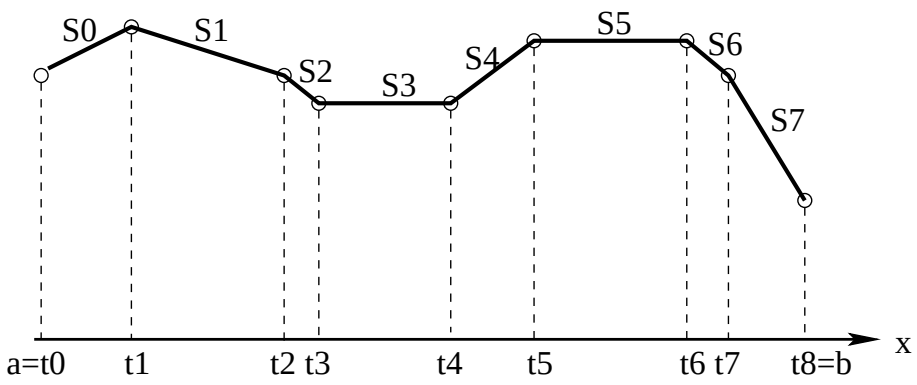
LECTURE 6: Approximation by spline functions

October 10, 2011

1 First-degree and second-degree splines

1.1 First-degree spline

A *spline function* is a function that consists of polynomial pieces joined together with certain smoothness conditions. For example, the *polygonal* function is a spline of degree 1 which consists of linear polynomials joined together to achieve continuity. The points $t_0, t_1, \ldots, t_n$ are called *knots*.



In explicit form, the function must be defined piece by piece:

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

where each piece of $S(x)$ is a linear polynomial:

$$S_i(x) = a_i x + b_i$$

A function such as $S(x)$ is called *piecewise linear*.

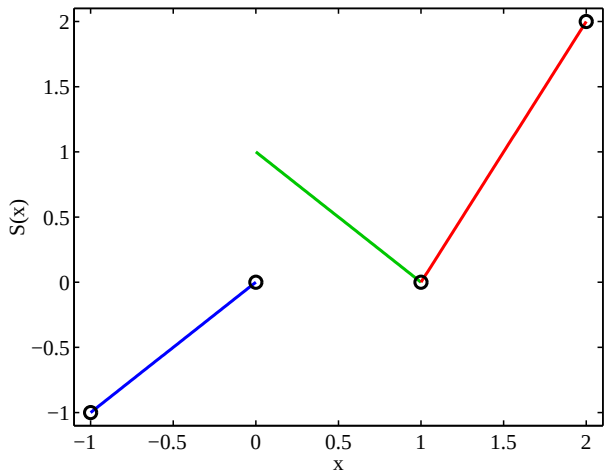
Definition.
A function S is called a **spline of degree 1** if:

1. The domain of S is an interval $[a,b]$.
2. S is continuous on $[a,b]$.
3. There is a partitioning of the interval $a = t_0 < t_1 < \ldots < t_n = b$ such that S is a linear polynomial on each subinterval $[t_i, t_{i+1}]$.

Example. The following function

$$S(x) = \begin{cases} x & x \in [-1,0] \\ 1-x & x \in (0,1) \\ 2x-2 & x \in [1,2] \end{cases}$$

is **not** a spline because it is discontinuous at $x = 0$.



Continuity of a function f at point s can be defined by the condition

$$\lim_{x \to s^+} f(x) = \lim_{x \to s^-} f(x) = f(s)$$

In other words, this means that the values of f must converge to the same limiting value $f(s)$ (which is the value of the function at s) from both above and below the point s.

Spline functions of degree 1 can be used for **interpolation**. Suppose that we have the following table of values:

x	t_0	t_1	$\ldots$	t_n
y	y_0	y_1	$\cdots$	y_n

This can be represented by $n+1$ points in the xy-plane, and we can draw a polygonal line (a spline of degree 1) through the points.

The equation for the line segment on the interval $[t_i, t_{i+1}]$ is given by

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

Here m_i is the slope of the line.

The following pseudocode is a function which utilizes $n+1$ table values to evaluate the function $S(x)$:

```
real function Spline1(n, (t_i), (y_i), x)
real array (t_i)_{0:n}, (y_i)_{0:n}
integer i, n
real x
for i=n-1 to 0 step -1 do
   if x - t_i ≤ 0 then exit loop
end for
Spline1 ← y_i + (x - t_i)[(y_{i+1} - y_i)/(t_{i+1} - t_i)]
end function Spline1
```

1.2 Modulus of continuity

Suppose that f is defined on an interval $[a,b]$. The *modulus of continuity* of f is

$$\omega(f;h) = \sup\{|f(u)-f(v)| : a \le u \le v \le b, |u-v| \le h\}$$

Here sup is the *supremum* which is the least upper bound of a given set of real numbers.

The modulus of continuity, $\omega(f;h)$, measures *how much* f *can change* over a small interval h. It can be used to assess the *goodness of fit* when we interpolate a function with a first-degree spline.

First-degree polynomial accuracy theorem.
If p is the first-degree polynomial that interpolates a function f at the endpoints of an interval $[a,b]$, then with $h = b-a$ we have

$$|f(x)-p(x)| \le \omega(f;h) \quad (a \le x \le b)$$

Proof. The linear function p is given explicitly by

$$p(x) = \left(\frac{x-a}{b-a}\right) f(b) + \left(\frac{b-x}{b-a}\right) f(a)$$

Hence,

$$f(x)-p(x) = \left(\frac{x-a}{b-a}\right)[f(x)-f(b)] + \left(\frac{b-x}{b-a}\right)[f(x)-f(a)]$$

Then we have

$$\begin{aligned}
|f(x)-p(x)| &\le \left(\frac{x-a}{b-a}\right)|f(x)-f(b)| + \left(\frac{b-x}{b-a}\right)|f(x)-f(a)| \\
&\le \left(\frac{x-a}{b-a}\right)\omega(f;h) + \left(\frac{b-x}{b-a}\right)\omega(f;h) \\
&= \left[\left(\frac{x-a}{b-a}\right) + \left(\frac{b-x}{b-a}\right)\right]\omega(f;h) \\
&= \omega(f;h)
\end{aligned}$$

First-degree spline accuracy theorem.
Let p be a first-degree spline having knots $a = x_0 < x_i < \ldots < x_n = b$. If p interpolates a function f at these knots, then with $h = \max(x_i - x_{i-1})$ we have

$$|f(x) - p(x)| \leq \omega(f;h) \quad (a \leq x \leq b)$$

This tells us that if more knots are inserted in such a way that the maximum spacing h goes to zero, then the corresponding first-degree spline will converge uniformly to f.

Note that this type of result does not exist in the polynomial interpolation theory! There increasing the number of nodes may even lead to larger errors!

1.3 Second-degree splines

We are now considering piecewise quadratic functions, denoted by Q.

Definition.
A function Q is called a **spline of degree 2** if:

1. The domain of Q is an interval $[a,b]$.
2. Q and Q' are continuous on $[a,b]$.
3. There are points t_i (called knots) such that $a = t_0 < t_1 < \ldots < t_n = b$ and Q is a polynomial of degree at most 2 on each subinterval $[t_i, t_{i+1}]$.

In brief, a **quadratic spline** is a continuously differentiable piecewise quadratic function, where quadratic includes all linear combinations of the basic functions $1, x, x^2$.

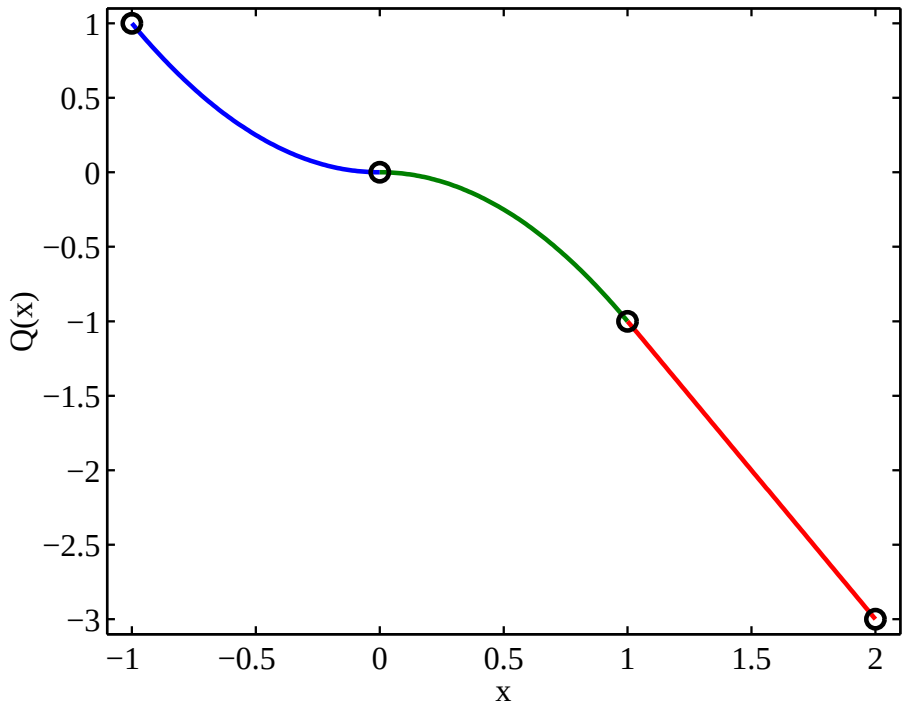
Example. Consider the following function

$$Q(x) = \begin{cases} x^2 & x \leq 0 \\ -x^2 & 0 \leq x \leq 1 \\ 1-2x & x \geq 1 \end{cases}$$

The function is obviously piecewise quadratic. We can determine whether Q and Q' are continuous by considering all the knot points separately:

$$\lim_{x\to 0^-} Q(x) = \lim_{x\to 0^-} x^2 = 0 \qquad \lim_{x\to 0^+} Q(x) = \lim_{x\to 0^+} -x^2 = 0$$
$$\lim_{x\to 1^-} Q(x) = \lim_{x\to 1^-} -x^2 = -1 \qquad \lim_{x\to 1^+} Q(x) = \lim_{x\to 1^+} (1-2x) = -1$$
$$\lim_{x\to 0^-} Q'(x) = \lim_{x\to 0^-} 2x = 0 \qquad \lim_{x\to 0^+} Q'(x) = \lim_{x\to 0^+} -2x = 0$$
$$\lim_{x\to 1^-} Q'(x) = \lim_{x\to 1^-} -2x = -2 \qquad \lim_{x\to 1^+} Q'(x) = \lim_{x\to 1^+} (-2) = -2$$

Thus, $Q(x)$ is a quadratic spline.



1.4 Interpolating quadratic spline

Quadratic splines are not used in applications as often as natural cubic splines, but understanding the simpler second-degree spline theory will help in grasping the more common third-degree splines.

Consider an interpolation problem with the following table of values:

x	t_0	t_1	$\dots$	t_n
y	y_0	y_1	$\dots$	y_n

Think of the *nodes* of the interpolation problem as being also the *knots* for the spline function to be constructed. A quadratic spline consists of n separate quadratic functions, $Q_i(x) = a_i x^2 + b_i x + c_i$. Thus we have $3n$ coefficients to determine.

The following **conditions** are imposed in order to determine the unknown coefficients:

(i) On each subinterval $[t_i, t_{i+1}]$, the quadratic spline function Q_i must satisfy the interpolation conditions: $Q_i(t_i) = y_i$ and $Q_i(t_{i+1}) = y_{i+1}$. This imposes $2n$ conditions.

(ii) The continuity of Q does not impose any further conditions. But the continuity of Q' at each of the interior points gives $n-1$ conditions.

(iii) We are now one condition short, since we have a total of $3n-1$ conditions from (i) and (ii). The last condition can be applied in different ways; e.g. $Q'(t_0) = 0$ or $Q_0'' = 0$.

Derivation of the quadratic interpolating spline

We seek for a piecewise quadratic function

$$Q(x) = \begin{cases} Q_0(x) & x \in [t_0, t_1] \\ Q_1(x) & x \in [t_1, t_2] \\ \vdots & \\ Q_{n-1}(x) & x \in [t_{n-1}, t_n] \end{cases}$$

which is continuously differentiable on the entire interval $[t_0, t_n]$ and which interpolates the table:

$$Q(t_i) = y_i \qquad 0 \le i \le n$$

Denote $z_i = Q'(t_i)$. We can write the following formula for Q_i:

$$Q_i(x) = \frac{z_{i+1} - z_i}{2(t_{i+1} - t_i)}(x - t_i)^2 + z_i(x - t_i) + y_i$$

You can verify that: $Q_i(t_i) = y_i$, $Q_i'(t_i) = z_i$ and $Q_i'(t_{i+1}) = z_{i+1}$.

In order for Q to be continuous and to interpolate the data table, it is necessary and sufficient that $Q_i(t_{i+1}) = y_{i+1}$ for $i = 0, 1, \ldots, n-1$. This gives us the following:

$$z_{i+1} = -z_i + 2\left(\frac{y_{i+1} - y_i}{t_{i+1} - t_i}\right) \quad (0 \leq i \leq n-1)$$

This equation can be used to obtain the vector $[z_0, z_1, \ldots, z_n]^T$, starting with an arbitrary value for $z_0 = Q'(t_0)$.

Algorithm.

(i.) Select z_0 arbitrarily and compute z_1, z_2, ..., z_n using the recursion formula above.

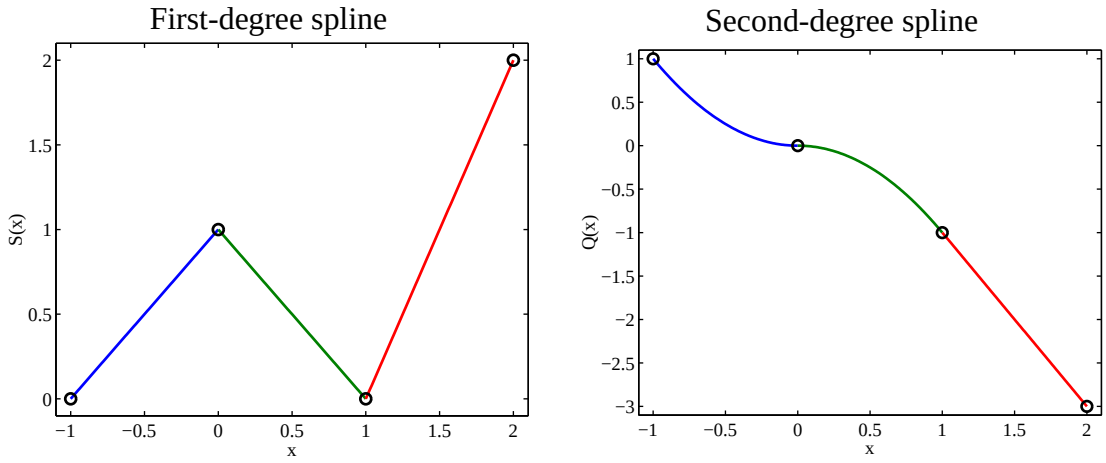
(ii.) The quadratic spline interpolating function Q can now be constructed using the table values for y_i and t_i together with the obtained values for z_i.

2 Natural cubic splines

2.1 Splines of higher degree

First- and second-degree splines are not so useful for actual applications, because their low-order derivatives are discontinuous.

- For first-degree splines, the slope of the spline may change abruptly at the knots.
- For second-degree splines, the discontinuity is in the second derivative which means that the curvature of the quadratic spline changes abruptly at each node.



Higher-degree splines are useful whenever more smoothness is needed in the approximating function.

Definition.
A function S is called a **spline of degree k** if:

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

2.2 Natural cubic spline

Splines of degree 3 are called **cubic splines**. In this case, the spline function has two continuous derivatives which makes the graph of the function appear smooth.

We next turn to interpolating a table of given values using a cubic spline whose knots coincide with the x values in the table.

x	t_0	t_1	$\ldots$	t_n
y	y_0	y_1	$\ldots$	y_n

The t_i's are the knots and are assumed to be arranged in ascending order. The function S, which we are constructing, consists of n cubic polynomial pieces:

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

Here S_i denotes the cubic polynomial that will be used on the subinterval $[t_i, t_{i+1}]$.

The *interpolation conditions* are

$$S_i(t_i) = y_i \quad (0 \le i \le n)$$

The *continuity conditions* are imposed only at the interior knots $t_1, t_2, \ldots, t_{n-1}$:

$$\lim_{x \to t_i^-} S^{(k)}(t_i) = \lim_{x \to t_i^+} S^{(k)}(t_i) \quad (k = 0, 1, 2)$$

Two *extra conditions* are needed in order to use all the degrees of freedom available. We can choose:

$$S''(t_0) = S''(t_n) = 0$$

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

Example.

Determine the parameters a, b, c, d, e, f, g and h so that $S(x)$ is a natural cubic spline, where

$$S(x) = \begin{cases} ax^3 + bx^2 + cx + d & x \in [-1, 0] \\ ex^3 + fx^2 + gx + h & x \in [0, 1] \end{cases}$$

with interpolation conditions $S(-1) = 1$, $S(0) = 2$ and $S(1) = -1$.

Solution.
Let the two cubic polynomials be $S_0(x)$ and $S_1(x)$.
From the interpolation conditions we have

$$S_0(0) = d = 2$$

$$S_1(0) = h = 2$$

$$S_0(-1) = -a + b - c = -1$$

$$S_1(1) = e + f + g = -3$$

The first derivative of $S(x)$ is

$$S'(x) = \begin{cases} 3ax^2 + 2bx + c \\ 3ex^2 + 2fx + g \end{cases}$$

From the continuity condition of S' we get

$$S'(0) = c = g$$

The second derivative is given by

$$S''(x) = \begin{cases} 6ax + 2b \\ 6ex + 2f \end{cases}$$

From the continuity of S'' we get

$$S''(0) = b = f$$

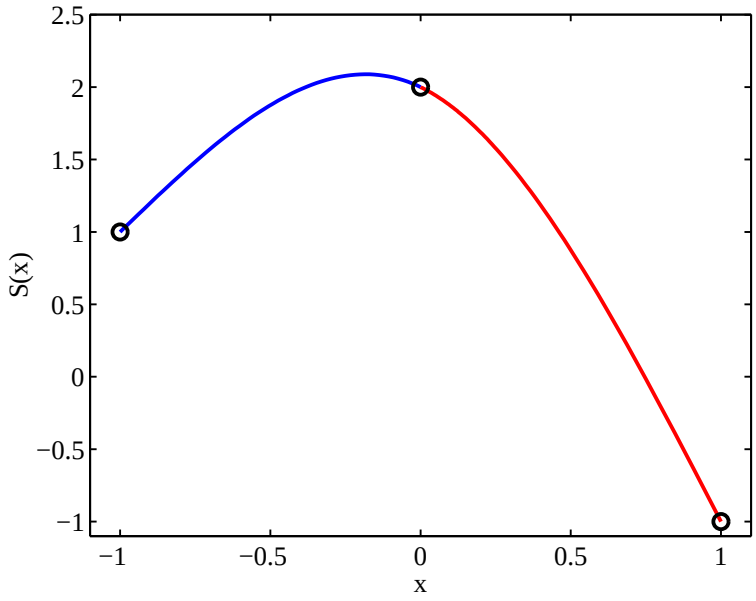
Two extra conditions are

$$S''(-1) = -6a + 2b = 0$$

and

$$S''(1) = -6e + 2f = 0$$

From all these equations, we obtain
$a = -1$, $b = -3$, $c = -1$, $d = 2$, $e = 1$, $f = -3$ and $h = 2$.



2.3 Algorithm for natural cubic spline

We will now develop a systematic procedure for determining the formula for a natural cubic spline, when the table of interpolation values is given.

From the continuity of S'', we can unambiguously define the following numbers

$$z_i \equiv S''(t_i) \quad (0 \leq i \leq n)$$

From the two extra conditions, we have $z_0 = z_n = 0$. The other values z_i are not yet known. We know that, on each subinterval $[t_i, t_{i+1}]$, S'' is a linear polynomial that takes the values z_i and z_{i+1} at the endpoints. Thus

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

with $h_i = t_{i+1} - t_i$ for $0 \leq i \leq n-1$.

Integrating this twice gives us the function S_i:

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

where c and d are constants of integration.

By adjusting the integration constants, we obtain a more convenient form for S_i:

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

where C_i and D_i are constants.

We can now impose the interpolation conditions $S_i(t_i) = y_i$ and $S_i(t_{i+1}) = y_{i+1}$ to determine the appropriate values of C_i and D_i.

The result is

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

When the values $z_0, z_1, \ldots, z_n$ have been determined, the spline function $S(x)$ is obtained piece by piece from this equation.

We now determine the z_i's. We use the remaining condition - namely the continuity of S'. At the interior knots t_i for $1 \leq i \leq n-1$, we must have $S'_{i-1}(t_i) = S'_i(t_i)$. We have

$$S'_i(x) = \frac{z_{i+1}}{2h_i}(x-t_i)^2 - \frac{z_i}{2h_i}(t_{i+1}-x)^2 + \frac{y_{i+1}}{h_i} - \frac{h_i}{6}z_{i+1} - \frac{y_i}{h_i} + \frac{h_i}{6}z_i$$

This gives

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

where

$$b_i = \frac{1}{h_i}(y_{i+1} - y_i)$$

and

$$h_i = t_{i+1} - t_i.$$

Analogously, we have

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

When these are set as equals, we get after rearrangement

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

for $1 \leq i \leq n-1$.

By letting

$$u_i = 2(h_{i-1}+h_i)$$
$$v_i = 6(b_i - b_{i-1})$$

we obtain a *tridiagonal system of equations*:

$$\begin{cases} z_0 & = 0 \\ h_{i-1}z_{i-1} + u_i z_i + h_i z_{i+1} & = v_i \quad (1 \leq i \leq n-1) \\ z_n & = 0 \end{cases}$$

which is to be solved for the z_i's.

The first and last equations come from the natural cubic spline conditions $S''(t_0) = S''(t_n) = 0$.

The tridiagonal system can be written in matrix form as follows:

$$\begin{bmatrix} 1 & 0 & & & & \\ h_0 & u_1 & h_1 & & & \\ & h_1 & u_2 & h_2 & & \\ & & \ddots & \ddots & \ddots & \\ & & & & 0 & 1 \end{bmatrix} \begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ \vdots \\ z_{n-1} \\ z_n \end{bmatrix} = \begin{bmatrix} 0 \\ v_1 \\ v_2 \\ \vdots \\ v_{n-1} \\ 0 \end{bmatrix}$$

On eliminating the first and last equations, we obtain

$$\begin{bmatrix} u_1 & h_1 & & & \\ h_1 & u_2 & h_2 & & \\ & \ddots & \ddots & \ddots & \\ & & h_{n-3} & u_{n-2} & h_{n-2} \\ & & & h_{n-2} & u_{n-1} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_{n-2} \\ z_{n-1} \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_{n-2} \\ v_{n-1} \end{bmatrix}$$

which is a symmetric tridiagonal system of order $n-1$.

Using the ideas presented in Chapter 5 of these lecture notes, we can develop an algorithm for solving such a tridiagonal system.

The *forward elimination* phase in Gaussian elimination without pivoting would modify the u_i's and v_i's as follows:

$$\begin{cases} u_i \leftarrow u_i - \dfrac{h_{i-1}^2}{u_{i-1}} \\ v_i \leftarrow v_i - \dfrac{h_{i-1}v_{i-1}}{u_{i-1}} \end{cases} \quad (i = 2,3,\ldots,n-1)$$

The *back substitution yields*

$$\begin{cases} z_{n-i} \leftarrow \dfrac{v_{n-1}}{u_{n-1}} \\ z_i \leftarrow \dfrac{v_i - h_i z_{i+1}}{u_i} \end{cases} \quad (i = n-2, n-3, \ldots, 1)$$

Algorithm

Given the interpolation points (t_i, y_i) for $i = 0, 1, \ldots, n$:

1. Compute for $i = 0, 1, \ldots, n-1$

$$\begin{cases} h_i = t_{i+1} - t_i \\ b_i = \frac{1}{h_i}(y_{i+1} - y_i) \end{cases}$$

2. Set

$$\begin{cases} u_1 = 2(h_0 + h_1) \\ v_1 = 6(b_1 - b_0) \end{cases}$$

and compute inductively for $i = 1, 2, \ldots, n-1$

$$\begin{cases} u_i = 2(h_i + h_{i-1}) - \frac{h_{i-1}^2}{u_{i-1}} \\ v_i = 6(b_i - b_{i-1}) - \frac{h_{i-1} v_{i-1}}{u_{i-1}} \end{cases}$$

3. Set

$$\begin{cases} z_n = 0 \\ z_0 = 0 \end{cases}$$

and compute inductively for $i = n-1, n-2, \ldots, 1$

$$z_i = \frac{v_i - h_i z_{i+1}}{u_i}$$

This algorithm could potentially fail because of divisions by zero in steps 2 and 3. Therefore, let us prove that $u_i \neq 0$ for all i.

It is clear that $u_1 > h_1 > 0$. If $u_{i-1} > h_{i-1}$, then $u_i > h_i$ because

$$u_i = 2(h_i + h_{i-1}) - \frac{h_{i-1}^2}{u_{i-1}} > 2(h_i + h_{i-1}) - h_{i-1} > h_i$$

Then by induction, $u_i > 0$ for $i = 1, 2, \ldots, n-1$.

Evaluation of $\mathbf{S_i(x)}$

The following form for the cubic polynomial S_i

$$S_i(x) = \frac{z_{i+1}}{6h_i}(x-t_i)^3 + \frac{z_i}{6h_i}(t_{i+1}-x)^3 + \left(\frac{y_{i+1}}{h_i} - \frac{h_i}{6}z_{i+1}\right)(x-t_i) + \left(\frac{y_i}{h_i} - \frac{h_i}{6}z_i\right)(t_{i+1}-x)$$

is not computationally very efficient.

A preferable form would be

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

Notice that the equation above is the Taylor expansion of S_i about the point t_i. Hence,

$$A_i = S_i(t_i) \qquad B_i = S_i'(t_i) \qquad C_i = \frac{1}{2}S_i''(t_i) \qquad D_i = \frac{1}{6}S_i'''(t_i)$$

Therefore, $A_i = y_i$ and $C_i = z_i/2$.

The coefficient of x^3 is D_i in the second form, whereas in the previously given form it is $(z_{i+1}-z_i)/6h_i$. Thus

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

Finally, the value of $S_i'(t_i)$ is given by

$$B_i = -\frac{h_i}{6}z_{i+1} - \frac{h_i}{3}z_i + \frac{1}{h_i}(y_{i+1}-y_i)$$

Thus the nested form of $S_i(x)$ is

$$S_i(x) = y_i + (x-t_i)\left(B_i + (x-t_i)\left(\frac{z_i}{2} + \frac{1}{6h_i}(x-t_i)(z_{i+1}-z_i)\right)\right)$$

Implementation

The following procedure solves the $(n+1)\times(n+1)$ tridiagonal system. The result is stored in the array (z_i).

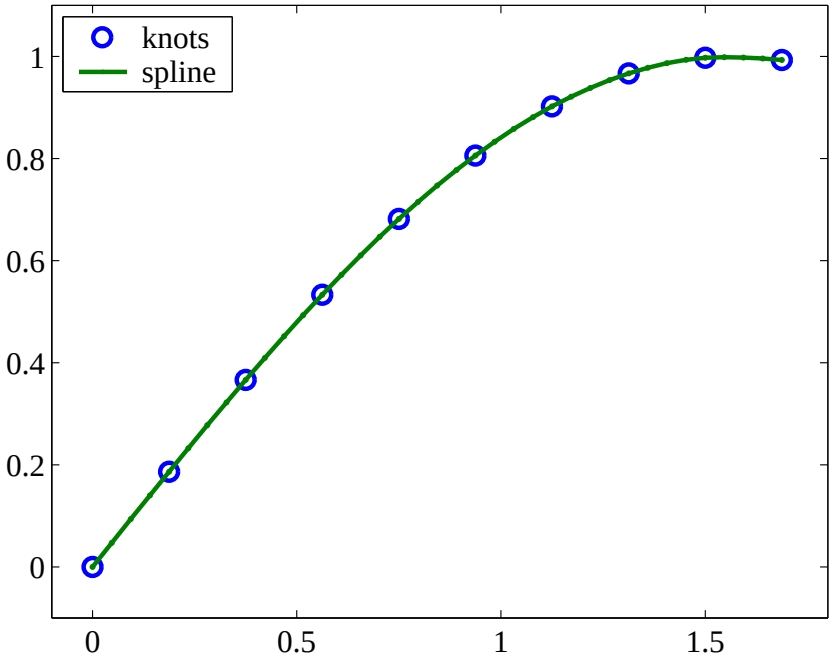
```
void Spline3_Coeff(int n,double *t,double *y,double *z)
{
  int i;
  double *h, *b, *u, *v;

  h = (double *) malloc((size_t) (n*sizeof(double)));
  b = (double *) malloc((size_t) (n*sizeof(double)));
  u = (double *) malloc((size_t) (n*sizeof(double)));
  v = (double *) malloc((size_t) (n*sizeof(double)));

  for(i=0; i<n; i++){
    h[i] = t[i+1]-t[i];
    b[i] = (y[i+1]-y[i])/h[i];
  }
  u[1]= 2.0*(h[0]+h[1]);
  v[1]= 6.0*(b[1]-b[0]);
  for(i=2; i<n; i++){
    u[i]= 2.0*(h[i]+h[i-1])-h[i-1]*h[i-1]/u[i-1];
    v[i]= 6.0*(b[i]-b[i-1])-h[i-1]*v[i-1]/u[i-1];
  }
  z[n]=0;
  for(i=n-1; i>0; i--){
    z[i] = (v[i]-h[i]*z[i+1])/u[i];
  }
  z[0]=0;

  free(h); free(b); free(u); free(v);
}
```

The following procedure evaluates the natural cubic spline $S(x)$ for a given value of x. The nested form is used in the evaluation of the S_i's.

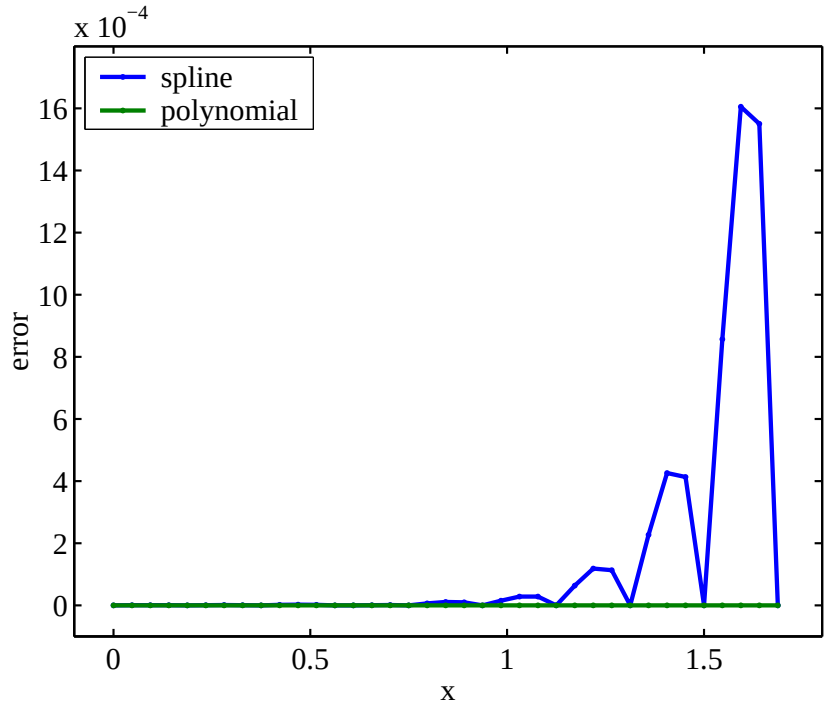
```
double Spline3_Eval(int n, double *t,
                    double *y, double *z, double x)
{
  int i;
  double h, tmp, result;

  for(i=n-1; i>=0; i--) {
    if(x-t[i]>=0)
      break;
  }
  h = t[i+1]-t[i];
  tmp = 0.5*z[i] + (x-t[i])*(z[i+1]-z[i])/(6.0*h);
  tmp = -(h/6.0)*(z[i+1]+2.0*z[i])+(y[i+1]-y[i])/h + (x-t[i])*tmp;
  result = y[i] + (x-t[i])*tmp;

  return(result);
}
```

Example

As an example, the natural cubic spline routines were implemented in a program which determines the natural cubic spline interpolant for $\sin x$ at ten equidistant knots in the interval $[0, 1.6875]$. The spline function is evaluated at 37 equally spaced points in the same interval. The figure below shows the spline function and the ten equidistant knots.



The figure below shows the error $|S(x) - sin(x)|$, and for comparison, also the error $|p(x) - sin(x)|$ obtained for *the Newton form of the interpolating polynomial* (Chapter 3). We note that in this case the spline interpolant is *not as accurate* as the polynomial!



Example 2

Here is another example which illustrates the differences between polynomial interpolation and cubic spline interpolation. Consider the **serpentine curve** given by

$$y = \frac{x}{1/4 + x^2}$$

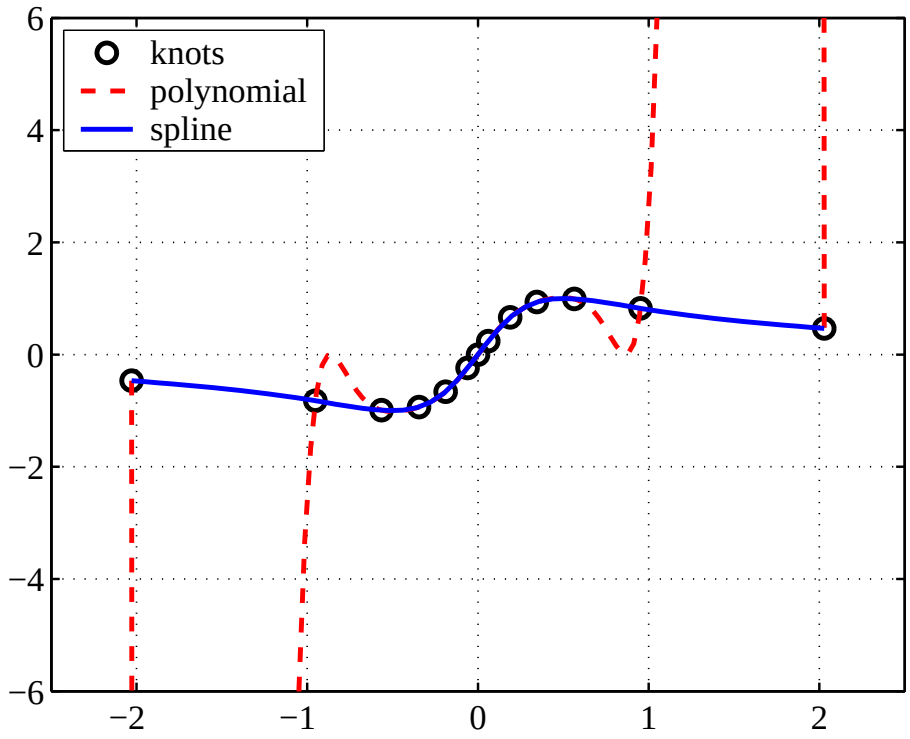
In order to have non-uniformly spaced knots, we write the curve in parametric form:

$$\begin{cases} x = \frac{1}{2}\tan\theta \\ y = \sin 2\theta \end{cases}$$

and take $\theta = i(\pi/12)$, where $i = -5, \ldots, 5$.

If we use the parametric representation to generate the knots $\{x(\theta), y(\theta)\}$, the order of the knots must be rearranged so that the array (t_i) runs from the smallest value to the largest. The values in array (y_i) are also rearranged to correspond to the ordering of (t_i). After this, we can use the procedures `Spline3_Coeff` and `Spline3_Eval` to determine the cubic spline interpolant of the serpentine curve.

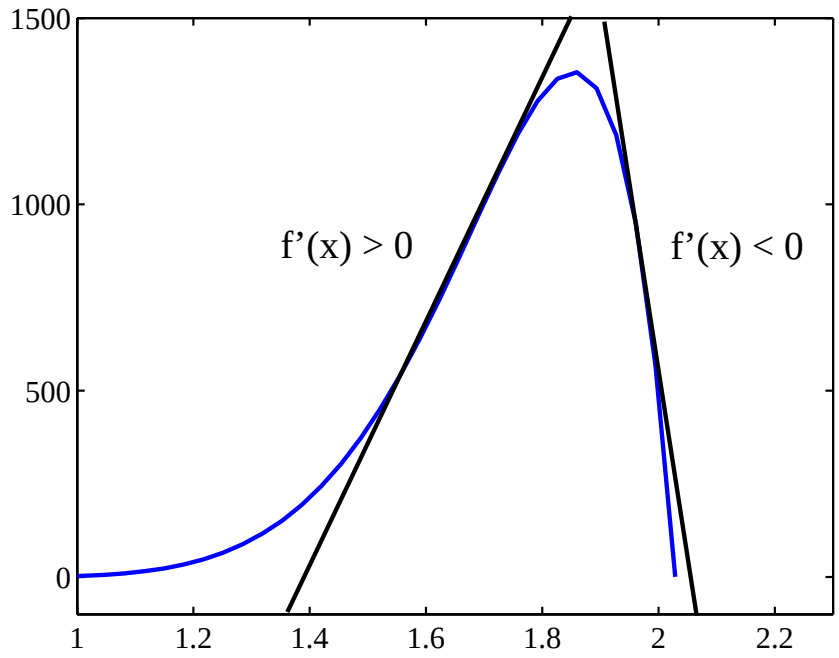
The figure below shows the 13 knots, the polynomial interpolant and the cubic spline interpolant of the serpentine curve. Notice how the polynomial becomes wildly oscillatory, whereas the spline is an excellent fit.



2.4 Smoothness property

The previous example illustrates why spline functions are better for data fitting than ordinary polynomials. Interpolation by high-degree polynomials is often unsatisfactory because polynomials exhibit *oscillations*.

Wild oscillations in a function can be attributed to its derivatives being very large. For example, for the curve in the figure below, $f'(x)$ is first large and positive and soon after large and negative. Consequently, there is a point where $f''(x)$ is large (since the value of f' changes rapidly).



Spline functions do *not* exhibit such oscillatory behavior. In fact, from a certain point of view, spline functions are the *optimal functions* for curve fitting.

Cubic spline smoothness theorem.
If S is the natural cubic spline function that interpolates a twice-continuously differentiable function f at knots $a = t_0 < t_1 < \ldots < t_n = b$, then

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

This theorem states that the average value of $[S''(x)]^2$ on the interval $[a,b]$ is never larger than the average value of $[f''(x)]^2$ on the same interval. Since $[f''(x)]^2$ is related to the curvature of f, we know that the spline interpolant will not oscillate more than the function f itself does.

3 B splines

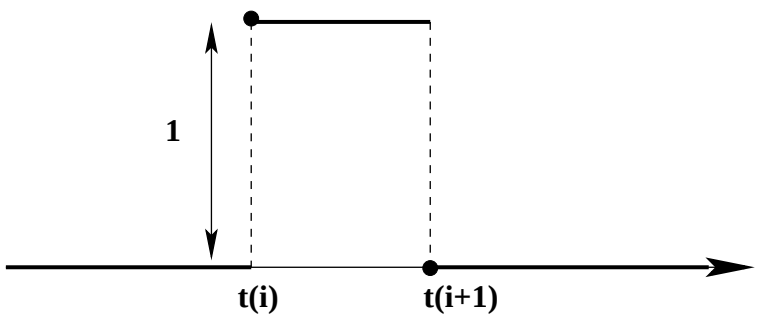
B splines are special spline functions that are well suited for numerical tasks. B splines are often used in software packages for approximating data, and therefore, familiarity with these functions is useful for anyone using such library codes. The name of B splines comes from the fact that they form a *basis* for the set of all splines.

Here we assume that an infinite set of knots $\{t_i\}$ has been prescribed in such a way that

$$\begin{cases} \ldots < t_{-2} < t_{-1} < t_0 < t_1 < t_2 < \ldots \\ \lim_{i\to\infty} t_i = \infty = -\lim_{i\to\infty} t_{-i} \end{cases}$$

The B splines depend on this set of knots, although the notation does not show the dependence. The **B splines of degree 0** are defined by

$$B_i^0(x) = \begin{cases} 1 & t_i \le x < t_{i+1} \\ 0 & \text{otherwise} \end{cases}$$



Obviously B_i^0 is discontinuous, but it is continuous from the right, even at the points where the jump occurs:

$$\lim_{x\to t_i^+} B_i^0(x) = 1 = B_i^0(t_i) \quad \text{and} \quad \lim_{x\to t_{i+1}^+} B_i^0(x) = 0 = B_i^0(t_{i+1})$$

The *support* of a function f is defined to be the set of points where $f(x) \neq 0$. Thus the support of B_i^0 is the half-open interval $[t_i, t_{i+1})$. Two further observations can be made:

$$\begin{cases} B_i^0(x) & \ge 0 \quad \text{for all } x \text{ and for all } i \\ \sum_{i=-\infty}^{\infty} B_i^0(x) & = 1 \quad \text{for all } x \end{cases}$$

In the second case, the convergence of the infinite series is obvious because for a given x, only one term in the series is other than 0.

As a final remark: Any spline of degree 0 that is continuous from the right and is based on the previously defined set of knots can be expressed as a linear combination of the B splines B_i^0; i.e. if the spline of degree 0 is given by

$$S(x) = b_i \qquad \text{if} \quad t_i \le x < t_{i+1} \quad (i = 0, \pm 1, \pm 2, \ldots)$$

then S can be written as

$$S = \sum_{i=-\infty}^{\infty} b_i \, B_i^0(x)$$

Higher degree splines

Using the functions B_i^0 as a starting point, we can generate all the *higher degree splines* by a simple *recursive* definition:

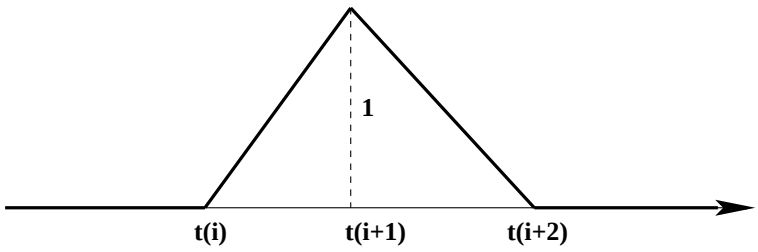
$$B_i^k(x) = \left(\frac{x - t_i}{t_{i+k} - t_i}\right) B_i^{k-1}(x) + \left(\frac{t_{i+k+1} - x}{t_{i+k+1} - t_{i+1}}\right) B_{i+1}^{k-1}(x) \quad (*)$$

where $k = 1, 2, \ldots$ and $i = 0, \pm 1, \pm 2, \ldots$.

As an illustration, let us determine B_i^1 in an alternative form:

$$B_i^1(x) = \left(\frac{x - t_i}{t_{i+1} - t_i}\right) B_i^0(x) + \left(\frac{t_{i+2} - x}{t_{i+2} - t_{i+1}}\right) B_{i+1}^0(x)$$

$$= \begin{cases} 0 & x \geq t_{i+2} \quad \text{or} \quad x \leq t_i \\ \dfrac{x - t_i}{t_{i+1} - t_i} & t_i \leq x < t_{i+1} \\ \dfrac{t_{i+2} - x}{t_{i+2} - t_{i+1}} & t_{i+1} \leq x < t_{i+2} \end{cases}$$



The splines B_i^1 are sometimes called *hat functions* or *chapeau functions* due to their triangular shape. In general, the functions $B_i^k(x)$ are called **B splines of degree k**. Since these functions are defined recursively, the degree of the functions increases by 1 at each step. Therefore, $B_i^1(x)$ is piecewise linear, $B_i^2(x)$ is piecewise quadratic, and so on.

It can be shown that

$$\text{i.} \qquad B_i^k(x) = 0 \qquad x \notin [t_i, t_{i+k+1})$$

and that

$$\text{ii.} \qquad B_i^k(x) > 0 \qquad x \in (t_i, t_{i+k+1})$$

The *principal use* of the B splines B_i^k is as the basis for the set of all kth degree splines that have the same knot sequence. Our first task is to develop an efficient method for evaluating a function of the form

$$f(x) = \sum_{i=-\infty}^{\infty} C_i^k B_i^k(x)$$

under the assumption that the coefficients C_i^k are given.

Using the recursive definition of $B_i^k(x)$, we obtain

$$\begin{aligned} f(x) &= \sum_{i=-\infty}^{\infty} C_i^k \left[\left(\frac{x-t_i}{t_{i+k}-t_i} \right) B_i^{k-1}(x) + \left(\frac{t_{i+k+1}-x}{t_{i+k+1}-t_{i+1}} \right) B_{i+1}^{k-1}(x) \right] \\ &= \sum_{i=-\infty}^{\infty} \left[C_i^k \left(\frac{x-t_i}{t_{i+k}-t_i} \right) + C_{i-1}^k \left(\frac{t_{i+k}-x}{t_{i+k}-t_i} \right) \right] B_i^{k-1}(x) \\ &= \sum_{i=-\infty}^{\infty} C_i^{k-1} B_i^{k-1}(x) \end{aligned}$$

where C_i^{k-1} is defined to be the appropriate coefficient from the second line of the equation above.

This algebraic manipulation shows how a linear combination of $B_i^k(x)$ can be expressed as a linear combination of $B_i^{k-1}(x)$. Repeating the process $k-1$ times, gives

$$f(x) = \sum_{i=-\infty}^{\infty} C_i^0 B_i^0(x)$$

The formula by which the coefficients C_i^{j-1} are obtained is

$$C_i^{j-1} = C_i^j \left(\frac{x-t_i}{t_{i+j}-t_i} \right) + C_{i-1}^j \left(\frac{t_{i+j}-x}{t_{i+j}-t_i} \right) \qquad (*)$$

Now, in order to calculate $f(x)$ on the interval $t_m \le x < t_{m+1}$ we only need $k+1$ coefficients since

$$f(x) = \sum_{i=-\infty}^{\infty} C_i^k B_i^k(x) = \sum_{i=m-k}^{m} C_i^k B_i^k(x)$$

The coefficients $C_m^k, C_{m-1}^k, \ldots, C_{m-k}^k$ can be calculated from the equation (*) by forming the following triangular array:

$$\begin{array}{ccccc} C_m^k & C_m^{k-1} & \cdots & \cdots & C_m^0 \\ C_{m-1}^k & C_{m-1}^{k-1} & \cdots & C_{m-1}^1 & \\ \vdots & \cdots & & & \\ C_{m-k}^k & & & & \end{array}$$

We can now establish that

$$f(x) = \sum_{i=-\infty}^{\infty} B_i^k(x) = 1 \qquad \text{for all } x \text{ and all } k \ge 0$$

We already know this for $k = 0$. For $k > 0$, we set $C_i^k = 1$ for all i. We can show (by induction) that all the subsequent coefficients $C_i^k, C_i^{k-1}, \ldots, C_i^0$ are also equal to 1. Thus, at the end,

$$f(x) = \sum_{i=-\infty}^{\infty} C_i^0 B_i^0(x) = \sum_{i=-\infty}^{\infty} B_i^0(x) = 1$$

Therefore the sum of all B splines of degree k is unity.

Differentiation and integration

In many applications, B splines can be used as substitutes for complex functions. Differentiation and integration are important examples.

The smoothness of B splines B_i^k increases with the index k. It can be shown that B_i^k has a continuous $(k-1)$th derivative. A basic result about the **derivatives of B splines** is

$$\frac{d}{dx}B_i^k(x) = \left(\frac{k}{t_{i+k}-t_i}\right)B_i^{k-1}(x) - \left(\frac{k}{t_{i+k+1}-t_{i+1}}\right)B_{i+1}^{k-1}(x)$$

This can be proven by induction using the recursive formula (*). Using this result, we get the following useful formula

$$\frac{d}{dx}\sum_{i=-\infty}^{\infty} c_i B_i^k(x) = \sum_{i=-\infty}^{\infty} d_i B_i^{k-1}(x)$$

where

$$d_i = k\left(\frac{c_i - c_{i-1}}{t_{i+k}-t_i}\right)$$

B splines are also recommended for **numerical integration**. Here is a basic result:

$$\int_{-\infty}^{x} B_i^k(s)ds = \left(\frac{t_{i+k+1}-t_{i+1}}{k+1}\right)\sum_{j=1}^{\infty} B_j^{k+1}(x)$$

This can be verified by differentiating both sides with respect to x.

This leads to the following useful formula

$$\int_{-\infty}^{x}\sum_{i=-\infty}^{\infty} c_i B_i^k(s)ds = \sum_{i=-\infty}^{\infty} e_i B_i^{k+1}(x)$$

where

$$e_i = \frac{1}{k+1}\sum_{j=-\infty}^{i} c_j(t_{j+k+1}-t_j)$$

This formula gives an indefinite integral of any function expressed as a linear combination of B splines. Any definite integral can be obtained by selecting a specific value for x.

For example, if x is a knot, say $x = t_m$, then

$$\int_{-\infty}^{t_m}\sum_{i=-\infty}^{\infty} c_i B_i^k(s)ds = \sum_{i=-\infty}^{\infty} e_i B_i^{k+1}(t_m) = \sum_{i=m-k-1}^{m} e_i B_i^{k+1}(t_m)$$

Matlab has a *Spline* toolbox which can be used for many tasks involving splines. For example, there are routines for interpolating data by splines and routines for least-squares fits to data. There are also many demonstrations. Use the Matlab help command to learn more about the properties.

4 Interpolation and approximation by B splines

We will now concentrate on the task of obtaining a B-spline representation of a given function. We begin by considering the problem of interpolating a set of data. Later a noninterpolatory method of approximation is discussed.

A basic question is how to determine the coefficients in the expression

$$S(x) = \sum_{i=-\infty}^{\infty} A_i B_{i-k}^k(x)$$

so that the resulting spline function interpolates a prescribed table of data:

x	t_0	t_1	$\dots$	t_n
y	y_0	y_1	$\dots$	y_n

Interpolation means that

$$S(t_i) = y_i \quad (0 \le i \le n)$$

B splines of degree 0

We begin with the simplest splines, corresponding to $k = 0$:

$$B_i^0(t_j) = \delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

Here the solution is obvious: we set $A_i = y_i$ for $0 \le i \le n$. All other coefficients are arbitrary.

Thus the zero-degree B spline

$$S(x) = \sum_{i=0}^{n} y_i B_i^0(x)$$

has the required interpolation property.

B splines of degree 1

The next case, $k = 1$, also has a simple solution. The first-degree spline $B_i^1(x)$ can be defined recursively using $B_i^0(x)$:

$$B_i^1(x) = \left(\frac{x - t_i}{t_{i+1} - t_i}\right) B_i^0(x) + \left(\frac{t_{i+2} - x}{t_{i+2} - t_{i+1}}\right) B_{i+1}^0(x)$$

$$= \begin{cases} 0 & x \ge t_{i+2} \quad \text{or} \quad x \le t_i \\ \dfrac{x - t_i}{t_{i+1} - t_i} & t_i < x < t_{i+1} \\ \dfrac{t_{i+2} - x}{t_{i+2} - t_{i+1}} & t_{i+1} \le x < t_{i+2} \end{cases}$$

We can now use the fact that at the nodes t_j we have

$$B_{i-1}^1(t_j) = \delta_{ij}$$

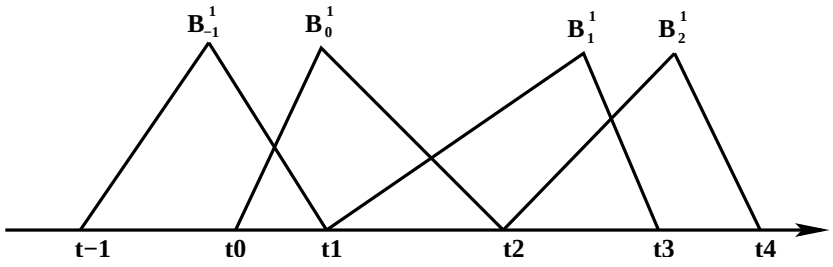
Hence, the first-degree B spline

$$S(x) = \sum_{i=0}^{n} y_i B_{i-1}^1(x)$$

has the required interpolation property.

If the table has four entries ($n = 3$), we use B_{-1}^1,B_0^1,B_1^1 and B_2^1. The knots $t_{-1}, t_0, t_1, t_2, t_3$ and t_4 are required for the definition of the four first-degree B splines. The knots t_{-1} and t_4 can be arbitrary.

The figure below shows the graphs of the four B_i^1 splines.



In both of these elementary cases, the unknown coefficients $A_0, A_1, \ldots, A_n$ were uniquely defined by the interpolation conditions. Any terms *outside* the range $\{0, 1, \ldots, n\}$ have no influence on the values of $S(x)$ at the knots $t_0, t_1, \ldots, t_n$.

For higher degree splines, there is some arbitrariness in choosing coefficients. In fact, *none* of the coefficients is uniquely determined by the interpolation conditions. This can be advantageous if other properties are sought from the solution.

B splines of degree 2

In the quadratic case, we want to determine the coefficients for the second-degree spline

$$S(x) = \sum_{i=-\infty}^{\infty} A_i B_{i-2}^2(x)$$

so that it interpolates the given table of $n+1$ data points.

At a node t_j, we can express $S(t_j)$ using the following equation

$$S(t_j) = \sum_{i=-\infty}^{\infty} A_i B_{i-2}^2(t_j) = \frac{1}{t_{j+1} - t_{j-1}} [A_j(t_{j+1} - t_j) + A_{j+1}(t_j - t_{j-1})]$$

Imposing the interpolation conditions $S(t_j) = y_j$, we obtain the following system of equations:

$$A_j(t_{j+1} - t_j) + A_{j+1}(t_j - t_{j-1}) = y_j(t_{j+1} - t_{j-1}) \qquad (0 \le j \le n)$$

This is a system of $n+1$ linear equations in $n+2$ unknowns $A_0, A_1, \ldots, A_{n+1}$. This gives us the necessary and sufficient conditions for the coefficients.

One way to solve the system of equations is to assign any value to A_0 and then compute $A_1, A_2, \ldots, A_{n+1}$ recursively. For this purpose, the equations could be rewritten in the following form

$$A_{j+1} = \alpha_j + \beta_j A_j \quad (0 \le j \le n)$$

where (for $0 \le j \le n$)

$$\begin{cases} \alpha_j & = y_j \left(\frac{t_{j+1} - t_{j-1}}{t_j - t_{j-1}} \right) \\ \beta_j & = \frac{t_j - t_{j+1}}{t_j - t_{j-1}} \end{cases}$$

In order to keep the coefficients small in magnitude, A_0 should be selected such that the following expression will be a minimum:

$$\Phi = \sum_{i=0}^{n+1} A_i^2$$

The process of determining A_0 is as follows: We can show that

$$A_{j+1} = \gamma_j + \delta_j A_0 \quad (0 \le j \le n)$$

where the coefficients γ_j and δ_j are obtained recursively by the following algorithm (for $1 \le j \le n$):

$$\begin{cases} \gamma_0 = \alpha_0 & \delta_0 = \beta_0 \\ \gamma_j = \alpha_j + \beta_j \gamma_{j-1} & \delta_j = \beta_j \delta_{j-1} \end{cases}$$

Then Φ is a quadratic function of A_0 as follows:

$$\begin{aligned} \Phi &= A_0^2 + A_1^2 + \ldots + A_{n+1}^2 \\ &= A_0^2 + (\gamma_0 + \delta_0 A_0)^2 + (\gamma_1 + \delta_1 A_0)^2 + \ldots + (\gamma_n + \delta_n A_0)^2 \end{aligned}$$

To find the minimum of Φ we take its derivative with respect to A_0 and set it equal to zero:

$$\frac{d\Phi}{dA_0} = 2A_0 + 2(\gamma_0 + \delta_0 A_0)\delta_0 + 2(\gamma_1 + \delta_1 A_0)\delta_1 + \ldots + 2(\gamma_n + \delta_n A_0)\delta_n = 0$$

This is equivalent to $qA_0 + p = 0$ where

$$\begin{cases} q & = 1 + \delta_0^2 + \delta_1^2 + \ldots + \delta_n^2 \\ p & = \gamma_0 \delta_0 + \gamma_1 \delta_1 + \ldots + \gamma_n \delta_n \end{cases}$$

Evaluation of the interpolating spline

Finally, a set of results is quoted here in order to construct a procedure for evaluating $S(x)$ once the coefficients A_i have been determined. The proof of these results is left as an exercise.

If the task is to evaluate the following second-degree spline

$$S(x) = \sum_{i=-\infty}^{\infty} A_i B_{i-2}^2(x)$$

at a given point x which lies in between the nodes t_{j-1} and t_j, then we can calculate $S(x)$ using the following equations

$$S(x) = \frac{1}{t_j - t_{j-1}}[d(x - t_{j-1}) + e(t_j - x)]$$

where the functions d and e are given by

$$d = \frac{1}{t_{j+1} - t_{j-1}}[A_{j+1}(x - t_{j-1}) + A_j(t_{j+1} - x)]$$

and

$$e = \frac{1}{t_j - t_{j-2}}[A_j(x - t_{j-2}) + A_{j-1}(t_j - x)]$$

We are now ready to construct a program which calculates a quadratic spline interpolant for a given set of data points.

Implementation

Two new routines are needed for the implementation of a program that determines a quadratic B spline interpolant for a given set of data points.

First, the following procedure, `BSpline2_Coef`, computes the coefficients $A_0, A_1, \ldots, A_{n+1}$ in the manner described before. t_i and y_i are the prescribed data points, a_i is the coefficient array, and h_i contains the subinterval lengths $h_i = t_i - t_{i-1}$.

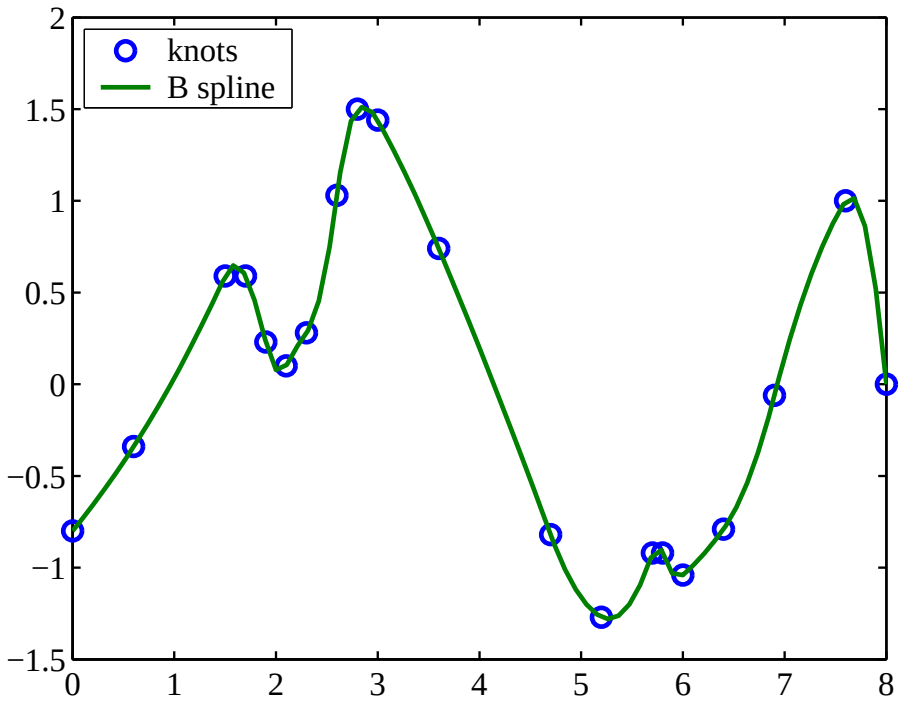
```
void BSpline2_Coeff(int n, double *t, double *y,
                    double *a, double *h)
{
  int i;
  double delta, gamma, p, q, r;

  for(i=1; i<=n; i++){
    h[i] = t[i]-t[i-1];
  }
  h[0] = h[1];
  h[n+1] = h[n];

  /* Determine A0 */
  delta = -1.0;
  gamma = 2.0 * y[0];
  p = delta*gamma;
  q = 2.0;
  for(i=1; i<=n; i++){
    r = h[i+1]/h[i];
    delta = -r*delta;
    gamma = -r*gamma + (r+1.0)*y[i];
    p = p + gamma*delta;
    q = q + delta*delta;
  }
  a[0]=-p/q;

  /* Determine other coefficients Ai */
  for(i=1; i<=n+1; i++){
    a[i] = ((h[i-1]+h[i])*y[i-1]-h[i]*a[i-1])/h[i-1];
  }
}
```

The following procedure, `BSpline2_Eval`, computes the values of the quadratic spline. Before calling this procedure, the arrays a_i and h_i must be determined using the previous procedure. The input variable x is a single real number that should lie between t_0 and t_n.

```
double BSpline2_Eval(int n, double *t, double *a,
                     double *h, double x)
{
  int i;
  double d, e, result;

  /* Check in which interval x lies */
  for(i=n-1; i>=0; i--) {
    if(x-t[i]>=0)
      break;
  }
  /* Evaluate S(x) */
  i = i+1;
  d = (a[i+1]*(x-t[i-1])+a[i]*(t[i]-x+h[i+1]))/(h[i]+h[i+1]);
  e = (a[i]*(x-t[i-1]+h[i-1])+a[i-1]*(t[i-1]-x+h[i]))/(h[i-1]+h[i]);

  result = (d*(x-t[i-1])+e*(t[i]-x))/h[i];

  return(result);
}
```

Output

Applied to a set 20 data points (for a randomly drawn free-hand curve), we obtain the following graph for the quadratic B spline.



The figure below shows a comparison of the quadratic B spline and a cubic spline.