SCILAB CODE
PRACTIAL 2.A
ROOT USING BISECTION METHOD

```
clc //clear console screen
clear //clear memory data

// Bisection : Compute root using bisection method
function root=bisection(f, a, b)

////Input : Bisection(f,a,b)
// f = Input Function; a = initial iteration Root ; b = Final iteration Root
// Output = Root

   tol=1.e-4;
//tol=Maximum error between iterations that can be tolerated
   maxit=50; // maxit = Maximum number of Iteration
   Ea = 1; // Assume Initials error is 1
   n = 1; // start with First iteration

// check for correct root iteration for correct root f(a)*f(b)<0
if (f(a(n))*f(b(n))) > 0 then
   disp("root of the equation does not belong to the given interval ");
   abort;

else while (1) //using while loop to do iteration method
      c(n) = (a(n) + b(n)) / 2; //bisection formula for nth value
// choose the root such that f(a)*f(c)<0 or f(b)*f(c)<0
if (f(a(n))*f(c(n))) < 0 then
         b(n+1) = c(n); // value is negative w.r.t. a so replace b with c
         a(n+1) = a(n);
      elseif (f(b(n))*f(c(n))) < 0 then
```

```
      a(n+1) = c(n); // value is negative w.r.t. b so replace a with c
      b(n+1) = b(n);
    else
      break //If value of function is Zero than stop process
    end

//find % error in iteration using abs(Absolute) to make value positive
    if n>1 then
      Ea(n) = 100 * abs((c(n)-c(n - 1)) / c(n));
     end

// Checking condition whether error is less than Tolerated value OR max
iteration reached
     if Ea(n) < tol | n==maxit then
       break
     end

// process end if any one of the condition satisfied else continue for next
iteration
    n = n + 1;
  end
end

// last value of c will give output as root once while loop stop
  root = c(n);
endfunction

// Define Function to find root of equation
function f=f1(x)
  f = x^3 - 2*x^2 - 2*x - 1;
endfunction
```

```
//Define Initial Value 'a' and 'b'
a=2;
b=3;

root = bisection(f1,a,b);
root = round(root*10^5)/10^5; //Round the final answer upto 5
decimal point as ^5
disp(root,"root of the equation = ")
```

Output :

root of the equation =

2.83117

{ With correct initial value 'a' and 'b' }

root of the equation does not belong to the given interval

{ With Wrong initial value 'a' and 'b' }

Please Note : Answer can varies with change in number of iteration

PRACTIAL 2.B/C

ROOT USING REGULA FALSI METHOD OR FALSE POSITION METHOD OR SECANTS METHOD

```
clc //clear console screen
clear //clear memory data

// RegulaFalsi : Compute root using regulafalsi method
function root=regulafalsi(f, a, b)

////Input : regulafalsi(f,a,b)
// f = Input Function; a = initial iteration Root ; b = Final iteration Root
// Output = Root

   tol=1.e-4; //tol=Maximum error between iterations that can be
tolerated
   maxit=5; // maxit = Maximum number of Iteration
   Ea = 1; // Assume Initials error is 1
   n = 1; // start with First iteration

// check for correct root iteration for correct root f(a)*f(b)<0
if (f(a(n))*f(b(n))) > 0 then
   disp("root of the equation does not belong to the given interval ");
   abort;

else while (1) //using while loop to do iteration method
c(n)=((a(n)*f(b(n)))-(b(n)*f(a(n))))/(f(b(n))-f(a(n)));
//regulafalsi formula for nth value

// choose the root such that f(a)*f(c)<0 or f(b)*f(c)<0
if (f(a(n))*f(c(n))) < 0 then
        b(n+1) = c(n); // value is negative w.r.t. a so replace b with c
```

```
        a(n+1) = a(n);
      elseif (f(b(n))*f(c(n))) < 0 then
        a(n+1) = c(n); // value is negative w.r.t. b so replace a with c
        b(n+1) = b(n);
      else
        break //If value of function is Zero than stop process
      end

//find % error in iteration using abs(Absolute) to make value positive
      if n>1 then
        Ea(n) = 100 * abs((c(n)-c(n - 1)) / c(n));
       end

// Checking condition whether error is less than Tolerated value OR max
iteration reached
       if Ea(n) < tol | n==maxit then
         break
       end

// process end if any one of the condition satisfied else continue for next
iteration
      n = n + 1;
   end
end

// last value of c will give output as root once while loop stop
   root = c(n);
endfunction

// Define Function to find root of equation
function f=f1(x)
   f = x^3 - 2*x^2 - 2*x - 1;
endfunction
```

```
//Define Initial Value 'a' and 'b'
a=2;
b=3;

root = regulafalsi(f1,a,b);
root = round(root*10^5)/10^5; //Round the final answer upto 5
decimal point as ^5
disp(root,"root of the equation = ")
```

Output :

root of the equation =

2.83117

{ With correct initial value 'a' and 'b' }

root of the equation does not belong to the given interval

{ With Wrong initial value 'a' and 'b' }

Please Note : Answer can varies with change in number of iteration

PRACTIAL 2.D
ROOT USING NEWTON RAPHSON METHOD

```
clc //clear console screen
clear //clear memory data

// Newton Raphson : Compute root using Regula Falsi method
function root=newtonraphson(f, df)

////Input : Newtonraphson(f,df)
// f = Input Function; df=Derivative of function
// Output = Root

   tol=1.e-4; //tol=Maximum error between iterations that can be
tolerated
   maxit=5; // maxit = Maximum number of Iteration
   n = 1; // start with First iteration
   x(n) = 3; // Assume any initial value from the near root

while (1) //using while loop to do iteration method
      x(n+1) = x(n)-(f(x(n))/df(x(n))); //Newton Raphson formula
for nth value

// Checking condition whether max iteration reached
       if n==maxit then
          break
       end

// process end if condition satisfied else continue for next iteration
```

```
    n = n + 1;
  end

// last value of c will give output as root once while loop stop
  root = x(n+1);
endfunction

// Define Function to find root of equation
function f=f1(x)
  f = x^3 - 2*x^2 - 2*x - 1;
endfunction

// Define Derivative of Function to find root of equation
function df=df1(x)
  df = 3*x^2 - 4*x - 2;
endfunction

root = newtonraphson(f1,df1);
root = round(root*10^5)/10^5; //Round the final answer upto 5
decimal point as ^5
disp(root,"root of the equation = ")
```

Output :

root of the equation =

2.83117

{ With correct initial value 'a' and 'b' }

Please Note : Answer can varies with change in number of iteration

SCILAB CODE
PRACTIAL 3.A1
CREATE FORWARD DIFFERENCE TABLE

```
clc // Clear your console screen
clear //Clear your previous Data

x = 10:10:50; // Value of x from table such as start:interval:final
y = [46 66 81 93 101]; // write value of y in matrix form

n =length(x); // counting value of n = (final - initial)/h, h=Interval

del = %nan *ones(n ,5) ; // Find del = %nan = not an integer and
ones = returns a (m1,m2) matrix full of ones.

del (: ,1) = y'; //First column of del count as dy/dy = y'

for j = 2:6 //Write data difference in matrix from 2nd column
onwards --> d2y, d3y & so on upto d5y as taken last till 6..

for i = 1 : n -j +1 //write data differences in row

del (i ,j) = del (i +1 ,j -1) - del (i ,j -1) ; //return value w.r.t (row,
column)

end

end
```

```
del (: ,1) =  [ ]; //return total Matrix in del

del =round(del *10^3) /10^3; // round upto 3 decimal places
mprintf( "%5s %7s %8s %9s %8s %8s %8s" , ' x ' , ' y ' , ' dy ' , 'd2y
' , ' d3y ' , ' d4y ' , ' d5y ' )
//print value in form of x, y, dy, d2y, d3y d4y and so on
//%5s means 5 space between dy d2y %7s mean 7 space between
d2y d3y & so on

disp([x' y' del]) //Output x y Del(matrix form)
```

Output :

x	y	dy	d2y	d3y	d4y	d5y
10.	46.	20.	- 5.	2.	- 3.	
20.	66.	15.	- 3.	- 1.	Nan	
30.	81.	12.	- 4.	Nan	Nan	
40.	93.	8.	Nan	Nan	Nan	
50.	101.	Nan	Nan	Nan	Nan	

PRACTIAL 3.B 1
CREATE BACKWARD DIFFERENCE TABLE

```
clc // Clear your console screen
clear //Clear your previous Data

x = 1:1:8; // Value of x from table such as start:interval:final
y = x^3; // write value of y in matrix form
n =length(x); // counting value of n = (final - initial)/h h=Interval

del = %nan *ones(n ,6) ;// Find del = %nan = not an integer and
ones = returns a (m1,m2) matrix full of ones upto d5y since we take
6.
del (: ,1) = y'; //First column of del count as dy/dy = y'

for j = 2:6 //Write data difference in matrix from 2nd column
onwards --> d2y, d3y & so on d5y as taken last till 6..
for i = 1:n -j +1 //write data differences in row
del (i+j -1 , j) = del (i+j -1 ,j -1) - del (i+j -2 ,j-1) ; //return value
w.r.t (row, column)
end
end

del (: ,1) =  [ ]; //return total metrix in del
del =round( del *10^3) /10^3; // round upto 3 deimal places
mprintf( "%5s %7s %8s %9s %8s %8s %8s" , ' x ' , ' y ' , ' dy ' , 'd2y
' , ' d3y ' , ' d4y ' , ' d5y ' )
//print value in form of x, y, dy, d2y, d3y d4y and so on
//%5s means 5 space between dy d2y %7s mean 7 space between
d2y d3y & so on
```

```
disp([x' y' del]) //Output x y Del(matrix form)
```

Output :

x	y	dy	d2y	d3y	d4y	d5y
1.	1.	Nan	Nan	Nan	Nan	Nan
2.	8.	7.	Nan	Nan	Nan	Nan
3.	27.	19.	12.	Nan	Nan	Nan
4.	64.	37.	18.	6.	Nan	Nan
5.	125.	61.	24.	6.	0.	Nan
6.	216.	91.	30.	6.	0.	0.
7.	343.	127.	36.	6.	0.	0.
8.	512.	169.	42.	6.	0.	0.

PRACTIAL 3.A 2
NEWTON FORWARD DIFFERENCE FORMULA

```
clc // Clear your console screen
clear //Clear your previous Data

x = 10:10:50; // Value of x from table such as start : interval : final
y = [46 66 81 93 101]; // write value of y in matrix form

n =length(x); // counting value of n = (final - initial)/h h=Interval

del = %nan *ones(n ,5) ; // Find del = %nan = not an integer and
ones = returns a (m1,m2) matrix full of ones last till 4. Take only
upto value is exist else output is nan..
del (: ,1) = y'; //First column of del count as dy/dx = y'

for j = 2:5 //Write data difference in matrix from 2nd column
onwards --> d2y, d3y & so on d5y as taken last till 5..
   for i = 1:n -j +1 //write data differences in row
      del (i ,j) = del (i +1 ,j -1) - del (i ,j -1) ; //return value w.r.t (row,
column)
   end
end
del =round( del *10^3) /10^3; // round upto 3 decimal places
mprintf( "%5s %7s %8s %9s %8s %8s " , ' x ' , ' y ' , ' dy ' , 'd2y ' , '
d3y ' , ' d4y ')
//print value in form of x, y, dy, d2y, d3y d4y and so on
//%5s means 5 space between dy d2y %7s mean 7 space between
d2y d3y & so on
disp([x' del]) //Output x y Del(matrix form)
```

```
del (: ,1) = []; // Select first value of each term
X = 12; // Input value of x to be find
h = x (2) - x (1) ; // Find Interval h
p = (X -x (1) ) / h;// find value of p required for formula
x0 = x (1) ; // consider First value
y0 = y (1) ;
dely0 = del (1 ,:) ;
Y = y0 ;
for i = 1:length( dely0 ) // apply for loop for formula with different
value of p and add
   t = 1;
   for j = 1: i
   t = t * (p -j +1) ;
   end
Y = Y + t* dely0 (i)/ factorial (i);
end
disp(Y , "f ( 1 2 ) = " )
```

Output :

x	y	dy	d2y	d3y	d4y	d5y
10.	46.	20.	- 5.	2.	- 3.	
20.	66.	15.	- 3.	- 1.	Nan	
30.	81.	12.	- 4.	Nan	Nan	
40.	93.	8.	Nan	Nan	Nan	
50.	101.	Nan	Nan	Nan	Nan	

f (1 2) = 50.5968

PRACTIAL 3.B 2
NEWTON BACKWARD DIFFERENCE FORMULA

```
clc // Clear your console screen
clear //Clear your previous Data

x = 10:10:50; // Value of x from table such as start:interval:final
y = [46 66 81 93 101]; // write value of y in matrix

n =length(x); // counting value of n = (final - initial)/h h=Interval
del = %nan *ones(n ,5) ; // Find del = %nan = not an integer and
ones = returns a (m1,m2) matrix full of ones last till 4. Take only
upto value is exist else output is nan..
del (: ,1) = y'; //First column of del count as dy/dx = y'

for j = 2:5 //Write data difference in matrix from 2nd column
onwards --> d2y, d3y & so on d5y as taken last till 5..
for i = 1:n -j +1 //write data differences in row
del (i+j -1 , j) = del (i+j -1 ,j -1) - del (i+j -2 ,j-1) ;//return value w.r.t
(row, column)
end
end

del =round( del *10^3) /10^3; // round upto 3 decimal places

mprintf( "%5s %7s %8s %9s %8s %8s " , ' x ' , ' y ' , ' dy ' , 'd2y ' , '
d3y ' , ' d4y ')
//print value in form of x, y, dy, d2y, d3y d4y and so on
//%5s means 5 space between dy d2y %7s mean 7 space between
d2y d3y & so on
disp([x' del]) //Output x y Del(matrix form)

X = 42; // put the value of x for which we hav eto find y
```

```
h = x (2) - x (1) ; // Find the interval
p = (X -x(n)) / h; // Find value of P as required in formula
xn = x(n); // Consider last data of column
yn = y(n);
delyn = del (n ,:) ;

Y = 0;  // just define intital value of y

for i = 0:length( delyn ) -1
t = 1;
for j = 1: i //Apply formula with p and add
t = t * (p + j -1) ;
end
Y = Y + t* delyn (i +1) / factorial (i);
end
disp(Y , "y ( 4 2 ) = " ) // Display result
```

Output :

x	y	dy	d2y	d3y	d4y	d5y
10.	46.	20.	- 5.	2.	- 3.	
20.	66.	15.	- 3.	- 1.	Nan	
30.	81.	12.	- 4.	Nan	Nan	
40.	93.	8.	Nan	Nan	Nan	
50.	101.	Nan	Nan	Nan	Nan	

f (4 2) = 95.0048

PRACTIAL 3.C
LAGRANGES FORMULA

```
clc // Clear Screen
clear //Clear previous data

//put coordinate of x and y
x = [0 1 2 4];
y = [1 1 2 5];
n =length(x); //Find total number of x

del = %nan *ones(n ,4) ;  // Find del = %nan = not an integer and
ones = returns a (m1,m2) matrix full of ones last till 4. Take only
upto value is exist else output is nan..
del (: ,1) = y'; //First column of del count as dy/dy = y'

for j = 2:4 //Write data difference in matrix from 2nd column
onwards --> d2y, d3y & so on d5y as taken last till 4..
for i = 1:n -j +1 //write data differences in row
del (i ,j) = ( del (i +1 ,j -1) - del (i ,j -1) ) / (x(i+j -1) - x(i)); //return
value w.r.t (row, column)
end
end

del (: ,1) = []; // use del as of each when needed

Y = 0; // Assume initial value of y as 0
X = 2.3; // put the value of x for which we hav eto find y
for i = 1: n // Apply formula
t = x;
t(i) = [];
p = 1;
for j = 1:length(t) //Apply formula with p and add
```

```
p = p * (X -t(j)) /( x(i) -t(j));
end
Y = Y + p*y(i);
end
disp(round(Y *10^4) /10^4 ," f ( 2 . 3 ) " ) // Display Result
```

Output :

f (2 . 3)

---→ 2.4202

PRACTICAL 4.A
GUASS JORDAN METHOD

```
clc
clear
// Enter Coefficient of Matrix A i.e. Coefficient x,y,z,
A = [1 2 1; 2 3 4; 4 3 2];
// Enter R.H.S. Value
B = [8; 20; 16];

// Calculate Number of Row and Column
n =length(B);

//Write in terms of Augmented Matrix
Aug = [A , B ];

//Forward Elimination or Row Operation to make lower Triangle
Element Zero
for j = 1:n -1
for i = j +1: n
Aug (i ,j:n +1) = Aug (i ,j:n +1) - Aug (i ,j) / Aug(j ,j) * Aug (j ,j:n +1)
;
//mprintf("\n k1 = %.2f x f ( %.4f , %.4f ) ",h,x0,y0)
//mprintf("\n R%i - %i R%i",i,n,Aug (i ,j) / Aug(j ,j))
disp(Aug)
end
end

//Backward Elimination or Row Operation to make Upper Triangle
element Zero
for j = n : -1:2
```

```
Aug (1: j -1 ,:) = Aug (1: j -1 ,:) - Aug (1: j -1 , j) / Aug(j ,j) * Aug (j ,:)
;
disp(Aug)
end

//Diagonal Normalization
for j =1: n
Aug (j ,:) = Aug (j ,:) / Aug (j ,j);
end

// Compare A with B
x = Aug (: , n +1) ;

//Display Solution of equation
disp("The Solution of Given Equations are :")
disp(strcat([ "x = " ,string(x (1) ) ]) )
disp(strcat([ "y = " ,string(x (2) ) ]) )
disp(strcat([ "z = " ,string(x (3) ) ]) )
```

Output :

```
1.   2.   1.   8.
0.  - 1.   2.   4.
4.   3.   2.   16.


1.   2.   1.   8.
0.  - 1.   2.   4.
0.  - 5.  - 2.  - 16.
```

$$\begin{matrix} 1. & 2. & 1. & 8. \\ 0. & -1. & 2. & 4. \\ 0. & 0. & -12. & -36. \end{matrix}$$

$$\begin{matrix} 1. & 2. & 0. & 5. \\ 0. & -1. & 0. & -2. \\ 0. & 0. & -12. & -36. \end{matrix}$$

$$\begin{matrix} 1. & 0. & 0. & 1. \\ 0. & -1. & 0. & -2. \\ 0. & 0. & -12. & -36. \end{matrix}$$

The Solution of Given Equations are :

$x = 1$

$y = 2$

$z = 3$

PRACTICAL 4.B
GUASS SEIDEL METHOD

```
clc
clear

// Enter Coefficient of Matrix A i.e. Coefficient x,y,z,
A = [10 1 1; 2 10 1; 2 2 10];
// Enter R.H.S. Value
B = [12; 13; 14];

// Calculate Number of Row and Column
n =length(B);

tol = 1e -4;
iter = 1;
maxit = 5;

//Intial guess
x =zeros(n ,1) ;

//Assuming to avoid variable size error
E =ones(n ,1) ;

S =diag(diag(A));
T = S-A;
xold = x;
while(1)
for i = 1: n
x(i , iter +1) = (B(i) + T(i ,:) * xold ) / A(i ,i);
E(i , iter +1) = (x(i , iter +1) - xold (i))/x(i , iter +1) *100;
xold (i) = x(i , iter +1) ;
end
```

```
if x (: , iter ) == 0
E = 1;
else
   E =sqrt((sum(( E (: , iter +1) ).^2) )/n);
end

if E <= tol | iter == maxit
break
end

iter = iter + 1;
end

X = x (: , iter );
x =round(x *10^5) /10^5;
x (: ,1) = [];

mprintf( ' %s %3s %11s %10s','Iteration No.','x1','x2','x3');

disp([(1: iter )' x']) ;
```

Output :

Iteration No.	x1	x2	x3
1.	1.2	1.06	0.948
2.	0.9992	1.00536	0.99909
3.	0.99956	1.00018	1.00005
4.	0.99998	1.	1.
5.	1.	1.	1.

PRACTICAL 6.A
TRAPEZOIDAL METHOD

```
clc;
clear;

// Value of x from table of numerical integration
x =[0 0.25 0.5 0.75 1];

//Value of y or f(x)
y =[1 0.9412 0.8 0.64 0.5];

// Find h (interval)
h=x(2)-x(1) ;

// Find total number of terms
n=length(x);

//Assume initial variable as Zero
area =0;

// Use for loop to add all value one by one

for i =1 : n

   if i ==1 | i == n then
      area = area + y (i) // For first and last value directly add
   else
      area = area +2* y (i) // Remaining terms multiply by 2
   end
end

// By trapezoidal formula multiply by (h/2)
```

```
area = area *( h /2) ;

printf( 'Value of Intergration By Trapezoidal Rule is = %f ' , area );
```

OUTPUT :

VALUE OF INTERGRATION BY TRAPEZOIDAL RULE IS = 0.782800

PRACTICAL 6.B
SIMPSON'S 1/3 RD METHOD

```
clc;
clear;

// Value of x from table of numerical integration
x =[0 0.25 0.5 0.75 1];

//Value of y or f(x)
y =[1 0.9412 0.8 0.64 0.5];

// Find h (interval)
h=x(2)-x(1) ;

// Find total number of terms
n=length(x);

//Assume initial variable as Zero
area =0;

// Use for loop to add all value one by one
for i = 1: n
   if i == 1| i == n then
      area = area +y(i) // For first and last value directly add
```

```
    elseif(modulo(i-1 ,2)) == 0 then
       area = area +2* y(i)  // For Even Number terms
    elseif(modulo(i-1 ,2)) ~= 0 then
       area = area +4* y(i)  // For Odd number terms
end

end
// By Simpson's 1/3rd formula multiply by (h/3)
area = area * (h/3);

printf( 'Value of Intergration By Simpsons 1/3rd Rule is = %f ' ,
area );
```

OUTPUT : VALUE OF INTERGRATION BY SIMPSONS 1/3RD RULE IS = 0.785400

PRACTICAL 6.C
SIMPSPNS 3/8TH METHOD

```
clc;
clear;

// Value of x from table of numerical integration
x =[0 0.25 0.5 0.75 1];

//Value of y or f(x)
y =[1 0.9412 0.8 0.64 0.5];

// Find h (interval)
h=x(2)-x(1) ;

// Find total number of terms
```

```
n=length(x);

//Assume initial variable as Zero
area =0;

// Use for loop to add all value one by one
for i = 1: n
   if i == 1| i == n then
      area = area +y(i) // For first and last value directly add
   elseif(modulo(i-1,3)) == 0 then
      area = area +2* y(i)  // For Multiple of 3
   else
      area = area +3* y(i)  // For Remaining terms
end

end
// By Simpson's 3/8 rd formula multiply by (3h/8)
area = area * (3*h/8);

printf( 'Value of Intergration By Simpsons 3/8th is = %f ' , area );
```

OUTPUT : VALUE OF INTERGRATION BY SIMPSONS 3/8TH IS = 0.750338

PRACTICAL 7.A
EULER'S METHOD

```
clc
clear
function [f]=dydx(x, y)
 f = (y-x) / (y+x);
 // Define the given function in question
endfunction
y0 = 1; // initial value already given
x0 = 0; // initial value already given
x = 0.4; // Finding value already given
n = 5; // Decide number of iteration to perform
h = (x-x0)/n;
xx = x0; // storing value in variable
y = y0; // storing value in variable
for i = 1:n // using for loop to perform iteration
      y = y +h* dydx (xx ,y); // Eulers Formula
   y = round (y *10^4) /10^4; // Rounding the number upto 4
decimal places
   xx = xx + h; // value of x1
end
//Display output
disp (y,"y ( x = 0 . 4 ) = ")
```

OUTPUT :

Y (X = 0 . 4) =
1.3106

PRACTICAL 7.B
EULER'S MODIFICATION METHOD

```
clc
clear

function [f]=dydt(x, y)
f = x + y; // Define the given function
endfunction

y0 = 1; // Initial value already given
x0 = 0; // Initial value already given
h = 0.1; // Initial value we have to assume or decide
x = 0.2; // Finding value already given
n = (x-x0)/h; // to find number of required iteration
xx = x0; // Define value to the variable

for i = 1:n // for loop to find n value of the eqation
y11 = y0 + h* dydt (xx ,y0); // First step same as euler formula
x1 = xx + h; // Finding value of x
// using modification formula to correct the- answer we taking 4
step to correct accurate
// No. of step to correct is decide by self
y12 = y0 + h /2*( dydt (xx ,y0) + dydt (x1 , y11 ));
y13 = y0 + h /2*( dydt (xx ,y0) + dydt (x1 , y12 ));
y14 = y0 + h /2*( dydt (xx ,y0) + dydt (x1 , y13 ));
y1 = y0 + h /2*( dydt (xx ,y0) + dydt (x1 , y14 ));
// Rounding final answer upto 4 digits
y1 = round (y1 *10^4) /10^4;
// For table output arrange value
y(i) = y1;
// assign new value to original variable for new iteration
y0 = y1;
```

```
xx = x1;

end
//Display Output
mprintf ("%5s %8s" , ' x ' , ' y ' )
disp ([[( x0+h:h:x)' y]])
```

OUTPUT :

X	Y
0.1	1.1105
0.2	1.2432

PRACTICAL 7.C 1
R-K 2ND ORDER METHOD

```
clc
clear
// Define the given function
function [f]=fun1(x, y)
f = (y/2) +(3*x);
endfunction

// Define the given initial value
x0 = 0;
y0 = 1;
h = 0.1;
x = 0.2;
// No of iteration required
n = (x-x0)/h;

// Calling Function - always defined first
// Alternatively you can also use formula directly in for loop
without calling function
```

```
// Define method to solve rk 2 order using formula or define
formula
function [f]=rk2(x, y)
k1 = h* fun1 (x,y);
k2 = h* fun1 (x + h/2,y + k1);
f = y + 1/2*(k1+k2);
// Display answer
mprintf("\n y(%.2f) = %.4f",x+h,f)
endfunction

// Start solving via iteration
for i = 1:n
y = rk2 (x0 ,y0);
// Called Function where formula is defined
// assign new value to originl variable for new iteration
x0 = x0 + h;
y0 = y;
//Rounding upto 5 decimal
y = round (y *10^5) /10^5;
end
```

OUTPUT :

Y(0.10) = 1.0588

Y(0.20) = 1.1513

PRACTICAL 7.C 2
R-K 4TH ORDER METHOD

```
clc
clear

// Define the given function
function [f]=fun1(x, y)
 f = y - x^2;
endfunction

// Define the given initial value
x0 = 0;
y0 = 1;
h = 0.25;
x = 0.5;
// No of iteration required
n = (x-x0)/h;
mprintf("The value of y by RK 4 th order Method")

// Calling Function - always defined first
// Define method to solve rk 2 order using formula or define
formula
function [f]=rk4(x, y)
k1 = h* fun1 (x,y);
k2 = h* fun1 (x + (h/2),y + (k1/2));
k3 = h* fun1 (x + (h/2),y + (k2/2));
k4 = h* fun1 (x + h,y + k3);
f = y + 1/6*(k1+2*k2+2*k3+k4);
// Display answer
mprintf("\n\n y(%.2f) = %.4f \n",x+h,f)
endfunction
```

```
// Start solving via iteration
for i = 1:n
y = rk4 (x0 ,y0);
x0 = x0 + h;
y0 = y;
//Rounding upto 4 decimal
y = round (y *10^4) /10^4;
end
```

OUTPUT :

THE VALUE OF Y BY RK 4 TH ORDER METHOD

Y(0.25) = 1.2785

Y(0.50) = 1.6013

PRACTICAL 8.A
LINEAR REGRESSION

```
clc
clear
//input the given value of x and y from table to matrix form
x = [20 30 35 40 45 50];
y = [10 11 11.8 12.4 13.5 14.4];
// Define variable for table
X = x .^2;
Y = y;
n =length(Y);
// write Required value in terms of matrix or eqation
// Define L.H.S
M1 = [sum(Y);sum(X .* Y) ];
//Define R.H.S.
M2 = [n sum(X);sum(X) sum(X .^2) ];
//Solve Matrix or Equation
A = M2 \ M1 ;
//Give variable to Required Value
a = A (1) ;
b = A (2) ;
// Display by taking roundup the number
disp("Equation of regression can be expressed as Y = a + bX")
disp(round(a *10^4) /10^4 , "a =" )
disp(round(b *10^4) /10^4 , "b =" )
```

OUTPUT :

Equation of regression can be expressed as Y = a + bX

A = 9.139

B = 0.0021

PRACTICAL 8.B
POLYNOMIAL REGRESSION

```
clc
clear
//input the given value of x and y from table to matrix form
X = 1:0.2:2;
Y = [0.98 1.4 1.86 2.55 2.28 3.2];

n =length(Y);
// write Required value in terms of matrix or eqation

// Define L.H.S
M1 = [sum(X .^2) sum(X .^3) sum(X .^4);  sum(X) sum(X.^2)
sum(X .^3); n sum(X) sum(X .^2) ];

//Define R.H.S.
M2 = [sum(X .^2 .* Y);sum(X .* Y);sum(Y) ];

//Solve Matrix or Equation
A = M1 \ M2 ;

//Solve Matrix or Equation
a = A (1) ;
b = A (2) ;
c = A (3) ;

// Display by taking roundup the number
disp("Equation of regression can be expressed as Y = a + bX +
cx^2")
disp(round(a *10^4) /10^4 , " a =" )
disp(round(b *10^4) /10^4 , "b =" )
disp(round(c *10^4) /10^4 , " c =" )
```

OUTPUT :

Equation of regression can be expressed as $Y = a + bx + cx^2$

A = - 1.4471

B = 2.6239

C = - 0.1875