Interpolation

IPCST
Seoul National University

Polynomial Interpolation

- Data points: (x_k, y_k), $k = 0,\ldots,n$ & $x_k < x_{k+1}$

- Find a polynomial of degree n

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

which satisfies $P_n(x_k) = y_k$ for $k = 0,\ldots,n$

→ Find the coefficients $a_n, a_{n-1}, \ldots, a_1, a_0$

Polynomial Interpolation

- Application or purpose
  - Function fitting to experimental data
    - A polynomial can be an approximation of some hidden function.
  - To find integrals or derivatives easily
    - Integration or differentiation of polynomials is easy.

Polynomial Interpolation

- Example 2.1 in Wen Shen

x_i	0	1	2/3
y_i	1	0	0.5

$$P_2(x) = a_2 x^2 + a_1 x + a_0.$$

$$\begin{aligned} x = 0,\ y = 1 &: \quad P_2(0) = a_0 = 1, \\ x = 1,\ y = 0 &: \quad P_2(1) = a_2 + a_1 + a_0 = 0, \\ x = 2/3,\ y = 0.5 &: \quad P_2(2/3) = (4/9)a_2 + (2/3)a_1 + a_0 = 0.5. \end{aligned}$$

$$\begin{pmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ \frac{4}{9} & \frac{2}{3} & 1 \end{pmatrix} \begin{pmatrix} a_2 \\ a_1 \\ a_0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0.5 \end{pmatrix} \Rightarrow \quad a_2 = -\frac{3}{4}, \quad a_1 = -\frac{1}{4}, \quad a_0 = 1.$$

Polynomial Interpolation

- Van der Monde method

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

$$\begin{array}{ll} P_n(x_0) = y_0 & a_n x_0^n + a_{n-1} x_0^{n-1} + \cdots + a_1 x_0 + a_0 = y_0 \\ \vdots & \vdots \\ P_n(x_n) = y_n & a_n x_n^n + a_{n-1} x_n^{n-1} + \cdots + a_1 x_n + a_0 = y_n \end{array}$$

$$\begin{pmatrix} x_0^n & x_0^{n-1} & \cdots & 1 \\ x_1^n & x_1^{n-1} & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ x_n^n & x_n^{n-1} & \cdots & 1 \end{pmatrix} \begin{pmatrix} a_n \\ a_{n-1} \\ \vdots \\ a_0 \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix} \Longrightarrow \mathbf{X}\mathbf{a} = \mathbf{y}$$

Van der Monde matrix

✓ Solving this by linear algebra

Polynomial Interpolation

- Van der Monde method
  - The van der Monde matrix $\mathbf{X}$ is invertible as long as x_i's are distinct.
  - But the matrix has usually a large condition number. → nearly singular → often huge numerical errors

Polynomial Interpolation

- Cardinal functions

$$l_i(x_j) = \delta_{ij} = \begin{cases} 1 & , \quad i = j \\ 0 & , \quad i \neq j \end{cases} \qquad i = 0, 1, \cdots, n$$

$$\begin{aligned} l_i(x) &= \prod_{j=0, j\neq i}^{n} \left(\frac{x - x_j}{x_i - x_j} \right) \\ &= \frac{x - x_0}{x_i - x_0} \cdot \frac{x - x_1}{x_i - x_1} \cdots \frac{x - x_{i-1}}{x_i - x_{i-1}} \cdot \frac{x - x_{i+1}}{x_i - x_{i+1}} \cdots \frac{x - x_n}{x_i - x_n} \end{aligned}$$

Polynomial Interpolation

- Lagrange interpolation polynomials

$$P_n(x) = \sum_{i=0}^{n} l_i(x) \cdot y_i$$

$$= \sum_{i=0}^{n} \left[\prod_{j=0, j\neq i}^{n} \left(\frac{x - x_j}{x_i - x_j} \right) \right] \cdot y_i$$

  – Check $P_n(x_j) = \sum_{i=0}^{n} l_i(x_j) \cdot y_i = y_j, \qquad \text{for every } \; j.$

Polynomial Interpolation

- Example 2.2 in Wen Shen

x_i	0	1	2/3
y_i	1	0	0.5

$$l_0(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} = \frac{(x-2/3)}{(0-2/3)}\frac{(x-1)}{(0-1)} = \frac{3}{2}\left(x-\frac{2}{3}\right)(x-1)$$

$$l_1(x) = \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} = \frac{(x-0)}{(2/3-0)}\frac{(x-1)}{(2/3-1)} = -\frac{9}{2}x(x-1)$$

$$l_2(x) = \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} = \frac{(x-0)}{(1-0)}\frac{(x-2/3)}{(1-2/3)} = 3x\left(x-\frac{2}{3}\right).$$

$$\begin{aligned} P_2(x) &= l_0(x)y_0 + l_1(x)y_1 + l_2(x)y_2 \\ &= \frac{3}{2}\left(x-\frac{2}{3}\right)(x-1) - \frac{9}{2}x(x-1)(0.5) + l_2(x)\cdot 0 \\ &= -\frac{3}{4}x^2 - \frac{1}{4}x + 1. \end{aligned}$$

Polynomial Interpolation

- Lagrange interpolation polynomials
  - Pros
    - Elegant formula
  - Cons
    - Slow computation; Computational time ~ $O(n^2)$
    - Not flexible; Just one point addition requires recalculation of the whole formula

$$P_n(x) = \sum_{i=0}^{n} \left[\prod_{j=0, j\neq i}^{n} \left(\frac{x - x_j}{x_i - x_j} \right) \right] \cdot y_i$$

Polynomial Interpolation

- Newton polynomials
  - a.k.a. Newton's divided differences interpolation polynomials
  - Main idea: flexible formula → a recursive form
    - Given $P_{k-1}(x)$ for k points → $P_k(x)$ for $k+1$ points
  - How?
    - $P_0(x) = y_0$ for (x_0, y_0)
    - $P_1(x) = y_0 + a_1(x - x_0)$ for (x_0, y_0) & (x_1, y_1)
    - $P_2(x) = y_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1)$ for (x_0, y_0), (x_1, y_1), & (x_2, y_2)

$$\vdots$$

    - $P_k(x) = P_{k-1}(x) + a_k(x - x_0)(x - x_1) \cdots (x - x_{k-1})$

Polynomial Interpolation

- Newton polynomials
  - Computing a_k
    - $P_1(x_1) = y_0 + a_1(x_1 - x_0) = y_1 \rightarrow a_1 = (y_1 - y_0)/(x_1 - x_0)$
    - $P_2(x_2) = P_1(x_2) + a_2(x_2 - x_0)(x_2 - x_1) = y_2$

    $\rightarrow a_2 = \frac{y_2 - P_1(x_2)}{(x_2 - x_0)(x_2 - x_1)}$

$$\vdots$$

$$a_k = \frac{y_k - P_{k-1}(x_k)}{(x_k - x_0)(x_k - x_1)\cdots(x_k - x_{k-1})}$$

$$P_k(x) = y_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_k(x - x_0)(x - x_1)\cdots(x - x_{k-1})$$

Polynomial Interpolation

- Newton polynomials
  - Newton's divided differences
    - $f[x_i] \coloneqq y_i$
    - $f\left[x_i, x_j\right] \coloneqq \frac{f[x_i]-f\left[x_j\right]}{x_i-x_j} = \frac{f\left[x_j\right]-f[x_i]}{x_j-x_i}$
    - $f\left[x_i, x_j, x_k\right] \coloneqq \frac{f\left[x_i,x_j\right]-f\left[x_j,x_k\right]}{x_i-x_k}$

      $\vdots$
    - $f[x_0, x_1, \dots, x_k] \coloneqq \frac{f[x_0,x_1,\cdots,x_{k-1}]-f[x_1,\cdots,x_{k-1},x_k]}{x_0-x_k}$
  - Computing a_k

    → Tidy if you use Newton's divided differences because $a_k = f[x_0, x_1, \dots, x_k]$

Polynomial Interpolation

- Newton polynomials
  - Computing a_k $(= f[x_0, x_1, \ldots, x_k])$

x_0	$f[x_0] = y_0$				
x_1	$f[x_1] = y_1$	$f[x_0, x_1] = \frac{f[x_1] - f[x_0]}{x_1 - x_0}$			
x_2	$f[x_2] = y_2$	$f[x_1, x_2] = \frac{f[x_2] - f[x_1]}{x_2 - x_1}$	$f[x_0, x_1, x_2]$		
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	
x_n	$f[x_n] = y_n$	$f[x_{n-1}, x_n] = \frac{f[x_n] - f[x_{n-1}]}{x_n - x_{n-1}}$	$f[x_{n-2}, x_{n-1}, x_n]$	$\ldots$	$f[x_0, x_1, \cdots, x_n]$

Table from Wen Shen

Polynomial Interpolation

- Example 2.3 in Wen Shen

x_i	0	1	$2/3$	$1/3$
y_i	1	0	$1/2$	0.866

$$
\begin{array}{c||c|c|c|c}
0 & \boxed{1} & & & \\
1 & 0 & \frac{0-1}{1-0} = \boxed{-1} & & \\
2/3 & 0.5 & \frac{0.5-0}{2/3-1} = -1.5 & \frac{-1.5-(-1)}{2/3-0} = \boxed{-0.75} & \\
1/3 & 0.866 & \frac{0.866-0.5}{1/3-2/3} = -1.098 & \frac{-1.098-(-1.5)}{1/3-1} = -0.603 & \frac{-0.603-(-0.75)}{1/3-0} = \boxed{0.441}
\end{array}
$$

$$
\begin{aligned}
P_3(x) &= a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1) + a_3(x-x_0)(x-x_1)(x-x_2) \\
&= \boxed{1} + \boxed{\text{-}1}\, x + \boxed{\text{-}0.75}\, x(x-1) + \boxed{0.441}\, x(x-1)(x-2/3).
\end{aligned}
$$

Polynomial Interpolation

- Newton polynomials
  - Nested form
    - $P_k(x) = y_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_k(x - x_0)(x - x_1)\cdots(x - x_{k-1})$
      $= y_0 + (x - x_0)(a_1 + (x - x_1)(a_2 + (x - 2)(a_3 + \cdots + a_k(x - x_{k-1}))) \ldots)$
    - Algorithm

$p = a_n$

for $k = n - 1, n - 2, \ldots, 0$

$\quad p = p(x - x_k) + a_k$

end

Polynomial Interpolation

- Newton polynomials
  - Computational complexity
    - Computing a_k: $O(n^2)$
    - Expanding a polynomial
      - Recursion form: $O(n^2)$
      - Nested form: $O(n)$

Polynomial Interpolation

- Fundamental theorem of algebra

Every polynomial of degree n, which is not identically zero, has exactly n roots (counting multiplicities. These roots may be real or complex). This means, in particular, if a polynomial of degree n has more than n distinct roots, then it must be identically zero.

Polynomial Interpolation

- **Theorem.** ***(Existence and Uniqueness of Polynomial Interpolation)***

Consider data points $(x_i, y_i)_{i=0}^{n}$, with the x_i's all distinct. Then there exists one and only polynomial $P_n(x)$ of degree $\leq n$ such that

$$P_n(x_i) = y_i, \qquad i = 0, 1, \cdots, n.$$

  - Proof) Wen Shen p. 29
    - Summary of the proof of the uniqueness
      - Assuming two distinct polynomials
      - Defining a difference function and proving that it is zero, by using the fundamental theorem of algebra

Polynomial Interpolation

- Errors in polynomial interpolation
  - Meaningful only if a function $f(x)$ (on the interval $a \leq x \leq b$) is the original function → what to be interpolated:

    $(x_i, f(x_i))$, $x_i \in [a, b], i = 0,1, \dots, n$.
  - Error function: $e(x) = f(x) - P_n(x)$
  - Interpolation error theorem (Wen Shen pp. 30~31)
    - For every $x \in [a, b]$, there exists some value $\xi \in [a, b]$, such that

$$e(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi) \prod_{i=0}^{n} (x - x_i)$$

Polynomial Interpolation

- Errors in polynomial interpolation
  - Interpolation error theorem
    - Proof

If f is a polynomial of degree n, $f(x) = P_n(x)$ by the uniqueness theorem of polynomial interpolation. Then $e(x) = 0$ and the proof is trivial.

Now consider the other case that f is not a polynomial of degree n.

If $x = x_i$, $e(x_i) = f(x_i) - P_n(x_i) = 0$.

Polynomial Interpolation

- Errors in polynomial interpolation
  - Interpolation error theorem
    - Proof

If $x \neq x_i$, we define

$$W(x) = \prod_{i=0}^{n} (x - x_i)$$

$$\varphi(x) = f(x) - P_n(x) - cW(x)$$

where $c = \frac{f(y) - P_n(y)}{W(y)}$ for a value y s.t. $a \leq y \leq b$ & $y \neq x_i$

$\varphi(x)$ has at least (n+2) roots because x_i's and y are roots.

Polynomial Interpolation

- Errors in polynomial interpolation
  - Interpolation error theorem
    - Proof

$\varphi(x)$ has at least (n+2) roots → $\varphi'(x)$ has at least (n+1) roots → $\varphi'''(x)$ has at least n roots → … → $\varphi^{(n+1)}(x)$ has at least 1 root.

So, we can call the root of $\varphi^{(n+1)}(x)$ as ξ.

$\varphi^{(n+1)}(\xi) = f^{(n+1)}(\xi) - 0 - cW^{(n+1)}(\xi) = 0.$

$W^{(n+1)}(x) = (n+1)!$ → $f^{(n+1)}(\xi) = cW^{(n+1)}(\xi) = \frac{f(y)-P_n(y)}{W(y)}\,(n+1)!$

$$\therefore e(x) = f(x) - P_n(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi) \prod_{i=0}^{n} (x - x_i)$$

Polynomial Interpolation

- Errors in polynomial interpolation
  - Uniform grid

$$x_i = a + ih, h = \frac{b-a}{n}$$

    - Lemma

$$\prod_{i=0}^{n} |x - x_i| \leq \frac{1}{4} h^{n+1} \cdot n!$$

    - Proof) Wen Shen p. 32
      - Summary of the proof
        - » $x = x_i$: trivial
        - » $x_i < x < x_{i+1}$: Find $\max|(x - x_i)(x - x_{i+1})|$
        - » Consider the other terms in the product

Polynomial Interpolation

- Errors in polynomial interpolation
  - Uniform grid

$$x_i = a + ih, h = \frac{b-a}{n}$$

    - Lemma

$$\prod_{i=0}^{n} |x - x_i| \leq \frac{1}{4} h^{n+1} \cdot n!$$

    - From this lemma, we can estimate the error

$$|e(x)| \leq \frac{1}{4(n+1)} \left| f^{(n+1)}(x) \right| h^{n+1} \leq \frac{M_{n+1}}{4(n+1)} h^{n+1}$$

$$M_{n+1} = \max_{x \in [a,b]} \left| f^{(n+1)}(x) \right| = \left\| f^{(n+1)} \right\|_{\infty}$$

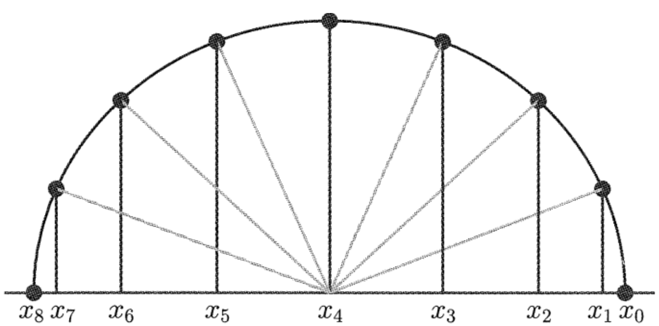
Polynomial Interpolation

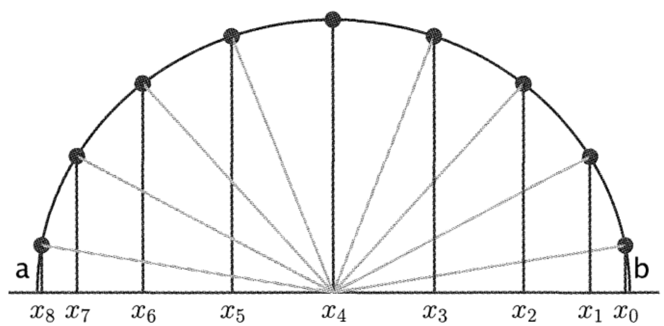
- Errors in polynomial interpolation
    - For an interval $[a, b]$
  - Chebyshev nodes of type I

$$\bar{x}_i = \frac{1}{2}(a+b) + \frac{1}{2}(b-a)\cos\left(\frac{i}{n}\pi\right)$$

  - Chebyshev nodes of type II

$$\bar{x}_i = \frac{1}{2}(a+b) + \frac{1}{2}(b-a)\cos\left(\frac{2i+1}{2n+2}\pi\right)$$





Figures from Wen Shen

Polynomial Interpolation

- Errors in polynomial interpolation
  - For an interval $[a, b]$
  - Chebyshev nodes of type I

$$\bar{x}_i = \frac{1}{2}(a+b) + \frac{1}{2}(b-a)\cos\left(\frac{i}{n}\pi\right)$$

  - Chebyshev nodes of type II

$$\bar{x}_i = \frac{1}{2}(a+b) + \frac{1}{2}(b-a)\cos\left(\frac{2i+1}{2n+2}\pi\right)$$

    - Error estimation? → from the property of cos and the interpolation error theorem

➢ Chebyshev nodes usually give smaller errors than the uniform grid.

Do It Yourself

- Consider $f(x) = \frac{1}{1+16x^2}$ on $[-1,1]$. Calculate upper bounds for error of polynomials interpolating this function with uniform grids of 4, 8, 16 nodes.
  - You may follow Wen Shen example 2.5 instead.

- [After this class]: Plot $f(x)$, the polynomials and their error functions.

Do It Yourself

- [After this class]: Consider $f(x) = \frac{1}{1+16x^2}$ on $[-1,1]$. Calculate upper bounds for error of polynomials interpolating this function with 4, 8, 16 Chebyshev nodes.
  - You may follow Wen Shen example 2.6 instead.

- [After this class]: Plot $f(x)$, the polynomials and their error functions.

Polynomial Interpolation

- Convergence
  - Generally, the error function $e(x) = f(x) - P_n(x)$ or $\int_a^b |e(x)|dx$ does not converge to zero as $n \to \infty$
  - The convergence depends on x_i's.
  - Increasing the order of the polynomial may not increase accuracy.

Polynomial Interpolation

- Disadvantages of polynomial interpolation
  - $P_n(x)$ should be n-times differentiable → Too smooth
  - Computational cost ~ $O(n^2)$
  - Big error in certain regions (especially near the limits)
  - Poor convergence

Spline Interpolation

- Connecting piecewise polynomials

- Properties
  - Correct interpolation
  - Low degree of smoothness (in comparison with polynomial interpolation)
  - Good convergence

- Usage examples
  - Visualization of discrete data
  - Graphic design

Spline Interpolation

- Data points: $(x_i, y_i), i = 0, \dots, n$ & $x_i < x_{i+1}$

- Spline function
  - Concatenation of piecewise polynomials

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

  - Continuous or smooth at $x = x_1, x_2, \dots, x_{n-1}$

Spline Interpolation

- Data points: $(x_i, y_i), i = 0, \ldots, n$ & $x_i < x_{i+1}$

- A spline of degree k
  - $S_i(x)$ is a polynomial of degree $\leq k$
  - $S(x)$ is $(k-1)$ times differentiable at $x = x_1, x_2, \ldots, x_{n-1}$ $(i = 1, \ldots, n-1)$

$$S_{i-1}(x_i) = S_i(x_i),$$
$$S'_{i-1}(x_i) = S'_i(x_i),$$
$$\vdots$$
$$S_{i-1}^{(k-1)}(x_i) = S_i^{(k-1)}(x_i).$$

Spline Interpolation

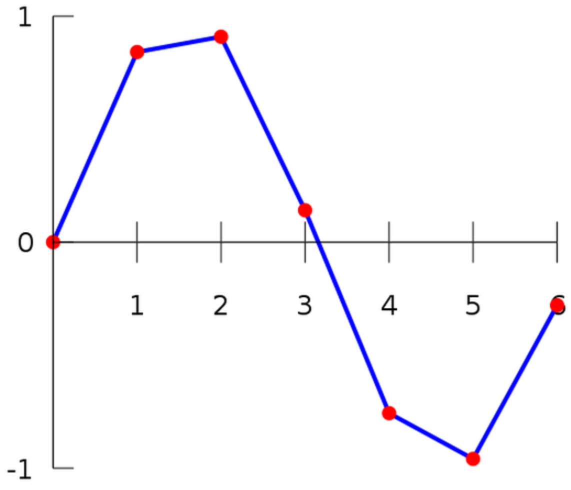
- Linear splines
  - Splines of degree 1

$$S_0(x_0) = y_0$$

$$S_{i-1}(x_i) = S_i(x_i) = y_i$$

$$(i = 1, \dots, n-1)$$

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



$$S_i(x) = y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i)$$

Figure from Wikipedia

Spline Interpolation

- Linear splines
  - Error theorem (Wen Shen pp. 50~51)
    - Let a function $f(x)$ (on the interval $a \le x \le b$) be the original function: $S(x_i) = f(x_i)$
    - If f' is continuous,

$$|f(x) - S(x)| \le \max_i \left\{ \frac{1}{2} h_i \max_{t_i \le x \le t_{i+1}} |f'(x)| \right\} \le \frac{1}{2} h \max_x |f'(x)|$$

    - If f'' is continuous,

$$|f(x) - S(x)| \le \max_i \left\{ \frac{1}{8} h_i^2 \max_{t_i \le x \le t_{i+1}} |f''(x)| \right\} \le \frac{1}{8} h^2 \max_x |f''(x)|$$

      - where $h = \max_i h_i$ and $h_i = x_{i+1} - x_i$

Spline Interpolation

- Natural cubic splines
  - Splines of degree 3 with end point conditions

$$
\begin{aligned}
S_i(x_i) &= y_i && (i = 0, 1, \dots, n-1) \\
S_i(x_{i+1}) &= y_{i+1} && (i = 0, 1, \dots, n-1) \\
S'_{i-1}(x_i) &= S'_i(x_i) && (i = 1, 2, \dots, n-1) \\
S''_{i-1}(x_i) &= S''_i(x_i) && (i = 1, 2, \dots, n-1)
\end{aligned}
$$

$$S''_0(x_0) = S''_{n-1}(x_n) = 0$$

  - How to find $S_i(x)$?
    - $S''_i(x)$ is a polynomial of degree 1 and $S''_{i-1}(x_i) = S''_i(x_i)$ → You can use the technique of linear splines

Spline Interpolation

- Natural cubic splines
  - Assume $S_i''(x_i) = z_i$ (of course, $z_0 = z_n = 0$)
  - Then $S_i''(x) = z_i + \frac{z_{i+1} - z_i}{h_i}(x - x_i)$
    - where $h_i = x_{i+1} - x_i$
  - But this form is not convenient. So, change its form

$$S_i''(x) = \frac{z_{i+1}}{h_i}(x - x_i) - \frac{z_i}{h_i}(x - x_{i+1})$$

  - Integrating twice,

$$S_i'(x) = \frac{z_{i+1}}{2h_i}(x - x_i)^2 - \frac{z_i}{2h_i}(x - x_{i+1})^2 + C_i$$

$$S_i(x) = \frac{z_{i+1}}{6h_i}(x - x_i)^3 - \frac{z_i}{6h_i}(x - x_{i+1})^3 + C_i x + D_i$$

Spline Interpolation

- Natural cubic splines
  - Continuity condition ($S_i(x_i) = y_i$ & $S_i(x_{i+1}) = y_{i+1}$) determines C_i & D_i

$$C_i = \frac{y_{i+1} - y_i}{h_i} - \frac{h_i}{6}(z_{i+1} - z_i)$$

  - The conditions for $S_i'(x)$ determines z_i's.

$$S_{i-1}'(x_i) = S_i'(x_i)$$

→ $$h_{i-1}z_{i-1} + 2(h_{i-1} + h_i)z_i + h_i z_{i+1} = 6(b_i - b_{i-1})$$

$$(i = 1, 2, \dots, n-1)$$

    - where $b_i = \frac{y_{i+1} - y_i}{h_i}$
    - Remember $z_0 = z_n = 0$

Spline Interpolation

- Natural cubic splines
  - → $\mathbf{Hz} = \mathbf{b}$ matrix-vector form

$$\mathbf{z} = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_{n-2} \\ z_{n-1} \end{pmatrix} \qquad \mathbf{b} = \begin{pmatrix} 6(b_1 - b_0) \\ 6(b_2 - b_1) \\ 6(b_3 - b_2) \\ \vdots \\ 6(b_{n-2} - b_{n-3}) \\ 6(b_{n-1} - b_{n-2}) \end{pmatrix}$$

$$\mathbf{H} = \begin{pmatrix} 2(h_0 + h_1) & h_1 & & & \\ h_1 & 2(h_1 + h_2) & h_2 & & \\ & h_2 & 2(h_2 + h_3) & h_3 & \\ & \ddots & \ddots & \ddots & \\ & & h_{n-3} & 2(h_{n-3} + h_{n-2}) & h_{n-2} \\ & & & h_{n-2} & 2(h_{n-2} + h_{n-1}) \end{pmatrix}$$

Spline Interpolation

- Natural cubic splines
  - Algorithm
    ① Set up the (tridiagonal) matrix-vector equation
    ② Solve for z_i's
    ③ Complete $S_i(x)$
  - Computational cost: $O(n)$

Spline Interpolation

- Theorem on natural cubic splines
  - *Let S be the natural cubic spline function that interpolates a twice-continuously differentiable function f at knots*

$$a = x_0 < x_1 < \cdots < x_n = b$$

then

$$\int_a^b [S''(x)]^2 dx \leq \int_a^b [f''(x)]^2 dx$$

- ➢ The natural cubic spline gives the least curvature among all interpolating functions.
- ➢ **Most natural**

Spline Interpolation

- Theorem on natural cubic splines
  - Proof

Let $g(x) = f(x) - S(x) \rightarrow g(x_i) = 0 \ (i = 0, 1, \dots, n)$

$f'' = S'' + g'' \rightarrow (f'')^2 = (S'')^2 + (g'')^2 + 2S''g''$

$$\Longrightarrow \int_a^b (f'')^2 dx = \int_a^b (S'')^2 dx + \int_a^b (g'')^2 dx + 2\int_a^b S''g''dx$$

$$\int_a^b S''g''dx = -\int_a^b S'''g'dx = \sum_{i=0}^{n-1} C_i \int_{x_i}^{x_{i+1}} g'dx = \sum_{i=0}^{n-1} C_i[g(x_{i+1}) - g(x_i)]$$
$$= 0$$

where $C_i = \frac{z_{i+1} - z_i}{h_i}$ $[S_i''(x_i) = z_i$ & $h_i = x_{i+1} - x_i]$

Spline Interpolation

- Theorem on natural cubic splines
  - Proof

$$\int_a^b (f'')^2 dx = \int_a^b (S'')^2 dx + \int_a^b (g'')^2 dx$$

$$\therefore \int_a^b (f'')^2 dx \geq \int_a^b (S'')^2 dx$$

References

- Wen Shen,
  An Introduction to Numerical Computation
- C. Moler,
  Numerical Computing with MATLAB
- Wikipedia

Further Study

- **Lebesgue constants**

- **Piecewise constant** interpolation and **quadratic Splines**