Numerical Methods for Computational Science and Engineering

Autumn Semester 2018, Week 8

Prof. Rima Alaifari, SAM, ETH Zurich

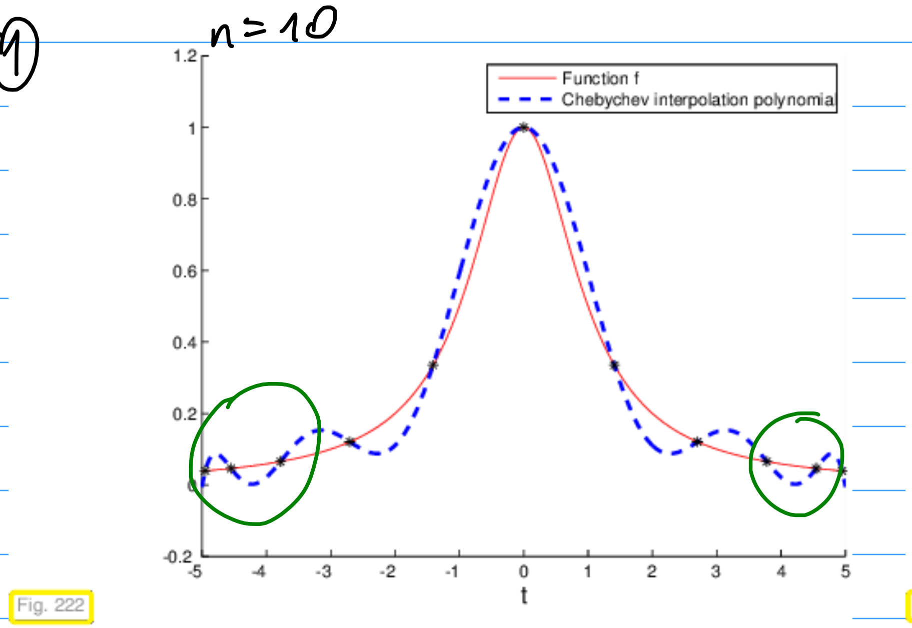
Recall: for Runge's example & Chebychev nodes:

$$\|f - I_{\mathcal{J}_n} f\|_{L^\infty([-5,5])} \leq 5^n \cdot 10$$

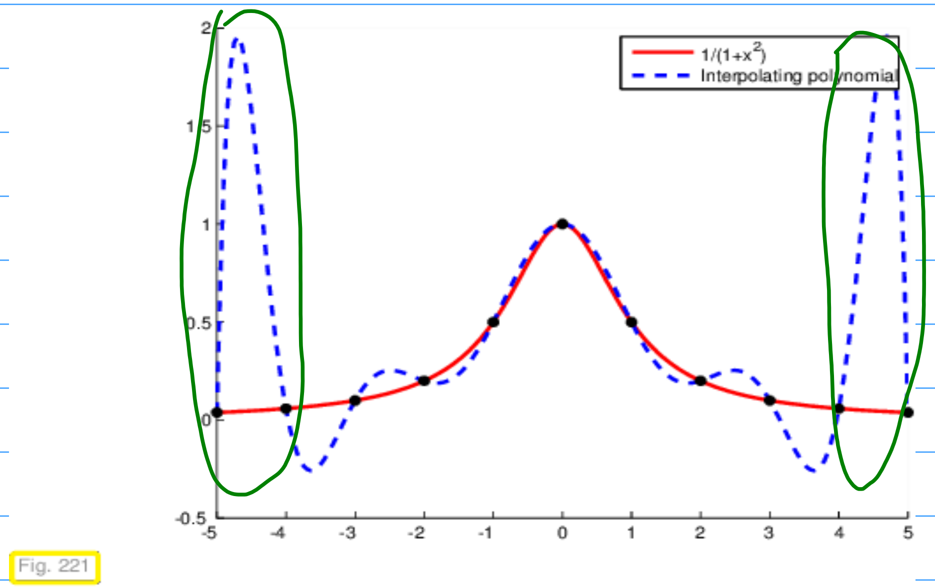
↑ still exp. growth

BUT: not necessarily divergence

(1) $n = 10$

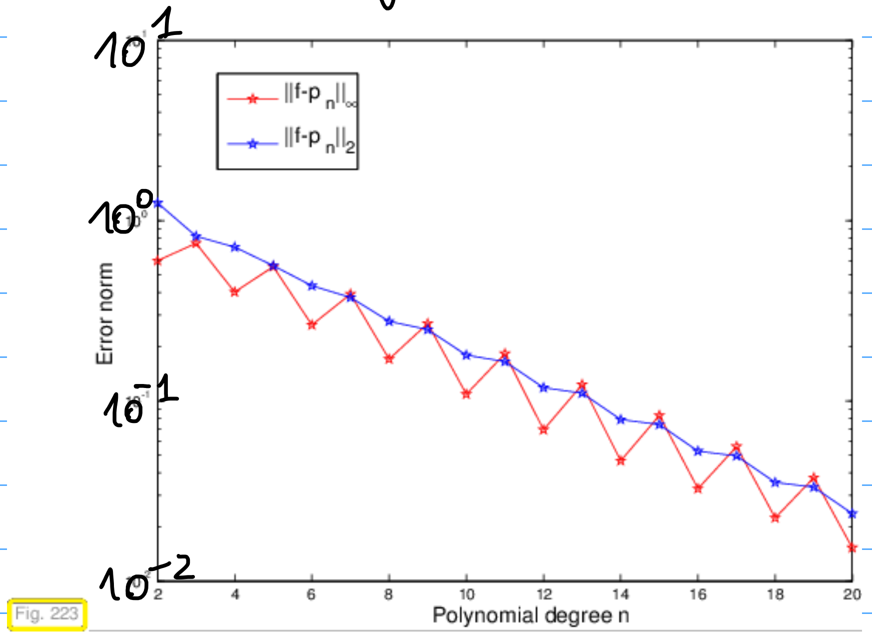


Chebychev nodes



Equidistant nodes

↑ Plot error in relation to # of nodes:



roughly: exponential convergence (mostly true when $f \in C^\infty$)

← lin-log plot

Exponential convergence: $\| f - I_{T_n} f \| = \mathcal{O}(q^n)$

$0 < q < 1$

Algebraic convergence: $\| f - I_{T_n} f \| = \mathcal{O}(n^{-p})$

$p > 0$

(asymptotic sense: $n \to \infty$)

② $f(t) = \begin{cases} 1/2\,(1+\cos(\pi t)) & |t| < 1 \\ 0 & |t| \in [1,2] \end{cases}$

$f \in C^1([-2,2])$, but not C^2

log-log-plot ↙

n=10

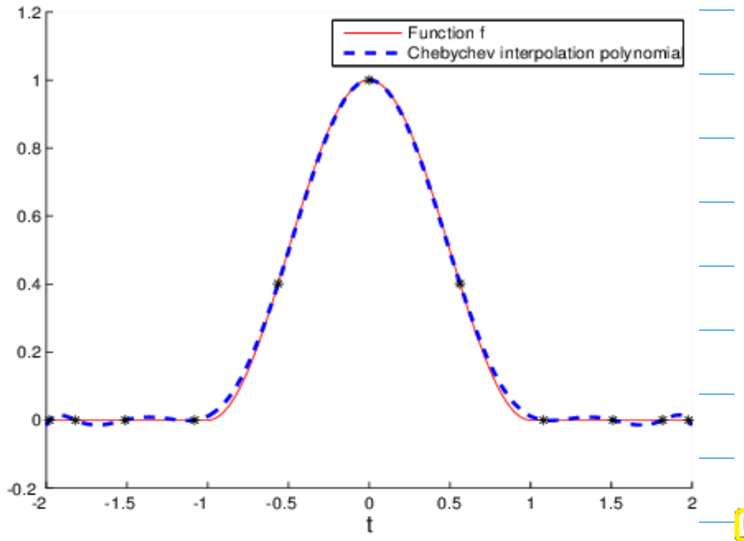


Fig. 227

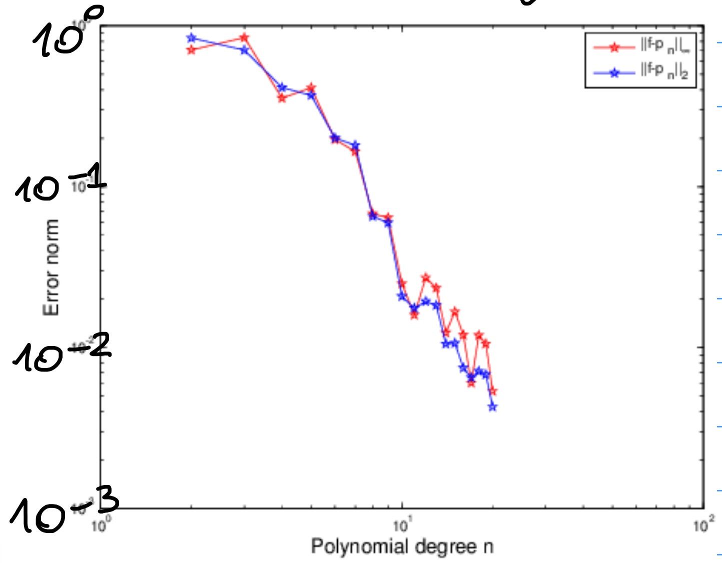


Fig. 228

Now: only alg. convergence

③ Hat function only C^0

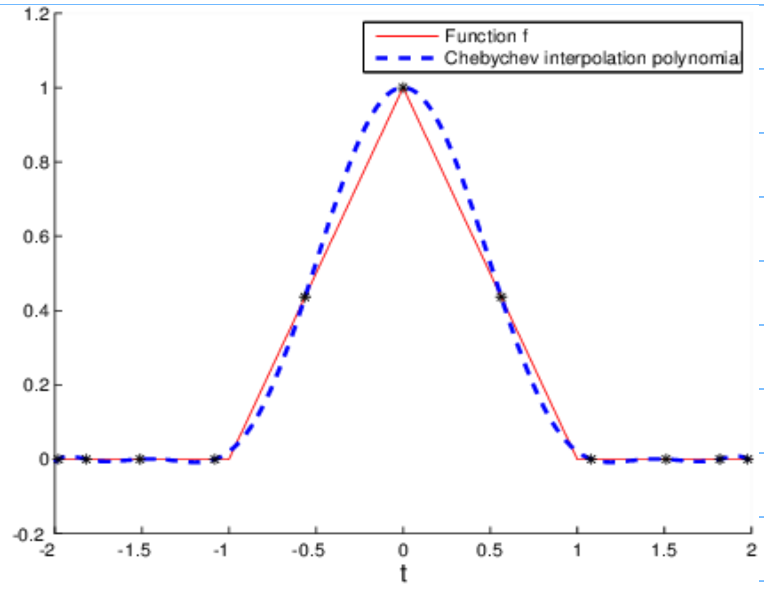


Fig. 224

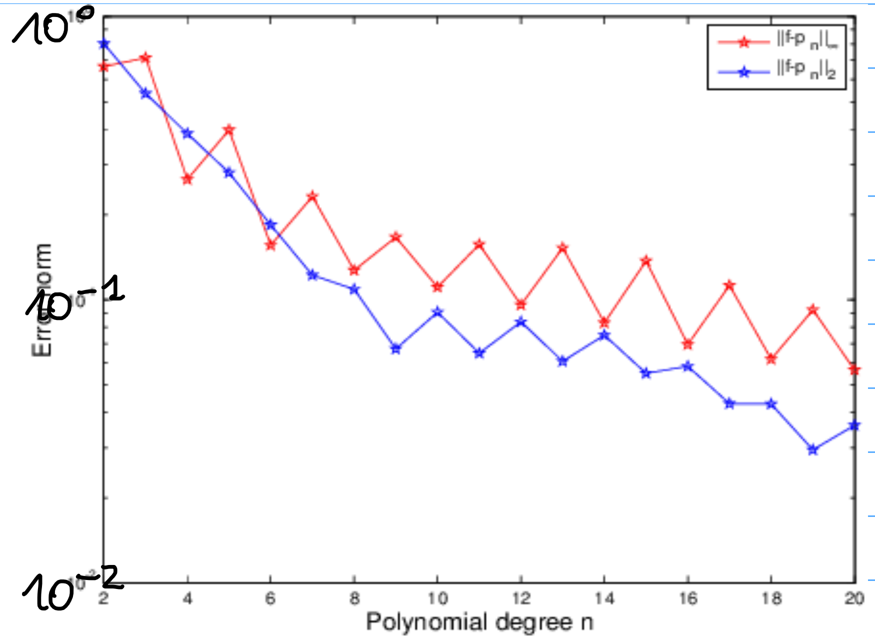
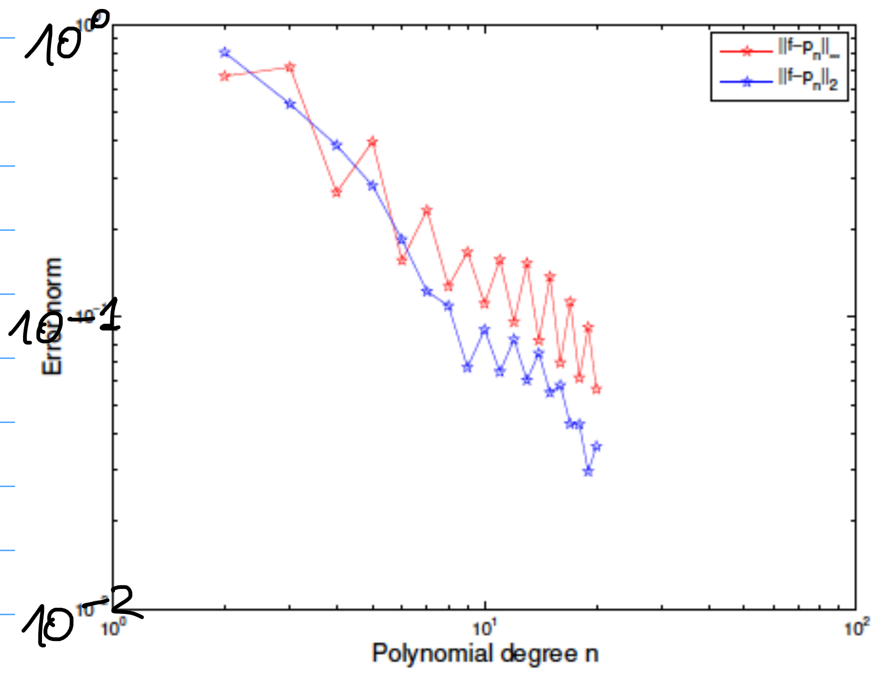


Fig. 225

lin-log-plot



alg. convergence

Implementation / computational aspects of Chebychev interpolation

The interpolant $p \in \mathcal{P}_n$ is given by

$$p(x) = \sum_{j=0}^{n} \alpha_j T_j(x) \qquad (*)$$

$T_j(x)$: basis of $\mathcal{P}_n$

↑ linear comb. of Chebychev polynomials

$\deg T_j = j \implies \{T_0, \dots, T_n\}$ basis of $\mathcal{P}_n$

Two aspects:

1. How to compute coefficients α_j efficiently?
2. Given coefficients α_j, how to evaluate $p(x)$ efficiently?

Start with (2):

Recall 3-term recursion of Chebychev polynomials:

$$T_j(x) = 2x\,T_{j-1}(x) - T_{j-2}(x) \qquad j=2,\dots,n$$

[Ex. $T_2(x) = 2x\cdot x - 1$]

Idea: Plug this into (*)

$$p(x) = \sum_{j=0}^{n-1} \alpha_j T_j(x) + \alpha_n \underbrace{T_n(x)}_{= 2x T_{n-1}(x) - T_{n-2}(x)}$$

↑ recursion

$$= \sum_{j=0}^{n-3} \alpha_j T_j(x) + \alpha_{n-2} T_{n-2}(x) + \alpha_{n-1} T_{n-1}(x) + \alpha_n \left[2x\,T_{n-1}(x) - T_{n-2}(x)\right]$$

$$= \sum_{j=0}^{n-3} \alpha_j T_j(x) + (\alpha_{n-2} - \alpha_n)\, T_{n-2}(x) + (\alpha_{n-1} + 2x\alpha_n)\, T_{n-1}(x)$$

This means:

Define new coefficients
$$\tilde{\alpha}_j := \begin{cases} \alpha_j + 2x\,\alpha_{j+1} & \text{if } j = n-1 \\ \alpha_j - \alpha_{j+2} & \text{if } j = n-2 \\ \alpha_j & \text{if } j \le n-3 \end{cases}$$

so that

$$p(x) = \sum_{j=0}^{n-1} \tilde{\alpha}_j \cdot T_j(x)$$

↑ $\deg p = n$

[Here: x is fixed. But: more generally, $\tilde{\alpha}_j$ as function in x]

So: Evaluation of T_n eliminated by modifying the coefficients.

We can proceed recursively: Clenshaw Algorithm

for $k = n, \dots, 1$: $\beta_{n+2} = \beta_{n+1} = 0$

$$\beta_k = \alpha_k + 2x\beta_{k+1} - \beta_{k+2}$$

$$\beta_0 = 2\alpha_0 + 2x\beta_1 - \beta_2$$

$$p(x) = \frac{1}{2}[\beta_0 - \beta_2] \quad (= \alpha_0 + x\beta_1 - \beta_2)$$

① Computation of coefficients α_j :

$$p(t_k) = f(t_k) = y_k \quad \text{interpolating conditions}$$

$k = 0, \dots, n$

$$t_k = \cos\left(\frac{2k+1}{2(n+1)}\pi\right)$$

$$s_k := \frac{2k+1}{4(n+1)} \quad \Rightarrow \quad t_k = \cos(2\pi s_k)$$

$$p(t_k) = p(\cos(2\pi s_k))$$

$$q(s) := p(\cos(2\pi s)) = \sum_{j=0}^{n} \alpha_j T_j(\cos 2\pi s)$$

↑ Chebychev interpolation

$$T_j(t) = \cos(j \cdot \arccos t) \qquad s \in [0, \tfrac{1}{2}]$$

$$T_j(\cos 2\pi s) = \cos(j \cdot 2\pi s)$$

↑ $t = \cos 2\pi s$

$$q(s) := \underline{p(\cos 2\pi s)} = \sum_{j=0}^{n} \alpha_j \cos(j 2\pi s)$$

$$\cos z = \frac{e^{iz} + e^{-iz}}{2}$$

⇝ "bring in complex exponentials to come closer to the FFT"

$$p(\cos 2\pi s) = \sum_{j=0}^{n} \frac{1}{2} \alpha_j \left[\exp(ij 2\pi s) + \exp(-ij 2\pi s) \right]$$

in total: range over $-n, \dots, n$

$$= \sum_{j=-n}^{n+1} \beta_j \exp(-2\pi i j s)$$

$$\beta_j := \begin{cases} \alpha_0 & j = 0 \\ \frac{1}{2}\alpha_j & j = 1, \dots, n \\ \frac{1}{2}\alpha_{-j} & j = -n, \dots, -1 \\ 0 & j = n+1 \end{cases}$$

↑ at the moment artificial

Goal: $(2n+2) \times (2n+2)$ system

Right now: points $s_0, \dots, s_n$

Expand by exploiting symmetry of q:

$$q(1-s) = p(\cos(2\pi(1-s))) = p(\cos 2\pi s) = q(s)$$

$$\cos(2\pi - 2\pi s) = \cos(-2\pi s) = \cos(2\pi s)$$

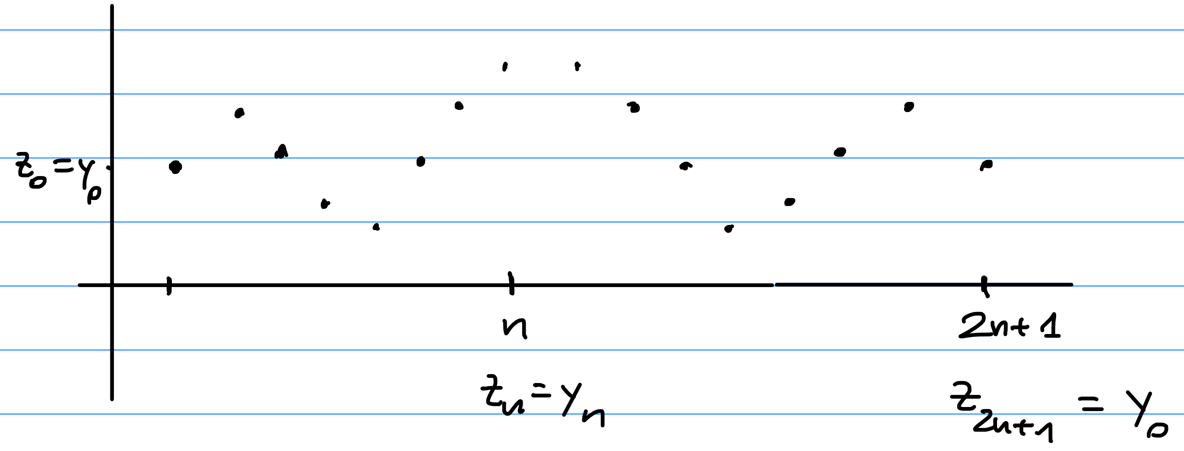
$$q(s_k) = q(1-s_k) = q(s_{2n+1-k}) = y_{2n+1-k}$$

↑ $s_k = \frac{2k+1}{4(n+1)}$

$$1 - s_k = s_{2n+1-k}$$

$$q\left(\frac{2k+1}{4(n+1)}\right) = \begin{cases} y_k & k=0,\dots,n \\ y_{2n+1-k} & k=n+1,\dots,2n+1 \end{cases}$$

$$=: z_k$$



Now: Find β_j's using FFT:

LSE for β_j:

$$\underbrace{q\left(\frac{2k+1}{4(n+1)}\right)} = z_k \qquad k=0,\dots,2n+1$$

Use $q(s) = \sum_{j=-n}^{n+1} \beta_j \exp(-2\pi i j s)$

$$\sum_{j=-n}^{n+1} \beta_j \exp\left(-2\pi i j\left(\frac{k}{2(n+1)} + \frac{1}{4(n+1)}\right)\right) = z_k \quad (= q(s_k))$$

$$\sum_{j=-n}^{n+1} \beta_j \exp\left(-\frac{\pi i j k}{n+1}\right) \exp\left(-\frac{\pi i j}{2(n+1)}\right) = z_k$$

$\uparrow$ $e^{x+y} = e^x e^y$

$$\sum_{j=0}^{2n+1} \beta_{j-n} \underbrace{\exp\left(-\frac{\pi i (j-n) k}{n+1}\right)} \exp\left(-\frac{\pi i (j-n)}{2(n+1)}\right) = z_k$$

$\uparrow$ index shift

$$\sum_{j=0}^{2n+1} \beta_{j-n} \underbrace{\exp\left(-\frac{\pi i j k}{n+1}\right)}_{\omega_{2(n+1)}^{jk}} \cdot \exp\left(-\frac{\pi i (j-n)}{2(n+1)}\right) = z_k \exp\left(-\frac{\pi i k n}{n+1}\right)$$

$$\Rightarrow \sum_{j=0}^{2n+1} \beta_{j-n} \exp\left(-\frac{\pi i (j-n)}{2(n+1)}\right) \omega_{2(n+1)}^{jk} = \exp\left(-\frac{\pi i n k}{n+1}\right) z_k$$

$$c := \left[\beta_{j-n} \exp\left(-\frac{\pi i (j-n)}{2(n+1)}\right)\right]_{j=0}^{2n+1}$$

$$b := \left[z_k \exp\left(-\frac{\pi i n k}{n+1}\right)\right]_{k=0}^{2n+1}$$

↑ RHS vector

$$\Rightarrow \boxed{F_{2n+2}\, c = b}$$

↑ $(2n+2)\times(2n+2)$ Fourier matrix

Use inverse FFT : $O(n \log n)$

to recover vector c

$\Rightarrow$ this gives β_j's $\Rightarrow$ gives α_j's

Splines

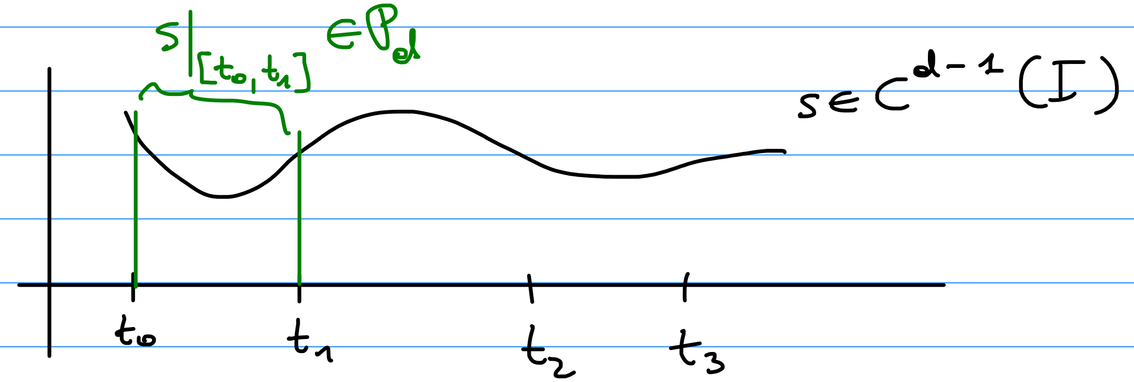
Piecewise polynomial interpolation

divide the full interval I into subintervals $[t_{i-1}, t_i]$

- on each $[t_{i-1}, t_i]$: polynomial of degree d
- matching of first $d-1$ derivatives at nodes t_i.

Definition 5.2.1 (Spline space). Given an interval $I := [a, b] \subset \mathbb{R}$ and a *knot set/mesh* $\mathcal{M} := \{a = t_0 < t_1 < \ldots < t_{n-1} < t_n = b\}$, the vector space $\mathcal{S}_{d,\mathcal{M}}$ of the *spline functions* of degree d (or order $d+1$) is defined by

$$\mathcal{S}_{d,\mathcal{M}} := \{s \in C^{d-1}(I) : s_j := s\,|_{[t_{j-1},t_j]} \in \mathcal{P}_d \ \forall j = 1, \ldots, n\}\,.$$



Spline space are mapped onto each other by differentiation and integration:

$$s \in \mathcal{S}_{d,\mathcal{M}} \quad \Rightarrow \quad s' \in \mathcal{S}_{d-1,\mathcal{M}} \quad \text{and} \quad \int_a^t s(\tau)\,\mathrm{d}\tau \in \mathcal{S}_{d+1,\mathcal{M}}\,.$$

Spline spaces of the lowest degrees:

- $d = 0$: $\mathcal{M}$-piecewise constant *discontinuous* functions.
- $d = 1$: $\mathcal{M}$-piecewise linear *continuous* functions.
- $d = 2$: *continuously differentiable* $\mathcal{M}$-piecewise quadratic functions.

Cubic splines: $d=3$ on each subinterval:

- polynomials of degree 3
- overall C^2 function

Dimension of $\mathcal{S}_{d,\mathcal{M}}$?

\# of intervals: n

degrees of freedom on each interval: $d+1$

constraints: at each interior point: d constraints

\# of int. points: $n-1$

$$\dim \mathcal{S}_{d,\mathcal{M}} = n(d+1) - (n-1)\cdot d = \underline{\underline{n+d}}$$

Cubic spline interpolation

$$\mathcal{S}_{3,\mathcal{M}} = \left\{ s \in C^2(I) : s_j := s\big|_{[t_{j-1},t_j]} \in \mathcal{P}_3, \ \forall j = 1,\dots,n \right\}$$

What does $s \in C^2(I)$ imply?

$$s_j(t_j) = s_{j+1}(t_j)$$

$$s_j'(t_j) = s_{j+1}'(t_j)$$

$$s_j''(t_j) = s_{j+1}''(t_j)$$

$$s_j = s\big|_{[t_{j-1}, t_j]} \in \mathcal{P}_3 \quad \forall j = 1, \dots, n$$

$\Rightarrow$ This allows to write

$$s_j(t) = a_j + b_j t + c_j t^2 + d_j t^3 \qquad (*)$$

In total: Interpolation task is to determine

$4n$ coefficients

$\Rightarrow$ We need $4n$ conditions

① Interpolating conditions:

$$\left.\begin{aligned} s_j(t_{j-1}) &= y_{j-1} \\ s_j(t_j) &= y_j \end{aligned}\right. \qquad \forall j = 1, \dots, n$$

In the form $(*)$:

$$\left.\begin{aligned} a_j + b_j t_{j-1} + c_j t_{j-1}^2 + d_j t_{j-1}^3 &= y_{j-1} \\ a_j + b_j t_j + c_j t_j^2 + d_j t_j^3 &= y_j \end{aligned}\right\} \; 2n \text{ conditions}$$

② Matching the first derivatives:

$$s_j'(t_j) = s_{j+1}'(t_j) \quad \forall j = 1,\dots,n-1$$

$$s_j'(t_j) = b_j + 2c_j t_j + 3d_j t_j^2$$
$$= b_{j+1} + 2c_{j+1} t_j + 3d_{j+1} t_j^2 = s_{j+1}'(t_j)$$

} $n-1$ conditions

③ Matching the 2^{nd} derivatives:

$$s_j''(t_j) = s_{j+1}''(t_j) \quad \forall j = 1,\dots,n-1$$

$$s_j''(t_j) = 2c_j + 6d_j t_j = 2c_{j+1} + 6d_{j+1} t_j = s_{j+1}''(t_j)$$

→ $n-1$ conditions

So far: $4n-2$ conditions

We need $4n$ conditions

Need 2 more conditions

Typical choice: natural/simple BCs:

$$s_1''(t_0) = 0$$

$$s_n''(t_n) = 0$$

Goal: Formulate LSE for the coefficients

$$\{a_j, b_j, c_j, d_j\}_{j=1,\dots,n}$$

Implementation of spline interpolation:

$\forall\, j = 1, \dots, n$

$$s_j(t) = \tilde{a}_j + \tilde{b}_j (t - t_{j-1}) + \tilde{c}_j (t - t_{j-1})^2 + \tilde{d}_j (t - t_{j-1})^3$$

with $s_j(t_j) = y_j = s_{j+1}(t_j)$

$s_j''(t_j) = \sigma_j = s_{j+1}''(t_j)$

we need to determine σ_j's!

$\tilde{a}_j = s_j(t_{j-1}) = y_{j-1}$ Define $h_j := t_j - t_{j-1}$

$$\left.\begin{aligned} \tilde{b}_j &= \frac{y_j - y_{j-1}}{h_j} - \frac{h_j (2\sigma_{j-1} + \sigma_j)}{6} \\ \tilde{c}_j &= \frac{\sigma_{j-1}}{2} \;\checkmark \qquad \tilde{d}_j = \frac{\sigma_j - \sigma_{j-1}}{6 h_j} \;\checkmark \end{aligned}\right\} (**)$$

Check $(**)$:

$$s_j''(t_{j-1}) = 2\tilde{c}_j = \sigma_{j-1}$$

$$s_j'(t) = \tilde{b}_j + 2\tilde{c}_j (t - t_{j-1}) + 3\tilde{d}_j (t - t_{j-1})^2 \quad \Leftarrow$$

$$s_j''(t) = 2\tilde{c}_j + 6\tilde{d}_j (t - t_{j-1})$$

$$s_j''(t_j) = \underbrace{2\tilde{c}_j}_{=\sigma_{j-1}} + 6\tilde{d}_j (t_j - t_{j-1}) = \sigma_j$$

$$\Rightarrow \quad \tilde{d}_j = \frac{\sigma_j - \sigma_{j-1}}{6 h_j}$$

Need to verify eqn for $\tilde{b}_j$ ⟶ use matching of <u>first</u> derivatives

$$s_j'(t_j) = \tilde{b}_j + 2\tilde{c}_j h_j + 3\tilde{d}_j h_j^2$$

$$s_{j+1}'(t_j) = \tilde{b}_{j+1}$$

match!

$$\Rightarrow \quad \tilde{b}_j + 2\tilde{c}_j h_j + 3\tilde{d}_j h_j^2 = \tilde{b}_{j+1}$$

$$\frac{y_j - y_{j-1}}{h_j} - \frac{h_j(2\sigma_{j-1} + \sigma_j)}{6} + \sigma_{j-1} h_j + \frac{\sigma_j - \sigma_{j-1}}{2} h_j = \tilde{b}_{j+1}$$

$$\tilde{b}_{j+1} = \frac{y_{j+1} - y_j}{h_{j+1}} - \frac{h_{j+1}(2\sigma_j + \sigma_{j+1})}{6}$$

! Postpone verification of $\tilde{b}_j$!

$$\sigma_{j-1}\left(-\frac{h_j}{3} - \frac{h_j}{2} + h_j\right) + \sigma_j\left(-\frac{h_j}{6} + \frac{h_j}{2} - \frac{h_{j+1}}{3}\right) + \sigma_{j+1}\frac{h_{j+1}}{6} = \underbrace{\frac{y_{j+1} - y_j}{h_{j+1}} - \frac{y_j - y_{j-1}}{h_j}}_{=: r_j}$$

Still need to verify

$$\Rightarrow \sigma_{j-1}\frac{h_j}{6} + \sigma_j \frac{h_j + h_{j+1}}{3} + \sigma_{j+1}\frac{h_{j+1}}{6} = r_j$$

$$j = 1, \dots, n-1$$

Natural BCs: $\sigma_0 = 0$

$\sigma_n = 0$

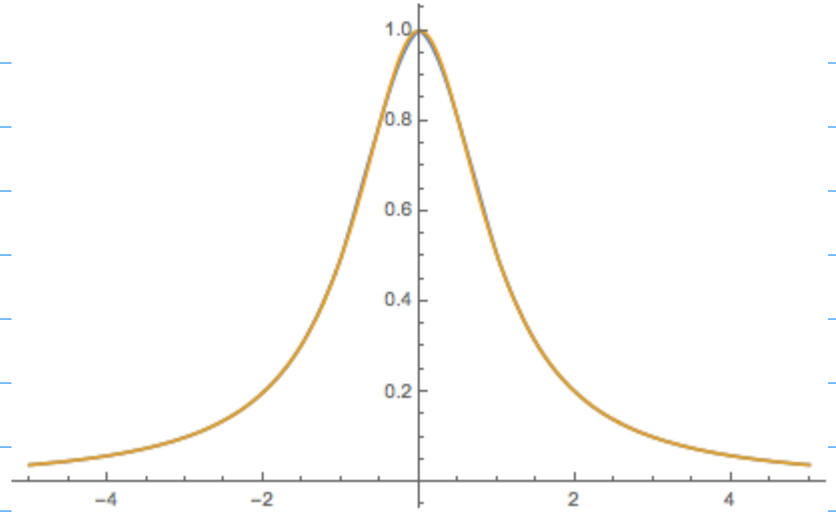
$$\begin{bmatrix} \frac{h_1+h_2}{3} & \frac{h_2}{6} & & & 0 \\ \frac{h_2}{6} & \frac{h_2+h_3}{3} & h_3/6 & & \\ & \frac{h_3}{6} & \ddots & \ddots & \\ & & \ddots & \ddots & \frac{h_{n-1}}{6} \\ 0 & & & \frac{h_{n-1}}{6} & \frac{h_{n-1}+h_n}{3} \end{bmatrix} \begin{bmatrix} \sigma_1 \\ \vdots \\ \vdots \\ \sigma_{n-1} \end{bmatrix} = \begin{bmatrix} r_1 \\ \vdots \\ \vdots \\ r_{n-1} \end{bmatrix}$$

$(n-1)\times(n-1)$ system to solve for $[\tilde{a}_1, \dots, \tilde{a}_{n-1}]^T$

From this: (**) gives the coefficients

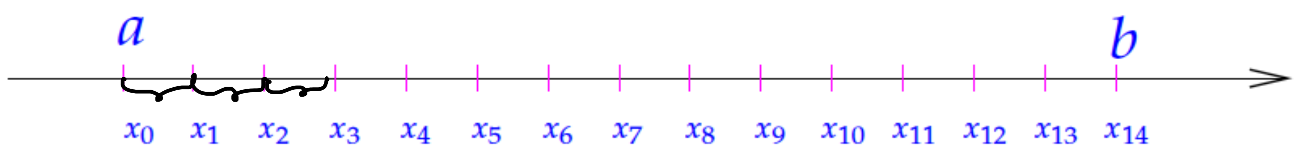
$$\{\tilde{a}_j, \tilde{b}_j, \tilde{c}_j, \tilde{d}_j\}_{j=1,\dots,n}$$

→ solved interp. problem



($f(t) = \frac{1}{1+t^2}$ on $[-5,5]$ with cubic splines $n=10$)

Piecewise polynomial Lagrange interp.



Terminology:

- $x_j \triangleq$ *nodes* of the mesh $\mathcal{M}$.
- $[x_{j-1}, x_j[\triangleq$ *intervals/cells* of the mesh.
- $h_{\mathcal{M}} := \max_j |x_j - x_{j-1}| \triangleq$ *mesh width*.
- If $x_j = a + jh$, we call the mesh *equidistant* or uniform with meshwidth $h > 0$.

General local Lagrange interpolation on a mesh

❶ Choose a *local degree* $n_j \in \mathbb{N}_0$ for each cell of the mesh, $j = 1, \dots, m$.

❷ Choose a set of *local* interpolation nodes

$$\mathcal{T}^j := \{t_0^j, \dots, t_{n_j}^j\} \subset I_j := [x_{j-1}, x_j], \quad j = 1, \dots, m,$$

for each mesh cell/grid interval I_j.

❸ Define a *piecewise polynomial* interpolant $s : [x_0, x_m] \to \mathbb{K}$:

$$s_j := s_{|I_j} \in \mathcal{P}_{n_j} \quad \text{and} \quad s_j(t_i^j) = f(t_i^j) \quad i = 0, \dots, n_j, \quad j = 1, \dots, m. \tag{6.40}$$

Owing to theorem 5.1.2, s_j is well defined.

If $t^j_{n_j} = t^{j+1}_0 \quad \forall j = 1, \dots, m-1$

$\Rightarrow s \in C^0([x_0, x_m])$

Recall:

For Chebychev interpolation

$$\|f - I_T f\|_{L^\infty(I)} \leq \frac{2^{-2n-1}}{(n+1)!} |I|^{n+1} \|f^{(n+1)}\|_{L^\infty(I)}$$

↑ length of interval enters in estimate

→ piecewise interpolation: each subinterval will be smaller

Note: interpolation error will decrease in mesh width h_M (algebraically)

(same is true for cubic splines)

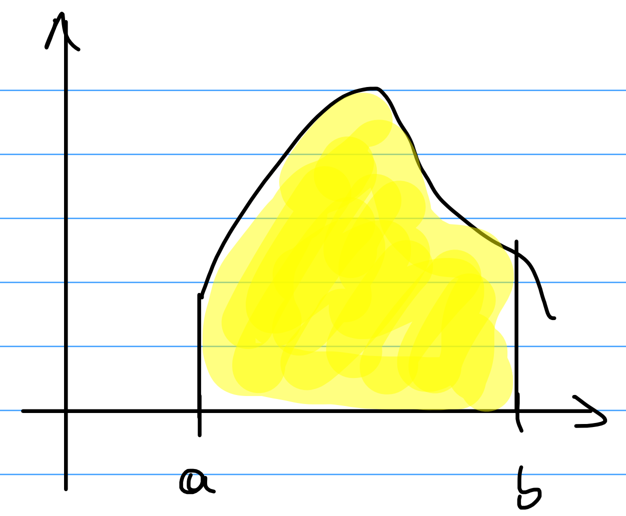
Difference to cubic splines:
- no global smoothness
- no need to solve a LSE

6. Numerical Quadrature

Goal: Approximate numerical evaluation of integrals

more precisely:

Find approximation to $\int_a^b f(t)\,dt$ using only point values of f.



Motivation:

1. We only have samples / point values of f
2. We have $f(t)$ but $\int f(t)\,dt$ is expensive/difficult
3. We may have $\int f(t)\,dt$ but numerical integration is easier than evaluating it.

Important application: FEM

6.1 Quadrature Formula

Q_n is an n-point quadrature formula (QF) on $[a,b]$ if

$$Q_n(f) \approx \int_a^b f(t)\,dt$$

and

Q_n is a weighted sum of point values

$$Q_n(f) = \sum_{j=1}^{n} w_j f(c_j)$$

w_j: quadrature weights; c_j: quadrature nodes

Cost of evaluation of $Q_n(f)$: • n point eval. of f

• n multiplications/ additions

Once we have QF for the interval $[-1,1]$

$\rightarrow$ can easily formulate the QF on arbitrary interval $[a,b]$

Idea: $$\int_a^b f(t)\,dt = \int_{-1}^{1} f(\Phi(t))\,\Phi'(\tau)\,d\tau$$

$$\Phi(\tau) = \frac{1}{2}(1-\tau)a + \frac{1}{2}(1+\tau)b$$