Applied Programming

Piecewise Interpolation with Splines

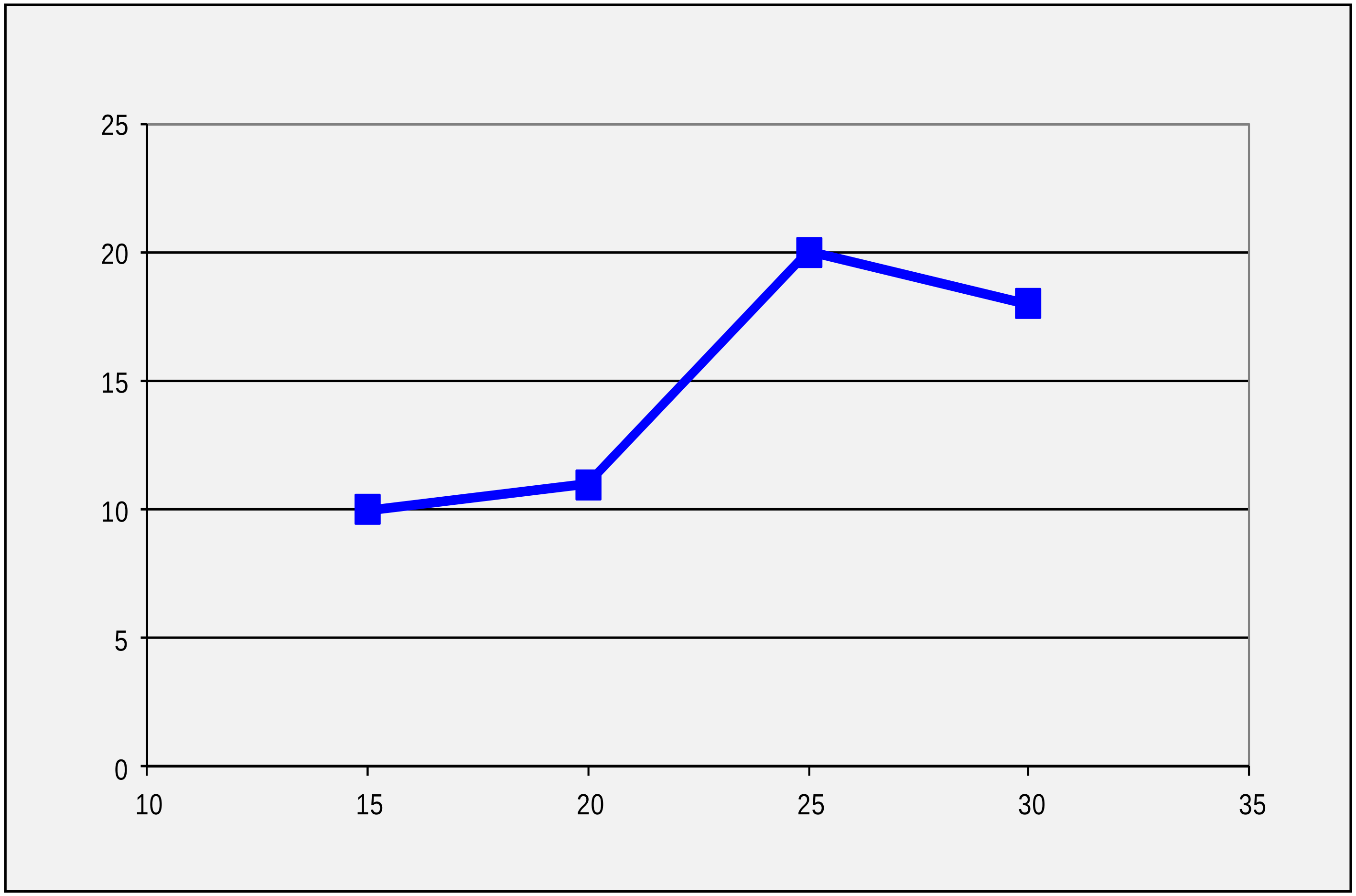
Piecewise Interpolation

- Interpolation with ***high order polynomials exhibit undesirable oscillatory behavior***
  - usually with polynomials of degree $n>4$

- One way to address this issue is:
  - Instead of finding a high order polynomial to interpolate all data points we could perform low order (e.g., up to 3) interpolation over multiple (non-overlapping) regions.

This is called **Piecewise Interpolation**

Example: Piecewise **Linear** Interpolation



Why this may be unsatisfactory ?

Piecewise Interpolation

Observations:

- A ***piecewise linear*** interpolant ***exhibits "kinks"*** at the (interior) interpolating points due to discontinuous first derivatives @ end points.
- ***Higher order polynomials*** (e.g., *piecewise quadratic*), allow us to ***eliminate these "kinks"*** by requesting that two contiguous pieces have continuous first derivatives

Why is this important ?

In robotic ***path planning*** and ***CNC*** (Computerized Numerical Control) *machining* it is necessary to have a path with a continuous (smooth) second derivative

Spline Interpolation

- ***Splines*** are ***piecewise continuous polynomials*** (of "low order").
- They are use to ***interpolate*** a set of points **AND** satisfy additional ***smoothness constraints***
- The polynomial order used in each piece can be:
  - 1^{st} order splines (piecewise lines)
  - 2^{nd} order splines (piecewise quadratic)
  - 3^{rd} order splines (piecewise cubic)
  - ….

Spline Interpolation

- ***Linear splines*** implement ***piecewise linear interpolation*** and guarantee ***continuity***

- ***Quadratic splines*** implement ***piecewise quadratic interpolation*** and allow for ***smooth first derivatives*** *(slope)* at the junctions.

- *Cubic splines* implement ***piecewise cubic interpolation*** and allow for ***smooth first and second derivatives*** *(curvature)* at the junctions.

The big Idea

- If we can create a series of 3rd order equations such that:
  - The **end** position of one equation matches up with the **start** of the next AND
  - The **1st and 2nd** derivatives are **continuous** (the same) at the transition point
- Then we can create a system that can map ANY number of points into a smooth series of equations
  - Useful for path planning (robots, cutting) and modeling complex systems (chemistry, process control, etc)

Cubic Spline Equations

- The general form of a cubic equations is:

$$s(z) = a + b\,z + c\,z^2 + d\,z^3$$

We will construct a family of cubic "spline" equations so, in general:

$$s_j(z) = a_j + b_j\,z + c_j\,z^2 + d_j\,z^3$$

Where j is our spline section

We will define z to be $(x - x_j)$, x_j is a constant associated with the ***starting x position of the spline***

Cubic Spline Interpolation

- Given the data set:

  $\{(x_0, y_0), \ldots (x_n, y_n)\}, \quad x_i \mathrel{!=} x_k \qquad \textit{for all } i \mathrel{!=} k$

A **cubic spline** $s(x)$ on $[x_0, y_n]$ is a pricewise function that on each subinterval $[x_j, x_{j+1}]$, $j = 0, 1, \ldots, n-1$ is defined as:

$$\boldsymbol{s_j(x) = a_j + b_j (x - x_j) + c_j (x - x_j)^2 + d_j (x - x_j)^3}$$

Note: "n" is the number of splines,

"n+1" is the number of points

Cubic Spline Interpolation

- A ***cubic spline*** is a piecewise function composed of ***n cubic polynomials***, each described by ***4 parameters (a,b,c,d)*** that satisfies the following constraints:

➢ Interpolation:	(n+1) eqs (at each point)
➢ Continuity	(n-1) eqs (at interior pts)
➢ Continuity of 1st derivative	(n-1) eqs (at interior pts)
➢ Continuity of 2nd derivative:	(n-1) eqs (at interior pts)
Total:	**4n – 2 equations**

Note that for a data set of ***n+1*** points we need a spline with ***n*** pieces

Cubic Spline Interpolation

➢ Interpolation:	(n+1) eqs (at each point)
➢ Continuity	(n-1) eqs (at interior pts)
➢ Continuity of 1st derivative	(n-1) eqs (at interior pts)
➢ Continuity of 2nd derivative:	(n-1) eqs (at interior pts)
Total:	**4n – 2 equations**

- All the ***equations*** that satisfy the above constraints ***are linear***. For a unique solution we ***need 4n independent equations***
- Have 4n-2 equations, we need 2 more equations (called boundary conditions)

Note that for a data set of ***n+1*** points we need a spline with ***n*** pieces

Cubic Splines Equations

The j^{th} cubic spline segment and its derivative

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

$$s_j'(x) = b_j + 2\,c_j(x - x_j) + 3\,d_j(x - x_j)^2$$

$$s_j''(x) = 2\,c_j + 6\,d_j(x - x_j)$$

Note: The "c" parameter is key!

The goal to create a family of equations in "c", the coefficient of the 2nd derivative and solve the matrix

Splines – big picture

- Link a series of 3rd order lines (splines)

$$\boxed{s_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3}$$

  together, in series, to create the appearance of a smooth function that touches all the identified points.

- Use the spline equation, 1st derivative, 2nd derivative and "initial" conditions to create a system of equations that solve for "c" (one of the spline parameters)
  - Use "c" to find the rest of the parameters

Cubic Splines Equations

$$\boxed{s_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3}$$

- **Interpolation** at **left end-point** (***n*** eq)

$$s_j(x_j) = y_j, j = 0, 1, \ldots, n - 1$$

$$\boxed{a_j = y_j}$$

$\{(x_j, y_j)\},$

Data set

- Intuitively, this constraint "anchors" the cubic splines to their left end-points, *e.g.*, $\mathbf{s_j}(x)$ is anchored to ($\mathbf{x_j}$, $\mathbf{y_j}$), etc.

Cubic Splines Equations

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

- **Interpolation** at **right end-point** (*1* eq)

$$y_n = s_{n-1}(x_n)$$

$$h_j = x_{j+1} - x_j$$

$$y_n = y_{n-1} + b_{n-1}h_{n-1} + c_{n-1}h_{n-1}^2 + d_{n-1}h_{n-1}^3$$

- The right end of the previous spline better line up with the left end of the next spline.
- h_j - a convenience function

Cubic Splines Equations

- **Given:**

$$\boxed{y_n = y_{n-1} + b_{n-1}h_{n-1} + c_{n-1}h_{n-1}^2 + d_{n-1}h_{n-1}^3}$$

- **Continuity** (***n-1*** eq) (@ *interior points, e.g. knots*)

$$s_{j+1}(x_{j+1}) = s_j(x_{j+1}), j = 0, \ldots, n-2$$

$$\boxed{y_{j+1} = y_j + b_j h_j + c_j h_j^2 + d_j h_j^3}$$

- Can be rewritten as:

$$b_j h_j + c_j h_j^2 + d_j h_j^3 = y_{j+1} - y_j, j = 0, \ldots, n-1$$

$$\boxed{h_j = x_{j+1} - x_j}$$

Cubic Splines Equations …

$$\boxed{s_j'(x) = b_j + 2\,c_j(x - x_j) + 3\,d_j(x - x_j)^2}$$

- Continuity of **1st derivative** (***n-1*** eq) (@ *interior pts)*

$$s_{j+1}'(x_{j+1}) = s_j'(x_{j+1}), j = 0, \ldots, n-2$$

$$\boxed{b_{j+1} = b_j + 2c_j h_j + 3d_j h_j^2}$$

- So:

$$b_j + 2c_j h_j + 3d_j h_j^2 - b_{j+1} = 0, \quad j = 0, \ldots, n-2$$

$$\boxed{h_j = x_{j+1} - x_j}$$

Cubic Splines Equations …

$$\boxed{s_j''(x) = 2\,c_j + 6\,d_j(x - x_j)}$$

- Continuity of **2nd derivative** (***n-1*** eq) (@ *interior pts)*

$$s_{j+1}''(x_{j+1}) = s_j''(x_{j+1}), j = 0, \ldots, n-2$$

$$\boxed{c_{j+1} = c_j + 3d_j h_j} \qquad \boxed{h_j = x_{j+1} - x_j}$$

- So:

$$c_j + 3d_j h_j - c_{j+1} = 0, \quad j = 0, \ldots, n-2$$

Cubic Splines Equations

- Trivial Equations (nothing to be solved for)

$$\boxed{a_j = y_j, j = 0, \ldots, n-1}$$

- ***System of linear equations*** *(3n-2* eq*)*

$$\boxed{\begin{array}{r} b_j h_j + c_j h_j^2 + d_j h_j^3 = y_{j+1} - y_j, j = 0, \ldots, n-1 \\ b_j + 2c_j h_j + 3d_j h_j^2 - b_{j+1} = 0, \quad j = 0, \ldots, n-2 \\ c_j + 3d_j h_j - c_{j+1} = 0, \quad j = 0, \ldots, n-2 \end{array}}$$

  – We need to solve a system of linear equations to ***find 3n variables*** b_j, c_j and d_j (*j=0,…n-1*)

- ***Need two more equations!***

Cubic Splines Equations

Given:

$$b_j h_j + c_j h_j^2 + d_j h_j^3 = y_{j+1} - y_j, j = 0, \ldots, n-1$$

$$b_j + 2c_j h_j + 3d_j h_j^2 - b_{j+1} = 0, \quad j = 0, \ldots, n-2$$

$$c_j + 3d_j h_j - c_{j+1} = 0, \quad j = 0, \ldots, n-2$$

In matrix form:

$j = 0, \ldots, n-2$

$$H_j = \begin{bmatrix} h_j & h_j^2 & h_j^3 \\ 1 & 2h_j & 3h_j^2 \\ 0 & 1 & 3h_j \end{bmatrix}, \boldsymbol{v}_j = \begin{bmatrix} b_j \\ c_j \\ d_j \end{bmatrix}, \boldsymbol{w}_j = \begin{bmatrix} y_{j+1} - y_j \\ 0 \\ 0 \end{bmatrix}$$

Cubic Splines Equations

- The first *3n-2* eqs are (in matrix form)

$$\begin{bmatrix} H_o & -S & 0 & 0 & 0 \\ 0 & H_1 & -S & \ddots & \\ \vdots & \ddots & & \ddots & 0 \\ 0 & 0 & 0 & H_{n-2} & -S \\ 0 & 0 & 0 & 0 & \widetilde{H}_{n-1} \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_o \\ \boldsymbol{v}_1 \\ \vdots \\ \boldsymbol{v}_{n-2} \\ \tilde{v}_{n-1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{w}_o \\ \boldsymbol{w}_1 \\ \vdots \\ \boldsymbol{w}_{n-2} \\ \tilde{w}_{n-1} \end{bmatrix}$$

Note the last row

Where:

$$\widetilde{H}_{n-1} = \begin{bmatrix} h_{n-1} & h_{n-1}^2 & h_{n-1}^3 \end{bmatrix}, v_{n-1} = b_{n-1}, w_{n-1} = y_n - y_{n-1}$$

and $S = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$ is a shift forward matrix

Warning: The equations are not used in practice, they are here just for reference

Additional Equations

We still need **2 more equations** to solve the system of equations

- But we have exhausted all the physical constraints

Make stuff up!

- Assume something about the constraints

Additional Equations

The **2 equations** are obtained by ***imposing boundary conditions***.

- ***Natural*** *(Zero 2nd derivatives at end-points)*
- ***Clamped*** *(Prescribed 1st derivatives at end-points)*
- ***Not-a-Knot*** *(Continuous 3rd derivatives at x_1 and x_{n-1})*
- ***Periodic*** *("Joined" end-points: $x_o=x_n$)*

Additional Equations

- ***Natural*** *(Zero 2nd derivatives at end-points)*

  $s_o''(x_o) = 0$

  $s_{n-1}''(x_n) = 0$

  set to zero curvature at endpoints

- ***Clamped*** *(Prescribed 1st derivatives at end-points)*

  $s_o'(x_o) = f'(x_0)$

  $s_n'(x_n) = f'(x_n)$

  clamp end-points at prescribed angles

Additional Equations

- ***Not-a-Knot*** *(Continuous 3rd derivatives at* x_1 *and* x_{n-1}*), e.g. the splines at the ends*

  $$s_o'''(x_1) = s_1'''(x_1)$$

  $$s_{n-1}'''(x_1) = s_{n-2}'''(x_{n-1})$$

- ***Periodic*** *("Joined" end-points:* $x_o = x_n$*)*

  $$s_o'(x_o) = s_{n-1}'(x_n)$$

  $$s''(x_o) = s''(x_n)$$

Solving the Spline Equations

- First the variables $\boldsymbol{b_j}$ and $\boldsymbol{d_j}$ are eliminated from the equations using the following:

$$c_j + 3d_j h_j - c_{j+1} = 0, \quad j = 0, \ldots, n-2$$

$$d_j = \frac{c_{j+1} - c_j}{3h_j}, \quad j = 0, \ldots, n-1$$

$$b_j h_j + c_j h_j^2 + d_j h_j^3 = y_{j+1} - y_j, j = 0, \ldots, n-1$$

$$b_j = \frac{y_{j+1} - y_j}{h_j} - \frac{c_{j+1} + 2c_j}{3} h_j, \quad j = 0, \ldots, n-1$$

$$\boxed{h_j = x_{j+1} - x_j}$$

Solve for “c” - work

- From before: $b_{j+1} = b_j + 2c_j h_j + 3d_j h_j^2$

$$b_j = \frac{y_{j+1} - y_j}{h_j} - \frac{c_{j+1} + 2c_j}{3} h_j, \qquad d_j = \frac{c_{j+1} - c_j}{3h_j}, \quad j = 0, \ldots, n-1$$

- Set: $b_j = \frac{y_{j+1} - y_j}{h_j} - \frac{c_{j+1} + 2c_j}{3} h_j, \ = b_{j-1} + 2c_{j-1}h_{j-1} + 3d_{j-1}h_{j-1}^2$
- Substituting d_j and d_{j-1}
- Simplifying and rearranging with h&c on one side and y on the other gives:

$$h_{j-1}c_{j-1} + 2(h_{j-1} + h_j)c_j + h_j c_{j+1}$$

$$= 3\left(\frac{y_{j+1} - y_j}{h_j} - \frac{y_j - y_{j-1}}{h_{j-1}}\right) \quad j=1,\ldots,n-1$$

Matrix view

From the previous slide:

$$h_{j-1}c_{j-1} + 2(h_{j-1} + h_j)c_j + h_j c_{j+1} = 3\left(\frac{y_{j+1} - yj}{h_j} - \frac{y_j - yj_{-1}}{h_{j\_1}}\right)$$

Let: $\alpha_j = 3\left(\frac{y_{j+1} - yj}{h_j} - \frac{y_j - yj_{-1}}{h_{j\_1}}\right)$ $j=1,\ldots,n-1$

So: $h_{j-1}c_{j-1} + 2(h_{j-1} + h_j)c_j + h_j c_{j+1} = \alpha_j$

This results in a matrix form of:

$$[\mathbf{H}]\ [\mathbf{c}] = [\boldsymbol{\alpha}]$$

Solving the Spline Equations

- Matrix left side

$h_j = x_{j+1} - x_j$

$$h_{j-1}c_{j-1} + 2(h_{j-1} + h_j)c_j + h_jc_{j+1}$$

- A column vector right side (called alpha)

$$\alpha_j = 3\left(\frac{y_{j+1} - y_j}{h_j} - \frac{y_j - y_{j-1}}{h_{j-1}}\right) \quad j=1,\dots,n-1$$

- The ***2 additional equations*** (for $j=0$ and $j=n$) will be provided by the ***boundary conditions.***

Natural Splines

- The endpoints of a natural spline ***do not have any curvature***: *(Zero 2nd derivatives at end-points)*

$$s_o''(x_o) = 0 \text{ and } s_{n-1}''(x_n) = 0$$

At the first end-point

$$s_o''(x_o) = 0 \Rightarrow c_0 = 0$$

At the second end-point

$$\boxed{s_j''(x) = 2\,c_j + 6\,d_j(x - x_j)}$$

$$\boxed{h_j = x_{j+1} - x_j}$$

$$s_{n-1}''(x_n) = 0 \Rightarrow c_{n-1} + 3d_{n-1}h_{n-1} = 0$$

Important: This equation has the ***same form*** as the equations for ***continuity of the 2nd derivative*** at interior points and is usually appended to those equations

Evaluate at: j = 1

$$h_{j-1}c_{j-1} + 2(h_{j-1} + h_j)c_j + h_j c_{j+1} = \alpha_j$$

Given: $c_0 = 0$

$$h_{1-1}c_{1-1} + 2(h_{1-1} + h_1)c_1 + h_1 c_{1+1} = \alpha_1$$

$$h_0 c_0 + 2(h_0 + h_1)c_1 + h_1 c_2 = \alpha_1$$

$$2(h_0 + h_1)c_1 + h_1 c_2 = \alpha_1$$

Evaluate at: j = n-1

$$h_{j-1}c_{j-1} + 2(h_{j-1} + h_j)c_j + h_j c_{j+1} = \alpha_j$$

Given: $c_{n-1} + 3d_{n-1}h_{n-1} = 0$ & $c_j + 3d_j h_j = c_{j+1}$

$$c_{n-1} + 3d_{n-1}h_{n-1} = c_{n-1+1}$$

$$c_{n-1} + 3d_{n-1}h_{n-1} = c_n \qquad so\ c_n = 0$$

$$h_{n-1-1}c_{n-1-1} + 2(h_{n-1-1} + h_{n-1})c_{n-1} + h_{n-1}c_{n-1+1} = \alpha_{n-1}$$

$$h_{n-2}c_{n-2} + 2(h_{n-2} + h_{n-1})c_{n-1} + h_{n-1}c_n = \alpha_{n-1}$$

$$h_{n-2}c_{n-2} + 2(h_{n-2} + h_{n-1})c_{n-1} = \alpha_{n-1}$$

Natural Splines

- The two additional equations given by the boundary conditions are

$$2(h_0 + h_1)c_1 + h_1 c_2 = \alpha_1 \qquad \text{(we were given } c_0 = 0\text{)}$$

$$h_{n-2}c_{n-2} + 2(h_{n-2} + h_{n-1})c_{n-1} = \alpha_{n-1}$$

- When added to the remaining *n-2* equations

$$h_{j-1}c_{j-1} + 2(h_{j-1} + h_j)c_j + h_j c_{j+1} = \alpha_j \qquad j=2,\ldots,n-2$$

- All use the same right hand side

$$\alpha_j = 3\left(\frac{y_{j+1} - y_j}{h_j} - \frac{y_j - y_{j-1}}{h_{j-1}}\right) \qquad j = 1, \ldots, n-1$$

they form a tridiagonal system (see next slide)

Natural Splines: Finding *c*

- The ***(n-1)(n-1) coefficient matrix*** of the system we need to solve to find the coefficients $c_1,\dots,c_{n-1}$ is a sparse ***symmetric, tridiagonal*** (***"strictly diagonal dominant"***)

$$\begin{bmatrix} 2(h_0+h_1) & h_1 & & & \\ h_1 & 2(h_1+h_2) & h_2 & & \\ & h_2 & 2(h_2+h_3) & h_3 & \\ & & \cdots & \cdots & \cdots \\ & & & & h_{n-2} \\ & & & h_{n-2} & 2(h_{n-2}+h_{n-1}) \end{bmatrix} c=\alpha$$

We do not need to find c_o because it is 0 $s_o''(x_o)=0 \Rightarrow c_0=0$

Strict diagonal matrices **do not require pivoting** in Gaussian Elimination

Natural Splines: Finding α

- From before (a system of n-1 linear equations)

$$\alpha_j = 3\left(\frac{y_{j+1} - y_j}{h_j} - \frac{y_j - y_{j-1}}{h_{j-1}}\right) \qquad j=1,\ldots,n-1$$

- The initial conditions force:
  - c[0] = 0;
  - c[N] = 0;

Note: In practical implementations the "c" spline matrix H is allocated with one extra entry to hold c[N].

Clamped Splines

- *Prescribed 1st derivatives at end-points*

Recall: $s_j'(x) = b_j + 2\,c_j(x - x_j) + 3\,d_j(x - x_j)^2$

- If the end points are "clamped" (given) then

$$s_0'(x_0) = f'(x_0) = y_0' = b_o$$
$$s_{n-1}'(x_n) = f'(x_n) = y_n' = b_{n-1} + 2c_{n-1}h_{n-1} + 3d_{n-1}h_{n-1}$$

- These constraints lead to the following equations (***the first and last***):

$$2h_0c_0 + h_0c_1 = 3\left(\frac{y_1 - y_0}{h_0} - y'_0\right)$$

$$h_{n-1}c_{n-1} + 2h_{n-1}c_n = 3\left(y'_n - \frac{y_n - yn_{-1}}{h_{n\_1}}\right)$$

Clamped Spline Equations c & α

- Summary: System of ***n+1*** linear equations is

$$2h_0c_0 + h_0c_1 = 3\left(\frac{y_1 - y_0}{h_0} - y'_0\right) \qquad j=0$$

$$h_{j-1}c_{j-1} + 2(h_{j-1} + h_j)c_j + h_jc_{j+1} = \alpha_j \qquad j = 1; \ldots n-1$$

$$\alpha_j = 3\left(\frac{y_{j+1} - y_j}{h_j} - \frac{y_j - y_{j-1}}{h_{j-1}}\right) \qquad j=1,\ldots,n-1$$

$$h_{n-1}c_{n-1} + 2h_{n-1}c_n = 3\left(y'_n - \frac{y_n - y_{n-1}}{h_{n-1}}\right) \qquad j=n$$

where c_j , $j=0,\ldots,n$ are the (n+1) unknowns

Clamped Splines: Finding *c*

- The ***(n+1)(n+1) coefficient matrix*** of the system we need to solve ***to find*** the coefficients $c_0, c_1, \ldots, c_n$ is ***symmetric, tridiagonal*** (***strict diagonal dominant***)

$$\begin{bmatrix} 2h_0 & h_0 & & & & \\ h_0 & 2(h_0+h_1) & h_1 & & & \\ & h_1 & 2(h_1+h_2) & h_2 & & \\ & & h_2 & \ldots & & \ldots \\ & & & \ldots & \ldots & \\ & & & & h_{n-2} & \\ & & & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\ & & & & h_{n-1} & 2h_{n-1} \end{bmatrix} c = \alpha$$

- Since the matrix is sparse we can *exploit the (**tridiagonal and symmetric**) structure* of this matrix to solve the system.
- Using LU solvers specialized to tridiagonal symmetric matrices allows us to ***solve the system in O(n) (instead of O(n^3))***

Not-a-Knot Cubic Spline

- *Continuous 3rd derivatives at x_1 and x_{n-1}*
- Use the following ***n-3 linear equations*** for $j=2,\ldots,n-2$

$$a_j = y_j$$
$$h_j = x_{j+1} - x_j$$

$$h_{j-1}c_{j-1} + 2(h_{j-1} + h_j)c_j + h_j c_{j+1} = \alpha_j$$

$$\alpha_j = 3\left(\frac{y_{j+1} - y_j}{h_j} - \frac{y_j - y_{j-1}}{h_{j-1}}\right) \qquad j=2,\ldots,n-2$$

- Add two equations (***the first and the last***)

$$\left(3h_0 + 2h_1 + \frac{h_0^2}{h_1}\right)c_1 + \left(h_1 - \frac{h_0^2}{h_1}\right)c_2 = \alpha_1$$

$$\left(h_{n-2} - \frac{h_{n-1}^2}{h_{n-2}}\right)c_{n-2} + \left(3h_{n-1} + 2h_{n-2} + \frac{h_{n-1}^2}{h_{n-2}}\right)c_{n-1} = \alpha_{n-1}$$

Not-a-Knot Cubic Spline c & α

- Summary: (System of ***n-1*** linear equations)

$$a_j = y_j$$
$$h_j = x_{j+1} - x_j$$

$$(3h_0 + 2h_1 + \frac{h^2_0}{h_1})c_1 + (h_1 - \frac{h^2_0}{h_1})c_2 = \alpha_1$$ j=1

$$h_{j-1}c_{j-1} + 2(h_{j-1} + h_j)c_j + h_j c_{j+1} = \alpha_j$$ j=2,..n-2

$$(h_{n-2} - \frac{h^2_{n-1}}{h_{n-2}})c_{n-2} + (3h_{n-1} + 2h_{n-2} + \frac{h^2_{n-1}}{h_{n-2}})c_{n-1} = \alpha_{n-1}$$ j=n-1

- Right hand side

$$\alpha_j = 3(\frac{y_{j+1} - y_j}{h_j} - \frac{y_j - y_{j-1}}{h_{j-1}})$$ j=1,…,n-1

Not-a-Knot: Finding *c*

- The ***(n-1)(n-1) coefficient matrix*** that needs to be solved to find the coefficients $c_1, \ldots, c_{n-1}$ is

$$\begin{bmatrix} 3h_0+2h_1+\frac{h^2_0}{h_1} & h_1-\frac{h^2_0}{h_1} & & & \\ h_1 & 2(h_1+h_2) & h_2 & & \\ & h_2 & 2(h_2+h_3) & h_3 & \\ & & \cdots & \cdots & \cdots \\ & & h_{n-3} & 2(h_{n-2}+h_{n-1}) & h_{n-2} \\ & & & h_{n-2}-\frac{h^2_{n-1}}{h_{n-2}} & 3h_{n-1}+2h_{n-2}+\frac{h^2_{n-1}}{h_{n-2}} \end{bmatrix} c=\alpha$$

- Note: This does not include the starting and ending spline

- This system is ***tridiagonal but not symmetric***

Additional Equations

- ***Not-a-Knot:*** *- (Continuous 3rd derivatives at x_1 and x_{n-1}):*

  $s_o'''(x_1) = s_1'''(x_1)$ & $s_{n-1}'''(x_{n-1}) = s_{n-2}'''(x_{n-1})$

  – ***3rd derivative gives: $d_0 = d_1$ and $d_{n-1} = d_{n-2}$***

The previous matrix can be though of as giving:

$c_1 \ldots c_{n-2}$

$$d_j = \frac{c_{j+1} - c_j}{3h_j}, \quad j = 0, \ldots, n-1$$

$$d_0 = d_1 \quad \Rightarrow \quad \frac{c_1 - c_0}{3h_0} = \frac{c_2 - c_1}{3h_1}$$

$$h_1c_1 - h_1c_0 = c_2h_0 - c_1h_0$$

$$h_1c_0 = h_1c_1 - c_2h_0 + c_1h_0$$

$$c_0 = c_1 + (c_1 - c_2)h_0/h_1$$

Additional Equations

- ***$d_{n-1} = d_{n-2}$***

$$d_{n-1} = d_{n-2} \quad \Rightarrow \quad \frac{c_n - c_{n-1}}{3h_{n-1}} = \frac{c_{n-1} - c_{n-2}}{3h_{n-2}} \qquad d_j = \frac{c_{j+1} - c_j}{3h_j},$$

$$h_{n-2}c_n - h_{n-2}c_{n-1} = c_{n-1}h_{n-1} - c_{n-2}h_{n-1}$$

$$h_{n-2}c_n = c_{n-1}h_{n-1} - c_{n-2}h_{n-1} - h_{n-2}c_{n-1}$$

$$c_n = (c_{n-1} - c_{n-2})h_{n-1}/h_{n-2} + c_{n-1}$$

Note: In practical applications c_0 and c_n are calculated after the H matrix and new entries are “added” before the start and after the end of the c vector

Periodic Boundary Conditions

- This splines describe closed curves: the first point and the last point are the same
- The ***(n+1) (n+1) coefficient matrix*** of the system we need to solve to find the ***c*** coefficients is

$$
\begin{bmatrix}
2(h_{n-1}+h_0) & h_0 & & & & h_{n-1} \\
h_0 & 2(h_0+h_1) & h_1 & & & \\
 & h_1 & 2(h_1+h_2) & h_2 & & \\
 & & h_2 & \cdots & & \cdots \\
 & & & \cdots & \cdots \; h_{n-2} & \\
 & & & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\
h_{n-1} & & & & h_{n-1} & 2(h_{n-2}+h_{n-1})
\end{bmatrix}
$$

Not required in this class

Note: The above symmetric matrix is called a **circulant matrix** (often encountered in FFT computations)

Spline Summary

- Natural:

$$\begin{bmatrix} 2(h_0+h_1) & h_1 & & & \\ h_1 & 2(h_1+h_2) & h_2 & & \\ & h_2 & 2(h_2+h_3) & h_3 & \\ & & \cdots & \cdots & \cdots \\ & & & & h_{n-2} \\ & & & h_{n-2} & 2(hn_{-2}+h_{n-1}) \end{bmatrix}$$

- Clamped

$$\begin{bmatrix} 2h_0 & h_0 & & & & \\ h_0 & 2(h_0+h_1) & h_1 & & & \\ & h_1 & 2(h_1+h_2) & h_2 & & \\ & & h_2 & \cdots & & \cdots \\ & & & \cdots & h_{n-2} & \\ & & & h_{n-2} & 2(hn_{-2}+h_{n-1}) & h_{n-1} \\ & & & & h_{n-1} & 2hn_{-1}) \end{bmatrix}$$

- Not-a-knot

$$\begin{bmatrix} 3h_0+2h_1+\frac{h^2_0}{h_1} & h_1-\frac{h^2_0}{h_1} & & & \\ h_1 & 2(h_1+h_2) & h_2 & & \\ & h_2 & 2(h_2+h_3) & h_3 & \\ & & \cdots & \cdots & \cdots \\ & & h_{n-3} & 2(hn_{-2}+h_{n-1}) & h_{n-2} \\ & & & h_{n-2}-\frac{h^2_{n-1}}{h_{n-2}} & 3hn_{-1}+2h_{n-2}+\frac{h^2_{n-1}}{h_{n-2}} \end{bmatrix}$$

The core code is identical, only the initial conditions change

Reminder: a,b,d Spline Equations

- The previous simplifications results in equations for the spline parameters a, b and d, only in terms of c, y and x (or h), all of which are now known.

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

$$\boxed{a_j = y_j, j = 0, \ldots, n-1}$$

$$b_j = \frac{y_{j+1} - y_j}{h_j} - \frac{c_{j+1} + 2c_j}{3} h_j, \quad j = 0, \ldots, n-1$$

c_j = from matrix solution

$$d_j = \frac{c_{j+1} - c_j}{3h_j}, \quad j = 0, \ldots, n-1$$

$$\boxed{h_j = x_{j+1} - x_j}$$

LU Factorization of Tridiagonals

- A tridiagonal system can be written as:

$$\begin{bmatrix} q_1 & r_1 & & & & \\ p_1 & q_2 & r_2 & & & \\ & p_2 & q_3 & r_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & p_{n-2} & q_{n-1} & r_{n-1} \\ & & & & p_{n-1} & q_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \vdots \\ \beta_{n-1} \\ \beta_n \end{bmatrix}$$

- Note: A clever programmer will represent the generic matrix A as three vectors (to save memory):
  - sub-diagonal $(n-1)$-vector $\boldsymbol{p}$
  - diagonal n-vector $\boldsymbol{q}$
  - super-diagonal $(n-1)$-vector r

Where: $\boldsymbol{n}$ is the number of columns/rows in the matrix, (not the number of spline points)

LU Factorization of Tridiagonals

- *LU factorization* does not require partial pivoting (because the matrix is diagonal dominant).
  - The number of FLOPS required drops from $O(n^3)$ to $O(n)$.

$$L = \begin{bmatrix} 1 & & & & & \\ \ell_1 & 1 & & & & \\ & \ell_2 & 1 & & & \\ & & \ddots & \ddots & & \\ & & & \ell_{n-2} & 1 & \\ & & & & \ell_{n-1} & 1 \end{bmatrix}, \quad U = \begin{bmatrix} d_1 & u_1 & & & & \\ & d_2 & u_2 & & & \\ & & d_3 & u_3 & & \\ & & & \ddots & \ddots & \\ & & & & d_{n-1} & u_{n-1} \\ & & & & & d_n \end{bmatrix}$$

- $Ax = \beta \Rightarrow LUx = \beta$ is solved as usual
  - Solve $Lz = \beta$ for z by forward-substitution
  - Solve $Ux = z$ by back-substitution

Tridiagonal pseudo-code

/* LU Factorization or Elimination */
$d_0 = q_0$; $u_0 = r_0$; $l_0 = p_0/d_0$
for i = 1,2,…,n−2
$d_i = q_i - l_{i-1} u_{i-1}$
$u_i = r_i$
$l_i = p_i/d_i$
$d_{n-1} = q_{n-1} - l_{n-2} * u_{n-2}$

Note: that indexing starts at 0
p,q,r are the diagonal data from a tridiagonal matrix

d,l,u are the standard terms from an LU matrix

/* Forward Substitution: Solving for z */
$z_0 = \beta_0$
for i = 1,2,,...,n-1 { $z_i = \beta_i - l_{i-1} * z_{i-1}$ }

/* Back Substitution Solving for x */
$X_{n-1} = z_{n-1}/d_{n-1}$
for i = n −2,n −3,...,0 { $x_i = (z_i - u_i x_{i+1})/d_i$}

Symmetric Tridiagonal Solver

This algorithm **overwrites** ***b*** with the solution to $T\,\boldsymbol{x} = \boldsymbol{b}$. (**d** stores the ***diagonal*** and **e** the **super diagonal** of T)

```
/* Matrix indexing starts at 1 */
      for k = 2:n
        t = e(k-1);
        e(k-1) = t/d(k-1);
        d(k) = d(k) - t*e(k-1)
      end
      for k = 2:n
        b(k) = b(k) - e(k-1)*b(k-1)
      end
      b(n-1)=b(n-1)/d(n-1)
      for k = n-1:-1:1
        b(k) = b(k)/d(k) - e(k)*b(k+1)
      end
```

Precondition:

T is tridiagonal, symmetric

This algorithm requires ***8n* FLOP**

(Reference: Algorithm 4.3.6; G. Golub and C. Van Loan, Matrix Computations)

Tridiagonal Improvements

- The algorithm given can be improved:
  - It is not necessary to calculate any u_i because $u_i = r_i$.
  - Work "in place"
    - overwriting p with l, q with d, r with u and β with the solution x.
- **References**

  - B. Bradie (2006), *A Friendly Introduction to Numerical Analysis*, Prentice Hall, Upper Saddle River, NJ

Summary: Cubic Spline Interpolation

Construction:

Finding the coefficients of a cubic spline that interpolates n+1 points requires *two steps*:

1) Compute the (***n+1***) coefficients **c** solving the corresponding tridiagonal system of equations
2) Use the spline formulas to obtain the remaining coefficients **b** and **d**

Notes:

- The a coefficients are determined directly from the data: $a_i=y_i$ (no need to "compute" them)
- For **clamped splines**, the **derivative** of the function **at the end-points**, y'_0, y'_n must be provided (as additional data)

Summary: Cubic Spline Interpolation

Evaluation:

To evaluate a cubic spline *s(x)* at a point *x* also requires *two steps:* $x_j(.)$ $x < x_{j+1}$

1) Given *x*, *find the interval* where it belongs. This will tell us *which cubic polynomial* $s_j(.)$ should be evaluated.

2) Evaluate the corresponding cubic polynomial, *e.g.*, $s_j(.)$ at the give point *x* to obtain the desired value (using *nested evaluation*)

Error Estimate (Theoretical Result)

- An estimate of the maximum error in the approximation of a function *f* (four times differentiable) with a clamped cubic spline is

$$e_{max} \leq \frac{5}{384} h^4 \max_{x \in | a,b |} | f^{(4)}(x)|$$

$$h = \max_{0 \leq i \leq n-1} (x_{i+1} - x_i)$$

Evaluating Splines

- The result of calculating spline coefficients is table with "x" point ranges and spline coefficients in: d, c, b, a

- Sample spline table:

X0, X1, d, c, b, a   N= 3

0.0000000	0.3141593	14.9167590	-11.0524281	2.0000000	0.0000000
0.3141593	0.6283185	-4.1670526	3.0062863	-0.5277700	0.0000008
0.6283185	0.9424778	4.1223494	-0.9210683	0.1273205	0.0017007

Evaluating Splines

- To evaluate a spline at a point
  - Find the corresponding row in the table
  - Subtract the table start value from the point
  - Calculate the spline

x0,	x1,	d,	c,	b,	a
0.0000000	0.3141593	14.9167590	-11.0524281	2.0000000	0.0000000
0.3141593	0.6283185	-4.1670526	3.0062863	-0.5277700	0.0000008
0.6283185	0.9424778	4.1223494	-0.9210683	0.1273205	0.0017007

E.g: The point x= 0.35 is calculated using the spline defined in row 2

Evaluating Splines

- Find s(0.35) given the following spline :

x0	x1	d	c	b	a
0.3141593	0.6283185	-4.1670526	3.0062863	-0.5277700	0.0000008

- Reminder: $s(z) = a + b\,z + c\,z^2 + d\,z^3$

s(0.35) = 0.0000008
-0.5277700 * (0.35- 0.3141593)
+3.0062863*(0.35- 0.3141593)**2
-4.1670526 *(0.35- 0.3141593)**3

s(0.35) = -0.01524

HW Hints

- Logically need to solve a matrix for “c”
  - Actually implemented with 3 vectors: p, q, r

- Requires:
  - “h” vector (based on x values we know)
  - “α” vector (based on y values we know)

HW Hints

- Clamped "α"
- Has 2 special cases j=0 & j = N
  - $\alpha_0 = 3(\frac{y_1 - y_0}{h_0} - y'_0)$
  - $\alpha_n = 3\,(y'_n - \frac{y_n - yn_{-1}}{h_{n\_1}})$
- General case (j=1,…,n-1):
  - $\alpha_j = 3(\frac{y_{j+1} - yj}{h_j} - \frac{y_j - yj_{-1}}{h_{j\_1}})$

So the α vector must be size n+1

HW Hints

- "H" matrix
  - Not a matrix, implemented as 3 vectors
  - p – outside bottom
  - q – middle
  - r – outside top
  - p & r are "shorter" than q
  - How long is "q"?
    - Same length as α

$$\begin{bmatrix} q_1 & r_1 & & & & \\ p_1 & q_2 & r_2 & & & \\ & p_2 & q_3 & r_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & p_{n-2} & q_{n-1} & r_{n-1} \\ & & & & p_{n-1} & q_n \end{bmatrix}$$

Problem 1

- A cubic spline was constructed to interpolate the data points

x	5	6	7	8	9
y	3	2	-1	-2	-3

- The following table of coefficients was reported

i	D_i	C_i	B_i	a_i
0	-0.6786	0.0000	-0.3214	3.0000
1	1.3929	-2.0357	-2.3571	2.0000
2	-0.8929	2.1429	-2.2500	-1.0000
3	0.1786	-0.5357	-0.6429	-2.0000

- In what interval is the spline piece $s_1(x)$ defined ?
- $s1(x)$ interpolated in [6,7]

Problem 2

- Write the spline equation for: $s_2(x)$

i	D_i	C_i	B_i	a_i
0	-0.6786	0.0000	-0.3214	3.0000
1	1.3929	-2.0357	-2.3571	2.0000
2	-0.8929	2.1429	-2.2500	-1.0000
3	0.1786	-0.5357	-0.6429	-2.0000

$$s_2(x) = a_2 + b_2(x-x_2) + c_2(x-x_2)^2 + d_2(x-x_2)^3$$

$$s_2(x) = -1-2.25(x-2)+2.1429(x-2)^2-.8929(x-2)^3$$

Problem 3

- Write the spline equation and evaluate for $x = 2.3$

x	D_i	C_i	B_i	a_i
0	-0.6786	0.0000	-0.3214	3.0000
1	1.3929	-2.0357	-2.3571	2.0000
2	-0.8929	2.1429	-2.2500	-1.0000
3	0.1786	-0.5357	-0.6429	-2.0000

$x = 2.3$ is between 2 & 3, so use $s_2(x)$

$$s_2(x) = -1-2.25(x-x_2)+2.1429(x-x_2)^2-.8929(x-x_2)^3$$

$$s_2(2.3) = -1-2.25(2.3-2)+2.1429(2.3-2)^2-.8929(2.3-2)^3$$

$$s_2(2.3) = -1.5062473$$

Problem 4

- Give the following spline table:

x	D_i	C_i	B_i	a_i
0	-0.6786	0.0000	-0.3214	3.0000
1	1.3929	-2.0357	-2.3571	2.0000
2	-0.8929	2.1429	-2.2500	-1.0000
3	0.1786	-0.5357	-0.6429	-2.0000

- What is the "y" value for "x=1.00001"?

1.00001 is VERY close to "x=1" so the "a" term will dominate. Remember the spline equations are defined such that "b, c & d" have little effect when x is near the spline orign.

This is a HANDY WAY to verify spline evaluation code!

Therefore y= "2.0000"

Problem 5

- How do you “visualize” splines?

- Simply evaluate the spline repetitively using small increments and then plot the resulting (x,y) pairs.

Problem 6

- What boundary condition must have been satisfied?

i	D_i	C_i	B_i	a_i
0	-0.6786	0.0000	-0.3214	3.0000
1	1.3929	-2.0357	-2.3571	2.0000
2	-0.8929	2.1429	-2.2500	-1.0000
3	0.1786	-0.5357	-0.6429	-2.0000

$c_0 = 0$

This is the ***natural boundary condition***

Tridiagonal pseudo-code (1)

/* LU Factorization or Elimination */
$d_1 = q_1$; $u_1 = r_1$; $l_1 = p_1/d_1$
for i = 2,3,…,n −1
$d_i = q_i - l_{i-1} u_{i-1}$
$u_i = r_i$
$l_i = p_i/d_i$
$d_n = q_n - l_{n-1}u_{n-1}$

/* Forward Substitution: Solving for z */
$z_1 = \beta_1$
for i = 2,3,...,n { $z_i = \beta_i - l_{i-1} z_{i-1}$ }

/* Back Substitution Solving for x */
$x_n = z_n/d_n$
for i = n −1,n −2,...,1 { $x_i = (z_i - u_i x_{i+1})/d_i$}

Note: that indexing starts at 1
p,q,r are the diagonal data from a tridiagonal matrix

d,l,u are the standard terms from an LU matrix