Akima sub-spline interpolation

René Munk Thalund

June 2022

1 Introduction

We seek to implement the Akima sub-spline interpolations method in `C#`. The Akima sub-spline are cubic sub-splines, that seeks to alleviate some of the shortcomings of regular cubic splines while sacrificing continuity of the second derivative of the interpolation function. Furthermore since the polynomial coefficients for each segment are calculated using only a few input points locally around the segment, the Akima sub splines can be constructed without the large equation set, Gauss elimination and back-substitution needed in for cubic splines. Array indexing corresponding to the source code used below.

2 Basic construction

The piece-wise polynomials used on a set of n input points, (x_i, y_i), with $n = 0, 1, \ldots, n-1$ have the form:

$$A_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3 \tag{1}$$

It's helpful to define the x-increments and slopes:

$$h_i = x_{i+1} - xi \tag{2}$$

$$p_i = \frac{y_{i+1} - yi}{h_i}, \text{ both with } n = 0, 1, \ldots, n-2 \tag{3}$$

The first derivatives $A_i'(x) = b_i$, are found using a weighed average of slopes of neighbouring input points;

$$\begin{aligned} b_i &= \frac{w_i p_{i-1} + w_{i-2} p_i}{w_{i+1} + w_{i-1}}, \text{ if } w_i + w_{i-2} \neq 0 \\ b_i &= \frac{p_{i-1} + p_i}{2}, \text{ otherwise} \end{aligned} \tag{4}$$

The weights are given by $w_i = |p_{i+1} - p_i|$. The quadratic and cubic coiffecients are then calculated as:

$$a_i = y_i,\ c_i = \frac{2p_i - 2b_i - b_{i+1}}{h_i} \text{ and } d_i = \frac{b_i + b_{i+1} - 2p_i}{h_i^2} \tag{5}$$

3 Slope at end-points

The formula 4 for b_i, the spline's tangent slope at (x_i, y_i) relies two adjacent input points on either side and hence doesn't work for b_0, b_1, b_{n-2} and b_{n-1}. My implimentation offers three different schemes for the four end-point slopes:

3.1 Naive end-points

As suggested in the course notes, and the Wiki page on Akima sub-splines, the four slopes can naively be chosen as:

$$b_0 = p_0,\ b_1 = \frac{p_0 + p_1}{2},\ b_{n-2} = \frac{p_{n-3} + p_{n-2}}{2} \text{ and } b_{n-1} = p_{n-2} \tag{6}$$

3.2 Akima end points

Akima suggested adding four supplementary slopes p_{-2}, p_{-1}, p_{n-1} and p_n to be constructed as:

$$p_{-2} = 3p_0 - 2p_1,\ p_{-1} = 2p_0 - p_1,\ p_{n-1} = 2p_{n-2} - p_{n-3} \text{ and } p_n = 3pn - 2 - 2pn - 3 \qquad (7)$$

Given those, the entire set of $\{b_i\}$ can be calculated using equation 4. Comparing my `C#` results with Python results, I found that this end-point scheme is the one used for the latter, see figure 5.

3.3 Bica's end points

In a 2014 paper (https://doi.org/10.1016/J.CAGD.2014.03.001), A.M Bica devised a method so to minimise *partial quadratic oscillation in average* (PQOA) in the intervals on the end sub-intervals $[x_0, x_2]$ and $[x_{n-3}, x_{n-1}]$. The quite involved definitions on the PQOA is out of the scope, but I found it worthwhile to implement an option to use the four Bica-slopes, given by:

$$b_0 = \frac{1}{\delta} \cdot \left[\frac{y_1 - y_0}{4h_0} + \frac{9h_1^3 \cdot b_2}{16(h_0^3 + h_1^3)} + \frac{3h_0^2 \cdot (y_1 - y_0)}{16(h_0^3 + h_1^3)} + \frac{3h_1^2 \cdot (y_2 - y_1)}{16(h_0^3 + h_1^3)} \right]$$

$$b_1 = \frac{1}{\delta} \cdot \left[\frac{3h_1^3 \cdot b_2}{4(h_0^3 + h_1^3)} + \frac{7h_0^2 \cdot (y_1 - y_0)}{16(h_0^3 + h_1^3)} + \frac{h_1^2 \cdot (y_2 - y_1)}{4(h_0^3 + h_1^3)} \right], \text{ where}$$

$$\delta = \frac{7h_0^3 + 16h_1^3}{16(h_0^3 + h_1^3)}$$

$$b_{n-1} = \frac{1}{\delta'} \cdot \left[\frac{y_{n-1} - y_{n-2}}{4h_{n-2}} + \frac{9h_{n-3}^3 \cdot b_{n-3}}{16(h_{n-3}^3 + h_{n-2}^3)} + \frac{3h_{h-2}^2 \cdot (y_{n-2} - y_{n-1})}{16(h_{n-3}^3 + h_{n-2}^3)} + \frac{3h_{n-3}^2 \cdot (y_{n-3} - y_{n-2})}{16(h_{n-3}^3 + h_{n-2}^3)} \right]$$

$$b_{n-2} = \frac{1}{\delta'} \cdot \left[\frac{3h_{n-3}^3 \cdot b_{n-3}}{4(h_{n-3}^3 + h_{n-2}^3)} + \frac{7h_{n-2}^2 \cdot (y_{n-1} - y_{n-2})}{16(h_{n-3}^3 + h_{n-2}^3)} + \frac{h_{n-3}^2 \cdot (y_{n-2} - y_{n-3})}{4(h_{n-3}^3 + h_{n-2}^3)} \right], \text{ where}$$

$$\delta' = \frac{7h_{n-2}^3 + 16h_{n-3}^3}{16(h_{n-3}^3 + h_{n-2}^3)}$$

When comparing to the Bica paper, note that it considers $n+1$ input points (where I use n), and Bica indexes h_i from 1 (where I index from 0 in order to maintain consistensy with the code.)

4 Optimization

- After reading in the input data, the program calculates an array of coarse accumulated integration areas at the input x values. This array is kept and makes subsequent integration to interpoint x values faster.
- The last found binary search interval is kept, and checked as first option when a new integration point is requested. If z values all over the place are needed this adds two comparisons extra per if the next point is not in the same interval. However, when a large range of monotonic increasing or desceasing z values is requested – like when writing n consequtive z-values to the output file – it saves greatly on the search.

5 Results

I tested my Akima sub-spline routine on three strategic datasets. One with values of $sin(x)$, one with values of a step function and one with more arbitrary values, to check the difference in the three end-point schemes. The results are seen in figures 1 through 5.

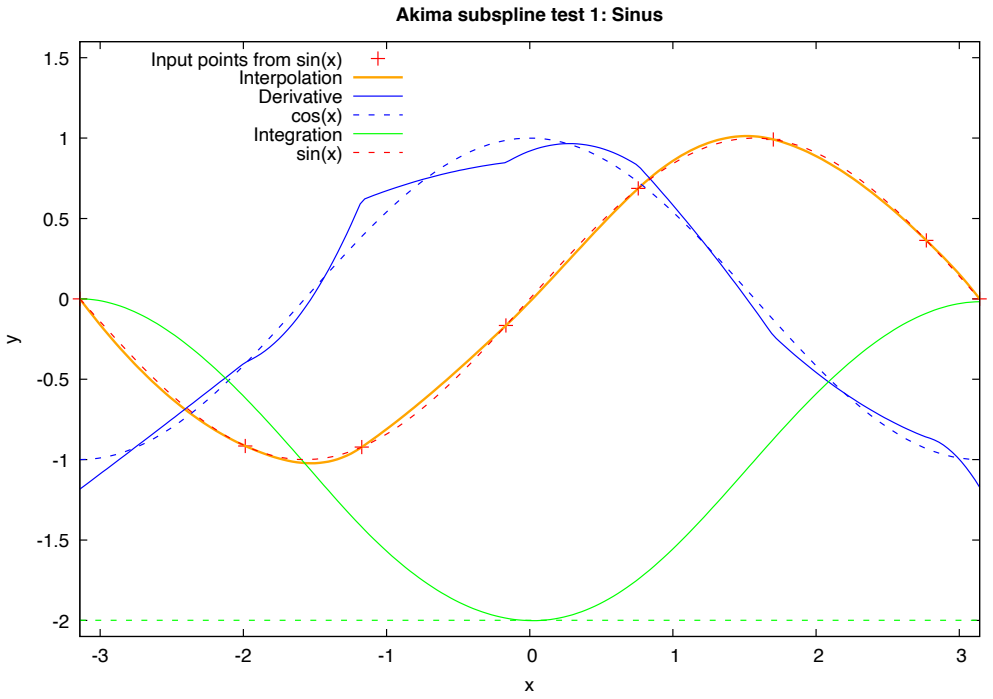

Figure 1: Akima sub-splines on a dataset of $sin(x)$, with the x-values at irregular intervals. We notice that while the sub-spline (here using the Akima end point scheme) succeeds to connect the points smoothly, the resulting spline does not follow the mathematical sin(x) curve too well. The jagged line of the derivative (blue) is a tell-tell of the discontinuous second derivative – a trade-off to avoid the oscillating artefacts of cubic spline. The mathematical derivative $cos(x)$ plotted for reference. The integral (green) behaves more gently. Mathematically the anti-derivative of $sin(x)$ would reach -2 at $x = 0$ and be back at 0 at $x = \pi$ and this is replicated acceptably by the spline integration.

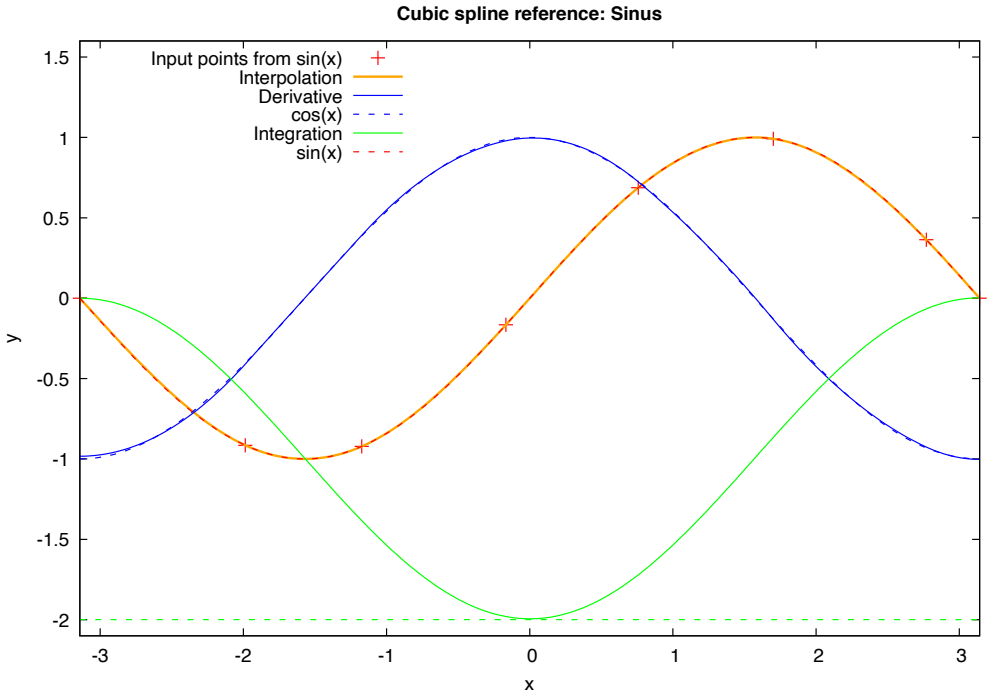

Figure 2: For reference my Cubic spline routine on the same dataset as in 1. The cubic spline replicates $sin(x)$ and it's derivative excellently given just 8 input points over a full cycle.

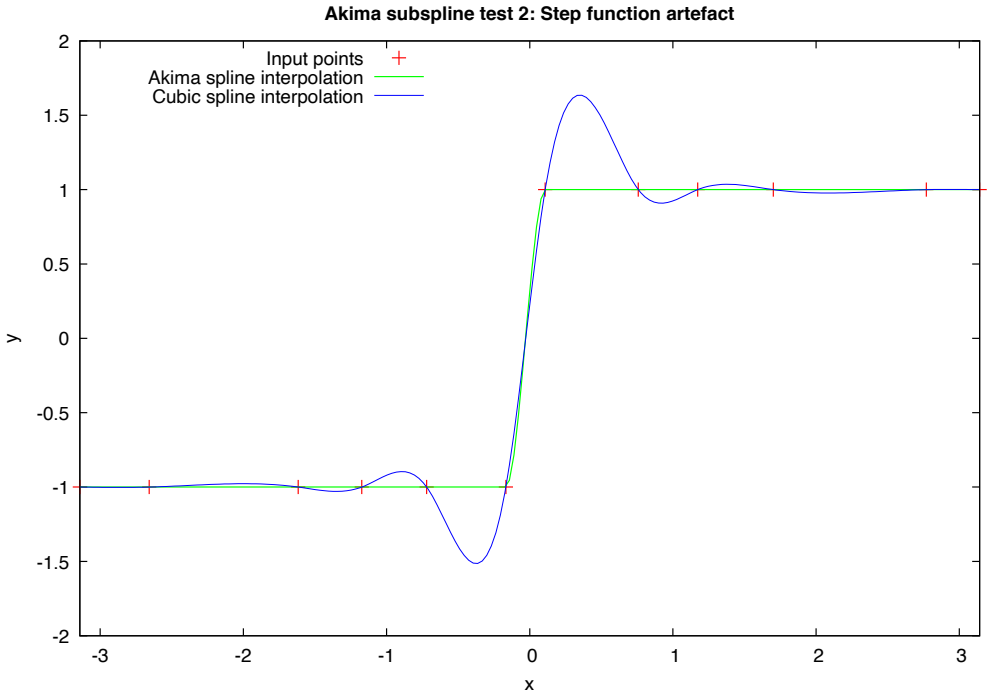

Figure 3: A data with a sudden step from -1 to 1 and otherwise constant values dementrates the advantage of Akima sub-splines. While the cubic spline interpolation (blue) has damped oscillations on either side of the jump, the akima sub-splines (green) stays constant besides on the interval with the actual step.

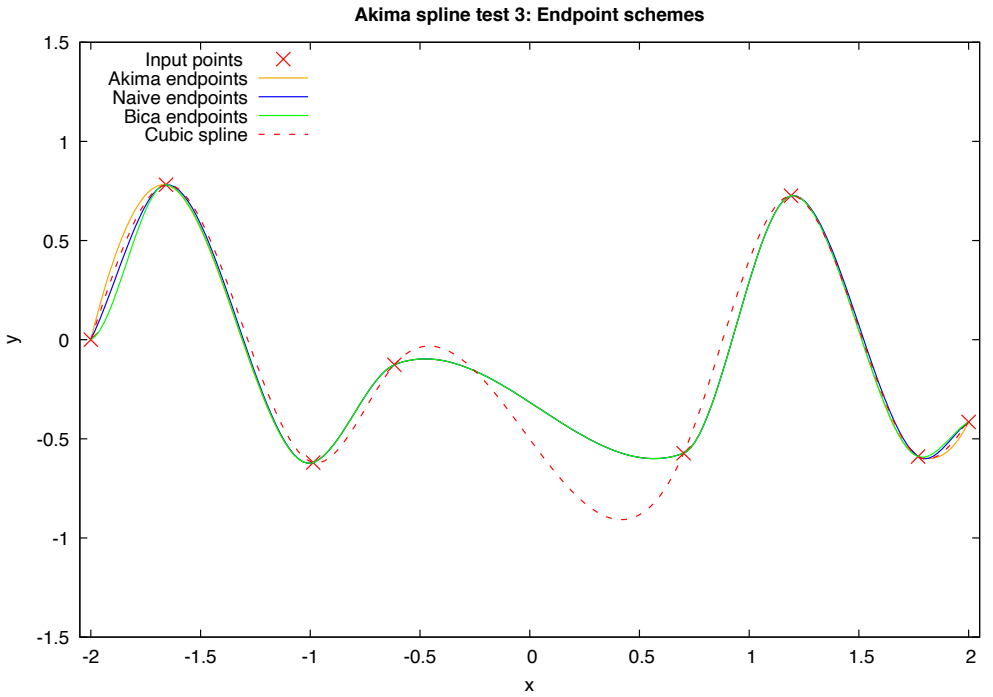

Figure 4: The three different end-point schemes used on a dataset with irregular oscillations. As expected the difference is only in the first two and last two intervals. The intervals between them are identical. For comparison the cubic spline interpolation is also shown.

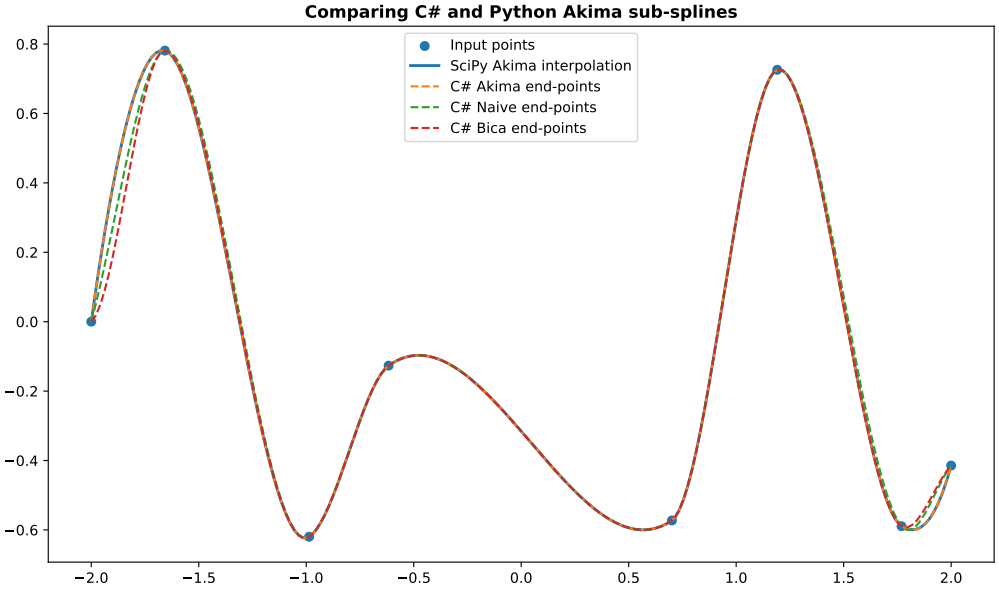

Figure 5: Comparing the input data set used in 5 with interpolation values from Python's `scipy.interpolate.Akima1DInterpolator`-routine. First of all the comparison verifies that my routine is calculating the sub-splines correctly. Secondly, I was curious on which end-point scheme is used by `SciPy`. Not surprisingly it turns out to be Akima's own suggestion of doing the algorithm on a larger set of p_is with two additional elements added at either end.