USING THE EDITOR AND CREATING PROGRAMS AND FUNCTIONS

% Programs are kept in an “m-file” which is a series of commands kept in the file that is executed from MATLAB by typing the program (file) name from the command prompt “>>”. We construct the program in a special utility program known as the **editor**.. Editors can also be word processing packages set to the non-document (or text) modes (ie. ASCII code).
% Functions which you create are called user-written functions. As you work in the field you will need special functions that are useful to you. These functions are called in your programs the same as you would any MATLAB intrinsic function. They are created with the Editor and saved as m-files but their construction needs special rules. Functions are covered in more detail latter in this workbook.

PROGRAMS (CALLED SCRIPT FILES)

The editor we will be using is built into current versions of MATLAB and is located in our windows environment.(EARLIER VERSIONS THE FOLLOWING IS A BIT DIFFERENT).
% To invoke the editor we click on **NEW SCRIPT located on the left top menu and** a separate window open with an “untitled” heading. We construct the various commands line by line in this window and when we **save** it we give it a name like “test” and test.m will be saved. When we type the script file name in the Command window ie in this case >>test
All our commands in the file are executed line by line.
.. We can also use the **print** option from a menu or the mouse (right click) to put our work on paper or pdf file if available on your system. Don’t forget after you construct and finally debug your m-files to **save** them.

% You will have to provide a **name for the program file** since the default is untitled.m. MATLAB program files have regular filenames and by default are provided with the extension '**.m**'. Be careful in some versions when you use the “Save As” option since if in the filename you drop the “m” ,MATLAB will just show your choice of the name with two extensions, the m file and mat file (data file covered later).
Names like 'lab8.m' or 'radar.m' or 'density.m' are typical types of names. There are two types of 'm' files in MATLAB as mentioned, one is for programs and is known as a 'script' file, the other is for functions that we create and is known as a 'function' file (**These function types will be covered in more detail later).**
% Programs and functions you create, as well as, data files reside in a folder which is called the “**current folder”** and is the default option for the Layout, the current folder appears as a window showing the names of your files usually on the left of the command window.

SUMMARY OF CONSTRUCTING MATLAB SCRIPT PROGRAM FILES

% If we just enter commands at the MATLAB prompt, as we did above, we call that process **interactive**. If the task at hand gets long and involved then we can use the editor to construct a list of commands in a m-file. Files. After we write up each of the MATLAB commands in the file, and save the file for example, 'torque.m' from the editor, we get the program to run in the MATLAB environment by simply typing the **'filename'**, for example.
>> torque

% This will activate the 'torque' file and MATLAB will execute it one line at a time and undertake the action of each command in the script file.

SAMPLE SCRIPT M-FILE OR PROGRAM FILE)

% Invoke the editor by the “New Script” Icon and type the following lines in the Editor and a save the file with the name **first if you have not done so in class.**

```
%first  demo file called “fun”
% This is my first script file.
u=input('Please enter upper range for the graph?  ');
tint = u/100;
t=0:tint:u;
%note point by point vector operator   .*   since we have two row vectors
z=exp(-2*t).*(7*sin(6*pi*t));
plot(t,z)
title('Darth Veda warp engine');
```

% Here we have introduced the **input function (details described below)** which will wait for a value after the program is invoked and put that value in the variable u and even prompt the user of the program with some statement as to what to give.
% Note the use of the value of u variable in “tint” and finally in the t range command used to generate a sine curve. Also note we added a title before the graphics window opened.
%**close the editor** after you save the file with the name “fun”
%Start the script file by typing its name in the command window!
>> fun

% What happens next is that MATLAB executes each command which we now describe. The first line starting with a '%' symbol is ignored by MATLAB and permits us to add **comments** to our programs. The second line contains the new command '**input**' which has several functions and is very useful for script files. First, the input command expects a number to be typed at the computer which is followed by the 'enter key'. The command causes that number to be assigned to a variable, 'u' in this case. Second, the input command prompts the user with the string of characters that are placed within the single quotes and parenthesis. This is done so that the user of the program which may not be the original programmer knows what is wanted of them. MATLAB executes this line as follows and the user will respond.

Please enter upper range for the graph? 4   (response was 4)

% The variable u now has the value 4.
The line tint=u/100 assigns the value .04 to the variable 'tint' which is used as the interval in the row vector t defined with the next line.
The statement t=0:tint:u constructs a row vector equivalent to t=0:.04:4
The row vector z is constructed next with the sine function of t as stated. We will next see a plot of the function z(t) with t going from 0 to 4 in steps of the variable “tint” which in this case would be 4 from the 'plot' command.
In addition, our program 'fun.m' also labeled the graph with our name with the 'title' command and displayed the graph in a figure window (we can now **print** the result if we want by clicking print in file menu as before.
%NOTE: You may have to go back to the editor several times to get out any errors you inadvertently put in the file. Be careful on Input.
%Once we have created a program **we will most likely have to return** to it, in order to **correct errors** (if we have closed the editor. Recall Programmers use the term **debug** for the removal and correction of errors). We can invoke the editor by the **‘Open”** (file) icon on the top line of our windows environment and we are given a choice of files to open which includes the most recent one. By choosing a file the editor automatically opens with the file. You can also later run your script even if you clear the memory since MATLAB will know the name of your program which resides in the “current folder”. As you construct your own programs and functions you are building a library of files in your MATLAB environment.

RUN “fun” several times with different numbers for the upper level and be sure you understand all the commands in the fun.m program file.

SOME BASIC INPUT/OUTPUT FEATURES OF MATLAB

.% The task of Programming or constructing a 'script' file in MATLAB demands the ability to make a program have intelligence to solve a given situation, as well as, interact with a user and to report the results of calculations in a clear manner. Later we will see MATLAB also handles repetition and selection to give a program intelligence, but the program's intelligence is based on the programmer's skills. The more you program the greater your abilities.

% The **interactive nature** of a program is provided by commands that are generally called **INPUT/OUTPUT.** When we need information for our program we say we are using an **INPUT** process. A simple way is to get values for variables as the program is executing. We saw before that MATLAB provides the '**input()**' command to undertake a combination of 'prompting' the user, as well as, initializing variables from within a script file. After calculations are performed the program will **output** its results. There are several 'OUTPUT' commands at our disposal. You will encounter the use of one output function in this section. Namely the function '**disp()**'. ("display") We will first consider the **disp()** function which can display text within single quotes within the parentheses similar to the input() function prompt or it can be used to display values of variables.

% Displaying text with the disp() function might go something like
>>disp('You are about to undertake the an amazing journey!'); **TRY THIS**
You are about to undertake an amazing journey!

%Note the use of single quote **'** rather than the usual double **" to enclose the text.**
% **Column headings and general output** information are applications of the disp() function. Consider that we are about to reproduce a table of positions, velocities and timings then the necessary column headings with appropriate spacing might be produced with; Though we might have to position the output to match columns we are creating in this simple use of disp()..

>> disp('TIME POSITION VELOCITY');
TIME POSITION VELOCITY
For more information we can get info on this command via "help" which gives us
>> help disp
disp Display array. You get something like what follows….
disp(X) displays the array(**variables, row or column vectors**) without printing the array name. In all other ways it's the same as leaving the semicolon off an expression except that empty arrays don't display. If X is a string, the text is displayed as above.
Illustrations that show some uses of the disp() command function with variables follow:

```
>> x=10
x =    10
>> disp(x)
   10

t=0:1:5;        %  creating a column of output data (recall transpose operator ' .
>> disp(t')
    0
    1
    2
    3
    4
    5
z=40:-10:-10
z =    40    30    20    10     0   -10
>> disp ([t' z'])   %fun with columns
    0   40
    1   30
    2   20
    3   10
    4    0
    5  -10
```

SAVING VARIABLES

% Since we lose our variables when we end a MATLAB session it is a good idea to save not only your graphics work but also the variables and their values if you are working on project that cannot be done quickly.. This is also true when you are working out a problem interactively and will not finish your work at one session at the computer.
% You can save all the variables (and their values) you have been working on by using the **'save'** command. We say we are saving our **workspace**. We can save to a specific file by naming it as in;
>> save sess1

% This last command will write on your library a file named 'sess1.mat' which is in binary format (ie numbers in powers of 2)
%
% A file name for all variables is not necessary if we issue the 'save' command alone as in;
>> save

% MATLAB will create a file called MATLAB.mat

% save can be selective as in
>> **save temp x w z**

% This last command saves only the variables x, w and z to the file 'temp.mat'

% When you close MATLAB you can be sure all the variables you want are saved in the 'mat' file created on your disk.

% Once you have returned you can get all your variables back into memory with the 'load' command.

Example of restoring your variables are:

>> load sess1

% This restores to the active memory from the file called sess1.mat on your disk drive all of your variables saved above that you were working on at the time you saved them. Only the values of the variables and their names are saved not the statements (equations) used!

>> **load temp**
% This will return the variables x, w and z saved above.

>> load
% This will return all variables that were saved with the 'save' command that were stored in the special file 'MATLAB.mat'.
% The 'load' command can also be used as a means to get numerical data into the MATLAB environment from data files a technique we will employ later.

% **CLASS ROOM TASK DO NOW!**
If you have no variables in memory create a few and then 'save' them to a file with a name. Check what is in memory with the 'whos' command. use “clear” issue the 'whos' command and all variables are gone. Then use the 'load' command to retrieve them. Check the result with the 'whos' command.

CONTROLLING THE FLOW OF A PROGRAM

% MATLAB as we saw can be used interactively, most problem solving in engineering and the sciences the user will need to construct stand alone programs(ie **SCRIPT and FUNCTION files**). MATLAB has that ability and one can learn the basic programming skills enough for the user to understand many of the functions and programs used, as of this writing, 2014, PYTHON has become popular in some sciences (like Astrophysics). The reader should learn a lower level programming language which may be used in their ultimate employment. Currently, C++ is highly recommended but an earlier language FORTRAN is still being used in Engineering.
%The key to problem solving in science and engineering is learning to use in a language, logic or **Selection** of paths in a program and the ability of a program to **Repeat (Loop)** statements based on some logical decision. Both **Selection and Repetition** processes which control the flow of the programming are common in the languages mentioned and the basics of each follow. Thus, picking up another language will be a lot easier if you master what follows. Of course, if you have used controls in other languages what follows will look very familiar.

FLOW CONTROL –REPETITION (THE "FOR" LOOP)

% The intelligence of a program as we mentioned above is obtained by two processes known as **SELECTION** and **REPETITION.** Selection is the ability of our program to pick one or more alternative operations from a series of possible outcomes. Selection is based upon logical conditions that arise in our program depending upon the particular problem we are solving. Repetition is simply to repeat commands as often as is needed to obtain a wanted result. Repetition can also depend upon logical conditions within the problem. As also mentioned above all programming languages provide various commands to undertake these processes.
Good programming skills are developed by understanding a broad range of applications for each command structure. Selection and Repetition commands are usually said to **control** the **flow** of our program.
% We now will take up introductory use of commands in each of these area. As we proceed in the text to follow, we will build up more involved use of each of the commands discussed below. The MATLAB prompt will be shown when we are illustrating interactive commands and not when MATLAB is executing the group of commands in a 'script' program file.

% Tirelessly repeating of commands are a major reason computers have become so popular. In technical calculations the process of being able to repeat a number of commands is invaluable, since repeating commands means the computer must return to a previous starting point, we say we are in a **loop.**

The **FOR** Loop

% The MATLAB FOR statement starts the Loop and all commands that follow are repeated a number of times dependent upon an INDEX variable which counts the number of repetitions. Since it has to be known which commands are to be repeated an END statement is used to mark how far the repetition process should be carried out. Consider the following model ran interactively (TRY IT of course one line at a time)!

```
» for i=1:5
    disp ('hello')
  end
hello
hello
hello
hello
hello
```

The index variable 'i' above will start at 1 and end at 5 (counting by 1). In general the MATLAB system will execute the 'for loop' by first setting the index variable to its first value (1 here) then carry out any commands between the 'for' and 'end' statements (known as the **body** of the loop) and then return to the **for** statement and **increment** the variable **i**. After MATLAB executes all

commands between the **for** statement and the **end** statement and returns to the 'for' line the index will change by +1 in this example. When the index reaches its final value the body of the loop is executed for the last time. Control passes to any commands that might follow the **end** statement (especially in a script file) (none in this example) and they would be executed next. **BE CAREFUL**: The value of the variable **i** is **single valued**. The variable i is **not** immediately made into a row vector with values from 1 to 5
as would be the case, with the plain statement i=1:5; The variable **i** changes, one value at a time, each time the computer returns to the **for** statement. This fact means in a script file we might be able to use the variable I, in this case, in some calculations. This is illustrated below after the next example.
% The expression the index variable equals can more involved, as in this next example whose **for** loop is placed on one line. This one line form is constructed by inserting **comma's** so that MATLAB can recognize each separate statement. This form is useful for short purpose loops in our programs.
» n=4;
» for i=1:n,disp('good day'),end
good day
good day
good day
good day

Here, in this latter case, we controlled the number of times the loop executed by setting the last value 'n' before the loop. In a situation that expects a user who may or may not be the programmer we might setup code that resembles the following m file that was called **test1.m** and whose commands (usually called **code**) we can see with the MATLAB **type** command. Enter the editor and setup this test1.m shown by the type command to duplicate what follows. Note the end of each line in the m-file has a semicolon "**;"** marker except for the '**for'** and "**end"** statements. After doing this try **help for** for additional information and advanced examples.

% DEMO file test1.m

```
f=input('Please enter the first number?');
l=input('Please enter the second number?');
sum =0;
for i=f:l
  sum = sum +i;
end
disp ('The sum of all integer values from the first to the last is');
disp(sum);
```

% Note the use of the disp() function to output the value in the variable **sum**.
%also note the use of ";" and the for/end does not use it.
% We now execute this m-file by using its name
» test1

```
 Please enter the first number? 1              %user entered 1
 Please enter the second number? 5             %user entered 5
The sum of all integer values from the first to the last is
    15
```

% The program effectively added up all integers from 1 to 5! We can think of this process as the sum of the series 1+2+3+4+5 .Make sure you understand how this was done since the summation process is very important.

% We next construct a typical **physics solution** similar to what you did above interactively with a **for loop**. The program is in the script file 'vel.m'.N
Which looks like;
» type ACC98

% This program will calculate the distance and velocity of a falling object after 2 seconds of falling
% “a” is the acceleration of gravity= 9.8 m/sec^2 x is distance in meters, v is velocity In m/sec
Here we are treating the variables as scalars (ie one value at a time gets printed.)

```
% class demo acc98
% Here we are creating and displaying one value at a time, not vectors
% for the variables t, x and v but create a vector in the display, a 1 X 3 matrix.
a=9.8
disp('    TIME      DISTANCE  VELOCITY');
for t=0:0.1:2
  x=0.5*a*t^2;
  v=a*t;
  disp([t x v]);
end;
```

% We now execute it and get its outputs!
» vel

```
TIME   DISTANCE   VELOCITY
     0     0     0
    0.1000    0.0490    0.9800
    0.2000    0.1960    1.9600
    0.3000    0.4410    2.9400
    0.4000    0.7840    3.9200
    0.5000    1.2250    4.9000
...                        % All output is not shown.
    1.9000   17.6890   18.6200
    2.0000   19.6000   19.6000
```

% NOTE: the following Points about this last program.
1. The program has a **comment** starting with '%'. MATLAB will ignore all such lines and thus helps us document important information about our program within the file.'
2. All commands that form the body of the loop are indented to show that they belong to the “for” loop. This is important for program readability and should be used by you.
3. We use a **step of 0.1** for our index variable 't' which is the **time** variable for the problem. Thus, we get decimal time values at each pass of the loop since each value of t is increased by 0.1 seconds.

4. We used a trick with the **disp** function to get our three outputs of time, distance (x variable) and velocity (v variable). This involved forcing all our calculated variables at the time of output into **one row vector** with brackets in the disp() function. Ie **disp([t x v]);** The disp() function will normally only output one matrix variable. We just constructed a one dimensional matrix with three components (row vector as before) with our single value variables t, x and v. This trick gives us columns but if you run this program you will see that the output though correct is not as organized as we might like it. Later, we will use the fprintf() function for better organized output.
5. vaiables are scalars and change at each loop.

% **TASK**: **invoke the editor and construct the above file and run it with if you did not do this in class or understand the presentation.**

TASK: in class we do the following.
% We could have improved the program for any time, t, by having the upper limit of the for loop come in as an input. For example adding the statement before the loop as **n=input**('How many seconds for fall'); and then modify the **for** statement **for t=0:0.1:n**

TASK: Do this if you did not do this in class, for a better grasp of the lesson. To quickly re-edit the program just click on its name in the "current folder" window and the editor with the program will appear, Modify it and then save it. Then run it again.

% The next three illustrations, build up in sequence more sophistication and solve **power series**(a way with polynomials to express functions) problems on the way.
First we will develop a program to solve the mathematical problem of factorial. Consider 5! this is equivalent to 1 x 2 x 3 x 4 x 5 by definition. We note this is similar to the sum of the first five numbers we did above. A **for** loop is natural to build up a product series just as it was for the sum of a series as the following program segment shows.
%DEMO FACTORIAL 5! Fact.m run this if you did not do it in class

```
p=1;
u=input('enter the number to get its factorial')
for f=1:u
  p=p*f;
end
disp ('The factorial ! of your number');
disp (u)
disp('is')
disp (p);
```

% This last segment will effectively calculate 5!. 'Play computer' with it to see that it does just that. By changing the upper limit (5) of the loop variable f one can obtain any factorial.

% We next consider obtaining the sum of the famous (you will see this if you have not yet) power series for 1/(1-x) for x < 1 namely.
% power series $1/(1-x) = 1 + x + x^2 + x^3 + ... + x^n$
% In other words the series is a way to get the original function and the more terms you choose to evaluate the more accurate the final numerical answer.
% Since this is series expression is a sum of terms and we have a general expression for the series(x^n), we can use the **for** loop to sum these terms up for a given value of x. The values of x are assumed small. Set up the following program named **power1.m** to solve this problem.

DEMO power1
% power series $1/(1-x) = 1 + x + x^2 + x^3 + ... + x^n$

```
x=0.2;
u=input('enter the number terms of the series you want');
sum = 1;
for n=1:u
  sum = sum + x^n;
end
disp('The approximate value of this function is');
disp(sum);
```

% We note that the variable sum started at 1 (we say we **initialized** the variable sum) in this case so that the number of terms in the series that are added is actually 5 terms even though the loop variable's upper limit is 4**. Play computer with this program to see the changes in the variables as this program is executed and to get a deeper understanding of the action involved**.
For example to better understand the series expansion consider if x =.5 the function 1/(1-.5) =2 But for the first five terms you would get 1.9375 but for 10 terms you get 1.9980. So you see the more terms of a power series used the closer to the true answer. Power series are generated for

much more complex functions than illustrated here.
We combine the basic logic used in the two previous examples by using a **for** loop to calculate the sum of the first n terms of the power series expansion of the exponential function e^x with x =0.1. The power series is given by
$e^x = 1 + x + x^2/2! + x^3/3! + ... + x^n/n!$

Our objective is to build a sum using the general term $x^n/n!$ and note that the variable 'n' is just a counter. The factorial is handled in the loop by noting that the statement f= f.*n, each pass of the loop you are multiplying by a higher number or recreating factorials. The program below will build the appropriate factorial to be divided. The final sum is the value we want.
The program looks like; construct it and call it ex.m
Note: the most precise value from the function exp() is exp(.1) = 1.105170918075648

DEMO ex.m
% only 2 terms of this series is evaluated
% $e^x = 1 + x + x^2/2! + x^3/3! + ... + x^n/n!$
```
format long
sum =1;
x=0.1;
f=1;
for n=1:2
  f = f*n;
  sum = sum + x^n/f.;
end
disp(sum);
```

% We execute the program which gives us a two term power series approximate value for $e^{0.1}$.
» ex we get ans = 1.105000000000000
Note how each variable was initialized as it was and how the factorial and sum terms worked!

LABORATORY TASKS print all to hand in! Source codes and outputs.
Source code can be printed from the editor. Each program counts
As a separate lab grade due to the nature of the assignment. 1=100 .9 =90 etc

first demo repeated here!

```
% This is my first script file.
u=input('Please enter upper range for the graph?  ');
tint = u/100;
t=0:tint:u;
%note point by point vector operator   .*   since we have two row vectors
z=exp(-2*t).*(7*sin(6*pi*t));
plot(t,z)
title('Darth Veda warp engine');
```

17. Use the editor and create the program “first” above. Run the program for u = 6 and print up the graph. Repeat for u=100 and print up the graph **with your name on it**. Print the program also.
Note: we are using row vectors in this program

18.(2 PTS) Use the editor and create the 4 examples above one at a time and run them with understanding, ask the prof if you are not sure what is going on, (script files: acc988,fact,ower1 and ex.)

19.(2 PTS) Write a program that will generate a table converting radians to degrees. Start with 0 radians and end at pi radians in increments of pi/4 radians. Recall pi radians = 180 degrees. Hand in program and output which should be in a two column table with headings. **Use row vectors do not use a “for” loop that uses scalar variables (one value at a time).**

20.(2 PTS) Write a program that asks the user for the upper temperature in Celsius they want to convert to Fahrenheit. Where a table of values that convert from -20 C to the upper limit is produced at an increment of 1 degree Celsius between values. Output both temperatures in a table, as well as have a heading for each column. Add meaningful comments in the program for your use. Print the program, as well as the output (not too long). Recall F=9/5C+32 and **Use row vectors do not use a “for” loop that uses scalar variables (one value at a time).**

21.(3 PTS) Write a 'script' file using a “for” loop, that solves for the temperature as a function of time which is given as a polynomial equal to $300t^3 + 55t^2 - 8t - 23$ for t starting at -1 and going to +2 in steps of 0.5.Since you will use a for loop the values are generated at each **loop hence they are scalars**! **(NO VECTORS!)** The program outputs the time and the temperature in a table with appropriate column headings. Add **MEANINGFUL** comments for you the programmer and the user, as well as, indenting the body of the loop in your design of the program. Then graph the function produced by having the script file define appropriate row vectors to plot.

22. (2 PTS) Write a program that will ask a user for ten numbers. After all ten numbers are entered the program will produce the average value of all the numbers. Hint: Use a **for** loop with an input() command in the loop body. **NO VECTORS**

23..(3 PTS) Modify the power series example for e^x so that the user has control on the number of terms that are summed. Appropriate prompts should be used and the program should be executed for 4, 10 and 30 terms in the series. Compare your results with a calculator or use the intrinsic function exp(.1) for the exponential function “e” of natural logarithms. Use format long and explore the best number of terms to get a precise answer that is possible? i.e. State how many terms gives us the best answer and show the result if possible?