Interpolation

Interpolation is the process of finding a formula often a polynomial whose graph will pass through a given set of points to represent the value of a function. It has a various number of applications in engineering and science, that are used to construct new data points within the range of a discrete data set of known data points. This method is always needed to compute the value of a function for an intermediate value of the independent function. In short, interpolation is a process of determining the unknown values that lie in between the known data points.

It is mostly used to predict the unknown values for any geographical related data points such as noise level, rainfall, elevation, and so on.

Linear interpolation

It is a method for estimating the value of a function at a point between two known data points (x_0, y_0) and (x_1, y_1) by finding a polynomial that passes through the data points. In this method, the data points are represented as the coefficients of a polynomial equation, which is then used to estimate the value of the function at an unknown point. The formula can written as

$$P_1 = y_0 \left(\frac{x_1 - x}{x_1 - x_0} \right) + y_1 \left(\frac{x - x_0}{x_1 - x_0} \right)$$

.

In this chapter, we shall use Sagemath to compute interpolation with two data point. The code is given in the diagram below

```
def do_linear_interpolation (point_1 , point_2 , point=None):
    # Given two points
    y_0 = point_1[-1]
    y_1 = point_2[-1]
    x_0 = point_1[0]
    x_1 = point_2[0]
    denom_value = x_1 - x_0


    left_equation = getEquation(x=x_1, y= y_0 ,denominator= denom_value)
    right_equation = getEquation(x=x_0, y=y_1, denominator = denom_value, second_point=True )
    if(not point):
        print("The Polynomial of the equation is",left_equation ,"+" , right_equation)
    else:
        print("The Polynomial of the equation is",left_equation ,"+" , right_equation)
        print(f"The value of y at x = {point} is {solveEquation(x_0, x_1, y_0, y_1, a=point)}")  # Print the interpolated value with 13 decimal places



    # according to the formula for interpolation , we obtain the polynomial for the two points


def getEquation(x , y , denominator , second_point = False ):
    value_1 = y * x
    value_2 = str(y) + "x"
    if(second_point):
        numerator_1 = "(" + str(value_2) + "-" + str(value_1) + ")" + "/" + str(denominator)
    else:
        numerator_1 = "(" + str(value_1) + "-" + str(value_2) + ")" + "/" + str(denominator)
    return numerator_1;



def solveEquation(x_0 , x_1 , y_0 , y_1 , a ):
    denominator= (x_1 - x_0)
    right_equation = ((x_1 - a) * y_0)/denominator
    left_equation = ((a - x_0) * y_1)/denominator
    return right_equation + left_equation



do_linear_interpolation( point_1 = [x_0,y_0] , point_2 = [x_1,y_1] , point=x)
```

Figure 1: Sagemath code for Linear Interpolation.

Example 1

For the data points (2, 3) and (5, 7), find $P_1(x)$.
By substituting the values in the code above we obtained $\frac{15-3x}{3} + \frac{7x-14}{3}$

```
def do_linear_interpolation (point_1 , point_2 , point=None):
    # Given two points
    y_0 = point_1[-1]
    y_1 = point_2[-1]
    x_0 = point_1[0]
    x_1 = point_2[0]
    denom_value = x_1 - x_0


    left_equation = getEquation(x=x_1, y= y_0 ,denominator= denom_value)
    right_equation = getEquation(x=x_0, y=y_1, denominator = denom_value, second_point=True )
    if(not point):
        print("The Polynomial of the equation is",left_equation ,"+" , right_equation)
    else:
        print("The Polynomial of the equation is",left_equation ,"+" , right_equation)
        print(f"The value of y at x = {point} is {solveEquation(x_0, x_1, y_0, y_1, a=point)}")  # Print the interpolated value with 13 decimal places



    # according to the formula for interpolation , we obtain the polynomial for the two points


def getEquation(x , y , denominator , second_point = False ):
    value_1 = y * x
    value_2 = str(y) + "x"
    if(second_point):
        numerator_1 = "(" + str(value_2) + "-" + str(value_1) + ")" + "/" + str(denominator)
    else:
        numerator_1 = "(" + str(value_1) + "-" + str(value_2) + ")" + "/" + str(denominator)
    return numerator_1;


def solveEquation(x_0 , x_1 , y_0 , y_1 , a ):
    denominator= (x_1 - x_0)
    right_equation = ((x_1 - a) * y_0)/denominator
    left_equation = ((a - x_0) * y_1)/denominator
    return right_equation + left_equation


do_linear_interpolation( point_1 = [2,3] , point_2 = [5,7])
```

Evaluate

Language: Sage

Share

```
The Polynomial of the equation is (15-3x)/3 + (7x-14)/3
```

Help | Powered by SageMath

Figure 2: Solution to example 1.

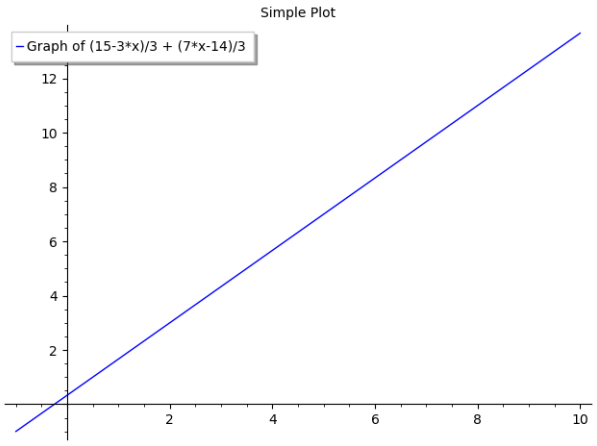

Figure 3: Graph of $\frac{15-3x}{3} + \frac{7x-14}{3}$ by Sagemath.

Example 2

For the data points (0.82, 2.270500) and (0.83, 2.293319, find $P_1(x)$ and evaluate $P_1(0.826)$.
By substituting the points in the Sagemath code above we obtained 2.2841914

```
def do_linear_interpolation (point_1 , point_2 , point=None):
    # Given two points
    y_0 = point_1[-1]
    y_1 = point_2[-1]
    x_0 = point_1[0]
    x_1 = point_2[0]
    denom_value = x_1 - x_0


    left_equation = getEquation(x=x_1, y= y_0 ,denominator= denom_value)
    right_equation = getEquation(x=x_0, y=y_1, denominator = denom_value, second_point=True )
    if(not point):
        print("The Polynomial of the equation is",left_equation ,"+" , right_equation)
    else:
        print("The Polynomial of the equation is",left_equation ,"+" , right_equation)
        print(f"The value of y at x = {point} is {solveEquation(x_0, x_1, y_0, y_1, a=point)}")  # Print the interpolated value with 13 decimal places



    # according to the formula for interpolation , we obtain the polynomial for the two points


def getEquation(x , y , denominator , second_point = False ):
    value_1 = y * x
    value_2 = str(y) + "x"
    if(second_point):
        numerator_1 = "(" + str(value_2) + "-" + str(value_1) + ")" + "/" + str(denominator)
    else:
        numerator_1 = "(" + str(value_1) + "-" + str(value_2) + ")" + "/" + str(denominator)
    return numerator_1;



def solveEquation(x_0 , x_1 , y_0 , y_1 , a ):
    denominator= (x_1 - x_0)
    right_equation = ((x_1 - a) * y_0)/denominator
    left_equation = ((a - x_0) * y_1)/denominator
    return right_equation + left_equation


do_linear_interpolation( point_1 = [0.82, 2.270500] , point_2 = [0.83, 2.293319], point=0.826)
```

Evaluate

Language: Sage

Share

```
The Polynomial of the equation is (1.88451500000000-2.27050000000000x)/0.0100000000000000 + (2.29331900000000x-1.88052158000000)/0.0100000000000000
The value of y at x = 0.826000000000000 is 2.28419140000000
```

Help | Powered by SageMath

Figure 4: Solution to example 2.

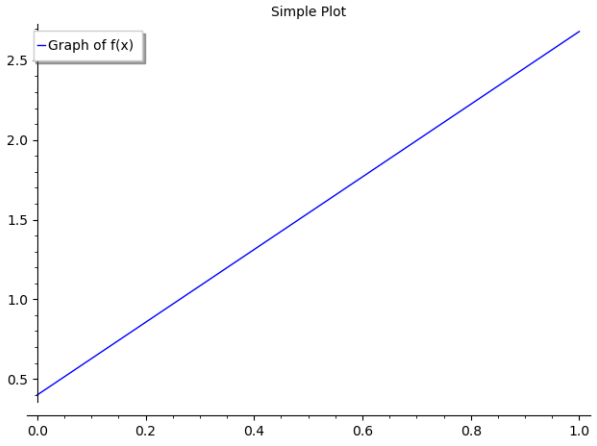

Figure 5: Graph of $P_1(x)$ by Sagemath.

Higher Order Interpolation

Higher-order interpolation, also known as polynomial interpolation, refers to the process of finding a polynomial of a higher degree that passes through a set of data points. While linear interpolation uses first-degree (linear) polynomials, higher-order interpolation uses polynomials of a higher degree (quadratic, cubic, etc.) to represent the data.

The general form of a polynomial for higher-order interpolation can be written as:

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

Where n is the degree of the polynomial and $a_n, a_{n-1}, \cdots, a_0$ are coefficients to be determined based on the given data points.

To perform higher-order interpolation, we typically need to set up a system of equations using the data points and then solve for the coefficients of the polynomial.

Given a set of data points $(x_0, y_0), (x_1, y_1), \cdots, (x_n, y_n)$, where x_i and y_i are the known data points, and a target x-coordinate x at which we want to estimate the value, the solution is given by Lagrange's formula

$$P_n(x) = y_0 L_0(x) + y_1 L_1(x) + y_2 L_2(x) + \cdots + y_n L_n(x)$$

where the Lagrange's basis functions are given by

$$L_k(x) = \frac{(x - x_0)(x - x_1) \cdots (x - x_{k-1})(x - x_{k+1}) \cdots (x - x_n)}{(x_k - x_0)(x_k - x_1) \cdots (x_k - x_{k-1})(x_k - x_{k+1}) \cdots (x_k - x_n)}$$

$$\text{for k} = 0,\ 1,\ 2, \ldots,\ \text{n}$$

The Sagemath code to determine higher order Interpolation is shown below:

```
from sympy import symbols, simplify

# We Define the Lagrange interpolation function
def lagrange_interpolation(x_values, y_values):
    n = len(x_values)
    x = symbols('x')
    result = 0

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

    return simplify(result)

# Example usage:
x_values = [x_0, x_1, x_2, ..., x_n]
y_values = [y_0, y_1, y_2,..., y_n]

interpolation_polynomial = lagrange_interpolation(x_values, y_values)
expanded_polynomial = expand(interpolation_polynomial)
print(expanded_polynomial)
```

Figure 6: Sagemath code for Higher Order Interpolation.

Example 3

Determine the Polynomial $P_2(x) = a_0 + a_1x + a_2x2$ whose graph passes through the points (1, 4), (2, 0) and (3, 12).
Solution

```
from sympy import symbols, simplify

# We Define the Lagrange interpolation function
def lagrange_interpolation(x_values, y_values):
    n = len(x_values)
    x = symbols('x')
    result = 0

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

    return simplify(result)

# Example usage: Determine the Polynomial P_2(x) = a_0 + a_1x + a_2x2 whose graph passes through the points (1, 4), (2, 0) and (3, 12).
x_values = [1, 2, 3]
y_values = [4, 0, 12]

interpolation_polynomial = lagrange_interpolation(x_values, y_values)
expanded_polynomial = expand(interpolation_polynomial)
print(f"Hence, the Polynomial is {expanded_polynomial}")
```

Evaluate

Language: Sage

Share

```
Hence, the Polynomial is 8*x**2 - 28*x + 24
```

Help | Powered by SageMath

Figure 7: Solution to example 3.

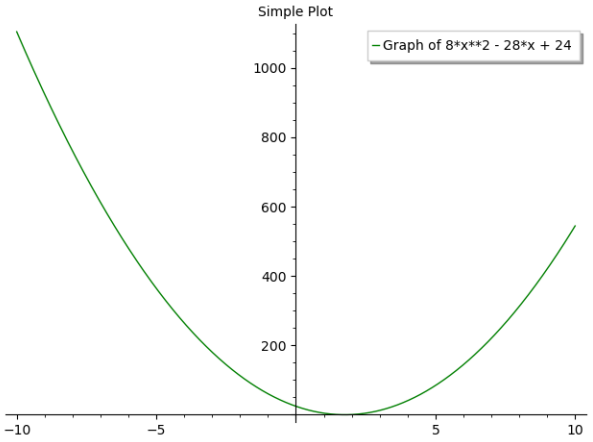

Figure 8: Graph of $8x^2 - 28x + 24$ by Sagemath.