Maths 363 Matlab Primer

We will work through most of this in class but the best way to learn this is to try it on a computer. Note that you can find more detailed information on the course website and by using Matlab's help.

Part 1

- Using MATLAB as a calculator.
- Constants
- Arithmetic Operators
- Arithmetic Expressions
- Elementary Functions
- Help
- Variables, workspace, who, clear, reserved words
- A simple graph
- Character Strings
- Concatenation
- `disp` command
- `format` command
- Introduction to Script Files
- Better Looking Output
- Boolean Expressions
- Introduction to Flow Control - If Statement

MATLAB is a computer language/application designed for scientific computation.

Starting

Click on the Windows start symbol at the bottom left of the screen, click Programs from the menu that appears, then move to cick Matlab and then again. This opens the Command Window where you can use MATLAB like a calculator.

Prompt

When MATLAB is ready to accept a new instruction it shows a prompt `>>`.

Enter

Press the enter key when you have typed in your instruction. It does not matter if the cursor is not at the end of the line.

Continuation of a Line

If your instruction is too long to conveniently fit on a single line you can break it after an operator by typing three dots then using return and continuing your instruction.

Constants

Some constants are:

`5`	`-206`	`0.001`	`8.1468`
pi	3.1638E-3	5.24E12	

Arithmetic Operators

The arithmetic operators in *decreasing* order of precedence are:

arithmetic operator	operation performed
`^`	raise to a power
`/`	division
`*`	multiplication
`-`	subtraction
`+`	addition

You often need to use brackets to make sure that the operations are carried out in the correct order.

```
>>12/(1+3)
ans=3
```

Arithmetic Expressions

```
>>2*3
ans=6

>>12/5
ans=2.400

>>2^3
ans=8

>>10-(1+3)
ans=6
```

Write in MATLAB

1. $3^2 + 5$
2. 3^{2+5}

3. $\frac{60}{2+3}$
4. $\frac{60+3}{2}$
5. $-2 \times 5 \times -7$
6. $\frac{12-3}{5+1}$
7. $\frac{1}{2^4}$
8. $(\frac{3}{4})^4$
9. 5π

Elementary Functions

Most of the usual mathematical functions are implemented in MATLAB.

```
>> cos(0)
>> 6*sin(pi/2)
>> exp(1)
>> log(exp(3))
>> sqrt(9)
```

Note:

- MATLAB uses radians, not degrees.
- The `log` is the natural log (often labelled `ln` on calculators). The log base 10 is `log10`.

Help

If you know the name of the function that you want help on use the help command.
For example,

```
>> help log
```

will give you some notes on the function `log`.

Variables and Memory

A variable is a labelled piece of computer memory.
s=5.6
The number 5.6 is stored in the computer memory and labelled.

- The workspace is the memory and variables that are in use. This can be saved.
- `who` lists the variable names in current use.
- `whos` lists the variables and how much memory they use.
- `clear X` clears the variable `X`.
- `clear all` clears all the variables.
- Reserved words are variable names that you cannot use.
- Matlab is case sensitive. That means that the upper case and lower case of a letter are different. For example, the variable called `s` is different from the variable called `S`.

A Simple Graph

```
>> x=linspace(0,2*pi,50);
>> plot(x,cos(x))
```

The first line sets up a variable called `x` which is 50 equally spaced numbers starting with 0 and ending with 2π. The second statement plots the values in `x` along the horizontal axis and the cos of each value up the vertical axis.

Adding Labels

```
>> xlabel('x')
>> ylabel('cos x')
>> title('A graph')
```

Copying Previous Input Lines

The key ↑ will copy the previous command into the current line. Use it twice to get the one before etc.

Character Strings

Variables can be used to store numeric values (eg -6.5e3) or strings (eg 'Hello').
Note that the single quote marks are used for strings in MATLAB.

```
>> n=5;
>> y='Maths 363';
```

The first character of string `y` above is `y(1)`.
The 7th to 9th characters inclusive are `y(7:9)`.

`length(y)` gives the length of string `y`.

Concatenation

This is following one string by another.

```
>> x='I am studying'
x =
I am studying
>> y='Maths 363'
y =
Maths 363
>>[x y]
ans =
I am studyingMaths 363
```

`format` Command

MATLAB always calculates internally to 16 significant figures. Use `format` to control the number of decimal places in the output.

`format short`	4 decimal places (the default)
`format long`	14 decimal places
`format short e`	scientific notation with 4 d.p.
`format long e`	scientific notation with 14 d.p.

Example

```
disp(5.55744525*2)
   11.1149
format long e
disp(5.55744525*2)
     1.111489050000000e+01
```

`disp` Command

`disp(x)` will display `x`, whether it is a number or a string.

Examples

`disp` command	output
`disp(5.7)`	`5.7000`
`disp('Hello')`	`Hello`
`disp(15*4)`	`60`
`disp('15*4')`	`15*4`

Script Files

A *program* is a sequence of statements that are performed in order. They are stored in a *script file.*

A *file* is the basic unit that the computer stores on mass storage (such as hard disk or floppy disk). Files can store pictures, or programs (such as MATLAB script files) or other data (such as text files).

To *run* a program either type the name of its file in the Command Window or use the Debug menu in its window and choose Save and Run.

Relations

Relations are comparisons which are either true or false.

In MATLAB: false is represented by 0
true is represented by 1
(or any other non-zero number)

Examples

relation	result
`6>3`	true (or 1)
`3>6`	false (or 0)
`7==4`	false (or 0)
`7~=4`	true (or 1)

Boolean Expressions

Relations may be combined using logical operators. Use brackets to make sure of the order of evaluation.

`~`	logical NOT	changes to opposite truth value
`|`	logical OR	at least one relation is true
`&`	logical AND	both relations are true

Examples

expression	result
`(6>3) & (4>2)`	true (or 1)
`7==4 | 2<6`	true (or 1)
`~(2>4)`	true (or 1)
`(~(2>4)) | 2>1`	true (or 1)
`(2<0)&((1>-1)|(4==2))`	false(or 0)

Quick Quiz

```
 ( (5>0) | (5<10)) &(2==3)

(~(5>0)) | ((1+3==4) & (5<12))
```

Better Looking Output

Use function `sprintf` to produce strings which include numerical results.
For example, if `depth` is a variable equal to 18.98525 then

```
disp(sprintf('Depth is %4.2f metres.',depth))
```

The output would then be

```
Depth is 18.98 metres.
```

Boolean Expressions in IF statements

```
x=5;
if (x>2)|(x<0)
  x=2;
end
```

Exercise

No more than 10 drinks may be served to one person. If more than 10 are ordered, the number is reduced to 10 and a message printed out. Write a script file to check the variable `drinks` which is set in the command window and display the appropriate messages.

Sample Output

```
You have ordered 20 drinks. Reduced to 10.
You have ordered 5 drinks.
```

Maths 363 Matlab Primer

Part 2

- `input` statement
- `rand` function
- `if` statement
- `if .. else` statement
- `if .. elseif ..else` statement
- Introduction to vectors
- `for` statement
- `While` statement
- Functions
- More on Functions
- Introduction to 2 dimensional arrays
- More on Vectors (1 dimensional arrays)
- More on Matrices (two dimensional arrays)

`input` statement

This is used to enter values for a MATLAB program. It includes a string which is used as a prompt so the user knows what to enter.

Example

```
x=input('Enter a positive number ');
```

When this statement is executed, the string `Enter a positive number` will appear in the command window. The user then types in a value, and this value is assigned to the variable `x`.

`rand` function
Gives a random number between 0 and 1.

`if` statement

The simplest type is

`if` *boolean expression*
some statements
`end`

Example

```
p=input('Enter a positive number ');
if p<=0
   disp(sprintf('%4.2f: not positive',p))
end
```

Other `if` statements

- `if` *boolean expression*
  some statements
  `else`
  some more statements
  `end`
- `if` *boolean expression*
  some statements
  `elseif` *boolean expression*
  some more statements
  `else`
  some more statements

```
   end
```

Example

A coin is tossed. If the result is a head the computer wins. If the result is a tail the coin is tossed a second time and if it is a head the computer wins and if a tail the user wins. Write a script file to play this game and display who wins.

```
toss1=rand;
toss2=rand;
if toss1>0.5
    disp('Computer wins')
elseif toss2>0.5
    disp('Computer wins')
else
    disp('You win')
end
```

Vectors

A vector is a one dimensional array of numbers.
Some examples are:
`[1,3,2,9,5],  [-1 3 8 0 -2 1],`
`1:100, 1:0.1:10, linspace(0,10,20)`

We can find how many elements in a vector by using the function *length*.
`length([1,3,2,9,5])` is 5.
`length(1:0.1:10)` is 91.

A vector of length 1 (ie just a number) is called a scalar.

The vector `[]` is the null vector. It has length zero.

We can refer to the element of a vector by using its index. For example, if we use the MATLAB statement
`v=[1,3,2,9,5];`
then we can use `v(3)` to refer to the third element, which is 2.
We can use a vector for the index in order to refer to more than one element. For the vector above, `v(2:4)` refers to the second to fourth elements inclusive of `v`. This will be `[3,2,9]`.

Examples
Let `w=[9,2,10,-5,0,-2,4,1]`. Write down:

1. `w(5)`
2. `2:3:8`
3. `1:2:8`
4. `w(2:3:8)`
5. `w(8:-3:2)`
6. `w(length(w):-1:1)`

`for` statement

The syntax is

```
for variable=vector
    some statements
end
```

Example
Write a script file to calculate the squares of the integers up to 20.

```
% A script file to print out the
% squares from 1 to 20.
for i=1:20
   disp(sprintf('%2.0f squared is %4.0f.',i,i^2))
end
```

Now change the script file to calculate the squares of the odd numbers up to 21.

```
% A script file to print out the odd
% squares to 20.
for i=1:2:21
   disp(sprintf('%2.0f squared is %4.0f.',i,i^2))
end
```

Now change it to display them backwards

```
% A script file to print out the squares
% backwards from 20.
for i=21:-2:1
   disp(sprintf('%2.0f squared is %4.0f.',i,i^2))
end
```

Example
Write a script file to calculate the sum of the integers up to 100.

```
total=0;
for n=1:100
   total=total+n;
end
disp(sprintf('Sum up to 100 is %3.0f.' total))
```

`while` statement

The `while` statement is used to execute a loop while a given boolean expression is true.

The syntax is

```
while   boolean expression
    some statements
end
```

Example
Write a script file which prompts the user for a number M and then computes and displays the integers and their squares while the square is less than M.

```
% Calculates squares while square is
% less than M
M=input('Enter the maximum number ');
k=1;
while k^2 < M
   disp(sprintf('%2.0f squared is %4.0f.',k,k^2))
   k=k+1;
end
```

The `for` statement is used when we know in advance how many times we want to execute a loop. For example, to evaluate

$$\sum_{i=1}^{n} i^2$$

we know that the loop must be repeated n times even if we do not know the value of n before we run the program. With the `while` statement we can test for desired condition each time we execute the loop.

We could use a `while` loop to keep repeating some process until the user enters a zero, say.

Functions

Syntax of a Function Header

`function [` output `]=` funct-name `(`input`)`

If there is only one output variable, you omit the [ ]. There may be no output variables.

For example,

```
function [y,z]= f(x)
```

The statements that follow the function header will calculate the output variables in terms of the input variables.

The function header and statements are typed into a file. Each function is in a separate file.

Example

```
function [vol,p_l]=can(h,r,p_c)
%returns a vector containing the volume
%vol (litres) and price/litre p_l of can.
%Inputs are height h (cm), radius r (cm),
%price p_c (dollars).
vol=pi*h*r^2/1000;
p_l=p_c/vol;
```

This will be typed into a file and saved as `can`. It is important to use the function name as the file name. We can use the function by typing `can(12,3,1)`.

Use of Variables

A function communicates with the rest of the workspace only through the input and output variables.

Any other variables that you use inside the function are known only inside the function. They are not defined in the Command Window for example.

Example

```
function v=facts(n)
% function finds vector of
% factorials
p=1;
v=[1];
for i=2:n
   p=i*p;
   v=[v p];
end
```

Another Function

```
function [a,b]=first(w)
%w is a vector
i=1;
while w(i)<=0
    i=i+1;
end
a=i;
b=w(i:length(w));
```

- What is the result of `[r,s]=first([-2 9 -3 8 9])`?

- What does the function `first` do?

2-D Arrays - Matrices

We have studied 1-D arrays (vectors) which are like a list of numbers. The 2-D arrays go both across and down, like the matrices we study in mathematics.

Example

The matrix

$$\begin{pmatrix} 1 & -5 & -4 \\ -2 & 7 & 0 \\ 5 & 0 & -1 \end{pmatrix}$$

can be written in Matlab by

```
A=[1 -5 -4; -2 7 0; 5 0 -1]
```

or

```
A=[1, -5 ,-4; -2 ,7, 0; 5, 0, -1]
```

The numbers in a row are separated by spaces or commas (as in a vector), and the end of the row is marked by a `;`.

If we want to refer to the element in the 3rd row and the 1st column then we use `A(3,1)`. Note that the row number is the first index and the column number is the second index.

If a matrix is multiplied by a vector in Matlab, then the result is the same as the usual matrix/vector multiplication.

If `v=[1; 4; -2]` then `A*v` will be `[-11;26;7]`.

Exercise

Write Matlab statements to calculate

$$\begin{pmatrix} 1 & 0 & 2 \\ 2 & 0 & 0 \\ 2 & 5 & 1 \end{pmatrix} \begin{pmatrix} 3 \\ -2 \\ 7 \end{pmatrix}$$

Picking out rows and columns

If we want to pick out just the first row of matrix `A`, then we use `A(1,:)`. It picks out the components which have their first index as 1.
To pick out the second column use `A(:,2)`.

Examples

```
C=[25,14,0,29,21;19,12,10,28,18];
```

Find `C(1,:)` and `C(:,2)`.

Some Vector Functions

`length` returns the length of a vector (i.e. how many elements or components it has).
`min` and `max` returns minimum and maximum elements respectively of the vector.
`sum` returns sum of the elements in vector.

`prod` returns the product of the elements in the vector.
`transpose` returns the transpose of the vector i.e. a row vector becomes a column vector and vice versa. Also use `v'`.

Some Vector Operations

You can add and subtract vectors, and multiply vectors by scalars in the usual way.

If you multiply vectors together, i.e. `v*w`, then it is matrix multiplication that you do. This means that you can only multiply a row vector by a column vector (and the result is a scalar), or multiply a column vector by a row vector (and the result is a matrix). All other multiplications will give an error.

```
v=[1,3,5,7]; w=[2;4;5;6];
```
Find:
```
v*w                w*v
```

Element by Element Operations

Two arrays can be multiplied together element by element. For example, for two vectors, `v` and `w`, `v.*w` is a vector of the products of the corresponding elements of `v` and `w`. The two arrays multiplied will need to have the same size and shape. Also called componentwise operations.

We can also take powers of arrays element by element. Each element of the array is raised to that power to form the result.

Examples

`v=[-5 0 2 1]; w=[9 1 5 2];`
Then
`v+w` is `[4 1 7 3]`.
`v-w` is `[-14 -1 -3 -1]`.
`v.*w` is `[-45 0 10 2]`.
`v.^2` is `[25 0 4 1]`.

Plotting

When we plot a graph, we make a vector of x values for the horizontal axis and then a vector of y values that correspond to these x values. For example, suppose we want a rough graph of $y = x^2$ from $x = -5$ to $x = 5$. We could use `x=[-5,-4,-3,-2,-1,0,1,2,3,4,5]`. To make the `y` vector we need to square *each* of these `x` values. This means we want element by element squaring, not to multiply the vector `x` by itself in matrix multiplication. So we write `y=x.^2`.

Most of the built in functions in Matlab, such as `sin` and `cos`, can be used with vectors or matrices. The function is applied to each element. If we write our own functions that we might want to use for plotting we should make them suitable for vector input parameters.

Boolean Expressions with Vectors

We can use Boolean expressions to pick out elements of vectors which satisfy Boolean expressions.
For example, if `r=[7,-4,0,6,-2,1]`,
then `r>0` is `[1,0,0,1,0,1]`. It shows the positions of the elements of `r` which are positive.

Some Matrix Functions

`size` returns the dimensions of a matrix (i.e. how many rows and columns it has).

`sum` returns the sum over rows or columns.
`sum(A,1)` is a vector of the sums of the columns.
`sum(A,2)` is a vector of the sums of the rows.

`prod` as for `sum` but with products.

`transpose` returns the transpose of the matrix, i.e. a rows become columns and vice versa.

Some Matrix Operations

You can add and subtract matrices, and multiply matrices by scalars in the usual way.
If you multiply two matrices together then it is matrix multiplication and the number of columns of the first matrix must equal the number of rows of the second.
If you want the power of a matrix then the matrix must square, i.e. the number of rows equals the number of columns.

Matrices can also be multiplied together, or powers may be evaluated, element by element as for vectors.

Examples

Write a Matlab function to evaluate the function

$$f(x) = \frac{2x+1}{x+3}e^{-x}.$$

Write a script file using the above function to plot a graph for f using 50 values from $x = 0$ to $x = 20$.

Write a function with input parameter a matrix `K`. The output parameter will be a column vector of the sum of the absolute values of the elements in each row.

Special Matrices

`diag([1,3,-2,7])`=

`eye(4)`=

`ones(2)`=

`ones(2,1)`=

`zeros(2,3)`=