

Interpolation

Problem setting: Given $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$, $x_1 < x_2 < \dots < x_n$, construct a function f :

$$f(x_i) = y_i, \quad i = 1, 2, \dots, n.$$

For the simplicity of discussion, sometimes we assume $i = 0, 1, \dots, n$.

Desirable properties of f :

- smooth: analytic and $|f''(x)|$ not too large
- simple: polynomial of minimum degree

1 Polynomial Interpolation

Advantages: easy to evaluate and differentiate

Weierstrass Approximation Theorem:

If f is any continuous function on the finite closed interval $[a, b]$, then for every $\epsilon > 0$ there exists a polynomial $p_n(x)$ of degree $n = n(\epsilon)$ such that

$$\max_{x \in [a, b]} |f(x) - p_n(x)| < \epsilon.$$

Although the proof of Weierstrass theorem is constructive, this method is impractical because the degree of the polynomial is often too high.

2 Global Polynomial Interpolation

Use one interpolating polynomial for the entire interval.

A naive approach

Given $(x_0, y_0), \dots, (x_n, y_n)$, we construct the linear system:

$$\begin{bmatrix} 1 & x_0 & \cdots & x_0^n \\ 1 & x_1 & \cdots & x_1^n \\ \vdots & \vdots & \cdots & \vdots \\ 1 & x_n & \cdots & x_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}.$$

Solve for the coefficients a_i of the polynomial

$$p_n(y)(x) = a_0 + a_1x + \cdots + a_nx^n$$

- Unique solution, since x_i are distinct. The Vandermonde matrix is non-singular.
- Problems: the coefficient matrix is ill-conditioned; the calculation of the powers can easily overflow or underflow.

What is the condition number of the coefficient matrix constructed by $x_i = 1900 + 10i$, $i = 0, 1, \dots, 7$?

A more practical way: Lagrange polynomials

Basis of polynomials: $\{l_j(x)\}$ ($j = 0, 1, \dots, n$) of degree n such that

$$l_j(x_i) = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}$$

Construct

$$l_j(x) = \prod_{i \neq j} \frac{x - x_i}{x_j - x_i},$$

thus

$$p_n(y)(x) = \sum_{j=0}^n l_j(x)y_j$$

is the interpolating polynomial.

- The Lagrange polynomials are easy to construct, but expensive to evaluate.
- The naive approach is hard to compute, but easy to evaluate. The most efficient and stable way to evaluate a polynomial $p_n(x) = a_nx^n + \cdots + a_1x + a_0$ is the following algorithm.

Algorithm. Evaluate the polynomial $a_nx^n + \cdots + a_1x + a_0$ at $x = t$.

```

p = a_n;
for i = n - 1 to 0
    p = p * t + a_i;
end

```

The Newton Form

This is an interpolation method which is not so difficult to construct and not so hard to evaluate. In this method, we use the basis

$$1, x - x_0, (x - x_0)(x - x_1), \dots, (x - x_0)(x - x_1) \cdots (x - x_{n-1})$$

and write the interpolating polynomial as a linear combination of the basis:

$$p_n(x) = c_0 + c_1(x - x_0) + \cdots + c_n(x - x_0)(x - x_1) \cdots (x - x_{n-1}).$$

Divided differences

How do we determine the coefficients c_k ? The coefficients c_k form the solution of the following lower triangular system

$$\begin{bmatrix} 1 & & & & \\ 1 & x_1 - x_0 & & & \\ \vdots & \vdots & \ddots & & \\ 1 & x_n - x_0 & \dots & (x_n - x_0)(x_n - x_1) \cdots (x_n - x_{n-1}) & \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}.$$

In theory, this system can be solved in $O(n^2)$ operations. Unfortunately, the computation of the products in the coefficient matrix can easily overflow or underflow. Now, we derive another way of calculating c_k .

- Since this is a lower triangular system, c_k is the leading coefficient of the polynomial interpolating at $(x_0, y_0), \dots, (x_k, y_k)$. In other words, adding a new pair (x_k, y_k) does not affect the previously computed c_0, c_1, \dots, c_{k-1} .

Let $f[x_j, x_{j+1}, \dots, x_k]$ ($k \geq j$) denote the leading coefficient of the $(k - j)$ -degree polynomial interpolating at $(x_j, y_j), \dots, (x_k, y_k)$. In particular,

$$f[x_k] = y_k, \quad \text{for } k = 0, \dots, n$$

and

$$f[x_0, x_1, \dots, x_k] = c_k.$$

We can prove a two-term recursion:

$$f[x_j, \dots, x_k] = \frac{f[x_{j+1}, \dots, x_k] - f[x_j, \dots, x_{k-1}]}{x_k - x_j}, \quad \text{for } k > j.$$

Define

$p(x)$: the polynomial interpolating at $(x_j, y_j), \dots, (x_k, y_k)$

$q(x)$: the polynomial interpolating at $(x_j, y_j), \dots, (x_{k-1}, y_{k-1})$

$r(x)$: the polynomial interpolating at $(x_{j+1}, y_{j+1}), \dots, (x_k, y_k)$

then

$$p(x) = q(x) + \frac{x - x_j}{x_k - x_j}(r(x) - q(x));$$

since both sides have the same value at $x = x_j, x_{j+1}, \dots, x_{k-1}, x_k$. Comparing the leading coefficients on the both sides, we have

$$f[x_j, \dots, x_k] = \frac{f[x_{j+1}, \dots, x_k] - f[x_j, \dots, x_{k-1}]}{x_k - x_j}.$$

This equality suggests a triangular table for computing c_k

$$\begin{array}{ccccccc} y_0 & = & f[x_0] & & & & \\ y_1 & = & f[x_1] & f[x_0, x_1] & & & \\ y_2 & = & f[x_2] & f[x_1, x_2] & f[x_0, x_1, x_2] & & \\ y_3 & = & f[x_3] & f[x_2, x_3] & f[x_1, x_2, x_3] & f[x_0, x_1, x_2, x_3] & \end{array}$$

and the following algorithm, assuming that y_i are stored in a vector \mathbf{c} .

```

for j=1 to n
  for i=n to j
    c(i) = (c(i) - c(i-1))/(x(i) - x(i-j));
  end
end
end

```

Evaluation

Evaluating a Newton form is similar to evaluating a polynomial:

```

p = c_n;
for i = n - 1 to 0
  p = p(t - x_i) + c_i;
end

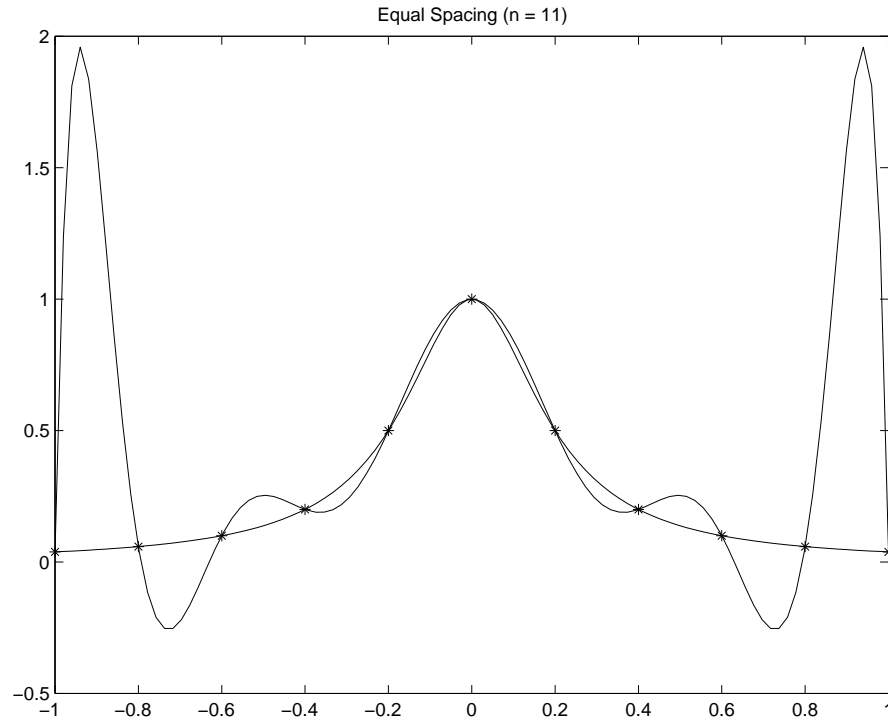
```

The dangers of global polynomial interpolation

An example. Runge's function (continuous derivatives of all order)

$$y(x) = \frac{1}{1 + 25x^2} \quad \text{on} \quad [-1, 1]$$

equally spaces $x_0 = -1, x_1, \dots, x_n = 1$.



A possible solution: Change the positions of the knots x_i , Chebyshev points instead of equal spacing. But, it does not always work.

- It is often best not to use global polynomial interpolation.

3 Piecewise Polynomial Interpolation

Partition

$$\alpha = x_1 < x_2 < \cdots < x_n = \beta,$$

interpolate on each $[x_i, x_{i+1}]$ with a low degree polynomial.

Linear

Linear polynomial on each subinterval:

$$L_i(z) = a_i + b_i(z - x_i), \quad z \in [x_i, x_{i+1}]$$

$$a_i = y_i \quad b_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$$

Algorithm. Piecewise linear interpolation. Given vectors \mathbf{x} and \mathbf{y} with interpolating points, this function returns the piecewise linear interpolation coefficients in the vectors \mathbf{a} and \mathbf{b} .

```
function [a,b] = pwL(x,y)
n = length(x);
a = y(1:n-1);
b = diff(y)./diff(x);
```

Evaluation

Given the piecewise interpolation $L(z)$ represented by the coefficient vectors a, b , how do we evaluate this function at $z \in [\alpha, \beta]$?

First, we locate $[x_i, x_{i+1}]$ such that $z \in [x_i, x_{i+1}]$. Then, we evaluate $L(z)$ using $L_i(z)$.

- Methods: linear search, binary search.
- Observation: If $[x_i, x_{i+1}]$ is associated with the current z , then it is likely that this subinterval will be the one for the next value.
- Idea: Use the previous subinterval as a guess. If not, do binary search.

Algorithm. Given the vector x of breakpoints and a scalar z between x_1 and x_n , this function locates i so that $x_i \leq z \leq x_{i+1}$. The optional g is a guess.

```
function i = Locate(x,z,g)
if nargin==3 % try the guess
    if (x(g)<=z) & (z<=x(g+1))
        i = g;
        return % quick return
    end
end

n = length(x);
if z==x(n)
    i = n-1; % quick return
else % binary search
    left = 1; right = n;
```

```

while right > left+1
    mid = floor((left + right)/2);
    if z < x(mid)
        right = mid;
    else
        left = mid;
    end
end
i = left;
end

```

Algorithm. Given a piecewise linear interpolation coefficients in a and b and its breakpoints in x , this function returns the values of the interpolation evaluated at the points in z .

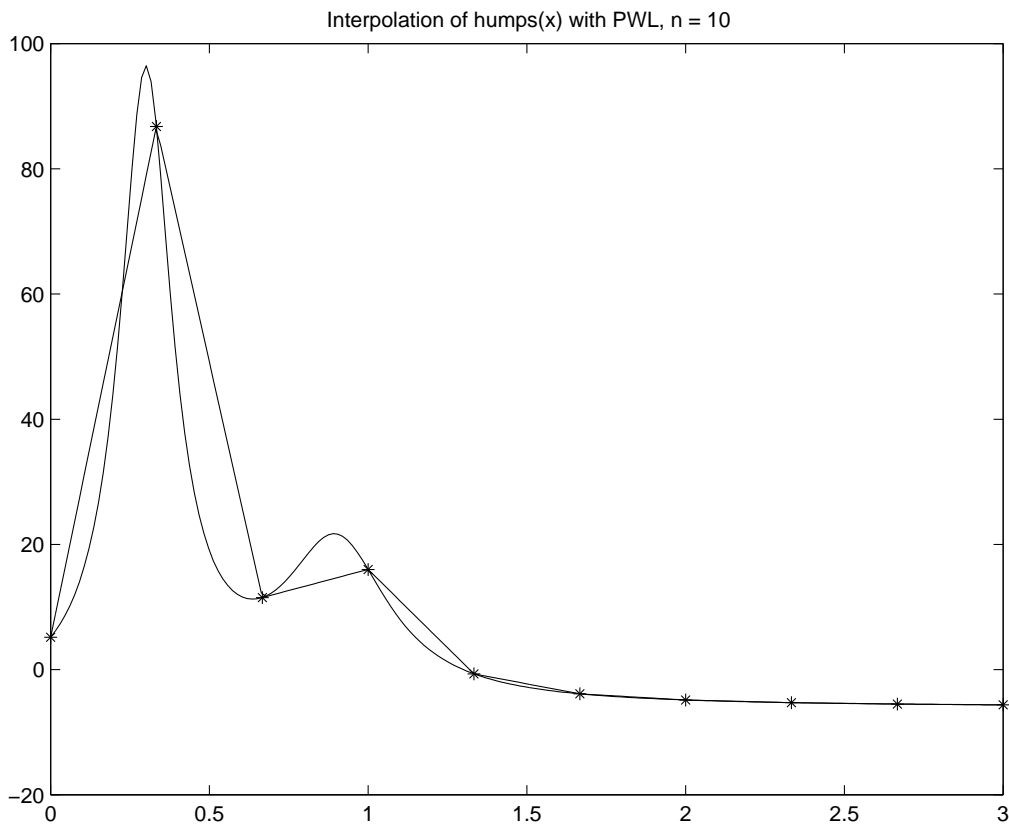
```

function v = pwLEval(a,b,x,z)
    m = length(z);
    v = zeros(m,1);
    g = 1;
    for j=1:m
        i = Locate(x,z(j),g);
        v(j) = a(i) + b(i)*(z(j) - x(i));
        g = i;
    end
end

```

Example.

$$y = \frac{1}{(x - 0.3)^2 + 0.01} + \frac{1}{(x - 0.9)^2 + 0.04} - 6$$



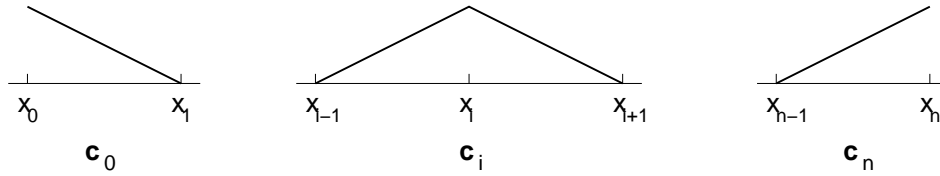
Hat Functions

The hat functions defined by

$$c_i(z) = \begin{cases} (z - x_{i-1})/h_{i-1} & \text{if } z \in [x_{i-1}, x_i], \\ (x_{i+1} - z)/h_i & \text{if } z \in [x_i, x_{i+1}], \\ 0 & \text{elsewhere,} \end{cases}$$

where the hat function $c_0(z)$ omits the first of the above cases; $c_n(z)$, the second, form a basis for piecewise linear polynomial interpolation. A piecewise linear polynomial interpolation $L(z)$ can be expressed as a linear combination of the hat functions

$$L(z) = \sum_{i=0}^n y_i c_i(z).$$



4 Natural Cubic Spline

Given $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$, find $s(x)$:

- in each subinterval $[x_i, x_{i+1}]$, $s(x)$ is cubic. There are $n - 1$ intervals, four coefficients each interval, total of $4(n - 1)$ coefficients.
- $s(x_i) = y_i$, $i = 1, \dots, n$. For each interval $[x_i, x_{i+1}]$, $s(x_i) = y_i$ and $s(x_{i+1}) = y_{i+1}$. There are $2(n - 1)$ constraints.
- $s'(x)$ and $s''(x)$ are continuous at x_2, x_3, \dots, x_{n-1} , total of $2(n - 2)$ constraints.
- $s''(x_1) = s''(x_n) = 0$, two constraints.

Constructing $s(x)$

In the subinterval $[x_i, x_{i+1}]$, let

$$h_i = x_{i+1} - x_i, \quad w = (x - x_i)/h_i, \quad \bar{w} = 1 - w.$$

We have introduced a new variable w to replace x . Notice that $w: 0 \rightarrow 1$, $\bar{w}: 1 \rightarrow 0$, as $x: x_i \rightarrow x_{i+1}$

Construct

$$s(x) = wy_{i+1} + \bar{w}y_i + h_i^2[(w^3 - w)\sigma_{i+1} + (\bar{w}^3 - \bar{w})\sigma_i]$$

where σ_i , total of n , to be determined. The first two terms are constructed so that $s(x)$ interpolates (x_i, y_i) and (x_{i+1}, y_{i+1}) and the last two terms are constructed so that $s''(x)$ interpolates $(x_i, 6\sigma_i)$ and $(x_{i+1}, 6\sigma_{i+1})$. Specifically, using $w' = 1/h_i$ and $\bar{w}' = -1/h_i$, we can verify

1. $s(x_i) = y_i$, $s(x_{i+1}) = y_{i+1}$, independent of σ , which leads to the continuity of $s(x)$.
2. $s''(x) = 6w\sigma_{i+1} + 6\bar{w}\sigma_i$.

Clearly $s''(x_i) = 6\sigma_i$, meaning $s''(x)$ is continuous. It then remains to make $s'(x)$ continuous.

Consider

$$s'(x) = \frac{y_{i+1} - y_i}{h_i} + h_i[(3w^2 - 1)\sigma_{i+1} - (3\bar{w}^2 - 1)\sigma_i]$$

Let $\Delta_i = (y_{i+1} - y_i)/h_i$. On $[x_i, x_{i+1}]$,

$$s'_+(x_i) = \Delta_i + h_i(-\sigma_{i+1} - 2\sigma_i).$$

On $[x_{i-1}, x_i]$,

$$s'_-(x_i) = \Delta_{i-1} + h_{i-1}(2\sigma_i + \sigma_{i-1}).$$

Making $s'(x)$ continuous

$$s'_+(x_i) = s'_-(x_i), \quad i = 2, 3, \dots, n-1$$

we get $n-2$ equations

$$h_{i-1}\sigma_{i-1} + 2(h_{i-1} + h_i)\sigma_i + h_i\sigma_{i+1} = \Delta_i - \Delta_{i-1}$$

for $i = 2, 3, \dots, n-1$. Set $\sigma_1 = \sigma_n = 0$.

Matrix form

Diagonal: $[2(h_1 + h_2), \dots, 2(h_{n-2} + h_{n-1})]$

supper/subdiagonal: $[h_2, \dots, h_{n-2}]$

unknowns: $[\sigma_2, \dots, \sigma_{n-1}]^T$

right-hand side: $[\Delta_2 - \Delta_1, \dots, \Delta_{n-1} - \Delta_{n-2}]^T$

- The matrix is symmetric
- The matrix is tridiagonal
- The matrix is diagonally dominant when $x_1 < x_2 < \dots < x_n$
- Can apply Gaussian elimination without pivoting

Evaluating $s(x)$

If $s(x)$ is evaluated many times, arrange $s(x)$ so that

$$s(x) = y_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3,$$

for $x_i \leq x \leq x_{i+1}$
 and store b_i, c_i, d_i (instead of σ_i)

$$b_i = \frac{y_{i+1} - y_i}{h_i} - h_i(\sigma_{i+1} + 2\sigma_i), \quad c_i = 3\sigma_i, \quad d_i = \frac{\sigma_{i+1} - \sigma_i}{h_i},$$

for $i = 1, 2, \dots, n - 1$

Algorithm. Given a vector \mathbf{x} with breakpoints and vector \mathbf{y} with function values, this algorithm computes the coefficients $\mathbf{b}, \mathbf{c}, \mathbf{d}$ of natural spline interpolation.

1. Compute h_i and Δ_i ;
2. Form the tridiagonal matrix (two arrays) and the right hand side;
3. Solve for σ_i ;
4. Compute the coefficients \mathbf{b}, \mathbf{c} , and \mathbf{d} .

Summary

- Global polynomial interpolation: General ideas and methods
- Piecewise polynomial interpolation: Construction of piecewise polynomial (linear and cubic), evaluation of a piecewise function.
- Basis functions: Lagrange polynomials and hat functions.

Software packages

IMSL csint, csdec, csher, csval

MATLAB polyfit, spline, ppval

NAG e01aef, e01baf, e01bef, e02bbf, e01bff

References

- George E. Forsythe, Michael A. Malcolm, and Cleve B. Moler, *Computer Methods for Mathematical Computations*, Prentice-Hall, Inc., 1977.
- Charles F. Van Loan, *Introduction to Scientific Computing*, Prentice-Hall, 1997.