Cubic Spline Interpolation
MAE 5093

Charles O'Neill

28 May 2002

Abstract

A cubic spline routine was developed for unequally spaced sequential data points. Cubic spline theory is reviewed. A Visual Basic computer program in Excel was created to fit a spline to input data points. Three testcases are used to validate the routine. Conclusions regarding the cubic spline routine are made.

Introduction

The objective is to fit a cubic spline to data points. A typical curve fit involves forming one equation through all n points. In contrast, a spline allowing each segment to have a unique equation while still constraining the curve fit to the data properties.

This paper will develop the governing equations for a cubic spline. A computer implementation using Visual Basic will be presented. Three testcases will validate the spline method and the computer code. Finally, conclusions will be discussed.

Theory

Spline theory is simple. Over *n* intervals, the routine fits *n* equations subject to the boundary conditions of *n+1* data points. The derivations of Lilley[1] and Wheatly[2] are used. The derivation assumes a functional form for the curve fit. This equation form is simplified and then solved for the curve fit equation.

The assumed form for the cubic polynomial curve fit for each segment is,

$$y = a_i (x - x_i)^3 + b_i (x - x_i)^2 + c_i (x - x_i) + d_i$$

where the spacing between successive data points is

$$h_i = x_{i+1} - x_i$$

The cubic spline constrains the function value, 1st derivative and 2nd derivative. The routine must ensure that $y(x)$, $y'(x)$ and $y''(x)$ are equal at the interior node points for adjacent segments. Substituting a variable *S* for the polynomial's second derivative reduces the number of equations from *a, b, c, d* for each segment to only *S* for each segment.

For the i^{th} segment, the S governing equation is,

$$h_{i-1} S_{i-1} + (2h_{i-1} + 2h_i) S_i + h_i S_{i+1} = 6\left(\frac{y_{i+1} - y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}} \right)$$

In matrix form, the governing equations reduce to a tri-diagonal form.

$$\begin{bmatrix} 2(h_1+h_2) & h_2 & & \\ h_2 & 2(h_2+h_3) & \ddots & \\ & \ddots & \ddots & h_{n-2} \\ & & h_{n-2} & 2(h_{n-2}+h_{n-1}) \end{bmatrix} \begin{bmatrix} S_2 \\ S_i \\ \vdots \\ S_{n-1} \end{bmatrix} = 6 \begin{bmatrix} \frac{y_3-y_2}{h_2} - \frac{y_2-y_1}{h_1} \\ \\ \vdots \\ \frac{y_n-y_{n-1}}{h_{n-1}} - \frac{y_{n-1}-y_{n-2}}{h_{n-2}} \end{bmatrix}$$

S_1 and S_n are zero for the *natural* spline boundary condition. If different boundary conditions are needed, the appropriate changes can be made to the governing equations.

Finally, the cubic spline properties are found by substituting into the following equations. These *a, b, c* and *d* values correspond to the polynomial definition for each segment.

$$a_i = (S_{i+1} - S_i)/6h_i$$
$$b_i = S_i/2$$
$$c_i = \frac{y_{i+1}-y_i}{h_i} - \frac{2h_iS_i + h_iS_{i+1}}{6}$$
$$d_i = y_i$$

Solution Method

A Visual Basic program was written in Excel to fit a cubic spline as described in the Theory section. The program is listed in the Appendix. The general program steps are given below.

1. **Problem Initialization** The program initializes the variables.
2. **Read in Data Values** The data values are read and the individual intervals are calculated.
3. **Determine S matrices** The influence coefficient values for the *S* matrix are determined for a natural spline. The constant matrix, *C*, is determined.
4. **Matrix Solver** A generic Tri-Diagonal-Matrix-Algorithm (TDMA) solver determines the *S* value at each interval.
5. **Calculate Cubic Parameters** The cubic parameters *a, b, c* and *d* are calculated at each interval from *S* and *h*.
6. **Write out** The program writes out the polynomial specification terms *a, b, c* and *d*.

Results

The cubic spline curve fit routine was validated with three testcases. The first is an equally spaced polynomial function. Next, an unequally spaced exponential function is fit. Finally, the cubic spline routine is tested on a *bump* function. The results are compared with known examples where possible.

Equal Data Spacing

This testcase fits a cubic spline to $f(x) = x^3 - 8$ along the x interval from 0 to 4. This problem is solved as an example in Lilley's[1] notes. The cubic spline routine determines the same S values as the example. Figure 1 shows a plot of the function and the cubic spline.

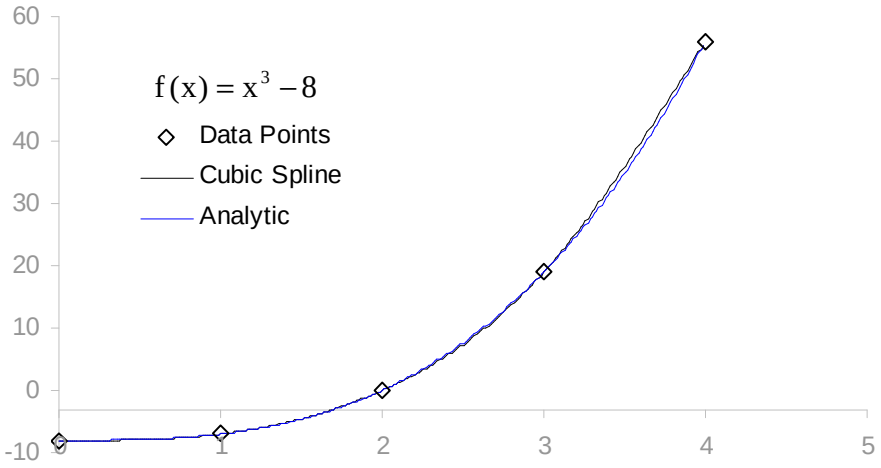

Figure 1. Equal Spacing

The spline was fit to 5 data points. The spline is *natural*, S equals zero at the ends. Overall fit is good except between x values of 3 and 4. This difference is caused by the *natural* spline boundary conditions at x=0 and 4. Changing the spline to reflect the correct 2nd derivative at x=4 would help the fit. Also, the *a, b, c* and *d* parameters would exactly recover the polynomial function.

Unequal Data Spacing

This testcase fits a cubic spline to $f(x) = 2e^x - x^2$ with unequal data spacing. Gerald and Wheatly[2] solve the problem in Example 3.6. The cubic spline routine finds the same S, a, b, c and d parameters as Gerald and Wheatly. Figure 2 shows a plot of the function and the cubic spline.

Notice that the spline creates a good fit between for segments with 2nd derivatives near zero. The third segment (x= 1.5 to 2.25) has the worst fit because of the natural spline boundary condition at point 4. This is the same 2nd derivative problem as discussed in the equal spacing testcase.

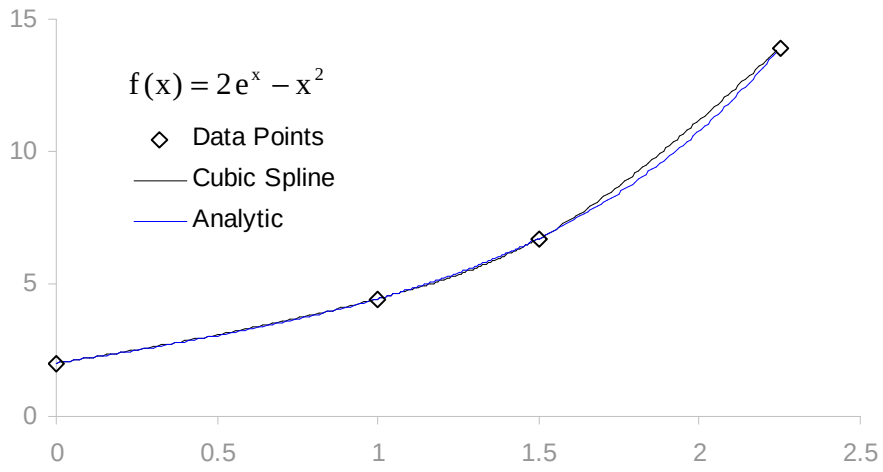

Figure 2. Unequal Spacing

Bump Function

The bump testcase fits a cubic to $f(x) = \cos(x)^{10}$ with seven data points. Gerald[2] used this function to illustrate problems with other interpolation methods. Figure 3 shows the function and the cubic spline fit.

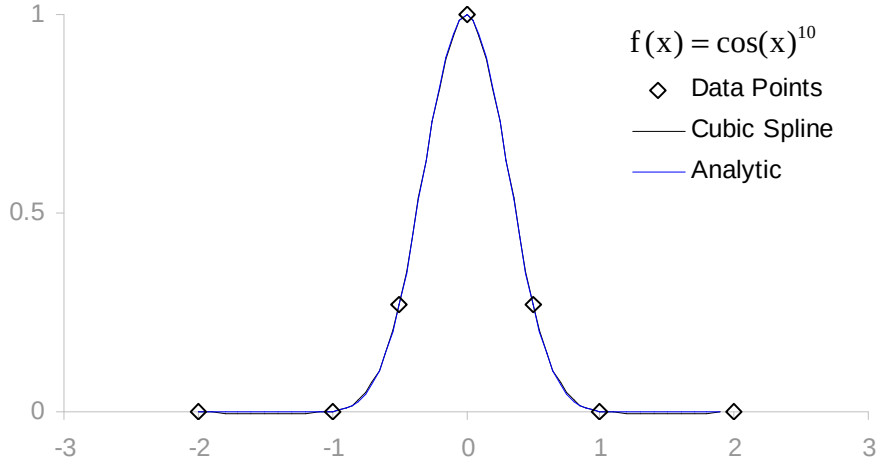

Figure 3. Bump Testcase

Visually, the cubic spline dips below into negative values between x= –2 to –1 and x= 1 to 2. The actual function remains positive. It appears that the best fit occurs in the middle. This makes sense because the end points are less constrained to the actual function when compared to the middle points.

Conclusions

A cubic spline curve fit routine was successfully implemented. A curve fit program was written with Visual Basic inside Excel. Three testcases were used to validate the curve fit routine.

The results show that the cubic spline provides an adequate curve fit for most data sets. Problems occurred at the end segments because of differences between the *natural* spline boundary condition and the function's actual 2nd derivative. Including 2nd derivative information for the endpoints would doubtlessly improve the curve fit.

Additionally, the spline method also allows for harmonic solutions by forcing the same function properties onto the end points. Variations on this harmonic method would be needed for unknown period oscillations or simultaneous multi-mode data points. Far from the end points, even the current code would yield good curve fits for harmonic data points.

The cubic spline is an easy to implement curve fit routine. Because the method involves connecting individual segments, the cubic spline avoids oscillation problems in the curve fit. Overall, the cubic spline provides a good curve fit for arbitrary data points.

References

[1] Lilley, D. G., *Numerical Methods*, Stillwater, OK, 2002. NOT AVAILABLE

[2] Gerald, C., and Wheatley, P., *Applied Numerical Analysis*, Addison-Wesley, 1994.

Comments and Thanks (July 2008)

Special thanks to C. Selover and Dr. M. Maixner for finding and correcting typos and bugs.

For a forgotten project in a long-ago summer numerical method course, this project has attracted a surprising number of comments.

Please invest in a good numerical methods book. The Lilley[1] reference was a series of class notes which are not currently available. I suggest that you either use the Gerald reference or buy Lilley's forthcoming book. Hamming's Numerical Methods published by Dover is nice. Another reasonable reference is the Numerical Recipes series by Press, et al.

Visual Basic defines the array S(10) with elements from 0 to 10. For S, I was only using elements 1 to 10, which accounts for the "Shift to TDMA coordinate system" code (The TDMA matrix inversion starts at element 0). This indexing method is sub-optimal.

Appendix: Computer Code

```
Option Explicit
' Cubic Spline
' Project 1 MAE 5093
' 5-28-02
' Copyright (c) 2002 Charles O'Neill
'
' Permission is hereby granted, free of charge, to any person
' obtaining a copy of this software and associated documentation
' files (the "Software"), to deal in the Software without
' restriction, including without limitation the rights to use,
' copy, modify, merge, publish, distribute, sublicense, and/or sell
' copies of the Software, and to permit persons to whom the
' Software is furnished to do so, subject to the following
' conditions:
'
' The above copyright notice and this permission notice shall be
' included in all copies or substantial portions of the Software.
'
' THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
' EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
' OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
' NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
' HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
' WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
' FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
' OTHER DEALINGS IN THE SOFTWARE.

Sub Main()

'Data sizing
Dim x(10) As Double, y(10) As Double, norder As Integer
Dim h(10) As Double
Dim i As Integer, j As Integer
'Cubic sizing
Dim S(10) As Double
'TDMA sizing (share A,B,C,D with cubic)
Dim B(10) As Double, D(10) As Double, A(10) As Double, C(10) As Double, R As Double
Dim ntdma As Integer

'Setup Lagrange polynomial ------------------------------------------
'Read in data-point order
norder = ActiveSheet.Cells(3, 3)

'Read in Data Values
For i = 1 To norder
    x(i) = ActiveSheet.Cells(4 + i, 3)
    y(i) = ActiveSheet.Cells(4 + i, 4)
Next i

'Determine the width of the ith interval
For i = 1 To (norder - 1)
    h(i) = x(i + 1) - x(i)
    ActiveSheet.Cells(4 + i, 5) = h(i)
Next i

'Set S matrix influence coefficients for natural spline
For i = 2 To (norder - 1)
    j = i - 1 'Shift to TDMA coordinate system
    D(j) = 2 * (h(i - 1) + h(i))
    A(j) = h(i) 'Ignore A(norder)
    B(j) = h(i - 1) 'Ignore B(0)
```

```
Next i

'Set Constant Matrix C
For i = 2 To (norder - 1)
    j = i - 1 'Shift to TDMA coordinate system
    C(j) = 6 * ((y(i + 1) - y(i)) / h(i) - (y(i) - y(i - 1)) / h(i - 1))
Next i

' Max tdma length
    ntdma = norder - 2

'TDMA ----------------------------------
' Upper Triangularization
For i = 2 To ntdma
    R = B(i) / D(i - 1)
    D(i) = D(i) - R * A(i - 1)
    C(i) = C(i) - R * C(i - 1)
Next i
' Directly set the last C
    C(ntdma) = C(ntdma) / D(ntdma)
' Back Substitute
For i = (ntdma - 1) To 1 Step (-1)
    C(i) = (C(i) - A(i) * C(i + 1)) / D(i)
Next i
'End of TDMA ------------------------------

'Switch from C to S
For i = 2 To (norder - 1)
    j = i - 1 'Shift from TDMA coordinate system
    S(i) = C(j)
Next i
'End conditions
S(1) = 0
S(norder) = 0

'Calculate cubic ai,bi,ci and di from S and h
For i = 1 To (norder - 1)
    A(i) = (S(i + 1) - S(i)) / (6 * h(i))
    B(i) = S(i) / 2
    C(i) = (y(i + 1) - y(i)) / h(i) - (2 * h(i) * S(i) + h(i) * S(i + 1)) / 6
    D(i) = y(i)
Next i

'Write out S,a,b,c,d Values
For i = 1 To norder
    ActiveSheet.Cells(4 + i, 7) = S(i)
    ActiveSheet.Cells(4 + i, 8) = A(i)
    ActiveSheet.Cells(4 + i, 9) = B(i)
    ActiveSheet.Cells(4 + i, 10) = C(i)
    ActiveSheet.Cells(4 + i, 11) = D(i)
Next i
End Sub
```

```
Sub plotter()
'Data sizing
Dim x(10) As Double, y(10) As Double, norder As Integer, nstep As Integer
Dim h(10) As Double
Dim i As Integer, j As Integer, step As Integer
Dim B(10) As Double, D(10) As Double, A(10) As Double, C(10) As Double
Dim xs As Double, ys As Double

'Read in data-point order
norder = ActiveSheet.Cells(3, 3)

'Read in Cubic properties
For i = 1 To norder
    x(i) = ActiveSheet.Cells(4 + i, 3)
    h(i) = ActiveSheet.Cells(4 + i, 5)
    A(i) = ActiveSheet.Cells(4 + i, 8)
    B(i) = ActiveSheet.Cells(4 + i, 9)
    C(i) = ActiveSheet.Cells(4 + i, 10)
    D(i) = ActiveSheet.Cells(4 + i, 11)
Next i

'Read in steps
nstep = ActiveSheet.Cells(18, 3)

'Determine and write out x,Y
step = 0
For i = 1 To (norder - 1) 'Discrete function step
    For j = 1 To nstep
        step = step + 1
        xs = x(i) + (h(i) / nstep) * (j - 1)
        ys = A(i) * (xs - x(i)) ^ 3 + B(i) * (xs - x(i)) ^ 2 + C(i) * (xs - x(i)) +
D(i)
        ActiveSheet.Cells(step, 15) = xs
        ActiveSheet.Cells(step, 16) = ys
    Next j
Next i

End Sub
```