Lab Report: Newton Divided Differences

Introduction

I will first be coding out the Newton's divided difference as the function newton that will take input X, Y and would produce the output F, the divided difference. In order to produce all the divided difference, I will make use of recursion and indexing in my matlab routine and store the necessary differences to produce the Lagrange polynomial. Using the provided table of x and f(x), I store them as X=[0.0,0.1,0.3,0.6,1.0] and Y=[-6.0,-5.89483,-5.65014,-5.17788,-4.28172], respectively, and implement them to my routine to compute an approximation of the evaluation point x = 0.05 and the estimate of the error of using the fourth degree Lagrange interpolating polynomial to ensure accuracy of the computation. Since order of X does not matter to compute x in example 1, as the absolute difference between x and xi of X is the same order as X, I will approximate another evaluation point x = 1.05 with an addition point x = 1.1 and f(1.1) = -3.99583 added to X and Y. In this case, x is in between 1.0 and 1.1 so I will create an additional routine to reorder the X values from largest to smallest with Y following through.

Algorithm Method

Part A: To create the function for newton, I first need to ensure that when the execution only involves x0 and x1 or x1 and x2 and so on, the divided difference does not rely on any other computation as it is the zero degree. Using an if statement that check if there are only two variable in the list, it compute the divided difference using the initial computation of F = (f(x1)-f(x0))/(x1-x0). Proceeding to the recursion algorithm, it simply call the function itself again and again if the list has more than two variable. I set the recursion to the formula of Newton divided difference which takes the element from list Y from 1 to n subtracting element 0 to n-1. Since the denominator does not require calling the function itself, it's simply the very last element of X subtracting the first element.

Part B: In order to create Lagrange interpolation polynomial and find its estimation evaluated at x, I created a list labeled poly to store all the diagonal coefficient and an empty list labeled eval_x to store all four degree Lagrange interpolation polynomials. Since P0(x) = f(x0), I put the

first element of Y in list poly and set a variable labeled sum that also equal to the first element of Y. This sum is the summation of all the diagonal divided difference I will need at the end to compute all the polynomial at different degrees. I start with two for loops, i that goes from 1 to length of X minus one since Matlab counts from 1 instead of 0, and j that goes from 1 to i. The for loops will create an algorithm that would compute the divided difference using the first two Y elements and then proceed to use that divided difference to compute the second degree polynomial and so on with the help of the recursion algorithm implemented in my function newton. The variable product is created beginning with 1 to multiply all the (x-X(j)) value needed to create the polynomial. To ensure that only the diagonal coefficient or divided differences are being added to the list poly, I used an if statement that requires the ith case is equal to j. At the end of the loops, I summed up the divided difference with its respected product and add them to list eval_x for storage. I end up with the result of P1(0.05) ≈ -5.9474 using the first degree Lagrange interpolation polynomial, P2(0.05) ≈ -5.9488 using the second degree, and P3(0.05) ≈ P4(0.05) ≈ -5.9487 using the third and fourth degree. To compute error of fourth degree Lagrange interpolating polynomial I subtract P4(0.05) from P3(0.05) and got the error of -2.1662e-05 which it's justify as it is insignificantly small. As to evaluate the Lagrange polynomial at x = 1.05, I use the same routine as previous problem but with the order of X and Y different. The X and Y needs to be from largest to smallest to get a more accurate evaluation at point x. To do this I simply create a for loop and an if statement to compare if one abs(X(i)-x) is smaller than another abs(X(i+1)-x), if it is then it would be added to the start of the new_X list and if not then it would be added to the end of the new list. Proceeding with the code I can construct the interpolating polynomial of degree 5 using list poly : P5(x) = -3.9958 + 2.8589(x-1.1) + 1.237(x-1.1)(x-1) + 0.3566(x-1.1)(x-1)(x-0.6) + 0.0786(x-1.1)(x-1)(x-0.6)(x-0.3) + 0.0142(x-1.1)(x-1)(x-0.6)(x-0.3)(x-.1), note that it omits the last element of X as desired. Evaluating at x = 1.05, I got the result of P1(1.05) ≈ -4.1388 using the first degree Lagrange interpolation polynomial, P2(1.05) ≈ -4.1419 using the second degree, and P3(1.05) ≈ P4(1.05) ≈ P5(1.05) ≈ -4.1423 using the third, fourth, and fifth degree. In this case, the estimate of the error for the fifth degree Lagrange interpolating polynomial is -1.1350e-05 by computing the difference between P4(1.05) and P5(1.05).

Conclusion

By evaluating at two different evaluation point, I observed that as higher the nth degree Lagrange interpolating polynomial is, the more accurate and closer the approximation is from its previous n-1th degree as its estimate of the error is also getting smaller and smaller. Additionally, the Newton divided difference methods are used to successively generate the polynomials themselves. This is shown from problem 2 as adding new point to the list does not affect what it has been calculated before and so that I can continue adding more points to get a higher degree lagrange polynomial for a better approximation.