1 polynomial interpolation

goal: given (x_i, f_i), $i = 0, \ldots, n$,

$$\text{find } p \in \mathcal{P}_n \text{ s.t. } p(x_i) = f_i,\ i = 0, \ldots, n. \tag{1.1}$$

applications (examples):

- "Extrapolation": typically $f_i = f(x_i)$ for an (unknown) function f. For $\overline{x} \notin \{x_0, \ldots, x_n\}$ the value $p(\overline{x})$ yields an approximation to $f(\overline{x})$.
- "Dense output/plotting of f", if only the values $f_i = f(x_i)$ are given (or, e.g., function evaluations are too expensive)
- Approximation of f: integration or differentiation of $f \to$ integrate or differentiate the interpolating polynom p

1.1 Existence and uniqueness of the polynomial interpolation problem

Theorem 1.1 (Lagrange interpolation) *Let the points ("knots") x_i, $i = 0, \ldots, n$, be distrinct. Then there exists, for all values $(f_i)_{i=0}^n \subset \mathbb{R}$, a unique interpolating polynomial $p \in \mathcal{P}_n$. It is given by*

$$p(x) = \sum_{i=0}^{n} f_i \ell_i(x), \qquad \ell_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} \tag{1.2}$$

The polynomials $(\ell_i)_{i=0}^n$ are called Lagrange basis *of the space $\mathcal{P}_n$ w.r.t. the points $(x_i)_{i=0}^n$.*

Proof: *1. step:* one observes that $\ell_i \in \mathcal{P}_n$, $i = 0, \ldots, n$.
2. step: one asserts that $\ell_i(x_j) = \delta_{ij}$, i.e., $\ell_i(x_i) = 1$ and $\ell_i(x_j) = 0$ for $j \neq i$.
3. step: Steps 1+2 imply that p given by (1.2) is a solution to the polynomial interpolation problem.
4. step: Uniqueness: Let p_1, $p_2 \in \mathcal{P}_n$ be two interpolating polynomials. Then, the difference $p := p_1 - p_2$ is a polynomial of degree n with (at least) $n + 1$ zeros. Hence, $p \equiv 0$, i.e., $p_1 = p_2$. □

Example 1.2 **slide 2**
The polynomial $p \in \mathcal{P}_2$ interpolating the data

$$(0, 0), \quad (\frac{\pi}{4}, \frac{\sqrt{2}}{2}) \quad (\frac{\pi}{2}, 1)$$

is given by

$$
\begin{aligned}
p(x) &= 0 \cdot \ell_0(x) + \frac{\sqrt{2}}{2} \cdot \ell_1(x) + 1 \cdot \ell_2(x), \\
\ell_0(x) &= \frac{(x-\pi/4)(x-\pi/2)}{(0-\pi/4)(0-\pi/2)} = 1 - (1.909...)x + (0.8105...)x^2, \\
\ell_1(x) &= \frac{(x-0)(x-\pi/2)}{(\pi/4-0)(\pi/4-\pi/2)} = (2.546...)x - (1.62...)x^2 \\
\ell_2(x) &= \frac{(x-0)(x-\pi/4)}{(\pi/2-0)(\pi/2-\pi/4)} = -(0.636...)x + (0.81...)x^2.
\end{aligned}
$$

That is, $p(x) = (1.164...)x - (0.3357...)x^2$ ∎

Example 1.3 *The data of Example 1.2 were chosen to be the values* $f_i = \sin(x_i)$*, i.e.,* $f(x) = \sin x$*. An approximation to* $f'(0) = 1$ *could be obtained as* $f'(0) \approx p'(0) = 1.164....$ *An approximation to* $\int_0^{\pi/2} f(x)\,dx = 1$ *is given by* $\int_0^{\pi/2} p(x)\,dx = 1.00232...$ ∎

1.2 Neville-Scheme

It is not efficient to evaluate the interpolating polynomial $p(x)$ at a point x based on (1.2) since it involves many (redundant) multiplications when evaluating the ℓ_i. Traditionally, an interpolating polynomial is evaluated at a point x with the aid of the *Neville scheme*:

Theorem 1.4 *Let* $x_0, \ldots, x_n$*, be distinct knots and let* f_i*,* $i = 0, \ldots n$*, be the corresponding values. Denote by* $p_{j,m} \in \mathcal{P}_m$ *the solution of*

$$\text{find } p \in \mathcal{P}_m, \text{ s.t. } p(x_k) = f_k \text{ for } k = j, j+1, \ldots, j+m. \tag{1.3}$$

Then, there hold the recursions:

$$p_{j,0} = f_j, \quad j = 0, \ldots, n \tag{1.4}$$

$$p_{j,m}(x) = \frac{(x-x_j)p_{j+1,m-1}(x) - (x-x_{j+m})p_{j,m-1}(x)}{x_{j+m}-x_j} \qquad m \geq 1 \tag{1.5}$$

The solution p *of (1.1) is* $p(x) = p_{0,n}(x)$*.*

Proof: (1.4) ✓
(1.5) Let $\pi :=$ be the right-hand side of (1.5). Then:

- $\pi \in \mathcal{P}_m$
- $\pi(x_j) = p_{j,m-1}(x_j) = f_j$
- $\pi(x_{j+m}) = p_{j+1,m-1}(x_{j+m}) = f_{j+m}$

- for $j+1 \le i \le j+m-1$ there holds

$$\pi(x_i) = \frac{(x_i - x_j)\overbrace{p_{j+1,m-1}(x_i)}^{=f_i} - (x_i - x_{j+m})\overbrace{p_{j,m-1}(x_i)}^{=f_i}}{x_{j+m} - x_j} =$$
$$= \frac{(x_i - x_j - x_i + x_{j+m})f_i}{x_{j+m} - x_j} = f_i$$

  Theorem 1.1 implies $\pi = p_{j,m}$.

□

Theorem 1.4 shows that evaluating p at x can be realized with the following scheme:

$$\begin{array}{c|cccccccccc}
x_0 & f_0 & =: & p_{0,0}(x) & \longrightarrow & p_{0,1}(x) & \longrightarrow & p_{0,2}(x) & \longrightarrow \ \ldots \ \longrightarrow & p_{0,n}(x) = p(x) \\
 & & & & \nearrow & & \nearrow & \vdots & & \nearrow \\
x_1 & f_1 & =: & p_{1,0}(x) & \longrightarrow & p_{1,1}(x) & & \vdots & & \\
 & & & & \nearrow & \vdots & & \vdots & & \\
x_2 & f_2 & =: & p_{2,0}(x) & & \vdots & & \vdots & & \\
\vdots & \vdots & & \vdots & & \vdots & & \vdots & & \\
\vdots & \vdots & & \vdots & & \vdots & \longrightarrow & p_{n-2,2}(x) & & \\
\vdots & \vdots & & \vdots & & \vdots & \nearrow & & & \\
\vdots & \vdots & & \vdots & \longrightarrow & p_{n-1,1}(x) & & & & \\
\vdots & \vdots & & \vdots & \nearrow & & & & & \\
x_n & f_n & =: & p_{n,0}(x) & & & & & &
\end{array}$$

⟦ here, the operation " $\begin{array}{c}\longrightarrow\\ \nearrow\end{array}$ " is realized by formula (1.5) ⟧

slide 3

Exercise 1.5 *Formulate explicitly the algorithm that computes (in a 2-dimensional array) the values $p_{i,j}$. How many multiplications (in dependence on n) are needed? (It suffices to state α in the complexity bound $O(n^\alpha)$.)* ■

The scheme computes the values "column by column". If *merely* the last value $p(x)$ is required, then one can be more memory efficient by overwriting the given vector of data:

Algorithm 1.6 (Aitken-Neville Scheme)
Input: knot vector $x \in \mathbb{R}^{n+1}$, vector $y \in \mathbb{R}^{n+1}$ of values, evaluation point $\overline{x} \in \mathbb{R}$
Output: $p(\overline{x})$, p solves (1.1)

for $m = 1 : n$ ***do***
  for $j = 0 : n - m$ ***do*** ▷ *array has triangular form*

$y_j := \frac{(\overline{x}-x_j)\,y_{j+1}-(\overline{x}-x_{j+m})\,y_j}{x_{j+m}-x_j}$
end for
end for
return y_0

Remark 1.7 • *Cost of Alg. 1.6:* $O(n^2)$

- *The knots x_i need <u>not</u> be sorted.*
- *The Neville scheme, i.e., the algorithm formulated in Exercise 1.5 is particularly convenient, if additional data points are added at a later time: one merely appends one additional row at the bottom.*

finis 1.DS

1.3 Newton representation of the interpolating polynomial (CSE)

The cost of evaluating the interpolating polynomial p at *a single* point x are $O(n^2)$. If the interpolating polynomial has to be evaluated in *many* points x (e.g., for plotting), then it is of interest to reduce the cost (i.e., number of floating point operations) from $O(n^2)$ to $O(n)$ per evaluation point x. The "classical" way to achieve this is with the *Horner scheme*.
The **Newton polynomials** ω_j, $j = 0, \ldots, n$, w.r.t. the knots $x_0, x_1, \ldots, x_n$, are given by

$$\omega_j(x) := \prod_{i=0}^{j-1}(x - x_i) \qquad \text{(an empty product is defined to be 1);}$$

written explicitly:

$$1,\ (x-x_0),\ (x-x_0)(x-x_1),\ (x-x_0)(x-x_1)(x-x_2),\ \ldots,\ (x-x_0)(x-x_1)\cdots(x-x_{n-1}). \quad (1.6)$$

These polynomials form a basis of $\mathcal{P}_n$. That is, for every polynomial $p(x)$ of degree n there are coefficients $d_0, \ldots, d_n$, such that

$$p(x) = d_0 \cdot 1 + d_1(x - x_0) + d_2(x - x_0)(x - x_1) + d_3(x - x_0)(x - x_1)(x - x_2) \quad (1.7)$$
$$+ \cdots + d_n(x - x_0)(x - x_1)\cdots(x - x_{n-1}). \quad (1.8)$$

Once the coefficients d_i are available, the polynomial $p(x)$ can be evaluated very efficiently by rearranging (1.7) as follows:

$$\begin{aligned} p(x) &= d_0 + d_1(x - x_0) + d_2(x - x_0)(x - x_1) + \ldots + d_n(x - x_0)(x - x_1)\ldots(x - x_{n-1}) = \\ &= d_0 + (x - x_0)\Big[d_1 + (x - x_1)\Big[d_2 + (x - x_2)\Big[\ldots\big[d_{n-1} + (x - x_{n-1})[d_n]\big]\ldots\Big]\Big]\Big] \end{aligned}$$

This procedure is formalized in the following "Horner scheme":

Algorithm 1.8 (Horner scheme)
Input: *knots x_i, coefficients d_i, evaluation point $\overline{x}$*
Output: $p(\overline{x}) = \sum_{j=0}^{n} d_j \, \omega_j(\overline{x})$

$y := d_n$
for $j = n-1 : -1 : 0$ ***do***
  $y = d_j + (\overline{x} - x_j)y$
end for
return y

Remark 1.9 *Cost:*

- $O(n^2)$ *to compute the coefficients* d_j *(→ see below)*
- $O(n)$ *to evaluate* $p(t)$ *using Alg. 1.8*

⇒ *Horner scheme is useful, if* p *is evaluated at "many" points* t*.*
The Horner scheme is particularly economical on multiplications. Thus, the Horner scheme is useful in situations where multiplications are expensive. An example is the evaluation of matrix polynomials $p(A) = \sum_{i=0}^{n} a_i \mathbf{A}^i$*, since the multiplication of two* $N \times N$ *matrices* $\mathbf{A}$, $\mathbf{B}$ *costs* $O(N^3)$ *floating point operations.*

We now answer the question how to determine the coefficients d_i in (1.7) for given data

$$(x_0, f_0), \quad (x_1, f_1), \quad \ldots, \quad (x_n, f_n).$$

This is achieved by using successively the interpolation conditions:

$x = x_0$ in (1.7)

$$f_0 \; = \; p(x_0) \; = \; d_0 \tag{1.9}$$

$x = x_1$ in (1.7)

$$f_1 \; = \; p(x_1) \; = \; d_0 + d_1(x_1 - x_0) = \; f_0 + d_1(x_1 - x_0)$$

$$\Rightarrow d_1 \; = \; \frac{f_1 - f_0}{x_1 - x_0} \tag{1.10}$$

$x = x_2$ in (1.7)

$$\begin{aligned} f_2 \; = \; p(x_2) \quad &= \quad d_0 + d_1(x_2 - x_0) + d_2(x_2 - x_0)(x_2 - x_1) \\ &= \quad f_0 + \frac{f_1 - f_0}{x_1 - x_0}(x_2 - x_0) + d_2(x_2 - x_0)(x_2 - x_1) \end{aligned}$$

Rearranging yields

$$\begin{aligned}
f_2 - f_1 + f_1 - f_0 - \frac{f_1 - f_0}{x_1 - x_0}(x_2 - x_0) &= d_2(x_2 - x_0)(x_2 - x_1) \\
\iff \frac{f_2 - f_1}{x_2 - x_1} + \frac{(f_1 - f_0)(x_1 - x_0)}{(x_1 - x_0)(x_2 - x_1)} - \frac{(f_1 - f_0)(x_2 - x_0)}{(x_1 - x_0)(x_2 - x_1)} &= d_2(x_2 - x_0) \\
\iff \frac{f_2 - f_1}{x_2 - x_1} - \frac{(f_1 - f_0)(x_0 - x_1) + (f_1 - f_0)(x_2 - x_0)}{(x_1 - x_0)(x_2 - x_1)} &= d_2(x_2 - x_0) \\
\iff \frac{f_2 - f_1}{x_2 - x_1} - \frac{f_1 - f_0}{x_1 - x_0} &= d_2(x_2 - x_0)
\end{aligned}$$

and finally

$$\frac{\frac{f_2 - f_1}{x_2 - x_1} - \frac{f_1 - f_0}{x_1 - x_0}}{x_2 - x_0} = d_2 \tag{1.11}$$

$$\vdots$$

(1.9), (1.10), and (1.11) suggest to define the so-called **divided differences** :

zeroth divided difference

$$f[x_0] := f(x_0) = f_0$$

first divided difference

$$f[x_0, x_1] := \frac{f(x_1) - f(x_0)}{x_1 - x_0} = \frac{f_1 - f_0}{x_1 - x_0} = \frac{f[x_1] - f[x_0]}{x_1 - x_0}$$

second divided difference

$$f[x_0, x_1, x_2] := \frac{\frac{f_2 - f_1}{x_2 - x_1} - \frac{f_1 - f_0}{x_1 - x_0}}{x_2 - x_0} = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0}$$

Analogously, we obtain the third divided difference

$$f[x_0, x_1, x_2, x_3] := \frac{\frac{\frac{f_3 - f_2}{x_3 - x_2} - \frac{f_2 - f_1}{x_2 - x_1}}{x_3 - x_1} - \frac{\frac{f_2 - f_1}{x_2 - x_1} - \frac{f_1 - f_0}{x_1 - x_0}}{x_2 - x_0}}{x_3 - x_0} = \frac{f[x_1, x_2, x_3] - f[x_0, x_1, x_2]}{x_3 - x_0}.$$

We recognize how the k-th divided difference should be defined:
The denominator is the difference $x_k - x_0$, the numerator is the difference between the $(k-1)$-th divided difference for the knots $x_1, \ldots, x_k$ and the $(k-1)$-th divided difference for the knots $x_0, x_1, \ldots, x_{k-1}$. Formally:

Definition 1.10 *The* **divided differences** *are given by the following recursion:*

$$f[x_i] = f(x_i) = f_i, \quad i = 0, 1, \ldots, n,$$

and

$$f[x_0, x_1, \ldots, x_k] := \frac{f[x_1, \ldots, x_k] - f[x_0, \ldots, x_{k-1}]}{x_k - x_0}. \tag{1.12}$$

The above discussion suggests that the coefficients d_i in (1.7) are given by the divided differences. This is indeed the case:

Theorem 1.11 *Let the knots $x_0, \ldots, x_n$ be distinct. Then the interpolating polynomial p has the form*

$$\begin{aligned} p(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) \\ + \cdots + f[x_0, x_1, \ldots, x_n](x - x_0) \cdots (x - x_{n-1}). \end{aligned} \tag{1.13}$$

Proof: For any polynomial $\pi \in \mathcal{P}_n$ of the form $\pi(x) = \sum_{i=0}^{n} a_i x^i$ we define its *leading coefficient* $lc(\pi) := a_n$. We show, with the notation of Theorem 1.4 that, for any j, k

$$lc(p_{j,k}) = f[x_j, \ldots, x_{j+k}]. \tag{1.14}$$

To see (1.14), we proceed by induction on k. By definition, we have $p_{j,0} = f[x_j]$ for all j. Let us assume that (1.14) holds true for all $k \leq K$. Then with the aid of Theorem 1.4

$$\begin{aligned} lc(p_{j,K+1}) &\overset{\text{Thm. 1.4}}{=} \frac{lc(p_{j+1,K}) - lc(p_{j,K})}{x_{j+(K+1)} - x_j} \overset{\text{induction hyp.}}{=} \frac{f[x_{j+1}, \ldots, x_{j+1+K}] - f[x_j, \ldots, x_{j+K}]}{x_{j+(K+1)} - x_j} \\ &\overset{\text{Def. 1.10}}{=} f[x_j, \ldots, x_{j+K+1}]. \end{aligned}$$

This shows (1.14). From (1.14) we obtain the claim of the theorem (why?). □

Remark 1.12 *Divided differences can be interpreted as approximations to derivatives.*

1. *Consider the specific knots $x_1 = x_0 + h$, $x_2 = x_0 + 2h$, $x_3 = x_0 + 3h, \ldots$ for small h. Then we have (the $\approx$ becomes an equality in the limit $h \to 0$):*

$$\begin{aligned} f[x_0, x_1] &= \frac{f_1 - f_0}{h} \approx f'(x_0) \\ f[x_0, x_1, x_2] &= \frac{f[x_1, x_2] - f[x_0, x_1]}{2h} \approx \frac{1}{2}\frac{f'(x_1) - f'(x_0)}{h} \approx \frac{1}{2} f''(x_0) \\ f[x_0, x_1, x_2, x_3] &= \frac{f[x_1, x_2, x_3] - f[x_0, x_1, x_2]}{3h} \approx \frac{1}{3}\frac{\frac{1}{2}f''(x_1) - \frac{1}{2}f''(x_0)}{h} \approx \frac{1}{2 \cdot 3} f'''(x_0). \end{aligned}$$

   In general, one has

$$f[x_0, x_1, \ldots, x_k] \approx \frac{1}{k!} f^{(k)}(x_0). \tag{1.15}$$

2. *This observation suggests to* define *for $x_0 = x_1 = \ldots = x_k$ the divided difference by*

$$f[x_0, x_1, \ldots, x_k] := \frac{1}{k!} f^{(k)}(x_0).$$

   This definition also allows one to generalize the definition of divided differences to the case when some knots coincide. With this generalized notion of divided differences, the statement of Theorem 1.11 is also true if some knots coincide.

3. In general, for any knot sequence $x_0, \ldots, x_n$ there is an intermediate point

$$\xi \in (\min\{x_0, \ldots, x_k\}, \max\{x_0, \ldots, x_k\})$$

such that

$$f[x_0, \ldots, x_k] = \frac{1}{k!} f^{(k)}(\xi)$$

■

Exercise 1.13 *Formulate an algorithm similar to the Neville scheme to compute the divided differences $f[x_0], \ldots, f[x_0, \ldots, x_n]$. How expensive is the evaluation of an interpolating polynomial of degree n in M points?* ■

1.4 Extrapolation as a prime application of the Neville scheme

slide 4

A typical application of the Neville scheme is the extrapolation of a function value that is not directly accessible. The following example determines the derivative of a function if only function values are available.

Exercise 1.14 *Let $u(x) = \exp(x)$. We seek an approximation to $u'(0)$. Define the function*[1]

$$h \mapsto D(h) := \begin{cases} \frac{u(0+h)-u(0)}{h} & h \neq 0 \\ u'(0) & h = 0 \end{cases}$$

Compute the Neville scheme for $h = 2^{-j}$, $j = 0, 1, \ldots, 10$. Compute a second array containing the actually errors[2]*. What do you observe in the first, second, and third column of the Neville scheme?*

slide 5

■

1.5 a simple error estimate

We now assume that the values f_i are point values of a function f, i.e., $f_i = f(x_i)$.
Question: how big is the error $f(x) - p(x)$ for the interpolating polynomial p?
We have:

[1] the given definition of $D(0)$ is natural since it is the limit $\lim_{h\to 0} D(h)$. It is a formal definition since the algorithm will not require knowledge of $D(0)$.

[2] recall: $u'(0) = \exp(0) = 1$

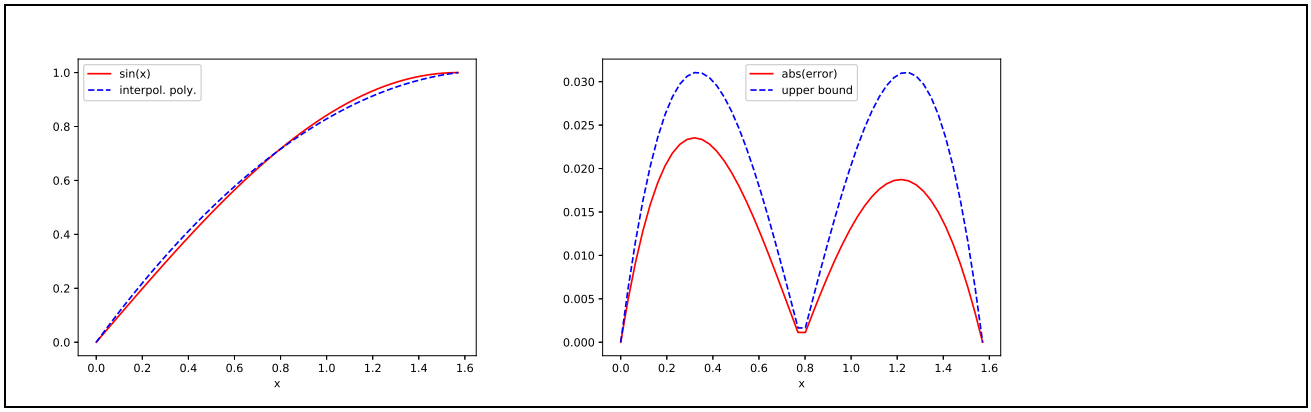


Figure 1.1: Left: $f(x)$ and the interpolating polynomial. Right: absolute value of the error and upper bound.

Theorem 1.15 *Let $[a,b] \subset \mathbb{R}$ and the knots $x_i \in [a,b]$, $i = 0,\dots,n$, be distinct. Let $f \in C^{(n+1)}([a,b])$, and let p be the interpolating polynomial. Then there exists a $\xi \in (a,b)$ such that*

$$f(x) - p(x) = (x - x_0)\cdots(x - x_n)\frac{f^{(n+1)}(\xi)}{(n+1)!} = \omega_{n+1}(x)\frac{f^{(n+1)}(\xi)}{(n+1)!}, \tag{1.16}$$

where

$$\omega_{n+1}(x) := \prod_{j=0}^{n}(x - x_i) = (x - x_0)\cdots(x - x_n).$$

Proof: *1. step:* (recalling the mean value theorem/Rolle's theorem) Let $g \in C^1([a',b'])$ for an interval $[a',b']$ with $g(a') = g(b')$. Then there exists $\xi \in (a',b')$ such that $g'(\xi) = 0$.
2. step: The claim is trivial for $x \in \{x_0,\dots,x_n\}$. (Why?)
3. step: Let $x \notin \{x_0,\dots,x_n\}$ be fixed. Consider the function

$$t \mapsto g(t) := f(t) - p(t) - K\omega_{n+1}(t), \qquad K := \frac{f(x) - p(x)}{\omega_{n+1}(x)}$$

Then, g has at least $n+2$ zeros (the knots x_i, $i = 0,\dots,n$, and x). By the first step, g' has at least $n+1$ distinct zeros. Hence, (again by the first step) g'' has n distinct zeros. Repeating these considerations one sees that $g^{(n+1)}$ has at least one zero ξ. Hence, (note: $p^{(n+1)} \equiv 0$ since $p \in \mathcal{P}_n$ and $\omega_{n+1}^{(n+1)}(x) = (n+1)!$)

$$0 = g^{(n+1)}(\xi) = f^{(n+1)}(\xi) - p^{(n+1)}(\xi) - K\omega_{n+1}^{(n+1)}(\xi) = f^{(n+1)}(\xi) - K(n+1)!.$$

Hence, $K = \frac{f^{(n+1)}(\xi)}{(n+1)!}$, which completes the proof. □

The error formula (1.16) yields bounds for the interpolation error:

Example 1.16 *(cf. Example 1.2) Let $f(x) = \sin x$ and $[a,b] = [0,\pi/2]$. Let $x_0 = 0$, $x_1 = \pi/4$, $x_2 = \pi/2$. Then the interpolating polynomial $p \in \mathcal{P}_2$ satisfies in view of $\max_{y\in\mathbb{R}} |f^{(3)}(y)| = \max_{y\in\mathbb{R}} |-\cos y| \le 1$*

$$|f(x) - p(x)| \le |\omega_3(x)|\frac{|f^{(3)}(\xi)|}{3!} \le \frac{1}{6}|\omega_3(x)| = \frac{1}{6}|(x-0)(x-\pi/4)(x-\pi/2)|.$$

Fig. 1.1 visualizes this estimate. The upper bound is pretty good in this example: it overestimates the error merely by a factor 1.5. ■

h	m=0	m=1	m=2	m=3	m=4	m=5	m=6	m=7
2^0	1.00_0	4.14_{-1}	2.52_{-1}	1.68_{-1}	1.15_{-1}	8.06_{-2}	5.66_{-2}	3.99_{-2}
2^{-1}	7.07_{-1}	2.93_{-1}	1.79_{-1}	1.19_{-1}	8.17_{-2}	5.70_{-2}	4.00_{-2}	2.82_{-2}
2^{-2}	5.00_{-1}	2.07_{-1}	1.26_{-1}	8.40_{-2}	5.77_{-2}	4.03_{-2}	2.83_{-2}	
2^{-3}	3.54_{-1}	1.46_{-1}	8.93_{-2}	5.94_{-2}	4.08_{-2}	2.85_{-2}		
2^{-4}	2.50_{-1}	1.04_{-1}	6.31_{-2}	4.20_{-2}	2.89_{-2}			
2^{-5}	1.77_{-1}	7.32_{-2}	4.46_{-2}	2.97_{-2}				
2^{-6}	1.25_{-1}	5.18_{-2}	3.16_{-2}					
2^{-7}	8.84_{-2}	3.66_{-2}						
2^{-8}	6.25_{-2}							
Fehler	$\sqrt{h}$	$\sqrt{h}$	$\sqrt{h}$	$\sqrt{h}$	$\sqrt{h}$	$\sqrt{h}$		

Figure 1.2: (cf. Example 1.18) Extrapolation error at $h = 0$ for the function $h^{-1}(u(h) - u(0))$ with $u(x) = |x|^{3/2}$.

The error formula explains the convergence behavior that was observed in Exercise 1.14 for the columns of the Neville scheme:

Theorem 1.17 *Let $f \in C^{(n+1)}([a,b])$ and $h_i = q^i$, $i = 0, 1, \ldots$, for a $0 < q < 1$. Let $x_0 \in [a,b]$. Denote by $p_{i,m} \in \mathcal{P}_m$ the polynomial that interpolates f in the points $x_0 + h_{i+j}$, $j = 0, \ldots, m$. Then there exists a constant $C > 0$ (which depends on f, m, and q), such that for $m \leq n+1$*

$$|f(x_0) - p_{i,m}(x_0)| \leq C h_i^{m+1} \tag{1.17}$$

slide 5

The assumption that f be smooth (i.e. n in Theorem 1.17 is fairly large), is essential for the rapid convergence behavior in the columns of the Neville scheme:

Example 1.18 **slide 6**

Consider the Neville scheme as in Exercise 1.14 for the function $u(x) = |x|^{3/2}$, i.e., $D(h) = \sqrt{|h|}$. Then D is not smooth—it is not even differentiable at $h = 0$. Fig. 1.2 shows the errors $|D(0) - p_{i,m}(0)|$. We observe that increasing m does not lead to better results. ∎

Often the interpolation error is measured in a norm, e.g., the *maximum norm*. For an interval $[a,b]$, the maximum norm $\|g\|_{\infty,[a,b]}$ of a function $g \in C([a,b])$ is defined by

$$\|g\|_{\infty,[a,b]} := \max_{x \in [a,b]} |g(x)|. \tag{1.18}$$

Theorem 1.15 implies for the interpolation error

$$\|f - p\|_{\infty,[a,b]} \leq \|\omega_{n+1}\|_{\infty,[a,b]} \frac{\|f^{(n+1)}\|_{\infty,[a,b]}}{(n+1)!} \leq (b-a)^{n+1} \frac{\|f^{(n+1)}\|_{\infty,[a,b]}}{(n+1)!}$$

Often, one approximates functions by *piecewise* polynomials as illustrated in the following exercise:

Exercise 1.19 *The goal is the approximate the function f on the interval $[a,b]$ by a piecewise polynomial of degree n. Proceed as follows: Partition $[a,b]$ in N subintervals $[t_j, t_{j+1}]$, $j = 0, \ldots, N-1$, of length $h = (b-a)/N$ with $t_j = a + jh$. In each subinterval $[t_j, t_{j+1}]$ select*

as the interpolation points $x_{i,j} := t_j + \frac{1}{n}ih$, $i = 0, \ldots, n$, and approximate f on $[t_j, t_{j+1}]$ by the polynomial that interpolates f in the points $x_{i,j}$, $i = 0, \ldots, n$. In this way, one obtains a function p that is a polynomial of degree n on each subinterval. Show:

$$\|f - p\|_{\infty,[a,b]} \leq \frac{1}{(n+1)!} h^{n+1} \|f^{(n+1)}\|_{\infty,[a,b]}.$$

Sketch the function p for the case $n = 1$. ∎

1.6 Extrapolation of function with additional structure

Sometimes, the function f to be approximated has additional structure that can (and should!) be exploited. We illustrate this phenomenon for the approximation of the derivative of a function using symmetric difference quotients:

Example 1.20 slide 7
Given a function u consider the function

$$D_{sym}(h) := \frac{u(0+h) - u(0-h)}{2h} = u'(0) + \frac{1}{3!}u^{(3)}(0)h^2 + \frac{1}{5!}u^{(5)}h^4 + \cdots$$

The goal is to approximate $D_{sym}(0)$ using only evaluations of u. We recognize that D_{sym} is a function of h^2, i.e., $D_{sym}(h) = \widetilde{D}(h^2)$. If $(h_i, D_{sym}(h_i))$, $i = 0, \ldots, n$, are given, then one could obtain an approximation of $D_{sym}(0)$ in 2 ways:

1. *Interpolate the data $(h_i, D_{sym}(h_i))$, $i = 0, \ldots, n$, and evaluate the interpolating polynomial at $h = 0$.*
2. *Interpolate the data $(h_i^2, D_{sym}(h_i)) = (h_i^2, \widetilde{D}(h_i^2))$, $i = 0, \ldots, n$, and evaluate the interpolating polynomial at $h^2 = 0$.*

Effectively, the first approach interpolates the function D_{sym} whereas the second approach interpolates the function $\widetilde{D}$. In practice, the interpolation of $\widetilde{D}$ is again realized with a Neville scheme:

h	h^2	$m = 0$	$m = 1$	$m = 2$	$m = 3$
h_0	h_0^2	$D_{sym}(h_0) = D_{00}$	D_{01}	D_{02}	D_{03}
h_1	h_1^2	$D_{sym}(h_1) = D_{10}$	D_{11}	D_{12}	D_{13}
h_2	h_2^2	$D_{sym}(h_2) = D_{20}$	D_{21}	D_{22}	D_{23}
h_3	h_3^2	$D_{sym}(h_3) = D_{30}$	D_{31}	D_{32}	D_{33}
h_4	h_4^2	$D_{sym}(h_4) = D_{40}$	D_{41}	D_{42}	$\vdots$
h_5	h_5^2	$D_{sym}(h_5) = D_{50}$	D_{51}	$\vdots$	
h_6	h_6^2	$D_{sym}(h_6) = D_{60}$	$\vdots$		
$\vdots$	$\vdots$	$\vdots$			

h		$m=0$	$m=1$	$m=2$	$m=3$	$m=4$	$m=5$
1		1.175201193643802	0.909180028331188	**1.001**883888526739	**0.999**862158028692	**1.000000**383252421	**0.99999999**3723462
2^{-1}		1.042190610987495	**0.97**8707923477851	**1.000**114874340948	**0.99999**1744175937	**1.0000000**05896242	0
2^{-2}		1.010449267232673	**0.99**4763136625174	**1.000007**135446563	**0.999999**489538722	0	0
2^{-3}		1.002606201928923	**0.99**8696135741217	**1.000000**445277203	0	0	0
2^{-4}		1.000651168835070	**0.999**674367893206	0	0	0	0
2^{-5}		1.000162768364138	0	0	0	0	0

h	h^2	$m=0$	$m=1$	$m=2$	$m=3$	$m=4$	$m=5$
1	1	1.175201193643802	**0.997**853750102059	**1.000003**157261889	**0.99999999**319035	**1.000000000000025**	**1.000000000000001**
2^{-1}	2^{-2}	1.042190610987495	**0.999**868819314399	**1.000000**048661892	**0.99999999997**365	**1.000000000000001**	
2^{-2}	2^{-4}	1.010449267232673	**0.99999**1846827674	**1.000000000**757749	**0.999999999999990**		
2^{-3}	2^{-6}	1.002606201928923	**0.999999**491137119	**1.000000000**11830			
2^{-4}	2^{-8}	1.000651168835070	**0.9999999**68207161				
2^{-5}	2^{-10}	1.000162768364138					

Figure 1.3: (cf. Example 1.20) Top: Extrapolation of the function $h \mapsto D_{sym}(h)$. Bottom: Extrapolation of the function $h^2 \mapsto \widetilde{D}(h^2)$ for $u(x) = \exp(x)$. Correct digits are marked in boldface.

$$\begin{aligned} D_{i0} &= D_{sym}(h_i) \\ D_{ij} &= D_{(i+1)\,(j-1)} - \frac{h_{i+j}^2}{h_{i+j}^2 - h_i^2}\left[D_{(i+1)\,(j-1)} - D_{i\,(j-1)}\right], \qquad j \geq 1 \end{aligned}$$

Fig. 1.3 illustrates both approaches for the function $u(x) = \exp(x)$. We observe that the extrapolation of $\widetilde{D}$ yields much better results than the extrapolation of D_{sym} at comparable costs. Intuitively, this can be seen as follows: Let $\widetilde{p} \in \mathcal{P}_n$ be the interpolant for the points $(h_{i+j}^2, D_{sym}(h_{i+j}))$, $j = 0, \ldots, m$. Then, $\widetilde{p}(h^2) \in \mathcal{P}_{2m}$ interpolates the data (Exercise! Note: D_{sym} is a symmetric function)

$$(h_{i+j}, D_{sym}(h_{i+j}), (-h_{i+j}, D_{sym}(-h_{i+j})) = (-h_{i+j}, D_{sym}(h_{i+j})), \qquad j = 0, \ldots, m.$$

Effectively, therefore, $\widetilde{p}(h^2)$ is an interpolating polynomial of degree $2m$ instead of m. From Theorem 1.17 we therefore expect error bounds of the form Ch_i^{2m} in column m of the Neville scheme. ∎

Exercise 1.14 and Example 1.20 present two different ways to approximate the derivative of a function using difference quotients. The extrapolation based on "symmetric difference quotients" D_{sym} of Example 1.20 yields much more accurate approximations than the extrapolation based on the one-sided difference quotients of Exercise 1.14 at the same computational cost. For smooth u, the former method is therefore preferred. finis 2.DS

1.7 Chebyshev polynomials

1.7.1 Chebyshev points

Question: If one is allowed to choose the interpolation points, which one should one choose?
The representation of the interpolation error (1.16) has the advantage of being an *equality*. It has the disadvantage that the intermediate point ξ is not known and depends on the function f and the chosen knots x_i. Typically, one does not study the error in single points but studies

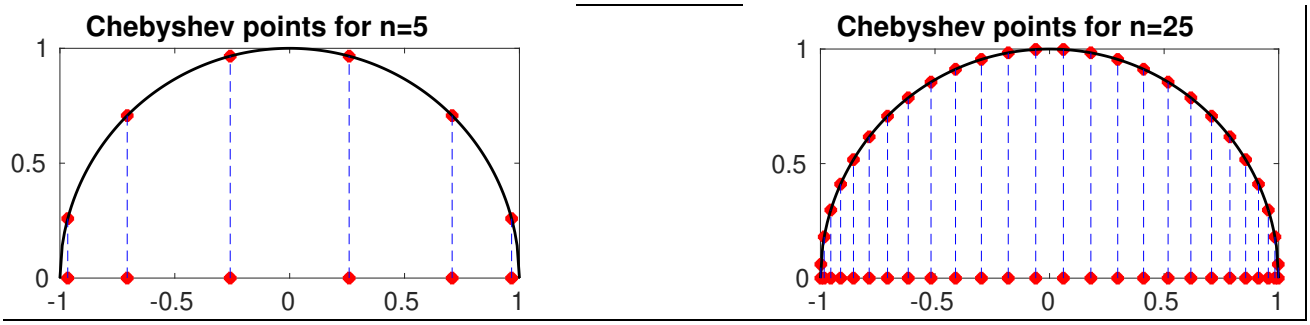


Figure 1.4: Chebyshev points $x_{i,n}^{Cheb}$, $i = 0, \dots, n$, for $n = 5$ (left) and $n = 25$ (right).

the interpolation error in a norm. Here, we consider the maximum norm and estimate

$$\|f - p\|_{\infty,[a,b]} \leq \underbrace{\|\omega_{n+1}\|_{\infty,[a,b]}}_{\text{depends } \textit{solely} \text{ on the knots}} \underbrace{\frac{\|f^{(n+1)}\|_{\infty,[a,b]}}{(n+1)!}}_{\text{depends } \textit{solely} \text{ on } f \text{ and } n}$$

This shows that a sensible strategy to choose the knots x_i, $i = 0, \dots, n$, is to minimize $\|\omega_{n+1}\|_{\infty,[a,b]}$:

$$\text{given } n, \text{ find } x_i \in [a,b] \text{ s.t.} \|\omega_{n+1}\|_{\infty,[a,b]} \text{ is minimal,} \tag{1.19}$$

where again $\omega_{n+1}(x) = (x - x_0) \cdots (x - x_n)$. This minimization problem has a solution, the so-called Chebyshev points:

Theorem 1.21 (Chebyshev points) *The minimization problem* (1.19) *has a solution given by*

$$x_i = \frac{a+b}{2} + \frac{b-a}{2} x_{i,n}^{Cheb}, \qquad x_{i,n}^{Cheb} := \cos\left(\pi \frac{2i+1}{2n+2}\right), \qquad i = 0, \dots, n. \tag{1.20}$$

For this choice of interpolation points, there holds

$$\|\omega_{n+1}^{Cheb}\|_{\infty,[a,b]} = 2\left(\frac{b-a}{4}\right)^{n+1}$$

In particular, for every choice of interpolation points x_i with corresponding polynomial ω_{n+1} there holds

$$\|\omega_{n+1}\|_{\infty,[a,b]} \geq \|\omega_{n+1}^{Cheb}\|_{\infty,[a,b]}$$

Example 1.22 slide 8
The Chebyshev points $x_{i,n}^{Cheb}$, $i = 0, \dots, n$, for the interval $[-1, 1]$ are not *uniformly distributed in the interval $[-1, 1]$ but more closely spaced near the endpoints ± 1. Fig. 1.4 illustrates this.* ∎

1.7.2 Error bounds for Chebyshev interpolation

Question: How does the interpolation error compare to the best approximation error?
We fix the interval $[a, b] = [-1, 1]$ and denote by $I_n^{Cheb} f \in \mathcal{P}_n$ the polyomial of degree n that interpolates f in the Chebyshev points.

Exercise 1.23 *The mapping $f \mapsto I_n^{Cheb} f$ is a linear map, i.e., for continuous functions f, g and $\lambda \in \mathbb{R}$ there holds $I_n^{Cheb}(f+g) = (I_n^{Cheb} f) + (I_n^{Cheb} g)$ as well as $I_n^{Cheb}(\lambda f) = \lambda I_n^{Cheb} f$.* ∎

Exercise 1.24 *Show that $I_n^{Cheb} f = f$ for all polynomials $f \in \mathcal{P}_n$.* Hint: *Uniqueness of polynomial interpolation, Theorem 1.1.*

We define the *Lebesgue number* Λ_n^{Cheb} by

$$\Lambda_n^{Cheb} := \max_{x \in [-1,1]} \sum_{i=0}^{n} |\ell_i^{Cheb}(x)|, \tag{1.21}$$

where ℓ_i^{Cheb} are the Lagrange interpolation polynomials for the Chebyshev points.

Theorem 1.25 (Chebyshev interpolation) *There holds:*

(i)
$$\|I_n^{Cheb} f\|_{\infty,[-1,1]} \leq \Lambda_n \|f\|_{\infty,[-1,1]} \tag{1.22}$$

(ii) There holds:
$$\|f - I_n^{Cheb} f\|_{\infty,[-1,1]} \leq (1 + \Lambda_n^{Cheb}) \min_{q \in \mathcal{P}_n} \|f - q\|_{\infty,[-1,1]}$$

(iii) $\Lambda_n^{Cheb} \leq \frac{2}{\pi} \ln(n+1) + 1$

Proof: *Proof of (i):*

$$\begin{aligned} \|I_n^{Cheb} f\|_{\infty,[-1,1]} &= \max_{x \in [-1,1]} |(I_n^{Cheb} f)(x)| = \max_{x \in [-1,1]} |\sum_{i=0}^{n} f(x_{i,n}^{Cheb}) \ell_i^{Cheb}(x)| \\ &\leq \max_{i=0,\ldots,n} |f(x_{i,n}^{Cheb})| \max_{x \in [-1,1]} |\sum_{i=0}^{n} |\ell_i^{Cheb}(x)| \leq \|f\|_{\infty,[-1,1]} \Lambda_n^{Cheb} \end{aligned}$$

Proof of (ii): We employ Exercise 1.23, 1.24 and obtain for arbitrary $q \in \mathcal{P}_n$

$$\begin{aligned} \|f - I_n^{Cheb} f\|_{\infty,[-1,1]} &\overset{\text{Exer. 1.23, 1.24}}{=} \|f - q - I_n^{Cheb}(f-q)\|_{\infty,[-1,1]} \\ &\leq \|f - q\|_{\infty,[-1,1]} + \|I_n^{Cheb}(f-q)\|_{\infty,[-1,1]} \\ &\overset{(i)}{\leq} \|f - q\|_{\infty,[-1,1]} + \Lambda_n^{Cheb} \|(f-q)\|_{\infty,[-1,1]} = (1 + \Lambda_n^{Cheb}) \|f - q\|_{\infty,[-1,1]} \end{aligned}$$

Proof of (iii): Literature. □

Remark 1.26 (Interpretation of Λ_n^{Cheb}) *1. The factor $1+\Lambda_n^{Cheb}$ measures how much worse the approximation of f by the Chebyshev interpolation is compared to the* best possible *polynomial approximation (in the norm $\|\cdot\|_{\infty,[-1,1]}$). The logarithmic growth of Λ_n^{Cheb} is very slow so that Chebyshev interpolation is typically very good: for example, for (the already rather high polynomial degree) $n = 20$ one has $\Lambda_n^{Cheb} \approx 2.9$ and thus $1 + \Lambda_{20}^{Cheb} \leq 4$.*

2. Λ_n^{Cheb} *can also be understood as an amplification factor: If, instead of the exact function values* $f(x_{i,n}^{Cheb})$*, perturbed values* $\widetilde{f}_i$ *with* $|\widetilde{f}_i - f(x_{i,n}^{Cheb})| \leq \delta$ *are employed, then the "perturbed" interpolation polynomial* $\sum_i \widetilde{f}_i \ell_i^{Cheb}$ *satisfies (Exercise!)*

$$\|(\sum_{i=0}^{n} \widetilde{f}_i \ell_i^{Cheb}) - I_n^{Cheb} f\|_{\infty,[-1,1]} \leq \Lambda_n^{Cheb} \delta.$$

In other words: Since Λ_n^{Cheb} *of Chebyshev interpolation is moderate, perturbations or errors in the values* $f(x_{i,n}^{Cheb})$ *have a rather small impact on the error in the interpolating polynomial.*

■

Chebyshev interpolation converges very rapidly for for *smooth* functions:

Exercise 1.27 *Consider the function* $f(x) = (4-x^2)^{-1}$*. Give an upper bound for* $\min_{q\in\mathcal{P}_n} \|f - q\|_{\infty,[-1,1]}$ *by selecting* q *as the Taylor polynomial of* f *about a suitable point.*
Determine the interpolating polynomials $I_n^{Cheb} f$ *for* $n = 1, \ldots, 10$*. Plot the error semilogarithmically (*`semilogy` *in* `matlab` *or* `matplotlib.pyplot.semilogy` *in* `python`*) versus* n*. To that end, approximate the error* $\|f - I_n^{Cheb} f\|_{\infty,[-1,1]}$ *by simply computing the error in 100 points that are uniformly distributed over* $[-1, 1]$*.*

1.7.3 Interpolation with uniform point distribution

For large n, the choice of the interpolation points may strongly impact the approximation quality of the interpolation process. Whereas interpolation in the Chebyshev points usually yields very good results, other systems of points may produce poor results even for functions f that may seem "harmless'. The following example illustrates this:

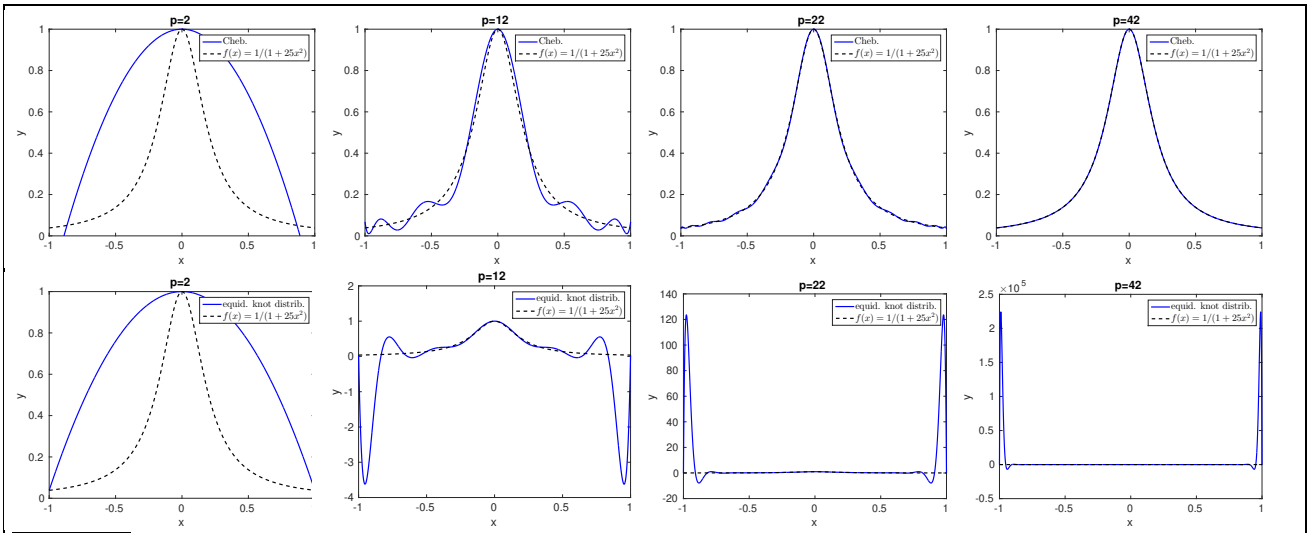
Example 1.28 (Runge example) *Consider* $f(x) = (1 + 25x^2)^{-1}$ *on the interval* $[-1, 1]$*. Fig. 1.5 shows the interpolation in Chebyshev and equidistant points. Whereas Chebyshev interpolation works well, we observe failure for the interpolation in equidistant points.* ■

The famous example of Runge of Example 1.28 shows that one should not use equidistant points for interpolation by polynomials of high degree. If the data set is based on (more or less) equidistant points, then one typically approximates by splines, i.e., *piecewise* polynomials of a *fixed* degree (e.g., $n \in \{1, 2, 3\}$) as illustrated in Exercise 1.19. An important representative of of this class is the "cubic spline" (see Section 1.8.2.)

1.8 Splines (CSE)

slide 9a

Splines are *piecewise* polynomials on a partition Δ of an interval $[a, b]$. The partition Δ is described by the knots $a = x_0 < x_1 < \cdots x_n = b$. We denote the *elements* by $I_i = (x_i, x_{i+1})$, $i = 0, \ldots, n-1$ and set $h_i := x_{i+1} - x_i$. We also set $h := \max_i h_i$.



slide 9

Figure 1.5: Interpolation of $(1 + 25x^2)^{-1}$ on $[-1, 1]$. Top row: interpolation in Chebyshev points ($n = 2$, 12, 22, 42). bottom row: Interpolation in equidistant points ($n = 2$, 12, 22, 42).

For a partition Δ (described by the knots x_i, $i = 0, \ldots, n$) and p, $r \in \mathbb{N}_0$ the spline space $S^{p,r}(\Delta)$ is defined as

$$S^{p,r}(\Delta) := \{u \in C^r([a, b]) \,|\, u|_{I_i} \in \mathcal{P}_p \quad \forall i\}. \tag{1.23}$$

Given values f_i, $i = 0, \ldots, n$, we say that $s \in S^{p,r}(\Delta)$ is an interpolating spline if

$$s(x_i) = f_i, \qquad i = 0, \ldots, n. \tag{1.24}$$

1.8.1 Piecewise linear approximation

The simplest case is $p = 1$ and $r = 0$. The interpolation problem: Given knots $a = x_0 < x_1 < \cdots < x_n = b$ and the corresponding partition,

$$\text{find } s \in S^{1,0}(\Delta) \text{ s.t.} \quad s(x_i) = f_i, \qquad i = 0, \ldots, n. \tag{1.25}$$

It is uniquely solvable and has as the solution

$$s(x) = \sum_{i=0}^{n} f_i \varphi_i(x),$$

where the φ_i continuous, piecewise linear function defined by the condition $\varphi_i(x_j) = \delta_{ij}$ (Exercise: sketch the φ_i!) Concerning error estimates, one has from a generalization of Exercise 1.19 (check this!)

$$\|f - s\|_{\infty,[a,b]} \leq Ch^2 \|f''\|_{\infty,[a,b]}.$$

1.8.2 the classical cubic spline

The classical cubic spline space is given by the choices $p = 3$ and $r = 2$. The interpolation problem is:

$$\text{find } s \in S^{3,2}(\Delta) \text{ s.t.} \quad s(x_i) = f_i, \qquad i = 0, \ldots, n. \tag{1.26}$$

Obviously, (1.26) represents a system of $n+1$ equations. We now show that $\dim S^{3,2}(\Delta) = n+3$. Hence, we will have to impose addition constraints.

Lemma 1.29 *Let Δ be a partition given by $n+1$ (distinct) knots $x_0, \ldots, x_n$. Then*

$$\dim S^{p,r}(\Delta) = n(p+1) - (n-1)(r+1) \tag{1.27}$$

Proof: Instead of a formal proof, we simply count the number of degrees of freedom/parameters needed to describe a spline: We have $\dim \mathcal{P}_p = p+1$ so that the space of *discontinuous* piecewise polynomials of degree p is $(p+1)n$. The condition of C^r continuity at the $n-1$ interior knots $x_1, \ldots, x_{n-1}$ imposes $(n-1)(r+1)$ conditions. Thus, we expect $\dim S^{p,r}(\Delta) = n(p+1) - (n-1)(r+1)$. □

For the case $p = 3$, $r = 2$, we get $\dim S^{3,2}(\Delta) = 4n - 3(n-1) = n+3$. The interpolation conditions (1.26) yield $n+1$ conditions. Hence, two more conditions have to be imposed. These two extra conditions are selected depending on the application. Typically, one of the following four choices is made:

1. *Complete/clamped spline:* The user provides two additional values f_0', $f_n' \in \mathbb{R}$ and imposes the following two additional conditions:
$$s'(x_0) = f_0', \qquad s'(x_n) = f_n'. \tag{1.28}$$
2. *Periodic spline:* one assumes $f_0 = f_n$ and imposes additionally
$$s'(x_0) = s'(x_n), \qquad s''(x_0) = s''(x_n). \tag{1.29}$$
3. *Natural spline:* one imposes
$$s''(x_0) = 0, \qquad s''(x_n) = 0. \tag{1.30}$$
4. *"not-a-knot condition":* one requires that the jump of s''' at the knots x_1 and x_{n-1} be zero:
$$\lim_{x \to x_1-} s'''(x) = \lim_{x \to x_1+} s'''(x), \qquad \lim_{x \to x_{n-1}-} s'''(x) = \lim_{x \to x_{n-1}+} s'''(x). \tag{1.31}$$

Concerning the accuracy of the interpolation method, we have:

Theorem 1.30 *Let $f \in C^4([a,b])$ and $h := \max_i h_i$. Let $f_i = f(x_i)$, $i = 0, \ldots, n$. Then the estimates*

$$\|f - s\|_{\infty,[a,b]} \le Ch^4 \|f^{(4)}\|_{\infty,[a,b]}, \qquad \|(f-s)'\|_{\infty,[a,b]} \le Ch^3 \|f^{(4)}\|_{\infty,[a,b]}$$

hold in the following cases:

(i) s is the complete spline and $f_0' = f'(x_0)$ and $f_n' = f'(x_n)$.

(ii) s is the periodic spline and f is additionally periodic, i.e., $f \in C^4(\mathbb{R})$ and $f(x+(b-a)) = f(x)$ for all $x \in \mathbb{R}$.

(iii) s is the not-a-knot spline.

In particular, in each of these cases, the spline interpolation problem is uniquely solvable.

Remark 1.31 *If only the values $f_i = f(x_i)$ are available and a good spline approximation to f is sought, then typically the not-a-knot interpolation is chosen. This is the standard choice of the* `spline` *command in* `matlab`. ∎

minimization property of cubic splines

By Theorem 1.30, the cubic spline interpolation problems with any of the above 4 extra conditions is uniquely solvable. In the three cases "complete spline", "natural spline", and "periodic spline" the interpolating spline has an optimality property:

Theorem 1.32 ("energy minimization" of cubic splines) *Let $I = [a, b]$ and Δ be a partition given by $a = x_0 < x_1 < \cdots x_n = b$. Let f_i, $i = 0, \ldots, n$, be given values.*

(i) (complete spline) Let f_0', $f_n' \in \mathbb{R}$ be additionally be given. Then the complete spline $s \in S^{3,2}(\Delta)$ satisfies

$$\|s''\|_{L^2(I)} \leq \|y''\|_{L^2(I)} \qquad \forall y \in \mathcal{C}_{complete},$$

where $\mathcal{C}_{complete}$ is given by

$$\mathcal{C}_{complete} = \{v \in C^2(I) \,|\, v(x_i) = f_i \text{ for } i = 0, \ldots, n \text{ and } v'(x_0) = f_0',\ v'(x_n) = f_n'\}.$$

(ii) (natural spline) The natural spline $s \in S^{3,2}(\Delta)$ satisfies

$$\|s''\|_{L^2(I)} \leq \|y''\|_{L^2(I)} \qquad \forall y \in \mathcal{C}_{nat},$$

where $\mathcal{C}_{nat}$ is given by

$$\mathcal{C}_{nat} = \{v \in C^2(I) \,|\, v(x_i) = f_i \text{ for } i = 0, \ldots, n \text{ and } v''(x_0) = v''(x_n) = 0\}.$$

(iii) (periodic spline) Assume $f_0 = f_n$. Then the periodic spline $s \in S^{3,2}(\Delta)$ satisfies

$$\|s''\|_{L^2(I)} \leq \|y''\|_{L^2(I)} \qquad \forall y \in \mathcal{C}_{per},$$

where $\mathcal{C}_{per}$ is given by

$$\mathcal{C}_{per} = \{v \in C^2(I) \,|\, v(x_i) = f_i \text{ for } i = 0, \ldots, n \text{ and } v'(x_0) = v'(x_n) \text{ and } v''(x_0) = v''(x_n)\}.$$

Remark 1.33 *The minimization property explains the name "spline". If one studies the deflection of an elastic "spline", then the theory of linear elasticity states that the deflection is such that the spline's elastic energy is minimized. If y describes the deflection of this spline, then in good approximation, the elastic energy of a spline is given by (ignoring physical units) $\frac{1}{2}\|y''\|^2_{L^2(I)}$. Hence, if the spline is forced to pass through points (x_i, f_i), $i = 0, \ldots, n$, then the sought deflection s is the minimizer of the problem:*

$$\begin{aligned} &\textit{minimize} \quad \frac{1}{2}\|y''\|^2_{L^2(I)} \\ &\textit{under the constraint } y(x_i) = f_i,\ i = 0, \ldots, n \textit{ (plus possibly further conditions)} \end{aligned}$$

Theorem 1.32 states that the minimizer is the interpolating cubic spline if the additional constraints are that the spline is the "complete", "natural", or "periodic" one. ∎

computation of the cubic spline

The computation of the interpolating spline can be reduced to the solution of a linear system of equations. In principle, one could make the ansatz that s is a cubic polynomial on each element $I_i = (x_i, x_{i+1})$. The interpolation conditions $s(x_i) = f_i$, the continuity conditions

$$\lim_{x \to x_i -} s^{(j)}(x) = \lim_{x \to x_i +} s^{(j)}(x), \qquad i = 1, \ldots, n-1, \quad j = 0, 1, 2$$

and the two additional conditions for complete/natural/periodic/not-a-knot splines describe a linear system of equations that can be solved.

1.8.3 remarks on splines

Exercise 1.34 *Show: for $r \geq p$, one has $S^{p,r}(\Delta) = \mathcal{P}_p$ irrespective of the partition Δ.* ∎

Remark 1.35 *For fixed, (low) r the spaces $S^{p,r}$ are much more local than then spaces $\mathcal{P}_p$. In polynomial interpolation, changing one data value f_i affects the interpolant* everywhere. *For splines (with small r), the effect is much more local, i.e., a value only affects the spline interpolant in the neighborhood of the data point. This is of interest, e.g., in the following situations:*

1. *some data values have large errors (e.g., measurement errors): then the spline is only wrong near the corresponding knot. In contrast, in polynomial interpolation, the approximation is affected everywhere.*
2. *point evaluation: if a sline is truely local (e.g., in the case $r = 0$), then the evaluation of a spline at a point x requires only the data points near x, i.e., a local calculation.*

∎

Example 1.36 *Fig. 1.6 shows polynomial interpolation and the (complete) cubic spline interpolation of the Runge example (cf. Example 1.28) on $[-1, 1]$. For $n = 8$, the $n+1 = 9$ knots are uniformly distributed in $[-1, 1]$. We observe that, while the polynomial interpolation is rather poor, the cubic spline is very good.* ∎

1.9 Remarks on Hermite interpolation

A generalization of polynomial interpolation is Hermite interpolation. Its most general form is as follows: Let $x_0, \ldots, x_n$ be $n+1$ distinct knots, and let $d_i \in \mathbb{N}_0$ be given for each i. Then, given values f_i^j, $i = 0, \ldots, n$, $j = 0, \ldots, d_i$, the *Hermite interpolant* is given by: Find $p \in \mathcal{P}_{n+\sum_{i=0}^n d_i}$ s.t.

$$p^{(j)}(x_i) = f_i^j, \quad i = 0, \ldots, n, \quad j = 0, \ldots, d_i. \tag{1.32}$$

Remark 1.37 *Hermite interpolation generalizes the polynomial interpolation problem (1.1): the choice $d_0 = d_1 = \cdots = d_n = 0$ reproduces (1.1). Another extreme case is $n = 0$ and $d_0 = N$. Then $p(x) = \sum_{j=0}^N \frac{f_0^j}{j!}(x - x_0)^j$. In particular, for $f_0^j = f^{(j)}(x_0)$, we obtain the Taylor polynomial of f of degree N.* ∎

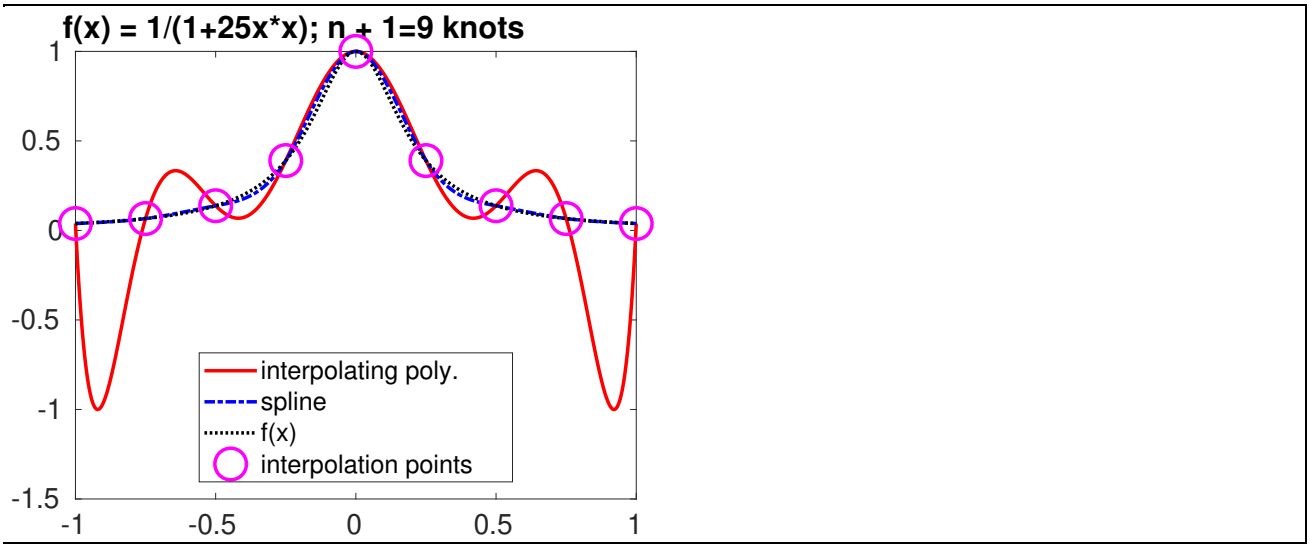


Figure 1.6: polynomial interpolation and cubic spline interpolation for uniform knot distribution; Runge example

One can show that problem (1.32) is uniquely solvable. One can also show that, if $f_i^j = f^{(j)}(x_i)$ for a sufficiently smooth f, then an error bound analogous to that of Theorem 1.15 holds true (see literature).