PRACTICAL No: 1

❖ Program on Roots of Equation

a) Bisection Method.
b) Newton Raphson Method.
c) Successive Approximation Method.

PRACTICAL : 01

Input code: -

```
#Bisection Method
#Name :om take
#Roll No : 351041
#G.R. No : 22110800
#Ay 23-24 SEM-1
#Batch-A2
import math as m
def st_bsm(fun,x1,x2,acc,maxitr):
    while fun(x1)*fun(x2)>0:
        print("Intial guesses are wrong, enter new values ")
        x1 = float(input("Enter the value of x1 = "))
        x2 = float(input("Enter the value of x2 = "))
    for itr in range(maxitr):
        x0 = (x1+x2)/2
        if abs(x0-x1)<acc:
            break;
        if fun(x0)*fun(x1)<0:
            x1 = x1
            x2 = x0
        elif fun(x0)*fun(x1)>0:
            x1 = x0
            x2 = x2
        print("The roots of the equation : %.4f"%x0)
```

```
st_bsm(lambda x: m.cos(x)-1.3*x,1,2,0.0001,100)
```

Output:-

```
Intial guesses are wrong, enter new values
Enter the value of x1 = 0
Enter the value of x2 = 2
The roots of the equation : 1.0000
The roots of the equation : 0.5000
The roots of the equation : 0.7500
The roots of the equation : 0.6250
The roots of the equation : 0.5625
The roots of the equation : 0.5938
The roots of the equation : 0.6094
The roots of the equation : 0.6172
The roots of the equation : 0.6211
The roots of the equation : 0.6230
The roots of the equation : 0.6240
The roots of the equation : 0.6245
The roots of the equation : 0.6243
The roots of the equation : 0.6241
```

Newton's Raphosan Method

Input code: -

```
#Newton's Raphosan Method
#Name :om take
#Roll No : 351041
#G.R. No : 22110800
#Ay 23-24 SEM-1
#Batch-A2
import math as m
def st_nrm(fun,dfun,ddfun,x1,acc,maxitr):
    while abs(fun(x1)*ddfun(x1)/dfun(x1)**2)>1:
        print("initial guess is wrong, enter new value")
        x1 = float(input("enter value of x1 = "))
    for itr in range(maxitr):
        x0 = x1 - (fun(x1)/dfun(x1))
        if abs(x1-x0)<acc:
            break;
        else:
            x1 = x0
    print("the roots of the equation = %.4f"%x0)


st_nrm(lambda x: x**3-5*x+3,lambda x:3*x**2-5,lambda x:6*x,0,0.0001,10)
```

output:-

```
the roots of the equation = 0.6566
```

Sussecssive Approximation Method

Input code: -

```
#Sussecssive Approximation Method
#Name : om take
#Roll No : 351041
#G.R. No : 22110800
#Ay 23-24 SEM-1
#Batch-A2
import math as m
def st_sam(gun,dgun,x1,acc,maxitr):
    while abs(dgun(x1))>1:
        print("wrong initial guess, enter new value of x1")
        x1 = float(input("enter value of x1"))
    for itr in range(maxitr):
        x0 = gun(x1)
        if abs(x0-x1)<acc:
           break;
        else:
            x1 = x0
    print("Root of the equation = %.4f"%x0)
```

```
st_sam(lambda x: m.exp(x)*m.cos(x)-1.4+x, lambda x: -
m.exp(x)*m.sin(x)+m.exp(x)*m.cos(x)+1,1,0.0001,100)
```

Output:-

Root of the equation = 1.0698

PRACTICAL No: 2

❖ Program on Simultaneous Equation

a) Gauss Elimination Method.
b) Gauss-Siedal Method

PRACTICAL No : 02

Gauss Elimination Method

Input code: -

```
#Gauss Elimination Method
#Name :om take
#Roll No : 351041
#G.R. No : 22110800
#Ay 23-24 SEM-1
#Batch-A2
import numpy as np
def st_gem(a,d):
    a = a.astype(float)
    d = d.astype(float)
    n = len(d)
    for i in range(n):
        for k in range(i+1,n):
            fact  = a[k,i]/a[i,i]
            for j in range (n):
                a[k,j]-fact*a[i,j]
            d[k] = d[k]-fact*d[i]
    x = np .zeros(n)
    for i in range (n-1,-1,-1):
        temp = 0
        for j in range (i+1,n):
            temp = temp + a[i,j]*x[j]
        x[i] = (d[i]-temp)/ a[i,i]
        print ("x(%i)= %.4f"%(i,x[i]))


st_gem(np.array ([[4,1,2,3],[3,4,1,2],[2,3,4,1],[1,2,3,4]]),np.array
([40,40,40,40]))
```

Output:-

```
x(3)= 3.9062
x(2)= 2.1484
x(1)= 0.0098
x(0)= 5.9937
```

Gauss-Seidal Method

Input code: -

```
#Gauss-Seidal Method
#Name : om take
#Roll No : 351041
#G.R. No : 22110800
#Ay 23-24 SEM-1
#Batch-A2
import numpy as np
def st_gsm(a,d,maxitr):
     a=a.astype(float)
     d=d.astype(float)
     n=len(d)
     x= np.zeros(n)
     for itr in range(maxitr):
         for i in range(n):
             temp=0
             for j in range(n):
                 if j!=i:
                     temp=temp+a[i,j]*x[j]
                 x[i]=(d[i]-temp)/a[i,i]
         for i in range(n):
             print("x(%i): %.4f"%(i,x[i]))
         print(" \n")
```

```
st_gsm(np.array([[4,1,2,3],[3,4,1,2],[2,3,4,1],[1,2,3,4]]),np.array([40
,40,40,40]),2)
```

Output:-

```
x(0): 10.0000
x(1): 2.5000
x(2): 3.1250
x(3): 3.9062


x(0): 4.8828
x(1): 3.6035
x(2): 3.8794
x(3): 4.0680
```

PRACTICAL No: 3

❖ Program on Numerical Integration

a) Trapezoidal Rule.
b) Simpson's 1/3rd Rule.
c) Simpson's 3/8th Rule.

PRACTICAL No : 03

Trapezoidal Rule

Input code: -

```
#Trapezoidal Rule
#Name : om take
#Roll No : 351041
#G.R. No : 22110800
#Ay 23-24 SEM-1
#Batch-A2
import math as m
def st_tr(fun,x0,xn,n):
	h=(xn-x0)/n
	y0=fun(x0)
	yn=fun(xn)
	yr=0
	for i in range (1,n,1):
	    yr = yr + fun(x0 + i * h)
	I= 0.5 * h * (y0 + yn + 2 * yr)
	print("Integration : %0.4f"%I)
```

```
st_tr(lambda x: 1/(1+x*x),0,6,6)
```

Output:-

```
Integration : 1.4108
```

Simpson's Rules (1/3rd)

Input code: -

```
#Simpson's Rules (1/3rd)
#Name : om take
#Roll No : 351041
#G.R. No : 22110800
#Ay 23-24 SEM-1
#Batch-A2
import math as m
def st_S13(fun,x0,xn,n):
     h=(xn-x0)/n
     y0=fun(x0)
     yn=fun(xn)
     yodd=0
     yeven=0
     for i in range (1,n,2):
         yodd = yodd + fun(x0 + i * h)
     for j in range (2,n,2):
         yeven = yeven + fun(x0 + j * h)
     I= (1/3) * h * (y0 + yn + 4 * yodd + 2 * yeven)
     print("Integration : %0.4f"%I)
```

```
st_S13(lambda x: 1/(1+x*x),0,6,6)
```

Output:-

```
Integration : 1.3662
```

Simpson's Rules (3/8th)

Input code: -

```
#Simpson's Rules (3/8th )
#Name :om take
#Roll No : 351041
#G.R. No : 22110800
#Ay 23-24 SEM-1
#Batch-A2
import math as m
def st_S38(fun,x0,xn,n):
      h=(xn-x0)/n
      y0=fun(x0)
      yn=fun(xn)
      yr=0
      ym3=0
      for i in range (1,n,1):
          yr = yr + fun(x0 + i * h)
      for j in range (3,n,3):
          ym3 = ym3 + fun(x0 + j * h)
      I= (3/8) * h * (y0 + yn + 3 * (yr - ym3) + 2 * ym3)
      print("Integration : %0.4f"%I)
```

```
st_S13(lambda x: 1/(1+x*x),0,6,6)
```

Output:-

```
Integration : 1.3662
```

PRACTICAL No: 4

❖ Program on Curve Fitting

a) Straight line.
b) Power or Exponential Equation.
c) Quadratic Equation.

PRACTICAL No : 04

Curve fitting-straight line

Input code: -

```
#Curve fitting-straight line
#Name : om take
#Roll No : 351041
#G.R. No : 22110800
#Ay 23-24 SEM-1
#Batch-A2
import numpy as np
def st_slcf(x,y):
    x = x.astype (float)
    y = y.astype(float)
    A = np.array([ [len(x) , sum(x)],
                   [sum(x) , sum(x*x)]])
    B = np.array([ [sum(y)],
                   [sum(x*y)]])
    C = np.linalg .solve(A,B)
    print("y =%.4f + %.4f *x"%(C[0],C[1]))
```

```
st_slcf(np.array([6,7,7,8,8,8,9,9,10]),np.array([5,5,4,5,4,3,4,3,3]))
```

Output:-

```
y =8.0000 + -0.5000 *x
```

Curve fitting-Power Equation

Input code: -

```
#Curve fitting-Power Equation
#Name :om take
#Roll No : 351041
#G.R. No : 22110800
#Ay 23-24 SEM-1
#Batch-A2
import numpy as np
import math as m
def st_pe(x,y):
 x=x.astype(float)
 y=y.astype(float)
 n=len(x)
 X=np.log(x)
 Y=np.log(y)
 A=np.array([[n,sum(X)],[sum(X),sum(X*X)]])
 C=np.array([[sum(Y)],[sum(X*Y)]])
 B=np.linalg.solve(A,C)
 alpha= m.exp(B[0])
 beta= B[1]
 print("y=%.4f*x^%.4f"%(alpha,beta))
```

```
st_pe(np.array([61,26,7,2.6]),np.array([350,400,50,600]))
```

Output:-

```
y=210.4654*x^0.0741
```

Curve fitting: Quadratic Equation

Input code: -

```
#Curve fitting: Quadratic Equation
#Name : om take
#Roll No : 351041
#G.R. No : 22110800
#Ay 23-24 SEM-1
#Batch-A2
import numpy as np
def st_qcf(x,y):
    x = x.astype (float)
    y = y.astype(float)
    A = np.array ([ [len(x),sum(x),sum(x*x)],
                   [sum(x),sum(x*x),sum(x*x*x)],
                   [sum(x*x),sum(x*x*x),sum(x*x*x*x)]])
    B = np.array([[sum(y)],
                 [sum(x*y)],
                 [sum(x*x*y)]])
    C = np.linalg.solve(A,B)
    print("y = %.4f + % .4f * x + %.4f x*x "%(C[0],C[1],C[2]))
```

```
st_qcf(np.array([6,7,7,8,8,8,9,9,10]),np.array([5,5,4,5,4,3,4,3,3]))
```

output:-

```
y = 8.0000 + -0.5000 * x + 0.0000 x*x
```

PRACTICALNo: 5

❖ Program on Interpolation

a) Lagrange's Interpolation.
b) Newton's Forward Interpolation

PRACTICAL No : 05

Lagrange’s Interpolation

Input code: -

```
#Lagrange’s Interpolation
#Name :om take
#Roll No : 351041
#G.R. No : 22110800
#Ay 23-24 SEM-1
#Batch-A2
import numpy as np
def st_li(x,y,xr):
  x = x.astype(float)
  y = y.astype(float)
  n = len(x)
  yr = 0
  for i in range(n):
    L = 1
    for j in range(n):
      if i != j:
        L = L *((xr - x[j])/(x[i]-x[j]))
      yr = yr + y[i] * L
  print("y at x = %.4f is equal to %.4f"%(xr,yr))
```

```
st_li(np.array([100,150,200,250,300,350,400]),np.array([10.63,13.03,15.
04,16.81,18.42,19.90,21.27]),160)
```

Output:-

```
y at x = 160.0000 is equal to 140.1077
```

Newton's Forward Interpolation

Input code: -

```
#Newton's Forward Interpolation
#Name : om take
#Roll No : 351041
#G.R. No : 22110800
#Ay 23-24 SEM-1
#Batch-A2
import numpy as np
import math as m
def st_nfdi(x,y,xr):
  x = x.astype(float)
  y = y.astype(float)
  n = len(x)
  delta = np.zeros((n-1,n-1))

  for j in range(n-1):
    for i in range((n-1)-j):
      if j == 0:
        delta[i,j] = y[i+1] - y[i]
      else:
        delta[i,j] = delta[i+1,j-1] - delta[i,j-1]

  h = x[1] - x[0]
  u = (xr - x[0])/h
  term = 0
  mult = 1

  for j in range(n-1):
    mult = mult * (u-j)
    term = term + delta[0,j] / m.factorial(j+1) * mult
  yr = y[0]+term
  print("y at x = %.4f is equal to %.4f" %(xr,yr))
```

```
st_nfdi(np.array([100,150,200,250,300,350,400]),np.array([10.63,13.03,15.04,16.81,18.42,19.90,21.27]),160)
```

Output:-

```
y at x = 160.0000 is equal to 13.4573
```

PRACTICAL No: 6

❖ Program on Ordinary Differential

a) Euler’s Method.
b) Runge-Kutta Method second order.

PRACTICAL No : 06

Euler Method

Input code: -

```
#Eular method
#Name :om take
#Roll No : 351041
#G.R. No : 22110800
#Ay 23-24 SEM-1
#Batch-A2
import math as m
def st_em(fun,x0,y0,xn,n):
  h=(xn-x0)/n
  for i in range(1,n+1):
    ynew=y0+h*fun(x0,y0)
    x0=x0+h
    y0=ynew
    print("x=%.4f"%x0,"y=%.4f"%y0)



st_em(lambda x,y:m.sqrt(x+m.sqrt(y)),2,4,2.5,10)
```

output:-

```
x=2.0500 y=4.1000
x=2.1000 y=4.2009
x=2.1500 y=4.3028
x=2.2000 y=4.4055
x=2.2500 y=4.5092
x=2.3000 y=4.6138
x=2.3500 y=4.7192
x=2.4000 y=4.8256
x=2.4500 y=4.9328
x=2.5000 y=5.0408
```

Runge Kutta 2nd Order Method

Input code: -

```
#Name : om take
#Roll No : 351041
#G.R. No : 22110800
#Ay 23-24 SEM-1
#Batch-A2
```

```
name="Pra_rk2"
import math as m
def pra_rk2(fun,x0,y0,xn,n):
  h=(xn - x0)/n
  for i in range (1,n+1):
    k1 = h* fun(x0,y0)
    k2 = h * fun(x0+h/2,y0+k1/2)
    ynew = y0 +(k1+k2)/2
    print("x= %.4f; y=%.4f ; ynew =%.4f"%(x0,y0,ynew))
    x0=x0+h
    y0=ynew
  print("xn=%.4f ; yn=%.4f"%(xn, ynew))
```

```
pra_rk2(lambda x,y:2-x*y,5,2,5.1,10)
```

```
x= 5.0000; y=2.0000 ; ynew =1.9210
x= 5.0100; y=1.9210 ; ynew =1.8456
x= 5.0200; y=1.8456 ; ynew =1.7738
x= 5.0300; y=1.7738 ; ynew =1.7054
x= 5.0400; y=1.7054 ; ynew =1.6403
x= 5.0500; y=1.6403 ; ynew =1.5782
x= 5.0600; y=1.5782 ; ynew =1.5191
x= 5.0700; y=1.5191 ; ynew =1.4627
x= 5.0800; y=1.4627 ; ynew =1.4091
x= 5.0900; y=1.4091 ; ynew =1.3580
xn=5.1000 ; yn=1.3580
```

Runge Kutta 4nd Order Method

Input code: -

```
#Name:om take
#Roll No : 351041
#G.R. No : 22110800
#Ay 23-24 SEM-1
#Batch-A2
```

```python
import math as m
def pra_rk4(fun,x0,y0,xn,n):
  h=(xn - x0)/n
  for i in range (1,n+1):
    k1 = h* fun(x0,y0)
    k2 = h * fun(x0+h/2,y0+k1/2)
    k3 = h* fun(x0+h/2,y0+k2/2)
    k4 = h* fun(x0+h,y0+k3)
    ynew = y0 + 1/6 * (k1+2*k2+2*k3+k4)
    print("x= %.4f; y=%.4f ; ynew =%.4f"%(x0,y0,ynew))
    x0=x0+h
    y0=ynew
  print("xn=%.4f ; yn=%.4f"%(xn, ynew))
```

```python
pra_rk4(lambda x,y: x+y ,0,1,1,5)
```

```
x= 0.0000; y=1.0000 ; ynew =1.2428
x= 0.2000; y=1.2428 ; ynew =1.5836
x= 0.4000; y=1.5836 ; ynew =2.0442
x= 0.6000; y=2.0442 ; ynew =2.6510
x= 0.8000; y=2.6510 ; ynew =3.4365
xn=1.0000 ; yn=3.4365
```