Piecewise polynomial interpolation

For certain x-values $x_1 \le x_2 \le \cdots \le x_n$ we are given the function values $y_i = f(x_i)$. In some cases below we will also assume that we are additionally given some derivatives $s_i = f'(x_i)$. We want to find an interpolating function $p(x)$ which satisfies all the given data and is hopefully close to the function $f(x)$.

We could use a **single interpolating polynomial** $p(x)$. But this is usually a **bad idea**: for a large value of n we will obtain large oscillations.

We should **only use an interpolating polynomial** if we know that this will not be a problem and several of the following conditions hold

- the derivatives $f^{(k)}$ do not grow very fast (e.g., $f(x) = \sin x$)
- the points $x_1, \ldots, x_n$ are close together, and we evaluate $p(x)$ at a point $\tilde{x}$ inside of the interval $[x_1, x_n]$
- for equidistant nodes, we only evaluate $p(x)$ for $\tilde{x}$ near the center of the interval $[x_1, \ldots, x_n]$
- if we want to evaluate $p(x)$ over a whole interval $[a,b]$ we should choose $x_1, \ldots, x_n$ as Chebyshev nodes for this interval.

In all other cases it is much better to use a piecewise polynomial: We break the interval $[a,b]$ into smaller subintervals, and use polynomial interpolation with low degree polynomials on each subinterval. Typically we choose polynomial degree of about 3. This is a good compromise between small errors and control of oscillations.

Piecewise linear interpolation

We are given x-values $x_1, \ldots, x_n$ and y-values $y_i = f(x_i)$ for $i = 1, \ldots, n$. With $h_i := x_{i+1} - x_i$ we get

$$L_i(x) := f[x_i] + f[x_i, x_{i+1}](x - x_i) = y_i + \frac{y_{i+1} - y_i}{h}(x - x_i). \tag{1}$$

We then define $p(x)$ as the piecewise linear function with

$$\text{for } x \in [x_i, x_{i+1}]: \qquad p(x) = L_i(x)$$

We then have from the error formula for polynomial interpolation with 2 points that

$$\text{for } x \in [x_i, x_{i+1}]: \qquad f(x) - p(x) = \frac{f''(t)}{2!}(x - x_i)(x - x_{i+1})$$

$$|f(x) - p(x)| \le \tfrac{1}{2} \max_{t \in [x_i, x_{i+1}]} \left|f''(t)\right| \cdot \frac{h_i^2}{4}$$

since the function $|(x - x_i)(x - x_{i+1})|$ has its maximum $\frac{h_i}{2} \cdot \frac{h_i}{2}$ in the midpoint of the interval $[x_i, x_{i+1}]$.

We see that the interpolation error satisfies $|f(x) - p(x)| \le Ch_i^2$ where $C = \frac{1}{8} \max_{t \in [x_1, x_n]} |f''(t)|$. If we choose equidistant points with $h_i = (b-a)/(n-1)$ we have $|f(x) - p(x)| \le C(b-a)^2/n^2$, i.e., doubling the number of points reduces the error bound by a factor of 4.

However, if the function $f(x)$ has different behavior on different parts of the interval we can get better results by choosing the points $x_1, \ldots, x_n$ accordingly: If $|f''(x)|$ is small in a certain region we can use a wider spacing h_i; if $|f''(x)|$ is large in another reason we should place the nodes more closely, so that h_i is small there. In this way we can achieve a small overall error

$$\max_{x \in [x_1, x_n]} |f(x) - p(x)| \le \tfrac{1}{8} \max_{i = 1, \ldots, n-1} \left(h_i^2 \max_{t \in [x_i, x_{i+1}]} \left|f''(t)\right| \right)$$

with a small number of nodes. We say the choice of the nodes $x_1, \ldots, x_n$ is **adapted** to the behavior of the function f.

One advantage of piecewise linear interpolation is that the behavior of p resembles the behavior of f:

- whereever the function f is increasing/decreasing, we have that the function p is increasing/decreasing

However, we have drawbacks:

- the function $p(x)$ is not smooth: it has kinks (jumps of $p'(x)$) at the nodes $x_2, \ldots, x_{n-1}$ in general
- the error $|f(x) - p(x)| \le Ch_i^2$ for $x \in [x_i, x_{i+1}]$ only decreases fairly slowly with decreasing spacing h_i. We would rather have a higher power like Ch_i^4.

Piecewise cubic Hermite interpolation

Both of these drawbacks can be fixed by using a piecewise cubic polynomial $p(x)$.

We assume that we are given

- $x_1, \ldots, x_n$
- $y_1, \ldots, y_n$ where $y_i = f(x_i)$
- $s_1, \ldots, s_n$ where $s_i = f'(x_i)$

In this case we can construct on each interval $[x_i, x_{i+1}]$ a cubic Hermite polynomial $p_i(x)$ with

$$p_i(x_i) = y_i, \quad p'(x_i) = s_i, \qquad p(x_{i+1}) = y_{i+1}, \quad p'(x_{i+1}) = s_{i+1}.$$

E.g., on the first interval we obtain the following divided difference table: Let $r_1 := \dfrac{y_2 - y_1}{h_1}$

$$\begin{array}{c|cccc}
x_1 & y_1 & s_1 & \frac{r_1 - s_1}{h_1} & \frac{s_2 - 2r_1 + s_1}{h_1^2} \\
x_1 & y_1 & r_1 & \frac{s_2 - r_1}{h_1} & \\
x_2 & y_2 & s_2 & & \\
x_2 & y_2 & & &
\end{array}$$

yielding the following for the interpolating polynomial $p_1(x)$ on the interval $[x_1, x_2]$:

$$p_1(x) = y_1 + s_1(x - x_1) + \frac{r_1 - s_1}{h_1}(x - x_1)^2 + \frac{s_2 - 2r_1 + s_1}{h_1^2}(x - x_1)^2(x - x_2) \tag{2}$$

$$p_1''(x) = \frac{r_1 - s_1}{h_1} \cdot 2 + \frac{s_2 - 2r_1 + s_1}{h_1^2}\left[2(x - x_2) + 4(x - x_1)\right]$$

$$p_1''(x_1) = \frac{r_1 - s_1}{h_1} \cdot 2 + \frac{s_2 - 2r_1 + s_1}{h_1^2}\left[-2h_1\right] = \frac{6r_1 - 4s_1 - 2s_2}{h_1} \tag{3}$$

$$p_1''(x_2) = \frac{r_1 - s_1}{h_1} \cdot 2 + \frac{s_2 - 2r_1 + s_1}{h_1^2}\left[4h_1\right] = \frac{-6r_1 + 2s_1 + 4s_2}{h_1} \tag{4}$$

(We will need the second derivative later). In the same way we define $p_i(x)$ on the interval $[x_i, x_{i+1}]$.

The piecewise cubic Hermite polynomial $p(x)$ is then given by

$$\text{for } x \in [x_i, x_{i+1}]: \qquad p(x) = p_i(x) \tag{5}$$

Then we obtain from the error formula for polynomial interpolation with 4 points $x_i, x_i, x_{i+1}, x_{i+1}$ that

$$\text{for } x \in [x_i, x_{i+1}]: \qquad f(x) - p(x) = \frac{f^{(4)}(t)}{4!}(x - x_i)^2(x - x_{i+1})^2$$

$$|f(x) - p(x)| \le \tfrac{1}{24} \max_{t \in [x_i, x_{i+1}]} \left|f''(t)\right| \cdot \frac{h_i^4}{16}$$

since the function $|(x - x_i)(x - x_{i+1})|$ has its maximum $\frac{h_i}{2} \cdot \frac{h_i}{2}$ in the midpoint of the interval $[x_i, x_{i+1}]$.

We see that the interpolation error satisfies $|f(x) - p(x)| \le C h_i^4$ where $C = \frac{1}{24 \cdot 16} \max_{t \in [x_1, x_n]} \left|f^{(4)}(t)\right|$. If we choose equidistant points with $h_i = (b - a)/(n - 1)$ we have $|f(x) - p(x)| \le C(b - a)^4/n^4$, i.e., doubling the number of points reduces the error bound by a factor of 16.

However, if the function $f(x)$ has different behavior on different parts of the interval we can get better results by choosing the points $x_1, \ldots, x_n$ accordingly: If $\left|f^{(4)}(x)\right|$ is small in a certain region we can use a wider spacing h_i; if $\left|f^{(4)}(x)\right|$ is large in another reason we should place the nodes more closely, so that h_i is small there. In this way we can achieve a small overall error

$$\max_{x \in [x_1, x_n]} |f(x) - p(x)| \le \tfrac{1}{24 \cdot 16} \max_{i = 1, \ldots, n-1} \left(h_i^4 \max_{t \in [x_i, x_{i+1}]} \left|f^{(4)}(t)\right| \right)$$

with a small number of nodes.

Again, a major advantage of using piecewise polynomials is that we can pick a nonuniform spacing of the nodes adapted to the behavior of the function f.

The cubic Hermite spline has the following drawbacks:

- We need the derivatives $s_i = f'(x_i)$ at all nodes $x_1, \ldots, x_n$. In many cases these values are not available.
- We have that $p'(x)$ is continuous, but $p''(x)$ has jumps at the points $x_2, \ldots, x_{n-1}$ in general. We would like to have a smoother function $p(x)$.

Complete cubic spline

The complete cubic spline fixes these two problems. We now assume that we are given

- $x_1, \ldots, x_n$
- $y_1, \ldots, y_n$ where $y_i = f(x_i)$
- $s_1 = f'(x_1)$ and $s_n = f'(x_n)$,

i.e., we only need the derivatives at the endpoints (see the section "Not-a-knot spline" below if these are not available). If these values are given, we can pick *arbitrary numbers* $s_2, \ldots, s_{n-1}$ and obtain with (5) a piecewise cubic function $p(x)$ which interpolates all the given data values.

How should we pick the $n-2$ numbers $s_2, \ldots, s_{n-1}$ to obtain a "nice function" $p(x)$? We can actually use this freedom to achieve a function $p(x)$ where $p''(x)$ is continuous at the points $x_2, \ldots, x_{n-1}$: We want to pick the $n-2$ numbers $x_2, \ldots, x_{n-1}$ such that the $n-2$ equations

$$p''_{i-1}(x_i) = p''_i(x_i) \qquad i = 2, \ldots, n-2 \tag{6}$$

are satisfied. E.g., we want that the second derivatives from the left and the right coincide at the point x_2: Using (3) and (4) with indices shifted by 1) we get for $i = 2$ the equation

$$\begin{aligned} p''_1(x_2) &\stackrel{!}{=} p''_2(x_1) \\ \frac{-6r_1 + 2s_1 + 4s_2}{h_1} &= \frac{6r_2 - 4s_2 - 2s_3}{h_2} \\ \frac{2}{h_1}s_1 + \left(\frac{4}{h_1} + \frac{4}{h_2}\right)s_2 + \frac{2}{h_2}s_3 &= 6\left(\frac{r_1}{h_1} + \frac{r_2}{h_2}\right) \end{aligned}$$

Note that the value s_1 in the first equation and the value s_n in the last equation are given, and should therefore be moved to the right hand side. Hence we obtain the tridiagonal linear system (after diving each equation by 2)

$$\begin{bmatrix} \frac{2}{h_1} + \frac{2}{h_2} & \frac{1}{h_2} & & & \\ \frac{1}{h_2} & \frac{2}{h_2} + \frac{2}{h_3} & \frac{1}{h_3} & & \\ & \ddots & \ddots & \ddots & \\ & & \frac{1}{h_{n-3}} & \frac{2}{h_{n-3}} + \frac{2}{h_{n-2}} & \frac{1}{h_{n-2}} \\ & & & \frac{1}{h_{n-2}} & \frac{2}{h_{n-2}} + \frac{2}{h_{n-1}} \end{bmatrix} \begin{bmatrix} s_2 \\ s_3 \\ \vdots \\ s_{n-2} \\ s_{n-1} \end{bmatrix} = \begin{bmatrix} 3\left(\frac{r_1}{h_1} + \frac{r_2}{h_2}\right) - \frac{s_1}{h_1} \\ 3\left(\frac{r_2}{h_2} + \frac{r_3}{h_3}\right) \\ \vdots \\ 3\left(\frac{r_{n-3}}{h_{n-3}} + \frac{r_{n-2}}{h_{n-2}}\right) \\ 3\left(\frac{r_{n-2}}{h_{n-2}} + \frac{r_{n-1}}{h_{n-1}}\right) - \frac{s_n}{h_{n-1}} \end{bmatrix} \tag{7}$$

This gives the following algorithm for finding the cubic spline interpolation:

- for $i = 1, \ldots, n-1$: let $h_i := x_{i+1} - x_i$, $r_i := \frac{y_{i+1} - y_i}{h_i}$
- define the matrix A on the left hand side of (7) and the vector b on the right hand side of (7)
- solve the tridiagonal linear system $A \begin{bmatrix} s_2 \\ \vdots \\ s_{n-1} \end{bmatrix} = b$ using Gaussian elimination without pivoting

For a given point $\tilde{x} \in [x_1, x_n]$ we evaluate the cubic spline as follows:

- find the interval $[x_i, x_{i+1}]$ containing $\tilde{x}$
- evaluate $p_i(\tilde{x})$ using (2)

In **Matlab** we can find the **complete cubic spline** as follows: `yt = spline(`$[x_1,\ldots,x_n]$`,`$[s_1,y_1,\ldots,y_n,s_n]$`,xt)`
Here `xt` is a vector of points where we want to evaluate the spline, and `yt` is the corresponding vector of function values.

"Optimal energy" property for complete cubic spline

It turns out that a complete cubic spline gives a "smooth" function $p(x)$ "without large oscillations". In fact, the complete cubic spline is the optimal interpolating curve in a certain sense.

Historically, people constructing ships used thin flexible rulers made of wood (called "splines") to find "smooth curves" passing through given points (x_i, y_i). For a thin piece of wood of length L one needs a certain energy to bend it into a curve with curvature $\kappa(s)$ along the arc length $s \in [0, L]$:

$$E = \int_{s=0}^{L} C \cdot \kappa(s)^2 ds$$

Here C is a stiffness constant. If one tries to pass a thin piece of wood through a number of points and allows it to relax it will assume the shape with lowest possible energy E.

If we describe the curve by a function $y = p(x)$ we have for small slopes $p'(x)$ that $\kappa(s) \approx p''(x)$ and

$$E \approx E_0 := \int_{x=x_1}^{x_n} C \cdot p''(x)^2 dx.$$

It turns out that the complete cubic spline is the smoothest possible interpolating function in the following sense:

Among *all* functions $p(x)$ (not only piecewise polynomials) satisfying

$$p(x_1) = y_1, \ldots, p(x_n) = y_n, \qquad p'(x_1) = s_1, \quad p'(x_n) = s_n$$

the complete cubic spline has the lowest possible "energy" $\displaystyle\int_{x=x_1}^{x_n} p''(x)^2 dx$.

Not-a-knot cubic spline

Now assume that **we are not given any derivatives values**. We are given only $x_1, \ldots, x_n$ and the function values $y_1, \ldots, y_n$. In this case the best way to proceed is as follows: First drop x_2 and x_{n-1} and consider only the x-values $x_1, x_3, x_4, \ldots, x_{n-3}, x_{n-2}, x_n$ with the corresponding y-values. If we pick arbitrary values s_1, s_n we can find the interpolating cubic spline function $p(x)$ as explained above. The function $p(x)$ is a cubic function on the interval $[x_1, x_3]$ given by (2) with index 3 in place of 1 ("x_2 is not a knot"). Similarly $p(x)$ is a cubic function on the interval $[x_{n-2}, x_n]$ given by (2) with indices $n-2, n$ in place of 1,2 ("x_{n-1} is not a knot").

In order to determine s_1, s_n we need two additional equations: We get them from the points (x_2, y_2) and (x_{n-1}, y_{n-1}) and require

$$p(x_2) = y_2, \qquad p(x_{n-1}) = y_{n-1}$$

The first equation depends on s_1, s_3. The last equation depends on s_{n-2}, s_n. We therefore obtain a tridiagonal linear system for the unknowns $s_1, s_3, s_4, \ldots, s_{n-3}, s_{n-2}, s_n$. We solve this linear system using Gaussian elimination without pivoting and obtain a cubic spline function called the "not-a-not cubic spline".

In **Matlab** we can find the **not-a-knot cubic spline** as follows: `yt = spline(`$[x_1,\ldots,x_n]$`,`$[y_1,\ldots,y_n]$`,xt)`
Here `xt` is a vector of points where we want to evaluate the spline, and `yt` is the corresponding vector of function values.