0. What is MATLAB?[1]

MATLAB stands for ***mat****rix* ***lab****oratory* and is one of the most popular software for numerical computation. MATLAB's basic data element is an array (or matrix), which makes programming to solve problems involving vector and matrix formulations (like those found in dynamic equation systems of macro and econometrics) fairly straightforward.

MATLAB is designed to solve problems numerically, that is, in finite-precision arithmetic. Therefore it produces approximate rather than exact solutions, and should not be confused with a symbolic computation system (SCS) such as Mathematica or Maple[2]. It should be understood that this does not make Matlab better or worse than an SCS; it is a tool designed for different tasks and is therefore not directly comparable.

The following is a *brief introduction* to MATLAB aimed at getting you up and running. You will almost certainly need to spend time at your computer with the MATLAB manual, help tools of the program itself, and any other reference book you can find on the topic.

1. Finding Your Way Around MATLAB

Double-click on the MATLAB icon on your desktop. The **MATLAB Desktop** will open—this is the Graphical User Interface for MATLAB. The desktop you see may include only one window or may have two or three. One of the windows within the desktop (or the only one you see) is called the ***Command*** window. This is where you will most often interact with MATLAB. A prompt, >>, is displayed in the *Command* window followed by a blinking cursor, which lets you know that MATLAB is ready to accept your commands.

Try typing the following:

```
>> ex1 = ones(2)
```

The output should look like this:

```
ex1 =


     1     1

     1     1
```

[1] This document is compiled from various resources. So almost all errors are theirs.
[2] Actually, there is a Symbolic Math Toolbox in MATLAB that carries out symbolic computations, but it is much less sophisticated than Mathematica or Maple.

This represents a 2x2 matrix of ones. Matlab inserts extra blank lines between practically everything. To turn off this feature, type

```
>> format compact
```

To the left of the command window, we see *Workspace* and *Current Directory* window. While the first shows the variables that are currently in MATLAB's memory, second one simply shows the contents of the folder that you are working in. The *Workspace* window shows you the variables that you have created and, most notably, their dimensions. A variable in the *Workspace* is available for use. On the bottom left you have *Command History* window – this is where past commands are remembered. If you want to re-run a previous command you can double click on it from this window (Right click on the command for more options.)

1.1 Help System

Easiest way to get help in MATLAB is to click on the question mark on the toolbar. This invokes the help system of the software where you could go through the contents of its manual, search for certain keywords and even watch some demos about how to use MATLAB.

If you know the name of a Matlab function you need help with, type

```
>> help function-name
```

to see the help text contained in the function definition itself on Command window. This is a better method for quick reference.

2. Syntax

Matlab works by executing the mathematical statements you enter in the command window. By default, any output is immediately printed to the window.

You are also allowed to assign a name to an expression for your convenience. Keep in mind that the name you assign is only a name, and it does not represent a mathematical variable (as it would in Maple, for example). Every name must have a value at all times. If you try to read the value of an unassigned name, you will get an error.

Nearly everything in Matlab is a matrix, whether it looks like it or not. This takes some getting used to. We'll be introducing matrix-style operations along with their scalar counterparts so you can understand the patterns that arise in the syntax.

2.1 Simple Math

MATLAB can do simple math just like a calculator; try typing:

```
>> 10+13
```

Now hit "Enter"

```
ans =

    23
```

MATLAB evaluates the command and returns the answer (`ans =`). You can also store the numbers as variables and operate on them:

```
>> num1 = 10

num1 =

     10

>> num2 = 13;

>> num1 + num2

ans =

    23
```

Note that MATLAB didn't show that "`num2 = 13`" when you entered that variable. The semicolon at the end of the command line tells MATLAB to evaluate the command but not display the answer.

Another point is that usage of lowercase/uppercase letters matters in MATLAB:

```
>> Num1-10

??? Undefined function or variable 'Num1'.
```

Standard order of operations would apply when a string of calculations are typed up.

```
>> 3^2*4-1

ans =

    35
```

So use parenthesis if you need a certain order:

```
>> (ans+1)/4

ans =

     9
```

2.2 Built-in Functions

MATLAB has hundreds of pre-defined functions that you can use in your computations. Among them are: trigonometric and inverse trigonometric functions (`sin, cos, asin, acos, atan`, etc.), exponential function (`exp`), logarithm functions (`log` is log to base e, while `log10` and `log2` are logs to bases 10 or 2.)

Matlab also has many other more sophisticated functions for solving linear equations, getting eigenvalues of matrices, solving differential equations or calculating integrals numerically. Help button is your friend!

3. Vectors and Matrices

Matlab is most used to work with matrices and vectors. Vectors are either row vectors or column vectors and it is usually important to be clear as to what kind of vector you mean.

➔ To create a row vector enter the name for the vector and the elements of the vector separated by spaces (or commas) surrounded by square brackets.

```
>> A = [1 2 3]
```

➔ To create a column vector, separate each element by a semicolon.

```
>> B = [4;5;6]
```

➔ To enter a matrix, combine the row and column notation.

```
>> C = [1 4 9;8 4 7;2 6 3]

C =

     1     4     9

     8     4     7

     2     6     3

>> G = [4 2 8;5 9 1;4 1 3];
```

➔ To extract parts of a matrix you can use one of the following commands:

- to display (or operate) on a particular element, row, or column of a matrix use its address. The general syntax is `matrixname(row#,column#)`.

```
>> C(2,3)      %Displays the element in the 2nd row and 3rd
column of C.

ans =

     3
```

Note that anything typed after a `%` sign is not evaluated. **Use this feature frequently when writing programs in MATLAB!

```
>> C(:,3) %Displays the 3rd column of C

>> C(2,:) %Displays the 2nd row of C
```

The colon tells MATLAB to include all rows or columns.

- To extract a smaller sized matrix from an existing matrix use the address of the desired elements:

```
>> C(1:2,2:3)  %Displays the first two elements of the 2nd
and third column

ans =

     4    9

     4    7
```

- To extract the main diagonal use the `diag( )` command:

```
>> Cdiag = diag(C)

Cdiag =

     1

     4

     3
```

➔ To delete rows and columns from a matrix use just a pair of square brackets:

```
>> C(:,2)=[]

C =

     1     9

     8     7

     2     3
```

➔ Three special matrices that you'll often use are the zero matrix, the identity matrix and matrixes/vectors of ones.

```
>> D=zeros(3,3) %displays a 3x3 matrix of zeros

>> E=eye(4,4) %displays a 4x4 identity matrix

>> F=ones(3,1) %displays a 3x1 matrix of ones
```

3.1 Manipulating Matrices

The symbols for basic arithmetic operations with matrices are:

`+` for addition—e.g., `>> C+G`

`-` for subtraction—e.g., `>> C-G`

`*` for multiplication—e.g., `>> C*G`

`inv( )` for inverting a matrix:

```
>> Cinv = inv(C)

Cinv =

   -0.1034    0.1448   -0.0276

   -0.0345   -0.0517    0.2241

    0.1379    0.0069   -0.0966
```

`/` or `\` for division—e.g., `>> C/G` or `G\C` for "right" division or "left" division

Note that matrices must be *conformable* for these operations to be defined. Also note that `C/G = C*inv(G)` and that `G\C = inv(G)*C`, which are generally ***not*** the same.

Other useful operators include:

`'` for transposition—e.g., `>> C'`

`det( )` for the determinant—e.g., `>> det(G)`

`trace( )` for the trace—e.g., `>> trace(G)`

`eig( )` for the eigenvalues—e.g., `>> eig(G)`

3.1.1 Concatenation:

MATLAB can easily join matrices together to make a larger matrix:

```
>>A=[3 7 4;5 9 2;4 6 1];
>>B= [A A+12; A*3 A/2]
B =
    3.0000    7.0000    4.0000   15.0000   19.0000   16.0000
    5.0000    9.0000    2.0000   17.0000   21.0000   14.0000
    4.0000    6.0000    1.0000   16.0000   18.0000   13.0000
    9.0000   21.0000   12.0000    1.5000    3.5000    2.0000
   15.0000   27.0000    6.0000    2.5000    4.5000    1.0000
   12.0000   18.0000    3.0000    2.0000    3.0000    0.5000
```

3.1.2 Element-by-Element Operations

Sometimes it's useful to have MATLAB perform an operation on each element of a matrix. For example,

```
>>A^2 %Performs the usual matrix multiplication A*A
>>A.^2 %Squares each element of the matrix A
```

MATLAB can also perform element-by-element multiplication—e.g., `A.*2` – and division—e.g., `A./2` (or equivalently, `2.\A`).

3.2 Solving Linear Equations

X = A\B: Denotes the solution to the matrix equation AX = B.

X = B/A: Denotes the solution to the matrix equation XA = B.

4. Writing Simple Programs

The capabilities of Matlab can be extended through programs written in its own programming language. It provides the standard constructs, such as loops and conditionals; these constructs can be used interactively to reduce the tedium of repetitive tasks, or collected in programs stored in ``m-files" (nothing more than a text file with extension ``.m").

4.1 Script M-files

If you just want to enter in some simple problems for MATLAB to solve, using the *Command* window is fast and easy. But you will often have a long sequence of commands on many variables for MATLAB to evaluate—sometimes for hundreds of repetitions! MATLAB allows you to type your commands in a text file, called a ***script or M-File***, and then have the commands in the *M-file* evaluated just as if they were entered in the *Command* window.

To create an *M-File*, click on the New M-file icon on the MATLAB desktop toolbar, or choose **New/M-File** from the **File** menu. You can use the text window that appears to enter commands. Try entering

```
%M-file example, 10/09/07; written by yourname

num1 = 10

num2 = 13;

num1+num2

A=[1 2 3]
```

Now click the Run icon on the toolbar to execute your commands. MATLAB asks you to save your file before it is run; use the **Save file as:** dialog box to name and save your file. In the CSSCR lab, save your *M-File* to the */temp* folder. (MATLAB may open a dialog box asking about your *Current Directory*, if so, select the option that changes your *Current Directory* to the */temp* folder where you've saved your *M-File*.)

The results of running your *M-File* appear in the MATLAB command window.

4.2 For Loops

A For Loop executes a set of commands a given number of times. Try running this simple program from an M-File:

```
clear; %this clears all variables from the workspace
n=10;
beta=zeros(n,1); %Create an nx1 vector to hold Beta vector

for i=1:n %set # of times to execute the following commands
    beta(i,1)=i+1; %formula for the ith element of beta
end       %end the For loop

beta      %print beta in the command window
```

4.3 While Loops

A While Loop executes a set of commands repeatedly, until a controlling expression is no longer true.

```
clear;
b=0;t=0; %Enter the initial values of variables b and t
while 2^b<200 %Enter the controlling expression
    b=b+1;
    t=t+2^b;  %These two lines are the commands to be executed
end  %End the While Loop
```

4.4 If-Else-End Conditional

The If-Else-End construct evaluates a logical expression and executes a command, or group of commands, based on the value of that expression. Try running this simple program from an M-File:

```
clear;
b=randn;     %pick b from the N(0,1) distribution
```

```
if b>0
    count=1; %The variable count will be equal to 1 if b>0
else
    count=0; %count is zero otherwise
end
count        %show count in the command window
```

These constructs can be used together, or nested within themselves and/or one another, allowing you to write powerful programs in which the results of past calculations affect subsequent operations.

4.5 The Current Directory and Search Path

To use an *M-File* that you have created, MATLAB needs to know where to find it. MATLAB looks on a *Search Path* so you need to make sure that the directory in which you saved your *M-File* is on this path. The easiest way to do this is to make the directory in which you've saved your *M-File* the *Current Directory.*

Alternatively, you can put that directory on the *Search Path* by selecting **Set Path** in **File** menu or by typing the following in the *Command* window:

```
>> path(path, 'directory')
```

For example in the Econ grad computer lab you would type

```
>> path(path, 'z:\')
```
when logged on with your userid.

4.6 Function M-Files

These M-Files differ from the script M-Files you've been working with in that they accept *input arguments* and return *output arguments*. A Function M-File operates on variables contained within its own workspace, which is separate from the workspace you've been accessing from the command line or through script M-Files. To write a function M-File, open a new M-File and type the following:

```
function x=test1(a,B) %x is the output argument from
                      %the function "test1"; a and b
```

```
                        %are its input arguments

x=a'*B*a+23  %define x as a function of the input arguments
```

Now save your function M-File—you must save your file with **exactly** the same name you gave the function—*test1.m* in this example (make sure you save it to either the current directory or a directory that is on the *search path*). In the command window, enter any conformable "a" and "B" (note: you do not need to name them "a" and "B"—the function will operate on any conformable inputs that you give it). In the command window type the following:

```
>>c=[2;3];

>>D=[3 6;8 4];  %c and D are the input arguments for test1
```

Now call your function by typing

```
>>test1(c,D);

x =

   155
```

5. Plotting Graphs

MATLAB has powerful graphing features. To get started plotting graphs, try this example of a simple 2-D graph.

```
n=150;

h=1/n;

x=0:h:1;         %this means x starts at 0, and goes to 1

                 %by increments of h

y=sin(2*pi*x);

plot(x,y)        %The plot command

title('Graph of y=sin(2pix)')        %Give the graph a title

xlabel('x axis'); ylabel('y axis')  %Label the axes
```

When the graph appears, use the menu and toolbar to modify it and/or copy and paste it into a Word document.

6. Working With Data

You will certainly need to export your results from MATLAB or import data from other sources. Let's start by saving some of what we did until now. The basic command format is the `save` command followed by the name of the file to create. This is followed by a list of the variables which are to be saved in the file. To create an ASCII file which can be read by a spreadsheet the list of variables is followed by the command `-ascii`.

`>> save exampleFile A B C` This saves the A, B, and C variables in a MATLAB data file (with .mat extension) in the current directory.

`>> save C:/exampleFileAll -ascii` This saves all variables in the Workspace in a ASCII file in the directory C.

Now let's clear all variables in the workspace:

```
>> clear all
```

Loading data from a .mat file is easy. Just use `'load'` instead of the command `'save'`.

```
>> load exampleFile A B
```

Note that you can choose which variables to load. This command loads only A and B, if you wanted to load all variables you could simply write "`load exampleFile`".

A quick method of importing text or binary data from a file (e.g., Excel files) is to use the MATLAB Import Wizard. Open the Import Wizard by selecting **File -> Import Data** at the Command Window.

Specify or browse for the file containing the data you want to import and you will see a preview of what the file contains. Select the data you want and click **Finish**. (For more information, see Help file for 'Importing Text Data')