Spline Interpolation

Reading: Cheney and Kincaid, Sections 6.1 & 6.2

Motivation: Runge's phenomenon suggests using high degree polynomials to do interpolation may be risky. To avoid the problem, we use piecewise low degree polynomials to do interpolation.

Spline functions

Def. A function S is called a spline of degree k if

- The domain of S is an interval $[a, b]$.
- $S, S', \ldots S^{(k-1)}$ are continuous on $[a, b]$.
- There are points t_i (the knots of S) such that $a = t_0 < t_1 < \cdots < t_n = b$ and such that S is a polynomial of degree at most k on each $[t_i, t_{i+1}]$.

For $k = 1, 2, 3$, the splines are called linear splines, quadratic splines and cubic splines, respectively. Here we are mainly interested in linear splines and cubic splines.

ordered

Interpolation by Linear Splines

Problem: Given $n + 1$ points (t_0, y_0), (t_1, y_1),…,(t_n, y_n), where without loss of generality we assume $t_0 < t_1 < \cdots < t_n$, we seek a linear spline $S(x)$ such that $S(t_i) = y_i$ for $0 \le i \le n$ and t_i are the knots of $S(x)$.

Solution: Obviously we can write

$$S(x) = \begin{cases} S_0(x), & t_0 \le x \le t_1 \\ S_1(x), & t_1 \le x \le t_2 \\ \vdots & \\ S_{n-1}(x), & t_{n-1} \le x \le t_n \end{cases}$$

where

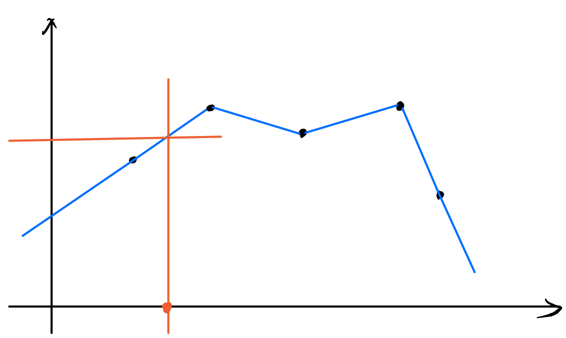
$$S_i(x) = y_i + m_i(x - t_i), \quad m_i = \frac{y_{i+1} - y_i}{t_{i+1} - t_i}, \quad t_i \le x \le t_{i+1}.$$

So $S(x)$ is a piecewise linear polynomial.

Algorithm for evaluating $S(x)$ (given x, t_i, y_i and m_i, $i = 0, 1, \ldots, n$):

for $i = 0 : n - 1$
    if $x - t_{i+1} \le 0$,
        exit loop
    end
end
$S \leftarrow y_i + m_i(x - t_i)$

find interval where x lies.



Remarks:

- When $x < t_0$, the algoritm gives $S = y_0 + m_0(x - t_0)$; when $x > t_n$, it gives $S = y_{n-1} + m_{n-1}(x - t_{n-1})$.
- A *binary search* can be used to find the desired interval which consists of x. This is more efficient on average.

Interpolation by Cubic Splines

not testable

For a linear spline, generally S' is not continuous, so its graph is lack of smoothness. For a quadratic spline, generally S'' is not continuous, so the curvature of its graph changes abruptly at each knot. So in practice, the most frequently used splines are cubic splines.

Problem: Given $n+1$ points (t_0, y_0), (t_1, y_1),...,(t_n, y_n), where without loss of generality we assume $t_0 < t_1 < \cdots < t_n$, we seek a cubic spline $S(x)$ such that $S(t_i) = y_i$ for $0 \le i \le n$ and t_i are the knots of $S(x)$.

Obviously we can write

$$S(x) = \begin{cases} S_0(x), & t_0 \le x \le t_1 \\ S_1(x), & t_1 \le x \le t_2 \\ \vdots & \\ S_{n-1}(x), & t_{n-1} \le x \le t_n \end{cases}$$

n+1 points
n intervals
n cubic polynomials.

were S_i is a cubic polynomial on $[t_i, t_{i+1}]$.

Number of unknowns:
Each S_i has 4 unknowns. So there is a total of $4n$ unknowns.

? quadratic

Number of conditions:
$S(t_i) = y_i$ for $i = 0, 1, \ldots, n$ result in $n+1$ conditions. $S_{i-1}^{(k)}(t_i) = S_i^{(k)}(t_i)$ for $k = 0, 1, 2$ and $i = 1, \ldots n-1$ lead to $3(n-1)$ conditions. So there is a total of $4n-2$ conditions.
In order to get a unique solution, we need 2 more extra conditions. Here we impose the following two conditions:

$$\boxed{S''(t_0) = S''(t_n) = 0.}$$

extra conditions

The resulting spline function is called a **natural cubic spline**.

Constructing a Natural Cubic Spline

Let $z_i = S''(t_i)$ $(0 \le i \le n)$. On $[t_i, t_{i+1}]$, $S_i''(x)$ is a linear polynomial and $S_i''(t_i) = z_i$ and $S_i''(t_{i+1}) = z_{i+1}$. So we can write

lagrange form.

$$S_i''(x) = \frac{x - t_{i+1}}{t_i - t_{i+1}} z_i + \frac{x - t_i}{t_{i+1} - t_i} z_{i+1}. = \frac{z_{i+1} - z_i}{t_{i+1} - t_i}(x - t_i) + z_i$$

$$= z_i + m_i(x - t_i$$

Integrating $S_i''(x)$ twice, we obtain

$$S_i(x) = (t_{i+1} - x)^3 \frac{z_i}{6h_i} + (x - t_i)^3 \frac{z_{i+1}}{6h_i} + cx + d, \tag{1}$$

where $h_i \equiv t_{i+1} - t_i$, and c and d are constants of integration. We impose the conditions $S_i(t_i) = y_i$ and $S_i(t_{i+1}) = y_{i+1}$. Then we have

$$\begin{aligned} h_i^3 \frac{z_i}{6h_i} + ct_i + d &= y_i, \\ h_i^3 \frac{z_{i+1}}{6h_i} + ct_{i+1} + d &= y_{i+1}. \end{aligned}$$

Solving the above linear system for c and d gives

$$\begin{aligned} c &= (y_{i+1} - y_i)/h_i - h_i(z_{i+1} - z_i)/6 \\ d &= (y_i t_{i+1} - y_{i+1} t_i)/h_i + h_i(t_i z_{i+1} - t_{i+1} z_i)/6. \end{aligned}$$

Now we have to determine the z_i and z_{i+1} in $S_i(x)$. In order to do this, we impose conditions $S'_{i-1}(t_i) = S'_i(t_i)$. From eqn (1) we obtain

$$\begin{aligned} S'_i(x) &= -(t_{i+1} - x)^2 z_i/(2h_i) + (x - t_i)^2 z_{i+1}/(2h_i) \\ &\quad +(y_{i+1} - y_i)/h_i - (z_{i+1} - z_i)h_i/6. \end{aligned}$$

Thus, with $b_i \equiv (y_{i+1} - y_i)/h_i$,

$$S'_i(t_i) = -\frac{1}{3}h_i z_i - \frac{1}{6}h_i z_{i+1} + b_i.$$

Analogously, we have

$$\begin{aligned} S'_{i-1}(x) &= -(t_i - x)^2 z_{i-1}/(2h_{i-1}) + (x - t_{i-1})^2 z_i/(2h_{i-1}) \\ &\quad +(y_i - y_{i-1})/h_{i-1} - (z_i - z_{i-1})h_{i-1}/6. \end{aligned}$$

Then

$$S'_{i-1}(t_i) = \frac{1}{6}h_{i-1} z_{i-1} + \frac{1}{3}h_{i-1} z_i + b_{i-1}.$$

The equalites $S'_{i-1}(t_i) = S'_i(t_i)$ for $i = 1, 2, \ldots, n-1$ lead to

$$h_{i-1} z_{i-1} + 2(h_{i-1} + h_i) z_i + h_i z_{i+1} = 6(b_i - b_{i-1}), \quad i = 1, 2, \ldots, n-1.$$

Notice $z_0 = z_n = 0$. So we have the following tridiagonal system

$$\begin{bmatrix} 2(h_0+h_1) & h_1 & & & \\ h_1 & 2(h_1+h_2) & h_2 & & \\ & \bullet & \bullet & \bullet & \\ & & h_{n-3} & 2(h_{n-3}+h_{n-2}) & h_{n-2} \\ & & & h_{n-2} & 2(h_{n-2}+h_{n-1}) \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \bullet \\ z_{n-2} \\ z_{n-1} \end{bmatrix} = \begin{bmatrix} 6(b_1 - b_0) \\ 6(b_2 - b_1) \\ \bullet \\ 6(b_{n-2} - b_{n-3}) \\ 6(b_{n-1} - b_{n-2}) \end{bmatrix}$$

This can be solved by GENP.

Algorithm for finding z_i, $i = 0, \ldots, n$ (given t_i, y_i, $i = 0, \ldots, n$):

for $i = 0 : n-1$
    $h_i \leftarrow t_{i+1} - t_i$
    $b_i \leftarrow (y_{i+1} - y_i)/h_i$
end
% Forward elimination
$u_1 \leftarrow 2(h_0 + h_1)$
$v_1 \leftarrow 6(b_1 - b_0)$
for $i = 2 : n-1$
    $mult \leftarrow h_{i-1}/u_{i-1}$
    $u_i \leftarrow 2(h_{i-1} + h_i) - mult * h_{i-1}$

$v_i \leftarrow 6(b_i - b_{i-1}) - mult * v_{i-1}$
end
% Back substitution
$z_n \leftarrow 0$
for $i = n-1 : -1 : 1$
$z_i \leftarrow (v_i - h_i z_{i+1})/u_i$
end
$z_0 \leftarrow 0$

Note: The trdiagonal matrix is strictly diagonally dominant by column, so GENP will give the same result as GEPP (there are no row interchanges in GEPP).

Evaluation of $S(x)$

$$\begin{aligned} S_i(x) &= (t_{i+1}-x)^3 \frac{z_i}{6h_i} + (x-t_i)^3 \frac{z_{i+1}}{6h_i} + \frac{y_{i+1}-y_i}{h_i} - \frac{h_i}{6}(z_{i+1}-z_i) \\ &\quad + \frac{y_i t_{i+1} - y_{i+1} t_i}{h_i} + \frac{h_i}{6}(t_i z_{i+1} - t_{i+1} z_i). \end{aligned}$$

This is not the best computational form. As we want to utilize nested multiplication, we write

$$S_i(x) = A_i + B_i(x-t_i) + C_i(x-t_i)^2 + D_i(x-t_i)^3.$$

Notice $A_i = S_i(t_i)$, $B_i = S_i'(t_i)$, $C_i = \frac{1}{2}S_i''(t_i)$, $D_i = \frac{1}{6}S_i'''(t_i)$. Then we can obtain

$$\begin{aligned} A_i &= y_i \\ B_i &= -h_i z_{i+1}/6 - h_i z_i/3 + (y_{i+1}-y_i)/h_i \\ C_i &= z_i/2 \\ D_i &= (z_{i+1}-z_i)/(6h_i) \end{aligned}$$

$$S_i(x) = A_i + (x-t_i)(B_i + (x-t_i)(c_i + (x-t_i)D_i)).$$

Algorithm for evaluating $S(x)$ (given x, t_i, y_i and z_i for $i = 0, 1, \ldots, n$):

for $i = 0 : n-1$
if $x - t_{i+1} \le 0$
exit loop
end
end
$h \leftarrow t_{i+1} - t_i$
$B \leftarrow -hz_{i+1}/6 - hz_i/3 + (y_{i+1}-y_i)/h$
$D \leftarrow (z_{i+1}-z_i)/(6h)$
$S \leftarrow y_i + (x-t_i)(B + (x-t_i)(z_i/2 + (x-t_i)D))$

Note: As the note we made for the evaluation of the linear spline, we can use a binary search to find the desired interval consisting of x and it is more efficient on average.

MATLAB built-in function for spline: `spline`