Interpolation and polynomial Approximation

- need to estimate intermediate values between precise data points. The most common method used is polynomial interpolation

The topics covered are

- Evaluating polynomial coefficients with (working) simultaneous equations ~~which~~ results in a ill-conditioned Vandermonde matrix which is known to have large condition number.

- Knowing how to perform an interpolation with a Lagrange polynomial

- Recognizing that higher-order polynomials can manifest large errors

- Understanding that splines minimize errors by fitting lower-order polynomials to data in piecewise fashion

- Recognizing why cubic splines are preferable.
- Understanding natural & clamped end conditions.
- Knowing how to compute the coefficients of a cubic spline.

Polynomial Interpolation

Statement of the problem:

Given $(x_0, f(x_0)), (x_1, f(x_1)), \dots, (x_n, f(x_n))$ points ($f(x)$ is the unknown), determine a polynomial $P(x)$ s.t. $P(x_i) = f(x_i)$ for all i. It is assumed that $x_0 < x_1 < x_2 < \cdots < x_n$.

Consider two solutions:

A) Suppose $P(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_n x^n$

Then $c = (c_0, c_1, \dots, c_n)^T$ can be computed by solving:

$$Ac = b \quad \text{where}$$

$$A = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ & & \vdots & & \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix}_{(n+1)\times(n+1)} \quad \text{and} \quad b^T = (f(x_0), f(x_1), \dots, f(x_n))^T$$

$$\det A = \prod_{0 \le i < j \le n} (x_j - x_i)$$

A is non-singular but could have large condition number.

Complexity of computing c is $O(n^3)$

(B) Lagrange Polynomial

$$P(x) = f(x_0) \cdot L_{n,0}(x) + f(x_1) \cdot L_{n,1}(x) + \cdots + f(x_n) \cdot L_{n,n}(x)$$

where

$$L_{n,k}(x) = \frac{(x-x_0)(x-x_1)\cdots(x-x_{k-1})(x-x_{k+1})\cdots(x-x_n)}{(x_k-x_0)(x_k-x_1)\cdots(x_k-x_{k-1})(x_k-x_{k+1})\cdots(x_k-x_n)}$$

$\therefore L_{n,k}(x_k) = 1$ & $L_n(x_j) = 0 \quad \forall j \neq k$

& $L_{n,k}(x)$ is a polynomial of degree n.

$$\therefore P(x) = \sum_{j=0}^{n} L_{n,j}(x) \cdot f(x_j)$$

$$= \sum_{j=0}^{n} \underbrace{\frac{f(x_j)}{\prod_{\substack{k=0 \\ k\neq j}} (x_j - x_k)}}_{=\alpha_j} \cdot \frac{\prod_{k=0}^{n} (x-x_k)}{(x-x_j)} \; q(x)$$

$$= q(x) \sum_{j=0}^{n} \frac{\alpha_j}{x-x_j}.$$

Complexity : • Computing $\alpha_0, \alpha_1, \cdots, \alpha_n$ costs $O(n^2)$ in total.

Evaluating $P(x)$ for some x (α_j's are known)

• Compute $q(x) = \prod_{j=0}^{n} (x - x_j)$: $O(n)$ time

• Now $P(x) = q(x) \sum_{j=0}^{n} \frac{\alpha_j}{x - x_j}$ can be computed in $O(n)$ time.

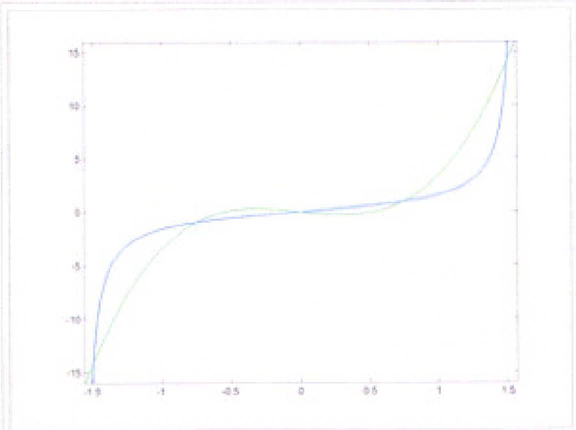
Problem with Lagrange Polynomial

• Each time x_k changes, all Lagrange polynomial changes

• Evaluation of $P(x)$ is not efficient.

Example

We wish to interpolate $f(x) = \tan(x)$ at the points



The tangent function and its interpolant

$x_0 = -1.5 \qquad f(x_0) = -14.1014$

$x_1 = -0.75 \qquad f(x_1) = -0.931596$

$x_2 = 0 \qquad f(x_2) = 0$

$x_3 = 0.75 \qquad f(x_3) = 0.931596$

$x_4 = 1.5 \qquad f(x_4) = 14.1014$

Now, the basis polynomials are:

$$L_{4,0}(x) = \ell_0(x) = \frac{x-x_1}{x_0-x_1}\cdot\frac{x-x_2}{x_0-x_2}\cdot\frac{x-x_3}{x_0-x_3}\cdot\frac{x-x_4}{x_0-x_4} = \frac{1}{243}x(2x-3)(4x-3)(4x+3)$$

$$L_{4,1}(x) = \ell_1(x) = \frac{x-x_0}{x_1-x_0}\cdot\frac{x-x_2}{x_1-x_2}\cdot\frac{x-x_3}{x_1-x_3}\cdot\frac{x-x_4}{x_1-x_4} = -\frac{8}{243}x(2x-3)(2x+3)(4x-3)$$

$$\ell_2(x) = \frac{x-x_0}{x_2-x_0}\cdot\frac{x-x_1}{x_2-x_1}\cdot\frac{x-x_3}{x_2-x_3}\cdot\frac{x-x_4}{x_2-x_4} = \frac{1}{243}(243-540x^2+192x^4)$$

$$\ell_3(x) = \frac{x-x_0}{x_3-x_0}\cdot\frac{x-x_1}{x_3-x_1}\cdot\frac{x-x_2}{x_3-x_2}\cdot\frac{x-x_4}{x_3-x_4} = -\frac{8}{243}x(2x-3)(2x+3)(4x+3)$$

$$L_{4,4}(x) = \ell_4(x) = \frac{x-x_0}{x_4-x_0}\cdot\frac{x-x_1}{x_4-x_1}\cdot\frac{x-x_2}{x_4-x_2}\cdot\frac{x-x_3}{x_4-x_3} = \frac{1}{243}x(2x+3)(4x-3)(4x+3)$$

Thus the interpolating polynomial then is

$$\frac{1}{243}\Big(f(x_0)x(2x-3)(4x-3)(4x+3) - 8f(x_1)x(2x-3)(2x+3)(4x-3)$$
$$+f(x_2)(243-540x^2+192x^4) - 8f(x_3)x(2x-3)(2x+3)(4x+3)$$
$$+f(x_4)x(2x+3)(4x-3)(4x+3)\Big)$$

~~$= -1.47748x + 4.82456x^3$~~ 47748

A simpler polynomial (Newton's approach)

(not discussed in the class, but will be covered in tutorial.)

Let $P_k(x)$ denote the interpolating polynomial determined by the first $k+1$ points $(x_0, f(x_0)), (x_1, f(x_1)), \cdots, (x_k, f(x_k))$.

i.e. $P_k(x_j) = f(x_j)$, $j = 0, 1, 2, \cdots, k$ & $P_k(x)$ is a polynomial of degree k.

Let $P_{k+1}(x)$ be the interpolating polynomial of $(x_0, f(x_1)), (x_1, f(x_2)), \cdots, (x_{k+1}, f(x_{k+1}))$. where

(1) — $P_{k+1}(x) = P_k(x) + a_{k+1} \cdot (x - x_0)(x - x_1) \cdots (x - x_k)$

where a_{k+1} is a coefficient to be determined.

$\therefore$ $P_{k+1}(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_{k+1}(x - x_0) \cdots (x - x_k)$

We want $P_{k+1}(x_{k+1}) = f(x_{k+1})$. Therefore,

(2) — $$a_{k+1} = \frac{f(x_{k+1}) - P_k(x_{k+1})}{(x_{k+1} - x_0)(x_{k+1} - x_1) \cdots (x_{k+1} - x_k)}$$

$\therefore$ (1) with (2) gives interpolating polynomial of first $k+2$ points $(x_j, f(x_j))$, $j = 0, 1, 2, \cdots, k+1$.

where $P_{k+1}(x_j) = f(x_j)$, $j = 0, 1, 2, \cdots$ & $k+1$

$$\therefore P_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1)$$
$$+ \cdots + a_{k+1}(x - x_0)(x - x_1) \cdots (x - x_k).$$

Evaluating $P_n(x)$

Brute force : $O(n^2)$

Easily doable in $O(n)$ by computing

$(x - x_0)$; $(x - x_0)(x - x_1)$; $(x - x_0)(x - x_1)(x - x_2)$;

in order. (Why?)

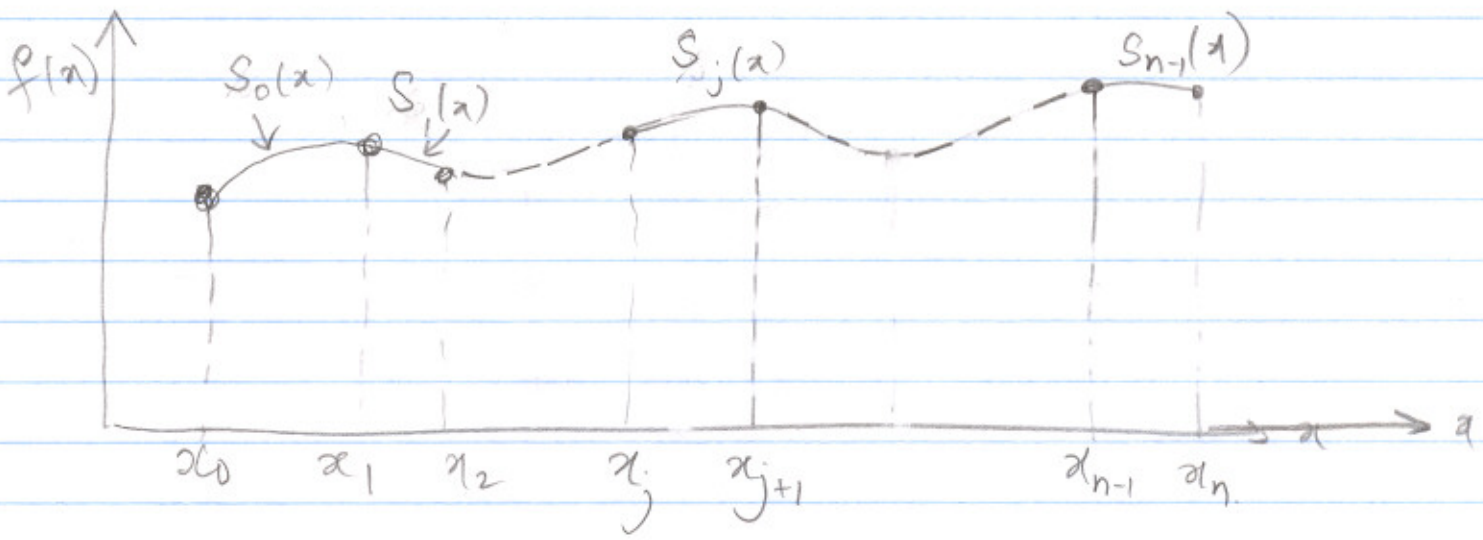
```
p ← 1
sum ← a0
for k ← 0 to n-1 do
    p ← (x - xk) * p
    sum ← sum + p
```

Notes

The Lagrange form of the interpolation polynomial shows the linear character of polynomial interpolation & the uniqueness of the interpolation polynomial. Therefore, it is preferred in proofs & theoretical arguments. But as can be seen from the construction, $L_{n,j}(x)$ changes for all j when any x_k changes. A better form of the interpolation polynomial for a practical purpose is the Newton polynomial. Like Lagrange, Newton polynomial can be evaluated efficiently using nested multiplication.

Cubic Spline.

The Notations used for splines is shown below



Each $S_j(x)$ is a cubic polynomial of the form $S_j(x) = a_j + b_j(x-x_j) + c_j(x-x_j)^2 + d_j(x-x_j)^3$ for $j = 0, 1, 2, \cdots, n-1$.

The collection of the cubic polynomials is denoted by the polynomial $S(x) = \langle S_0(x), S_1(x), \cdots, S_{n-1}(x) \rangle$.

The formal definition of a cubic spline interpolant is described in the text (Def^n 3.10, pp 139). The condition (c) can be ignored since (c) can be deduced from (b) directly

We use h_j to denote $x_{j+1} - x_j$, $j = 0, 1, \cdots, n-1$.

From condition (b) of the def^n (Defn. 3.10) we get

$$a_j = f(x_j), \quad j = 0, 1, \cdots, n-1. \quad \text{— (1)}$$

From condition (c) of the def^n (Def. 3.10)

$$a_{j+1} = a_j + b_j h_j + c_j h_j^2 + d_j h_j^3 \quad \text{— (2)}$$

$$j = 0, 1, 2, \cdots, n-2.$$

From condition (d) of the def^n (Defn 3.10),

$$b_{j+1} = b_j + 2c_j h_j + 3d_j h_j^2, \quad j = 0, 1, \cdots n-2. \quad \text{— (3)}$$

From condition (e) of Def^n 3.10,

$$c_{j+1} = c_j + 3d_j h_j, \quad j = 0, 1, 2, \cdots, n-2 \quad \text{— (4)}$$

Solving (4) for d_j, we get

$$d_j = \frac{c_{j+1} - c_j}{3h_j}, \quad j = 0, 1, 2, \cdots, n-2 \quad \text{— (5)}$$

Substituting d_j in (4) to the equations ② & ③ we get

$$a_{j+1} = a_j + b_j h_j + \frac{h_j^2}{3}(2c_j + c_{j+1}), \quad j = 0, 1, 2, \cdots, n-2$$

or $$b_j = \frac{1}{h_j}(a_{j+1} - a_j) - \frac{h_j}{3}(2c_j + c_{j+1}), \quad j = 0, 1, 2, \cdots, n-2 \quad \text{— (6)}$$

& ~~$b_{j+1} = b_j + 2c_j h_j + 3h_j^2$~~

$$b_{j+1} = b_j + h_j(c_j + c_{j+1}), \quad j = 0, 1, 2, \cdots, n-2. \quad \text{— (7)}$$

Index of (6) can be reduced by 1 to obtain

$$b_{j-1} = \frac{1}{h_{j-1}}(a_j - a_{j-1}) - \frac{h_{j-1}}{3}(2c_{j-1} + c_j), \quad \text{— (8)}$$

$$j = 1, 2, \cdots, n-1$$

Similarly, the index of (7) can be reduced to obtain

$$b_j = b_{j-1} + h_{j-1}(c_{j-1} + c_j), \quad j = 1, 2, \cdots, n-1 \quad \text{— (9)}$$

Equations (6) & (8) can be substituted in (9) to obtain

$$h_{j-1}c_{j-1} + 2(h_{j-1} + h_j)c_j + h_j c_{j+1} = \frac{3}{h_j}(a_{j+1} - a_j) - \frac{3}{h_{j-1}}(a_j - a_{j-1}) \quad \text{— (10)}$$

$$j = 1, 2, \cdots, n-1.$$

Equation (10) can be written for interior knots (i.e at $x_1, x_2, \cdots, x_{n-1}$). These equations can be combined to write:

$$\begin{bmatrix} h_0 & 2(h_0+h_1) & h_1 & & & & \\ & h_1 & 2(h_1+h_2) & h_3 & & & \\ & & h_2 & 2(h_2+h_3) & h_4 & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_n \end{bmatrix}$$

$$= \begin{bmatrix} \frac{3}{h_1}(a_2-a_1) - \frac{3}{h_0}(a_1-a_0) \\ \frac{3}{h_2}(a_3-a_2) - \frac{3}{h_1}(a_2-a_1) \\ \vdots \\ \frac{3}{h_{n-1}}(a_n-a_{n-1}) - \frac{3}{h_{n-2}}(a_{n-1}-a_{n-2}) \end{bmatrix}$$

The matrix is $(n-2 \times n)$ matrix. We need to add two more rows before we can solve the system.

We now consider boundary conditions. Let us consider only the natural boundary i.e. condition f(i) of Def^n 3.10.

Condition $S''(x_0) = 0$

i.e. $S_0''(x_0) = 0$

where $S_0(x_0) = a_0 + b_0(x - x_0) + c_0(x - x_0)^2 + d_0(x - x_0)^3$

i.e. $S_0''(x_0) = 2c_0 = 0$ — (11)

Also condition $S''(x_n) = 0$

i.e. $S_{n-1}''(x_n) = 0$ gives

$c_{n-1} + 3d_{n-1}h_n = 0$

We call $c_n = c_{n-1} + 3d_{n-1}h_n = 0$ (12)

The system $Ac = b$ is a tridiagonal system. Moreover, A is a strictly diagonally dominant matrix, and therefore Gaussian elimination without pivoting can be used. Solving this system therefore requires linear time to determine $[c_0, c_1, \cdots, c_n]^T$

Thus $d_0, d_1, \cdots, d_{n-2}$ can be computed in (5). d_{n-1} can be computed using equation (12).

Similarly, $b_0, b_1, \cdots, b_{n-1}$ can be computed using equation (3) once c_j & d_j, $j = 0, \cdots, n-1$ are all known.

Therefore, for natural boundary,
we need to solve $Ac = b$ for c
where

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

$$c^T = [c_0, c_1, \cdots, c_n]^T$$

$$\& \quad b = \begin{bmatrix} 0 \\ \frac{3}{h_1}(a_2-a_1) - \frac{3}{h_0}(a_1-a_0) \\ \frac{3}{h_2}(a_3-a_2) - \frac{3}{h_1}(a_2-a_1) \\ \\ \frac{3}{h_{n-1}}(a_n-a_{n-1}) - \frac{3}{h_{n-2}}(a_{n-1}-a_{n-2}) \\ 0 \end{bmatrix}$$