LAB-6

Newton and Lagrange's Polynomial Interpolation

Rakshit Pandhi - 202201426

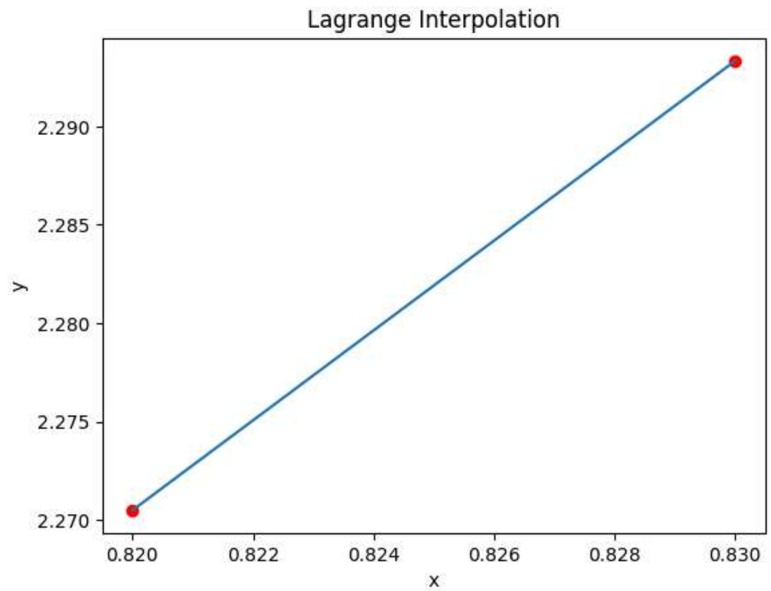
Kalp Shah - 202201457

Lagrange Interpolation

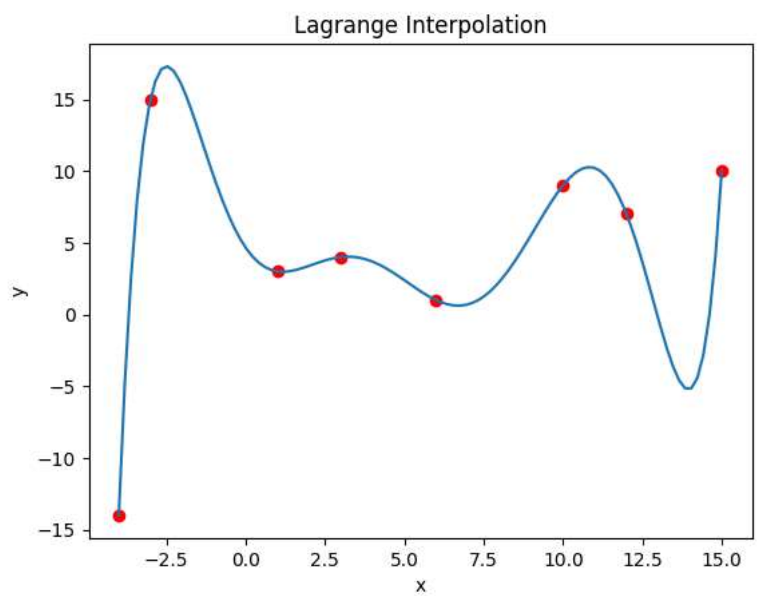
```
import matplotlib.pyplot as plt
import numpy as np


def mypolyint(data):
  n = len(data)
  x_values = [point[0] for point in data]
  y_values = [point[1] for point in data]
  coeffs = [0] * n
  for i in range(n):
    p = [1]
    for j in range(n):
      if i != j:
        p = poly_mul(p, [-x_values[j] / (x_values[i] - x_values[j]), 1 / (x_values[i] - x_values[j])])
    coeffs = poly_add(coeffs, poly_mul_scalar(p, y_values[i]))

  equation = "P(x) = "
  for i, c in enumerate(coeffs):
    if i > 0:
      equation += " + "
    equation += f"{c:.5f}x^{i}"
  print(equation)
  print("Coefficients:", coeffs)


  x_range=np.linspace(min(x_values), max(x_values), 100)
  y_interp = [lagrange_interpolation(x_val, x_values, y_values) for x_val in x_range]
  plt.plot(x_range, y_interp)
  plt.scatter(x_values, y_values, color='red')
  plt.title('Lagrange Interpolation')
  plt.xlabel('x')
  plt.ylabel('y')
  plt.show()

  return coeffs

def lagrange_interpolation(x, x_values, y_values):
  n = len(x_values)
  interpolated_y = 0
  for i in range(n):
    L_i = 1
    for j in range(n):
      if i != j:
        L_i *= (x - x_values[j]) / (x_values[i] - x_values[j])
    interpolated_y += y_values[i] * L_i
  return interpolated_y


def poly_mul(p1, p2):
  n1 = len(p1)
  n2 = len(p2)
  result = [0] * (n1 + n2 - 1)
  for i in range(n1):
    for j in range(n2):
      result[i + j] += p1[i] * p2[j]
  return result

def poly_add(p1, p2):
  n1 = len(p1)
  n2 = len(p2)
  result = [0] * max(n1, n2)
  for i in range(n1):
```

```
    for i in range(n1):
      result[i] += p1[i]
    for i in range(n2):
      result[i] += p2[i]
    return result

def poly_mul_scalar(p, s):
    return [c * s for c in p]

data1 = [[0.82, 2.2705], [0.83,2.293319]]
coeffs1 = mypolyint(data1)

data2 = [[1,3], [3,4], [6,1], [-3, 15], [-4, -14], [15, 10], [10, 9], [12, 7]]
coeffs2 = mypolyint(data2)
```

```
P(x) = 0.39934x^0 + 2.28190x^1
Coefficients: [0.39934200000004694, 2.281899999999979]
```




```
P(x) = 4.65535x^0 + -3.23295x^1 + 1.74638x^2 + -0.07491x^3 + -0.11718x^4 + 0.02516x^5 + -0.00189x^6 + 0.00005x^7
Coefficients: [4.655345698578781, -3.232948160861696, 1.746375860053805, -0.07490930615147398, -0.11718305344314814, 0.02516454582835813
```



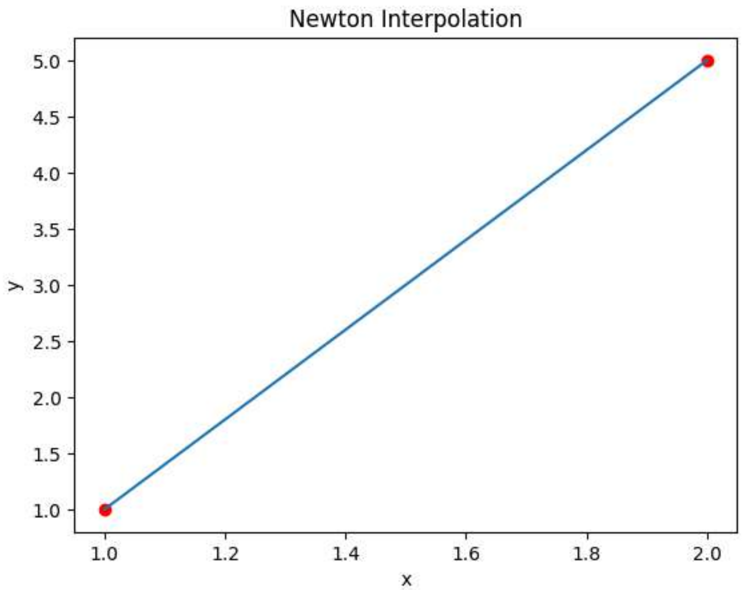
Newton Interpolation

```
import numpy as np
import matplotlib.pyplot as plt

def mynewtonint(data):
    n = len(data)
    x_values = [point[0] for point in data]
```

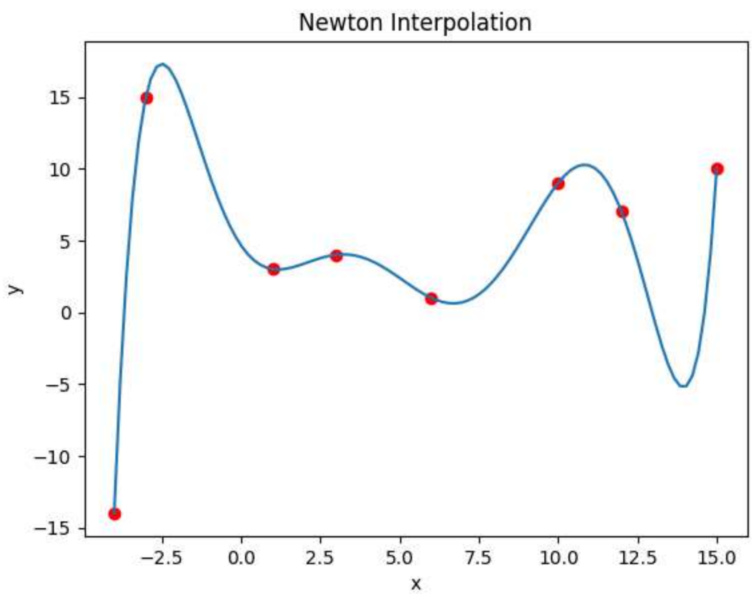
```
    y_values = [point[1] for point in data]

    # Calculate divided differences
    f = np.zeros((n, n))
    f[:,0] = y_values
    for j in range(1,n):
      for i in range(n-j):
        f[i][j] = (f[i+1][j-1] - f[i][j-1]) / (x_values[i+j]-x_values[i])

    # Extract coefficients
    coeffs = f[0,:]

    # Print the polynomial equation
    equation = "P(x) = "
    for i, c in enumerate(coeffs):
      if i > 0:
        equation += " + "
      equation += f"{c:.2f}"
      for j in range(i):
        equation += f"(x - {x_values[j]:.5f})"
    print(equation)

    # Expand the polynomial and collect coefficients
    expanded_coeffs = expand_polynomial(coeffs, x_values)

    # Print the expanded polynomial equation
    expanded_equation = "P(x) = "
    for i, c in enumerate(expanded_coeffs):
      if i > 0:
        if c >= 0:
          expanded_equation += " + "
        else:
          expanded_equation += " - "
          c = -c
      expanded_equation += f"{c:.5f}x^{i}"
    print(expanded_equation)

    # Generate points for plotting
    x_range = np.linspace(min(x_values), max(x_values), 100)
    y_interp = [newton_interpolation(x_val, x_values, coeffs) for x_val in x_range]

    # Plot the polynomial and the data points
    plt.plot(x_range, y_interp)
    plt.scatter(x_values, y_values, color='red')
    plt.title('Newton Interpolation')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.show()

    return coeffs

  def newton_interpolation(x, x_values, coeffs):
    n = len(x_values)
    result = coeffs[n-1]
    for i in range(n-2, -1, -1):
      result = result * (x - x_values[i]) + coeffs[i]
    return result

  def expand_polynomial(coeffs, x_values):
    n = len(coeffs)
    expanded_coeffs = [0] * n
    for i in range(n):
      term = [coeffs[i]]
      for j in range(i):
        term = poly_mul(term, [-x_values[j], 1])
      expanded_coeffs = poly_add(expanded_coeffs, term)
    return expanded_coeffs

  def poly_mul(p1, p2):
    n1 = len(p1)
    n2 = len(p2)
    result = [0] * (n1 + n2 - 1)
    for i in range(n1):
      for j in range(n2):
        result[i + j] += p1[i] * p2[j]
    return result
```

```
def poly_add(p1, p2):
  n1 = len(p1)
  n2 = len(p2)
  result = [0] * max(n1, n2)
  for i in range(n1):
    result[i] += p1[i]
  for i in range(n2):
    result[i] += p2[i]
  return result

data1 = [[1, 1], [2,5]]
coeffs1 = mynewtonint(data1)

data2 = [[1,3], [3,4], [6,1], [-3, 15], [-4, -14], [15, 10], [10, 9], [12, 7]]
coeffs2 = mynewtonint(data2)
```

```
P(x) = 1.00 + 4.00(x - 1.00000)
P(x) = -3.00000x^0 + 4.00000x^1
```



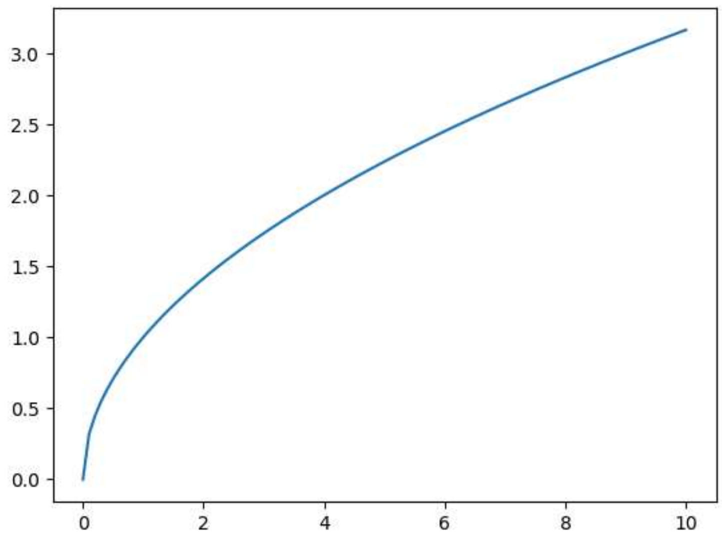

```
P(x) = 3.00 + 0.50(x - 1.00000) + -0.30(x - 1.00000)(x - 3.00000) + -0.10(x - 1.00000)(x - 3.00000)(x - 6.00000) + -0.11(x - 1.00000)
P(x) = 4.65535x^0 - 3.23295x^1 + 1.74638x^2 - 0.07491x^3 - 0.11718x^4 + 0.02516x^5 - 0.00189x^6 + 0.00005x^7
```



Newton Interpolation

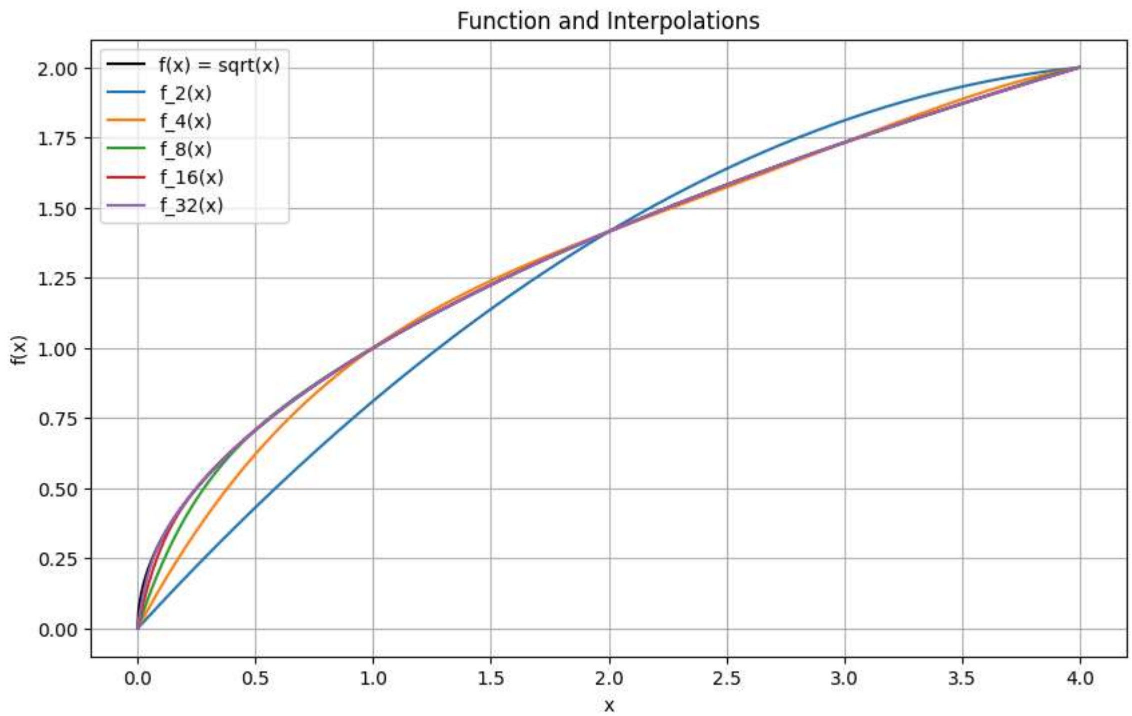
```
import numpy as np
import matplotlib.pyplot as plt

def function():
```

```
  x=np.linspace(0,10,100)
  y=(x)**(0.5)
  plt.plot(x, y)
  plt.show()

function()
start = 0
end = 4
num_points = 3

points = np.linspace(start, end, num_points)

print(points)
data1=[]
```

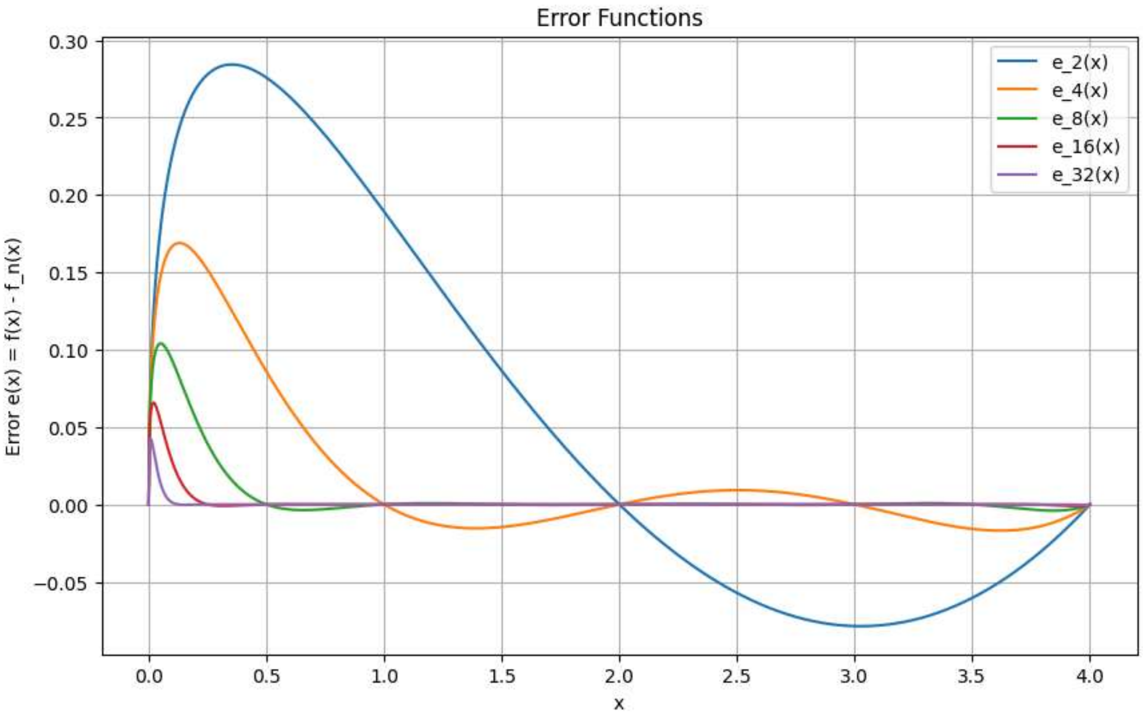


```
import numpy as np
import matplotlib.pyplot as plt


def f(x):
    return np.sqrt(x)


def newton_coefficients(x, y):
    n = len(x)
    coef = np.copy(y)
    for j in range(1, n):
        for i in range(n-1, j-1, -1):
            coef[i] = (coef[i] - coef[i-1]) / (x[i] - x[i-j])
    return coef


def newton_poly(x_vals, x, coef):
    n = len(coef)
    p = coef[-1]
    for i in range(n-2, -1, -1):
        p = p * (x_vals - x[i]) + coef[i]
    return p

x_interval = np.linspace(0, 4, 1000)
y_actual = f(x_interval)

n_values = [2, 4, 8, 16, 32]

max_errors = []

plt.figure(figsize=(10, 6))
plt.plot(x_interval, y_actual, label='f(x) = sqrt(x)', color='black')


for n in n_values:

    x_samples = np.linspace(0, 4, n+1)
```

```
        y_samples = f(x_samples)


        coef = newton_coefficients(x_samples, y_samples)


        y_interp = newton_poly(x_interval, x_samples, coef)

        plt.plot(x_interval, y_interp, label=f'f_{n}(x)')

        error = np.abs(y_actual - y_interp)
        max_errors.append(np.max(error))

    plt.title("Function and Interpolations")
    plt.xlabel("x")
    plt.ylabel("f(x)")
    plt.legend()
    plt.grid(True)
    plt.show()
```




```
    plt.figure(figsize=(10, 6))
    for n, max_error in zip(n_values, max_errors):
        x_samples = np.linspace(0, 4, n+1)
        y_samples = f(x_samples)
        coef = newton_coefficients(x_samples, y_samples)
        y_interp = newton_poly(x_interval, x_samples, coef)

        # Plot error
        error = y_actual - y_interp
        plt.plot(x_interval, error, label=f'e_{n}(x)')

    plt.title("Error Functions")
    plt.xlabel("x")
    plt.ylabel("Error e(x) = f(x) - f_n(x)")
    plt.legend()
    plt.grid(True)
    plt.show()
```




```
plt.figure(figsize=(10, 6))
plt.plot(n_values, max_errors, marker='o')
plt.title("Maximum Error vs n")
plt.xlabel("n")
plt.ylabel("Maximum Error E_n")
plt.grid(True)
plt.show()
```