Numerical Analysis Project 2 Fernando Thaureaux: 6236069

Writing the Code

-Language: Python

-Input:

- time - the values for how long the car has been observed (x-axis on a graph)
- z - duplication of each data point for time for Hermite Interpolation
- distance - the values for what position the car is in after x time (y-axis)
- speed - the value for how fast the car is moving at any given time

-Output: The Hermite Polynomial evaluation at time = 10, which is the approximate position of the car at 10 seconds, and a plot of the position of the car from 0 to 13 seconds.

To start the divided differences table for Q(i,j) we set the function values for distance f(xi) and speed, the derivative, f' (xi). For the first divided difference we can use: Q(2i+1, 1) = f' (xi) and Q(2i, 0) = Q(2i+1, 0) = f(xi). For the second (and later) divided differences we can use: $Q(i, j) = \frac{Q(i, j-1) - Q(i-1, j-1)}{z(i) - z(i-j)}$, so that we construct the Hermite Polynomial: H(x) = Q(0, 0) + Q(1, 1)(x - x0) + Q(2, 2)(x - x0)^2 + Q(3, 3)(x - x0)^2 (x - x1) … to then evaluate H(10) to predict the distance/position at 10 seconds.

For plotting, a list of x time points are created equally spaced from 0 to 13 seconds. The function/polynomial is run at each time interval to find the distance f(x) values, and those points are plotted on a graph. The graph should be smooth because Hermite Interpolation includes the first derivative at each function value.

Choice of Parameters

Distance is a function of time, as Distance = rate * time, and in Calculus we are taught that speed is the first derivative of distance. Given:

Time	0	3	5	8	13
Distance	0	225	383	623	993
Speed	75	77	80	74	72

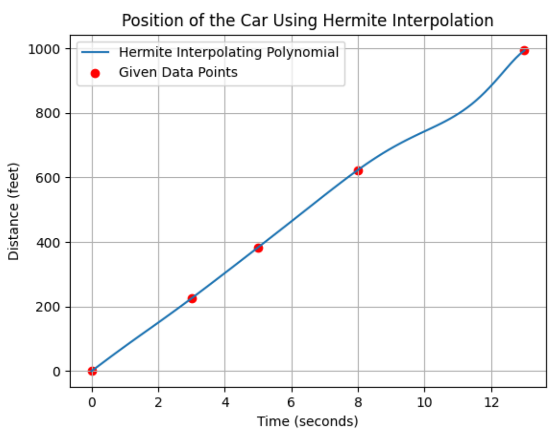
Results and Analysis

Table as output by the program

Divided Difference Table (Q):
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 75, 0, 0, 0, 0, 0, 0, 0, 0]
[225, 75.0, 0, 0, 0, 0, 0, 0, 0, 0]
[225, 77, 0.6667, 0.2222, 0, 0, 0, 0, 0, 0]
[383, 79.0, 1.0, 0.0667, -0.0311, 0, 0, 0, 0, 0]
[383, 80, 0.5, -0.25, -0.0633, -0.0064, 0, 0, 0, 0]
[623, 80.0, 0, -0.1, 0.03, 0.0117, 0.0023, 0, 0, 0]
[623, 74, -2.0, -0.6667, -0.1133, -0.0287, -0.005, -0.0009, 0, 0]
[993, 74.0, 0, 0.25, 0.1146, 0.0228, 0.0051, 0.0008, 0.0001, 0]
[993, 72, -0.4, -0.08, -0.0413, -0.0195, -0.0042, -0.0009, -0.0001, -0.0]
Predicted position at t = 10 seconds: 742.50 feet

This result makes sense because the number is between 623 ft and 993ft, leaning towards 623 ft, as 10 seconds is closer to the 8 second mark than the 13 second mark. Also, at this point in the graph ‘speed’ is not increasing, it is in fact stabilizing and even beginning to decrease.

Graph



As expected, the graph of the car is at its slowest between 8 and 13 seconds, as those are the values where speed is at its lowest point. Distance is increasing more slowly (speed decrease) as shown from the small down curve in [8, 11].

Code

Hermite interpolation:

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

# Input data
time = np.array([0, 3, 5, 8, 13])
```

```
distance = np.array([0, 225, 383, 623, 993])
speed = np.array([75, 77, 80, 74, 72])

# Doubling the nodes for Hermite interpolation
z = np.repeat(time, 2)
n = len(z)
Q = np.zeros((n, n))

# Step 2: Setting initial values in the divided difference table
for i in range(len(time)):
    Q[2 * i][0] = distance[i]
    Q[2 * i + 1][0] = distance[i]
    Q[2 * i + 1][1] = speed[i]
    if i != 0:
        Q[2 * i][1] = (Q[2 * i][0] - Q[2 * i - 1][0]) / (z[2 * i] - z[2 * i - 1])

# Step 4: Filling the rest of the divided difference table
for i in range(2, n):
    for j in range(2, i + 1):
        Q[i][j] = (Q[i][j - 1] - Q[i - 1][j - 1]) / (z[i] - z[i - j])

# Displaying the divided difference table
print("Divided Difference Table (Q):")
print(Q)

# Function to evaluate the Hermite polynomial at a given x using the divided differences
def hermite_polynomial(x):
    result = Q[0, 0]
    product_term = 1
    for i in range(1, n):
        product_term *= (x - z[i - 1])
        result += Q[i, i] * product_term
    return result

# Predicting the position at t = 10
predicted_position = hermite_polynomial(10)
print(f"Predicted position at t = 10 seconds: {predicted_position:.2f} feet")

# Plotting the position of the car from t = 0 to t = 13
t_values = np.linspace(0, 13, 100)
position_values = [hermite_polynomial(t) for t in t_values]
plt.plot(t_values, position_values, label="Hermite Interpolating Polynomial")
plt.scatter(time, distance, color="red", label="Given Data Points")
plt.xlabel("Time (seconds)")
plt.ylabel("Distance (feet)")
plt.title("Position of the Car Using Hermite Interpolation")
plt.legend()
plt.grid(True)
plt.show()
```