Assignment 5

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import CubicSpline


def forward_difference_table(x, y):
    """Calculate forward difference table"""
    n = len(y)
    table = np.zeros((n, n))
    table[:, 0] = y

    for j in range(1, n):
        for i in range(n - j):
            table[i, j] = table[i + 1, j - 1] - table[i, j - 1]

    return table


def backward_difference_table(x, y):
    """Calculate backward difference table"""
    n = len(y)
    table = np.zeros((n, n))
    table[:, 0] = y

    for j in range(1, n):
        for i in range(j, n):
            table[i, j] = table[i, j - 1] - table[i - 1, j - 1]
```

```python
    return table

def newton_forward(x, y, x_interp):
    """Newton's Forward Interpolation"""
    h = x[1] - x[0]
    u = (x_interp - x[0]) / h

    diff_table = forward_difference_table(x, y)
    n = len(x)

    result = y[0]
    u_term = 1
    factorial = 1

    for i in range(1, n):
        u_term *= (u - i + 1)
        factorial *= i
        result += (u_term * diff_table[0, i]) / factorial

    return result

def newton_backward(x, y, x_interp):
    """Newton's Backward Interpolation"""
    h = x[1] - x[0]
    u = (x_interp - x[-1]) / h

    diff_table = backward_difference_table(x, y)
    n = len(x)
```

```
    result = y[-1]
    u_term = 1
    factorial = 1

    for i in range(1, n):
        u_term *= (u + i - 1)
        factorial *= i
        result += (u_term * diff_table[-1, i]) / factorial

    return result

def lagrange(x, y, x_interp):
    """Lagrange Interpolation"""
    n = len(x)
    result = 0

    for i in range(n):
        term = y[i]
        for j in range(n):
            if i != j:
                term *= (x_interp - x[j]) / (x[i] - x[j])
        result += term

    return result

def divided_difference(x, y):
    """Calculate divided differences"""
```

```
    n = len(x)

    coef = np.zeros((n, n))

    coef[:,0] = y


    for j in range(1, n):

        for i in range(n - j):

            coef[i,j] = (coef[i+1,j-1] - coef[i,j-1]) / (x[i+j] - x[i])


    return coef


def newton_divided_diff(x, y, x_interp):

    """Newton's Divided Difference Interpolation"""

    coef = divided_difference(x, y)

    n = len(x)

    result = coef[0,0]


    for i in range(1, n):

        term = coef[0,i]

        for j in range(i):

            term *= (x_interp - x[j])

        result += term


    return result


def plot_interpolation(x, y, x_interp, y_interp, title):

    """Plot interpolation results"""

    plt.figure(figsize=(10, 6))

    plt.scatter(x, y, color='blue', label='Data points')
```

```
    plt.scatter(x_interp, y_interp, color='red', label='Interpolated point')

    # Create smooth curve for visualization
    x_smooth = np.linspace(min(x), max(x), 100)
    if len(x) > 2:  # Only create smooth curve if we have enough points
        cs = CubicSpline(x, y)
        plt.plot(x_smooth, cs(x_smooth), 'g--', label='Smooth curve')

    plt.xlabel('x')
    plt.ylabel('y')
    plt.title(title)
    plt.legend()
    plt.grid(True)
    plt.show()

# Task 1: Newton's Forward Interpolation
print("Task 1: Newton's Forward Interpolation")
x1 = np.array([0, 1, 2, 3, 4])
y1 = np.array([1, 2.7, 5.8, 10.4, 16.5])
x1_interp = 2.5
y1_interp = newton_forward(x1, y1, x1_interp)
print(f"f({x1_interp}) ≈ {y1_interp:.4f}")
plot_interpolation(x1, y1, x1_interp, y1_interp, "Newton's Forward Interpolation")

# Task 2: Newton's Backward Interpolation
print("\nTask 2: Newton's Backward Interpolation")
x2 = np.array([3, 4, 5, 6, 7])
y2 = np.array([2.2, 3.5, 5.1, 7.3, 10.0])
```

```
x2_interp = 5.5

y2_interp = newton_backward(x2, y2, x2_interp)

print(f"f({x2_interp}) ≈ {y2_interp:.4f}")

plot_interpolation(x2, y2, x2_interp, y2_interp, "Newton's Backward Interpolation")


# Task 3: Central Difference Interpolation

print("\nTask 3: Central Difference Interpolation")

x3 = np.array([10, 12, 14, 16, 18])

y3 = np.array([100, 144, 196, 256, 324])

x3_interp = 13

cs3 = CubicSpline(x3, y3)

y3_interp = cs3(x3_interp)

print(f"f({x3_interp}) ≈ {y3_interp:.4f}")

plot_interpolation(x3, y3, x3_interp, y3_interp, "Central Difference Interpolation")


# Task 4: Lagrange Interpolation

print("\nTask 4: Lagrange Interpolation")

x4 = np.array([2, 5, 8, 10])

y4 = np.array([1.4, 2.3, 3.8, 4.6])

x4_interp = 6

y4_interp = lagrange(x4, y4, x4_interp)

print(f"f({x4_interp}) ≈ {y4_interp:.4f}")

plot_interpolation(x4, y4, x4_interp, y4_interp, "Lagrange Interpolation")


# Task 5: Newton's Divided Difference

print("\nTask 5: Newton's Divided Difference")

x5 = np.array([0, 2, 5, 8])

y5 = np.array([4, 8, 14, 25])
```

```
x5_interp = 3

y5_interp = newton_divided_diff(x5, y5, x5_interp)

print(f"f({x5_interp}) ≈ {y5_interp:.4f}")

plot_interpolation(x5, y5, x5_interp, y5_interp, "Newton's Divided Difference")


# Task 6: Cubic Spline Interpolation

print("\nTask 6: Cubic Spline Interpolation")

x6 = np.array([1, 2, 3, 4, 5])

y6 = np.array([2.3, 3.1, 4.9, 6.5, 8.1])

x6_interp = np.array([2.5, 4.3])

cs6 = CubicSpline(x6, y6)

y6_interp = cs6(x6_interp)

print(f"f(2.5) ≈ {y6_interp[0]:.4f}")

print(f"f(4.3) ≈ {y6_interp[1]:.4f}")

plot_interpolation(x6, y6, x6_interp, y6_interp, "Cubic Spline Interpolation")
```