INTERPOLATION IN LAGRANGE AND NEWTON DIVIDED DIFFERENCE

**Introduction:**

The purpose of this project is to implement interpolation methods, specifically Lagrange interpolation and Newton divided difference interpolation. Interpolation is a mathematical technique used to estimate values between known data points. In this project, the user can input data points and select an interpolation method to calculate interpolated values. The project also includes a graphical representation of the interpolation results.

**Methodology:**

The project utilizes two interpolation methods :

a. Lagrange Interpolation: It uses the Lagrange polynomial to approximate the function that passes through the given data points.

b. Newton Divided Difference Interpolation: It employs divided differences to construct the interpolating polynomial.

**Implementation:**

The project is implemented using the following components:a. Libraries: The numpy library is used for mathematical computations, matplotlib.pyplot library for plotting graphs, and tkinter library for creating the graphical user interface (GUI).

Functions:

lagrange\_interpolation: Implements the Lagrange interpolation method.

divided\_difference: Computes the divided difference coefficients for the Newton interpolation method.

newton\_interpolation: Implements the Newton divided difference interpolation method.

calculate\_interpolation: Reads input data points and performs interpolation based on the selected method.

plot\_graph: Plots the data points and the interpolated curve.

**Results:**

Upon running the program, the user is prompted to input x and y data points as well as x values to interpolate. The user can select the interpolation method, either Lagrange or Newton divided difference. After clicking the "Calculate" button, the program calculates the interpolated values and displays them in a message box. Additionally, a graph is generated showing the data points and the interpolated curve.

**Conclusion:**

The project successfully implements Lagrange interpolation and Newton divided difference interpolation methods. It provides a user-friendly interface for inputting data points and selecting the interpolation method. The program accurately calculates the interpolated values and presents them to the user. The graphical representation helps visualize the interpolation results.

**Future Enhancements:**

There are several potential improvements and additional features that can be incorporated into the project, such as:

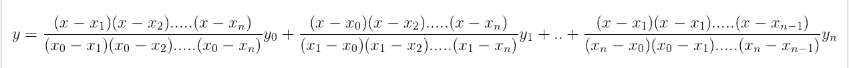
Adding support for more interpolation methods, such as cubic spline interpolation or polynomial interpolation.

Allowing the user to customize the plot's appearance, including colors, markers, and line styles.

Implementing error estimation techniques to assess the accuracy of the interpolation results.

Saving the generated graphs as image files for further analysis or documentation.

**FORMULA OF LAGRANGE:**



**FORMULA OF NEWTON:**

(x) = f [x0] + (x - x0) f [x0, x1] + (x - x0) (x - x1) f [x0, x1, x2] +  (x - x0) (x - x1) (x - x2)f [x0, x1,  x2, x3]

CODE

importnumpy asnp

importmatplotlib.pyplot asplt

importtkinter astk

fromtkinter importmessagebox, ttk

deflagrange\_interpolation(x, y, x\_interp):

n =l=len(x)

y\_interp =[=[]

foforxi inx\_interp:

yi =0.0=0.0

forjforj inrange(n):

term =1.0 =1.0

fork ifork inrange(n):

ifk !=j:ifk !=j:

term \*=(xi -x[k\*=(xi -x[k]) /(x[j] -x[k])

yi += y[j] \* term 畣慬

y\_interp.append(yi) 张츀

return y\_interp ⸱

def divided\_difference(x, y):

n = len(x) )

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

coefficients[:, 0] = y

for j in range(1, n):

for i in range(n - j): 檘᪣

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

return coefficients[0] 

def newton\_interpolation(x, y, x\_interp):

coefficients = divided\_difference(x, y)

n = len(x) 츫

y\_interp = [] 3

for xi in x\_interp:

yi = coefficients[n - 1] 孋܂

for k in range(n - 2, -1, -1):

yi = coefficients[k] + (xi - x[k]) \* yi

y\_interp.append(yi) 倮

return y\_interp

def calculate\_interpolation():

x\_pts = [float(x) for x in entry\_x\_pts.get().split(',')] 䥕

y\_pts = [float(y) for y in entry\_y\_pts.get().split(',')] 䞴

x\_vals = [float(x) for x in entry\_x\_vals.get().split(',')] 䌤

if selected\_method.get() == 'Lagrange':

y\_interp = lagrange\_interpolation(x\_pts, y\_pts, x\_vals) 䋻ᶼ塅

else:

y\_interp = newton\_interpolation(x\_pts, y\_pts, x\_vals)

messagebox.showinfo("Interpolation Result", "Interpolated Values:\n{}".format(y\_interp))

plot\_graph(x\_pts, y\_pts, x\_vals, y\_interp)

def plot\_graph(x\_pts, y\_pts, x\_interp, y\_interp):

plt.figure(figsize=(8, 6)) 慬

plt.plot(x\_pts, y\_pts, 'ro', label='Data Points')

# Generate a smooth line for interpolation t

x\_smooth = np.linspace(min(x\_pts), max(x\_pts), 100)

if selected\_method.get() == 'Lagrange':

y\_smooth = lagrange\_interpolation(x\_pts, y\_pts, x\_smooth)

else: 媃

y\_smooth = newton\_interpolation(x\_pts, y\_pts, x\_smooth) 䀈Ნ夥

plt.plot(x\_smooth, y\_smooth, 'b-', label='Interpolation')

plt.scatter(x\_pts, y\_pts, color='red')

plt.xlabel("X")

plt.ylabel("Y")

plt.title("Interpolation")

plt.legend() 츧

plt.grid(True) 2

plt.tight\_layout()

plt.show() 츭

window = tk.Tk()

window.title("Interpolation")

window.geometry("400x300")

label\_x\_pts = tk.Label(window, text="Enter x data points:")

label\_x\_pts.pack()

entry\_x\_pts = tk.Entry(window)

entry\_x\_pts.pack()

label\_y\_pts = tk.Label(window, text="Enter y data points:")

label\_y\_pts.pack()

entry\_y\_pts = tk.Entry(window)

entry\_y\_pts.pack()

label\_x\_vals = tk.Label(window, text="Enter x values to interpolate:")

label\_x\_vals.pack()

entry\_x\_vals = tk.Entry(window)

entry\_x\_vals.pack()

label\_method = tk.Label(window, text="Select interpolation method:")

label\_method.pack()

selected\_method = tk.StringVar()

selected\_method.set("Lagrange")

radio\_lagrange = tk.Radiobutton(window, text="Lagrange", variable=selected\_method, value="Lagrange")

radio\_lagrange.pack()

radio\_newton = tk.Radiobutton(window, text="Newton Divided Difference", variable=selected\_method, value="Newton")

radio\_newton.pack()

button\_calculate = tk.Button(window, text="Calculate", command=calculate\_interpolation)

button\_calculate.pack()

window.mainloop()

OUTPUT

