
>> f = x^2 + 2*x*y + y^2
```

you should get a result something like

```
??? Undefined function or variable y
```

This is self-explanatory. If you now type `y=4;` and then repeat the calculation of `f` you should have more success.

Incidentally, a quick way of repeating a previous MATLAB instruction is to press the 'up-arrow' key (↑) until you recover the command you want. Try it now. If the original instruction was not quite correct, or if you want to develop a new instruction from a complex but similar previous one, you can use the same trick. Recover the command which

you want then use the sideways arrows (← and →) together with the delete key to edit the old command suitably. As an exercise, try using your previous work to calculate the volume of a right-circular cylinder with radius 2 and height 1/4. Did you get π?

If you have difficulty remembering the names of variables which you have assigned, you can try typing `who` or `whos`. Try them both now. Do you recognise the variables listed?

1.2 Vectors and plots

One of the pleasures in learning to use MATLAB is discovering the simplicity of plotting things. The basic principle is:

(i) select a sequence of x-values that is, a vector of values;
(ii) evaluate $y = f(x)$, that is, obtain a corresponding vector of y-values
(iii) plot y vs x.

Before doing this, it is worth spending a moment learning something about how MATLAB deals with vectors.

1.2.1 Vectors

Type in the following examples which all result in vector-valued variables. Pause to think about the result in each case.

```
>> u=[2,2,3]
>> u=[2 2 3]
>> v=[1,0,-1]
>> w=u-2*v
>> range=1:13
>> odd=1:2:13
>> down=20:-0.5:0
>> even=odd+1
>> xgrid=0:.05:1; x=xgrid*pi
>> y=sin(x)
```

The first two lines demonstrate that elements of a vector can be separated by spaces or commas. If you are worried about inserting blank spaces by accident you can stick to the comma notation. Thus `[1+1 2 3]` is the same as `[2,2,3]`, whereas `[1 +1 2 3]` is the same as `[1,1,2,3]`!

Note that vectors can be of any length. They can be row vectors as here, or column vectors like

```
>> w'
ans =

     0
     2
     5
```

where the apostrophe denotes transpose ($\top$). In MATLAB, vectors are treated simply as a special case of matrices which you will learn much more about in the next chapter.

Notice what happens when the displayed vector is too long to fit on a line. MATLAB just displays as many elements as it can and then puts the rest on the following lines. The elements in a row vector are treated as 'columns'.

An elementary function of a vector `x`, such as `sin(x)`, is also a vector of the same kind. We can use this fact to create plots of functions as shown in the next section.

MATLAB knows how to multiply matrices of compatible size. This will be discussed in greater detail in the next chapter. For now, try typing

```
>> w*w'
>> w'*w
>> u*w'
>> u*u
```

Can you make sense of the results? Why did the last one not work?

Now suppose you want a set of values `z` given by $z = y^2$, where `y` is the vector of values already assigned. From the above experiments you will realise that

```
>> z=y*y
```

is not understood by MATLAB.

```
>> z=y*y'
```

is understood but is evaluated as the scalar product $\mathbf{y} \cdot \mathbf{y}$! What you need to do, to force MATLAB to multiply things *element-by-element*, is to type

```
>> z=y.*y
```

where the `.` inserted before the `*` symbol is the key feature forcing element-by-element operation. Similarly, `u./v` and `y.^2` are understood as element-by-element operations with vectors of the same size.

1.2.2 Plotting things

Type `whos` at this point to verify that you have vectors `x` and `y` defined as above. They should both be 1×21 matrices (that is, row vectors).

Plotting is easy. Just type

```
>> plot(x,y)
```

and sit back. A nice simple graph of $y = \sin x$ vs x will magically appear. The axes are chosen automatically to suit the range of variables used. This is the simplest possible case. In later work you will want to do more elaborate things. For now, try the following:

```
>> title('Graph of y=sin(x)')
>> xlabel('x')
>> ylabel('y')
>> y1=2*x;
>> hold on
>> plot(x,y1,'r')
```

You can probably figure out the significance of each of these commands. For example, `y1=2*x` defined new function values $y = 2x$, `hold on` told MATLAB to keep the same graph and `plot(x,y1,'r')` plotted the next graph on top. Note that the axes were adjusted† and the second curve was plotted in red.

In the above examples of `plot`, MATLAB joined up the 21 points with straight-line segments. Should you not want this, you can specify plotting points using a choice of symbols as follows. Try this

```
>> hold off
>> plot(x,y,'+')
>> plot(x,y,'g*')
>> plot(x,y,'w.')
```

† Assuming you are using MATLAB version 4. There are a number of small differences between this and earlier versions, particularly with graphics commands.

Did MATLAB do what you expected? Did you remember to use the ↑ key to reuse previous commands? If you need some help with how to use any MATLAB instruction you can type for example

```
>> help plot
>> help hold
>> help sin
```

and so on.

1.3 Creating and editing script files

Once you get going, you may find it tiresome to keep reentering the same, or similar, sequences of commands. Fortunately, there is a simple way round this: you simply store any frequently used sequence of commands in a file called a '*script*' or '*M-file*'. You can then invoke this list of commands as often as needed.

For example, in a particular session you might want to find the distance between two points A and B whose position vectors are given by $\mathbf{a} = (1, 0, -2)$ and $\mathbf{b} = (2, 3, 1)$ respectively. Knowing that the vector displacement between them is

$$\mathbf{d} = \mathbf{b} - \mathbf{a}$$

and that

$$|d|^2 = \mathbf{d} \cdot \mathbf{d}\,,$$

you might use the following sequence of MATLAB instructions:

```
>> a=[1,0,-2];
>> b=[2,3,1];
>> d=b-a;
>> dd=d*d';
>> dist=sqrt(dd)
```

to solve that particular problem. This is fine, but suppose you have a set of five points and want to check which pair is the closest together? You would obviously want to store as many as possible of these steps in a 'script' (file) to reuse as required.

1.3.1 Editing and saving a text file

We first need to review file-handling and editing.

Non-Windows users

If you are *not* using Microsoft Windows, you will at this point need to make some slight alteration to procedures. However, no matter how MATLAB has been installed on your system, there will undoubtedly be a *text editor* of some sort. Assuming it is called `edit`, the easiest way to invoke it from MATLAB is probably to type

```
>> !edit fname
```

where fname is the name of a text file which either exists or will exist, by the time you have finished! If that doesn't work, consult someone knowledgeable about your system setup or ask a patient friend.

Windows users

Windows comes with its own basic text file editor called *Notepad*, whose icon is usually found within the Accessories Group. A typical MATLAB setup within Windows makes direct use of this accessory so we will concentrate on this method. To create and edit a new file called `myfile.m`, *from within MATLAB*, proceed as follows:

(i) In the `MATLAB Command Window` menu, click on `File`.
(ii) Click `New` then `M-file`.
(iii) Within `Notepad` which you have now started, you can type some lines, for example

```
% myfile.m
% It doesn't do very much, just identifies itself.
% These 3 lines are comment lines which MATLAB ignores.
disp(' I am an M-file')
```

(iv) Click `File` then `Save As`.
(v) Within the box `File Name` which is open waiting, type in `myfile.m`.
(vi) Click `OK`.

You have now created a file which MATLAB can find and use. You hope!

Back in the `MATLAB Command Window` you can now ask MATLAB whether it can find the file. Type

```
>> type myfile
```

and you should see a list of the lines which you typed in. If not, go back to step (i) and start Notepad again by clicking `File` in the `MATLAB Command Window` followed this time by `Open M-file`. You should see an