SAVING AND IMPORTING DATA FROM A DATA FILES AND PROCESSING AS A ONE DIMENSIONAL ARRAY

If we save data in a file sequentially than we can call it back sequentially into a row vector.
Consider this m file that creates a file that you can load data into called rain.txt
We are weather folks interested in rain fall each hour in inches for a 12 hours.
>> type fcreate

```
fid = fopen('rain.txt','w');
y= input('rainfall in inches? ');
while y>=0
   fprintf(fid,'%5.1f',y);
   y= input('rainfall in inches? ');
end
fprintf(‘rain data file has been created.\n’);
fclose(fid);
```

fid is the file identifier assigned by the system.
Explore; Help fprintf, help fopen, help fclose, etc

Running this program will create a data file of numbers which are added in spaces of 4 in the file. We add 12 numbers to the file rain.txt

```
>> fcreate
rainfall in inches? 2.0
rainfall in inches? 3.1
rainfall in inches? 2.8
rainfall in inches? 0.7
rainfall in inches? 1.1
rainfall in inches? 0.0
rainfall in inches? 1.1
rainfall in inches? 0.4
rainfall in inches? 0.0
rainfall in inches? 0.0
rainfall in inches? 3.2
rainfall in inches? 2.1
rainfall in inches? -9
Rain data file has been created.
```

in our workspace we now have a file rain.txt which looks like

```
2.0 3.1 2.8 0.7 1.1 0.0 1.1 0.4 0.0 0.0 3.2 2.1
```

To recall the file and its date we use the **“load”** command as follows:

```
>> load rain.txt
```

We have now in our workspace a variable called rain which **has become 1 x 12 row vector.**
So we can now refer to each value of the vector, as before, by an index.

```
>> rain(1)  ans =    2
>> rain(2)  ans =    3.1000
>> rain(3  )ans =    2.8000
```

Note the values are in the order we stored them in the file which effectively is a one dimensional array (a row vector)
Note the precision is the typical short format.

Since rain is now a row vector we can see the contents by typing the arrays name:

```
>> rain
rain =  Columns 1 through 4
    2.0000    3.1000    2.8000    0.7000
  Columns 5 through 8
    1.1000         0    1.1000    0.4000
  Columns 9 through 12
         0         0    3.2000    2.1000
```

.
To use a data file in a program we just **load** it and use its properties.

```
>> type rainfall

% This program will process rain data over period of time for New York
% city depending on the file it loads. Here it loads a file
% called rain.txt The amount of rain fall in inches for each hour.
% The processing is done with loading the file rain.txt to the one dimensional
% array( row vector) called, rain. Average and above and below averages are done.
fprintf('Rainfall report\n');
load rain.txt;
%we need to know how many values in the array.
s =size(rain);
% s will come out as a [1  N} that is 1 x N matrix
% with N the number of values in the array
total = 0;
N=s(2);
for i=1:N
   total = total + rain(i);
end;
average = total/N;
% NOTE: WITH A ROW VECTOR WE COULD HAVE USED MATLAB   %FUNCTION
mean()  or sum() to obtain the average
% We now calculate How many values were below  and above average
countlow =0;
counthi=0;
countequal=0;
for i=1:N
  if rain(i) > average
     counthi = counthi+1;
  elseif rain(i) < average
     countlow = countlow +1;
  else
     countequal=countequal+1;
  end;
end;
% This program will illustrate the use of a more organized output by
% using the fprintf() output function. Note the use of the format
% statements by observing the output when you run the program.

fprintf('The average rain fall was %5.3f inches \n',average );
fprintf('There were %3.0f hours with rainfall above average\n',counthi);
fprintf('in addition, there were %3.0f hours below average\n',countlow);
fprintf(' and there were %3.0f hours equal to the  average\n',countequal;
```

% As we now run the rainfall program carefully study each output and note any improvements or changes you might make if you use fprintf() rather than display().

```
>> rainfall
Rainfall report
The average rain fall was 1.375 inches
There were   5 hours with rainfall above average
in addition, there were   7 hours below average
  and there were   0 hours equal to the  average
```

FUNCTIONS IN MATLAB

% MATLAB is designed as a functional language, by which we mean that numerous functions are available to the scientific programmer. In fact functions for various fields of study are available for purchase. These functions as well as those supplied with the original package are m-files. We will see below how we can build our own function m-files.

REMINDER ON SOME INTRINSIC FUNCTIONS

% As we have seen before we were able to call upon MATLAB functions to undertake various mathematical tasks. Some of these functions are fundamental to the language ("BUILT IN") or are said to be **intrinsic** .Others which we will use reside as m-files in some directory on a disk and are called **extrinsic**. If you use the **help** command it will show you the names of all functions, as well as, the directories of the disk for the extrinsic ones.

% Intrinsic function examples: include many common ones such as sqrt(x), sin(x), cos(x), tan(x), asin(x), abs(x), acos().

Information on a function can be obtained by typing help followed with the name of the function. for example:

```
>> help acos
ACOS ACOS(X) is the arccosine of the elements of X. Complex
        results are obtained if ABS(x) > 1.0 for some element.
```

% Other examples of using help for intrinsic (built-in) functions follow:

```
ATAN  ATAN(X) is the arctangent of the elements of X.
ATAN2  TAN2(Y,X) is the four quadrant arctangent of the real
        elements of X and Y.
ABS   ABS(X) is the absolute value of the elements of X. When
        X is complex, ABS(X) is the complex modulus (magnitude) of
        the elements of X. See also ANGLE.
REAL  REAL(X) is the real part of X. See also IMAG and ABS.
IMAG  IMAG(X) is the imaginary part of X.
        Imaginary numbers are not entered into MATLAB using the
        letters I or J as might be expected. This is because I and
        J are often used as indices. To enter imaginary numbers,
        SQRT(-1) is used.
        For example, 3+2i is entered as 3+2*sqrt(-1). Alternatively,
        this could be done as i = SQRT(-1); 3+2*i.
CONJ  CONJ(X) is the complex conjugate of X.
ROUND       ROUND(X) rounds the elements of X to the nearest
        integers.
FIX   FIX(X) rounds the elements of X to the nearest integers
        towards zero.
FLOOR       FLOOR(X) rounds the elements of X to the nearest integers
        towards minus infinity.
CEIL  CEIL(X) rounds the elements of X to the nearest integers
        towards infinity.
SIGN  SIGN(X) function. For each element of X, SIGN(X) returns 1
        if the element is greater than zero, 0 if it equals zero
        and -1 if it is less than zero.
        For complex X, SIGN(X) = X ./ ABS(X).
REM   Remainder. The remainder REM(x,y) is x - y * n where
        n = fix(x/y) is the integer nearest the exact value x/y.
EXP   EXP(X) is the exponential of the elements of X, e to the X.
LOG   LOG(X) is the natural logarithm of the elements of X.
        Complex results are produced if X is not positive.
SINH  SINH(X) is the hyperbolic sine of the elements of X.
COSH  COSH(X) is the hyperbolic cosine of the elements of X.
TANH  TANH(X) is the hyperbolic tangent of the elements of X.
ASINH  ASINH(X) is the inverse hyperbolic sine of the elements of X.
ACOSH  ACOSH(X) is the inverse hyperbolic cosine of the elements of X.
ATANH  ATANH(X) is the inverse hyperbolic tangent of the elements of X.
LOG10  LOG10(X)  is  the  logarithm  base 10 of the  elements of X.
        Complex results are produced if X  is not positive.
```

% A MATLAB reference manual has tables of functions which are defined to get a more complete picture of the built-in and extrinsic m-file functions. Clicking on the left hand side of the command window on the f_x brings up the manual. Many reference texts exist also!

CREATING FUNCTION M-FILES

% In scientific and engineering applications we usually create our own functions. This is done by using our editor to create an m-file which contains a **'function' definition line(SEE EXAMPLE BELOW)**. note that the **function m-file** has the **same name** as the **function.**

% DEMO of a mathematical function called humps!
» type humps % That is humps.m

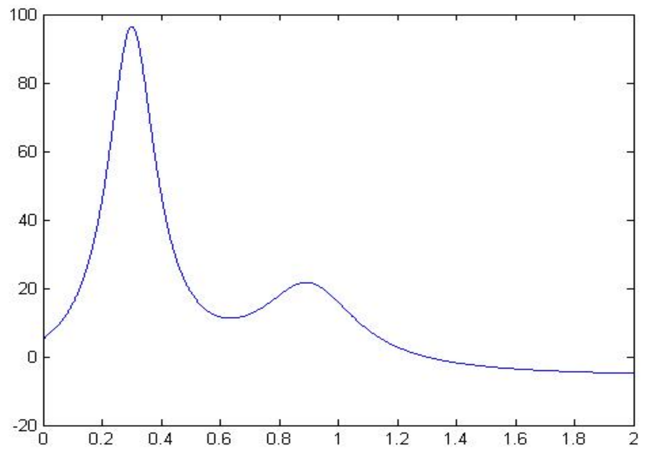
```
function y = humps(x)         %function definition line says a single variable is needed.
%       Humps(x) is a function simulating several hills
y = 1 ./ ((x-.3).^2 + .01) + 1 ./ ((x-.9).^2 + .04) - 6;
```

% The fist line in this 'humps' function lets MATLAB system know that this m- file is a function that expects a single **argument x** and will **return**
a **value y** which is **defined** by the last mathematical expression(last line of file).
% if x is a vector then y is also a vector as before!

DEMO USE OF humps() in a program called gohumps

```
>> type gohumps
%illustating use of a defined user function humps()
t=0:.001:2;
y=humps(t);
plot(t,y)
```

Gives us the picture of the humps function as(note about 2000 pts in this graph)



% The next example expects **one argument an array x** but will **return two values** (Note the size() intrinsic function in this example also returns 2 values). For this function one returned value is the mean of all members of the array and the other is standard deviation of all values You will do a detailed example in the Laboratory task to understand how the standard deviation is arrived at in statistics, step by step. Used in many fields of science when you have large numbers of data. For example in Thermodynamics of gases.

```
» DEMO type stat
% Robbins function for mean and standard deviation expects
an array as argument and returns two values which are the
mean and standard deviation of the values in the array.

function   [mean,stdev]=stat(x)
[m,n]=size(x);
N=n-1;
mean=sum(x)/n;
stdev=sqrt((sum((x-mean).^2))/N);
% using this function on the previous data from the data file rain.txt. we get
 >> [u,v]=stat(rain)
u =    1.3750
v =    1.2241
% for this function we can test the output with intrinsic functions mean() and std()
>> mean(rain)
ans =    1.3750
>> std(rain)
ans =    1.2241
```

% We note the **help facility** reads the m-file and gives us the
» **initial comments** in the m-file for example

```
>> help stat
  Robbins function for mean and Standard deviation
```

% Some functions can also have **more than one argument** but **return only one** value. for example the built-in function atan2

```
» help atan2
ATAN2        ATAN2(Y,X) is the four quadrant arctangent of the real
           elements of X and Y.
```

% We now illustrate a simple **multiple argument** m-file function in the file named **ave3.m**.

DEMO

```
>> type ave3
% This is a simple function that gets the mean
% of the sum of three arrays.
%.if x,y,z are arrays and they are added element by element for the final
%mean value!

function y = ave3(x,y,z)
y= mean(x+y+z);
```

% WE NOTE AGAIN THAT THE FUNCTION NAME IS ALSO THE FUNCION M- FILE NAME!
% We now note the use of the above **ave3()** function of 3 arguments

```
» u=[1 3 4];
» v=[0 5 15];
» w=[9 27 23];
» z = ave3(u,v,w)
z =    29
```

% Note that this last variable, z, and the function variable z, are not the same!
The function z is not known outside the function it only exist to calculate the function!
% Further note, u,v,w are called the **arguments of the function while x,y,z in the function are called** the **parameters** of the function. Arguments/parameters very important names in computer languages, learn the distinction.

LABORATORY TASK(S)

Reminder, use the command 'type" to list your m- file, once the file is debugged..
>> type filename   will produce a listing of the program. Be sure to copy the command and listing. Do not run programs line by line in the interactive window or print from such, it will not count as a program.

30(10PTS). You are to construct a program to compute the STANDARD DEVIATION of a group of numbers. The standard deviation gives one a sense of the spread of data around the mean (what we called the average above).

0. Create the m-file like fcreate.m illustrated above.

Print it up with the type command!

Run it to put ut the following numbers in a data file called **num.txt**

3.2 3.1 3.5 3.2 3.0 3.1 3.9 3.6 3.9 3.6 3.9 3.4 3.2   3.3 6.7 3.6 3.7 3.7

1.Now create the **program** called stand.m to compute the standard deviation of the data it obtains from the data file, num.txt.
Outputs of the numbers and their **deviations** calculated below should be in **labeled columns**.

2. program should load the above data file which is now a vector!

3. Compute the sum of all of the numbers: Output this value. Use **sum()! Not a Scalar calculation!**

4. Obtain the average (called the mean value in statistics) and output it.
Use mean()

5 Compute the difference of each value from the average called the
deviation using a **for loop** and an expression like
dev(i)=num(i)-ave;   output it along with its number(i) in labeled table. Here we treat each number individually as a scalar and thus create a new vector dev in the end.

6 **Simultaneously** (**within the same loop**) compute the sum of all the deviations call it the **checksum**   and output it **after the loop. Ie checksum =?**

7 Simultaneously **sum all the squares of the deviations in the loop.**

8 **After** the loop compute the average of the last sum (the average of the deviations squared) but use (one less than the total number). This number is called the **variance** in statistics and output it.

9 Compute the square root of the last average- known as the **standard deviation** and output it.

10. Have your program use the MATLAB function **std()** on the row vector of numbers! Output these values and compare them with your calculations above . What are the differences if any, show percent difference it there is a difference. It should be very close if not 0!

The program you construct will carry out all actions discussed in the 10 parts above. Print program (type filename)   and output and the data file to hand in

31(4 PTS)- Construct a **function m-file** called **hill()** for the function given as

$4\text{asinh}(x)\, e^{-0.2x}$

Print up the code for this function.
Use" type hill" for listing and print to hand in.

32(4 PTS). Construct **a script m-file** that:
Uses the above function file to solve for y = hill(x) with 0.3<= x <=12 in steps of 0.01 and then plot y verses x . print the graph
Using MATLAB functions the program also produces the following and outputs appropriate statements to the user.
3 finds the median value of y
4 finds the mean value of y
5 finds the maximum and minimum values of y
6.finds the standard deviations of y

33(4 PTS) . This time write a program to load num.txt that will take the square root of each value and save them to a file called rootfun.txt. Be sure to save numbers with 4 decimal precision**. Print up the num.txt, the m file created for this task and output file rootfun.txt.**

34.(3pts) Improving your odds in the lottery
Explore the "rand" function by running it several times in the command window.

Use this function to create a random generation of integer numbers from 1 to 10
Be sure when you run your creation that no zero appears and all values are integer for full credit

Once you get the idea you can look over a years worth of lottery numbers, for example powerball and see the range in each position and use your creation to get random values in that range. So for example if you see the first powernball over the year only has numbers from 1 to 33 your random generator can do that and also by modifying created random numbers within the range of the 2nd powerball and 3rd powerball etc. Hopefully this will help? Good luck. Buy me dinner if you win..LOL