SAPTHAGIRI NPS UNIVERSITY

Department of Mathematics

School of Engineering and Technology

Lab Manual for Computational Mathematics using Python Programming

(Second Semester)

24BTPHY202: COMPUTATIONAL MATHEMATICS USING PYTHON PROGRAMMING

LIST OF PROGRAMS:

Lab 1: Find the Curvature and Radius of Curvature of a given curve.

Lab 2: Find the solution for Lagrange's linear partial differential equation

Lab 3: Solution of ordinary differential equation of first order and first degree by Maclaurin's expansion.

Lab 4: Program to compute area and volume by double and triple integration.

Lab 5: Evaluation of Beta and Gamma functions.

Lab 6: Solution of the algebraic and transcendental equations by Regula-Falsi and Newton Raphson Method.

Lab 7: Interpolation /Extrapolation using Newton's forward and backward difference formula.

Lab 8: Computation of area under the curve using Trapezoidal, Simpson's $\left(\frac{1}{3}\right)^{rd}$ and Simpsons $\left(\frac{3}{8}\right)^{th}$ rule.

PYTHON PROGRAMMING LANGUAGE

Introduction: Python is a high-level, general-purpose programming language known for its simplicity and readability. It was created by Guido van Rossum and first released in 1991. Python supports multiple programming paradigms, including procedural, object-oriented, and functional programming. It has become one of the most popular programming languages, widely used for web development, data science, artificial intelligence, automation, scripting, and more.

Key features of Python include:

1. **Readability:** Python's syntax is designed to be clear and readable, making it easy for developers to write and maintain code.

2. **Versatility:** Python can be used for various applications, from simple scripts to complex software development.

3. **Large Standard Library:** Python comes with a comprehensive standard library that provides modules and packages for various tasks, reducing the need for external dependencies.

4. **Community Support:** Python has a large and active community of developers, which contributes to the availability of numerous third-party libraries and frameworks.

5. **Interpreted Language:** Python is an interpreted language, meaning that code can be executed directly without compilation. This makes development and debugging more straightforward.

6. **Cross-platform:** Python is compatible with various operating systems, including Windows, macOS, and Linux, making it a cross-platform language.

Anaconda Navigator is a package manager, environment manager and python distribution with a collection of 1500 + open-source package with free community support.

Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. It supports various programming languages, with Python being one of the most widely used.

Basics of Python:

- ❖ To run the program → Shift + Enter
- ❖ Addition → +
- ❖ Subtraction → -
- ❖ Multiplication → *
- ❖ Division → / (Quotient as answer)
- ❖ Floor division → // (Quotient value without decimal as answer)
- ❖ Modulus → % (Remainder as answer)
- ❖ Absolute value → abs() (Returns the absolute value of a number)
- ❖ Round value → round() (function rounds a number to a specified number of decimal places)
- ❖ Power value → ** or pow(base value, power value)
- ❖ math.sqrt(x) →Returns the square root of x.
- ❖ math.exp(x) →Returns the e raised to the power x.
- ❖ math.log(x, base) → Returns the logarithm of x to the given base. If the base is not specified, it returns the natural logarithm.
- ❖ math.log10(x) →Returns the base-10 logarithm of x.
- ❖ math.factorial(x) → Returns the factorial of x.
- ❖ math.gcd(a, b) →Returns the greatest common divisor of a and b.

Solved examples:

```
2+7
```
```
9
```

```
9-2
```
```
7
```

```
7*7
```
```
49
```

```
7/2
```
3.5

```
7//2
```
3

```
7%2
```
1

```
abs(-7)
```
7

```
round(3.6)
```
4

```
pow(2,3)
```
8

(or)

```
2**3
```
8

```
import math
math.sqrt(25)
```
5.0

```
import math
math.exp(2)
```
7.3890560

```
import math
math.log(1)
```
0.0

(or)

```
import math
math.log(4,2)
```
2

```
import math
math.factorial(3)
```
6

```
import math
math.gcd(2,3)
```

```
1
```

Syntax for creating the matrix:

Syntax:

```
import numpy as np

# Creating a 3x3 matrix
matrix = np.array([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
])

print(matrix)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

Trigonometric Functions:

Syntax:

```
math.sin(x), math.cos(x), math.tan(x):
```

Returns the sine, cosine, and tangent of x, respectively.

Examples:

```
math.sin(0)
```

```
0
```

Differentiation:

Syntax:

```
from sympy import*
```

```
x,y=symbols(‘x,y’)
function=diff(function, independent variable)
display(function)
```

Example:

Differentiate $f = x^2 + 2x$ w.r.t. x

Code:

```
from sympy import*
x,y=symbols('x,y')
function=diff(x**2+2*x, x)
display(function)
```

```
2x+2
```

Differential equations:

Syntax:

```
from sympy import*
init_printing()
x,y=symbols(‘x,y’)
y=Function(“y”)(x)
y1=Derivative(y,x)z1=dsolve(Eq(),y)
display(z1)
```

Example: Solve $\frac{dy}{dx} + y\tan x - y^3 \sec x = 0$

Code:

```
from sympy import*
init_printing()
x,y=symbols('x,y')
y=Function("y")(x)
y1=Derivative(y,x)
z1=dsolve(Eq(y1+y*tan(x)-y**3*sec(x),0),y)
display(z1)
```

Output: $y(x) = -\sqrt{\frac{1}{C_1 - 2\sin x}}\cos x$ or $y(x) = \sqrt{\frac{1}{C_1 - 2\sin x}}\cos x$

Integrals:

Syntax:

```
from sympy import*
x,y=symbols(‘x,y’)
function=integrate(function, independent variable)
display(function)
```

Example: Integrate $f = x^2 + 2x$

Code:

```
from sympy import*
x,y=symbols('x,y')
f=x**2+3*x
W1=integrate(f,x)
display(W1)
```

Output: $\frac{x^3}{3} + \frac{3x^2}{2}$

Double and triple integral integration:

Evaluate the integral $\int_0^1 \int_0^x (x^2 + y^2)dydx$

Code:

```
from sympy import*
x,y,z=symbols('x,y,z')
f=x**2+y**2
W1=integrate(f, (y,0,x), (x,0,1))
display(W1)
```

Output: 1/3

Lab 1: Find the Curvature and Radius of Curvature of a given curve.

Use python

1. To find radius of curvature.

Radius of curvature:

Formula to calculate Radius of curvature in polar form is $\rho = \frac{(r^2+{r_1}^2)^{3/2}}{r^2+2{r_1}^2-rr_2}$

Syntax for the commands used:

The print(): function prints the specified message to the screen or other standard output device.

1. Find the radius of curvature, $r = 4(1 + cost)$ at $t = \frac{\pi}{2}$.

Code:

```python
from sympy import symbols, cos, pi, Derivative

# Define symbols
t = symbols('t')

# Define r in terms of t
r = 4 * (1 + cos(t))

# Find the first and second derivatives of r with respect
to t
r1 = Derivative(r, t).doit()
r2 = Derivative(r1, t).doit()

# Define the radius of curvature formula
rho = (r**2 + r1**2)**(1.5) / (r**2 + 2*r1**2 - r*r2)

# Substitute t = pi/2 in rho
rho1 = rho.subs(t, pi/2)

# Print the radius of curvature
print(f'The radius of curvature is {rho1:.4f} units')
```

Output:

The radius of curvature is 3.7712 units

2. Find the radius of curvature for $r = a\sin(nt)$ at $t = \frac{\pi}{2}$ and $n = 1$.

Code:

```python
from sympy import symbols, sin, pi, Derivative, simplify

# Define symbols
t, r, a, n = symbols('t r a n')

# Define r in terms of t
r = a * sin(n * t)

# Find the first and second derivatives of r with respect
to t
r1 = Derivative(r, t).doit()
r2 = Derivative(r1, t).doit()

# Define the radius of curvature formula
rho = (r**2 + r1**2)**1.5 / (r**2 + 2 * r1**2 - r * r2)

# Substitute t = pi/2 and n = 1 in rho
rho1 = rho.subs({t: pi / 2, n: 1})

# Simplify the result
rho1_simplified = simplify(rho1)

# Print the radius of curvature
print("The radius of curvature is")
display(rho1_simplified)
```

Output:
The radius of curvature is

$$\frac{(a^2)^{1.5}}{2a^2}$$

Exercise:

3. Find the radius of curvature, $r = a(1 - \cos(t))$ at $t = \frac{\pi}{2}$.
4. Find the radius of curvature, $r = a\cos(t)$ at $t = \frac{\pi}{4}$.

Parametric curves:

The formula to calculate Radius of curvature is $\rho = \frac{(x'^2+y'^2)^{\frac{3}{2}}}{y''x'-x''y^1}$

$$x' = \frac{dx}{dt}, x'' = \frac{d^2x}{dt^2}; \; y' = \frac{dy}{dt}, y'' = \frac{d^2y}{dt^2}$$

5. Find radius of curvature of $x = a\cos(t)$ and $y = a\sin(t)$

Code:

```
from sympy import symbols, sqrt, sin, cos, pi, Derivative,
simplify, ratsimp
from sympy.abc import x, y, t, a

# Define parametric equations
y = a * sin(t)
x = a * cos(t)

# Calculate the derivatives of the parametric equations
dydx = simplify(Derivative(y, t).doit()) /
simplify(Derivative(x, t).doit())

# Define the radius of curvature formula
rho = simplify((1 + dydx**2)**1.5 / (Derivative(dydx,
t).doit() / Derivative(x, t).doit()))

# Display the radius of curvature
print('Radius of curvature is')
display(ratsimp(rho))
```

Output:

Radius of curvature is

$$-a(\frac{1}{sin^2 t})^{0.5}\sin(t)$$

Exercise:

6. Find radius of curvature of $x = a(t - \sin(t)$ and $y = \mathrm{a}(1 - \cos(t))$

Lab 2: Find the solution for Lagrange's linear partial differential equation

Use python

1. To find the solution for Lagrange's Method.

Lagrange's Method:

Syntax for the commands used:

- print(): function prints the specified message to the screen, or other standard output device.
- display(): function will take input as many objects created using the classes mentioned above and display them sequentially.

1. Find the solution for Lagrange's Method $ap + aq = z$. where $z = f(x, y)$

Code:

```
from sympy.solvers.pde import pdsolve
from sympy import Function , Eq , cot , classify_pde ,
pprint
from sympy . abc import x , y , a
f = Function ('f')
z = f (x , y )
zx = z . diff ( x )
zy = z . diff ( y )
# Solve ap+aq=z
eq = Eq ( a*zx+a*zy , z )
pprint ( eq )
print ("\n")
soln = pdsolve ( eq , z )
pprint ( soln )
```

Output:

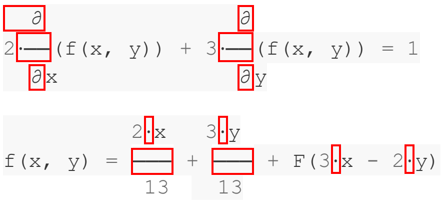
```
  ∂                ∂
a·──(f(x, y)) + a·──(f(x, y)) = f(x, y)
  ∂x               ∂y
                                  a·x + a·y
                                  ─────────
                                       2
                                    2·a
f(x, y) = F(a·x - a·y)·e
```

2. Find the solution for Lagrange’s Method $2p + 3q = 1$.

Code:

```python
from sympy . solvers . pde import pdsolve
from sympy import Function , Eq , cot , classify_pde ,
pprint
from sympy . abc import x , y , a
f = Function ('f')
z = f (x , y )
zx = z . diff ( x )
zy = z . diff ( y )
# Solve 2p+3q=1
eq = Eq ( 2*zx+3*zy , 1 )
pprint ( eq )
print ("\n")
soln = pdsolve ( eq , z )
pprint ( soln )
```

Output:



```
  ∂                ∂
2·──(f(x, y)) + 3·──(f(x, y)) = 1
  ∂x               ∂y

            2·x    3·y
f(x, y) = ──── + ──── + F(3·x - 2·y)
            13     13
```

Exercise:

3. Find the solution for Lagrange’s Method $xzp + yzq = xy$.
4. Find the solution for Lagrange’s Method $(mz - ny)p + (nx - lz)q = ly - mx$.

Lab 3: Solution of ordinary differential equation of first order and first degree by Maclaurin’s expansion

Use python

1. To find the solution for Maclaurin’s expansion.

Maclaurin’s expansion:

Syntax for the commands used:

- sp.symbols('x'): Creates a symbolic variable x.
- sp.exp(x): Defines the exponential function
- sp.series(f, x, 0, num_terms): Computes the Maclaurin series of the function f around x=0 up to the specified number of terms (num_terms).
- .removeO(): Removes the big-O notation term from the series, representing the order of the omitted terms.

5. Obtain the Maclaurin’s expansion of e^x

Code:

```
import sympy as sp

# Define the variable and the function
x = sp.symbols('x')
f = sp.exp(x)   # The exponential function

# Define the number of terms in the expansion
num_terms = 10

# Compute the Maclaurin series expansion
maclaurin_series = sp.series(f, x, 0, num_terms).removeO()

# Display the result
print(maclaurin_series)
```

Output:

```
x**9/362880 + x**8/40320 + x**7/5040 + x**6/720 + x**5/120 + x**4/24 + x**3/6 +
x**2/2 + x + 1
```

6. Obtain the Maclaurin's expansion of $e^{x \cos x}$.

Code:

```
import sympy as sp

# Define the variable and the function
x = sp.symbols('x')
f = sp.exp(x*cos(x))   # The exponential function

# Define the number of terms in the expansion
num_terms = 10

# Compute the Maclaurin series expansion
maclaurin_series = sp.series(f, x, 0, num_terms).removeO()

# Display the result
print(maclaurin_series)
```

Output:

```
-58*x**9/2835 + 193*x**8/4480 + 13*x**7/105 + 61*x**6/720 – x**5/5 – 11*x**4/24 –
x**3/3 + x**2/2 + x + 1
```

Exercise:

7. Obtain the Maclaurin's expansion of $\sin(x)$
8. Obtain the Maclaurin's expansion of $e^{xsin(x)}$

Lab 4: Program to compute area and volume by double and triple integration.

Use python

1. to evaluate double and triple integration.
2. to compute area and volume.

Syntax for the commands used:

1. Data pretty printer in Python:

```
pprint()
```

2. Integrate:

```
integrate(function,(variable, min_limit, max_limit))
```

Double and triple integration:

1. Evaluate the integral $\int_0^1 \int_x^{\sqrt{x}} xydydx$

Code:

```
from sympy import *
x,y=symbols('x y')
w1=integrate(x*y,(y,sqrt(x),x),(x,0,1))
print(w1)
```

Output:

```
-1/24
```

2. Evaluate the integral $\int_0^3 \int_0^{3-x} \int_0^{3-x-y} (xyz)\, dzdydx$

Code:

```
from sympy import *
x,y,z=symbol('x y z')
w2=integrate((x*y*z),(z,0,3-x-y),(y,0,3-x),(x,0,3))
print(w2)
```

Output:

81/80

Exercise:

3. Evaluate $\int_0^1 \int_0^1 \frac{dxdy}{\sqrt{(1-x^2)(1-y^2)}}$
4. Evaluate $\int_0^3 \int_1^2 xy(1+x+y)dydx$

Area:

- import scipy.integrate as integrate: Imports the integrate module from the SciPy library, which provides functions for numerical integration.

5. Find the area lying between the parabola $y = 4x - x^2$ and the line $y = x$.

Code:

```
import scipy.integrate as integrate

# Define the functions for the parabola and the line
def parabola(x):
    return 4 * x - x ** 2

def line(x):
    return x

# Define the integrand as the difference between the
parabola and the line
def integrand(x):
    return parabola(x) - line(x)

# Define the limits of integration (the points of
intersection)
a = 0
b = 3

# Perform the integration
area, error = integrate.quad(integrand, a, b)

# Print the result
print(f"The area lying between the parabola y = 4x - x^2
and the line y = x is {area}")
```

Output:

```
The area lying between the parabola y = 4x - x^2 and the line y = x is 4.5
```

6. Find the smaller of the areas bounded by the ellipse $4x^2 + 9y^2 = 36$ and the straight line $2x + 3y = 6$.

Code:

```python
import numpy as np
from scipy.integrate import quad

# Define the upper part of the ellipse
def upper_ellipse(x):
    return (2/3) * np.sqrt(9 - x**2)

# Define the line
def line(x):
    return (6 - 2*x) / 3

# --ne the integrand as the difference between the upper
ellipse and the line
def integrand(x):
    return upper_ellipse(x) - line(x)

# Define the limits of integration (the points of
intersection)
a = -1.8
b = 3

# Perform the integration
area, error = integrate.quad(integrand, a, b)

# Print the result
print(f"The area lying between the ellipse and the line is
{area:.4f} square units")
```

Output:

```
The area lying between the ellipse and the line is 0.4029 square units
```

Exercise:

7. Find the area bounded by $x = 2y - y^2$ and $x = y^2$.
8. Find the area between the parabolas $y^2 = 4ax$ and $x^2 = 4ay$.

Volume:

9. Find the volume of the tetrahedron $\frac{x}{a} + \frac{y}{b} + \frac{z}{c} = 1$.

```python
import scipy.integrate as integrate

# Define the limits of the variables
a = 1  # Replace with actual value of 'a'
b = 1  # Replace with actual value of 'b'
c = 1  # Replace with actual value of 'c'

# Define the integrand functions
def z_limit(x, y):
    return c * (1 - x / a - y / b)

def y_limit(x):
    return b * (1 - x / a)

# Perform the triple integration
volume, error = integrate.tplquad(lambda x, y, z: 1,
                                  0, a,
                                  lambda x: 0, y_limit,
                                  lambda x, y: 0, z_limit)

# Print the result
print(f"The volume of the tetrahedron is {volume:.4f} cubic
units")
```

Output:

```
The volume of the tetrahedron is 0.1667 cubic units
```

10. Find the volume bounded by the cylinder $x^2 + y^2 = 4$ and the planes $y + z = 4$ and $z = 0$.

Code:

```
import sympy as sp

# Define the variables
r, theta, z = sp.symbols('r theta z')

# Define the bounds of the variables
r_min, r_max = 0, 2
theta_min, theta_max = 0, 2 * sp.pi
z_min, z_max = 0, 4 - r * sp.sin(theta)

# Define the integrand (r in cylindrical coordinates)
integrand = r

# Perform the integration step by step
integral_z = sp.integrate(integrand, (z, z_min, z_max))
integral_r = sp.integrate(integral_z, (r, r_min, r_max))
integral_theta = sp.integrate(integral_r, (theta,
theta_min, theta_max))

# Simplify the result
volume = sp.simplify(integral_theta)
volume
```

Output:

16π

Lab 5: Evaluation of Beta and Gamma functions.

Use python

1. to evaluate improper integrals using Beta function.
2. to evaluate improper integrals using Gamma function.

Syntax for commands used:

- Gamma

```
math.gamma(x)
```

  Parameters:
  x: The number whose gamma value needs to be computed.
- Beta

```
math.beta(x,y)
```

  Parameters : x ,y: The numbers whose beta value needs to be computed.

- Note: We can evaluate improper integral involving infinity by using inf.

1. Evaluate $\Gamma\left(\frac{5}{2}\right)$.

Code:

```
import math

result = math.gamma(5/2)
print("Gamma(5/2) =", result)
```

Output:

```
Gamma(5/2) = 1.3293403881791372
```

2. Evaluate $\beta(5,3)$.

Code:

```
#beta and gamma functions
 from sympy import beta, gamma
 m=input('m :');
 n=input('n :');
```

```
m=float(m);
n=float(n);
s=beta(m,n);
t=gamma(n)
print('gamma (',n,') is %0.3f'%t)
print('Beta (',m,n,') is %0.3f'%s)
```

Output:

```
m : 5
n : 3
gamma ( 3.0 ) is 2.000
Beta ( 5.0 3.0 ) is 0.010
```

Exercise:

3. Evaluate $\Gamma\left(\frac{5}{2}\right)$.
4. Evaluate $\Gamma\left(\frac{9}{2}\right)$.
5. Evaluate $\beta(4,5)$
6. Evaluate $\beta\left(\frac{9}{2},\frac{7}{2}\right)$

Lab 6: Solution of the algebraic and transcendental equations by Regula-Falsi and Newton Raphson Method.

Use python

1. To solve an algebraic and transcendental equation by the Regula-Falsi method.
2. To solve algebraic and transcendental equations by the Newton-Raphson method.

Regula-Falsi method:

1. Find a real root of the equation $f(x) = x^3 - 2x - 5$ by Regula Falsi Method correct to three decimal places.

Code:

```
from sympy import *

x = Symbol('x')
g = input('Enter the function (in terms of x): ')   # 
Example: x**3 - 2*x - 5
f = lambdify(x, g)

a = float(input('Enter a value: '))   # Example: 2
b = float(input('Enter b value: '))   # Example: 3
N = int(input('Enter number of iterations: '))   # Example: 
5

for i in range(1, N + 1):
    c = (a * f(b) - b * f(a)) / (f(b) - f(a))
    if f(a) * f(c) < 0:
        b = c
    else:
        a = c
    print('Iteration %d \t The root %0.3f \t Function value 
%0.3f' % (i, c, f(c)))
```

Output:

```
Enter the function (in terms of x):  x**3-2*x-5
Enter a value:  2
Enter b value:  3
Enter number of iterations:  6
Iteration 1          The root 2.059       Function value -0.391
Iteration 2          The root 2.081       Function value -0.147
Iteration 3          The root 2.090       Function value -0.055
Iteration 4          The root 2.093       Function value -0.020
Iteration 5          The root 2.094       Function value -0.007
Iteration 6          The root 2.094       Function value -0.003
```

Exercise:

2. Find the real root of the equation $cosx = 3x - 1$ correct to three decimals by using the Regula Falsi method.
3. Find the root of the equation $e^x - cosx = 0$ by the Regula Falsi method

Newton-Raphson method:

4. Find a real root of the equation $3x - cosx - 1$ correct to three decimal places using Newton Raphson method.

Code:

```
from sympy import *
x= Symbol ('x')
g = input ('Enter the function ') #%3x -cos(x)-1; %
function
f= lambdify (x , g )
dg = diff ( g ) ;
df= lambdify (x , dg )
x0= float ( input ('Enter the initial approximation ') ) ;
# x0=1
n= int( input ('Enter the number of iterations ') ) ; #n=5;
for i in range (1 , n+1 ):
    x1 =( x0 - ( f ( x0 )/df ( x0 ) ) )
    x0 = x1
    print ('iteration %d \t the root %0.3f \t function
value %0.3f \n'%(i , x1 , f ( x1 ) ) ) ; # print all
iteration # value x0 = x1
```

Output:

```
Enter the function  3*x-cos(x)-1
Enter the intial approximation  1
Enter the number of iterations  5
```

```
itration 1   the root 0.620        function value 0.046

itration 2   the root 0.607        function value 0.000

itration 3   the root 0.607        function value 0.000

itration 4   the root 0.607        function value 0.000

itration 5   the root 0.607        function value -0.000
```

Exercise:

5. Find a real root of the equation $x^3 - 5x + 11 = 0$ correct to three decimal places using Newton Raphson method.
6. Find a real root of the equation $e^x - sinx = 0$ correct to three decimal places using Newton Raphson method.

Lab 7: Interpolation /Extrapolation using Newton’s forward and backward difference formula

Use python

1. to interpolate using Newton’s Forward interpolation method.
2. to interpolate using Newton’s backward interpolation method.

Newton’s Forward interpolation method:

1. Use Newtons forward interpolation to obtain the interpolating polynomial and hence calculate $y(1.4)$ for the following:

x	1	2	3	4	5
y	10	26	58	112	194

Code:

```python
from sympy import symbols, simplify, pprint, lambdify
import numpy as np

# Read number of data points
n = int(input('Enter number of data points: '))

x = np.zeros((n))
y = np.zeros((n, 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][0] = float(input('y[' + str(i) + ']='))
# Generating forward difference table
for i in range(1, n):
    for j in range(n - i):
        y[j][i] = y[j + 1][i - 1] - y[j][i - 1]

# Displaying forward difference table
print('\nFORWARD DIFFERENCE TABLE\n')
for i in range(n):
    print('%0.2f' % (x[i]), end='')
    for j in range(n - i):
        print('\t\t%0.2f' % (y[i][j]), end='')
    print()

# Obtaining the polynomial
t = symbols('t')
p = 1
poly = y[0][0]
h = x[1] - x[0]

for i in range(1, n):
    p *= (t - x[i - 1]) / (i * h)
    poly += y[0][i] * p

simp_poly = simplify(poly)
print('\nTHE INTERPOLATING POLYNOMIAL IS\n')
pprint(simp_poly)

# If you want to interpolate at some point the next section will
help
inter = input('Do you want to interpolate at a point(y/n)? ')   # y
if inter == 'y':
    a = float(input('Enter the point: '))   # Example: 2
    interpol = lambdify(t, simp_poly)
    result = interpol(a)
    print('\nThe value of the function at',a,'is\n',result);
```

Output:

```
Enter number of data points:  5
Enter data for x and y:
x[0]= 1
y[0]= 10
x[1]= 2
y[1]= 26
x[2]= 3
y[2]= 58
x[3]= 4
y[3]= 112
x[4]= 5
y[4]= 194
```

```
FORWARD DIFFERENCE TABLE

1.00            10.00           16.00           16.00
        6.00            0.00
2.00            26.00           32.00           22.00
        6.00
3.00            58.00           54.00           28.00
4.00            112.00          82.00
5.00            194.00

THE INTERPOLATING POLYNOMIAL IS

  3     2
1.0·t  + 2.0·t  + 3.0·t + 4.0
Do you want to interpolate at a point(y/n)?  y
Enter the point:  1.4

The value of the function at 1.4 is
 14.863999999999997
```

Exercise:

2. Use Newton forward interpolation to obtain the interpolating polynomial and hence calculate $y(1.5)$ for the following:

x	0	1	2	3	4
y	7.4720	7.5854	7.6922	7.8119	7.9252

3. Use Newton forward interpolation to obtain the interpolating polynomial and hence calculate $y(1.4)$ for the following:

x	1	2	3	4	5
y	10	26	58	112	194

Newton's backward interpolation method:

4. Use Newtons backward interpolation to obtain the interpolating polynomial and hence calculate $y(8)$ for the following:

x	1	3	5	7	9
y	6	10	62	210	502

Code:

```python
from sympy import symbols, simplify, pprint, lambdify
import numpy as np
```

```
print("This will use Newton's backward interpolation
formula.")

# Reading number of data points
n = int(input('Enter number of data points: '))

# Initializing arrays for x and y values, and differences
of y
x = np.zeros((n))
y = np.zeros((n, 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][0] = float(input('y[' + str(i) + ']='))

# Generating backward difference table
for i in range(1, n):
    for j in range(n - 1, i - 2, -1):
        y[j][i] = y[j][i - 1] - y[j - 1][i - 1]

print('\nBACKWARD DIFFERENCE TABLE\n')
for i in range(n):
    print(f'{x[i]:0.2f}', end='')
    for j in range(i + 1):
        print(f'\t{y[i][j]:0.2f}', end='')
    print()

# Obtaining the polynomial
t = symbols('t')
f = []
p = (t - x[n - 1]) / (x[1] - x[0])
f.append(p)
for i in range(1, n - 1):
    f.append(f[i - 1] * (p + i) / (i + 1))

poly = y[n - 1][0]
for i in range(n - 1):
    poly = poly + y[n - 1][i + 1] * f[i]
```

```python
simp_poly = simplify(poly)

print('\nTHE INTERPOLATING POLYNOMIAL IS\n')
pprint(simp_poly)

# Interpolate at a specific point if desired
inter = input('Do you want to interpolate at a point (y/n)?
')
if inter.lower() == 'y':
    a = float(input('Enter the point: '))
    interpol = lambdify(t, simp_poly)
    result = interpol(a)
    print(f'\nThe value of the function at {a} is
{result}')
```

Output:

```
This will use Newton's backward interpolation formula.
Enter number of data points:  5
Enter data for x and y:
x[0]= 1
y[0]= 6
x[1]= 3
y[1]= 10
x[2]= 5
y[2]= 62
x[3]= 7
y[3]= 210
x[4]= 9
y[4]= 502

BACKWARD DIFFERENCE TABLE

1.00        6.00
3.00        10.00       4.00
5.00        62.00       52.00       48.00
7.00        210.00      148.00      96.00       48.00
9.00        502.00      292.00      144.00      48.00       0.00

THE INTERPOLATING POLYNOMIAL IS

   3      2
1.0·t  - 3.0·t  + 1.0·t + 7.0
Do you want to interpolate at a point (y/n)?  y
```

```
Enter the point:  8

The value of the function at 8.0 is 335.0
```

Exercise:

5. Use Newtons backward interpolation to obtain the interpolating polynomial and hence calculate $y(4.5)$ for the following:

x	1	2	3	4	5
y	1	8	27	64	125

6. Use Newtons backward interpolation to obtain the interpolating polynomial and hence calculate $y(8)$ for the following:

x	2	4	6	8	10
y	10	30	50	70	90

Lab 8: Computation of area under the curve using Trapezoidal, Simpson's $(\frac{1}{3})^{rd}$and Simpsons $(\frac{3}{8})^{th}$ rule.

Use python

1. to find area under the curve represented by a given function using Trapezoidal rule.

2. to find area under the curve represented by a given function using Simpson's $(\frac{1}{3})^{rd}$ rule.

3. to find area under the curve represented by a given function using Simpson's $(\frac{1}{3})^{rd}$ rule.

1. Evaluate $\int_0^1 \frac{dx}{1+x}$ taking seven ordinates by applying

a) Trapezoidal rule,

b) Simpson's $(\frac{1}{3})^{rd}$ rule

c) Simpson's $(\frac{3}{8})^{th}$ rule

Trapezoidal rule:

Code:

```
# Definition of the function to integrate
def my_func(x):
    return 1 / (1 + x)

# Function to implement trapezoidal method
def trapezoidal(x0, xn, n):
    h = (xn - x0) / n
    # Finding sum
    integration = my_func(x0) + my_func(xn)
    for i in range(1, n):
        k = x0 + i * h
        integration += 2 * my_func(k)
    # Proportioning sum of trapezoid areas
    integration *= h / 2
    return integration

# Input section
lower_limit = float(input("Enter lower limit of 
integration: "))
upper_limit = float(input("Enter upper limit of 
integration: "))
sub_interval = int(input("Enter number of sub intervals: 
"))

# Call trapezoidal() method and get result
result = trapezoidal(lower_limit, upper_limit, 
sub_interval)

# Print result
print("Integration result by Trapezoidal method is:", 
result)
```

Output:

```
Enter lower limit of integration:  0
Enter upper limit of integration:  1
Enter number of sub intervals:  10
Integration result by Trapezoidal method is: 0.693771403175428
```

Simpson's $(\frac{1}{3})^{rd}$ rule:

Code:

```
# Definition of the function to integrate
def my_func(x):
    return 1 / (1 + x)

# Function to implement Simpson's one-third rule
def simpson13(x0, xn, n):
    h = (xn - x0) / n
    # Finding sum
    integration = my_func(x0) + my_func(xn)
    k = x0 + h
    for i in range(1, n):
        if i % 2 == 0:
```

```python
            integration += 2 * my_func(k)
        else:
            integration += 4 * my_func(k)
        k += h
    # Finding final integration value
    integration *= h / 3
    return integration

# Input section
lower_limit = float(input("Enter lower limit of
integration: "))
upper_limit = float(input("Enter upper limit of
integration: "))
sub_interval = int(input("Enter number of sub intervals
(must be even): "))

# Ensure the number of sub intervals is even
if sub_interval % 2 != 0:
    sub_interval += 1
    print(f"Number of sub intervals adjusted to
{sub_interval} for Simpson's 1/3 rule.")

# Call simpson13() method and get result
result = simpson13(lower_limit, upper_limit, sub_interval)

# Print result
print(f"Integration result by Simpson's 1/3 method is:
{result:.6f}")
```

Output:

```
Enter lower limit of integration:  0
Enter upper limit of integration:  1
Enter number of sub intervals (must be even):  10
Integration result by Simpson's 1/3 method is: 0.693150
```

Simpson's $(\frac{3}{8})^{th}$ rule:

Code:

```python
def simpsons_3_8_rule(f, a, b, n):
    h = (b - a) / n
    s = f(a) + f(b)
    for i in range(1, n, 3):
        s += 3 * f(a + i * h)
    for i in range(2, n, 3):
        s += 3 * f(a + i * h)
    for i in range(3, n, 3):
        s += 2 * f(a + i * h)
    return s * 3 * h / 8

def f(x):
    return 1 / (1 + x )   # function here

a = 0   # lower limit
b = 1   # upper limit
n = 10   # number of sub intervals

result = simpsons_3_8_rule(f, a, b, n)
print(f'{result:.5f}')
```

Output:

```
0.68035
```

Exercise:

2. Evaluate $\int_0^6 \frac{dx}{1+x^2}$ taking seven ordinates by applying

a) Trapezoidal rule,
b) Simpson's $(\frac{1}{3})^{rd}$ rule
c) Simpson's $(\frac{3}{8})^{th}$ rule

3. Evaluate $\int_0^1 e^x dx$ taking seven ordinates by applying

a) Trapezoidal rule,

b) Simpson's $(\frac{1}{3})^{rd}$ rule

c) Simpson's $(\frac{3}{8})^{th}$ rule

4. Evaluate $\int_0^1 sin^2 x\, dx$ taking seven ordinates by applying

d) Trapezoidal rule,

e) Simpson's $(\frac{1}{3})^{rd}$ rule

f) Simpson's $(\frac{3}{8})^{th}$ rule