Math 105B Lab Report 3

Haley Feng ID: 71813343

Objective

The goal of this project is to write a function in Matlab to compute the coefficients for a Hermite polynomial interpolating function using divided differences and then to use the coefficients to compute the Hermite interpolating polynomial. Furthermore, I will implement the Hermite polynomial through an example of interpolating the function $f(x) = e^{0.1x^2}$ at a point x and at different degree.

Introduction

I will first be coding out the Hermite polynomial interpolating function as the function *hermite* that will take input X, Y, YP and would produce the output Q, the matrix of the divided differences, and z, the new X sequences. This function would be crucial to produce the first and second column in the matrix as it would take every value of X and Y and duplicate them to produce z and the first column and then compute the first divided differences in the second column. Since the divided differences cannot be computed if the index is even then odd, as it would produce zero, I make use of the provided YP, f'(x) in replace of the divided difference. After calling the function in my matlab routine, I used the constructed matrix from the function and continue computing the second divided difference and so on until all the divided differences are evaluated. Inputting the provided table of x, f(x), and f'(x), which I stored as X = [1,2,3]; Y = [1.105170918, 1.491824698, 2.459603111]; YP = [0.2210341836, 0.5967298792, 1.475761867]; respectively, and implementing them to my routine will interpolate the function at the point x = 1.25. To compute Hermite polynomial at degree 3 I will use the nodes x_0, x_1 and to compute at degree 3 I will use the nodes x_0, x_1, x_2. For the very last step of the codes, I'll be calculating the estimate of the errors and error bounds for the two approximation.

Algorithm Method

Part A: To create the function for *hermite*, I first need to set the length of X as variable n and create an empty array and matrix to store the values of f(x) and divided differences. Beginning with a for loop i that goes from 0 to n-1, I set z(2i+1) and z(2i+2) to the respected X(i+1) and Q(2i+1,1) and Q(2i+2,2) to the respected X(i+1). Sinces the index from Matlab starts from 1, I added +1 to all the i values. As for the divided difference, I set Q(2i+2,2) to the respected YP(i+1) and calculate Q(2i+1,2) using the formula F = (f(x1)-f(x0))/(x1-x0) and implementing it as (Q(2i+1,1)-Q(2i,1))/(z(2i+1)-z(2i)). When the function is run, it would compute the following code:

```
Q =

    1.1052         0         0         0         0         0
    1.1052    0.2210         0         0         0         0
    1.4918    0.3867         0         0         0         0
    1.4918    0.5967         0         0         0         0
    2.4596    0.9678         0         0         0         0
    2.4596    1.4758         0         0         0         0


z =

     1     1     2     2     3     3
```

Part B:

As to compute the rest of divided differences, I created two for loops, i that goes from 2 to 2n-1 and j that goes from 2 to i. Using the formula F again with different indexing, the algorithm would complete constructing the matrix Q that contains all the divided difference. In order to build the Hermite interpolating polynomial, only the diagonal divided differences are necessary so for step 3 I make use of the Hermite polynomial formula. I used another two more for loops that sums up all the diagonal divided differences with its respected z-z(j) evaluation. The outcome of the completed matrix is as shown below:

```
Q =

    1.1052         0         0         0         0         0
    1.1052    0.2210         0         0         0         0
    1.4918    0.3867    0.1656         0         0         0
    1.4918    0.5967    0.2101    0.0445         0         0
    2.4596    0.9678    0.3710    0.0805    0.0180         0
    2.4596    1.4758    0.5080    0.1369    0.0282    0.0051
```

Interpolating the function $f(x) = e^{0.1x^2}$ in degree 3, I would get the formula

$H_3(x) = 1.1052 + 0.221(x-1) + 0.1656(x-1)^2 + 0.1656(x-1)^2(x-2).$

At the point x=1.25, $H_3(1.25)$ would gives the approximate of 1.1687 and $H_5(1.25)$ would gives the approximate of 1.169. To calculate the estimate of the errors, I simple subtract $H_4(x)$ from $H_5(x)$ to get -3.1406e-04 and subtract $H_2(x)$ from $H_3(x)$ to get -0.0021. As for the error bound, I start by solving for the 4th and 6th derivative respectively and evaluating them at 3. Then following the interpolation error formula, I solved for (x-x0)^2(x-x1)^2 by evaluating at the given x = 1.25 and the same for (x-x0)^2(x-x1)^2(x-x3)^2. Plugging everything back to the formula, I calculated the error bound for degree 3 is $\left| f(1.25) - H_3(1.25) \right| = 0.0025$ and for degree 5 is $\left| f(1.25) - H_5(1.25) \right| = 4.4263e-04$.

Conclusion

By evaluating x at two different degree, I observed that the higher degree of Hermite polynomial calculates a closer approximation to f(x) as I can check the absolute error between $H_5(x)$ and f(x) is 1.0238e-04 which is smaller from the absolute difference between $H_3(x)$ and f(x), 4.2166e-04. As with Newton divided difference methods, the Hermite polynomial provide easy access to input new data if needed. The main difference in calculating would mainly be computing the first divided difference differently as it can make use of the derivative of the function.