Piecewise Cubic Approx.

We will consider three possibilities.

- Piecewise Cubic Interpolant: On each $[x_{i-1}, x_i]$, introduce $\xi_{i0} = x_{i-1}, \xi_{i3} = x_i$ and two additional points ξ_{i1}, ξ_{i2} to define $P_{i,3}(x)$. We then have,

$$|f(x) - S(x)| \leq \frac{L}{4!} h^4, \quad h = \max_{i=1}^{M} |x_i - x_{i-1}|.$$

Note that $S(x)$ is continuous but not 'smooth'.

Piecewise Cubic Hermite

Define $\bar{P}_{i,3}(x)$ to be the unique cubic polynomial associated with $[x_{i-1}, x_i]$ such that $\bar{P}_{i,3}(x_{i-1}) = f_{i-1},\ \bar{P}'_{i,3}(x_{i-1}) = f'_{i-1}$, $\bar{P}_{i,3}(x_i) = f_i$ and $\bar{P}'_{i,3}(x_i) = f'_i$.

- The resulting $\bar{S}(x)$ will be continuous and differentiable since $\bar{P}'_{i,3}(x_{i-1}) = f'_{i-1} = \bar{P}'_{i-1,3}(x_{i-1})$.
- The error in $\bar{S}(x)$ satisfies

$$|f(x) - \bar{S}(x)| \leq \frac{L}{4!}(x - x_{i-1})^2(x - x_i)^2, \leq \frac{L}{4!}(h/2)^4,$$

  where $L \geq \max_{a \leq x \leq b}\{|f^4(x)|\}$ and $h = \max_{i=1}^{M} |x_i - x_{i-1}|$.
- $\bar{S}(x)$ will be continuous and ‘smooth’ but $\bar{S}''(x)$ will usually be discontinuous.

Cubic Splines

The basic idea is to determine the polynomial, $\hat{P}_{i,3}(x)$, associated with $[x_{i-1}, x_i]$, by requiring $\hat{P}_{i,3}(x)$ to interpolate $f_{i-1},\ f_i$ and have continuous first and second derivatives at x_{i-1}.

- Consider the following representation of $\hat{P}_{i,3}(x)$,

$$\hat{P}_{i,3}(x) = c_{i0} + c_{i1}(x - x_{i-1}) + c_{i2}(x - x_{i-1})^2 + c_{i3}(x - x_{i-1})^3,$$

- The piecewise polynomial, $\hat{S}(x)$, is determined by specifying $4M$ linear equations which uniquely determine the c_{ij}'s. To do this we let $h_i = x_i - x_{i-1}$ and we associate 'interpolation' and 'smoothness' constraints with each subinterval.

Cubic Splines (cont.)

- On the first subinterval (3 interpolation constraints),

$$\begin{aligned}
\hat{P}'_{1,3}(x_0) = ? \quad &\Rightarrow \quad c_{11} = ?\ , \\
\hat{P}_{1,3}(x_0) = f_0 \quad &\Rightarrow \quad c_{10} = f_0, \\
\hat{P}_{1,3}(x_1) = f_1 \quad &\Rightarrow \quad c_{10} + c_{11}h_1 + c_{12}h_1^2 + c_{13}h_1^3 = f_1,
\end{aligned}$$

- On the second subinterval (2 interpolation and 2 smoothness constraints),

$$\begin{aligned}
\hat{P}_{2,3}(x_1) = f_1 \quad &\Rightarrow \quad c_{2,0} = f_1, \\
\hat{P}'_{2,3}(x_1) = \hat{P}'_{1,3}(x_1) \quad &\Rightarrow \quad c_{2,1} - c_{1,1} - 2c_{1,2}h_1 - 3c_{1,3}h_1^2 = 0, \\
\hat{P}''_{2,3}(x_1) = \hat{P}''_{1,3}(x_1) \quad &\Rightarrow \quad 2c_{2,2} - 2c_{1,2} - 6c_{1,3}h_1 = 0, \\
\hat{P}_{2,3}(x_2) = f_2 \quad &\Rightarrow \quad c_{2,0} + c_{2,1}h_2 + c_{2,2}h_2^2 + c_{2,3}h_2^3 = f_2,
\end{aligned}$$

Cubic Splines (cont.)

- In general on the i^{th} subinterval (2 interpolation and 2 smoothness constraints),

$$\begin{aligned}
\hat{P}_{i,3}(x_{i-1}) = f_{i-1} \quad &\Rightarrow \quad c_{i,0} = f_{i-1}, \\
\hat{P}'_{i,3}(x_{i-1}) = \hat{P}'_{i-1,3}(x_{i-1}) \quad &\Rightarrow \quad c_{i,1} - c_{i-1,1} - 2c_{i-1,2}h_{i-1} - 3c_{i-1,3}h_{i-1}^2 = 0, \\
\hat{P}''_{i,3}(x_{i-1}) = \hat{P}''_{i-1,3}(x_{i-1}) \quad &\Rightarrow \quad 2c_{i,2} - 2c_{i-1,2} - 6c_{i-1,3}h_{i-1} = 0, \\
\hat{P}_{i,3}(x_i) = f_i \quad &\Rightarrow \quad c_{i,0} + c_{i,1}h_i + c_{i,2}h_i^2 + c_{i,3}h_i^3 = f_i,
\end{aligned}$$

- And finally on the last subinterval we impose an additional interpolation constraint:

$$\hat{P}'_{M,3}(x_M) = c_{M,1} + 2c_{M,2}h_M + 3c_{M,3}h_M^2 = ?.$$

Observations

- The two extra interpolation constraints imposed at x_0 and x_M can be set by specifying $f_0^{'}$ and $f_M^{'}$ or by using the respective approximating values, $f[x_0 x_1]$ and $f[x_{M-1} x_M]$ or by replacing these constraints with $\hat{P}_{1,3}^{''}(x_0) = \hat{P}_{M,3}^{''}(x_M) = 0$. The latter choice leads to what are called 'natural splines'.
- The total number of linear equations is then $3 + 4(M-1) + 1 = 4M$. This system can be shown to be nonsingular as long as the x_i's are distinct.
- The error bound for splines is similar to that for cubic Hermite. That is, it can be shown that, in most cases

$$|f(x) - \hat{S}(x)| \leq \frac{5L}{4!}(h/2)^4$$

(although for natural splines we only have $O(h^2)$ accuracy).

Difficulties with Spline Approx.

- They do not preserve monotone data. That is if the data is monotone (increasing or decreasing) then, in some applications, so should the interpolant.
- They do not preserve discontinuous derivatives. For example in representing ‘corners’.
- They assume accurate data. In many applications (in particular in those arising in CAD and computer graphics) one often wants to represent the ‘general shape’ of the function or curve rather than insisting on strict interpolation. (Bezier curves and/or linear least squares can be used.)

Interpolation in 2D

Consider the problem of approximating $u(x, y)$ in a finite region of R^2 or approximating a surface in 3-dimensions. We can generalize the notion of piecewise polynomials to higher dimensions.

- The original region (or domain) is first decomposed (or partitioned) into a collection of regularly-shaped subregions. Usually rectangles or triangles are used.
- Over each subregion (the triangular or rectangular mesh element) one can define a bi-variate polynomial (in x and y) and use the total collection of such polynomials to define the bivariate piecewise polynomial, $S(x, y)$.
- One can define the coefficients of the polynomials associated with each element by imposing interpolation or continuity constraints at the boundaries.

Data Fitting

- Given data $\{x_i, f_i\}_{i=0}^m$ find the polynomial $p_n(x)$ of degree at most n, $\;p_n(x) = c_0 + c_1 x + \cdots c_n x^n$, such that $G(\underline{c})$ is minimized,

$$G(\underline{c}) = \sum_{i=0}^{m} (p_n(x_i) - f_i)^2.$$

- Other norms could be used–for example, $\hat{G}(\underline{c}) = \sum_{i=0}^m |p_n(x_i) - f_i|$ or $\tilde{G}(\underline{c}) = \max_{i=0}^m |p_n(x_i) - f_i|$, but these are not differentiable and the resulting algorithms are more complex.
- We will assume that the x_i's are distinct and $m > n$. From linear algebra we know that these assumptions will guarantee a full rank problem with a unique solution.

Data Fitting (cont)

For a given polynomial, $p_n(x)$, let $r_i(\underline{c}) = p_n(x_i) - f_i$ for $i = 0, 1, \cdots m$. We then have,

$$\begin{aligned} r_i(\underline{c}) &= c_0 + c_1 x_i + \cdots c_n x_i^n - f_i, \\ &= \left[A\underline{c} - \underline{f}\right]_i, \end{aligned}$$

where A is the $(m+1) \times (n+1)$ matrix,

$$A = \begin{bmatrix} 1 & x_0 & \cdots & x_0^n \\ 1 & x_1 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_m & \cdots & x_m^n \end{bmatrix},$$

$\underline{c} \in \Re^{n+1}$, $\underline{c} = (c_0, c_1, \cdots c_n)^T$ and $\underline{f} \in \Re^{m+1}$ is $\underline{f} = (f_0, f_1, \cdots f_m)^T$.

Computing the ‘Best Fit’

We will show that the polynomial that minimizes $G(\underline{c})$ is the solution of the linear system of equations,

$$A^T A\underline{c} = A^T \underline{f}.$$

These are called the Normal Equations. A solution always exists and can be computed in an efficient and stable way.

Computing the ‘Best Fit’ (cont)

From calculus we know that $G(\underline{c})$ is a minimum when,

$$\frac{\partial G}{\partial c_j} = 0 \text{ for } \quad j = 0, 1, \cdots n.$$

But since $G(\underline{c}) = \sum_{i=0}^{m} r_i^2(\underline{c})$ we have,

$$\begin{aligned}
\frac{\partial G}{\partial c_j} &= \frac{\partial}{\partial c_j}\left[\sum_{i=0}^{m} r_i^2(\underline{c})\right] \\
&= \sum_{i=0}^{m} \frac{\partial}{\partial c_j}(r_i^2(\underline{c})), \\
&= 2\sum_{i=0}^{m} r_i(\underline{c})\frac{\partial r_i(\underline{c})}{\partial c_j}.
\end{aligned}$$

Normal Equations (cont)

From the definition of $r_i(\underline{c})$ we have,

$$\frac{\partial r_i(\underline{c})}{\partial c_j} = \frac{\partial}{\partial c_j}\left[A\underline{c} - \underline{f}\right]_i \quad = (a_{i,j}) = x_i^j,$$

for $i = 0, 1, \cdots m; j = 0, 1, \cdots n$.
It then follows that

$$\frac{\partial G}{\partial c_j} = 2\sum_{i=0}^{m} r_i(\underline{c})a_{i,j} = 2\left(A^T\underline{r}\right)_j .$$

Therefore to achieve $\frac{\partial G}{\partial c_j} = 0$ for $j = 0, 1, \cdots n$ we must have,

$$\left(A^T\underline{r}\right)_j = 0, \text{ for } j = 0, 1, \cdots m.$$

Normal Equations (cont)

This is equivalent to asking that,

$$A^T \underline{r} = \underline{0} \text{ or } A^T \left(A\underline{c} - \underline{f} \right) = \underline{0}.$$

Note that the matrix $A^T A$ is a square $(n+1) \times (n+1)$ nonsingular matrix and we can therefore compute the best fit least squares polynomial, $p_n(x)$ by solving the linear system:

$$\boxed{\boldsymbol{A^T A \underline{c} = A^T \underline{f}}}$$

These linear equations are called the <u>Normal Equations</u>.

Efficient Solution of $A^TA\underline{c} = A^T\underline{f}$

To solve the Normal Equations efficiently we use a QR based algorithm that doesn't require the explicit computation of the (possibly ill-conditioned) matrix A^TA.
Consider forming the $\mathcal{QR}$ factorization (or Schur decomposition) of the $(m+1) \times (n+1)$ matrix A,

$$A = \mathcal{QR} = (Q_1 Q_2 \cdots Q_{n+1})\mathcal{R},$$

where $\mathcal{Q}$ is an orthogonal matrix and $\mathcal{R}$ is an upper triangular matrix. This is a standard factorization in numerical linear algebra and is usually accomplished using a modified Gram Schmidt algorithm or an algorithm based on the use of a sequence of 'Householder reflections'. We will consider the latter approach.

Efficient Algorithm (cont)

We determine $\mathcal{Q}$ as a product of $n+1$ Householder reflections:

$$\mathcal{Q}A = \mathcal{R} \Leftrightarrow Q_{n+1}^T(Q_n^T \cdots Q_1^T A)) \cdots) = \mathcal{R},$$

where each $Q_i = Q_i^T$ is an $(m+1) \times (m+1)$ Householder reflection, $\mathcal{R}$ is an $(m+1) \times (n+1)$ upper triangular matrix,

$$\mathcal{R} = \begin{bmatrix} x & x & \cdots & x \\ 0 & x & \cdots & x \\ 0 & 0 & \cdots & x \\ \vdots & \vdots & \vdots & x \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \equiv \begin{bmatrix} R \\ 0 \end{bmatrix},$$

and R is an $(n+1) \times (n+1)$ upper triangular matrix.

Efficient Algorithm (cont)

We then have,

$$A^TA = (\mathcal{QR})^T\mathcal{QR} = \mathcal{R}^T\mathcal{Q}^T\mathcal{QR} = \mathcal{R}^T\mathcal{R},$$

where

$$\mathcal{R}^T\mathcal{R} = \begin{bmatrix} x & 0 & \cdots & 0 \\ x & x & \cdots & 0 \\ \vdots & \vdots & \cdots & 0 \\ x & x & \cdots & 0 \end{bmatrix} \begin{bmatrix} x & x & \cdots & x \\ 0 & x & \cdots & x \\ 0 & 0 & \cdots & x \\ \vdots & \vdots & \cdots & x \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

.

$$= \begin{bmatrix} R^T & 0 \end{bmatrix} \begin{bmatrix} R \\ 0 \end{bmatrix} = R^TR.$$

Normal Equations - Summary

Therefore to solve the normal equations we need only solve the equivalent linear system,

$$R^T R\underline{c} = A^T \underline{f}.$$

This requires the computation of $A^T \underline{f}$ (at a cost of $(n+1)(m+1)$ flops) and two triangular linear systems (at a cost of n^2 flops). The cost of determining the $\mathcal{QR}$ factorization of A is $n^2 m + O(nm)$ and therefore the total cost of this algorithm to determine $\underline{c}$ is $n^2(m+1) + O(nm)$ flops.

Note that in most applications m is much larger than n and n is often less than 4.