ಬೆಂಗಳೂರು
ನಗರ ವಿಶ್ವವಿದ್ಯಾಲಯ

BENGALURU
CITY UNIVERSITY

ಆಗು ನೀ ಅನಿಕೇತನ
BE BOUNDLESS

B.Sc., VI SEMESTER

(NEP-2020 Batch)

Python Commands for

MATHEMATICS

PAPER 6.2

LAB MANUAL

Topic:

Algebraic and Transcendental equations, System of linear algebraic equations, Polynomial interpolation, Numerical Differentiation and integration.

Contents

Sl.No.	Title
	PART A: Algebraic and Transcendental Equations
Lab 1	Program to find root of an equation using Bisection and Regula-Falsi methods.
Lab 2	Program to find root of an equation using Newton-Raphson.
	Part B: System of Linear Algebraic Equations
Lab 3	Program to solve system of algebraic equations using Gauss-elimination method.
Lab 4	Program to solve system of algebraic equations using Gauss-Jordan method.
Lab 5	Program to solve system of algebraic equations using Gauss-Jacobi method.
Lab 6	Program to solve system of algebraic equations using Gauss-Seidel method.
Lab 7	Program to solve system of algebraic equations using Successive Over Relaxation (SOR) method.
	Part C: Polynomial Interpolations
Lab 8	Program to find the sums of powers of successive natural numbers using Newton-Gregory technique.
	Part D: Numerical Differentiation and Integration
Lab 9	Program to find differentiation at specified point using Newton-Gregory interpolation method.
Lab 10	Program to find the missing value of table using Lagrange method.
Lab 11	Program to evaluate integral using Simpson's $1/3^{rd}$ and $3/8^{th}$ rules.
Lab 12	Program to evaluate integral using Trapezoidal and Weddle rules.

PART A: Algebraic and Transcendental Equations

Lab 1: Program to find root of an equation using Bisection and Regula-Falsi methods.

"Bisection Methods"

Example 1: Find a root of the equation $x^3-4x-9=0$ in (2,3) using bisection method, correct to 3 decimal places.

Python Command:

```
from sympy import *
def f(x):
   return x**3-4*x-9;
a=2
b=3
step=1
while(step<=11):
   x=(a+b)/2
   print("Iterations=%d,   x=%0.3f,   f(x)=%0.3f"%(step,x,f(x)))
   if f(a)*f(x)<0:
      b=x
   else:
      a=x
   step=step+1
print('\n The required approximation root is = %0.3f ' %x)
```

Output:

```
Iterations=1,   x=2.500,   f(x)= -3.375
Iterations=2,   x=2.750,   f(x)= 0.797
Iterations=3,   x=2.625,   f(x)= -1.412
Iterations=4,   x=2.688,   f(x)= -0.339
Iterations=5,   x=2.719,   f(x)= 0.221
Iterations=6,   x=2.703,   f(x)= -0.061
Iterations=7,   x=2.711,   f(x)= 0.079
Iterations=8,   x=2.707,   f(x)= 0.009
Iterations=9,   x=2.705,   f(x)= -0.026
Iterations=10,   x=2.706,   f(x)= -0.009
Iterations=11,   x=2.707,   f(x)= 0.000

The required approximation root is = 2.707
```

Example 2: Use bisection method in six stages to obtain a root of the equation $f(x)=\cos(x)-xe^{x}=0$ in (0,1), correct to 4 decimal places.

Example 3: Use bisection method in four stages to find the real root of the equation $x\log_{10}(x)-102=0$ in (57,58), correct to 3 decimal places.

"Regula-Falsi methods"

Example 1: Find the real root of the equation cos(x) = 3x-1 using Regula-Falsi Method.

Python Command:

```
from sympy import *
def f(x):
    return (cos(x)+1-3*x)
a=float(input('Enter the value of a='))
b=float(input('Enter the value of b='))
if f(a)*f(b)>0.0:
    print('The root of f(x) does not lies between the a and b')
else:
    print('The root of f(x) lies between the a and b')
    print('\n\n ******REGULA-FALSI METHOD******')
    i=int(input('Enter no. of iterations='))
    step=1
    while (step<=i):
        x=(a*f(b)-b*f(a))/(f(b)-f(a))
        print('Iteration:%d, x=%0.6f  and f(x)=%0.6f' %(step,x,f(x)))
        if f(a)*f(x)<0:
            b=x
        else:
            a=x
        step=step+1
    print('\n Required root is = %0.6f'%x)
```

Output:

Enter the value of a=0
Enter the value of b=1
The root of f(x) lies between the a and b

********REGULA-FALSI METHOD********
Enter no. of iterations=5
Iteration:1, x=0.578085 and f(x)=0.103255
Iteration:2, x=0.605959 and f(x)=0.004081
Iteration:3, x=0.607057 and f(x)=0.000159
Iteration:4, x=0.607100 and f(x)=0.000006
Iteration:5, x=0.607102 and f(x)=0.000000

Required root is = 0.607102

Example 2. Find a real root of the equation $x^3 - 4x + 1 = 0$ by Regula falsi method.

Example 3. Compute the real root of $xlog_{10}x - 1.2 = 0$ by the method of false position.

Example 4. Find the real root of the equation $xe^x - 3 = 0$ by Regula-Falsi method correct to four decimal places.

Lab 2: Program to find root of an equation using Newton-Raphson method.

Example 1: Using Newton-Raphson method, find the real root of the equation 3x=cos(x)+1, near 0.5.

Python Command:

```
from sympy import *
def f(x):
   return 3*x-cos(x)-1
def g(x):
   return 3+sin(x)
x0=0.5
step=1
while(step<=3):
   x=x0-(f(x0)/g(x0))
   print("Iterations=%d,  x=%0.3f,  f(x)=%0.3f"%(step,x,f(x)))
   x0=x
   step=step+1
print( '\n The required approximation root is=%0.3f ' %x)
```

Output:

```
Iterations=1,  x=0.609,  f(x)=0.005
Iterations=2,  x=0.607,  f(x)=0.000
Iterations=3,  x=0.607,  f(x)=0.000
The required approximation root is=0.607
```

Example 2: Using Newton-Raphson method to find a real root of the equation x^3-2x-5=0, near 2.

Example 3: Using Newton-Raphson method, find a real root of the equation sin(x)=1- x (near x = 0.5).

Example 4. Using Newton-Raphson method to find the fourth root of 32.

Part B: System of Linear Algebraic Equations

Lab 3: Program to solve system of algebraic equations using Gauss-elimination method.

Example 1. Solve the system of equations

$x + y + z = 9;\ 2x - 3y + 4z = 13;\ 3x + 4y + 5z = 40$

using the Gauss-elimination method.

Python Command:

```
import numpy as np
import sys
n = int(input('Enter number of unknowns: '))
a = np.zeros((n,n+1))
x = np.zeros(n)
# Reading augmented matrix coefficients
print('Enter Augmented Matrix Coefficients:')
for i in range(n):
   for j in range(n+1):
      a[i][j]=float(input('a['+str(i)+']['+str(j)+']='))
# Applying Gauss Elimination
for i in range(n):
   if a[i][i] == 0.0:
      sys.exit('Divide by zero detected!')
   for j in range(i+1, n):
      ratio = a[j][i]/a[i][i]
      for k in range(n+1):
         a[j][k] = a[j][k] - ratio * a[i][k]
# Back Substitution
x[n-1] = a[n-1][n]/a[n-1][n-1]
for i in range(n-2,-1,-1):
   x[i] = a[i][n]
   for j in range(i+1,n):
      x[i] = x[i] - a[i][j]*x[j]
   x[i] = x[i]/a[i][i]
# Displaying solution
print('\nRequired solution is: ')
for i in range(n):
   print('X%d = %0.2f' %(i,x[i]), end = '\t')
```

Output:

Enter number of unknowns: 3
Enter Augmented Matrix Coefficients:
a[0][0]=1
a[0][1]=1
a[0][2]=1
a[0][3]=9
a[1][0]=2
a[1][1]=-3
a[1][2]=4
a[1][3]=13
a[2][0]=3
a[2][1]=4
a[2][2]=5
a[2][3]=40

Required solution is:
X0 = 1.00 X1 = 3.00 X2 = 5.00

Example 2. Solve the system of equations

$x + 10y - z = 3;\ 2x + 3y + 20z = 7;\ 10x - y + 2z = 4$

using the Gauss-elimination method.

Example 3. Solve the system of equations

$3x + 3y + 4z = 20;\ 2x + y + 3z = 13;\ x + y + 3z = 6$

using the Gauss-elimination method.

Lab 4: Program to solve system of algebraic equations using Gauss-Jordan method.

Example 1. Solve the system of equations

$$x + y - z = 7;\ x - y + 2z = 3;\ 2x + y + z = 9$$

Using the Gauss-Jordan method.

Python Command:

```
import numpy as np
import sys
n = int(input('Enter number of unknowns: '))
a = np.zeros((n,n+1))
x = np.zeros(n)
for i in range(n):
   for j in range(n+1):
      a[i][j] = float(input( 'a['+str(i)+']['+ str(j)+']='))
# Applying Gauss Jordan Elimination
for i in range(n):
   if a[i][i] == 0.0:
      sys.exit('Divide by zero detected!')
   for j in range(n):
      if i != j:
         ratio = a[j][i]/a[i][i]
         for k in range(n+1):
            a[j][k] = a[j][k]-ratio*a[i][k]
# Obtaining Solution
for i in range(n):
   x[i] = a[i][n]/a[i][i]
# Displaying solution
print('\n Required solution is: ')
for i in range(n):
       print('X%d = %0.2f' %(i,x[i]), end = '\t')
```

Output:

```
Enter number of unknowns: 3
Enter Augmented Matrix Coefficients:
a[0][0]=1
a[0][1]=1
a[0][2]=-1
a[0][3]=7
a[1][0]=1
a[1][1]=-1
a[1][2]=2
a[1][3]=3
a[2][0]=2
a[2][1]=1
a[2][2]=1
a[2][3]=9

Required solution is:
X0 = 6.00    X1 = -1.00   X2 = -2.00
```

Example 2. Solve the system of equations

$x + y + z = 1;\ 4x + 3y - z = 6;\ 3x + 5y + 3z = 4$

using the Gauss-Jordan method.

Example 3. Solve the system of equations

$2x + y + z = 10;\ 3x + 2y + 3z = 18;\ x + 4y + 9z = 16$

using the Gauss-Jordan method.

Lab 5: Program to solve system of algebraic equations using Gauss-Jacobi method.

Example 1: Solve 10x+2y+z=9, x+10y-z=-22, -2x+3y+10z=22.

Python Command:

```
from sympy import *
f1=lambda x,y,z:(9-2*y-z)/10
f2=lambda x,y,z:(-22-x+z)/10
f3=lambda x,y,z:(22+2*x-3*y)/10
#Initial conditions
x0=0;y0=0;z0=0
itr=1 #iterations
e=float(input('Enter the tolerable error:'))
condition=True
print('\n itr \tx \ty \tz \n')
while condition:
   x1=f1(x0,y0,z0)
   y1=f2(x0,y0,z0)
   z1=f3(x0,y0,z0)
   print('%d\t%0.4f\t%0.4f\t%0.4f\n'%(itr,x1,y1,z1))
   e1=abs(x0-x1);e2=abs(y0-y1);e3=abs(z0-z1)
   itr +=1
   x0=x1
   y0=y1
   z0=z1
   condition =e1>e and e2>e and e3>e
print('\n Solution: x=%0.4f, y=%0.4f, z=%0.4f\n'%(x1,y1,z1))
```

Output:

```
Enter the tolerable error:0.0001
 itr    x       y       z
1       0.9000      -2.2000     2.2000
2       1.1200      -2.0700     3.0400
3       1.0100      -2.0080     3.0450
4       0.9971      -1.9965     3.0044
5       0.9989      -1.9993     2.9984
6       1.0000      -2.0000     2.9996
7       1.0001      -2.0000     3.0000

 Solution: x=1.0001, y=-2.0000, z=3.0000
```

Solve the following system of equations by Gauss-Jacobi method.

Example 2. $5x + 2y + z = 12;$
$x + 4y + 2z = 15;$
$x + 2y + 5z = 0$

Example 3. $10x - 2y + z = 12;$
$x + 9y - z = 10;$
$2x - y + 11z = 20$

Example 4. $10x + y + z = 12;$
$2x + 10y + z = 13;$
$2x + 2y + 10z = 14$

Lab 6: Program to solve system of algebraic equations using Gauss-Seidel method.

Example 1: Solve 10x+2y+z=9, x+10y-z=22, 2x+3y+10z=22.

Python Command:

```
from sympy import *
var('x,y,z')
f1=lambda x,y,z:(9-2*y-z)/10
f2=lambda x,y,z:(-22-x+z)/10
f3=lambda x,y,z:(22+2*x-3*y)/10
x0=0;y0=0;z0=0
step=1
while (step<=5):
   x1=f1(x0,y0,z0)
   y1=f2(x1,y0,z0)
   z1=f3(x1,y1,z0)
   print("Iterations=%d,x=%0.3f,y=%0.3f,z=%0.3f "%(step,x1,y1,z1))
   x0=x1;y0=y1;z0=z1
   step=step+1
print('\n Solution: x=%0.3f, y=%0.3f, z=%0.3f \n ' %(x1,y1,z1))
```

Output:

```
Iterations=1,  x=0.900,  y=-2.290,  z=3.067
Iterations=2,  x=1.051,  y=-1.998,  z=3.010
Iterations=3,  x=0.999,  y=-1.999,  z=2.999
Iterations=4,  x=1.000,  y=-2.000,  z=3.000
Iterations=5,  x=1.000,  y=-2.000,  z=3.000

Solution: x=1.000, y=-2.000, z=3.000
```

Example 2: Solve

5x-2y+z = -4, x+6y-2z = -1, 3x+y+5z = 13.

Example 3: Solve

9x+2y+4z = 20, x+10y+4z = 6, 2x - 4y + 10z = -15.

Example 4: Solve

x+y+54z = 110, 27x+6y-z = 85, 6x + 15y + 2z = 72.

Lab 7: Program to solve system of algebraic equations using Successive Over Relaxation (SOR) method.

Example 1: Solve the system of equations

$3x - y + z = -1;\ -x + 3y - z = 7;\ x - y + 3z = -7$

using Successive Over Relaxation method.

Python Command:

```
from sympy import *
f1 = lambda x,y,z: (-1+y-z)/3
f2 = lambda x,y,z: (7+x+z)/3
f3 = lambda x,y,z: (-7-x+y)/3
# Initial setup
x0 = 0; y0 = 0; z0 =0
count = 1
# Reading tolerable error
e = float(input('Enter tolerable error: '))
# Reading relaxation factor
w = float(input("Enter relaxation factor: "))
# Implementation of successive over-relaxation
print('\nCount\tx\ty\tz\n')
condition = True
while condition:
   x1 = (1-w) * x0 + w * f1(x0,y0,z0)
   y1 = (1-w) * y0 + w * f2(x1,y0,z0)
   z1 = (1-w) * z0 + w * f3(x1,y1,z0)
   print('%d\t%0.4f\t%0.4f\t%0.4f\n' %(count, x1,y1,z1))
   e1 = abs(x0-x1);
   e2 = abs(y0-y1);
   e3 = abs(z0-z1);
   count += 1
   x0 = x1
   y0 = y1
   z0 = z1
   condition = e1>e and e2>e and e3>e
print('\nSolution: x = %0.3f, y = %0.3f and z = %0.3f\n'% (x1,y1,z1))
```

Output:

Enter tolerable error: 0.0001

Enter relaxation factor: 1.25

Count	x	y	z
1	-0.4167	2.7431	-1.6001
2	1.4972	2.1880	-2.2288
3	1.0494	1.8782	-2.0141
4	0.9428	2.0007	-1.9723
5	1.0031	2.0126	-2.0029
6	1.0057	1.9980	-2.0025
7	0.9988	1.9990	-1.9993
8	0.9996	2.0004	-1.9998
9	1.0002	2.0001	-2.0001
10	1.0000	2.0000	-2.0000

Solution: x = 1.000, y = 2.000 and z = -2.000

Example 2: Solve the system of equations

10x +2y - z = 7; x + 8y + 3z = -4; -2x - y + 10z = 9

using Successive Over Relaxation method.

Part C: Polynomial Interpolations

Lab 8: Program to find the sums of powers of successive natural numbers using Newton-Gregory technique.

Newton's Forward Interpolation Formula

1) If y(75)=246, y(80)=202, y(85)=118, y(90)=40. Find y(79).

Python Command:

```
from sympy import *
x=[75,80,85,90]
y=[246,202,118,40]
print("X values:",x)
print("Y values:",y)
X=79;h=5;sum1=y[0];term=1;k=1;n=len(x)
p=(X-x[0])/h
print("\n Difference table")
print(*x,sep="\t")
print("-"*40)
print(*y,sep="\t")
for i in range(0,n-1):
    z=[]
    for j in range(0,n-i-1):
        y[j]=y[j+1]-y[j]
        z.append(y[j])
    print(*z,sep="\t")
    term=term*(p-i)/k
    k=k+1
    sum1=sum1+term*y[0]
print("\n y(79)=",round(sum1,3))
```

Output:

```
X values: [75, 80, 85, 90]
Y values: [246, 202, 118, 40]
Difference table
75 80 85 90
--------------------------------
246 202 118 40
-44  -84  -78
-40    6
 46
y(79)= 215.472
```

2) Evaluate $y=e^{2x}$ for x=0.05 from the following table:

x	0	0.10	0.20	0.30	0.40
y	1.00	1.22	1.49	1.82	2.26

3) Estimate y at x=4.2 from the following table:

x	4	6	8	10
y	1	3	8	16

Newton's Backward Interpolation Formula

Example 1. Estimate f(7.5) from the following table:

X	1	2	3	4	5	6	7	8
Y	1	8	27	64	125	216	343	512

Python Command:

```
from sympy import *
x=[1,2,3,4,5,6,7,8]
y=[1,8,27,64,125,216,343,512]
print("X values:",x)
print("Y values:",y)
X=7.5; h=1; n=len(x); sum1=y[n-1]; term=1; k=1
p=(X-x[n-1])/h
print("\n Difference table")
print(*x,sep="\t")
print("-"*60)
print(*y,sep="\t")
for i in range(0,n-1):
    z=[]
    for j in range(0,n-i-1):
        y[j]=y[j+1]-y[j]
        z.append(y[j])
    print(*z,sep="\t")
    term=term*(p+i)/k
    k=k+1
    sum1=sum1+term*y[n-i-2]
print("\n y(7.5)=",round(sum1,3))
```

Output:

X values: [1, 2, 3, 4, 5, 6, 7, 8]
Y values: [1, 8, 27, 64, 125, 216, 343, 512]
Difference table
1 2 3 4 5 6 7 8
----
1 8 27 64 125 216 343 512
7 19 37 61 91 127 169
12 18 24 30 36 42
6 6 6 6 6
0 0 0 0
0 0 0
0 0
0
y(7.5)= 421.8752)

Example 2. Estimate f(1.28) from the following table:

x	1.15	1.20	1.25	1.30
f(x)	1.0723	1.0954	1.1180	1.1401

Example 3. The population of the town is as follows,

Year	1921	1931	1941	1951	1961	1971
Population in Lakhs	20	24	29	36	46	51

Estimate the population in the year 1955.

Part D: Numerical Differentiation and Integration

Lab 9: Program to find differentiation at specified point using Newton-Gregory interpolation method.

Newton's Forward Interpolation Formula

Example 1. Find y′ and y′′ at x = 1 from the following data

X	1.00	1.05	1.10	1.15	1.20	1.25	1.30
Y	1.00	1.0247	1.0488	1.0723	1.0954	1.1180	1.1401

Python command:

```
from sympy import *
from numpy import *
x=[1.00,1.05,1.10,1.15,1.20,1.25,1.30]
y=[1.00,1.0247,1.0488,1.0723,1.0954,1.1180,1.1401]
X=1;h=0.05;n=len(x)
d=zeros((n,n+1))
for i in range(0,n):
    d[i][0]=x[i]
    d[i][1]=y[i]
for i in range(1,n):
    for j in range(0,n-i):
        d[j][i+1]=d[j+1][i]-d[j][i]
print("\n Forward Difference Table \n")
for i in range(n):
    print(x[i],end="\t")
    for j in range(n-i):
        print(d[i][j+1].round(4),end="\t")
    print("")
sum1=d[0][2]-(1/2)*d[0][3]+(1/3)*d[0][4]-(1/4)*d[0][5]+(1/5)*
        d[0][6]-(1/6)*d[0][7]
sum2=d[0][3]-d[0][4]+(11/12)*d[0][5]-(5/6)*d[0][6]+(137/180)*
        d[0][7]
print("\n y'(1)=",round((sum1/h),4))
print("\n y''(1)=",round((sum2/(h**2)),4))
```

Output:

```
Forward Difference Table
1.0     1.0         0.0247    -0.0006    0.0        0.0002      -0.0005      0.0009
1.05    1.0247      0.0241    -0.0006  0.0002     -0.0003    0.0004
1.1     1.0488      0.0235     -0.0004     -0.0001   0.0001
1.15    1.0723      0.0231     -0.0005     -0.0
1.2     1.0954      0.0226     -0.0005
1.25    1.118       0.0221
1.3     1.1401
 y'(1)= 0.494    y''(1)= 0.274
```

Example 2. Find $\frac{dy}{dx}$ and $\frac{d^2y}{dx^2}$ at x = 1 from the following data

X	1	2	3	4	5	6
Y	1	8	27	64	125	216

Example 3. Find $\frac{dy}{dx}$ and $\frac{d^2y}{dx^2}$ at x = 51 from the following data

X	50	60	70	80	90
Y	19.96	36.65	58.81	77.21	94.61

Newton's Backward Interpolation Formula

Example 1. Find y′ and y′′ at x = 1.6 from the following data

X	1.0	1.1	1.2	1.3	1.4	1.5	1.6
Y	7.989	8.403	8.781	9.129	9.451	9.750	10.031

Python command:

```
from sympy import *
from numpy import *
x=[1.0,1.1,1.2,1.3,1.4,1.5,1.6]
y=[7.989,8.403,8.781,9.129,9.451,9.750,10.031]
X=1.6;h=0.1;n=len(x)
d=zeros((n,n+1))
for i in range(0,n):
    d[i][0]=x[i]
    d[i][1]=y[i]
for i in range(2,n):
    for j in range(n-1,i-2,-1):
        d[j][i]=d[j][i-1]-d[j-1][i-1]
print("Backward Difference Table")
for i in range(n):
    print(x[i],end="\t")
    for j in range(i+1):
        print(d[i][j+1].round(3),end="\t")
    print("")
sum1=d[6][2]+(1/2)*d[6][3]+(1/3)*d[6][4]+(1/4)*d[6][5]+(1/5)
        *d[6][6]+(1/6)*d[6][7]
sum2=d[6][3]+d[6][4]+(11/12)*d[6][5]+(5/6)*d[6][6]+(137/180)
        *d[6][7]
print("\n y'(1.6)=",round((sum1/h),4))
print("\n y''(1.6)=",round((sum2/(h**2)),4))
```

Output:

Backward Difference Table

1.0	7.989						
1.1	8.403	0.414					
1.2	8.781	0.378	-0.036				
1.3	9.129	0.348	-0.03	0.006			
1.4	9.451	0.322	-0.026	0.004	-0.002		
1.5	9.75	0.299	-0.023	0.003	-0.001	0.001	
1.6	10.031	0.281	-0.018	0.005	0.002	0.003	0.0

y'(1.6)= 2.7477

y''(1.6)= -0.8667

Example 2. Find $\frac{dy}{dx}$ and $\frac{d^2y}{dx^2}$ at x = 54 from the following table.

X	50	51	52	53	54
Y	3.6840	3.7084	3.7325	3.7563	3.7798

Lab 10: Program to find the missing value of table using Lagrange method.

Example 1. Find f(9) from the following data

x	5	7	11	13	17
y	150	392	1452	2366	5202

Using Lagrange's interpolation formula.

Python Command :

```
import numpy as np
n = int(input('Enter number of data points: '))
x = np.zeros((n))
y = np.zeros((n))
# Reading data points
print('Enter data for x and y: ')
for i in range(n):
   x[i] = float(input( 'x['+str(i)+']='))
   y[i] = float(input( 'y['+str(i)+']='))
# Reading interpolation point
xp = float(input('Enter interpolation point: '))
# Set interpolated value initially to zero
yp = 0
# Implementing Lagrange Interpolation
for i in range(n):
   p = 1
   for j in range(n):
      if i != j:
         p = p * (xp - x[j])/(x[i] - x[j])
   yp = yp + p * y[i]
# Displaying output
print('Interpolated value at %.3f is %.3f.' % (xp, yp))
```

Output :

Enter number of data points: 5
Enter data for x and y:
x[0]=5
y[0]=150
x[1]=7
y[1]=392
x[2]=11
y[2]=1452
x[3]=13
y[3]=2366
x[4]=17
y[4]=5202
Enter interpolation point: 9
Interpolated value at 9.000 is 810.000.

Example 2. Find f(10) from the following data

x	5	6	9	11
y	12	13	14	16

Using Lagrange's interpolation formula.

Output :

Enter number of data points: 4
Enter data for x and y:
x[0]=5
y[0]=12
x[1]=6
y[1]=13
x[2]=9
y[2]=14
x[3]=11
y[3]=16
Enter interpolation point: 10
Interpolated value at 10.000 is 14.667.

Lab 11: Program to evaluate integral using Simpson's 1/3rd and 3/8th rules.

"Simpson's 1/3rd rule"

Example 1. Evaluate $\int_0^1 \left[\sqrt{1+x^3}\right] dx$, taking n = 10 using Simpson's 1/3rd rule.

Python Command :

```
from sympy import *
var('x,y')
f=lambda x:sqrt(1+x**3)
x0=0; xn=1; n=10; h=(xn-x0)/n
soln=f(x0)+f(xn)
for i in range(1,n):
   if i%2==0:
      yn=2*f(x0+i*h)
   else:
      yn=4*f(x0+i*h)
   soln=soln+yn
soln=round(h/3*soln,4)
print("\n Estimated value of given integration:",soln)
```

Output :

```
Estimated value of given integration: 1.1114
```

Example 2. Evaluate $\int_0^1 \left[\frac{1}{1+x}\right] dx$, taking n = 6 using Simpson's 1/3rd rule.

Example 3. Evaluate $\int_0^6 \left[\frac{1}{1+x^2}\right] dx$ using Simpson's 1/3rd rule by taking n = 6.

"Simpson's 3/8th rule"

Example 1. Evaluate $\int_0^1 \left[\frac{1}{1+x}\right] dx$ using Simpson's 3/8th rule taking n = 6.

Python Command:

```
from sympy import *
var('x,y')
f=lambda x:1/(1+x)
x0=0; xn=1; n=6; h=(xn-x0)/n
soln=f(x0)+f(xn)
for i in range(1,n):
   if i%3==0:
      yn=2*f(x0+i*h)
   else:
      yn=3*f(x0+i*h)
   soln=soln+yn
soln=round(3*h/8*soln,4)
print("\n Estimated value of I =",soln)
```

Output:

```
Estimated value of I = 0.6932
```

Example 2. Evaluate $\int_0^6 \left[\frac{1}{1+x^2}\right] dx$ using Simpson's 3/8th rule taking n = 6.

Example 3. Evaluate $\int_0^3 \left[\frac{1}{(1+x)^2}\right] dx$ using Simpson's 3/8th rule taking n = 3.

Lab 12: Program to evaluate integral using Trapezoidal and Weddle rules.

"Trapezoidal rule"

Example 1. Evaluate: $\int_0^1 \left[\sqrt{1+x^3}\right] dx$**, taking n = 10 using trapezoidal rule.**

Python Command:

```
from sympy import *
var('x,y')
f=lambda x:sqrt(1+x**3)
x0=0; xn=1; n=10; h=(xn-x0)/n
soln=f(x0)+f(xn)
for i in range(1,n):
   soln=soln+2*f(x0+i*h)
soln=round(h/2*soln,4)
print("\n Estimated value of given integration:",soln)
```

Output:

```
Estimated value of given integration: 1.1123
```

Example 2. Evaluate: $\int_0^1 \left[\frac{1}{1+x}\right] dx$ **using trapezoidal rule by taking n = 6.**

Example 3. Evaluate: $\int_0^6 \left[\frac{1}{1+x^2}\right] dx$ **using trapezoidal rule by taking n = 6.**

"Weddle rule"

Example 1. Evaluate: $\int_0^6 \left[\frac{1}{1+x^2}\right] dx$ **using weddle's rule by taking n = 6.**

Python Command:

```
def f(x):
    return 1/(1+x*x)
def accept():
    global x0,xn,n
    x0=float(input("Enter the lower limit:"))
    xn=float(input("Enter the upper limit:"))
    n=int(input("Enter the number of intervals:"))
def weddle():
    s=0
    h=(xn-x0)/n
        if n%6==0:
                s=s+((3*h/10)*(f(x0)+f(x0+2*h)+5*f(x0+h)+6*f(x0+3*h)+
                f(x0+4*h)+5*f(x0+5*h)+f(x0+6*h)))
                print("The value after calculation is: ",round(s,6))
        else:
                print("The integration can not be done using Weddle's Rule")
#calling the functions to perform the integration using weddle's formula
x0,xn,n=0.0,0.0,0
accept()
weddle()
```

Output:

```
Enter the lower limit:0

Enter the upper limit:6

Enter the number of intervals:6

The value after calculation is:  1.373447
```

Example 2. Evaluate: $\int_{0.2}^{0.5} e^x \sin 2x \, dx$ **using weddle's rule by taking n = 6.**

Example 3. Evaluate: $\int_4^{5.2} \log x \, dx$ **using trapezoidal rule by taking n = 6.**