Cost is not terrible

Polynomial Interpolation (PI)

Reading: Cheney and Kincaid, Sections 4.1 & 4.2

Problem

Given $n+1$ points (x_0, y_0), (x_1, y_1),...,(x_n, y_n) or a table

x	x_0	x_1	$\cdots$	x_n
y	y_0	y_1	$\cdots$	y_n

where x_i are distinct, find a polynomial $p(x)$ with least degree such that $p(x_i) = y_i$ for $0 \leq i \leq n$, i.e., the polynomial curve passes through the given points. Here x_i are called **nodes**, and p is said to **interpolate** the $n+1$ points on the table.

The Vandermonde Approach

Theorem.There is a *unique* polynomial p of degree $\leq n$ such that $p(x_i) = y_i, \quad 0 \leq i \leq n$.

Pf. Let $p(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_n x^n$, where the coefficients c_i are to be determined. Set $p(x_i) = y_i$, then

$$\begin{aligned} c_0 + c_1 x_0 + c_2 x_0^2 + \dots c_n x_0^n &= y_0 \\ c_0 + c_1 x_1 + c_2 x_1^2 + \dots c_n x_1^n &= y_1 \\ &\dots\dots\dots\dots \\ c_0 + c_1 x_n + c_2 x_n^2 + \dots c_n x_n^n &= y_n \end{aligned}$$

Write the linear system as $Ac = y$:

prove uniqueness
without using lagrange

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdot & x_0^n \\ 1 & x_1 & x_1^2 & \cdot & x_1^n \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ 1 & x_n & x_n^2 & \cdot & x_n^n \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \cdot \\ c_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \cdot \\ y_n \end{bmatrix},$$

where A is called the Vandermonde matrix and

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

Thus A is nonsingular and $Ac = y$ has a unique solution $c = A^{-1}y$. $\sharp$

The above proof provides a method to compute the coefficients of the interpolating polynomial:

Algorithm:

Step 1: Form the linear system $Ac = y$.

Step 2: Solve $Ac = y$ by GEPP.

X **Cost analysis**: In step 1, $A(:, j+1) = A(:, j) \,.\!* \begin{bmatrix} x_0 \\ x_1 \\ \cdot \\ x_n \end{bmatrix}$ for $j = 2, \ldots, n$.

It needs $(n+1)(n-1) \approx n^2$ flops. In step 2, $\frac{2}{3}n^3$ flops are required. Total cost: $\frac{2}{3}n^3$ flops.

Note: Since A has structures, $Ac = y$ can be solved by some fast algorithms, costing as low as $O(n \log^2 n)$ flops.

Evaluating $p(x)$:
Nested multiplication

evaluate new point

$$\begin{aligned} p(x) &= c_0 + c_1 x + c_2 x^2 + \ldots c_n x^n \\ &= c_0 + x(c_1 + x(c_2 + \ldots + x(c_{n-1} + x c_n) \cdots)). \end{aligned}$$

Procedure for evaluating $p(x)$ for some x:

$p \leftarrow c_n$
for $i = n-1 : -1 : 0$
  $p \leftarrow c_i + xp$
end

Cost: $2n$ flops.

MATALB built-in function for polynomial interpolation: `polyfit(x,y,n)` . It finds the coefficients c_i.

The Lagrange Approach

The Lagrange form of the interpolating polynomial:

$$p(x) = \sum_{i=0}^{n} l_i(x) y_i,$$

where $l_i(x)$ is the cardinal polynomial defined as

property

$$l_i(x) = \frac{(x-x_0)\cdots \boxed{(x-x_{i-1})(x-x_{i+1})} \cdots (x-x_n)}{(x_i-x_0)\cdots \boxed{(x_i-x_{i-1})(x_i-x_{i+1})} \cdots (x_i-x_n)}, \qquad l_i(x_j) = \begin{cases} 0, & i \neq j \\ 1, & i = j \end{cases}$$

Obviously $p(x)$ defined above is a polynomial of degree $\leq n$ and $p(x_i) = y_i$ for $0 \leq i \leq n$.

We can rewrite $p(x)$:

$$p(x) = \sum_{i=0}^{n} l_i(x) y_i = \sum_{i=0}^{n} \frac{y_i}{\Pi_{j=0, j\neq i}^{n}(x_i - x_j)} \cdot \frac{\Pi_{j=0}^{n}(x - x_j)}{x - x_i} = q(x) \sum_{i=0}^{n} \frac{c_i}{x - x_i}$$

where $q(x) \equiv \Pi_{j=0}^{n}(x - x_j)$ and $c_i \equiv \frac{y_i}{\Pi_{j=0, j\neq i}^{n}(x_i - x_j)}$.

Cost of finding $c_0, c_1, \ldots, c_n$**:**
For each i, computing c_i needs 1 division, n subtractions, $n-1$ multiciplations, a total of $2n$ flops. So computing all c_i needs $2n * (n+1) \approx 2n^2$ flops.

Cost of evaluating $p(x)$ **for some** x (given c_i for $i = 0, \ldots, n$):
Computing $q(x)$ needs $2n+1$ flops. Computing $\frac{c_i}{x - x_i}$ needs 2 flops for each i. Thus computing $p(x)$ needs a total of $(2n+1) + ((n+1) * 2 + n) + 1 \approx 5n$ flops.
Note: In practice we usually do not use the Lagrange approach, since the evaluation of $p(x)$ is not efficient enough.

The Newton Approach

Idea: Suppose a polynomial $p_k(x)$ of degree at most k has been found to interpolate (x_0, y_0), (x_1, y_1),...,(x_k, y_k). We seek a polynomial $p_{k+1}(x)$ of degree at most $k+1$ to interpolate (x_0, y_0), (x_1, y_1),..., (x_k, y_k), (x_{k+1}, y_{k+1}).

Let $p_{k+1}(x) = p_k(x) + a_{k+1}(x - x_0)(x - x_1)\ldots(x - x_k)$, where a_{k+1} is to be determined. Obviously we have

$$p_{k+1}(x_i) = p_k(x_i) = y_i, \quad 0 \le i \le k.$$

Setting $p_{k+1}(x_{k+1}) = y_{k+1}$, we obtain

$$a_{k+1} = \frac{y_{k+1} - p_k(x_{k+1})}{(x_{k+1} - x_0)(x_{k+1} - x_1)\cdots(x_{k+1} - x_k)}.$$

This $p_{k+1}(x)$ with the above a_{k+1} interpolates $(x_0, y_0), \ldots, (x_{k+1}, y_{k+1})$. Also notice that $p_{k+1}(x)$ is a polynomial of degree at most $k+1$. So it is what we seek.

The Newton form of the interpolating polynomial:

$$p_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \ldots + a_n(x - x_0)(x - x_1)\cdots(x - x_{n-1}).$$

Evaluating $p_n(x)$:

Nested multiplication

$$\begin{aligned} p_n(x) &= a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \ldots + a_n(x - x_0)(x - x_1)\cdots(x - x_{n-1}) \\ &= a_0 + (x - x_0)(a_1 + (x - x_1)(a_2 + \cdots (x - x_{n-2})(a_{n-1} + (x - x_{n-1})a_n)\cdots)) \end{aligned}$$

Procedure for evaluating $p_n(x)$ for some x:

$p \leftarrow a_n$
for $i = n-1 : -1 : 0$
  $p \leftarrow a_i + (x - x_i)p$
end

Cost of this procedure: $3n$ flops.

Cost of computing a_1, a_2, $\ldots a_n$ by

$$a_{k+1} = \frac{y_{k+1} - \overset{3k}{p_k(x_{k+1})}}{(x_{k+1} - x_0)(x_{k+1} - x_1)\cdots(x_{k+1} - x_k)}.$$

Cost of computing a_{k+1}: $(1 + 3k) + [(k+1) + k] + 1 = 5k + 3$ flops.

Total cost: $\sum_{k=0}^{n-1}(5k+3) = \frac{5}{2}n^2 + \frac{1}{2}n \approx \frac{5}{2}n^2$ flops.

A more efficient method for computing $a_0, a_1, \ldots, a_n$

Since

$$p_n(x) = \sum_{i=0}^{n} a_i \prod_{j=0}^{i-1}(x - x_j)$$

interpolates (x_i, y_i) for $i = 0, 1 \ldots, n$, we have

$$p_n(x_i) = y_i, \quad i = 0, 1 \ldots, n.$$

This gives the linear system $Aa = y$:

$$\begin{bmatrix} 1 & & & & \\ 1 & x_1 - x_0 & & & \\ 1 & x_2 - x_0 & \prod_{j=0}^{1}(x_2 - x_j) & & \\ \cdot & \cdot & \cdot & \cdot & \\ 1 & x_n - x_0 & \prod_{j=0}^{1}(x_n - x_j) & \cdot & \prod_{j=0}^{n-1}(x_n - x_j) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \cdot \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \cdot \\ y_n \end{bmatrix},$$

Notice A is a lower triangular matrix and its diagonal elements are nonzero, so A is nonsingular and $Aa = y$ has a unique solution $a = A^{-1}y$.
Since A has special structure, we can design an efficient algorithm to compute the solution a.
The pattern of finding $a_0, a_1, \ldots, a_n$:

for $k = 0 : n-1$
  $a_k \leftarrow y_k$ (updated y_k)
  for $i = k+1 : n$
    subtract equation k from equation i
    and divide equation i by $x_i - x_k$
  end
end
$a_n \leftarrow y_n$

Notice when we update the equations we need only keep track of changes in the y vector.

Algorithm. Given x_i, y_i, find a_i $(i = 0, \ldots, n)$:

for $k = 0 : n-1$
  $a_k \leftarrow y_k$
  for $i = k+1 : n$
    $y_i \leftarrow (y_i - y_k)/(x_i - x_k)$
  end
end
$a_n \leftarrow y_n$

x	y (=f(x))	1	2	3
0	3 a_0			
1	4	1 a_1		
2	7	2	1 a_2	
4	19	4	1	0 a_3

$P_3(x) = 3 + (x-0) + (x-0)(x-1)$

Note: The computation of all a_i can be done in a table. An example will be given in class.

Cost: $\sum_{k=0}^{n-1} 3(n-k) = \frac{3}{2}n(n+1) \approx \frac{3}{2}n^2$ flops.

Consider $n=3$

$$P_3(x) = a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1) + a_3(x-x_0)(x-x_1)(x-x_2)$$

a_0

$$\begin{bmatrix} 1 & & & \\ 1 & x_1-x_0 & & \\ 1 & x_2-x_0 & (x_2-x_0)(x_2-x_1) & \\ 1 & x_3-x_0 & (x_3-x_0)(x_3-x_1) & (x_3-x_0)(x_3-x_1)(x_3-x_0) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

a_1

$$\begin{bmatrix} 1 & & & \\ 0 & \cancel{x_1-x_0}\ 1 & & \\ 0 & \cancel{x_2-x_0}\ 1 & (x_2-\cancel{x_0})(x_2-x_1) & \\ 0 & \cancel{x_3-x_0}\ 1 & (x_3-\cancel{x_0})(x_3-x_1) & (x_3-\cancel{x_0})(x_3-x_1)(x_3-x_0) \end{bmatrix} \begin{bmatrix} y_0 \\ (y_1-y_0)/(x_1-x_0) = y_1 \\ (y_2-y_0)/(x_2-x_0) = y_2 \\ (y_3-y_0)/(x_3-x_0) = y_3 \end{bmatrix}$$

a_3

$$\begin{bmatrix} 1 & & & \\ 0 & 1 & & \\ 0 & 0 & \cancel{(x_2-x_1)}\ 1 & \\ 0 & 0 & \cancel{(x_3-x_1)}\ 1 & (\cancel{x_3-x_1})(x_3-x_0) \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ (y_2-y_1)/(x_3-1) = y_2 \\ (y_3-y_1)/(x_3-x_1) = y_3 \end{bmatrix}$$

a_4.

$$\begin{bmatrix} 1 & & & \\ 0 & 1 & & \\ 0 & 0 & 1 & \\ 0 & 0 & 0 & \cancel{(x_3-x_0)}\ 1 \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ (y_3-y_2)/(x_3-x_0) = y_3 \end{bmatrix}$$

Dangers of High Degree Polynomial Interpolation

Let $y = f(x)$. We approximate f on $[a, b]$ by an interpolating polynomial p at $n+1$ nodes, i.e.,

$$p(x_i) = y_i = f(x_i).$$

Q. Is it true that f will be well approximate at all intermediate points as the number of nodes increases?

Answer: No. The Runge function: Runge Phenomenon

$$f(x) = 1/(1 + 25x^2), \quad x \in [-1, 1].$$

If p_n is the polynomial that interpolates the f at $n+1$ equally spaced points on $[-1, 1]$, then

$$\lim_{n\to\infty} \max_{-1\leq x\leq 1} |f(x) - p_n(x)| = \infty$$

So high-degree PI should generally be avoided.

Interpolation error theorem: If p is the polynomial of degree at most n that interpolates f at the $n+1$ distinct nodes $x_0, x_1, \ldots, x_n$ belonging to $[a, b]$ and if $f^{(n+1)}$ is continuous. Then for any x in $[a, b]$, there is z_x in (a, b) for which

$$f(x) - p(x) = \frac{1}{(n+1)!} f^{(n+1)}(z_x) \Pi_{i=0}^{n} (x - x_i).$$

useful for later