1. **BISECTION**

```
def func(x):
    return x**2 - 4

def bisection(a, b, tol):
    while (b - a) / 2 > tol:
        c = (a + b) / 2
        if func(c) == 0:
            return c
              elif func(c) * func(a) < 0:
                  b = c
              else:
                  a = c
          return (a + b) / 2

      a, b, tol = float(input("Enter lower bound (a): ")), float(input("Enter upper bound (b): ")), float(input("Enter tolerance: "))
      root = bisection(a, b, tol)
      print(f"Root: {root:.6f}")
```

Soln=2

2. NEWTON RAPHSON

```python
def func(x):
    return x**2 - 4

def derivative(x):
    return 2 * x

def newton_method(x, tolerance, max_iterations):
        for _ in range(max_iterations):
            x = x - func(x) / derivative(x)

            if abs(func(x)) < tolerance:
                return x

        return None

    x, tolerance, max_iterations = 2.0, 0.0001, 100

    root = newton_method(x, tolerance, max_iterations)

    if root is not None:
        print(f"Root: {root:.6f}")
    else:
        print("Newton's method did not converge.")
```

3. FALSE POSITION

```python
def func(x):
    return x**2 - 4

def false_position_method(a, b, tolerance, max_iterations):
    if func(a) * func(b) >= 0:
        return None

          for _ in range(max_iterations):
              c = (a * func(b) - b * func(a)) / (func(b) - func(a))

              if abs(func(c)) < tolerance:
                  return c

              if func(c) * func(a) < 0:
                  b = c
              else:
                  a = c

          return None

      a, b, tolerance, max_iterations = 1.0, 3.0, 0.0001, 100

      root = false_position_method(a, b, tolerance, max_iterations)

      if root is not None:
          print(f"Root: {root:.6f}")
```

4. GAUSS ELIMINATION

```
def gauss_elimination(matrix):
    n = len(matrix)

    for i in range(n):
        max_row = i
        for j in range(i + 1, n):
            if abs(matrix[j][i]) > abs(matrix[max_row][i]):
                max_row = j
        matrix[i], matrix[max_row] = matrix[max_row], matrix[i]

        for j in range(i + 1, n):
            factor = matrix[j][i] / matrix[i][i]
            for k in range(i, n + 1):
                matrix[j][k] -= factor * matrix[i][k]

    x = [0] * n
    for i in range(n - 1, -1, -1):
        x[i] = matrix[i][n] / matrix[i][i]
        for j in range(i - 1, -1, -1):
            matrix[j][n] -= matrix[j][i] * x[i]

    return x

matrix = [
    [2, 1, 5],
    [1, -1, 1]
]

solution = gauss_elimination(matrix)

for sol in solution:
    print(f"{sol:.6f}")
```

Soln= (2,1)

5. **GAUSS SIEDEL**

```
def gauss_seidel(A, b, x0, max_iterations, tolerance):
    n = len(A)
    x = x0.copy()

    for _ in range(max_iterations):
        for i in range(n):
            x[i] = (b[i] - sum(A[i][j] * x[j] for j in range(n) if j !=
i)) / A[i][i]

        if all(abs(x[i] - x0[i]) < tolerance for i in range(n)):
            return x

        x0 = x.copy()

    return None

A = [
    [3, 1],
    [-1, 4]
]
b = [9, 7]
x0 = [0, 0]
max_iterations = 100
tolerance = 0.0001

solution = gauss_seidel(A, b, x0, max_iterations, tolerance)

if solution is not None:
    print("Solution:", solution)
```

Soln= (2.23,2.3)

6. **GAUSS JORDAN**

```
def gauss_jordan(matrix):
    n = len(matrix)

    for i in range(n):
        pivot = matrix[i][i]

        for j in range(i, n + 1):
            matrix[i][j] /= pivot

        for k in range(n):
            if k != i:
                factor = matrix[k][i]
                for j in range(i, n + 1):
                    matrix[k][j] -= factor * matrix[i][j]

    solutions = [row[-1] for row in matrix]
    return solutions

# Example system of equations:
# 2x + y = 5
# x - y = 1

matrix = [
    [2, 1, 5],
    [1, -1, 1]
]

solutions = gauss_jordan(matrix)

for i, sol in enumerate(solutions):
    print(f"x{i + 1} = {sol:.6f}")
```

Soln=(2,1)

7. **NEWTON FORWARD INTERPOLATION METHOD**

```
def newton_forward_interpolation(x, x_values, y_values):

    n = len(x_values)
    h = x_values[1] - x_values[0]

    forward_diff = [y_values]

    for j in range(1, n):
        diff_row = [forward_diff[j - 1][i + 1] - forward_diff[j - 1][i]
for i in range(n - j)]
        forward_diff.append(diff_row)

    u = (x - x_values[0]) / h
    estimated_value = sum((u ** i) * (forward_diff[i][0] /
math.factorial(i)) for i in range(n))

    return estimated_value

import math

x_values = [0, 1, 2, 3]
y_values = [1, 2, 8, 18]
x = 1.5

estimated_value = newton_forward_interpolation(x, x_values, y_values)
print(f"Estimated value at x = {x}: {estimated_value:.6f}")
```

Soln= 7.56

OR

```python
import math

def newton_forward_interpolation(x, x_values, y_values):
    n = len(x_values)
    h = x_values[1] - x_values[0]

    forward_diff = [y_values]

    for j in range(1, n):
        diff_row = [forward_diff[j - 1][i + 1] - forward_diff[j - 1][i] for i in range(n - j)]
        forward_diff.append(diff_row)

    u = (x - x_values[0]) / h
    estimated_value = sum((u ** i) * (forward_diff[i][0] / math.factorial(i)) for i in range(n))

    return estimated_value

x_values = [0, 1, 2, 3]
y_values = [1, 2, 8, 18]
x = 1.5

estimated_value = newton_forward_interpolation(x, x_values, y_values)
print(f"Estimated value at x = {x}: {estimated_value:.6f}")
```

```
Estimated value at x = 1.5: 7.562500
```

8. LAGRANGE METHOD

```python
def lagrange_interpolation(x, x_values, y_values):
    n = len(x_values)
    estimated_value = 0

    for i in range(n):
        term = y_values[i]
        for j in range(n):
            if j != i:
                term *= (x - x_values[j]) / (x_values[i] - x_values[j])
        estimated_value = estimated_value + term

    return estimated_value

x_values = [0, 1, 2, 3]
y_values = [1, 2, 8, 18]
x = 1.5

estimated_value = lagrange_interpolation(x, x_values, y_values)
print(f"Estimated value at x = {x}: {estimated_value:.6f}")
```

```
Estimated value at x = 1.5: 4.437500
```