ASSIGNMENT

NUMERICAL STATISTICAL TECHNIQUES

Name : Aditya Aman

Batch: CS- A

Sem: 4^{th}

Roll no. : 09

Date: 27/02/2025

Submiteed to: Ms. Meena Varghese

1.Find the root of the equation x^3-4x-9=0 using bisection method, correct upto 3 decimal places

```
def f(x): return x**3 - 4*x - 9

def bisection(a, b, tol=0.001):

if f(a) * f(b) >= 0: print("Bisection method fails. Choose a different interval.");
return None

mid = (a + b) / 2.0

while abs(f(mid)) > tol:

if f(a) * f(mid) < 0: b = mid

else: a = mid

mid = (a + b) / 2.0

return round(mid, 3)

a, b = 2, 3

root = bisection(a, b)

if root is not None: print(f"Root of the equation x^3 - 4x - 9 = 0 is
approximately: {root}")
```

Output:

Root of the equation x^3 - 4x - 9 = 0 is approximately: 2.706

2.Using Gauss-Seidel method, solve : 20x+y-2z=17, 3x+20y-z=-18,2x-3y+20z=25

```
import numpy as np

def gauss_seidel(iterations=10, tol=1e-3):

x, y, z = 0, 0, 0

for _ in range(iterations):

x_new = (17 - y + 2*z) / 20

y_new = (-18 - 3*x_new + z) / 20
```

```
z_new = (25 - 2*x_new + 3*y_new) / 20
if abs(x - x_new) < tol and abs(y - y_new) < tol and abs(z - z_new) < tol: break
x, y, z = x_new, y_new, z_new
return round(x, 3), round(y, 3), round(z, 3)
x, y, z = gauss_seidel()
print(f"Solution: x = {x}, y = {y}, z = {z}")
```

Output:

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

3.Method of Least Square:

x : 1.0	1.5	2.0	2.5	3.0	3.5	4.0
y : 1.1	1.3	1.6	2.0	2.7	3.4	4.1

Code:

```
import numpy as np
x = np.array([1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0])
y = np.array([1.1, 1.3, 1.6, 2.0, 2.7, 3.4, 4.1])
n = len(x)
sum_x, sum_y, sum_xy, sum_x2 = np.sum(x), np.sum(y), np.sum(x*y),
np.sum(x**2)
m = (n * sum_xy - sum_x * sum_y) / (n * sum_x2 - sum_x**2)
c = (sum_y - m * sum_x) / n
print(f"Equation of best-fit line: y = {round(m, 3)}x + {round(c, 3)}")
```

Output:

Equation of best-fit line: y = 1.061x - 0.329

4.Using Trapezoidal rule, evaluate:

$$\int_0^6 dx/(1+x^2)$$

Code:

```
import numpy as np
def f(x): return 1 / (1 + x**2)
a, b, n = 0, 6, 6
h = (b - a) / n
x = np.linspace(a, b, n + 1)
y = f(x)
integral = (h / 2) * (y[0] + 2 * np.sum(y[1:-1]) + y[-1])
print(f"Approximate value of the integral: {round(integral, 3)}")
```

Output:

Approximate value of the integral: 1.373

5.Using Runge-Kutta (4^{th} order , find the approximated value of y at x=0.2 given dy/dx=x+y, y(0)=1

```
def f(x, y): return x + y
def runge_kutta(x0, y0, x, h):
    while x0 < x:
        k1 = h * f(x0, y0)
        k2 = h * f(x0 + h/2, y0 + k1/2)
        k3 = h * f(x0 + h/2, y0 + k2/2)
        k4 = h * f(x0 + h, y0 + k3)
        y0 += (k1 + 2*k2 + 2*k3 + k4) / 6
        x0 += h
    return round(y0, 3)
```

```
x0, y0, x, h = 0, 1, 0.2, 0.1

y_approx = runge_kutta(x0, y0, x, h)

print(f"Approximated value of y at x = {x}: {y_approx}")
```

Output:

Approximated value of y at x = 0.2: 1.221