DEPARTMENT OF ELECTRICAL ENGINEERING, IIT MADRAS

Assignment 1 - Polynomial Interpolation

Surajkumar H - EE11B075

September 11, 2014

1 QUESTION 1: 5-POINT 4TH ORDER INTERPOLATION

In this question, we sample the function $f(x) = sin(x + x^2)$ uniformly at 5 points in $[0,1]$. We then use these 5 sample values to interpolate (and extrapolate) the function values at points in the range given by

```
xx=np.linspace(0,1,5);
```

1.1 BACKGROUND AND FUNCTIONS

We examine how polynomial interpolation performs at these values and compare it with the theoretical (known) value of the function at these points. Since we have exactly 5 samples, we run 4^{th} order polynomial interpolation on all points in the range. We use the modified Neville's Algorithm. The original algorithm constructs the Lagrangian polynomial recursively, running interpolation on smaller sets (of say m) points, and uses adjacent sets to estimate the value of the interpolation using $m+1$ points. The recursive relation is

$$P_{i(i+1)\ldots(i+m)} = \frac{(x - x_{i+m})P_{i(i+1)\ldots(i+m-1)} + (x_i - x)P_{(i+1)(i+2)\ldots(i+m)}}{x_i - x_{i+m}} \tag{1.1}$$

where x is the point where the function value is to be inter(extra)polated. By design, the function values coincide with the data values at the known sample points.
The modified version of the algorithm (which we use here) starts with an initial estimate of the function value (we do this by finding the sample closest to the point of interest), and recursively computing *small differences* between adjacent estimates (using the smaller sets as

above). We choose one of these differences at each stage, and add it to the function estimate at the point of interest. Mathematically, we define for $m = 1, 2, \ldots, N-1$

$$\begin{aligned} C_{m,i} &= P_{i(i+1)\ldots(i+m)} - P_{i(i+1)\ldots(i+m-1)} \\ D_{m,i} &= P_{i(i+1)\ldots(i+m)} - P_{(i+1)(i+2)\ldots(i+m)} \end{aligned} \tag{1.2}$$

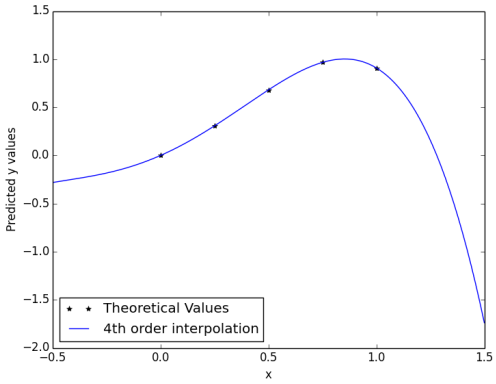
that is, the upper and lower differences in the ancestor tree (as seen in [1]). From this, we get

$$\begin{aligned} C_{m+1,i} &= \frac{(x_{i+m+1} - x)(C_{m,i+1} - D_{m,i})}{x_i - x_{i+m+1}} \\ D_{m+1,i} &= \frac{(x_i - x)(C_{m,i+1} - D_{m,i})}{x_i - x_{i+m+1}} \end{aligned} \tag{1.3}$$

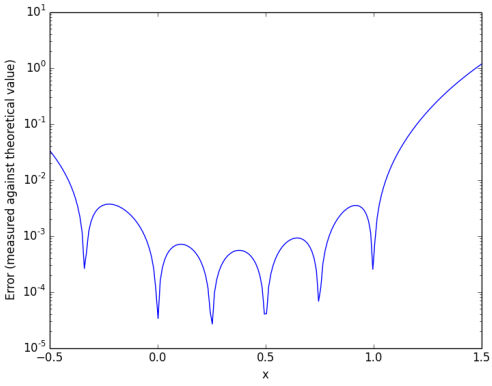
At each level m, the Cs and Ds represent the corrections made to the value estimate. We recursively compute these corrections using 1.3. We then choose a path through the ancestor tree which connects the final interpolated value ($m = N-1$), to the initial estimate. We then add all these corrections to get the interpolated value at the point x. In our implementation of the algorithm, we chose the correction based on whether the index estimate is above / below the average ($\frac{N-m}{2}$). This way, we ensure movement towards the rightmost element correctly.
A general python function POLINT was written, which computes the N^{th} order interpolated value at a single point from the set of data points and values. The function FINDNEAREST takes an input point, the entire data set, the order of interpolation N, and computes indices of the nearest N data points. Later, we test how the error varies with the order of interpolation. It is very helpful to have a function which finds the closes N samples for a given value, since we can use these to interpolate the function value at the unknown point x. Finally, we used NEARESTPOLINT to perform point by point inter(extra)polation on every point in xx (set of unknown values).

1.2 Interpolation, Extrapolation and Error

Figure (1.1a) shows the comparison of the theoretical, and the 4th order interpolated value at the intermediate and outer points. Figure (1.1b) shows the error in 4th order interpolation, taken with respect to the theoretical function values at the point (given by $f(x)$). We clearly see that the error at the interpolated points is of the order 10^{-3} for middle points, and 10^{-2} as we move closer to the edge. This makes sense, as the data at the edge doesn't give us equal information on how the function behaves on both sides (which middling values give us). The spike values indicate the known points, where the error is 0 (the log should be $-\infty$, but python ignores the value), and the points very close to the known point have a smaller error. As we go closer to the midpoint between any 2 known values, the error increases.
The behaviour beyond $[0, 1)$ is a lot more unpredictable. For this particular function, the error increases as we move further away from the known points. We are using N^{th} order (N=4) interpolation. We are trying to fit an Nth order polynomial to a given set of N points. As we can clearly see the polynomial fit $\to \pm\infty$ as $x \to \pm\infty$. This means as we go further away from the known data points, the extrapolated function blows up. Since the actual $f(x)$ is bounded (maximum value is 1), the error value grows larger with x (cannot guarantee monotonicity).



(a) Theoretical and interpolated values of $f(x)$



(b) 4^{th} order Interpolation error

Figure 1.1: Comparison of theoretical and 4^{th}-order inter(extra)polated function values, and the absolute error in interpolation

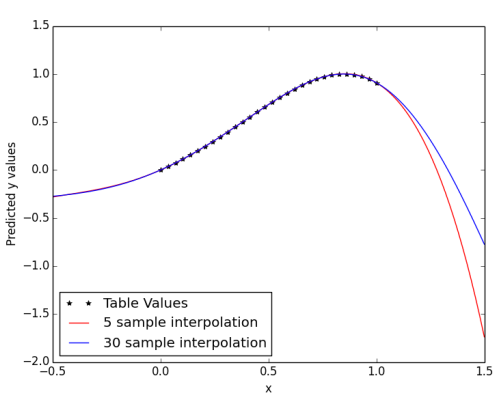
2 Question 2: 30-point 4th order Interpolation

In this problem, we investigate the effect of higher sample point density on interpolation. Specifically, we compare the previous case of $4th$ order interpolation with 5 sample points, to $4th$ order interpolation with 30 sample points. We use FINDNEAREST to find the nearest 5 points for any given point, and then use POLINT to find thee interpolated value. We run this over the same data set as in the first question.
Intuitively, we expect denser interpolation to give much lower error. This is because we fit the same order polynomial over a much closer set of points. Since the function is 'nice' (the function doesn't have discontinuous derivatives), the local behaviour of the function described by 5 close sample points should give a closer approximation of the function, as compared to 5 far away points.
Figure (2.1a) shows the comparison of the theoretical, the 5 sample and the 30 sample interpolation. Figure (2.1b) shows the absolute error comparison for the 2 cases. The plot agrees with our intuition. The interpolation error is roughly 10^{-3} for the 5 sample version, and 10^{-7} for the 30 sample version.

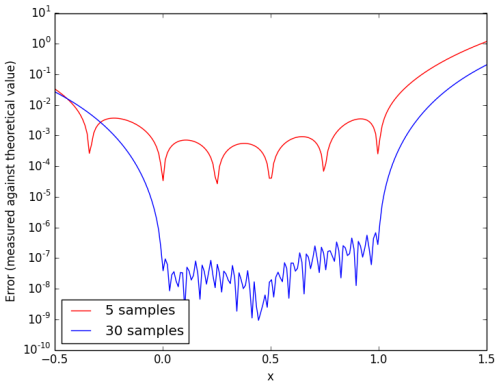
References

[1] William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery, "Numerical Recipes in C"

[2] J. Stoer, R. Bulirsch, "Introduction to Numerical Analysis"



(a) Theoretical and interpolated values of $f(x)$



(b) Comparison of 4^{th} order Interpolation error run for 5 and 30 sample points

Figure 2.1: Comparison of theoretical and 4^{th}-order inter(extra)polated function values obtained using 5 and 30 sample points