MATLAB Functions

What is a MATLAB function?

A MATLAB "function" is a MATLAB program that performs a sequence of operations specified in a text file (called an m-file because it must be saved with a file extension of *.m). A function accepts one or more MATLAB variables as inputs, operates on them in some way, and then returns one or more MATLAB variables as outputs and may also generate plots, etc. (sometimes a function doesn't return any output variables but instead just generates plots, etc.).

How do I Create a new MATLAB function?

Since an m-file is nothing more than a text file it *can* be created using any text editor – however, MATLAB provides its own editor that provides some particular features that are useful when writing/editing functions.

To open a new m-file: In the MATLAB command window, go to FILE on the toolbar, select NEW, then select M-FILE. This opens the MATLAB editor/debugger and gives an empty file in which you can create whatever m-file you want.

What do I have to put on the First Line of a MATLAB function?

< The 1st line of a function ***must*** contain the "function definition," which has a *general* structure like this (see also the specific example below)[1]:

```
function [Out_1,Out_2,…,Out_N] = function_name(In_1,In_2,…,In_M)
```

where Out_1,Out_2,…,Out_N are the N output variables and In_1,In_2,…,In_M are the M input variables;

< If there is only a single output variable use:

```
function Out_1 = function_name(In_1,In_2,…,In_M)
```

< If there is no output variable use:

```
function function_name(In_1,In_2,…,In_M)
```

What do I have to put after the 1st line?

After the first line, you just put a sequence of MATLAB commands – with one command per line – just like you are computing in MATLAB in the command line environment. This sequence of commands is what makes up your program – you perform computations using the input variables and other variables you create within the function and in doing so, you create the output variables you desire.

[1] Code in an m-file will be shown here using the Arial font, e.g., `x = cos(0.1*pi*n)`

How do I use the input variables in a MATLAB function?

When you are writing the lines that make up your function you can use the names of the input variables defined in the first line just like they are previously created variables. So if In_1 is one of the input variables you could then do something like this:

```
y=In_1.^2;
```

This takes the values in In_1, squares them, and assigns the result to the variable y.

Note: putting a semicolon at the end of an expression stops the display of the result – a good idea unless you really WANT to see the result (sometimes useful when debugging or verifying a function).

How do I make the output variables in a MATLAB function?

On any line in your function you can assign any result you compute to any one of the output variables specified. For example:

```
Out_1=cos(y);
```

will compute the cosine of the values in the variable y and then assigns the result to the variable Out_1, which will then be output by the function (assuming that Out_1 was specified as an output variable name).

How do I Save a MATLAB function?

Once you have finished writing your function you have to save it as an m-file before you can use it. This is done in the same way you save a file in any other application:

- go to FILE, and SAVE.
- type in the name that you want to use
  - it is best to always use the "function name" as the "file name"
  - you don't need to explicitly specify the file type as *.m
- navigate to the folder where you want to save the function file
  - see below for more details on "Where to Save an M-File?"
- click on SAVE

Where to Save an M-File?

It doesn't really matter where you store it… BUT when you want to use it, it needs to be somewhere in "MATLAB's path"or should be in MATLAB's present working directory (PWD)

- the path specifies all the folders where MATLAB will look for a function's file when the function is run
- the PWD specifies a single folder that MATLAB considers its primary folder for storing things – it is generally advisable to specify an appropriate PWD each time you start up MATLAB and specify it to be wherever you have the m-files you are working on
  - Click on FILE, click on SET PATH, click on BROWSE, navigate to the folder you want as PWD and click on it, and then click OK

How do I Run an M-File?

Once you have a function saved as an m-file with a name the same as the function name and in a folder that is either the PWD or is in MATLAB's path, you can run it from the command line:

```
>> [Out_1,Out_2,…,Out_N] = function_name(In_1,In_2,…,In_M)
```

How do I Test an M-File?

Once you have a file written you need to test it. When you try to run it the first time there is a good chance that it will have some syntax error that will cause its operation to terminate – MATLAB will tell you on what line the operation was stopped, so you can focus immediately on somewhere further along in the function and will give you some idea if what the problem is. Just keep at it and you'll eventually get it to run, but here is a tip:

- open the m-file that you are debugging
- click on DEBUG on the toolbar at top
- click on STOP IF ERROR
    - note: there some other options that can be useful, so try them out
- Now when you run your function and it encounters an error, it will stop and will put you into a mode that will allow you to view all the variables at the point at which the error occurred
    - When you are stopped (i.e., "in debug mode") you have a different prompt than MATLAB's standard >> prompt
    - To quit from the debug mode click on DEBUG and the click on QUIT DEBUGGING
- When you are all done using this feature you can turn it off by clicking on DEBUG on the toolbar and then clicking on STOP IF ERROR (which should be checked to indicate that it was turned on)
    - Then you are returned to having the standard prompt

Then, once you have it running without any syntax errors or warnings, you need to test it to verify that it really does what you intended it to do. Obviously, for simple functions you may be able to verify that it works by running a few examples and checking that the outputs are what you expect. Usually you need to do quite a few test cases to ensure that it is working correct. For more complex functions (or when you discover that the outputs don't match what you expect) you may want to check some of the intermediate results that you function computes to verify that they are working properly. To do this you set "breakpoints" that stop the operation of the function at a specified line and allow you then view from the command:

- open the m-file that you are debugging
- put the cursor in the line at which you want to set a breakpoint
- click on DEBUG on the toolbar at top and then click SET/CLEAR BREAKPOINT

- Now when you run your function, it will stop at that line and will put you into a mode that will allow you to view all the variables at that point
- When you are stopped (i.e., "in debug mode") you have a different prompt than MATLAB's standard >> prompt
    - When you are all done using this feature you can turn it off by repeating the process used to set the breakpoint

Once you have it stopped, any variable that has been computed up to that point can be inspected (plot it, look at its values, check its size, check if it is row or column, etc. You can even modify a variable's contents to correct a problem). Note that the ONLY variable you have access to are those that have been created inside the function or have been passed to it via the input arguments.

Note that you can set multiple breakpoints at a time – once you have stopped at the first one you can click on DEBUG and then click on CONTINUE and it will pick up execution immediately where it left off (but with the modified variables if you changed anything).

Note also that once you have a function stopped in debug mode you can "single-step" through the next several lines: click on DEBUG and click on SINGLE STEP.

Comments on Programming Style

1. In many ways, programming in MATLAB is a lot like programming in C, but there are some significant differences. Most notably, MATLAB can operate directly on vectors and matrices whereas in C you must operate directly on individual elements of an array. Because of this, loops are MUCH less common in MATLAB than they are in C: in C, if you want to add two vectors you have to loop over the elements in the vectors, adding an element to an element in each iteration of the loop; in MATLAB, you just issue a command to add the two vectors together and the vector addition of all the elements is done with this single command.

2. The "comment symbol" in MATLAB is %. Anything that occurs after a % on a line is considered to be comments.

3. It is often helpful to put several comment lines right after the function definition line. These comments explain what the function does, what the inputs and outputs are, and how to call the function.

4. Putting in lots of comments helps you and others understand what the function does – from a grading point of view you will have a higher probability of getting full credit if you write comments that tell me what your code is doing

An Example

Here is an example of a MATLAB function. Suppose you would like to write a general routine that will compute the value of some arbitrary polynomial at an arbitrary set of equally-spaced x

values. This turns out to be something that is best done using loops (or at least, I couldn't find a clever way to avoid loops – this is good because it at least gives an example of how to do loops). The function might look like this:

```
function [y,x] = poly_equi(power,coeffs,xmin,xmax,del_x)
%
% USAGE: [y,x] = poly_equi(power,coeffs,xmin,xmax,del_x);
%
% Inputs: power = an integer specifying the highest power of the polynomial
%             coeffs = a row vector of the polynomial's coefficients
%             xmin = the desired minimum x value for evaluation
%             xmax = the desired maximum x value for evaluation
%             del_x = the desired spacing between the x values
%
% Output: x = a row vector of the x values
%         y = a row vector of the y values
%
% Operation: if coeffs = [C_0 C_1 C_2 ... C_p], where p = power, this
function computes
%   In equation form, not in matlab syntax:   y = C_0 + C_1x + C_2x^2 + ... +
C_px^p


% First, generate the x values as a row vector
x = xmin:del_x:xmax;  % this uses MATLAB's colon operator to generate
                      % equally spaced points at a spacing of del_x

for n=0:power
   % Loop through the powers in the polynomial
   % and compute a row vector for each of the terms in the polynomial
   % and put that row vectro into each row of matrix X
   X(n+1,:)=coeffs(n+1)*x.^n;   % note use of "n+1" for indexing - matlab
can't index using "0"
end
%  You now have a matrix X who's nth row is C_n*x.^n
%  The x values run across this matrix, the power value runs down this matrix

%  Now to create the values of y we have to add the rows together:
if power>0
   y=sum(X);  % when matlab's sum function is applied to a matrix it sums
down the
             % columns to give a row vector
else
   y=X;   % if power=0 there is only a single row in X, so no need to sum
end
```

Now, to run this function we could do this:

```
» [y_0,x_0] = poly_equi(0,3,-5.1,5.0,0.2);
» [y_1,x_1] = poly_equi(1,[3 2],-5.1,5.0,0.2);
» [y_2,x_2] = poly_equi(2,[3 2 2],-5.1,5.0,0.2);
» subplot(3,1,1)
» plot(x_0,y_0)
» xlabel('x values')
```

```
» ylabel('y values')
» subplot(3,1,2)
» plot(x_1,y_1)
» xlabel('x values')
» ylabel('y values')
» subplot(3,1,3)
» plot(x_2,y_2)
» xlabel('x values')
» ylabel('y values')
```

Note that we ran the function three times – each time we put in different values for the input variables and called the output variables something different each time so that we could store the three different results. We then plotted the results on three different subplots and labeled the axes.

ALWAYS LABEL AXES!!!!! ALWAYS LABEL AXES!!!!!
ALWAYS LABEL AXES!!!!! ALWAYS LABEL AXES!!!!!

And label the UNITS of the values if it makes sense!!!!!!