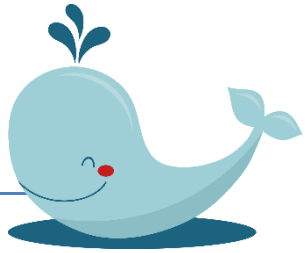


Standard Function Exercises

File: Standard Function Exercises (May 15,2018)

Course Learning Objectives:

- Apply syntax rules to design standard, recursive, anonymous, and nested functions
    - Demonstrate how to design, save, call, run, and debug a function, including
        - Identify input and output arguments to construct a function
        - Develop the algorithm within the function
        - Employ debugging strategies to debug a function

Action verbs for student learning outcomes:
https://www.mnstate.edu/assess/poa/actionverbs.aspx

Output Arguments: Syntax requirements

1. If your function returns one output, you can specify the output name after the function keyword in two ways:
   function myoutput = myFunction(X,Y)
   function [myoutput]=myFunction(X,Y)
2. If your function returns more than one output, enclose the output names in square brackets:
   function [A,B,C] = myFunction(Z)
3. If there is no output, you can omit the brackets or use empty square brackets:
   function myFunction(X,Y)
   function [ ] = myFunction(X,Y)

Input Arguments: Syntax requirements

4. If your function accepts any number of inputs, enclose their names in parentheses after the function name. Separate input names with commas:
   function [A,B] = myFunction(X,Y,Z)
5. If there are not inputs, you can omit the parentheses
   function [C] = myFunction

15-Exercises Standard Functions

Basic Exercises

1. Write a function to compute the complementary sine:

$$sinc(x) = \frac{\sin(x)}{x}$$

Then write a Driver program to calculate the complementary sine function (i.e., sinc(x)) for values of x from 1.0 up to 10.0 in increments of 0.1

$$sinc(x) = \frac{\sin(x)}{x}$$

2. Write a function to convert km to miles. Upgrade the previous functions to handle arrays at Input and Output.
3. Write a MATLAB user-defined function (call it **worldSalute**) aiming to salute the world as "Hello there world." The function uses no arguments and returns no values to the calling program.
4. Write three user-defined functions to calculate the hyperbolic sine, cosine, and tangent functions:

$$\sinh(x) = \frac{e^x - e^{-x}}{2} \qquad \cosh(x) = \frac{e^x + e^{-x}}{2} \qquad \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Choose a name to avoid syntax conflicts, e.g.: iSinh, iCosh, and iTanh, respectively. Use your functions to plot the shapes of the hyperbolic sine, cosine, and tangent functions by writing a program that computes them for common ranges.

MEDIUM COOK Exercises

5. Create the MATLAB **iSum** library **function** that adds up the elements of any 1D array, for instance the **t** array with N elements and returns the sum. The sum is defined by:

$$s_n = t_1 + t_2 + \cdots + t_n$$

Assume the **iSum** function will exist in MATLAB for the first time. You can use any other MATLAB library function within your function except by **sum**. Could you upgrade the above function to work also with 2D array input.

6. Developing a function using one as a model. Taking as basis (or source of inspiration) the algorithm in the function minimum2 on your notes, develop a new function and called it, **maximum2** which will find the maximum of an array-x. Your function should use only one argument, i.e., x. Show how you call your function in the command window:
7. Write the user-defined function **diffSquare** in charge of calculating the square of the absolute value of the difference between two (2) scalar values stored in x and y variables, respectively.
8. Expand the above function to work with two 1D arrays, x and y, with the same number of elements, each. Hence an array representing the absolute value of the differences between each corresponding element (element wise) in x and y are returned by the function

9. Create the **iMax** library function that finds the maximum element of an 1D array, for instance the maximum of the **x** array and returns the result. Assume that **x** has usually **n** elements. A convenient algorithm is to assume the first element of **x** is the largest, then replaced it by any other that if compared with it resulted larger, this approach continues until the last element. At each step two consecutive values on the list are compared and decision is made if the 2nd one is larger it replaces the 1st, otherwise the 1st continues to be the reference base for the next comparison. Use the **numel** function to determine the elements in **x**. Can't use the **max** library function.
10. Upgrade iMax to handle 2D array input
11. Write the function **myOnes(x).** A typical call with x=3 as the input argument, gives a 3 x 3 array of ones.
12. Write the function **myTriangulos(x).** **Where x is an integer variable.** A typical call with x=5 as the input argument, gives a triangle like:

55555
5555
555
55
5

13. Create the MATLAB library **myPi** parameter up to 15 decimal digits of accuracy, such that whenever pi is writen MATLAB will insert the pi value. This parameter can be created as a MATLAB function. In order to look as a parameter, write the function such as to call it, it won't need of input arguments.

    The first 100 decimal digits of π are **3.14159 26535 89793 23846 26433 83279 50288 41971 69399 37510 58209 74944 59230 78164 06286 20899 86280 34825 34211 70679…** Show how you can use myPi in a Driver program.

14. Write a function **prod** which computes the cumulative product of the elements in a vector. The cumulative product of the vector x, is defined by

$$p = (x_1)(x_2) \ldots (x_n)$$

15. Create the MATLAB **myLinspace(a,b,nn)** library **function** that returns a vector with nn equally spaced values from a up to b.

10.– Write a MATLAB script that uses the library function min() (see MATLAB help for syntax, arguments, and examples) to compute the minimum of seven integers and the position in the set stored in an array called **numset**=[11,7, 5, 9, 3, 15, 4]. The program must be in-charge of printing the data and the results with appropriate statements.

```
% This program computes the minimum of
% seven integers stored in an array called numset.
% The program uses the min library function

clear, clc
numset=[11 7 5 9 3 15 4];
% call the library function min
[minimo, position]=min(numset)
```

11.– Write the function **square** that displays at the left margin of the screen a solid square of asterisks whose side is specified in a parameter side. For example if side is 4, the function displays:

```
****
****
****
****
```

SOLUTION

```
function square(s)
% This function draws an square of "*" of side s

  for ii = 1:1: s
    for jj = 1:1:s
      fprintf('*');
    end
    fprintf('\n');
  end
```

Calling the function in the Command Window:

```
>>x=10;
>> square(x)

**********
**********
**********
**********
```

```
**********
**********
**********
**********
**********
**********
```

12.– Modify the function created in problem 7 to form the square out of whatever character is contained in a character parameter **squaresymbol.** If side is 5 and squaresymbol is "#" then the function should print:

```
#####
#####
#####
#####
#####
```

SOLUTION

```
function squareSymbol(s, c)
% This function draws an square of c-characters of % side s
  for ii = 1:1: s
    for jj = 1:1:s
      fprintf('%c',c);
    end
    fprintf('\n');
  end
```

Calling the function in the Command Window:

```
>> s=5
s =
    5
>> c='#'
c =
#
>> squareSymbol(s,c)
#####
#####
#####
#####
#####
```

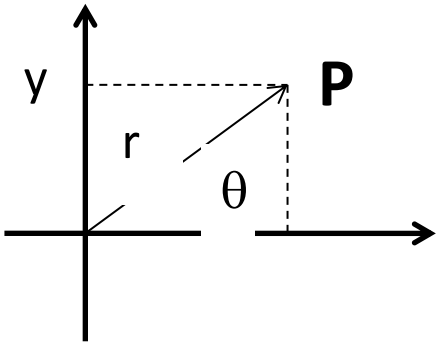
13.- The location of a point **P** in a Cartesian plane can be expressed in either the rectangular coordinates (x,y) or the polar coordinates (r,θ), as shown in the figure below. The relationships among these two sets of coordinates are given by the following equation:

$$x = r\cos\theta$$

$$y = r\sin\theta$$

$$r = \sqrt{x^2 + y^2}$$

$$\theta = \tan^{-1}\frac{y}{x}$$



Write two functions rectTOpolar and polarTOrect that convert coordinates from rectangular to polar form, and vice versa, where the angle θ is expressed in degrees.

Note that MATLAB's (and also C's) trigonometric functions work in radians, so we must convert from degrees to radians and vice versa when solving this problem. The relationship between degrees and radians is: 180 degrees = π radians
In these exercises you should pay attention to syntax requirements:

- ✓ Input and Output arguments
- ✓ Function header syntax (the first line in the function structure)
  - function "the key word" (without the key word you have nothing)
  - function name,
  - the variable(s) storing the result(s)
- ✓ Function body (the function computations and algorithm)
- ✓ Function termination statement (i.e., 'end')
- ✓ Calling the function from a program, command window or statement by providing appropriate arguments

HARD

1.– In a new programming job at Mathworks Headquarters your assignment is to create the MATLAB **natural logarithm** library **function**, as usual we called this **"iLog."** The **iLog** function returns the natural logarithm of a given scalar **x numerical value**. Advice the user that **x** must be a positive value. To compute the natural logarithm, use the Taylor series expansion below.

$$\mathbf{ln}(\mathbf{x}) = 2\left\{\left(\frac{\mathrm{x}-1}{\mathrm{x}+1}\right) + \frac{1}{3}\left(\frac{\mathrm{x}-1}{\mathrm{x}+1}\right)^{3} + \frac{1}{5}\left(\frac{\mathrm{x}-1}{\mathrm{x}+1}\right)^{5} + \cdots\right\} \qquad \mathrm{x} > 0$$

or

$$\mathbf{ln}(\mathbf{x}) = 2\sum_{2n-1}^{\infty} \frac{1}{2\mathrm{n}-1}\left(\frac{\mathrm{x}-1}{\mathrm{x}+1}\right)^{2\mathrm{n}-1}$$

You must decide the value of n

```
function   [s]  = iLog(x)
% Computes the natural logarithm with Taylor series
% x must be a positive value
        if x<=0
                fprintf(' Error, x must be positive');
        else
                s=0; t=(x-1)/(x+1)
                for k=1:2:19    % assumed n=10 terms
                        tt= (1/k)*t^k;
                        ss=ss+ tt;
                end
                s=2*ss
        end
end
```

```
        function   [s]  = iLog(x)
        % Computes the natural logarithm with Tayler series
        % x must be a positive value
                n=1:2:50    % assumed 50 terms
                t= (1./(2.*n-1)).*((x-1)/(x+1).^(2.*n-1);
                s=2*sum(t);
        end
```

2.- Assume you got a new programming job at Mathworks Headquarters and you have been requested to create the MATLAB **exponential** library **function, iExp,** that returns the exponential of a given scalar **x numerical value**. To compute the exponential, use the Taylor series expansion below:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots,$$

```
function   [s]  = iExp(x)
% Computes the exponential function

        s=1; f=1;
        for k=1:1:20    % assumed 20 terms
                f=f*k;
                t= (x^k)/f;
                s=s+ t;
        end
end
```

3.- You have been assigned to create the MATLAB **sine** library **function, iSin(x),** that returns the sine of a given angle **x in radians**. To compute the sine, use the Taylor series expansion below. *Hint:* you can use the **factorial** library function.

$$\sin(x) \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!}.$$

```
function   [s]  = iSin(x)
% Computes the sine function
        s=0; ii=2;
        for k=1:2:29    % assumed 14 terms
                t= (x^k)/factorial(k)
                s=s+(-1)^ii* t;
                ii=ii+1;
                end
end
```

```
function   [s]  = iSin(x)
% Computes the sine function
        s=0; f=1; jj=2;
        for k=1:2:29    % assumed 14 terms
                f=1;
                for ii=1:1:k
                        f=f*ii;
                end
                t= (x^k)/f;
                s=s+ (-1)^jj*t;
                jj=jj+1
        end
end
```

4.– Write a user-defined function called **cMP** to compute the coefficients in the Midpoint's integration rule.

$$I = 2h\left(f_2 + f_4 + f_6 + f_8 + \ldots f_{n-2} + f_n \right)$$

Note the ii-index in the formula jumps as 2, 4, 6 and that coefficients with ii=1, 3, 5, 7, .. n+1 are zero:

c=[0,1,0,1,0,1,…,1,0];

SOLUTION

```
function c=cMP(n)
        % Computes the coefficients in
        % Midpoint Integration Rule
            % n must be even
    for ii=1:1:n+1
       if mod(ii,2)==0
            c(ii)=1
       else
            c(ii)=0
       end
    end
end
```

c = zeros(1,n+1)

```
function c=cMP(n)
    % Computes the coefficients in Midpoint
     % Integration Rule
     % n must be even
                c(1:2:n+1)=0;
                c(2:2:n)=1;
end
```

One more compacted solution

```
function c=cMP(n)
    % Computes the coefficients in Midpoint
    % Integration Rule
    % n must be even
    c=mod(1:1:n+1,2)==0;
end
```

5.- The following formula generate **n** integer values from the uniform distribution on the interval [a, b]:

$$r = round(a + (b - a).* rand(1, n));$$

For instance, for **one** dice with range between a=1 and b=6, played n=6 times, with the formula above MATLAB produces:

```
r =
     2     3     6     3     4     2
```

Six integer values in the range [a, b] = [1,6].

Write a **function** named **dices** that receive the number of times, **n**, three dices will be played and returns the 3n values. Store the values of dice 1, 2, and 3 in the x-, y- and z- arrays respectively.

```
function [x,y,z] = dices(n)
        % The random generator of 3 dices played n times
        % etc.

        x =round(1+(6-1).*rand(1,n));
        y =round(1+(6-1).*rand(1,n));
        z =round(1+(6-1).*rand(1,n));
end
```

6.- Create the MATLAB user-defined **function** that computes de **iPi** function with the series shown below. To compute the series you must assume a number of terms. Write the function such as to call it, it won't need of input arguments.

$$\pi = \frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \frac{4}{13} \ldots = 4(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \frac{1}{13} \ldots)$$

```
function   [s]  = iPi          % no input argument
% Computes the mathematical pi parameter as a
% MATLAB library function

        ii=2; s=0;
        for k=1:2:19    % 10 iterations, 10 terms
                s=s+(-1^ii)*4/k
        ii=ii+1;
                end
end
```

10.1- Show how you can use the function iPi to compute the following statements.

7.- The Fibonacci series 0, 1, 1, 2, 3, 5, 8, 13, 21,… begins with the terms 0 and 1 and has the property that each succeeding term is the sum of the two preceding terms. This can be expressed as:

$$F_i = F_{i-1} + F_{i-2}$$

Write a nonrecursive function fibonacci(n) that calculates the nth Fibonacci number.

8.- Assume you will create the MATLAB **factorial** library **function, iFactorial(n),** that returns the factorial of n. To compute the factorial you can use the following definitions:

$$0! = 1$$
$$1! = 1$$
$$n! = 1\ x\ 2\ x\ 3\ x\ \dots n$$

```
function fac=iFactorial(k)
% This function computes the factorial of k

        fac=1
        if k==0
                fac=1
        elseif  k==1
                fac=1
        else

                for ii=1:1:k
                        fac=fac*ii;
                end
```

Not necessary, why?

```
        end
    end
```

9.- Write the function i**Trapz** in charge of implementing the Trapezoidal integration rule with x and f arrays as input arguments with n+1 elements . The output value is the integral result. A typical call to the function would be i**Trapz(x,f).** Hint: the following formulas can be considered in the solution.

n=length(x),
h = (x(1)-x(n+1))/n;

$$I = \frac{1}{2} h \left(\underbrace{f_1}_{t_1} + \underbrace{2 f_2}_{t_2} + \underbrace{2 f_3}_{t_3} + 2 f_4 + \ldots + \underbrace{2 f_n}_{t_n} + \underbrace{f_{n+1}}_{t_{n+1}} \right)$$

```
function  [I] = iTrapz(x,f)
    % Computes Trapezoidal integration for given function of x

        n=length(x);
        h = (x(1)-x(2))/n;

        c(1)=1;    c(2:1:n)=2;   c(n+1)=1;

        t=c.*f;
        I = (1/2)*h*sum(t);
end
```

10.- Write the function i**LinSpace(A,B)** which uses the matrix A and the vector B as input arguments to find the solution vector **X**, according to:

In general a system of **m** equations in **n** unknowns can be written as:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2 \\ \ldots & \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n &= b_m \end{aligned} \quad \Longrightarrow \quad AX=B$$

In matrix form:

$$A = \begin{bmatrix} a_{11} & a_{12} \ldots & a_{1n} \\ a_{21} & a_{22} \ldots & a_{2n} \\ \ldots & & \\ a_{m1} & a_{m2} \ldots & a_{mn} \end{bmatrix} \qquad X = \begin{bmatrix} x_1 \\ x_2 \\ \ldots \\ x_n \end{bmatrix} \qquad B = \begin{bmatrix} b_1 \\ b_2 \\ \ldots \\ b_m \end{bmatrix}$$

49

```
function [X] = iLinSpace(A,B)
        %  Computes the solution vector of the AX=B system of equations
```

```
X=A\B;

end
```

11.- Create the MATLAB user-defined **function** that computes the **pi** function with the series shown below. To compute the series you must assume a number of terms. Write the function such as to call it, it won't need of input arguments.

$$\pi = 3 + \cfrac{1^2}{6 + \cfrac{3^2}{6 + \cfrac{5^2}{6 + \cfrac{7^2}{6 + \cfrac{9^2}{6 + \cdots}}}}}$$

```
function   [s]  = pi        % no input argument
% Computes the mathematical pi parameter as a
% MATLAB library function

        s=11^2/6;
        for ii=9:-2:1    % 5 iterations
                s=6+(ii^2/s);
                end
        s=s+3;
end
```

Other possibilities to practice programming structures are:

$$\pi = \cfrac{4}{1 + \cfrac{1^2}{2 + \cfrac{3^2}{2 + \cfrac{5^2}{2 + \cfrac{7^2}{2 + \cfrac{9^2}{2 + \ddots}}}}}} = 3 + \cfrac{1^2}{6 + \cfrac{3^2}{6 + \cfrac{5^2}{6 + \cfrac{7^2}{6 + \cfrac{9^2}{6 + \ddots}}}}} = \cfrac{4}{1 + \cfrac{1^2}{3 + \cfrac{2^2}{5 + \cfrac{3^2}{7 + \cfrac{4^2}{9 + \ddots}}}}}$$

12.- Consider the function:

```
function [LG]=letterGrade(score)
        % Assigns a letter grade to given numerical grade, score
        % score=numerical grade
        % LG= the letter grade is a char variable
```

```
        if score>=90
                LG='A';
        elseif score>=80
                LG='B';
        elseif  score>=70
                LG='C';
        elseif score>=60
                LG='D';
        else
                LG='F';
        end
end
```

13.- An integer number is said to be a **perfect** number if its factors, including 1 (but not the number itself), sum to the number. For example, 6 is perfect number because 6 = 1 + 2 + 3. Write the function **perfect** that determines if parameter **number** is a perfect number. Use this function in a program that determines and prints all the perfect numbers between 1 and 1000. Print the factors of each number to confirm that the number is indeed perfect (ref: Problem 5.26, **C How to Program** by Deitel)

SOLUTION

```
% perfectMain.m
% This program reports if the numbers from 1 to 1000 are or not perfect

  fprintf(' For the integers from 1 to 1000: \n');

  for jj = 1:1:1000
    if (perfect(jj))
      fprintf(' %d is perfect \n ', jj);
    end
  end
```

```
function y=perfect(x)
% This function computes if a number x is perfect or not
%
  factorSum = 1;

  for ii = 2:1:(x/2)
    if mod(x,ii)== 0
      factorSum = factorSum + ii;
    end
```

```
  end

  if (factorSum == x)
    y= 1;
  else
    y=0;
  end
end
```

Running the program in the command window:

```
>> perfectMain
 For the integers from 1 to 1000:
 1 is perfect
 6 is perfect
 28 is perfect
 496 is perfect
```

NEXT FUNCTIONS ARE NOT RATED YET

31.- Road Traffic Density. Function random produces a number with a uniform probability distribution in the range=[0.0,1.0]. This function is suitable for simulating random events if each outcome has an equal probability of occurring. However, in many events, the probability of occurrence is not equal for every event, and a uniform probability distribution is not suitable for simulating such events.

For example, when traffic engineers studied the number of cars passing in a given location in a time interval of length t, they discovered that the probability of k cars passing during the interval t is given by the equation

$$P(k,t) = e^{-\lambda t}\frac{(\lambda t)^k}{k!}$$ for $t \geq 0$, $\lambda > 0$, and $k = 0,1,2,\ldots$

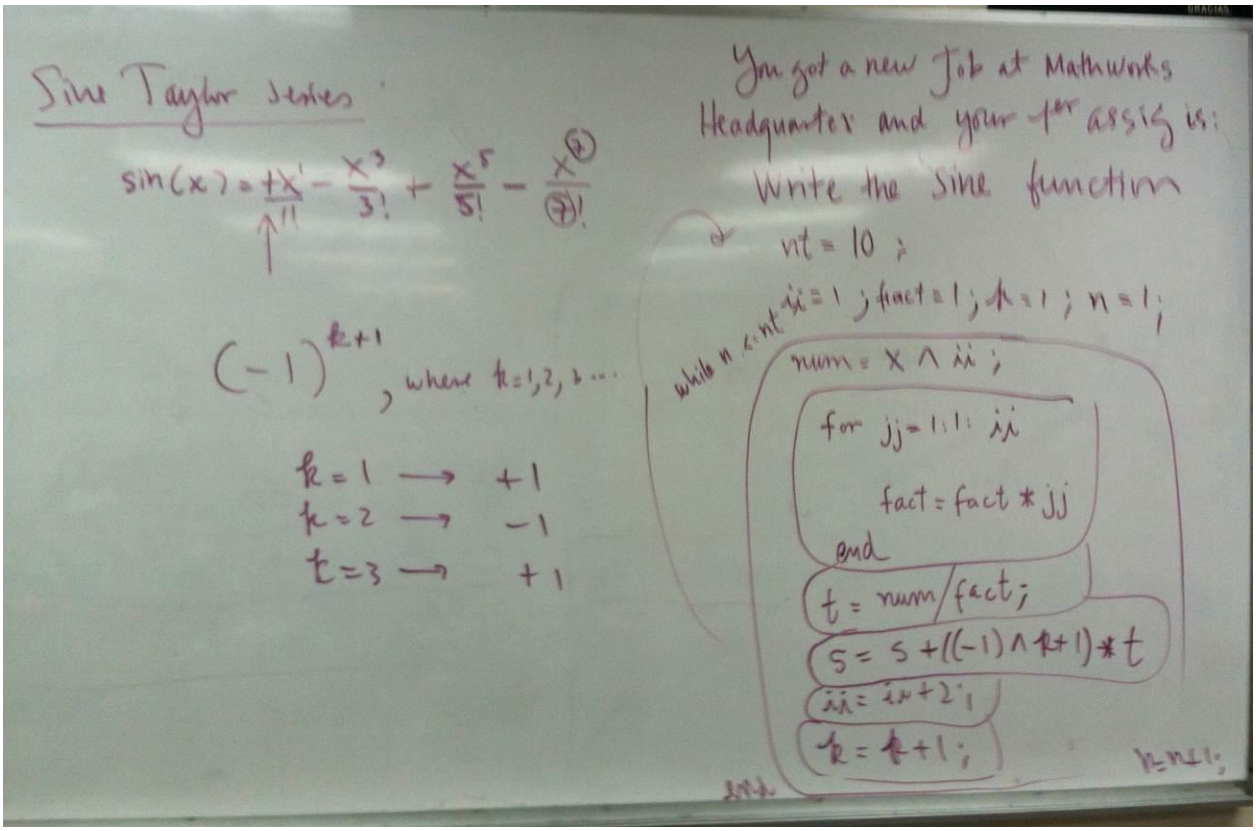
This probability distribution is known as the Poison distribution; it occurs in many applications in science and engineering. For example, the number of calls k to a telephone switchboard in time interval t, the number of bacteria k in a specific volume t of liquid, and the number of failures k of a complicated system in time t all have Poisson distributions.

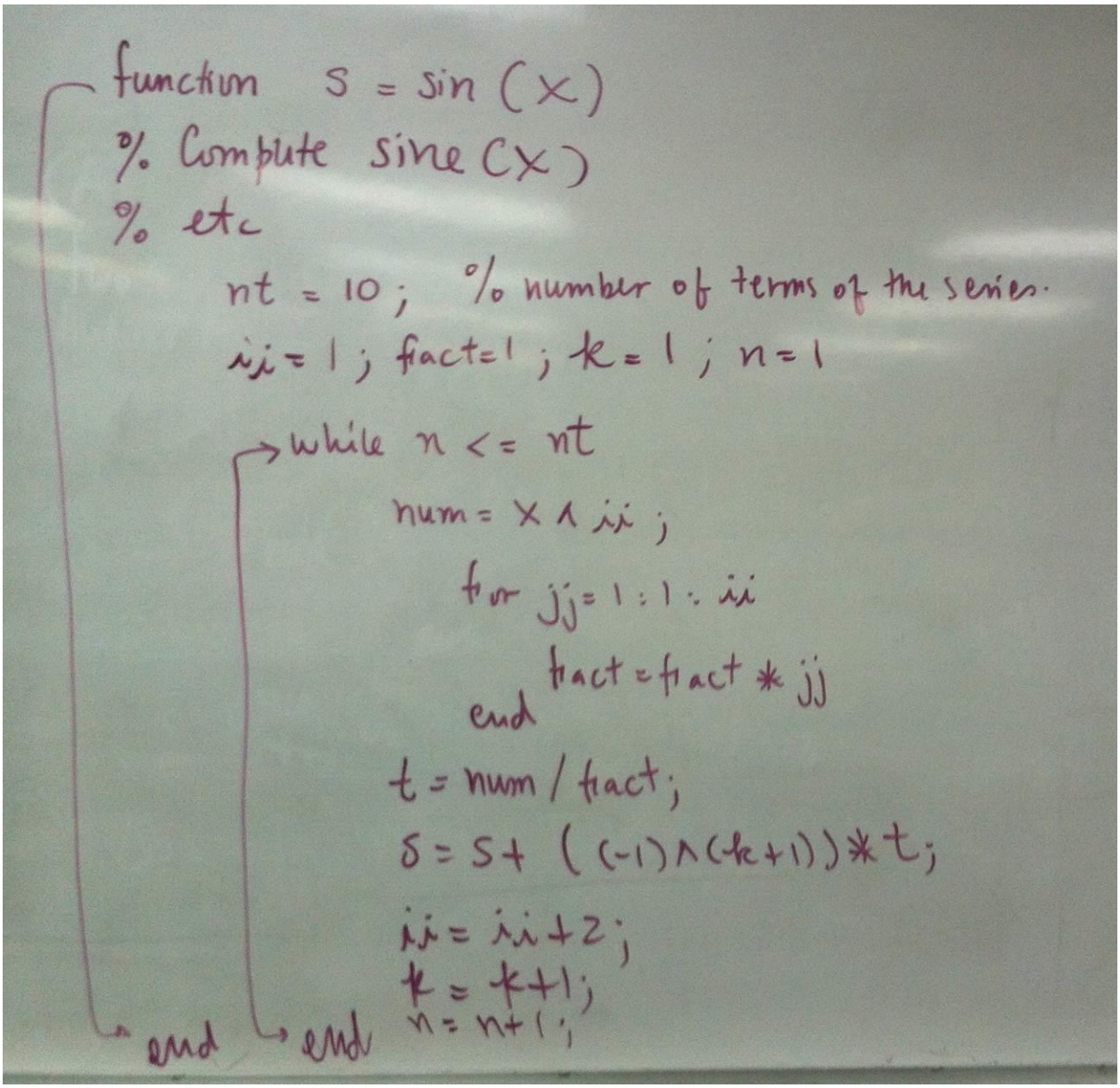
Write s function to evaluate the Poisson distribution of 0, 1, 2, …, 5 cars passing a particular point on a highway in 1 minute, given that λ is 1.6 per minute for that highway. Plot the Poisson distribution for t=1 and λ=1.6.

SOLUTION

```
function prob=poisson(k,t)
        % This function computes the poisson distribution
        % k= number of cars
        % t= time
        landa=1.6
        prob=exp(-landa*t)*(landa*t)^k/factorial(k) ;
end

function fac=factorial(k)
% This function computes the factorial of k

        fac=1
        if k==0
                fac=0
        elseif  k==1
                fac=1
        else

                for ii=1:1:k
                        fac=fac*ii;
                end
        end
```

32.- Write a program to compute the sine function with the Taylor series expantion. Once you have it, develop the **usin** function. Problem developed in class (july-02-2012)


Sine Taylor Series
$\sin(x) = \frac{+x^1}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!}$
$(-1)^{k+1}$, where k=1,2,3...
k=1 → +1
k=2 → -1
k=3 → +1
You got a new Job at Mathworks Headquarter and your 1st assig is:
Write the Sine function
nt = 10 ;
while n <= nt ii=1 ; fact=1 ; k=1 ; n=1 ;
num = X ^ ii ;
for jj=1:1: ii
fact = fact * jj
end
t = num/fact;
S = S +((-1)^k+1)*t
ii = ii+2;
k = k+1;
n=n+1;



1.- Show how you can use logical arrays and masks to double the even numbers in the x array. To test your program, assume the x array has n elements, where n =10:

Given: x =[1,2,3,4,5,6,7,8,9,10]

Ouput: x=[1,4,3,8,5,12,7,16,9,20]

SOLUTION

```
clc, clear;

x =[1,2,3,4,5,6,7,8,9,10];
b=mod(x,2)==0;
x(b)=2.*x(b);                % also, simpler:  x(b)=2;
```

33.- Recursive Functions

The underlying principle is very simple: a function that can call itself. In other word, something like this:

```
function f = fact(x)
% Computes the factorial of x

    if (x == 1)
       f= 1;
    else
       f= x*fact(x-1);         % function call
    end

end
```

Another version:

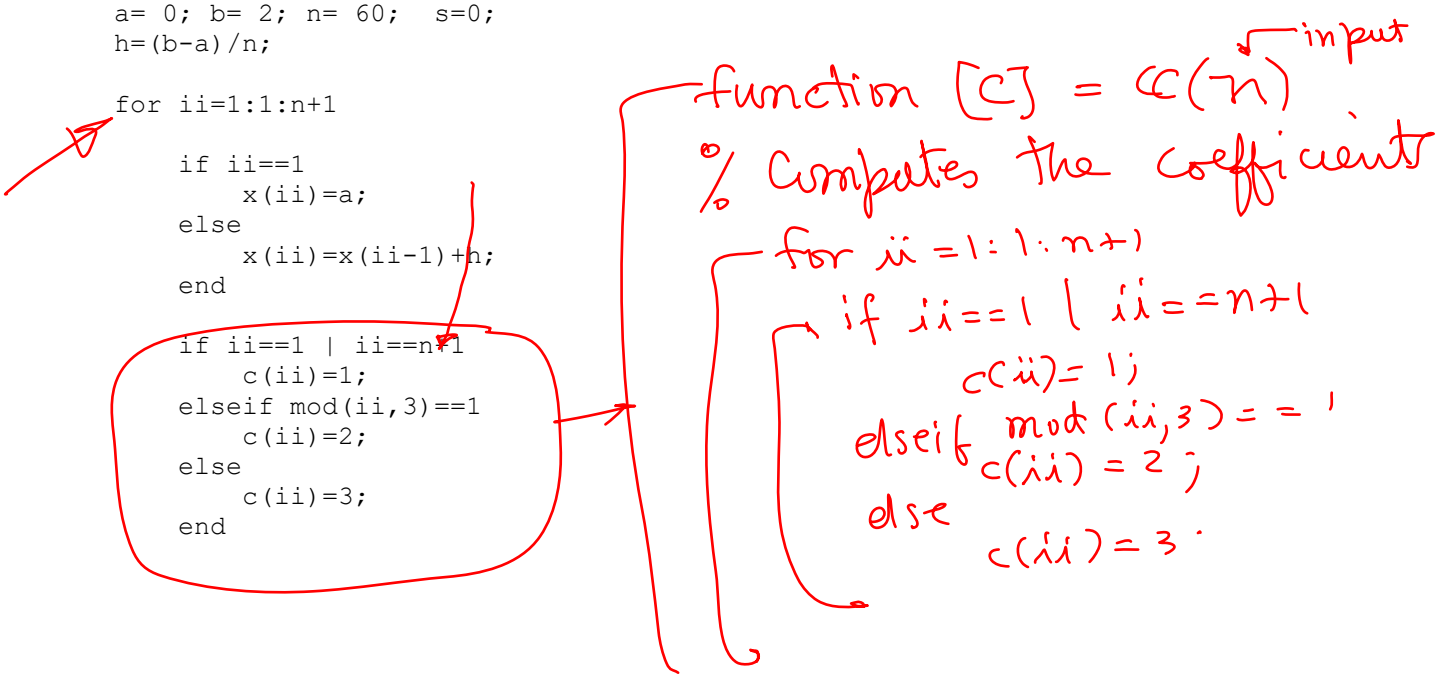
```
function f = fact(x)
% Computes the factorial of x

    If x>1
            f=x*fact(x-1);              % function call
    else
            f=1;
    end
end
```

FUNCTION DEVELOPMENT AND THE SIMPSON 3/8 RULE

We will use the following solutions for a deeper understanding of the function structure development

Solution #1. This solution resembles more the syntax of traditional high level languages, such as, FORTRAN, C, Basic, Pascal.

```
% Simpson 3/8 Integration Rule
% This program computes an integral using the Simpson (3/8) rule
%
% V.N:
% a = lower integration limit;  b = upper integration limit
% n = must be multiple of six, h=   ; x=  ; c=   ;  f=    ; t=    etc.
% I=

clc; clear;

a= 0; b= 2; n= 60;  s=0;
h=(b-a)/n;

for ii=1:1:n+1

    if ii==1
        x(ii)=a;
    else
        x(ii)=x(ii-1)+h;
    end

    if ii==1 | ii==n+1
        c(ii)=1;
    elseif mod(ii,3)==1
        c(ii)=2;
    else
        c(ii)=3;
    end
```



```
    f(ii)= 2+cos(2*sqrt(x(ii)));
    t(ii)=c(ii)*f(ii);

    s = s + t(ii);
end

    I = (3/8)*h*s;
    fprintf('The results of the integration is %f ',I);
```

Solution #2. This solution uses the more sophisticated MATLAB statements, whose syntax and features is a characteristic of MATLAB.

```
% Simpson 3/8 Integration Rule
% This program computes an integral using the Simpson (3/8) rule
% The code is vectorized
% V.N:
% a = lower integration limit
% b = etc.
% h= etc.
% n = must be multiple of six

clc; clear;

a= 0; b= 2; n= 60;
h=(b-a)/n;

x =[a:h:b];
f = 2 + cos(2.*sqrt(x));

c(1:1:n+1)=3;  % all are three
c(4:3:n-1)=2;  % some are replaced by 2
c(1)=1; c(n+1)=1; % extremes are replaced by 1

t=c.*f;

I = (3/8)*h*sum(t);
fprintf('The results of the integration is %f',I);
```

Solution #3 Functions

```
function cc=c(n)
% calculates the coefficients in Simpson 1/3 rule
% n = the number of panels must be multiple of six
      for k=1:1:n+1 % c-values are one more than the number of panels
            if ii==1 | ii==n+1
                  cc(k)=1;
            elseif mod(ii,3)==1
                  cc(k)=2;
            else
                  cc(k)=3;
            end
```

```
      end
end
```

Another alternative:

```
function cc=c(n)
% calculates the coefficients in Simpson 1/3 rule
% n = the number of panels must be multiple of six
      cc(1:1:n+1)=3;  % all are three
      cc(4:3:n-1)=2;  % some are replaced by 2
      cc(1)=1; c(n+1)=1; % extremes are replaced by 1

end
```

This solution uses the function c previously developed and the most sophisticated MATLAB statements. Note how the MATLAB code is becoming shorter.

```
% Simpson 3/8 Integration Rule
% This program computes an integral using the Simpson (3/8) rule
%
% V.N:
% a = lower integration limit
% b = etc.
% n = must be multiple of six

clc; clear;

a= 0; b= 2; n= 60;
h=(b-a)/n;

x =[a:h:b];
f = 2 + cos(2.*sqrt(x));

t=c(n).*f;

I = (3/8)*h*sum(t);
fprintf('The results of the integration is %f ',I);
```

24.- Write a MATLAB program that uses a function to verify if a list of 10 integer values stored in an array variable are odd or even. The program must be in-charge of reading and printing the data and the results with appropriate statements. Hint: develop a function that verifies if one value is odd or even then place the function within a loop to verify the list.

The following program was stored as oddevenmain.m

```
% This program uses a user-defined function to verify if
% a list of 10 inter values stored in an array
```

```
% variable are odd or even. The program is in-
% charge of reading and printing the data and
% the results with appropriate statements in
% external files.

listValues=[3 7 1 9 2 8 5 4 6 0]

% the program checks values one-by-one
for ii=1:1:10
   if oddeven(listValues(ii))
         fprintf(‘ %d is even’,ii);
else
         fprintf(‘ %d is odd’,ii);
end
```

To run the code type oddevenmain in the Editor

Following function was stored as oddeven.m file:

```
function sino=oddeven(x)
         % This function finds if a number is even or not
         % returns 1 if even, and zero otherwise
         if mod(x,2)== 0
            sino=1;
         else
            sino=0;
         end
end
```