Chapter 1

Performing calculations in MATLAB

In this chapter we'll learn how to perform calculations with MATLAB. First we'll learn how to use it as a sort of advanced calculator. Next we'll see how to use it to perform some linear algebra calculations. Then we'll take a look at how to use it for some simple stochastic simulations. We'll be able to do quite a lot with only a very little MATLAB knowledge.

Tip: Obtaining MATLAB

While you are a student at King's you can obtain MATLAB using the following link https://internal.kcl.ac.uk/it/software/matlab.aspx.

1.1 MATLAB's user interface

When you first open MATLAB the screen should look something like the schematic diagram shown in Figure 1.1:

The main part of the screen to focus on is the so-called **Command Window**. This is where you type commands that you want MATLAB to act on immediately.

For example you can enter the command:

```
a = 3
```

and MATLAB will create a variable called `a` and assign it the value 3. It will also print out a message to indicate what it has done. In this case it simply reports that `a = 3`.

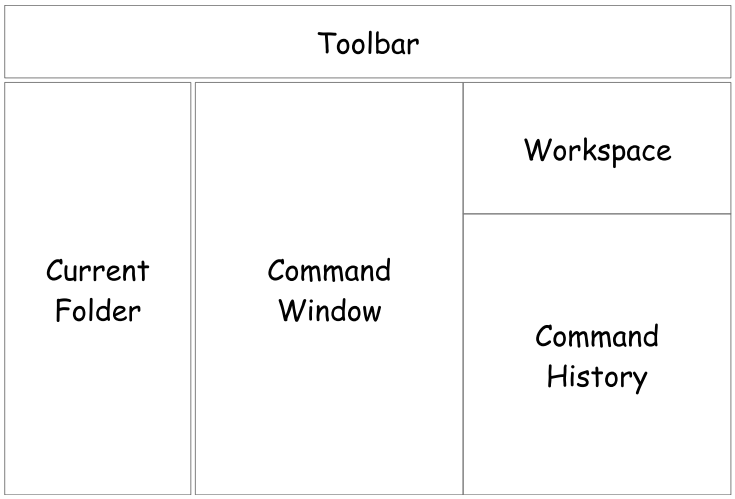

Figure 1.1: The layout of the MATLAB User Interface

Similarly we can create a variable called `b` and assign it the value 2 by entering:

```
b=2
```

If you now enter the command:

```
a + b
```

Matlab should report that the answer is 5.

If you now look over in the **Workspace** window you will see all the variables that you have created. These are `a`, `b` and a temporary variable called `ans` which is used to store the result of the last computation.

If you look over at the **Command History** window, you will see a record of all the commands that you have entered. You can use the command history to easily rerun commands.

An alternative method is to press the up and down arrows in the **Command Window**. Try doing this now to run the calculation of $a + b$ a second time.

You can change the values of any variable if you like for example by entering

```
a = b + 25
```

MATLAB even understands the statement

```
a = a + 1
```

This is how you say "add one to the variable a" in the MATLAB language.

Naturally you are not restricted to adding numbers. You can use `*` to multiply numbers, `/` to divide them, `-` to substract them, `^` to raise one to the power of another. In addition you can use brackets to group terms together.

You can also use the standard functions that you might expect such as `sin`, `cos`, `sqrt`, `exp` `log`. You should use brackets when you call a function. Here's an example:

```
sin( 360 )
ans =

    0.9589
```

1.2 Exercises

Experiment with the MATLAB command window to answer the following questions

1) What is the cube root of 2?

2) Does `sin` use degrees or radian's?

3) What base is used for logarithms using the `log` command?

4) What happens if you forget the brackets and type `log 1`?

5) What happens if you type `a+1=a` instead of `a=a+1`?

6) Use the up arrow to run the command `a=a+1` repeatedly. Check that it is doing what you expect.

7) Work out how to compute 10 factorial using the help if you can't guess.

1.3 Matrices in MATLAB

1.3.1 Matrix basics

The name MATLAB actually stands for "Matrix laboratory" and not "Maths laboratory". So you might expect it makes working with matrices very easy.

To store the 3×3 matrix:

$$\begin{pmatrix} 2 & 4 & 5 \\ -3 & 1 & 7 \\ 4 & 9 & 2 \end{pmatrix}$$

in a variable called `a` you use the following command:

```
a = [ 2 4 5 ; -3 1 7 ; 4 9 2 ]
```

Note that spaces are used to separate columns and semi-colons are used to separate the rows. If you aren't sure what a semi-colon is, see table 1.1

So we can create two 1×3 row vectors `v1` and `v2` as follows:

```
v1 = [ 1 2 3 ]
v2 = [ 4 5 6 ]
```

and a 3×1 column vectors `w1` and `w2` as follows:

```
w1 = [1; 2; 3 ]
w2 = [4; 5; 6 ]
```

Performing matrix operations is just as easy as performing normal calculations for example to multiply `a` and `w1` just enter

```
a * w1
```

or to add `w1` and `w2` just enter

```
w1 + w2
```

The operators `-` and `^` work as you expect too. For example you can use `^2` to square a matrix.

What about division? Well division of matrices isn't a standard mathematical operation, so you can't simply divide matrices. However, to invert a matrix you can use the function `inv` (raising it to the power -1 works too, but may be slower and less accurate since the function `inv` will have been optimized for computing inverses).

MATLAB uses the division operator \ for solving matrix equations - which is the matrix analog of division. But since matrix multiplication is non-commutative MATLAB makes a distinction between between dividing matrices on the right and dividing on the left. For example, suppose we want to solve the matrix

equation $Av = w$ where v and w are vectors and A is a matrix. Then the solution is $v = A^{-1}w$, which requires us "to divide w by A on the left". In MATLAB one can write `v = A \ w` to mean "let v be the solution of $Av = w$. If you wanted to solve the equation $vA = w$ where v and w are row vectors, you would write `v = w / A` but this is less common.

Another common function with matrices is to compute the *conjugate transpose*, also known as the adjoint of the matrix. For the real valued matrices we'll use in this course, the conjugate transpose is the same thing as the transpose. To compute the conjugate transpose in MATLAB you use the apostrophe symbol `'`. For example, here is how you compute the transpose of `v1`

```
v1'
```

It is quite normal to create column vectors by creating a row vector and then taking its transpose. Here is how we could have initialized the vectors `w1` and `w2`:

```
w1 = [1 2 3]'
w2 = [4 5 6]'
```

1.3.2 Creating matrices

As you have probably already found, entering a matrix by hand is quite a boring task. So MATLAB provides several functions to quickly create matrices with given values. Here are some functions that we will use a lot:

- `zeros` Create a matrix which consists only of zeros.
- `ones` Create a matrix consisting only of ones.
- `rand` Create a matrix containing random numbers uniformly distributed.
- `randn` Create a matrix containing random numbers which are normally distributed.

You use all of these functions in similar ways. To 4×6 matrix of zeros you would type:

```
zeros(4,6)
```

To create a 3×5 matrix of random numbers between 0 and 1 you would type:

```
rand(3,5)
```

Table 1.1: Some English words for punctuation marks

In computer programs punctuation is very important. Many non-native English speakers are unfamiliar with the names for the different punctuation marks used in programming. For their benefit, here is a guide:

Symbol	Term
.	Full stop or just "dot".
,	Comma.
:	Colon.
;	Semi-colon.
'	Apostrophe or single quote.
"	Double quote
_	Underscore.
()	Brackets, also called round brackets or parentheses.
[]	Square brackets.
{}	Curly brackets.
<>	Angle brackets.
~	Tilde or twiddle.
&	Ampersand or and sign.
|	Pipe or vertical line

As a shortcut, MATLAB allows you to pass only one parameter if you want to create square matrices. For example:

```
zeros(4)
```

creates a 4×4 square matrix of zeros.

Two other useful functions for creating matrices are

- `eye`. This creates an identity matrix. This will always be a square matrix, so you only pass in one parameter. For example `eye(5)` creates the 5×5 matrix. Notice that the name of this function is a pun. The word eye is pronounced the same as the letter I.
- `diag`. This creates a diagonal matrix. E.g. `diag([2 4 7])` creates a 3×3 diagonal matrix with the numbers 2, 4 and 7 on the diagonal. Note that `diag` takes a vector parameter.

Creating evenly spaced vectors is also a very common task. One way to do this is to use the special colon operator `:`.

To create a horizontal vector containing all the whole numbers between 1 and 100 one enters:

```
1:100
```

For the whole numbers between 20 and 50 one enters:

```
20:50
```

You can also specify the step size. For example, for the numbers between 20 and 50 going up in steps of 3 type

```
20:3:50
```

The general syntax is

```
<start value>:<step size>:<end value>
```

Note that the above box doesn't contain real code, you are meant to replace the text in angle brackets with your choice of variables and numbers. We'll use this notation without comment from now on.

One other way to create evenly spaced vectors is to use the function `linspace`. For example `linspace(30,70,10)` creates a vector containing 10 elements evenly spaced between 30 and 70. Whether you use the colon operator or `linspace` simply depends on whether you want to specify the number of steps or the step size.

1.3.3 Dot operators

Suppose that `dollarPrices` contains the stock price (in dollars) for the ACME corporation for every day in the last week and that `r` contains the USD to GBP exchange rate for every day in the last week:

```
dollarPrices = [ 100 105 103 102 103 ]
gbpToUsdRates = [ 0.61 0.62 0.63 0.62 0.61]
```

then to compute the stock price for ACME in GBP for every day in the last week, we want to multiply together the vectors s and r cell by cell. We do this using the operator `.*`. That is a star with a dot in front.

```
gbpPrices = dollarPrices .* gbpToUsdRates
```

So `.*` multiplies vectors in a very different (and simpler) way than standard matrix multiplication. In particular you can use `.*` to multiply an $m \times n$ matrix with another $m \times n$ matrix whereas you can use `.*` to multiply a $m \times a$ matrix with an $a \times n$ matrix.

There are dotted versions of the operators `^\lstinline` and `/` too for when you want to raise every cell of a matrix to a given power or when you want to divide matrices cell by cell. There is no need for dotted versions of $+$ and $-$ because $+$ and $-$ already add and subtract cell by cell.

It is always easy to work out if you want to use a dot operator or a standard matrix operator — apart from anything else you will usually get an error caused by the matrices being the wrong size if you attempt to use the wrong type of operator. Generally speaking you use normal matrix multiplication if you are thinking of your matrices as linear transformations and your vectors as points in a vector space. If, as in the stock example above, the vectors just contain data without any geometric interpretation, you will usually use dot operators.

Notice that functions such as `exp` and `sin` work on matrices, but they behave like the dotted operators. In otherwords they perform `exp` and `sin` component-wise. This is standard in MATLAB, whenever it makes sense, functions can take matrices as arguments and they perform their standard operation component by component.

Tip: Choosing variable names

At the start of this tutorial we used short variable names like `a` and `b`. We've now started using longer names like `gbpToUsdRates`.

Variable names in MATLAB should not contain spaces and should not begin with numbers but otherwise they can be a mix of numbers and letters. MATLAB is case sensitive, so the variables `a` and `A` are different. The fact that you can't use spaces isn't much of a problem, just use camel case (starting new words with a capital letter). Its called camel case because the words have humps like camels.

In mathematics it is conventional to use single letter variable names — to the extent that mathematicians borrow letters from Greek and Hebrew to make their formulae shorter. In computing longer names are preferred. The big advantage of longer names is that it is much easier to understand what is going on when you look at code with long variable names. The code: `gbpToUsdRates = dollarPrices .* gbpToUsdRates` is pretty self explanatory. I strongly recommend using long variable names.

1.3.4 Statistical functions

When you think of a matrix as being a matrix of data rather than a linear transformation, there are various statistical functions you might want to apply to the matrix: `sum`, `mean`, `std`, `median`, `prctile` all work in a similar way. For

example `sum(v)` computes the sum of the elements in a vector or the sum of the columns of a matrix.

The function `std` computes the standard deviation of a vector. By default it computes the sample standard deviation, but you can get it to compute the population standard devation if you want - see the online help.

The function `prctile` computes a given percentile of a vector of data.

Another very useful function is `size` which tells you the dimensions of the vector or array and `length` which tells you the largest dimension.

The function `hist` plots a histogram of a vector of data. For example, recall that the function `randn` generates normally distributed random numbers. So the following code will plot a histogram of a random sample of ten thousand numbers from the normal distribution:

```
sample = randn(10000, 1)
hist( sample )
```

There are two things you probably won't like about this - firstly the histogram doesn't contain very many bars so it doesn't look as normally distributed as you might like and secondly it isn't very helpful to print out the array of ten thousand numbers. Here is how you fix those issues:

```
sample = randn(10000, 1);
hist( sample, 100 )
```

We have added a semi-colon on the end of the first line. It means "don't print the result".

On the second line we have passed in an extra parameter of 100, this is the number of bars to show in the histogram.

1.4 Putting it all together

We can combine all of these ideas to perform some quite sophisticated calculations.

Example 1: Use MATLAB to compute the sum

$$1 + 2 + 2^2 + 2^3 + 2^4 + \ldots + 2^{10}$$

Solution: We first create a vector of the numbers 0 through 10. We then use the `.^` operator to compute the associated powers of 2 and store the result in powers. We then compute the sum.

```
x = 0:10;
powers = 2.^x;
sum( powers )
```

You could do this all in one line if you wanted: `sum( 2.^(0:10) )` but it is usually easier to understand code that has been broken into small pieces.

Example 2: A robot walks a distance X_1 east, a distance X_2 south and then climbs a distance X_3 up. The X_i are independent and normally distributed with mean 1. Negative distances should be interpreted in the obvious way. Using a MATLAB simulation, plot an approximate histogram of the total distance travelled.

Solution: We create a sample of 1000 possibilities.

```
X1 = randn(1000,1);
X2 = randn(1000,1);
X3 = randn(1000,1);
distance = sqrt( X1.^2 + X2.^2 + X3.^2 );
hist( distance, 20 );
```

1.5 Exercises

Use MATLAB to answer all these questions.

1) What is $\left(\frac{1}{\sqrt{2}}(1+i)\right)^4$?

2) What is the 95-th percentile of the normal distribution (with mean 0 and standard deviation 1)? Answer this question approximately by creating a large sample of normally distributed random numbers and then finding the 95th percentile.

3) How would you create a vector containing the first 50 odd integers in MATLAB? What is the sum of the first 50 odd integers?

4) How would you create a vector of the cubes of the first 50 odd integers in MATLAB?

5) What is the sum of the cubes of the first 50 odd integers?

6) Use the matrix inverse function `inv` to solve the equations:

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 5 \\ -2x_1 + 3x_2 + 4x_3 &= 6 \\ 1x_1 + 3x_2 + 2x_3 &= 7 \end{aligned}$$

Solve the same equations using the \ and / operators. MATLAB will use Gaussian elimination if you use the division operators, but will compute the matrix inverse if you use `inv`. These are two quite different algorithms for solving linear equations. Which is more efficient?

7) Recall that $\pi = 4(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \ldots)$. Compute π to three decimal places.

8) Create a sample of ten thousand numbers selected from a normal distribution with mean 10 and standard deviation 20. Plot a histogram to make sure it looks right. Also check your answer using the `mean` and `std` functions.

9) Use the documentation to find out how to use the function `randi`. Suppose that 100 dice are thrown and the numbers on the dice are added. Use randi to simulate throwing all 100 dice 10000 times and plot a histogram of the sum. What do you expect the histogram should look like and why?

1.6 Further Reading

The first three chapters of these notes are all about using MATLAB and cover everything needed for the course.

If you are struggling with MATLAB you could try MATLAB demystified [2].

MATLAB have many online resources you can use to learn MATLAB and develop your knowledge further [1].

1.7 Summary

We have learned how to use MATLAB as a sophisticated calculator.

We have learned how to store intermediate steps of a calculation in variables.

We have learned how to create matrices by entering them in full or by using the functions `zeros`, `ones`, `eye`, `randn` and `diag`.

We have learned how to perform operations on matrices. We can perform standard matrix operations such as multiplication with `*`. We can perform elementwise operations with operators such as `.*`.

We have learned how to compute statistics for a vector of data using functions such as `std`, `mean`, `sum`, `length` and `hist`.

Bibliography

[1] MATLAB. Matlab tutorials. `https://tinyurl.com/y74vxd8g`.

[2] David McMahon. *MATLAB demystified*. McGraw-Hill New York, 2007.