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.

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

```python
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.

The provided SageMath code defines a function 'do_linear_Iteration', which is a technique used to estimate a value between two known data points on a straight line. It takes two data points (x and y values) and, optionally, a point at which we want to perform the interpolation. Here's a breakdown of the code:

- do_linear_interpolation(point_1, point_2, point=None): This is the main function for linear interpolation. It takes two data points point_1 and point_2, and an optional point where you want to perform the interpolation.
- getEquation(x, y, denominator, second_point=False): This function is used to create the linear equation for a given point.
  - a. value_1 = y * x: Calculate the product of x and y.
  - b. value_2 = str(y) + "x": Create a string representing y * x.

c. if(second_point): If second_point is True, create the equation for the second point (subtract value_1 from value_2). Otherwise, create the equation for the first point (subtract value_2 from value_1).

- solveEquation(x_0, x_1, y_0, y_1, a): This function is used to solve the linear equation and calculate the interpolated value at a given a (x-coordinate). It calculates the following the denominator value as denominator $= (x_1 - x_0)$, the right-hand side of the linear equation for interpolation, the left-hand side of the linear equation for interpolation, and it returns the sum of the right and left-hand sides as the interpolated value.

- Finally, the 'do_linear_interpolation' function is called with specific data points point_1 and point_2 and an interpolation point 0.826. It prints the linear equations and the interpolated value at point.

The code effectively performs linear interpolation using the provided data points and calculates the interpolated value at a specific point if required. We shall now use it to solve as many examples as possible.

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 $\frac{15-3x}{3} + \frac{7x-14}{3}$

```python
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.

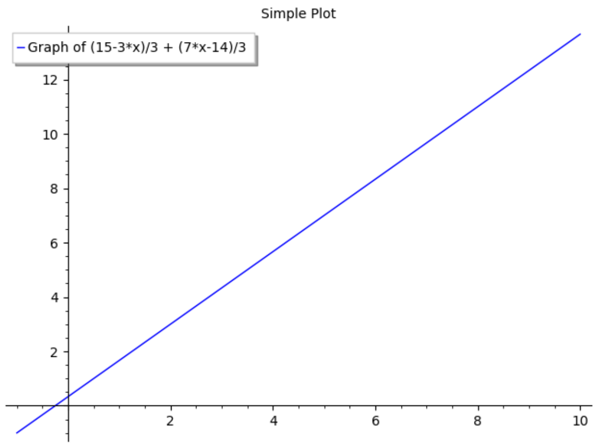


Figure 3: Graph of $\frac{15-3x}{3} + \frac{7x-14}{3}$ by Sagemath.

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.

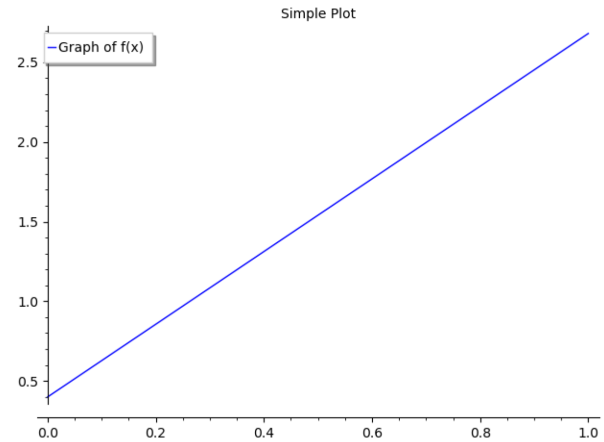


Figure 5: Graph of $P_1(x)$ by Sagemath.

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:

```python
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 The polynomial is given by $8x^2 - 28x + 24$

```python
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.

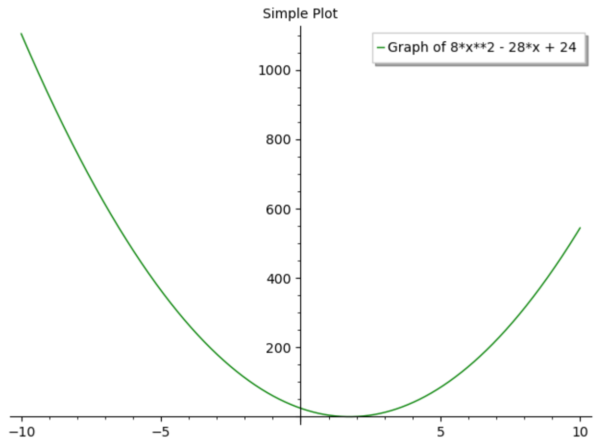


Figure 8: Graph of $8x^2 - 28x + 24$ by Sagemath.

Example 4

Use Lagrange's formula, to find the quadratic polynomial that takes the values:

x	0	1	3
y	0	1	0

Solution: Computations by Sagemath gives $\frac{1}{2}\left(3x - x^2\right)$ as the quadratic Polynomial.

```
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 = [0, 1, 3]
y_values = [0, 1, 0]

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 -x**2/2 + 3*x/2
```

Help | Powered by SageMath

Figure 9: Solution to example 4.

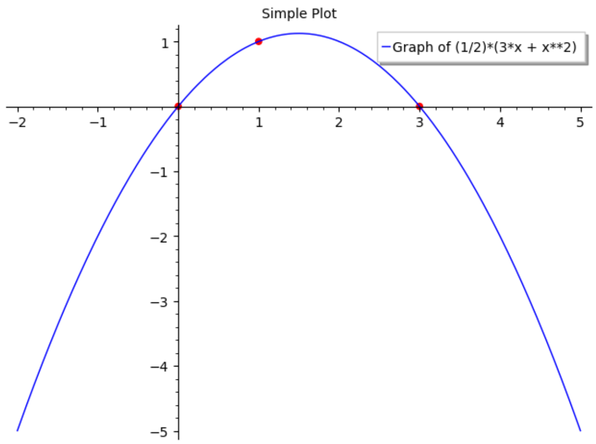


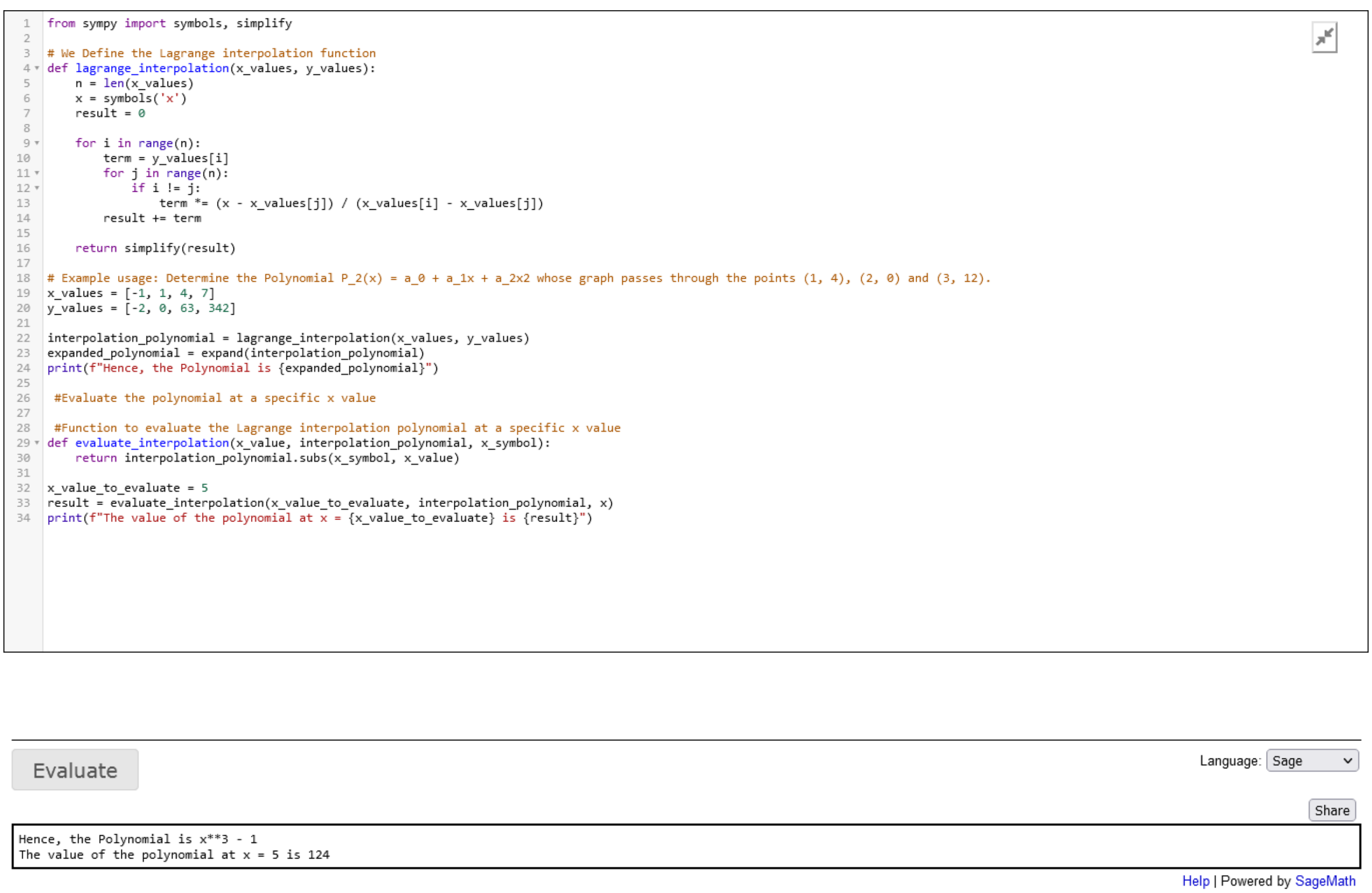
Figure 10: Graph of $\frac{1}{2}\left(3x - x^2\right)$ by Sagemath.

Example 5

Construct the Lagrange interpolation polynomial for the data:

x	-1	1	4	7
y	-2	0	63	342

Hence, interpolate at x = 5. Solution: The Lagrange polynomials is given by $x^3 - 1$, and value of the polynomial at $x = 5$ is 124.



```
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, 1, 4, 7]
y_values = [-2, 0, 63, 342]

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

 #Evaluate the polynomial at a specific x value

 #Function to evaluate the Lagrange interpolation polynomial at a specific x value
def evaluate_interpolation(x_value, interpolation_polynomial, x_symbol):
    return interpolation_polynomial.subs(x_symbol, x_value)

x_value_to_evaluate = 5
result = evaluate_interpolation(x_value_to_evaluate, interpolation_polynomial, x)
print(f"The value of the polynomial at x = {x_value_to_evaluate} is {result}")
```

Evaluate

Language: Sage

Share

```
Hence, the Polynomial is x**3 - 1
The value of the polynomial at x = 5 is 124
```

Help | Powered by SageMath

Figure 11: Solution to example 5.

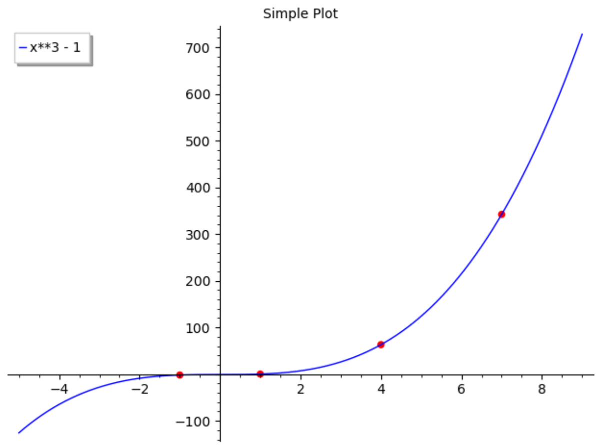


Figure 12: Graph of $x^3 - 1$

Real Life Applications of Interpolation

Interpolation plays a pivotal role in various real-life applications across a wide range of fields, contributing to the accurate representation and analysis of data in diverse scenarios. Its significance extends to the following domains:

- **Engineering**: It predicts material behavior under extreme conditions, such as high temperatures or pressure, contributing to the design and evaluation of structures and systems.
- **Statistical Analysis**: Interpolation helps in smoothing out data sets, ensuring even distribution, and eliminating erratic trends, benefiting areas like sales data analysis.
- **Weather Forecasting**: Meteorologists rely on interpolation to predict weather conditions in regions with limited direct observations, enhancing forecasting accuracy.
- **Finance**: In financial modeling, interpolation is used to estimate the value of securities not traded in the market, contributing to sound investment decisions.

In summary, the applications and uses of interpolation showcased here demonstrate its critical role in addressing real-life challenges, interpolation serves as a versatile tool for solving complex problems and enhancing our understanding of the world. By delving into these practical scenarios and discussing how interpolation aids in data analysis and prediction, we have highlighted the power of mathematical techniques in addressing real-world issues.

It's clear that we've not only explored the concept but have also written code to solve real-life problems, underscoring the practicality and relevance of interpolation in a wide range of fields.