CSE/NEUBEH 528 Homework 0: Introduction to Matlab

(Practice only: Do not turn in)

Okay, let's begin! Open Matlab by double-clicking the Matlab icon (on MS Windows systems) or typing "matlab" at the prompt (Linux). Remember that you can always get help by typing "helpdesk" or typing "help" or "help *topic*" on the topic you want to get help on. The >> indicates the prompt in the following examples. To quit Matlab, type "quit." To cancel a command, type control-c.

1. Matlab Basics

Elementary Arithmetic

Try adding, subtracting, multiplying, and dividing two numbers. For example, type:

```
>> 323 + 123
```

Note how the result is always stored in a variable called "ans."

(a) Type in an expression involving all four operators, e.g., 9+5/3*6-2. In what order does Matlab evaluate your expression, i.e., is it left-to-right order or do some operators have precedence over others (write down this precedence order if one exists, e.g., + before – before / …)?

Variables

Create your own variables x, y, and z by assigning arbitrary values to them, e.g., by typing "x = 17", "y = -5.5", etc.

(b) Type the above assignment statements again but now type a semicolon (;) at the end of each statement. What does it do?

(c) Type "whos". Now type "who", then "clear", and then "who". What do these commands do? (Remember that you can type "help whos" etc. to get more info on any command you are not sure about)

Vectors

Let's create a row vector. You can enter it manually:

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

Or if you want to do it over a range, say from 1 to 5 in increments of 2, you can enter:

```
>> b = [1:2:5]
```

Suppose you want a column vector instead of a row vector. Type:

```
>> c = [1; 2; 3]
```

Now type:

```
>> a+b
>> a-b
>> a*b
>> a*c

>> b'
>> a*b'
>> a.*b
>> a./b
>> a.^2
```

(d) What is the result of each? What do these operators do?

Matrices

Let's create some matrices. Type:

```
>> D = [1 5; 3 4]
>> E = [2 1 -3; -4 3 -9]
>> Z = zeros(3)
>> O = ones(3)
>> I = eye(2)
>> R = rand(3,2)
```

Note that matrices are typically denoted by capital letters. To refer to a specific element in a matrix (or vector, which is basically a matrix with only one row/column), say matrix E, use E(*i*,*j*) where *i* is the row and *j* is the column.

(e) Change the element at row 1, column 2 of D to "-1" using "=". Type: "D(:,2)". What does the colon in D(:,2) do?

Now type:

```
>> D - I
>> D*E
>> E*D
>> E'*D
>> inv(D)

>> D*inv(D)
>> inv(I)
>> inv(O)

>> inv(E)

>> pinv(E)
>> E*pinv(E)
```

(f) What are the results of each of the above commands? What do these operators do?

(g) Using what you have learned, solve the following for x and y using matrices and vectors. (If you need more details, see this khanacademy video). Show the equation in its matrix-vector form and solve it in Matlab using matrix/vector operations. (Plug in the x, y, z values to verify your result is correct).

x+4y-z=1
2x-3y+2z=21
-x+2y+z=17

2. Plotting

Try plotting the following sine function (this example is based on the Matlab demo, which you can access by typing “demo” at the prompt). Note that x is simply a vector from 0 to 20 in increments of 0.05.

```
>> x = 0:0.05:5;
>> y = sin(x.^2);
>> plot(x,y);
```

Now enter

```
>> z = cos(x);
>> plot(x,z)
```

Notice that the first graph was deleted. If you want two different windows with two different graphs, you can create a new window by typing “figure” and then using the “plot” command. To go back to a figure you just created, type “figure(*i*)” where *i* is the number identifying the figure (look at the window title to see what figure number a plot is). Use “close(*i*)” to close a figure window.

(a) Plot two functions (make up your own) on the same graph. (Use the “hold” command to do this, type “help hold” for details.) Use a solid line with a circle at each data point for one function and a plain dotted line for the second function. (Use the help command on “plot” for more info.) Label your axes and title your graph. (Use the “xlabel”, “ylabel” and “title” commands rather than the pulldown menus.)

(b) Plot the same two functions as in (a) but one below the other in the same figure window (use the “subplot” command).

3. Scripts, m-files, and Matlab Programming

It can be pretty tiresome to type the same set of commands over and over in Matlab. One way to make your life easier is by making scripts. These are just files (that end in a .m extension) that contain all your commands. Let’s try making one.

Open up a text editor. (I’d recommend Emacs, but use whatever you’d like; Windows Notepad is fine for our purposes.)

Enter all the commands that you used for 2(a) above into a new file. Save the file as *myscript*.m (where *myscript* is whatever you want to name your script) in a directory. Now you need to tell Matlab where you saved the file. You can do this either by changing directories until you find the directory that your script is in (use "cd" to see where you are, "cd *directoryname*" to go into a directory and "cd .." to get out of a directory) or you can use the "path" command to add the directory once and for all. After this is done, to run the script, type its name without the .m, i.e. type *myscript*. If you see your graph plotted, you've mastered writing and running m-files!

Conditional Statements

Scripts simplify things, but you can make scripts powerful using programming concepts such as conditional statements. For example:

```
if <condition>,
      <statement1>
      . . . etc.
else
      <statement2>
       . . . etc.
end
```

Here, if the condition is true, <statement1> . . . will be executed, if the condition was false, then only <statement2> will be executed.
Type in "help if" for more variations on the "if" statement. You may also nest your if statements, as follows.

```
if <condition1>
      <statement1>
      . . . etc.
      if <condition2>
            <statement2>
            . . . etc.
      end
end
```

Don't forget to end both if's!

(a) Suppose we are given two distinct numbers stored in the variables x and y. Create a new file named "greater.m". Enter into this file a single "if-else" statement which compares x with y and stores the greater of the two numbers in the new variable "result". Save your file and test it by entering values for x and y in Matlab, and running the file by entering "greater". Check the value of result by typing "result".
(b) This time, write the commands to find the "lesser" of two (possibly equal) numbers x and y, and save your code in a file named "lesser.m". However, this time, display the result immediately (i.e., without having the user type "result" after the m-file has executed). Also, if x and y are equal, display a message saying so to the screen. (Hints: use "= =" for testing equality, not "=" (why?); also use "disp")

Loops

Loops can make repetitive tasks a lot easier. The most basic kind of loop is the "while" loop.

```
while <condition>,
      <statement1>
       . . . etc.
end
```

Say we want to find the first multiple of 13 that is greater than 50. One way to do so would be to keep finding multiples of 13 until we're past 50.

```
n = 0;
while n <= 50,
n = n+13;
end
disp(n);
```

Here, we start off with n as 0, and 13 is repeatedly added until n is greater than or equal to 50.

(c) Another kind of loop is the "for" loop. Type "help for" and read the details. We have already seen how Matlab allows two appropriately-sized matrices A and B to be multiplied directly by simply entering "A*B". Using "for" loops, write your own version of Matlab code for multiplying two given matrices A and B, each of size N x N, where N is a variable denoting matrix size. Store your result A*B in the matrix "Result" and output this matrix after your "for" loops. Save your code in a file called "mymultiply.m". Now test your code by typing:

```
>> N = 5
>> A = rand(N)
>> B = eye(N)
>> mymultiply      **should output Result = A here
>> B = rand(N)
>> A*B
>> mymultiply      **should output Result = A*B here
```

Try out some really large values for N with corresponding random matrices A and B; try running your function as well as Matlab's "A*B" (Make sure you put a ";" after the statements "A = rand(N)", "B = rand(N)", etc. and do not output the Result matrix – these matrices will all be too big (and boring) to display!). Does Matlab's version of matrix-multiply run faster or take the same time as your version?

4. Creating Functions

Finally, let's write some real programs that can take in some input parameters and return results as a function of the inputs. Matlab has some built-in functions that we've already used, such as sin(x), rand(N), etc. Let's make some of our own.
Type "help function" and read the documentation and examples on how to create functions in Matlab.

(a) For your m-file "greater.m" above, you had to type x = *some-value* and y = *some-other-value* each time we used it. It would be easier if we could type greater(*some-value*, *some-other-value*), wouldn't it? Create a new file "bigger.m" such that it takes in as input "x" and "y", and returns the greater of the two numbers in the variable "result". Use the percent sign (%) is for commenting what each line of code does. Matlab ignores everything after the %. You can use comments anywhere in your code to clarify things. The comment immediately following the first line of a function is displayed whenever a user types “help greater”. Use this to document what your function (“bigger”) does, who wrote it, when, etc.

Test your function by typing “bigger(a,b)” where a and b are arbitrary numbers.

(b) Convert “mymultiply.m” to a function “mult(X,Y)” where X and Y are square matrices of the same size (Hint: Use “size” to figure out N). If the matrices are not square or are not of equal size, display an error message and stop; otherwise, proceed as in “mymultiply”. Comment your code using “%”. Test your function by creating different matrices A and B, and typing “mult(A,B)”. Compare your output to that of “A*B”.