Lab Report: Cubic Splines Interpolation

Introduction

I will first be coding out the cubic spline interpolating polynomial as the function *natural_splines* that will take input X and Y and would produce the output a, b, c, d, the coefficients for the corresponding cubic splines. Then, using the provided table of x and f(x), I'll compute and plot the points z = linspace(-2,2,10) depending on which interval the z-values is in. At the end of the code, the evaluated cubic splines would connect to form a line that should looks similar to the plot of original f(x) function. In order to plot the Lagrange interpolation, I'll simply use the *f_lagrange* function I had coded for assignment one. The function prints out the coefficients for the lagrange polynomial and produce a similar graph to that of function f(x). At the end of the code, I'll compare and contrast the two result along with the original function by looking at the error estimated and theoretical error estimate.

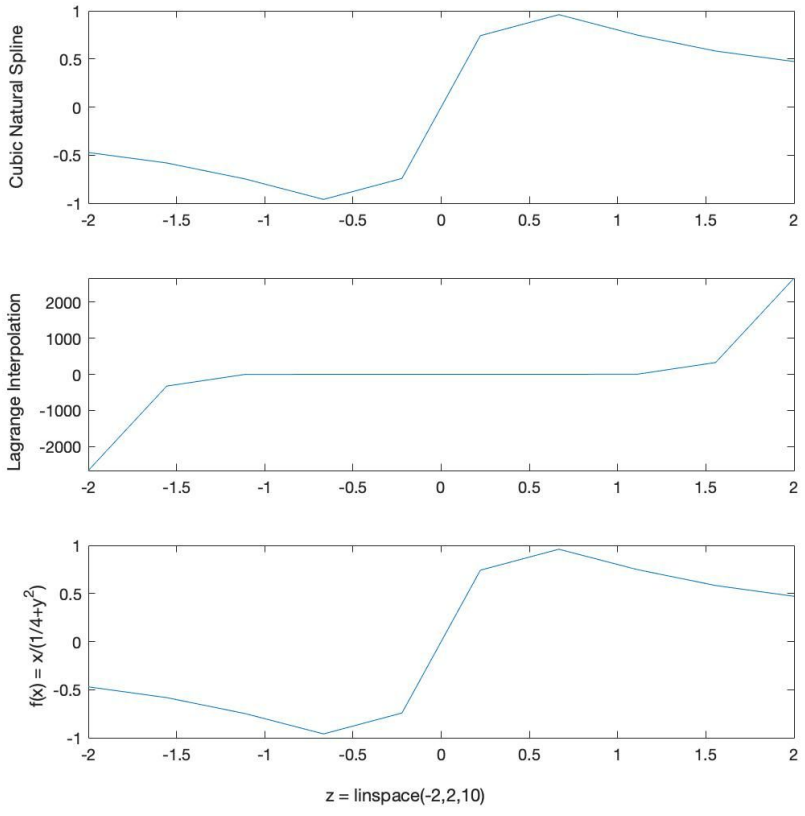
Algorithm Method

Part A: To create the function *natural_splines*, I first need to set the length of X as n and create all the empty lists to storage the computation for *a, b, c, d* in. Since *a* is the same as f(x) from the table, I can set it as *a* = Y. As for the rest of the code, I simply follow the guideline provided by Algorithm 3.4 to solve for the tridiagonal linear system. When the code is run, it would return elements of *a, b, c, d* from 1 to n-1 since the last element is empty.

Part B: Using the result of the function and the 10 uniformly spaced grid points from -2 to 2, I use two for loops and an if statement to compute the cubic spline. The evaluating points, which I call x instead of z, goes through the loop to check which interval, X(j) and X(j+1), it is in and when found, it gets computed in the equation $S(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3$ for $x_j \le x \le x_{j+1}$. The loop then breaks to check for the next interval of the next x. When the 10 S values get evaluated, I plot it against x. As to compute for the lagrange polynomial, the *f_lagrange* function takes the input of X, Y, and x. And so using the given points and the uniformly spaced points, it produce 10 lagrange coefficients as needed. Comparing the graph of cubic spline and lagrange, it is obvious that cubic spline produce a closer approximation to f(x).

As for the values of cubic spline, it is also a lot closer to values of f(x) as the error for each individual points (i.e [0.0023 -0.0011 -0.0005 0.0002 -0.0002 0.0002 -0.0002 0.0005 0.0011 -0.0023]) are smaller than the errors of Lagrange(i.e 1.0e+03 x [2.6661 0.3259 0.0014 -0.0000 0.0000 -0.0000 0.0000 -0.0014 -0.3259 -2.6661]). Furthermore, the maximum error for the cubic spline interpolant is calculated to be 24.4128. It is noted that this calculation is based on the fourth-order-error-bound for clamped cubic spline rather than the natural cubic spline. However, since the natural cubic spline maximum error would be a lot larger than the error of the clamped cubic spline, and the actual errors, the differences mentioned above, are all smaller than 24.4128, I can conclude that the actual error would also be smaller than the predicted error by the theoretical error estimate.

Graph



Conclusion

Using the cubic spline interpolation is more accurate in approximating a function since it divides the approximation interval into a collection of subintervals. Unlike the Lagrange interpolation polynomial, in which it uses only one single high-degree polynomial that can oscillate erratically, the cubic spline interpolation construct different approximating polynomial on each subinterval. Additionally, since this labs uses natural spline as boundary to the piecewise-polynomial approximation, it is not the most accurate form of f(x). A clamped boundary condition can perhaps produce an even more accurate approximated graph of f(x) in comparison to the graphs above.