```
PS D:\Mandip NM Lab> py .\bisection.py
Enter the lower bound (a): -1
Enter the upper bound (b): 1
Enter the number of iterations: 10
 Iteration          a          b               c           f(c)
         1  -1.000000   1.000000        0.000000      -4.000000
         2  -1.000000   0.000000       -0.500000      -0.750000
         3  -1.000000  -0.500000       -0.750000       1.062500
         4  -0.750000  -0.500000       -0.625000       0.140625
         5  -0.625000  -0.500000       -0.562500      -0.308594
         6  -0.625000  -0.562500       -0.593750      -0.084961
         7  -0.625000  -0.593750       -0.609375       0.027588
         8  -0.609375  -0.593750       -0.601562      -0.028748
         9  -0.609375  -0.601562       -0.605469      -0.000595

Convergence achieved!

The root of the equation is approximately: -0.605469
PS D:\Mandip NM Lab>
```

```
PS D:\Mandip NM Lab> py .\Newton_Raphson.py
Enter the value of a:2
Enter the number of iterations:5
Iteration(i+1):a=2.0, x=1.375, f1.375=2.34375

The root of the equation is approximately: {1.375}
PS D:\Mandip NM Lab>
```

```
PS D:\Mandip NM Lab> py .\secant.py
Enter the value of a: -3
Enter the value of b: -5
Enter the number of iterations: 5
Iteration 1: a=-3.0, b=-5.0, x=-1.75, f(x)=4.0625
Iteration 2: a=-5.0, b=-1.75, x=-0.2727272727272725, f(x)=6.983471074380166
Iteration 3: a=-1.75, b=-0.2727272727272725, x=-3.8045977011494254, f(x)=7.2565728629937905
Iteration 4: a=-0.2727272727272725, b=-3.8045977011494254, x=90.04054054054073, f(x)=8475.461102994923
Iteration 5: a=-3.8045977011494254, b=90.04054054054073, x=-3.8850154663398393, f(x)=7.5532833083404025

The root of the equation is approximately: -3.8850154663398393
PS D:\Mandip NM Lab>
```

```
PS D:\Mandip NM Lab> py .\fixed_point.py
Enter the initial guess: 2
Enter the number of iterations: 10
 Iteration              x           f(x)
         1       0.000000      -4.000000
         2      -0.666667       0.444444
         3      -0.592593      -0.093278
         4      -0.608139       0.018667
         5      -0.605028      -0.003774
         6      -0.605657       0.000762

Convergence achieved!

The root of the equation is approximately: -0.605657
PS D:\Mandip NM Lab>
```

```
PS D:\Mandip NM Lab> py .\lagrange.py
Enter x values: 1 2 3 4
Enter y values: 2 3 5 7
Enter the point to interpolate: 2.5
Interpolated value: 3.9375
PS D:\Mandip NM Lab>
```

```
PS D:\Mandip NM Lab> py .\Newton_divided.py
Enter x values: 1 2 3 4
Enter y values: 2 3 5 7
Enter the point to interpolate: 2.5
Interpolated value: 3.9375
PS D:\Mandip NM Lab>
```

```
PS D:\Mandip NM Lab> py .\newton_forward.py
Enter the number of positions: 4
Enter the value of x: 7
Enter the value of x at i = 0: 5
Enter the value of f(x) at i = 0: 12
Enter the value of x at i = 1: 6
Enter the value of f(x) at i = 1: 13
Enter the value of x at i = 2: 7
Enter the value of f(x) at i = 2: 14
Enter the value of x at i = 3: 8
Enter the value of f(x) at i = 3: 16

Interpolated Value = 14.000000
PS D:\Mandip NM Lab>
```

```
PS D:\Mandip NM Lab> py .\newton_backward.py
Enter number of data points: 4
Enter x values: 1 2 3 4
Enter y value for x[0]: 1
Enter y value for x[1]: 8
Enter y value for x[2]: 17
Enter y value for x[3]: 63
First Derivative: 46.0
Second Derivative: 27.5
PS D:\Mandip NM Lab>
```

```
PS D:\Mandip NM Lab> py '.\honer''s.py'
Enter the value of x: 2
Result of polynomial evaluation: 63.0
PS D:\Mandip NM Lab>
```

```
PS D:\Mandip NM Lab> py .\Linear_regression.py
Enter x values (space-separated): 1 2 3 4 5
Enter y values (space-separated): 2 4 5 4 5
Linear Regression Equation: y = 2.2000 + 0.6000x
PS D:\Mandip NM Lab>
```

```
PS D:\Mandip NM Lab> py .\Exponential_regression.py
Enter the number of points: 4
Enter the values of x and y:
Point 1 (x and y): 1 2
Point 2 (x and y): 2 4
Point 3 (x and y): 3 8
Point 4 (x and y): 4 16

The curve is: y = 1.000 e^(0.693 x)
PS D:\Mandip NM Lab>
```

```
PS D:\Mandip NM Lab> py .\polynomial_regression.py
Enter the number of points: 4
Enter degree of polynomial to be fitted: 2
Enter the values of x and y:
Point 1 (x and y): 1 2
Point 2 (x and y): 2 3
Point 3 (x and y): 3 5
Point 4 (x and y): 4 7

The polynomial equation is:
y = 1.250 + 0.450 x^1 + 0.250 x^2
PS D:\Mandip NM Lab>
```

```
PS D:\Mandip NM Lab> py .\gauss_elimination.py
Enter the number of variables: 3
Enter row 1 coefficients (space-separated): 2 1 -1
Enter row 2 coefficients (space-separated): -3 -1 2
Enter row 3 coefficients (space-separated): -2 1 2
Enter the constants (space-separated): 8 -11 -3

Solution:
x1 = 2.0000
x2 = 3.0000
x3 = -1.0000
PS D:\Mandip NM Lab>
```

```
PS D:\Mandip NM Lab> py .\gauss_jordan.py
Enter the number of variables: 3
Enter row 1 coefficients (space-separated): 1 2 3
Enter row 2 coefficients (space-separated): 0 1 4
Enter row 3 coefficients (space-separated): 5 6 0
Enter the constants (space-separated): 9 7 5

Solution:
x1 = -65.000000
x2 = 55.000000
x3 = -12.000000
PS D:\Mandip NM Lab>
```

```
PS D:\Mandip NM Lab> py .\matrix_inversion.py
Enter the number of rows/columns of the matrix: 3
Enter row 1 (space-separated values): 1 2 3
Enter row 2 (space-separated values): 0 1 4
Enter row 3 (space-separated values): 5 6 0

Inverse Matrix:
-24.000000 18.000000 5.000000
20.000000 -15.000000 -4.000000
-5.000000 4.000000 1.000000
PS D:\Mandip NM Lab>
```

```
PS D:\Mandip NM Lab> py .\matrix_LU_factorization.py
Enter matrix size: 3
Enter row 1: 4 3 2
Enter row 2: 3 6 3
Enter row 3: 2 3 5

L Matrix:
1.000000 0.000000 0.000000
0.750000 1.000000 0.000000
0.500000 0.400000 1.000000

U Matrix:
4.000000 3.000000 2.000000
0.000000 3.750000 1.500000
0.000000 0.000000 3.400000
PS D:\Mandip NM Lab>
```

```
PS D:\Mandip NM Lab> py .\jacobi_iteration.py
Enter matrix size: 3
Enter row 1: 4 -1 0
Enter row 2: -1 4 -1
Enter row 3: 0 -1 4
Enter RHS values: 15 10 10

Solution:
x1 = 4.910714
x2 = 4.642857
x3 = 3.660714
Converged in 16 iterations.
PS D:\Mandip NM Lab>
```

```
PS D:\Mandip NM Lab> py .\gauss_seidel.py
Enter matrix size: 3
Enter row 1: 4 -1 0
Enter row 2: -1 4 -1
Enter row 3: 0 -1 4
Enter RHS values: 15 10 10

Solution:
x1 = 4.910714
x2 = 4.642857
x3 = 3.660714
Converged in 10 iterations.
PS D:\Mandip NM Lab>
```

```
PS D:\Mandip NM Lab> py .\forward_Backward_diff.py
Enter x values: 1 2 3 4 5
Enter y values: 2 4 6 8 10

Forward Differences:
f'(1.0) ≈ 2.000000
f'(2.0) ≈ 2.000000
f'(3.0) ≈ 2.000000
f'(4.0) ≈ 2.000000

Backward Differences:
f'(2.0) ≈ 2.000000
f'(3.0) ≈ 2.000000
f'(4.0) ≈ 2.000000
f'(5.0) ≈ 2.000000
PS D:\Mandip NM Lab>
```

```
PS D:\Mandip NM Lab> py .\three_point.py
Enter the value at which derivative is required: 3
Enter increment h: 0.001
Value of Derivative = 23.999999999997357
PS D:\Mandip NM Lab>
```

```
PS D:\Mandip NM Lab> py .\derivative_newton_forward.py

Enter the number of points: 5
Enter values of x and f(x):
0 1
1 2
2 4
3 8
4 16
Enter the value at which derivative is needed: 2.5
Value of First Derivative = 3.057292
PS D:\Mandip NM Lab>
```

```
PS D:\Mandip NM Lab> py .\derivative_newton_backward.p
y
Enter the number of points: 5
Enter values of x and f(x):
0 1
1 2
2 4
3 8
4 16
Enter the value at which derivative is needed: 2.5
Value of First Derivative = 0.265625
PS D:\Mandip NM Lab>
```

```
Value of First Derivative = 0.265625
PS D:\Mandip NM Lab> py .\trapozoidal.py
Enter Lower and Upper Limit: 0 2
Value of Integration: 16.0
PS D:\Mandip NM Lab>
```

```
PS D:\Mandip NM Lab> py .\composite_trapozoidal.py
Enter Lower and Upper Limit: 0 2
Enter Number of Segments: 4
Value of Integration: 15.75
PS D:\Mandip NM Lab>
```

```
PS D:\Mandip NM Lab\composite_simpson1> py .\3.py
Enter Lower and Upper Limit (in radians): 0 2
Enter number of intervals (even number): 4
Value of Integration: 0.9096228049035733
PS D:\Mandip NM Lab\composite simpson1>
```

UTF-8 CRLF {} Python 3.13.1 Go Live Windsurf:

```
PS D:\Mandip NM Lab> py '.\simpson''s_1.py'
Enter Lower Limit (in radians): 0
Enter Upper Limit (in radians): 1.56
Value of Integration: 1.002157
PS D:\Mandip NM Lab>
```

```
PS D:\Mandip NM Lab> py .\simpson_3.py
Enter Lower and Upper Limit (in radians): 0 3.14
Enter number of intervals (multiple of 3): 6
Value of Integration: 0.0015942500550911196
PS D:\Mandip NM Lab>
```

```
PS D:\Mandip NM Lab> py .\Romberg_integration.py
Enter Lower Limit: 1
Enter Upper Limit: 5
Enter p (row index) of required T(p,q): 2
Enter q (column index) of required T(p,q): 1

Romberg Estimate of Integration = 1.622222
PS D:\Mandip NM Lab>
```

```
PS D:\Mandip NM Lab> py .\taylor.py
Enter initial value of x: 1
Enter initial value of y: 2
Enter x at which function is to be evaluated: 2

Function Value at x = 2.000000 is 41.333333
PS D:\Mandip NM Lab>
```

```
PS D:\Mandip NM Lab> py .\picard_method.py
Enter initial value of x: 0
Enter initial value of y: 1
Enter x at which function is to be evaluated: 2

Function Value at x = 2.000000 is 17.909297
PS D:\Mandip NM Lab>
```

```
PS D:\Mandip NM Lab> py .\heun.py
Enter initial value of x: 1
Enter initial value of y: 1
Enter x at which function is to be evaluated: 2
Enter the step size: 0.1
Function Value at x = 2.000000 is 3.986676
PS D:\Mandip NM Lab>
```

```
PS D:\Mandip NM Lab> py .\Rk_method.py
Enter initial value of x (x0): 1
Enter initial value of y (y0): 2
Enter x at which function is to be evaluated (xp): 2
Enter step size (h): 0.1
Function value at x = 2.00 is y = 10.3097
PS D:\Mandip NM Lab>
```

```
PS D:\Mandip NM Lab> py .\laplace.py
Enter the dimension of plate (n,n): 4
Enter temperatures at left, right, top, and bottom: 100 50 75 25
Enter accuracy limit: 0.001
Solution:
[82.86790795612319, 74.10224259581327, 68.42042510702204, 62.41336381062865]
[82.43557592914661, 70.22772765706463, 62.273183511570096, 56.299214294860846]
[76.75375844035537, 62.273183511570096, 54.31863936607556, 50.61739680606962]
[62.41336381062865, 47.96588096152751, 42.284063472736285, 41.958819665134115]
PS D:\Mandip NM Lab>
```

```
PS D:\Mandip NM Lab> py .\poison.py
Enter the dimension of plate (n,n): 4
Enter temperatures at left, right, top, and bottom: 100 50 75 25
Enter accuracy limit: 0.001
Converged after 28 iterations

Temperature Distribution:
 82.8679  74.1022  68.4204  62.4134
 82.4356  70.2277  62.2732  56.2992
 76.7538  62.2732  54.3186  50.6174
 62.4134  47.9659  42.2841  41.9588
PS D:\Mandip NM Lab>
```

```
PS D:\Mandip NM Lab> py .\shooting.py
Enter Boundary Conditions (xa): 0
Enter ya: 0
Enter xb: 1
Enter yb: 2
Enter x at which value is required (xp): 0.5
Enter step size (h): 0.1
Enter accuracy limit (E): 0.001
Initial g1 = 2.0000
```