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 pairwise 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 pairwise 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:

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

⟦ 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

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 ($\rightarrow$ *see below)*
- $O(n)$ *to evaluate* $p(t)$ *using Alg. 1.8*

$\Rightarrow$ *Horner scheme is useful, if* p *is evaluated at "many" points* t

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

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

Remark 1.11 *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

$$f[x_0, x_1, \ldots, x_k] \approx \frac{1}{k!}f^{(k)}(x_0). \tag{1.13}$$

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 work with divided difference when some of the knots coincide.

3. *In general, for any knot sequence $x_0, \ldots, x_n$ there is an intermediate point $\xi \in (\min\{x_0, \ldots, x_k\}, \max\{x_0$ such that*

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

∎

(1.9), (1.10), and (1.11) give

$$p(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) \tag{1.14}$$
$$+ \cdots + f[x_0, x_1, \ldots, x_n](x - x_0) \cdots (x - x_{n-1}), \tag{1.15}$$

Exercise 1.12 *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.