FACULTY OF SCIENCE

SCHOOL OF MATHEMATICS AND STATISTICS

INTRODUCTION TO MATLAB

2019

Contents

Chapter 1

You can also run Matlab on a mobile device using UNSW MyAccess service from https://aaa-access.unsw.edu.au/vpn/index.html .

Further information about Matlab is contained in chapter 2, will be given in lectures, or will be in the reference books [4, 1]. The book [3] by Cleve Moler, one of the creators of Matlab is available online, has an introduction to Matlab and also to numerical methods. Matlab has an extensive online help system.

In this chapter we will concentrate on the features of Matlab common to both Linux and Windows and also on how to create the les needed for your course.

1.2 The Matlab Window

To start a Matlab **session** (i.e. open a Matlab window), click on the Matlab Application Icon:

Figure 1.1: Matlab icon

After some time, a Matlab window, similar to that shown in gure 1.2, will appear.

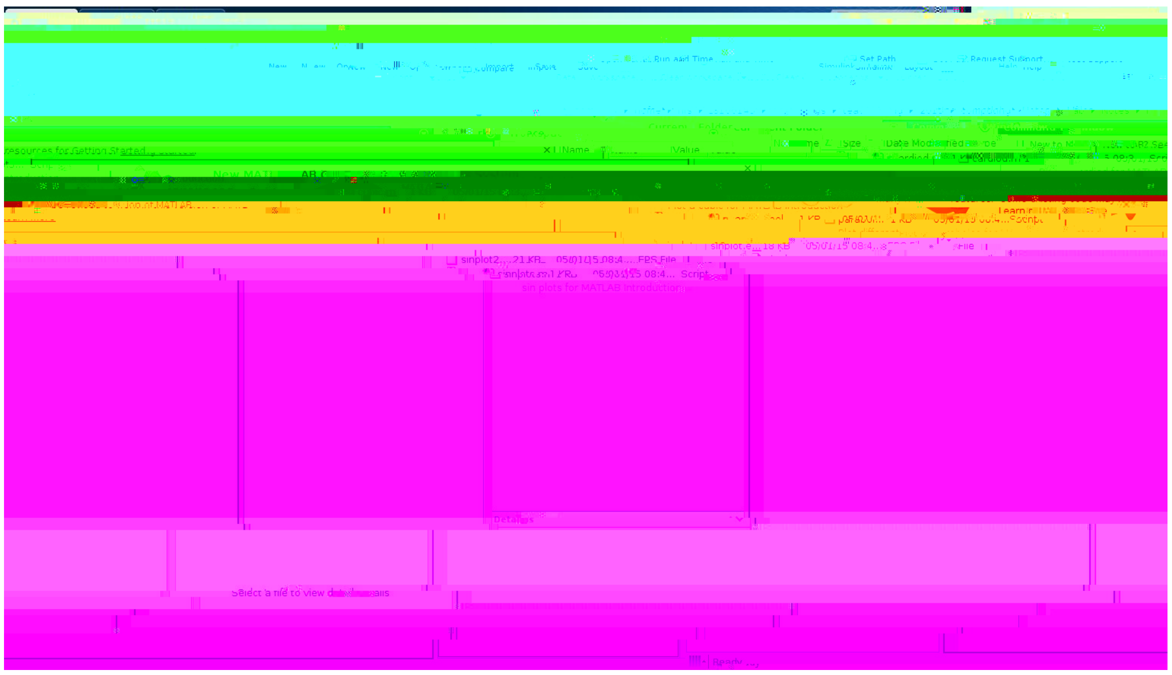


Figure 1.2: Initial Matlab windows

Figure 1.2 shows the Matlab windows when you start Matlab for the rst time. This window contains a tabbed set of menus across the top and a Search Documentation box in the top right hand corner. Most of the time you will be using the HOME tab. Immediately below the menu tab is a list of the folders you are in, ending in Matlab 's current folder. It is good practice to create folders for your di erent courses, all under your UNSW home drive (not on the local computer) so you can access them from any computer.

The rst time you start Matlab the main window will be split into several sub-windows, including the Current Folder , Command Window, Workspace and Command History.

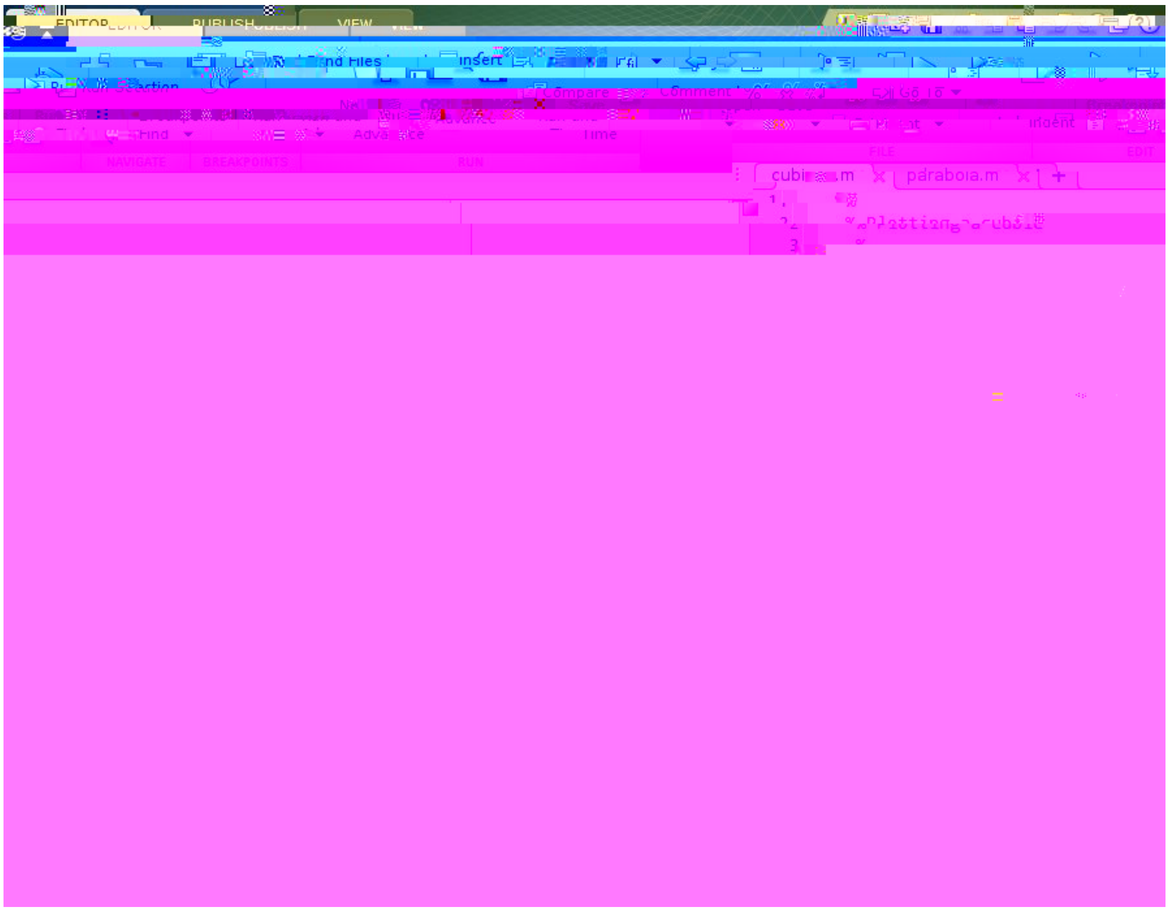
- The Current Folder sub-window lists all the les in the the current folder.
- The Command Window is where you type commands and see the results of commands that Matlab has executed. The Command Window has the Matlab prompt >>, indicating Matlab

Indenting: easily indent your Matlab code to re ect the program structure.

Debugging: breakpoints may be set on any executable line of a M- le and the values of any quantities inspected and manipulated.

Profiling: collect information on the amount of CPU time taken by functions and individual lines of code.

To start the Matlab Editor, click the **New Script** icon (this is the rst of the icons in the icon bar in Figure 1.2) or type the command edit in the command window. A new window appears: the Matlab Editor. Alternatively you can use the **Open** icon to open an existing le, or type the command edit cubic.m . If the le cubic.m is not in Matlab 's current folder then Matlab will ask you if you want to create the le. Figure 1.3 shows a Matlab Editor window, with a script le called cubic.m that will



```
% Plotting a cubic
```

Figure 1.3: A Matlab Editor Window

draw a cubic when executed. The Matlab editor window has its own menus and tabs with icons. These behave in roughly the same way as the corresponding icons in, for example, Microsoft Word. Note that clicking on the rst of the icons will open up a new **bu er** (editing space), so it is possible to have several M- les being edited at the same time. Just under the tabs with the various icons is a list of **tabs** with the names of the M- les being edited: click on the appropriate name to change to that le. You may see that there is a second le called parabola.m being edited in gure 1.3.

Before you can run an M- le, that is sequentially execute all the Matlab commands in the le, you must save the le ad the le must be in Matlab 's current folder. Check the les is listed in the Current Folder sub-window (see Figure 1.2), or type the command what in the command window. A very common mistake is to edit and save a le which is not in Matlab 's current folder, so it cannot be run.

Note that the lines beginning with a % sign are **comments**. Matlab ignores everything on a line that comes after a % sign.

When saving a script le, you will have to give the le a suitable name; with function les the editor will ll in the name for you | it will always be the name of the function (with the .m le extension added).

To prepare a Matlab M- le using a di erent editor, you work out what commands you want to use and simply use the editor to create a text le containing these commands (and suitable comments).

Don't forget to check that

irrelevant output is suppressed by ending appropriate commands with a semicolon

your le works

To check that yourscript le works enter the Matlab command clear to clear all variables in the current workspace (see section 2.1.5 for more information on the clear command). Then enter the name of the le (without the .m) and the le should run.

To check that afunction le works just use the function as if it were a standard Matlab function (without any .m t(w)cc7(our)7(050and)-328(ha)27(v)28(e)-327(a)-3g0(cr7(a)-3deal7

1.5 The Matlab interface

The Matlab tabbed interface, both for the main Matlab window and the Matlab editor has a large number of icons and associated menus. This tabbed interface was introduced in Matlab R2012b, so older versions have a (very) di erent graphical user interface (GUI). The underlying Matlab commands are still the same.

A context sensitive set of menus (that is it changes depending on which window/sub-window is active) can be obtained by right-clicking. This is often the most e cient way of accomplishing a task. For example if you have a complicated script open in the editor with various structural elements, then using the right-click and choosing Select All followed by Smart Indent causes Matlab to automatically indent loops and logical blocks.

Note that in the Matlab editor, roughly in the middle of the HOME tab, is a large green arrow labelled Run (see Figure 1.3). Clicking this arrow gets Matlab to rst save the current le that is being edited and then to try to run the le. Any syntax errors are reported in the command window, coloured in red. There will also be a link to the line in the le which caused the error. Clicking on this link will open the le in the Matlab editor and position the mouse as close to the source of the error as possible.

Warning The School of Mathematics and Statistics holds computer laboratory exams in many courses. For these exams the computers are in a special Linux based exam mode. While Matlab runs on Windows, Linux and Macintosh operating systems, the short-cut control keys are often di erent between the di erent operating systems. For example in the Windows version of the Matlab editor, Cut and Paste are Ctrl+c and Ctrl+v respectively, while under Linux Cut and Paste are Alt+w and Ctrl+y respectively. Thus it is preferable not to depend heavily on the use of short-cut keys. The context sensitive menu obtained by right-clicking gives you the same set of items under both Widows and Linux.

1.5.1 Help facilities

Matlab comes with a variety of help facilities as part of the software product (plus many more available over the www). A Matlab command in the le task.m uses the comments (up to the rst blank line) at the beginning of the le or function to provide documentation about the purpose of the script and input and output arguments for functions.

In the Matlab command widow, typing

```
help exp
```

displays information in the command sub-window about the exp function which is obtained from the comments at the beginning of the le exp.m. At the bottom of the information displayed in the command window are suggestions for related commands (for example the log command with help exp. (Do not worry if you have not heard of some of these commands at this stage).

Matlab's help browser, pictured in gure 1.4, provides another interface to Matlab's documentation.. The information from the comments at the beginning of a le may be displayed in Matlab's help browser using the links at the end of the information displayed by the help command, or directly with the command

```
doc exp
```

Figure 1.4: The Help browser window with a listing of available toolboxes

You can also search for information using the Search Documentation box in the top right hand corner of the main Matlab window { see Figure 1.2 or Matlab editor { see Figure 1.3. Alternatively Matlab 's help browser can be opened by clicking on the question mark ? in a small circle just to the left of the Search Documentation box. You can then enter the term you wish to search for. You can search the help pages either for a known function name or search for a phrase if you do not know the command name (for example, search on \log").

The documentation for both exp and log mentions complex numbers which you may have seen in high school mathematics, but will de nitely see in rst year university mathematics. One of the powers of packages like Matlab , in contrast to Microsoft Excel for example, is that Matlab can automatically work with complex numbers (this is both powerful and dangerous).

The best way to learn how to use Matlab is to experiment and try commands, and explore the online help facilities and examples. The Help browser has tutorials and information about Matlab **toolboxes** (additional packages of programs for special applications { see Figure 1.4) amongst other information. You may use the Help browser in the laboratory. It may also be used in Matlab tests held in the computer laboratory.

Chapter 2

MATLAB COMMANDS

This chapter provides an introduction to some Matlab commands and basic features of the language. More details will be provided during your course. There are many books on Matlab and its use in Engineering, Science and Business. Cleve Moler, one of the founders of Matlab , has a text book [3], while the Matlab Guide [1] is very useful for more advanced techniques.

The best way to use this chapter is rst to glance through it to get an idea of what Matlab can do (actually it can do far more than what we have described here), bearing in mind that many of the things in this chapter refer to mathematical ideas and processes covered in rst year mathematics courses. Later, when you are solving a speci c problem, read through the relevant sections of this chapter and your lecture notes, before preparing a list of Matlab commands to solve that problem. Then, when you are entering these commands, use Matlab 's Help browser (see section 1.5.1) for the exact syntax.

NOTE There are built in demos in Matlab . To use them, either enter the command

```
demo
```

during a Matlab session or select the \Demos" tab in the Matlab Help browser window. For further information, see section 1.5.1.

2.1 Basics

2.1.1 Arithmetic

The usual arithmetic operations are available in Matlab and you should use the following notation to enter them in commands.

addition	+
subtraction	-
multiplication	*
division	/
exponentiation	^

So a^b means a to the power b (i.e. a^b).

These follow the usual order of evaluation, i.e. anything in brackets, then powers, then multiplication or division, then addition or subtraction.

If you want to use a di erent order then you will have to insert brackets `( ' and `) ' in the appropriate places. For example -1 ^(1/2) means -(1 ^(1/2)) (i.e. 1), whereas (-1) ^(1/2) means $\sqrt{-1}$ (i.e. the imaginary number i , which is denoted 1.0000i in Matlab) and -1 ^1/2 gives 0:5000 .

2.1.2 Assigning variables

You use = to assign a value to a variable, for example

```
x=1,f = sin(x)
```

This assigns the value 1 to x and then sin(1) 0:8415 to the variable f . If x were an unknown (it had not been assigned a value), then you would get an error message.

What we have been doing is called **assigning a value to a variable** and the general format for doing it is

```
variablename= expression
```

After you have given an assignment command, Matlab will replace the named variable with its assigned value wherever that variable name occurs in the future.

If you do not assign an answer to a variable, then Matlab will assign the result of the calculation to the default variable ans, which you can then use in the next calculation like any other assigned variable.

2.1.3 Variable Names

Variable names must start with a letter and the initial letter can be followed by letters, digits and the underline character _".

There is e ectively no limit to the length of a name, but Matlab only looks at the rst few characters, where \few" depends on how that system is set up: on the version in the Mathematics and Statistics computer laboratories it is the rst 63 characters that count (see the command namelengthmax). Upper and lower case letters are treated as di erent in names. Here are some examples

```
t        t3       A_b       T       time
```

You should avoid using names already used as function names for your own variables, as then you would be unable to use the function. You can test to see if a name is being used by a command like

```
>> which -all tan_x
   tan_x not found.
```

This means that tan_x can be used as a variable. Anything else means it cannot.

Special Variables

Five names stand for constants that are important namely,

pi		3:14159265358979
i or j	i =	p 1
Inf	*1*	
eps	2 52	2:2 10 16 the **machine epsilon**

The machine epsilon eps is the smallest positive number such that Matlab considers 1+eps to be greater than 1.

2.1.4 Controlling Output

Often in doing Matlab calculations you will create a very long output that you do not need to actually see. You should get into the habit of **suppressing** long output by ending such commands with a semi-colon ; so that your Matlab screen does not get cluttered up. So for example

```
>>  a=3; b=5^2-a^2
b =
    16
```

75.947 -14(406 Tdr-363(y)27(ou)-35n)]TJ/F30 11.9552 Tf 246.e\fep(gre0(les)]TJ/.955fep(gtik)27(ed

2.1.5 clear

The command clear can be used to remove variables and functions from memory:

clear	clears all variables
clear functions	clears (i.e. forgets about) all M-files and other defined functions
clear a b	clears variables (or M-files) a and b only
clear all	clears everything: variables, M-files etc.

See the help on clear for further detail.

2.1.6 Number Formats

Matlab does all its calculations in IEEE double precision (64 bit) floating point binary arithmetic. This means that Matlab works to about 16 decimal digits and can handle floating point numbers as large as about 10^{308} and as small as about 10^{-308}. See the functions realmin and realmax.

To control how a number is displayed, you use the format command: changing the format has no effect on Matlab's internal calculations.

The following table shows the output of $\sqrt{2009}$ in the various formats.

command	output	
format short	44.8219	this is the default
format short e	4.4822e+01	that is, 4.4833×10^1, note rounding
format long	44.821869662029940	16 places (double precision)
format long e	4.482186966202994e+01	
format bank	44.82	as if it were money
format rat	14343/320	a rational approximation

There are two other possible types of output you might get from Matlab:

Result	Meaning
Inf	∞
NaN	not a number, e.g. $0/0$

2.1.7 Complex numbers

Matlab can also handle complex numbers, such as $i = \sqrt{-1}$. Matlab will recognise both i and j (if they have not been used as variable names) as this complex number. For example

```
>> 1/2+sqrt(3)*i/2
ans =
   0.5000 + 0.8660i
```

The commands real, imag, abs and angle when applied to a complex number give, respectively, the real part, imaginary part, modulus (absolute value) and argument of the complex number. For example:

```
>> z=1/2-3*i/4;
>> real(z),imag(z),abs(z),angle(z)
ans =
    0.5000
ans =
   -0.7500
ans =
    0.9014
ans =
   -0.9828
```

You could also have de ned the complex number z by `z = complex(1/2,-3/4)`

2.2 Saving Sessions, Input and Output

Matlab 's main purpose is to perform tasks on large amounts of data, so you need to be able to get data into Matlab and save data from Matlab for later processing. In some of the laboratories exercises, you may be asked to perform some analysis on data will be provided for you (such as the temperature at each point of a grid in a two or three dimensional object or daily share prices for a portfolio of fty stocks over two years). It is also useful to be able to record your Matlab session so that you can later rerun the same commands, maybe with di erent data or with minor modi cations.

2.2.1 Data Input and Output

The `save` command is used to save the values of some or all of the variables in your Matlab session. This command saves into a le known as a **Mat- le**. Note that you cannot edit these les, as the information is stored in a binary (non ASCII) format. Also, Mat- les must have the `.mat` le extension, which Matlab will add if you do not.

The `load` command does the opposite of `save`, and loads a Mat- le into the workspace. For example,

```
>> save price.mat jan feb
>> save apr21
>> load apr1
```

The rst command saves variables `jan` and `feb` to the le `price.mat` ; the second saves all variables into the le `apr21.mat`, with the `.mat` automatically added by Matlab ; the last command loads the le `apr1.mat`, again automatically adding the `.mat` extension.

2.2.2 Recording your Session

Matlab has a built in feature that allows you to re-run the previous commands: a **history** le. This le stores all the previous commands you have entered into Matlab as you type them. You can see this in the the **Command History** sub-window (see Figure 1.2).

If you are using more than a couple of commands it is better to write a script le (see section 2.6) when you use Matlab .

If the **Command History** sub-window is not displayed, tick the box \Command History" under the **Layout** icon. If you double click on a command in **Command History** sub-window, it will be entered into Matlab and executed. To select more than one line, hold down the **Ctrl** key and click each line. The you can right-click the mouse button, select copy, click in th command window, again right-click and select paste, to execute the selected commands.

The history le is separated into sections for each di erent Matlab session, and each of these will have a time stamp. An entire session's history can be \collapsed" by clicking on the symbol on the left of this time stamp.

There is an option on the Edit menu allowing you to clear your command history if you want to.

2.3 Built-in Functions

Although we will not discuss the creation of new functions until section 2.6, we will be using functions in the next few sections, and so we will need some functions which have already been de ned. Matlab has an enormous number of `initially-known' mathematical functions (i.e. ones which are already there when you start Matlab). These include the trigonometric functions

sin, cos, tan, csc (i.e. cosec), sec, cot

and their inverse functions

asin, acos, atan, acsc, asec, acot

and the hyperbolic functions

sinh, cosh, tanh, csch (i.e. cosech), sech, coth

and their inverse functions

asinh, acosh, atanh, acsch, asech, acoth

as well as, for example:

Function	Description	Example
abs	absolute value	abs(-2)
sqrt	square root	sqrt(4)
max	largest element in an array	max([132,129,66,120])
min	smallest element in an array	min([132,129,66,120])
factorial	factorial function	factorial(12)
round	round (up/down) to an integer	round(3.5)
floor	round down to an integer	floor(-3.1)
ceil	round up to an integer	ceil(-3.1)
exp	exponential	exp(1)
log	natural logarithm	log(exp(2))
log10	logarithm to base 10	log10(100)

Note that most of these function will work on one number, or if applied to a vector or matrix (see sections 2.4 and 2.8) to each element of the vector or matrix.

For a complete list of the initially-known Matlab functions, use the Help browser (see section 1.5.1).

2.4 Basic Vectors

From its beginning, Matlab was designed to work with matrices and vectors, as its name suggests. Everything in Matlab is, potentially, a matrix: a number on its own is really a 1 1 matrix to Matlab . Matrices and vectors are collectively called **arrays** in Matlab .

We begin by looking at vectors.

2.4.1 Row and Column vectors

There are two types of vectors in Matlab : **row vectors** and **column vectors**. Both types of vector have square brackets enclosing the elements (also called components), and for both the command size will give the dimensions of the array, while numel gives the total number of elements in an array.

A row vector is printed as a row, and when you de ne one you separate its elements by either commas or spaces. A column vector is printed as a column, and you use semi-colons or new lines to separate the elements. For example

```
>> v=[ 1  3 , sqrt(21) ]
v =
    1.0000    3.0000    4.5826
>> w=[1 ; 3 ; sqrt(21) ]
w =
    1.0000
    3.0000
    4.5826
>> size(v)
ans =
      1   3
```

You can convert a row vector to a column vector and vice versa using the apostrophe or **back quote** ' | we call this **transposing**.

```
>> v=[ 1  3  sqrt(21) ] , v'
v =
    1.0000    3.0000    4.5826
ans =
      1.0000
      3.0000
      4.5826
```

To refer to an element of a vector, for example the third element of vector v, use an expression like v(3) . This can be extended to extract sequences of elements using the colon notation, see section 2.4.3. You can change the value of an entry with something like w(2)=-3 as well. Note that in Matlab all vectors (and matrices) are indexed from 1 (that is v(1) is the rst element). You can also use end to refer to the last entry in a vector.

2.4.2 Vector arithmetic

Two vectors of the same size can be added and subtracted. In fact, you can make any linear combination of the vectors you want:

```
>> a=[1  -4  9 ]; b=[-2  2  3];
>> 2*a - 3*b
ans =
     8    -14    9
```

If the vectors are not compatible then you will get an error message.

You can apply functions to each element of a vector very simply:

```
>> v=[pi/4,pi/3,pi/2]; sin(v)
ans =
      0.7071      0.8660      1.0000
```

2.4.3 Colon and linspace

Entering a small vector by hand is not a problem, but Matlab was designed for big problems, and often these involve vectors whose entries have some regularity, such as consecutive integers, or consecutive odd integers going downwards. If the entries of a vector are an arithmeticsequence, then you can use the **colon operator** or the linspace command to build the vector. Both of these produce row vectors, which can be transposed to column vectors with the apostrophe. For example

```
>> a=[1:4]
a =
    1      2      3      4
>> b = linspace(1,4,4)
b =
    1      2      3      4
>>c = [7:-2:1]
c =
    7     5      3      1
```

In general using [a: b: c] where a, b and c are numbers will create a row vector whose rst element is a, second element a + b etc and whose last element is no greater than c (if b > 0, no less than c if b < 0). If there are only two numbers then Matlab assumes the **increment** (b above) is 1, as in the rst example.

On the other hand, linspace(a, b, c) creates a row vector with exactly c entries (100 if c is omitted) with entries equally spaced between a and b.

The colon operator can be used to extract more than one element at a time. Suppose vector w had 12 elements. Then the command

```
>> w( [ 1:2:5, 10:end ] )
```

will create a vector consisting of elements w(1),w(3),w(5),w(10),w(11),w(12) only.

2.5 Plotting

Matlab has a large number of plotting commands, used for various special plots. We will only look at a few of the simplest and easiest.

2.5.1 plot command

The basic plotting command is plot .

The le containing gure 2.1 was produced with the following commands

```
>> x = linspace(0,1,101);
>> y = sin(4*pi*x);
>> plot(x,y);
>> print -dps 'sinplot.ps'
```

Figure 2.1: Plot of $\sin(4 x)$ over $[0; 1]$

The rst command sets up a vector of 101 points along the x-axis, equally spaced between 0 and 1 (so 0.01 apart) and including both 0 and 1. Then we de ne $\sin(4 x)$ for each of these points. The plot command then plots the points $(x_i; y_i)$ for each x_i in the vector x and corresponding y_i in vector y, then joins them up with straight lines:

code	r	y	g	b	c	m	w	k
colour	red	yellow	green	blue	cyan	magenta	white	black

code	.	o	-	:	-.	--	x	*
style	points	circles	solid	dotted	dash-dot	dashed	x-marks	stars

The di erent styles and colours allow you to plot several graphs at once in a way you can tell them apart. To plot both sin(4 x) and cos(4x) you could use (with x and y as above)

```
>> z = cos(4*pi*x);
>> plot(x, y , 'r-' , x , z, 'b--')
```

Here the sin plot is red and solid, the cos plot blue and dashed.

2.5.4 Titles, axes and grids

The ezplot command will automatically put a title on a graph | it uses the function as the title, not surprisingly. You can put a title on any plot using the title command, for example

```
title('My first plot');
```

Figure 2.3: Plot of the cardioid $r = 1 + \cos(\ )$ using polar

in your session but do not want to save as an M- le. These anonymous functions are also used in numerical integration (see section 2.9) and other places. A simple example will illustrate the idea:

```
>> polynom=@(t) t.^2-2.*t-3
polynom =
    @(t) t.^2-2.*t-3
>> polynom(-2)
ans =
     5
```

Note the use of the compulsory @ symbol, which is used to create the *function handle*, in this case polynom. The parentheses immediately after the @ contain the function parameters, which behave like the parameters of a function le. It is possible to have more than one parameter, or even no parameters. However, even if there are no parameters to pass to the function, you must include the parentheses to call the function (see the Matlab help page on anonymous functions for an example).

2.7 Further Vectors

2.7.1 Ordinary Product

Given a row vector v and a column vector w both with the same number of elements, you can get Matlab to calculate the usual matrix (or dot) product of v and w using a star for what is really matrix multiplication. So for example

```
>> v=[1  3  5  7]; w = [-2 ; 3 ; 4 ; -5];
>> v*w
ans =
     -8
```

An alternative is to use the Matlab command

```
>> dotprod(v, w)
```

2.7.2 Array Arithmetic

One of Matlab more useful but unusual features is a heavy reliance on *array operators*. These are operations that are applied *element-by-element* to an array (a vector or matrix). We have already noticed that we can say, for example, sin(v) for a vector v and get a vector whose entries are the sines of the entries of v.

We can apply more basic functions to the elements of an array (or more than one array, as appropriate) by using *array operators*, sometimes called *dot operators*, as they use a dot. For example, we can create a vector whose elements are the cubes of the rst 5 integers by the command

```
>> [1:5].^3
```

Note the dot: .^3 means cube each member of the array separately. It's not the same as cubing an array in the usual mathematical sense you would use for, say, square matrices.

Other examples include .* , which can be applied to two arrays of exactly the same shape and will multiply corresponding entries together, and ./

You can get the size of a matrix using the command `size` , for example

```
>> size(A)
ans =
      2    3
```

`size` is an example of a function that returns a matrix: a 1 2 matrix in fact.

Entries of a matrix can be extracted or changed just as for a vector, although you need to give two indices of course. For example `A(2,1)` extracts the entry in the second row, rst column of A (if A has a second row). Similarly to vectors you can extract more than one element using the colon, and in this way create submatrices. Once again, the indexing begins from 1 and the keyword `end` can be used for the last entry, see section 2.4.1.

For example

```
>> B = [1 3 5 ; 2 4 6 ; 4 9 16 ];
>> C = B( 2:3 , : )
C =
   2     4     6
   4     9     16
```

Note that the colon on its own is equivalent to `1:end` and means all the rows (or columns) of the matrix.

In many applications matrices have some sort of structure and are most easily made by being built up from smaller matrices and/or vectors. One obvious example is creating the augmented matrix for a system of linear equations (see section 2.8.5 for solving linear equations). However, in Matlab you can not only **augment** matrices/vectors (put them side by side) but also **stack** them (put one on top of the other). For example:

```
>> A=[1 2; 3 4]; v=[-1; -1]; B= [-3 -2 ; 0 -1 ];
>> [A v] % augmenting
ans =
     1     2     -1
     3     4     -1
>> [A ; B] % stacking
ans =
     1     2
     3     4
    -3    -2
     0    -1
```

2.8.2 Special matrices

Matlab includes several useful commands for creating special types of matrices:

1. For a 3 3 (say) identity matrix, use `eye(3)`
2. For a 3 4 (say) matrix of zeros use `zeros(3,4)`
3. For a 3 2 (say) matrix of ones use `ones(3,2)`

4. To create a diagonal matrix whose entries are the elements of the vector v use diag(v)

These matrices can be particularly useful in stacking and augmenting matrices.

2.8.3 Standard Matrix Arithmetic

For the usual mathematical product of two matrices, or a matrix and a vector, use the * symbol on its own. The two arrays you multiply must have compatible dimensions. Also, do not forget that A*B and B*A will in general give di erent results

```
>> A = [1 2 3; 4 5 6];
>> B = [0 1 ; 1 0 ; 0 0];
>> A*B
ans =
        2       1
        5       4
>> B*A
ans =
        4       5       6
        1       2       3
        0       0       0
```

Later on in your courses you will need to use the various Matlab commands for calculating with matrices. For example,

inv(A)	for the matrix inverse;
det(A)	for the determinant;
eig(A)	for calculating eigenvalues and eigenvectors;
rank(A)	for the rank.

2.8.4 Matrix array arithmetic

Just as in the case of vectors, Matlab allows you to operate on each element of a matrix individually, so for example exp(A) will give a matrix whose $(i; j)$ th entry is $e^{a_{ij}}$. In later year courses you may come aco6or

the integer 0 represents false and 1 represents true. Suppose you had a variable x and you wanted to test to see if it is greater than ten (without actually looking at it). In Matlab this would look like

```
   >> x>10
ans =
       1
```

and the value of ans tells you that x is greater than 10.

There are 5 other **relational operators** apart from >, illustrated below. Note that they can all be used on arrays and then are applied elementwise, as is typical. Suppose we have a vector de ned by

```
>> x = [0 -1 2 4]
ans =
       [0   -1   2   4]
```

then we have the following possibilities

command	result	description
x==2	[0 0 1 0]	entries equal to 2
x>2	[0 0 0 1]	strictly greater than 2
x>=2	[0 0 1 1]	greater than or equal to 2
x<2	[1 1 0 0]	strictly less than 2
x<=2	[1 1 1 0]	less than or equal to 2
xs =2	[1 1 0 1]	not equal to 2

Note that the 2 on the right hand side is assumed to be an array of the right size all of whose entries are 2.

For more advanced uses of logicals we need the logical operators & (and), | (or) and s (not). So with the vector x above we get

```
>> x>=0 & x <=2
ans =
       [1   0   1   0]
>> x<0 | x>1
ans =
       [0   1   1   1]
>> s (x>2)
ans =
       [1   1   1   0]
```

2.10.2 If Statements

An if...elseif...end statement is known as a **conditional**, a **branch** or a **fork** | control is sent down one of two possible paths depending on the truth value of a boolean statement. For example

```
      if (x>3) | (y<=2)   ... end
```

```
    if (a>b) & (c>d) ... end
```

If the boolean is true then Matlab runs the commands after the boolean. If you want to, you can make Matlab do something else if the boolean is false, or do nothing; you do the former with an else clause. For example, the following commands nd the absolute value of a real number:

```
 if x>=0
       x
 else
     -x
 end
```

You can **nest** if statements as well; the general form of the if command is something like

```
    if condition1
           commseq1
    elseif condition2
           commseq2
    elseif condition3
           commseq3
    else
           commseq4
    end
```

2.10.3 Loops

Suppose that you want to execute a set of Matlab commands several times, changing the value of one variable n at each repetition. This is called creating a **loop**, and is very common in scienti c and nancial programming.

The way you create the loop depends on whether you know in advance exactly how many times you want to repeat the commands or not. If you know that you want to repeat the commands 100 times then you can use a construction of the type

```
 for n = 1:100
   commands
 end
```

The 1:100 is the colon operator we met before, and can be generalised here too, so 100:-2:0 would have n run through even numbers backwards. Also note that unlike many other languages, Matlab allows non-integer increments in loops, so h=0:0.1:1 is legal. The end is essential to tell Matlab where the commands to be repeated end.

If you do not know how many repetitions you want to make then you will have to tell Matlab to keep repeating until some condition is no longer satis ed, using a construction of the type

```
 while a<= b
   commands
 end
```

To illustrate, we give two examples. Firstly, consider the following commands

```
>>  t=linspace(0,2*pi,200);
>>  for  n  =  0.5:0.5:4
           polar(t,2+n*sin(t))
           pause
     end
```

This block will plot various polar curves known as **limacons**, with the pause statement

Bibliography

[1]