14. Spline Interpolations - Cubic Splines

Cheng Peng

West Chester University

Contents

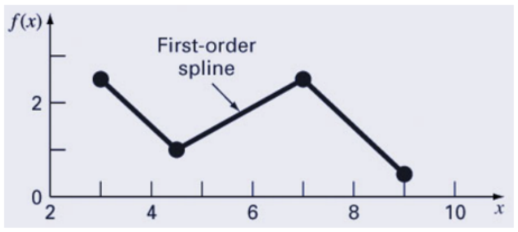
1 Introduction

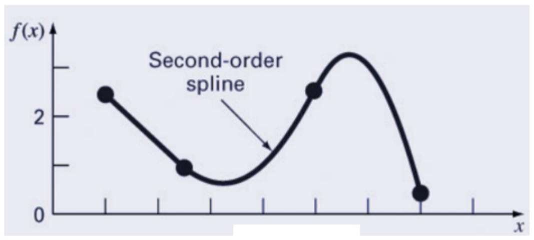
As we have seen from the previous note on linear and quadratic spline interpolation, splines are functions that are piece-wise polynomials. The coefficients of the polynomial differ from interval to interval, but the order of the polynomial is the same. An essential feature of splines is that function is continuous - i.e. has no breaks on the boundaries between two adjacent intervals. That is, they create smooth curves out of irregular data points.

In this note, we use the same idea to discuss cubic spline interpolation.

2 Cubic Splines

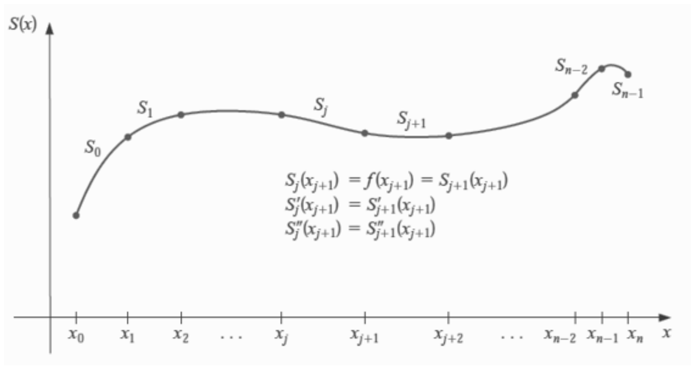
A cubic spline is a piece-wise cubic polynomial function. We will use the same logical process to develop the algorithm to find the cubic polynomials based on given knots.





2.1 Formulation of Cubic Spline

Suppose that we want to approximate $y = f(x)$ by cubic spline function $S(x)$. For a sampled set of points on the curve of $f(x)$: $\{(x_0, y_0).(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$.


$S(x)$
S_0
S_1
S_j
S_{j+1}
S_{n-2}
S_{n-1}
$S_j(x_{j+1}) = f(x_{j+1}) = S_{j+1}(x_{j+1})$
$S_j'(x_{j+1}) = S_{j+1}'(x_{j+1})$
$S_j''(x_{j+1}) = S_{j+1}''(x_{j+1})$
x_0 x_1 x_2 ... x_j x_{j+1} x_{j+2} ... x_{n-2} x_{n-1} x_n x

Assume further that $x_0 < x_1 < x_2 < \cdots < x_n$. On each interval $[x_0, x_1), [x_1, x_2), \cdots, [x_{n-1}, x_n]$, $S(x)$ is given by a different cubic polynomial. Let $S_i(x)$ be the cubic polynomial that represents $S(x)$ on $[x_i, x_{i+1}]$. Thus

$$S(x) = \begin{cases} S_0(x), & x \in [x_0, x_1) \\ S_1(x), & x \in [x_1, x_2) \\ \vdots & \vdots \\ S_{n-1}(x), & x \in [x_{n-1}, x_n] \end{cases}$$

Where

$$S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3$$

for $j = 0, 1, \cdots, n-1$. There are $4n$ unknowns to be determined subject to the following conditions:

Condition 1: $S(x_j) = f(x_j)$ for each $j = 0, 1, \cdots, n$; *($n+1$ equations)*

Condition 2: $S_{j+1}(x_{j+1}) = S_j(x_{j+1})$ for each $j = 0, 1, \cdots, n-2$; *($n-1$ equations)*

Condition 3: $S_{j+1}'(x_{j+1}) = S_j'(x_{j+1})$ for each $j = 0, 1, \cdots, n-2$; *($n-1$ equations)*

Condition 4: $S_{j+1}''(x_{j+1}) = S_j''(x_{j+1})$ for each $j = 0, 1, \cdots, n-2$; *($n-1$ equations)*

Condition 5: One of the following boundary conditions:

a) $S''(x_0) = S''(x_n) = 0 \Rightarrow$ **Natural Spline**. *(2 equations)*

b) $S'(x_0) = f'(x_0)$ and $S'(x_n) = f'(x_n) \Rightarrow$ **Clamped Spline**. *Remark: since $f(x)$ is unknown, we need to provide the two slopes according to the trend of the pattern of the scatter plot of the data. (2 equations)*

Example 1. Calculate **Cubic Natural Splines** based on given data $\{(1,5),(2,11),(4,8)\}$ are sampled from an unknown function $y=f(x)$. Find the approximated values of $f(1.5)$ and $f'(2)$ based on the cubic smoothing spline.

Solution: Let $S_0(x)=a_0+b_0(x-1)+c_0(x-1)^2+d_0(x-1)^3$ be the spline function on interval $[1,2)$ and $S_1(x)=a_1+b_1(x-2)+c_1(x-2)^2+d_1(x-2)^3$ be the spline function on interval $[2,4)$. We need the following 8 equations to determine the two cubic spline functions.

- The two splines pass individual points yielding
  1. $a_0=5$
  2. $a_0+b_0+c_0+d_0=11$
  3. $a_1=11$
  4. $a_1+2b_1+4c_1+8d_1=8$
- First order smoothness: $b_0+2c_0(x-1)+3d_0(x-1)^2=b_1+2c_1(x-2)+3d_1(x-2)^2$
  5. $b_0+2c_0+3d_0=b_1$
- Second order smoothness: $2c_0+6d_0(x-1)=2c_1+6d_1(x-2)$
  6. $2c_0+6d_0=2c_1$
- Boundary Conditions (natural spline):
  7. $d_0=0$
  8. $d_1=0$

We obtain the following system of equations

$$\begin{cases}1a_0+0b_0+0c_0+0d_0+0a_1+0b_1+0c_1+0d_1=5\\ 1a_0+1b_0+1c_0+1d_0+0a_1+0b_1+0c_1+0d_1=11\\ 0a_0+0b_0+0c_0+0d_0+1a_1+0b_1+0c_1+0d_1=11\\ 0a_0+0b_0+0c_0+0d_0+1a_1+2b_1+4c_1+8d_1=8\\ 0a_0+1b_0+2c_0+3d_0+0a_1-1b_1+0c_1+0d_1=0\\ 0a_0+0b_0+1c_0+3d_0+0a_1+0b_1-1c_1+0d_1=0\\ 0a_0+0b_0+0c_0+1d_0+0a_1+0b_1+0c_1+0d_1=0\\ 0a_0+0b_0+0c_0+0d_0+0a_1+0b_1+0c_1+1d_1=0\end{cases}$$

The matrix representation of the above system is given by

$$\begin{bmatrix}1&0&0&0&0&0&0&0\\1&1&1&1&0&0&0&0\\0&0&0&0&1&0&0&0\\0&0&0&0&1&2&4&8\\0&1&2&3&0&-1&0&0\\0&0&1&3&0&0&-1&0\\0&0&0&1&0&0&0&0\\0&1&0&0&0&01&0&1\end{bmatrix}\times\begin{bmatrix}a_0\\b_0\\c_0\\d_0\\a_1\\b_1\\c_1\\d_1\end{bmatrix}=\begin{bmatrix}5\\11\\11\\8\\0\\0\\0\\0\end{bmatrix}$$

The solution to the equation is

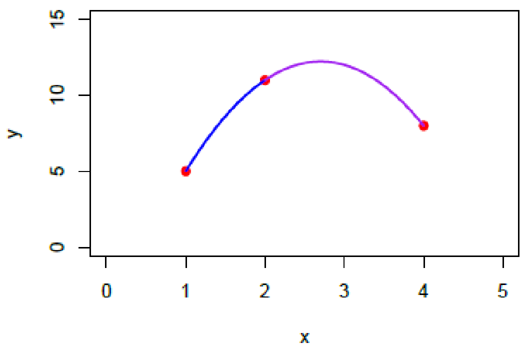
$$\begin{bmatrix} a_0 \\ b_0 \\ c_0 \\ d_0 \\ a_1 \\ b_1 \\ c_1 \\ d_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 4 & 8 \\ 0 & 1 & 2 & 3 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 3 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 01 & 0 & 1 \end{bmatrix}^{-1} \times \begin{bmatrix} 5 \\ 11 \\ 11 \\ 8 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 5 \\ 13.5 \\ -7.5 \\ 0 \\ 11 \\ 13.5 \\ -7.5 \\ 0 \end{bmatrix}.$$

We use R to solve the above equation.

Thais, $(a_0, b_0, c_0, d_0, a_1, b_1, c_1, d_1) = (5, 8.5, -2.5, 0, 11, 3.5, -2.5, 0)$. Therefore, the two resulting cubic spline functions (reduce to quadratic functions due to the choice of the natural spline) are

$$S(x) = \begin{cases} 5 + 8.5(x-1) - 2.5(x-1)^2 & \text{for } x \in [1, 2) \\ 11 + 3.5(x-2) - 2.5(x-2)^2 & \text{for } x \in [2, 4]. \end{cases}$$

Therefore, $f(1.5) \approx S_0(1.5) = 5 + 8.5 \times 0.5 - 2.5 \times 0.5^2 = 8.625$. $f'(2) = S_0'(2) = S_1'(2) = [11 + 3.5(x-2) - 2.5(x-2)^2]' = 3.5$.



Example 2: Calculate **Cubic Clamped Splines** based on given data points $\{(0, 1), (2, 2), (5, 0), (8, 0)\}$ are sampled from an unknown function $y = f(x)$.

Solution (sketch): We choose the slopes of the starting and ending points of $f(x)$ to be 2 and 1 respectively. *For convenience, we denote individual segments to have form $f_i(x) = a_i + b_i x + c_i x^2 + d_i x^3$*. Using the same set of conditions, we list the resulting equations of the coefficients of the four cubic polynomials according to each spline function in the following.

Segment 0: $f_0(x)$	
Slope at 0:	$b_0 = 2$
Curve through (0, 1):	$a_0 = 1$
Curve through (2, 2):	$a_0 + 2b_0 + 4c_0 + 8d_0 = 2$
Slopes match at join with f_1:	$b_0 + 4c_0 + 12d_0 - b_1 - 4c_1 - 12d_1 = 0$
Curvatures match at join with f_1:	$2c_0 + 12d_0 - 2c_1 - 12d_1 = 0$

Segment 1: $f_1(x)$	
Curve through (2, 2):	$a_1 + 2b_1 + 4c_1 + 8d_1 = 2$
Curve through (5, 0):	$a_1 + 5b_1 + 25c_1 + 125d_1 = 0$
Slopes match at join with f_2:	$b_1 + 10c_1 + 75d_1 - b_2 - 10c_2 - 75d_2 = 0$
Curvatures match at join with f_2:	$2c_1 + 30d_1 - 2c_2 - 30d_2 = 0$

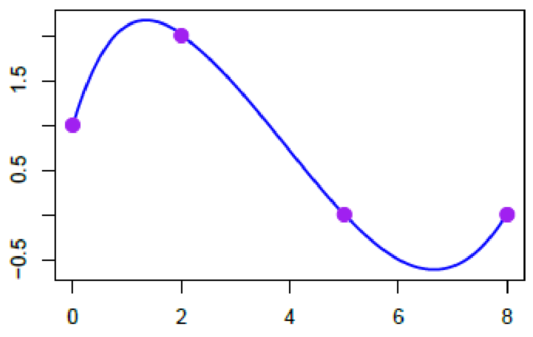
Segment 2: $f_2(x)$	
Curve through (5, 0):	$a_2 + 5b_2 + 25c_2 + 125d_2 = 0$
Curve through (8, 0):	$a_2 + 8b_2 + 64c_2 + 512d_2 = 0$
Slope at 8:	$b_2 + 16c_2 + 192d_2 = 1$

Rewriting the above system of equations in the matrix form, we have

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 4 & 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 4 & 12 & 0 & -1 & -4 & -12 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 12 & 0 & 0 & -2 & -12 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 4 & 8 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 5 & 25 & 125 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 10 & 75 & 0 & -1 & -10 & -75 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 30 & 0 & 0 & -2 & -30 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 5 & 25 & 125 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 8 & 64 & 512 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 16 & 192 \end{bmatrix} \begin{bmatrix} a_0 \\ b_0 \\ c_0 \\ d_0 \\ a_1 \\ b_1 \\ c_1 \\ d_1 \\ a_2 \\ b_2 \\ c_2 \\ d_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 2 \\ 0 \\ 0 \\ 2 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

The solution of the above system equations is $(a_0, b_0, c_0, d_0, a_1, b_1, c_1, d_1, a_2, b_2, c_2, d_2,) =$ (1, 2, -1.0395, 0.1447, 1.9201, 0.6199, -0.3494, 0.0297, 0.7018, 1.3509, -0.4956, 0.0395). Therefore, the cubic spline function is given explicitly by

$$S(x) = \begin{cases} 1 + 2x - 1.0339x^2 + 0.1447x^3 & \text{for } x \in [0, 2); \\ 1.9201 + 0.6199x - 0.3494x^2 + 0.0297x^3 & \text{for } x \in [2, 5); \\ 0.7018 + 1.3509x - 0.4956x^2 + 0.0395x^3 & \text{for } x \in [5, 8]. \end{cases}$$



2.2 Construction of Cubic Splines

CUBIC SPLINE FUNCTION: $S_j(x) = a_j + b_j(x-x_j) + c_j(x-x_j)^2 + d_j(x-x_j)^3$

OBJECTIVE: Find Coefficients: (a_i, b_i, c_i, d_i)**!**

Let $h_j = x_{j+1} - x_j$. Since there are $4n$ unknowns that need to be determined from the data, we need to set up $4n$ equations as follows:

1. From **condition 1**: $S_j(x_j) = y_j$ gives n equations: $a_j = f(x_j) = y_j$ for $j = 0, 1, 2, \cdots, n$.
2. From **Condition 2**: $S_{j+1}(x_{j+1}) = S_j(x_{j+1})$ gives $n-1$ equations: $a_{j+1} = a_j + b_j h_j + c_j h_j^2 + d_j h_j^3$ for $j = 0, 1, 2, \cdots, n-2$.
3. From **condition 3**: Note that $S_j'(x) = b_j + 2c_j(x-x_j) + 3d_j(x-x_j)^2$. The condition $S_{j+1}'(x_{j+1}) = S_j'(x_j)$ gives $n-1$ equations: $b_{j+1} = b_j + 2c_j h_j + 3d_j h_j^2$ for $j = 0, 1, 2, \cdots, n-2$.
4. From **condition 4**, $S_j''(x) = 2c_j + 6d_j(x-x_j)$: The condition $S_{j+1}''(x_{j+1}) = S_{j+1}''(x_j)$ gives $n-1$ equations: $c_{j+1} = c_j + 3d_j h_j$ for $j = 0, 1, 2, \cdots, n-2$.
5. **Boundary conditions**:
   a) $c_0 = c_n = 0$
   b) $S_n'(b) = f'(b) = b_n$ gives $h_{n-1}(c_{n-1} + 2c_n) = 3f'(b) - 3(a_n - a_{n-1})/h_{n-1}$ while $S_0'(a) = f'(a) = b_0$ gives the following equatione $h_0(2c_0 + c_1) = 3(a_1 - a_0)/h_0 - 3f'(a)$.

Instead of solving the system of $3n$ equations based on the given $n+1$ knots $\{(x_0, y_0), (x_1, y_1), \cdots, (x_n, y_n)\}$, we next simplify the original system to reduce it to a relatively small system. to summarize what we obtained earlier, we list the following $3n-2$ equations

A. $a_{j+1} = a_j + b_j h_j + c_j h_j^2 + d_j h_j^3$ for $j = 0, 1, 2, \cdots, n$.

B. $b_{j+1} = b_j + 2c_j h_j + 3d_j h_j^2$ for $j = 0, 1, 2, \cdots, n-1$.

C. $c_{j+1} = c_j + 3d_j h_j$ for $j = 0, 1, 2, \cdots, n-1$.

Note that a_j and h_j are known. Solve for d_j from (C), we have $d_j = (c_{j+1} - c_j)/(3h_j)$. Plug d_j into (A) and (B), and we have

D. $a_{j+1} = a_j + b_j h_j + h_j^2(2c_j + c_{j+1})/3$.

E. $b_{j+1} = b_j + h_j(c_j + c_{j+1})$

We can similarly solve b_j from (D) to get

F. $b_j = (a_{j+1} - a_j)/h_j - h_j(2c_j + c_{j+1})/3$ for $j = 0, 1, 2, \cdots, n-1$.

which implies

G. $b_{j-1} = (a_j - a_{j-1})/h_{j-1} - h_{j-1}(2c_{j-1} + c_j)/3$ for $j = 1, 2, \cdots, n-1$.

We re-write (re-index) (E) as follows

H. $b_j = b_{j-1} + h_{j-1}(c_{j-1} + c_j)$ for $j = 1, 2, 3, \cdots, n-1$. (see the change of the index).

We substitute b_{j-1} and b_j in (H) with the ones in (F) and (G) and obtain

I. $(a_{j+1}-a_j)/h_j - h_j(2c_j + c_{j+1})/3 = (a_j - a_{j-1})/h_{j-1} - h_{j-1}(2c_{j-1}+c_j)/3 + h_{j-1}(c_{j-1}+c_j)$ for $j = 1, 2, \cdots, n-1$, where c_j $(j = 1, 2, \ldots, n-1)$ are unknowns.

Finally, we end up with a linear system that involves unknowns $c_1, c_2, \cdots, c_{n-2}$ in the following standard form.

J.: $h_{j-1}c_{j-1} + 2c_j(h_{j-1}+h_j) + h_j c_{j+1} = 3(a_{j+1}-a_j)/h_j - 3(a_j - a_{j-1})/h_{j-1}$ for $j = 1, 2, \ldots, n-2$.

We next use the boundary condition **a)** to add two equations $c_0 = c_{n-1} = 0$ to (**J**). The resulting cubic spline is called the **NATURAL CUBIC SPLINE.**

The coefficient matrix of the system along with the two boundary conditions is given by

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ h_0 & 2(h_0+h_1) & h_1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & h_1 & 2(h_1+h_2) & h_2 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{bmatrix}$$

Since every row of A has three non-zero elements (except for the first and the last rows), S is also called **tridiagonal matrix**.

Definition A **diagonally dominant matrix** is the square matrix whose absolute value of the diagonal element is **greater than or equal to** the sum of the absolute value of corresponding off-diagonal elements.

Definition A **diagonally dominant matrix** is the square matrix whose absolute value of the diagonal element is **strictly greater than** the sum of the absolute value of corresponding off-diagonal elements.

Theorem 1 A strictly diagonally dominant complex matrix is non-singular.

Proof: One can use the concept of eigen theory and proof-by-contradiction to prove this theorem. The proof will not be given in this note.

Theorem 2: If f is defined at $a = x_0 < x_1 < \cdots < x_n = b$, then f has a unique natural cubic spline interpolated function $S(x)$.

Note that A is strictly diagonally dominant. Therefore, A is invertible. Therefore, there is a unique solution to the system. Therefore, the natural cubic spline function is unique.

The resulting matrix of the equation based on **(J)** and the two boundary conditions of natural cubic spline is given by

$$A \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{n-1} \\ c_n \end{bmatrix} = \begin{bmatrix} 0 \\ 3(a_2 - a_1)/h_1 - 3(a_1 - a_0)/h_0 \\ 3(a_3 - a_2)/h_2 - 3(a_2 - a_1)/h_1 \\ \vdots \\ 3(a_n - a_{n-1})/h_{n-1} - 3(a_{n-1} - a_{n-2})/h_{n-2} \\ 0 \end{bmatrix}$$

Using **theorem 2**, there is a unique solution

$$\begin{bmatrix} c_0 \\ c_2 \\ \vdots \\ c_{n-2} \\ c_{n-1} \end{bmatrix} = A^{-1} \begin{bmatrix} 0 \\ 3(a_2 - a_1)/h_1 - 3(a_1 - a_0)/h_0 \\ 3(a_3 - a_2)/h_2 - 3(a_2 - a_1)/h_1 \\ \vdots \\ 3(a_n - a_{n-1})/h_{n-1} - 3(a_{n-1} - a_{n-2})/h_{n-2} \\ 0 \end{bmatrix}$$

2.3 Cubic Spline Algorithm

Three key steps are needed in developing the algorithm.

1. Find the solution of equation (**J**) to find $c(c_0, c_1, \cdots, c_{n-1})$.
2. Using the backward substitution to find b_i and d_i $(i = 0, 1, \cdots, n-1)$

Because of multiple steps in simplification, the pseudo-code looks more complex than earlier algorithms.

Pseudo-code:

```
INPUT:   x        (x-coordinated of given knots)
         y        (y-coordinates of given knots)
         x.new    (new x-values for prediction)
OUTPUT:  coefficients
         y.new
STEP 1:  initialization
         A        (nxn zero matrix for solving c_i)
         Y        (nx1 zero matrix in the matrix equation J)
         n        (number of knots)
         m        (number of x.new)
         h = diff of adjacent input X
         aa       (coef of C-spline function)
         bb
         cc
         dd
STEP 2:  FOR i =2 TO (n-1) DO:
            A[i,(i-1):(i+1)] = (h[i-1], 2*(h[i-1]+h[i]), h[i])
            Y[i] = 3*(y[i+1]-y[i])/h[i] - 3*(y[i]-y[i-1])/h[i-1]
         ENDFOR
STEP 3:  Solve for cc  (coefficient of C-spline polynomials)
STEP 4:  aa = y
STEP 5:  FOR j = 1 TO (n-1) DO:
            dd[j] = (cc[i+1]-cc[i])/(3*h[i])
            bb[j] = (aa[i+1]-aa[i])/h[i] - h[i]*(2*cc[i] + cc[i+1])/3
         ENDFOR
STEP 6:  FOR k = 1 TO m DO:
            IF  (x[s] <= x.new[k] < x[s+1]) DO:
              y.new[k] = aa[i]+ bb[i]*(x.new[j] - x[i]) + cc[i]*(x.new[j] - x[i])^2 + dd[i]*(x.new[j] -
            ENDIF
         ENDFOR
STEP 7: RETURN y.new and C-spline coefficients
```

R Code

```r
###
Natural.CSpline = function(x,
                           y,
                           x.new){
  n = length(x)
  m = length(x.new)
  h = xvec[-1] - xvec[-n]
  A = matrix(rep(0,(n)^2), ncol=n)
  Y = rep(0,n)
  y.new = NULL
  Y[c(1,n)] = 0
  A[1,1] = 1
  A[n,n] = 1
  for (i in 2:(n-1)){
     A[i,(i-1):(i+1)] = c(h[i-1], 2*(h[i-1]+h[i]), h[i])
     Y[i] =3*(y[i+1]-y[i])/h[i]-3*(y[i]-y[i-1])/h[i-1]
  }
  cc = as.vector(solve(A)%*%Y)
  aa = y                 # input y values
  ID = 1:(n-1)
  bb = dd = NULL
  for (i in 1:(n-1)){
    dd[i] = (cc[i+1] - cc[i])/(3*h[i])
    bb[i] = (aa[i+1] - aa[i])/h[i] - h[i]*(2*cc[i] + cc[i+1])/3
  }
  cc = cc[-n]
  aa = aa[-n]
  coef = cbind(ID = ID, aa = aa, bb = bb, cc = cc, dd = dd)
  #predicted value
  dS = NULL                            # derivative of the C-spline.
  for (j in 1:length(x.new)){
  for (i in 1:n){
    if (x.new[j]<= x[i]){
       next
      }
     y.new[j] = aa[i] + bb[i]*(x.new[j] - x[i]) + cc[i]*(x.new[j] - x[i])^2 + dd[i]*(x.new[j] - x[i])^3
     dS[j] = bb[i] + 2*cc[i]*(x.new[j] - x[i]) +3* dd[i]*(x.new[j] - x[i])^2
   }
  }
 list(coef = coef, y.new = y.new, d.S = dS)
}
```

Example 3: Approximate $f(x) = \log(e^x + 2)$ using nodes $x = -1. - 0.5, 0, 0.5$.

```
xvec=c(-1,-0.5,0,0.5)
yvec=log(exp(xvec)+2)
###
Natural.CSpline(x = xvec, y = yvec, x.new = 0.25)
```

```
## $coef
##      ID        aa        bb         cc          dd
## [1,]  1 0.8619948 0.1756378 0.00000000  0.06565087
## [2,]  2 0.9580201 0.2248760 0.09847631  0.02828097
## [3,]  3 1.0986123 0.3445630 0.14089776 -0.09393184
##
## $y.new
## [1] 1.192091
##
## $d.S
## [1] 0.3973997
```

Example 4 (Example 2 of the textbook, page 150): Use the data points $(0, 1), (1, e), (2, e^2)$, and $(3, e^3)$ to form a natural spline $S(x)$ that approximates $f(x) = e^x$.

```
xvec=c(0, 1, 2, 3)
yvec=exp(xvec)
x.new = c(1, 1.5, 2)
###
Natural.CSpline(x = xvec, y = yvec, x.new = x.new)
```

```
## $coef
##      ID       aa       bb        cc         dd
## [1,]  1 1.000000 1.465998 0.0000000  0.2522842
## [2,]  2 2.718282 2.222850 0.7568526  1.6910714
## [3,]  3 7.389056 8.809770 5.8300668 -1.9433556
##
## $y.new
## [1] 2.718282 4.230304 7.389056
##
## $d.S
## [1] 2.222850 4.248006 8.809770
```

3 Clamped Splines

The only difference between natural splines and clamped splines is the use of boundary conditions. In the natural cubic spline, the boundary conditions are $c_0 - c_{n-1} = 0$.

The clamped splines use the (clamped) boundary conditions: $S'(x_0) = f'(x_0$ and $S'(x_n) = f'(x_n)$.

We will not develop an R function to implement the clamped cubic spline interpolation. Both R and MATLAB have built-in functions to perform clamped cubic spline interpolation.

4 Error Analysis

Theorem: Let $f \in C^4[a,b]$ with $\max_{a \le x \le b} |f(4)(x)| = M$. If S is the unique clamped cubic spline interpolant to f with respect to nodes $a = x_0 < x1 < \cdots < x_n = b$, then for all $x \in [a,b]$,

$$|f(x) - S(x)| \le \frac{5M \cdot \max_{0 \le x \le n-1} |x_{j+1} - x_j|^4}{384}$$

Using the above theorem, we can find the error bound of the clamped cubic spline.

The error bound of the natural spline is dependent on the 4-th order derivative of $f(x)$. It is more complicated than that of the clamped cubic spline. We will not introduce this in this course.

5 HW for Spline Methods

Using R/MATLAB code to do the following problems.

3.5.3(c,d).

3.5.11.

3.5.31.