MATLAB ASSIGNMENT 1

Exercise 1 : Conversion methods — 30 Points

Question 1: 15 points **Question 2: 15 points**

Question 1 **15 points**

Write a MATLAB function:

function [E8 , F8] = Convert2to8(E2, F2)

which takes as input two binary vectors E2 and F2 that represent respectively the integral and fractional parts of a positive binary number b, converts them to octals and outputs the results as 2 vectors E8 and F8 that represent respectively the integral and fractional parts of a positive octal number o.

Solution: **12 points**

```
function [E8 F8] = convert2to8(E2, F2)

% Conversion of the Integral part of b from binary to octal
n=length(E2);     % adding zeros for making length(E2) multiple of 3          2 points
r=rem(n,3);
if  r==1
    E2=[zeros(1,2) , E2];
elseif r==2
    E2=[zeros(1,1) , E2];
end
n2=length(E2);    % CONVERTING                                                4 points
n8=fix(n2/3);
E8 = zeros(1,n8);
for j = n8:-1:1
    E8(j)=E2(3*j)+E2(3*j - 1)*2 + E2(3*j - 2)*4 ;
end

% Conversion of the Fractional part of b from binary to octal
N=length(F2);     % adding zeros for making length(F2) multiple of 3          2 points
R=rem(N,3);
if  R==1
    F2=[ F2, zeros(1,2) ];
elseif R==2
    F2=[ F2, zeros(1,1) ];
end
N2=length(F2);    % CONVERTING                                                4 points
N8=fix(N2/3);
for j=N8:-1:1
    F8(j)=F2(3*j)+F2(3*j- 1)*2 + F2(3*j - 2)*4;
end
```

Test case: **3 points**

```
>> E2=[1 0 1 1 1];
>> F2 = [0 1 1 0 0 0 1];
>> [E8 F8] = convert2to8(E2, F2)
E8 =
   2   7
F8 =
   3   0   4
```

Question 2 **15 points**

Write a MATLAB function:

function [E10 F10] = Convert8to10(E8, F8)

which takes as input two octal vectors E8 and F8 that represent respectively the integral and fractional parts of a positive octal number o, converts them to base 10 and outputs the results as 2 decimal numbers, E10 and F10 that represent respectively the integral and fractional parts of the positive decimal number d, using Nested Polynomial Evaluation.

At the end, this function should also display d in its normalized floating point form.

Solution: **12 points**

```
function [E10 , F10 , d]=Convert8to10(E8, F8)

format long e

% Conversion of the Integral part of o from octal to decimal, using nested poly eval
n=length(E8);
if n~=0                                  3 points
    E10=E8(1);
    for i=2:n
        E10=E10*8 +E8(i);
    end
else
    E10=0;                               1 point
end

% Conversion of the Fractional part of o from octal to decimal, using nested poly eval
N=length(F8);
if N~=0                                  4 points
    F10=F8(1);
    for i=2:N
           F10=F10*8 + F8(i);
    end
    F10=8^(-N)*F10;
else
    F10=0;                               1 point
end

% Get d
d = E10 + F10;                           1 point
d = num2str(d,'%.16e');                  2 points
```

Test cases: **3 points**

>> E8 = [5 3 3]; F8 = [0 7 4 5]; [E10 , F10 , d]=Convert8to10(E8, F8) E10 = 347 F10 = 1.184082031250000e-001 d = 3.4711840820312500e+002	>> E8 = []; F8 = [0 7 4 5]; [E10 , F10 , d]=Convert8to10(E8, F8) E10 = 0 F10 = 1.184082031250000e-001 d = 1.1840820312500000e-001	>> E8 = [5 3 3]; F8 = []; [E10 , F10 , d]=Convert8to10(E8, F8) E10 = 347 F10 = 0 d = 3.4700000000000000e+002

Exercise 2 : Successors and Rounding Procedures 35 points

Question 1: 11 points **Question 2: 6 points** **Question 3: 13 points**

Let $x = +m \times 10^{ex}$ be a positive decimal number in F(10, p, e_{min}, e_{max}), written in normalized floating point form, with $-20 \le ex < +20$ and $p < 15$.

Question 1 11 points

Write a MATLAB function:

function [my, ey] = GetSuccessor(mx, ex, p)

which takes as inputs:

- mx : the mantissa of x, in standard normalized floating point notation.
- ex : the exponent of x.
- p : the precision of the floating point system to which x belongs (p<15).

Let y be the successor of x in F(10, p, - 20, +20). This function should output:

- my : the mantissa of y, displayed with a precision p *(the non significant digits of the fractional part need not to be displayed)*.
  HINT : first compute my, then use **num2str(my,p)** to output my in the required format.
- ey : the exponent of y

Solution: **8 points**

```
function [my, ey]=GetSuccessor(mx, ex, p)

epsm = 10^(-p+1);                 2 points
Mmax=10 - epsm;
if mx < Mmax                      3 points
    my = mx + epsm;
    my = num2str(my,p);
    ey = ex;
elseif mx == Mmax                 3 points
    my = 1;
    my = num2str(my,p);
    ey = ex + 1;
end
```

Test cases: **3 points**

>> mx=7.300; ex=2; p=4; >> [my, ey]=GetSuccessor(mx, ex, p) my = 7.301 ey = 2	>> mx=7.300000; ex=2; p=7; >> [my, ey]= GetSuccessor (mx, ex, p) my = 7.300001 ey = 2	>> mx=7.300000000; ex=2; p=10; >> [my, ey]= GetSuccessor (mx, ex, p) my = 7.300000001 ey = 2
>> mx=7.3000000000000; ex=2; p=14; >> [my, ey]= GetSuccessor (mx, ex, p) my = 7.3000000000001 ey = 2	>> mx=9.999999;ex=-3;p=7; >> [my, ey]= GetSuccessor (mx, ex, p) my = 1 ey = -2	>> mx=7.999999;ex=-3;p=7; >> [my, ey]= GetSuccessor (mx, ex, p) my = 8 ey = -3

Question 2 **6 points**

Let m = $+m_1.m_2m_3\ldots m_p$ be a positive decimal number whose integral part is m_1, and whose fractional part is $0.m_2m_3\ldots m_p$ ($m < 15$)
Write a MATLAB function:

function m = ConvertVectortoDecimal(M)

which takes as input a vector M of length p whose i^{th} component is the decimal digit m_i, for i = 1, …, p, and whose output is the decimal number m represented by M.
Use format long g *for displaying the output in double precision discarding the non significant zeros of the fractional part.*

Solution: **4 points**

```
function m = ConvertVectortoDecimal(M)

format long g
p=length(M);
m=M(1);
t=10^(-1);
for i=2:p
    m = m + M(i)* t ;
    t = t / 10 ;
end
```

Test case: **2 points**

```
>> m = ConvertVectortoDecimal([9,4,5,0,6,2,3])
m =
  9.450623
```

Question 3 **13 points**

Write a MATLAB function :

function [mz, ez] = Round(Mx, ex, n, t)

which takes as inputs:

- Mx : a vector of length p whose components represent the mantissa mx of the decimal number $x \in F(10, p, -20, +20)$,
- ex : the exponent of x,
- n : a positive integer less than or equal to p ($n \leq p$), representing the precision required to reach,
- t : a parameter taking the values 1 or 2.

This function should compute z: the representative of x in F(10, n,- 20, +20) by rounding x to the closest if t = 1 or by chopping if t = 2, and output:

- mz : the mantissa of z, displayed with a precision n,
  HINT: first compute mz, then use **num2str(mz,n)** to output mz in the required format (the non significant zeros of the fractional part will be discarded) .
- ez : the exponent of z.

At the end your function should also display z in normalized floating point representation in F(10, n, - 20; +20).

Solution: **10 points**

```
function[mz, ez] = Round(Mx, ex, n, t)

p = length(Mx) ;
mx = ConvertVectortoDecimal(Mx);          % convert vector to decimal
if n == p                                                          2 points
    mz = mx;
    mz = num2str(mz,n);
    ez = ex;
elseif n < p
    if t==2 || (t==1 && Mx(n+1) < 5)      % in that case round by chopping    3 points
        Mx = Mx(1:n) ;
        mx = ConvertVectortoDecimal(Mx);  % convert new vector to decimal
        mz = mx;
        mz = num2str(mz,n);
        ez = ex;
    elseif t==1 && Mx(n+1) >=5            % in that case round to the closest 3 points
        Mx = Mx(1:n) ;
        mx = ConvertVectortoDecimal(Mx);  % convert new vector to decimal
        [mz, ez]=GetSuccessor(mx, ex, n);
    end
end
disp(['z = +', mz, ' x 10 ^ ', num2str(ez) ])                      2 points
```

Test cases: **3 points**

>> [mz, ez]=Round([9,4,5,0,0,2,3], 3, 7, 1) z = +9.450023 x 10 ^ 3 mz = 9.450023 ez = 3	>> [mz, ez]=Round([9,4,5,0,0,2,3], 3, 6, 1) z = +9.45002 x 10 ^ 3 mz = 9.45002 ez = 3
>> [mz, ez]=Round([9,4,5,0,0,2,3], 3, 5, 1) z = +9.45 x 10 ^ 3 mz = 9.45 ez = 3	>> [mz, ez]=Round([9,4,5,7,8,9,3], 3, 5, 1) z = +9.4579 x 10 ^ 3 mz = 9.4579 ez = 3

Exercise 3 : Root finding methods — 40 points

Question 1: 18 points **Question 2: 12 points** **Question 3: 10 points**

The aim of this exercise is to approximate π by computing the root of f(x) = sin(x) in the interval (3, 4), based on Newton's method. For that purpose:

Question 1 — 18 points

Write 2 MATLAB functions:

function [sinx] = mysin(x,p)
function [cosx] = mycos(x,p)

that both input:

- a variable x representing some angle in radians
- a precision p.

Using Taylor's series expansion, these functions should compute respectively the sine and cosine of x, up to some tolerance $Tol = 0.5 \times 10^{-p+1}$, and output respectively:

- the values of sinx and cosx in F(10, p, -20, +20).

HINT: first compute sinx and cosx, then use num2str(. , p).

Note that:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots$$

N.B. Do not use the Matlab built-in function for the factorials

Solution: **14 points = 7 points + 7 points**

```
function sinx = mysin(x,p)

Tol = 0.5*10^(-p+1);                                    0.5 point

sinx = 0;
X = x^2;
t = x;
fact = 1;
err = t/fact;
k = 0;

while abs(err) > Tol
    sinx = sinx + err;  %i.e. + t/fact;                 1.5 points
    fact = fact*(2*k+2)*(2*k+3);                        1.5 points
    t = - t*X;          % or t = - t*X;                 1.5 points
    err = t/fact;       % or k = k+1;                   1.5 points
    k = k+1;            % or  err = (-1)^k*t/fact;
end

sinx = num2str(sinx,p);                                 0.5 point
```

```
function cosx = mycos(x,p)

Tol = 0.5*10^(-p+1);                    0.5 point

cosx = 1;
X = x^2;
t = - X;
fact = 2;
err = t/fact;
k = 1;

while abs(err) > Tol
    cosx = cosx + err;                  1.5 points
    fact = fact*(2*k+1)*(2*k+2);        1.5 points
    t = -t*X;                           1.5 points
    err = t/fact;                       1.5 points
    k = k+1;
end
cosx = num2str(cosx,p);                 0.5 point
```

Test cases: **4 points**

>> sinx = mysin(pi/6,14) sinx = 0.49999999999996 >> sinx = mysin(pi/4,14) sinx = 0.70710678118657 >> sinx = mysin(pi/3,14) sinx = 0.86602540378443	>> cosx = mycos(pi/6,14) cosx = 0.86602540378444 >> cosx = mycos(pi/4,14) cosx = 0.70710678118655 >> cosx = mycos(pi/3,14) cosx = 0.5

Question 2 12 points

Write a MATLAB function:

function [r,k] = myNewton(f,df,a,b,p,kmax)

that takes as inputs:

- a function f and its derivative df (as function handles),
- 2 real numbers a and b, where (a, b) is the interval locating the root of f,
- a precision p,
- and a maximum number of iterations kmax.

Based on Newton's method, this function should output:

- the approximation of the root r of f up to p decimal figures
  HINT: first compute r, then use num2str(r, p)
- k : the number of iterations used to reach the required precision p.

$Tol = 0.5 \times 10^{-p+1}$ is the relative error bound to the computed root.

Solution: 7 points

```
function [r,k] = myNewton(f,df,a,b,p,kmax)
Tol = 0.5*10^(-p+1);
R(1)=(a+b)/2;                                   1 point
k = 1;
RelErr = inf;
while RelErr > Tol && k<kmax                    4 points
    R(k+1) = R(k) - f(R(k))/df(R(k));
    RelErr = abs(R(k+1)-R(k)) / abs(R(k));
    k = k+1;
end
r = R(k);
r = num2str(r, p);                              1 point
if k>=kmax
    disp ( ' error, no convergence' ) ;         1 point
end
```

Test cases: 5 points

```
>> f=@(x)(x^2-2);
>> df = @(x)(2*x);
>> [r,k] = myNewton(f,df,1,2,3,20)
r =
1.41
k =
   3
>> [r,k] = myNewton(f,df,1,2,4,20)
r =
1.414
k =
   4
>> [r,k] = myNewton(f,df,1,2,7,20)
r =
1.414214
k =
   5
>> [r,k] = myNewton(f,df,1,2,10,20)
r =
1.414213562
k =
   5
>> [r,k] = myNewton(f,df,1,2,15,20)
r =
1.41421356237309
k =
   6
```

```
>> f=@(x)(x^2-2);
>> df = @(x)(2*x);
>> [r,k] = myNewton(f,df,-2,-1,3,20)
r =
-1.41
k =
   3
>> [r,k] = myNewton(f,df,-2,-1,4,20)
r =
-1.414
k =
   4
>> [r,k] = myNewton(f,df,-2,-1,7,20)
r =
-1.414214
k =
   5
>> [r,k] = myNewton(f,df,-2,-1,10,20)
r =
-1.414213562
k =
   5
>> [r,k] = myNewton(f,df,-2,-1,15,20)
r =
-1.41421356237309
k =
   6
```

Question 3 **10 points**

Write a MATLAB function:

function [mypi,errpi,k] = mypiNewton(p,kmax)

that takes as inputs the parameters p, kmax as defined in the previous question.

Using the previous function myNewton on the interval (3,4), and the functions mysin and mycos programmed in part 1, this function applies Newton's method for getting the approximation of π as a root of sinx, and outputs:

- pi : the approximation to π, up to p decimal figures,
- errpi : the absolute error |mypi - π| where π is the built-in variable of Matlab, computed in double precision,
- k : the number of iterations used in Newton's method to reach the precision p.

HINT: Note that after calling the functions myNewton, mysin and mycos , their outputs should be converted back to numbers using the command str2num(.)

Solution: **7 points**

```
function [mypi,errpi,k] = mypiNewton(p,kmax)

f=@(x)(str2num(mysin(x,p)));                    3 points
df=@(x)(str2num(mycos(x,p)));
a = 3;
b = 4;

[r,k] = myNewton(f,df,a,b,p,kmax);              2 points

mypi = str2num(r);                              2 points
errpi = abs(mypi - pi);
```

Test cases: **3 points**

>> [mypi,errpi,k] = mypiNewton(2,20) mypi = 3.1 errpi = 0.041592653589793 k = 3	>> [mypi,errpi,k] = mypiNewton(10,20) mypi = 3.141592654 errpi = 4.10206979495342e-010 k = 5
>> [mypi,errpi,k] = mypiNewton(3,20) mypi = 3.14 errpi = 0.00159265358979299 k = 4	>> [mypi,errpi,k] = mypiNewton(15,20) mypi = 3.1415926535898 errpi = 7.105427357601e-015 k = 5
>> [mypi,errpi,k] = mypiNewton(7,20) mypi = 3.141593 errpi = 3.46410206741865e-007 k = 4	>> [mypi,errpi,k] = mypiNewton(16,20) mypi = 3.14159265358979 errpi = 0 k = 5