Matlab Introduction

Basic Commands

You can use Matlab as a simple calculator, to perform elementary arithmetic operations.

```
>> 4*5
ans =
   20
```

or

```
>> 20 + 3
ans =
   23
```

The basic mathematical function are also available such as sin. The arguments to the function must be placed between parentheses, '()'.

```
>> sin(pi/2)              % Arguments to trigonometric functions must be in RADIANS not degrees
ans =
    1
>> asin( sin(pi/2) )      % Similarly the results of  arc sine are also in RADIANS
ans =
   1.5708
>> asin(sin(pi/2))*180/pi   % We can convert the result to degrees
ans =
   90
```

Assigning Variables

As calculations become more complex, it is convenient to assign names to our variables. e.g

```
>> a = 4;        % Adding the ';' suppresses echoing of the value
>> b = 5;
>> c = 3
c =
    3
```

You can check that a has the value 4

```
>> a
a =
    4
```

And calculate 4*5 again, assigning the result to new variable d.

```
>> d = a*b
d  =
   20
```

If you don't specify a variable for the result, matlab uses the defaul variable ans.

If you haven't programmed before, the slightly confusing aspect of assignment arises when the same variable appears on the left and right hand-side of an assignment.

```
>> a = a + 1
a =
    5
```

Matlab calculates the expression on the right hand side of '=' and assigns the result to 'a' overwriting the previous value. In Matlab if you want to test for equality of lhs & rhs you need to use '=='.

```
>> a == 5   % Does  'a' equal 5?
ans =
    1
>> a == a + 1    % Does 'a' equal 'a+1'?
ans =
    0
>> a         %  check a is still 5
a =
    5
```

In matlab something that is 'true' is represented by 1 or a non-zero number and something that is

'false' is represented by '0'

Matrix Creation

Matlab stands for 'MATrix LABoratory' and the basic data type of Matlab is a matrix. Vectors and scalars are simple special cases of matrix. You can use '[, ']' to create elementary matrices & vectors

```
>> A = [ 1 2 3; 4 5 6; 7 8 9]        % Creates a 3x3 matrix
                                      % a ;  => starts a new row
A =
    1    2    3
    4    5    6
    7    8    9
```

Anything after a '%' is treated as a comment by Matlab

```
>>  b = [ 5 3 2];                 % creates a row vector
```

The addition of a semi-colon at the end of the line prevents Matlab echoing the result of the expression in the command window.

```
>> c = [ 3;2;1];                  % creates a column vector
```

One can create a column vector from a row vector or visa-versa using the transpose operator '

```
>>  d = b';
```

You can find the size of matrices and vectors using the following

```
size(A)         % find row and column dimensions of A
ans =
    3    3

length(b)    % find number of elements in b
ans =
    3
```

Creating Special matrices

Vector with a fixed increment between elements can be created in a number of ways. If you know the increment then you can use a special version of vector creation operator '[ ]'

```
>> e = [1:7]          % creates a row vector with unit spacing from 1 to 7
e =
    1    2    3    4    5    6    7

>> e  = [1:2:7]     % creates vector with increment of 2
e =
    1    3    5    7

>> e = [0:0.1:1]    % creates a vector between 0 and 1 with a spacing of 0.1
e =
  Columns 1 through 10
         0       0.1       0.2       0.3       0.4       0.5       0.6       0.7       0.8       0.9
  Column 11
         1
```

If you know the limits of the vector you wish to create and the number of elements in it, use the function 'linspace'

```
>> e2 = linspace(1,11,5)  % creates uniformly spaced 5-element
e2 =
   1.0000    3.5000    6.0000    8.5000   11.0000
```

There is an alternative version called 'logspace' that creates a logarithmically spaced vector. This

function will be useful when you do resistivity.

```
>> f = ones(3,1)       % creates a column vector of ones
f =
    1
    1
    1

>> g = zeros(1,3)     % creates a column vector of zeros
g =
    0    0    0

>> B  = eye(3)       % creates a 3x3 identity matrix
B =
    1    0    0
    0    1    0
    0    0    1
```

Strings

Matlab also has a string data type which is a treated as a special kind of vector

```
>> str = 'hello world';    % Use single quotes to start/finish string
```

Accessing elements of a matrix or vector

A single element.

```
 >>  b(2)     % The 2nd element of vector b
ans =
    3
```

(If you type a Matlab expression without an assignment ("=" sign). Matlab will automatically assign the result to the default variable ans)

```
>>  A(2,3)    % The element in the 2nd row and 3rd column of A
ans =
    6
```

Accessing a range of elements.

```
>> b(2:3)       % The 2nd -> 3rd elements of b
ans =
    3    2
```

We can do the same with our string

```
>> str(1:4)
ans =
hell
```

```
>> A(1:2,1:2)       %     The 2x2 leading sub-matrix of A:
ans =
    1    2
    4    5
```

The is a special ':' notation to refer to an entire row or column of a matrix

```
>> A(:,2)     % accesses the 2nd column of A
ans =
    2
    5
    8
```

The special element 'end' always refers to the last element of a vector or the last row/column of a matrix. It is useful shorthand in many situations where you don't explicitly know the length of a vector

```
>> e(5:end)
ans =
       0.4        0.5        0.6        0.7        0.8        0.9          1
>> str(7:end)
ans =
world
```

Elementary Matrix Operators

The operations of addition (+), subtraction (-) and multiplication (*) follow the usual matrix rules. You can add or subtract matrices if they have the same dimensions. You can multiply matrices and vectors as long as their inner dimensions agree.

```
>> C=A+B
C =
    2    2    3
    4    6    6
    7    8   10
>> 3*B          % Scalar multiplication
ans =
    3    0    0
    0    3    0
    0    0    3
>> C=A+3*B
C =
    4    2    3
    4    8    6
    7    8   12

>> A*c          % Matrix vector multiplication
ans =
   10
   28
   46
```

Most of the standard mathematical functions will operate on a matrix/vector on an element by element basis. Thus

```
>> sqrt(b)
ans =
   2.2361    1.7321    1.4142
>> angles = [0:10:90]
angles =
    0   10   20   30   40   50   60   70   80   90

>> cos(angles*pi/180.)
ans =
  Columns 1 through 7
    1.0000    0.9848    0.9397    0.8660    0.7660    0.6428    0.5000
  Columns 8 through 10
    0.3420    0.1736    0.0000
```

The normal forms of multiply and divide are defined in terms of matrix operations. As a consequence, there are special forms of the multiply and divide operators, '.*' and './' that work element by element on a matrix. Thus like + & - the dimensions of the matrices must agree.

```
>> b.*b
ans =
   25    9    4

ones(1,3)./b
ans =
   0.2000    0.3333    0.5000

 A.*A
ans =
    1    4    9
   16   25   36
   49   64   81
```

There is also an exponentiation or power operator, that comes in both a full matrix form '^' and a term-by-term form '.^'. Thus A^2 is the same as A*A while b.^2 is the same as b.*b

```
>> b.^2
ans =
   25    9    4

>> b.^-1
ans =
   0.2000    0.3333    0.5000
```

Loading & Saving Variables to and from the workspace

Often you will be given data in the form of a table with a number of columns, if you save this data to disk as a text file with name, e.g., grav_data.txt, then you can load it into matlab using the command

```
>> data = load('grav_data.txt')
```

Matlab will create a matrix 'data' with the same number of columns as the input file, and number of rows equal to the number of data. On older versions of Matlab, an alternative form of the load command may be needed

```
>> load grav_data.txt     % Creates a matrix grav_data, the file  name minus the extension
>> data = grav_data;
```

The name of the input file is arbitrary, but you should always give it some form of extension such as '.txt', '.asc', to distinguish it from Matlab's own files When Matlab loads the data it will always create a matrix with the name of the file minus the extension.

The input file can have matlab comments at the end of lines, and even lines that are purely comments, but it all lines must have the same number of elements otherwise you will get an error.

Since the columns of the data matrix refer to different data types it is usually worth assigning the columns of the matrix to appropriately named vectors:

```
>> id = data(:,2);   % assuming 2nd column is data ID
>> x = data(:,3);    % assuming 3rd column is x position of measurement
>> y = data(:,4);
>> height = data(:,5);  % etc for each column of data.
```

You can save any or all of the matrices in your current workspace by using the command "save" thus

```
>> save temp
```

saves all the variables in the workspace to a binary file temp.mat in the current working directory. If you quit matlab and return later, you can restore the workspace by

```
>> load temp    % No extension given so matlab looks for the file temp.mat
```

Note, although temp.mat is a binary file, Matlab is smart enough that you can send this file to any other machine that runs matlab and load it successfully.

Writing out formatted data to a text file
If you want to write intermediate results out to a file you need to use the matlab function fprintf and supply it with a format string. The format string is modeled after the one used in the 'C' programing language. The format string offers much flexibility and thus complexity. However, you will probably only need to be able to specify the printing of integers and real numbers in the format string using:

%5d says write an integer 5 characters wide padding with spaces if necessary
%8.3f says write a real number 8 characters wide with 3 digits after the decimal point.

Also useful
%s says write a string variable using exactly the number of characters in the string

Your format string should include one specifier for each row of your matrix and end with a '\n' to tell matlab to write a 'carriage return' and end the line. Thus to write out id, x, and height from the example above

```
>> A = [id,x, height]';  % Combine 3 column vectors into a single matrix and transpose to give 3 rows
>> fprintf(1, '%5d  %10.5f  %8.3f\n', A);   % The spaces in the format string are included in the output
```

The above use of the fprintf function will print the data to the screen. If you want to write the data directly to a file you need to do the following:

```
>> fid = fopen('mydata.txt', 'w')       % Open file mydata.txt for writing 'w'
>> fprintf(fid, '%5d  %10.5f  %8.3f\n', A)
>> fclose(fid)                          % Close the file
```

Getting Help
These days, Matlab has extensive on line help available that can be accessed via the '?' button. you can also get quick help on function from the command line. If for example you want to know how to use the linspace function, type

```
>> help linspace
```

If you don't know the name of a particular function but want to find something that might be useful, try the "lookfor" function. For example to find functions associated with logarithms try

```
>> lookfor logarithm
```

You get a list of variables (matrices) in your current workspace using the function "who"

```
>> who
```

If you want to see the size of the matrices as well as the list use "whos"

```
>> whos
```

Simple Plotting
The basic plot command for matlab is "plot"
```
>>  x = [0:0.1:2*pi];
```

```
>> plot(x,sin(x))    % plots the vector sin(x) (y-axis) against x (x-axis)
>> plot(sin(x))      % special case plots sin(x) against an index vector
>> plot([1:length(x)], sin(x))  % same as the above
```

By default matlab will plot your data as a solid blue line.However you can change this by specifying a line color or a symbol

```
>> plot(x,sin(x),'r');   % plots data as a solid red line
>> hold on              % allows you to plot a second or subsequent line on the same plot
>> plot(x,sin(x),'*k');  % plots the data points of the line as black stars '*'
```

Other useful plotting commands

```
clf                     % Erases the current figure, allowing you to start again. I
title, xlabel, ylabel   % Adds title and labels to a plot. For example title('Bouguer gravity') adds a title.
                         % N.B the quotes  are necessary for a string. >>  help title etc for more details.
xlim([0 20])    % Sets the x limits of your plot to 0 and 20. In general matlab will set the limits of your plot
                % automatically to include all data, but sometimes you need fixed limits yourself
ylim([-5 5])    % ditto for the y-axis
axis equal    % sets the scaling of the x & y axis to be equal.
```

Creating Script files and Functions

You can simply type commands one at a time into the command window and have them execute immediately. However this procedure doesn't allow you to recreate your work quickly later or to easily rerun the same commands on a slightly different dataset. As you find a set of commands that work you should save them in a script, called a M-file. A M-file is simply a text file (ascii file) with a file name that ends with the extension ".m", which is required so matlab recognizes it as containing matlab commands.

Thus if we copy all the commands to calculate the gravity anomaly due to a buried sphere in the M-file sphereg.m. We can rerun all the commands in the file simply by typing sphereg at the prompt:-

```
>> sphereg;    % Runs the commands in sphereg.m (you must not include the ".m" here)
```

Example: Developing the M-file for a spherical anomaly

The following set of commands calculates the gravity anomaly due to a spherical inclusion buried at a depth of

```
>> h = 5;                % Depth to center of sphere (m)
>> m = 1000;             % Excess mass of sphere (kg)
>> x = [-5:0.1:5];       % surface measurement locations centered on sphere (m)
>> G = 6.67e-11;         % m^3 kg^-1 s^-2
>> r = sqrt(h*h + x.*x);   % radial distance to measurement point N.B .*
>> cosz = h * ones(1,length(x))./r;  % need to  vertical component of the anomaly.
>> gz = G*m*cosz./r.^2 * 1e5;       % calculate the anomaly, converting result to mgal.
>> plot(x,gz)                        % And plot it.
```

Now if we wanted to compare the anomaly for a sphere buried at 10 m we could retype all the commands. The alternative is to put the commands starting with the assignment of G into a file, called, for example, sphereg.m. (When making the file do not include the matlab prompts).

sphereg.m contains the following

```
G = 6.67e-11;
r = sqrt(h*h + x.*x);                  % radial distance to measurement point N.B .*
cosz = h * ones(1,length(x))./r;  % need to  vertical component of the anomaly.
gz = G*m*cosz./r.^2 * 1e5;        % calculate the anomaly, converting result to mgal.
```

Then the case of 2 different burial depths can be run as follows

```
>> h = 5;
>> m = 1000;
>> x = [-5:0.1:5];       % surface measurement locations centered on sphere (m)
>> sphereg;    % Runs the commands in sphereg.m (you must not include the ".m" here)
>> g5 = gz;    % save the result of the calculation in the vector g5;
>> h = 10;      % set up a new depth of burial
>> sphereg;   % rerun the calculations for the new depth
>> g10 = gz;   % save the new results
>> plot(x,g5)  % plot first results
>> hold on          % save the first plot
>> plot(x,g10,'r')  % plot second set of results in red.
>> clf              % clear out both plots
```