Cubic spline interpolation

by

Jens Hee
https://jenshee.dk

June 2019

Change log

21. June 2019

1. Document started.

7. March 2020

1. Meta data added.

Contents

Chapter 1

Cubic spline interpolation

Cubic spline interpolation requires all data samples to be available beforehand and is thus not suitable for sample rate change as the methods described in [1].
The interpolation is done by fitting a third degree polynomial to each interval such that a continuous curve with continuous first and second derivatives is obtained. Moreover the second derivative is required to be zero at the endpoints.
If the number of samples is n then there are $n-1$ polynomials each having four parameters and the problem can in theory be solved by solving $4(n-1)$ linear equations. Unfortunately the set of equations is ill-conditioned and large errors may be introduced for large values of n. A more attractive method is given in [2] and is summarized in the following.
First the second derivative for all samples is calculated from the following set of $n-2$ linear equations:

$$(x_i - x_{i-1})s^{''}(x_{i-1}) + 2(x_{i+1} - x_{i-1})s^{''}(x_i) + (x_{i+1} - x_i)s^{''}(x_{i+1}) = 6[s(x_i, x_{i+1}) - s(x_{i-1}, x_i)]$$

where:
$i = 2, 3, ..., n-1$
$s^{''}(x_i)$ is the second derivative at sample x_i. $s^{''}(x_1) = s^{''}(x_n) = 0$
$s(x_i, x_{i+1}) = (y_{i+1} - y_i)/(x_{i+1} - x_i)$.
(x_i, y_i) are the given samples.

The set of equations can be solved using Gauss elimination, but the iterative method given in [2] is more efficient. See also the program example below.
Once the $s^{''}(x_i)$ are known, $s^{''}(x)$ can be calculated easily for any x in (x_i, x_{i+1}) since it is a linear function in each interval:

$$s^{''}(x) = s^{''}(x_i) + (x - x_i)s^{''}(x_i, x_{i+1})$$

Finally it can be shown that $s(x)$ can be found from:

$$s(x) = s(x_i) + (x - x_i)s(x_i, x_{i+1}) + (x - x_i)(x - x_{i+1})s(x, x_i, x_{i+1})$$

where:

$$s(x, x_i, x_{i+1}) = \frac{1}{6}[s^{''}(x_i) + s^{''}(x) + s^{''}(x_{i+1})]$$

for x in (x_i, x_{i+1}).

```
void Spline()
{
    double[] x = new double[] { 0, 1, 2.2, 3, 4.4, 5 };
    double[] y = new double[] { 1, 3, 10, 7, 4, 0 };
    int N = x.Length;
    int N1 = N - 1;
    double[] t = new double[N1*10+1];

    double[] c = new double[N1];
    double[] b = new double[N1];
    double[] s2 = new double[N];

    double h_1 = x[1] - x[0];
    double dely_1 = (y[1] - y[0]) / h_1;

    for (int i = 1; i < N1; i++)
    {
        double h = x[i + 1] - x[i];
        double dely = (y[i + 1] - y[i]) / h;
        double h2 = h_1 + h;
        double delsqy = (dely - dely_1) / h2;
        c[i] = 3 * delsqy;
        b[i] = 0.5 * h_1 / h2;
        s2[i] = 2 * delsqy;

        h_1 = h;
        dely_1 = dely;
    }

    s2[0] = 0;
    s2[N1] = 0;
    double omega = 4 * (2 - Math.Sqrt(3));
    double eta;
    do
    {
        eta = 0;
        for (int i = 1; i < N1; i++)
        {
            double w = (c[i] - b[i] * s2[i - 1] - (0.5 - b[i]) * s2[i + 1] - s2[i]) *
                       omega;
            if (Math.Abs(w) > eta)
                eta = Math.Abs(w);
            s2[i] += w;
        }
    }
    while (eta >= 0.001);

    double[] ss = new double[t.Length];

    for (int i = 0; i < N1; i++)
        for (int j = 0; j < 10; j++)
        {
            double h = x[i + 1] - x[i];
            double t1 = x[i] + j / 10.0 * h;
            t[10 * i + j] = t1;
            double t2 = t1 - x[i];
            double t3 = t1 - x[i + 1];
            double s2x = s2[i] + t2 * (s2[i + 1] - s2[i]) / h;
            ss[10 * i + j] = y[i] + t2 * (y[i + 1] - y[i]) / h +
                             t2 * t3 * 1.0 / 6 * (s2[i] + s2x + s2[i + 1]);
        }
}
```

Bibliography

[1] Jens Hee, ”Interpolation for sample rate change”, http://jenshee.dk, June 2019.

[2] T. N. E Greville, ”Spline functions, interpolation, and numerical quadrature”, pp. 156 - 168 in *Mathematical methods for digital computers Volume II*, John Wiley & Sons, Inc., New York.