MatLab Programming – Lesson 1

(Work quickly to get through the material; you can always go back if you forget something.)

1) Log into your computer and open MatLab…

Arithmetic

You can use MatLab as a calculator in the command window at the command line.

Type the following commands into the command line and press [enter] to see the result.

   a) 2 + 2
   b) 3*(3 + 4)  MatLab knows order of operations.
   c) ans^2  The previous answer can be used.
   d) sqrt(4)  Square root function.
   e) exp(12)  Exponential function.
   f) log(10)  Logarithms are in base e.
   g) sin(pi)  ‘pi’ is recognized, trig in radians.
   h) format long  You can hit return after some statements to edit settings.
      pi
      format short  All numbers are stored with double precision, but
      pi you can change their appearance.
   i) 8^(1/3)
   j) exp(i*2.1 + .2)  ‘i’ recognized as imaginary number.
   k) [up arrow, repeat] You can scroll through previous commands.
   l) e[up arrow]  You can scroll through previous commands that
      start with ‘e’.
   m) angle(exp(i*2.1 + .2))  You can get angle of complex number.
   n) angle(exp(i*2.1 + .2))*180/pi  You can get angle of complex
      number in degrees.
   o) abs(exp(i*2.1 + .2))  You can get modulus of complex number.

¿Practice: What is the square root of -1?

Write you answer here. Compare with your neighbors.
**Variables**

MatLab allows you to store case-sensitive variables.

_Type the following commands into the command line and press [enter] to see the result._

a)  
m  
’m’ has not been assigned a value.

b)  
m = 10  
’m’ now equals 10.

c)  
m = 11;  
Semicolon stops the display of an action on the screen.

d)  
m  
Now it has a value so typing it returns the value.

e)  
M  
Capitalization matters.

f)  
M = m^m  
Defining M.

g)  
clear m  
Deletes assigned value of m.

h)  
m  
No value assigned.

i)  
m = ‘text string’  
You can also store strings of text in variables.

j)  
clear all  
Erases any variables you have assigned (useful at the beginning of a program).

**Symbolic**  
_(SKIP THIS SECTION FOR NOW, GO TO NEXT PAGE.)_

MatLab can perform symbolic manipulation & solving using Maple. We won’t explore this much because you will learn Mathematica. You can find more info at [http://www.mathworks.com/products/symbolic/](http://www.mathworks.com/products/symbolic/).

_Type the following commands into the command line and press [enter] to see the result._

a)  
syms x  
‘x’ is now a symbolic variable. If this command does not work, then the symbolic package is not installed. Skip to part d) and learn to open the help menu and use its index.

b)  
S = 2*x^2 – 7*x – 4

c)  
solve(S)

d)  
_Open the help menu (’?’ button) and type ’solve > [enter]’ in the index. Click on ‘Solving equations’. I spend a lot of time looking around in the help menu when programming._

e)  
clear all
Linear Algebra

Type the following commands into the command line and press [enter] to see the result.

a) \[ A = [ 3,2,1 ; 2,2,2 ; 1,-2,4 ] \] Create the square matrix ‘A’.

b) \[ A(3,2) \] Reference the number in the 3\(^{rd}\) row, 2\(^{nd}\) column.

c) \[ A(1,1:2) \] Reference only part of the matrix.

d) \[ B = A(1,1:end) \] Create a row vector from part of ‘A’.

e) \[ 2*A \] Multiply each element of ‘A’ by 2.

f) \[ A*A \] Perform matrix multiplication.

g) \[ B*A \] A 1x3 matrix times a 3x3 matrix.

h) \[ A*B \] A 3x3 matrix times a 1x3 matrix (does not work).

i) \[ A*\text{transpose}(B) \] A 3x3 matrix times a 3x1 matrix.

j) \[ A*\text{B’} \] A quicker way to take the transpose.

k) \[ A = \text{eye}(3) \] Create the square 3x3 identity matrix.

l) \[ B = \text{ones}(3,4) \] Create the 3x4 matrix of all ones.

m) \[ \text{zeros}(2,5) \] Create the 2x5 matrix of all zeros.

n) \[ C = [1,2,3] \] Create a row vector.

o) \[ D = [4;5;6] \] Create a column vector.

p) \[ E = [ [D], [D], [C]' ] \] Create a matrix from smaller matrices.

q) \[ A.*E \] Multiply matrices of same dimension element to element (different from matrix multiplication, \(A*E\)).

r) \[ \text{eig}(A) \] Solving for eigenvalues (important in quantum mechanics).

s) \[ \text{inv}(A) \] Taking the inverse of a matrix.

¿Worked example:

Suppose you wanted to solve the following system of equations with 3 equations for 3 variables:
\[ \begin{align*}
3x + 2y - z &= 7 \\
2x - y + 3z &= 2 \\
-x - y - z &= 4 \\
\end{align*} \]

Define “\( R = [ 3,2,-1,7 ; 2,-1,3,2 ; -1,-1,-1,4 ] \)” and try “\text{rref}(R)”’. This performs “row-reduced echelon form” solving on the system of equations.

Write you answer here for \( x, y, \) and \( z \). Compare with your neighbors.
Arrays

Aside from performing linear algebra tasks, matrices are a great way to organize and store data.

Type the following commands into the command line and press [enter] to see the result.

a) \[ F = [0:2:10] \]  
   Create an array stepping by two.

b) \[ [0 : \text{pi} : 100] \]  
   Step by non-integer works, but won’t reach last value.

c) \[ G = \text{linspace}(0,100,101) \]  
   Create 101 numbers between 0 and 100: i.e. 0,1,2,...,100.

d) \[ H = \text{logspace}(0,2,3) \]  
   Logarithmic stepping.

e) \[ x = \text{linspace}(0, 2*\text{pi}, 30) \]  
   Create an array of 30 numbers between 0 and \(2\pi\). The 1st number is zero, the last is exactly \(2\pi\).

f) \[ x(15) \]  
   Reference the 15th element of the x-array.

g) \[ x(\text{end}) \]  
   Reference the last element of the x-array.

h) \[ y = \sin(x) \]  
   Take the sine of all numbers in ‘x’.

This last entry demonstrates the power of arrays. You may operate on a set of data rather than one at a time.

¿Practice:

Find the natural logarithm for all the numbers between 0 and 100 using arrays.

Write which number has a natural log closest to 3.0?

Check with your neighbors.
**Simple (not really) 2-D Plotting**

A major reason to use MatLab is for the built-in graphics. Graphics in MatLab is not always easy, but then good graphics is not easy with any software.

*Type the following commands into the command line and press [enter] to see the result.*

a)  
\[ x = \text{linspace}(0, 2*\pi, 30) \]  
*Create the domain.*

b)  
\[ y = \sin(x) \]  
*Create the range of the \( \sin \) function over the domain, \( x \).*

c)  
\[ \text{plot}(x, y) \]  
*Create a graph.*

d)  
\[ \text{plot}(x, y, '.') \]  
*Create graph with data points.*

e)  
\[ \text{plot}(x, y, 'ro') \]  
*Create graph with red data circles.*

f)  
\[ \text{plot}(x, y, 'g-') \]  
*Plot green line.*

g)  
\[ \text{plot}(x, y, 'b:') \]  
*Dotted blue line.*

h)  
\[ \text{plot}(x, y, 'k--') \]  
*Dashed black line.*

i)  
\[ \text{plot}(x, y, 'r*-') \]

j)  
\[ \text{plot}(x, y, 'r+', 'markersize', 16) \]  
*Make the data points larger.*

k)  
\[ \text{plot}(x, y, 'r-', 'linewidth', 5) \]  
*Once you make a graph, you can edit features from the command line.*

hold on  
*You can overlay plots on the same graph.*

plot(x, y, 'bx', 'markersize', 14)  
*You can change the data size.*

hold off  
*Turn off the overlay.*

l) close  
*Be sure to save file (as jpg) before closing if you like the result and want to save it.*
Now let’s get a little fancy.

m) plot(x, y, ‘o’, ‘markersize’, 12, ‘markerfacecolor’, ’g’, ‘markeredgecolor’, ‘r’)

n) title(‘My Graph’)  
   Adds title.

o) title(‘My Graph’, ‘fontsize’, 16, ‘fontname’, ‘times’)
   Control title appearance.

p) axis([0, 2*pi, -1.5, 1.5])  
   Rescale axes.

q) xlabel(‘\theta’, ‘fontsize’, 16, ‘color’, ‘g’)  
   Label x-axis.

r) ylabel(‘\sin(\theta)’, ‘fontsize’, 16, ‘color’, [.8, .6, .3])
   Colors can be fine-tuned with 3-vector codes.

s) text(3.2, .1, ‘\leftarrow \sin(\pi)=0!’, ‘fontsize’, 14, ‘color’, ‘r’, ‘fontname’, ‘times’)
   You can add comments!

Lastly, the axes are still so small. It turns out that the graph is built upon the axes. So if you need the axes larger, you have to do this at the beginning.

   t) close

u) axes(‘fontsize’, 20, ‘fontname’, ‘arial’)

v) repeat steps m through s (copy/paste)

¿Practice:

While doing some physics homework, you find that a capacitor discharges exponentially with 
\[ Q = Q_o \exp(-t/\tau), \]
where \( Q_o = 3 \) coulombs and \( \tau = 2 \) seconds.

Create an awesome graph demonstrating this relationship from \( t = 0 \) to \( 3\tau \) using 100 points in your time domain.

Show it to your neighbor to impress them. Be sure to add insulting comments that they are sure to enjoy!
**Introduction to Scripting**

MatLab scripts are called m-files and they are the programs you make to run in MatLab. They are simple text files that can also be read by editors like MSWord. MatLab has a built in editor that is very useful for debugging.

*Type the following commands into the command line and press [enter] to see the result.*

a) *Use a file manager to create a directory called something like ‘reu_matlab’ on your thumb drive or your computer, OR use the MatLab command window like Unix to make and change directories.*

b) *Change the MatLab path to this directory (look for […] button at the top of the command window).*

c) *Open a new M-file with ‘file > new > M-file’.*

d) *Save this (initially blank) file as ‘intro_script.m’. M-files names must end in ‘.m’, can’t contain symbols and cannot be named the same thing as common functions.*

e) *Program the following script line by line into your M-file:*

   ```matlab
   close all; clear all; clc; format short; % Semicolons allow many commands on 1 line.
   x = linspace(-7, 7, 100000);   % A huge domain.
   y = sin(x) ./ x;   % The famous sinc function.
   plot(x,y);
   ```

   *Now hit ‘F5’. This will simultaneously run the script and resave the script. Note that the semicolons are necessary because otherwise the output prints (painfully slowly) to the screen. Try removing the semicolon from x = linspace(-7, 7, 100,000) and re-run the script. 100,000 print-outs, yikes! The ‘%’ allows comments to be added.*

¿Practice:

*From before:* While doing some physics homework, you find that a capacitor discharges exponentially with \( Q = Q_o \exp(-t/\tau) \), where \( Q_o = 3 \) coulombs and \( \tau = 2 \) seconds.

Make an M-file that graphs this relationship from \( t = 0 \) to \( 3\tau \) using 100 points in your time domain. Use color and lines (you don’t want to plot data points for \( 100,000 \) data points!).

Don’t forget to re-run your program with ‘F5’ in case you made typos.

MatLab will happily tell you where any problem is.
More on Graphing
You can use graph in 3-D.

a) Change the MatLab path to your ‘reu_matlab’ directory if not already done.
b) Open a new M-file with ‘file > new > M-file’.
c) Copy the following script into your M-file. Hit ‘F5’ to run.

close all;
clear all;
clc;
format short;

x = linspace(-2, 2, 30);
y = linspace(-7, 7, 30);

[X,Y] = meshgrid(x,y);

z = (X.^2).*(sin(Y));

surf(z);

It takes a lot of time to learn (and forget and then relearn) all the graphical programming
techniques for MatLab. Little details aren’t always even in the help menu! But if you need just a single
graph for a homework or research publication, then maybe it is best to give up and use the tools in the
figure menu.

¿Practice:
Explore the option menus of the figure window with your mouse. You can rotate, rescale, relabel,
etc. Use the menu to add a title to your graph, rotate it, label the axes, etc.

Save your finished graph as a jpeg, print it out, and show it to your neighbor. Don’t forget to add
some insults to it that your neighbors will find quite witty.
User input
You can ask the user to input stuff. *Copy the following script into a new M-file.*

close all;
clear all;
clc;
format short;

% This is a comment.

{%
This is a
many-lined comment
%}

user_response = input('What is your name? 
','s');
% \n is the skip-to-next-line statement.
% ‘s’ inputs data as string.

disp(' ');
disp('I want to tell you something.');

disp(' ');
disp(['I want to tell you something.']);

disp(' ');
disp(['Your name is ',user_response]);
disp(' ');

*Note that the brackets can be used to make a ‘matrix’ of strings.*

Practice:
Make a program that asks your neighbor a stupid question, and then berates them for their response.
**If-then logic**

Crucial for any real program. *Copy the following script into a new M-file.*

```matlab
close all;
clear all;
clc;
format short;

user_response = input('Are you happy? (1' for Y, 2' for N.) \n');
% If you want to single quote to appear as text, you
% have to type two of them!

if user_response == 1,
    disp('');
    disp('Oh brother!');
    disp('');
elseif user_response == 2,
    disp('');
    disp('Boo hoo.');
    disp('');
else,
    disp('');
    disp('Wrong answer!');
    disp('');
end;

user_response_2 = input('Type a whole number between 1 and 100 \n');

delta = 1e-9;

if abs(mod(user_response_2,2)) < delta,
    disp('Even');
end; % You don't need else statements.
% This is a safer way to test the value of a number
% than 'if mod(variable,2) == 2' because there
% can be some tiny error that makes the value
% different than a whole number.

if abs(mod(user_response_2,2) - 1) < delta,
    disp('Odd');
end;

Practice:
Create a program that asks the user for a number and checks to see if the number is divisible by 3.
Check with your neighbor.
**Numbered Loops**
Crucial for any real program. *Copy the following script into a new M-file.*

```matlab
close all;
clear all;
clc;
format short;
sum_var = 0;

for i_loop = 1:100,
    sum_var = sum_var + i_loop;
end;

disp(['The sum of 1 through 100 is ',num2str(sum_var)]);
% 'num2str' converts a number to text (string) for printing.
% Can you guess what 'str2num' does?
```

**Practice:**
Use a numbered loop and an ‘if’ statement to add all the odd numbers together between 1 and 1000. Check with your neighbor.

(END OF LESSON 1)