Akima Interpolation for Nonuniform 1D Data

David Eberly, Geometric Tools, Redmond WA 98052
https://www.geometrictools.com/

Created: August 13, 2013
Last Modified: September 11, 2020

Contents

This document describes how to generate a piecewise cubic polynomial interpolator for a collection of samples $\{(x_i, f_i)\}_{i=0}^{n-1}$ for an unknown underlying function $f(x)$. The interpolator is globally C^1-continuous. The f_i are estimates of the function at locations x_i; that is, $f_i \doteq f(x_i)$. The x-inputs are strictly ordered: $x_i < x_{i+1}$ for all i.

1 Cubic Polynomial Interpolation

Let $p(t) = c_0 + c_1 t + c_2 t^2 + c_3 t^3$ be a cubic polynomial for $t \in [0, 1]$. The coefficients are uniquely determined by specifying the polynomial values and the first derivative values at the endpoints. That is, let $p(0) = v_0$, $p'(0) = v_0'$, $p(1) = v_1$, and $p'(1) = v_1'$ for known quantites v_0, v_0', v_1, and v_1'. We have four linear equations in the four unknown coefficients. The solution is

$$c_0 = v_0, \;\; c_1 = v_0', \;\; c_2 = 3(v_1 - v_0 - v_0') - (v_1' - v_0'), \;\; c_3 = (v_1' - v_0') - 2(v_1 - v_0 - v_0')$$

We can construct two cubic polynomials that share an endpoints For example, let $q(t) = d_0 + d_1 t + d_2 t^2 + d_3 t^3$ for $t \in [1, 2]$. The coefficients are uniquely determined by specifying polynomial values and derivatives at the endpoints: $q(1) = p_1$, $q'(1) = p_1'$, $q(2) = p_2$, $q'(2) = p_2'$. The piecewise polynomial function

$$r(t) = \begin{cases} p(t), & t \in [0, 1] \\ q(t), & t \in [1, 2] \end{cases}$$

has a continuous derivative because $p(1) = q(1)$ and $p'(1) = q'(1)$.

Generally, we can construct a sequence of cubic polynomials, each consecutive pair sharing an endpoint. The piecewise polynomial function has a continuous derivative. Also, the intervals on which the polynomials are defined are not required to have the same length (which was 1 in the previous example). Using the notation for our samples, we want a sequence of cubic polynomials $p_i(x)$ for $0 \le i \le n-2$, each defined for $x \in [x_i, x_{i+1}]$. The polynomials must match the function samples, $p_i(x_i) = f_i$ for $0 \le i \le n-2$ and $p_{n-2}(x_{i+1}) = f_{n-1}$. We do not have derivative estimates f_i', but if we did, we also need $p_i'(x_i) = f_i'$ for $0 \le i \ne n-2$ and $p_{n-2}'(x_{i+1}) = f_{n-1}'$.

We can compute the polynomial coefficients one at a time by using a normalized variable. Specifically, for $x \in [x_i, x_{i+1}]$, define $u = (x - x_i)/(x_{i+1} - x_i) \in [0, 1]$. The polynomial may be written as

$$p_i(u) = c_{i0} + c_{i1} u + c_{i2} u^2 + c_{i3} u^3$$

To evaluate the piecewise polynomial at a value $x \in [x_0, x_{n-1}]$, compute the index i for the subinterval $[x_i, x_{i+1}]$ containing x, compute $u = (x - x_i)/(x_{i+1} - x_i)$, and then evaluate $p_i(u)$.

2 Estimates of the First Derivative

The construction of the piecewise polynomial function requires derivative estimates at the samples. These are typically not available in applications, so you must construct them from your function samples. To estimate f_i', you must use finite differences of function samples at neighboring samples. Not knowing the quality of the function samples, the simplest algorithm is to use a centered difference,

$$f_i' = \frac{f_{i+1} - f_{i-1}}{x_{i+1} - x_{i-1}}$$

The centered difference are used for all interior samples, but at the endpoints we use one-sided estimates,

$$f_0' = \frac{f_1 - f_0}{x_1 - x_0}, \quad f_{n-1}' = \frac{f_{n-1} - f_{n-2}}{x_{n-1} - x_{n-2}}$$

Larger neighborhoods may be used; see Derivative Approximation by Finite Differences.

3 Akima Estimates of the First Derivative

A more general formulation for estimating first derivatives is to use a weighted average of one-sided first derivatives,

$$f_i' = a_i \frac{f_i - f_{i-1}}{x_i - x_{i-1}} + b_{i+1} \frac{f_{i+1} - f_i}{x_{i+1} - x_i}$$

where $a_i \geq 0$, $b_{i+1} \geq 0$, and $a_i + b_{i+1} = 1$. The choice of weights can be tied to knowledge you have about the quality of the function samples. Observe that if $a_i = (x_i - x_{i-1})/(x_{i+1} - x_{i-1})$ and $b_{i+1} = (x_{i+1} - x_i)/(x_{i+1} - x_{i-1})$, we get the centered finite difference estimate. The a_i and b_{i+1} measure the relative location of x_i in the interval $[x_{i-1}, x_{i+1}]$.

The Akima estimates use weights that depend on how neighboring one-sided differences vary. The neighborhood consists of 5 samples. Define the one-sided differences

$$\sigma_{i+2} = \frac{f_{i+1} - f_i}{x_{i+1} - x_i}, \quad 0 \leq i \leq n-2$$

with boundary values

$$\sigma_0 = 2\sigma_1 - \sigma_2, \quad \sigma_1 = 2\sigma_2 - \sigma_3, \quad \sigma_{n+1} = 2\sigma_n - \sigma_{n-1}, \quad \sigma_{n+2} = 2\sigma_{n+1} - \sigma_n$$

Choose

$$a_i = \frac{|\sigma_{i+3} - \sigma_{i+2}|}{|\sigma_{i+3} - \sigma_{i+2}| + |\sigma_i - \sigma_{i+1}|}, \quad b_i = \frac{|\sigma_i - \sigma_{i+1}|}{|\sigma_{i+3} - \sigma_{i+2}| + |\sigma_i - \sigma_{i+1}|}$$

in which case the derivative estimates for use in the polynomial constructions are

$$f_i' = \frac{|\sigma_{i+3} - \sigma_{i+2}|\sigma_{i+1} + |\sigma_i - \sigma_{i+1}|\sigma_{i+2}}{|\sigma_{i+3} - \sigma_{i+2}| + |\sigma_i - \sigma_{i+1}|}$$

Observer that the denominator can be zero, in which case the weights are indeterminate forms $(0/0)$. When the denominator is zero, use the centered finite difference for f_i' (at an interior value, but use the one-sided differences at boundary values).

4 Implementation

An implementation is provided in the files IntpAkima1.h and IntpAkimaNonuniform1.h. Sample code for the interpolation is listed next.

```cpp
#include <Mathematics/IntpAkimaNonuniform1.h>
using namespace gte;

int main ()
{
    // Construct a data set with 10 samples.  Generally, X[i] can be any
    // numbers as long as they are increasing.  For simplicity, I just
    // chose them to be consecutive integers.
    int const numSamples = 10;
    double X[numSamples] = { 0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0 };
    double F[numSamples] =
    {
        0.13547700429678050,
        0.83500858999457950,
        0.96886777112423139,
        0.22103404298270490,
        0.30816705050700327,
        0.54722059636785192,
        0.18838197604718110,
        0.99288130191780666,
        0.99646132554800870,
        0.96769493701050258
    };

    // The interpolator does not take ownership of the input data, so it
    // does not matter whether X and F are stack variables or dynamically
    // allocated.
    IntpAkimaNonuniform1<double> interpolator(numSamples, X, F);

    // Interpolate at a number t with X[0] <= t <= X[numSamples-1].
    double t, function, derivative;

    t = 3.0;
    function = interpolator(t);  // or interpolator(0, t)
    derivative = interpolator(1, t);
    // function = 0.22103404298270490 = F[3]
    // derivative = -0.035590430593744664

    t = 3.1415927;
    function = interpolator(0, t);  // or interpolator(t)
    derivative = interpolator(1, t);
    // function = 0.21904360792075869
    // derivative = 0.0067899310784779107

    return 0;
}
```