

Levinson algorithm

(recursive, u_{n+1} not used!)

Linear recursion:

Computational cost $\sim (n-k)$ on level $k, k =$

$0, \dots, n-1$

➤ Asymptotic complexity $O(n^2)$



```

MATLAB-CODE Levinson algorithm
function [x,y] = levinson(u,b)
k = length(u)-1;
if (k == 0)
    x=b(1); y = u(1); return;
end
[xk,yk] = levinson(u(1:k),b(1:k));
sigma = 1-dot(u(1:k),yk);
t= (b(k+1)-dot(u(k:-1:1),xk))/sigma;
x= [ xk-t*yk(k:-1:1);t];
s= (u(k+1)-dot(u(k:-1:1),yk))/sigma;
y= [yk-s*yk(k:-1:1); s];

```

Remark 6.5.3 (Fast Toeplitz solvers).

FFT-based algorithms for solving $\mathbf{T}\mathbf{x} = \mathbf{b}$ with asymptotic complexity $O(n \log^3 n)$ [49] !



[10, Sect. 8.5]: Very detailed and elementary presentation, but the discrete Fourier transform through trigonometric interpolation, which is not covered in this chapter. Hardly addresses discrete convolution.

[29, Ch. IX] presents the topic from a mathematical point of few stressing approximation and trigonometric interpolation. Good reference for algorithms for circulant and Toeplitz matrices.

[47, Ch. 10] also discusses the discrete Fourier transform with emphasis on interpolation and (least squares) approximation. The presentation of signal processing differs from that of the course.

There is a vast number of books and survey papers dedicated to discrete Fourier transforms, see, for instance, [15, 6]. Issues and technical details way beyond the scope of the course are treated there.

Part II

Interpolation and Approximation

Introduction

Distinguish two fundamental concepts:

(I) **data interpolation** (point interpolation, also includes CAD applications):

Given: data points $(\mathbf{x}_i, \mathbf{y}_i), i = 1, \dots, m, \mathbf{x}_i \in D \subset \mathbb{R}^n, \mathbf{y}_i \in \mathbb{R}^d$

Goal: reconstruction of a (continuous) function $\mathbf{f} : D \mapsto \mathbb{R}^d$ satisfying **interpolation conditions**

$$f(\mathbf{x}_i) = \mathbf{y}_i, \quad i = 1, \dots, m$$

Additional requirements: • smoothness of \mathbf{f} , e.g. $\mathbf{f} \in C^1$, etc.

• shape of \mathbf{f} (positivity, monotonicity, convexity)

(II) **function approximation**:

Given: function $\mathbf{f} : D \subset \mathbb{R}^n \mapsto \mathbb{R}^d$ (often in procedural form $y = \text{feval}(\mathbf{x})$)

Goal: Find a "simple" (*) function $\tilde{\mathbf{f}} : D \mapsto \mathbb{R}^d$ such that the difference $\mathbf{f} - \tilde{\mathbf{f}}$ is "small" (♣)

(*): "simple" \sim described by small amount of information, easy to evaluate (e.g. polynomial or piecewise polynomial $\tilde{\mathbf{f}}$)

“small” $\sim \left\| \mathbf{f} - \tilde{\mathbf{f}} \right\|$ small for some norm $\|\cdot\|$ on space $C(D)$ of continuous functions, e.g. L^2 -norm
 $\|g\|_2^2 := \int_D |g(x)|^2 dx$, maximum norm $\|g\|_\infty := \max_{x \in D} |g(x)|$

Example 6.0.1 (Taylor approximation).

$$f \in C^k(I), \quad I \text{ interval}, \quad k \in \mathbb{N}, \quad T_k(t) := \frac{f^{(k)}(t_0)}{k!} (t - t_0)^k, \quad t_0 \in I.$$

The Taylor polynomial T_k of degree k approximates f in a neighbourhood $J \subset I$ of t_0 (J can be small!). The Taylor approximation is easy and direct but inefficient: a polynomial of lower degree gives the same accuracy.

Another technique:

Approximation by interpolation

$$f \xrightarrow{\text{sampling}} (x_i, y_i := f(x_i))_{i=1}^m \xrightarrow{\text{interpolation}} \tilde{f}: \tilde{f}(x_i) = y_i.$$

↑
free choice of nodes x_i

Remark 6.0.2 (Interpolation and approximation: enabling technologies).

Approximation and interpolation are useful for several numerical tasks, like integration, differentiation and computation of the solutions of differential equations, as well as for computer graphics: smooth curves and surfaces.

▶ this is a “foundations” part of the course

Remark 6.0.3 (Function representation).

! General function $f : D \subset \mathbb{R} \mapsto \mathbb{K}$, D interval, contains an “infinite amount of information”.

? How to represent f on a computer?

➔ Idea: parametrization, a finite number of parameters $\alpha_1, \dots, \alpha_n$, $n \in \mathbb{N}$, characterizes f .

Special case: Representation with finite linear combination of basis functions

$b_j : D \subset \mathbb{R} \mapsto \mathbb{K}$, $j = 1, \dots, n$:

$$f = \sum_{j=1}^n \alpha_j b_j, \quad \alpha_j \in \mathbb{K}.$$

➔ $f \in$ finite dimensional function space $V_n := \text{Span}\{b_1, \dots, b_n\}$.

Numerical Methods 401-0654

Numerical Methods 401-0654

V. Gradinaru
D-ITET,
D-MATL

V. Gradinaru
D-ITET,
D-MATL

6.0
p. 369

6.0
p. 37

Numerical Methods 401-0654

Numerical Methods 401-0654

7

Polynomial Interpolation

7.1 Polynomials

Notation: Vector space of the polynomials of degree $\leq k$, $k \in \mathbb{N}$:

$$\mathcal{P}_k := \{t \mapsto \alpha_k t^k + \alpha_{k-1} t^{k-1} + \dots + \alpha_1 t + \alpha_0, \alpha_j \in \mathbb{K}\}. \quad (7.1.1)$$

Terminology: the functions $t \mapsto t^k$, $k \in \mathbb{N}_0$, are called monomials

$$t \mapsto \alpha_k t^k + \alpha_{k-1} t^{k-1} + \dots + \alpha_0 = \text{monomial representation of a polynomial.}$$

Obvious: \mathcal{P}_k is a vector space. What is its dimension?

6.0
p. 370

7.1
p. 37

V. Gradinaru
D-ITET,
D-MATL

V. Gradinaru
D-ITET,
D-MATL

Theorem 7.1.1 (Dimension of space of polynomials).

$$\dim \mathcal{P}_k = k + 1 \text{ and } \mathcal{P}_k \subset C^\infty(\mathbb{R}).$$

Proof. Dimension formula by linear independence of monomials.

Why are polynomials important in computational mathematics ?

- Easy to compute, integrate and differentiate
- Vector space & algebra
- Analysis: Taylor polynomials & power series

Remark 7.1.1 (Polynomials in Matlab).

MATLAB: $\alpha_k t^k + \alpha_{k-1} t^{k-1} + \dots + \alpha_0$ → Vector $(\alpha_k, \alpha_{k-1}, \dots, \alpha_0)$ (ordered!).

Remark 7.1.2 (Horner scheme).

Evaluation of a polynomial in monomial representation: Horner scheme

$$p(t) = (t \cdots t(\alpha_n t + \alpha_{n-1}) + \alpha_{n-2}) + \dots + \alpha_1 + \alpha_0. \quad (7.1.2)$$

Code 7.1.3: Horner scheme, polynomial in MATLAB format

```
function y = polyval(p,x)
y = p(1); for i=2:length(p), y = x*y+p(i); end
```

Asymptotic complexity: $O(n)$

Use: MATLAB "built-in"-function `polyval(p,x)`;

7.2 Polynomial Interpolation: Theory

Goal: (re-)construction of a function from pairs of values (fit).

Lagrange polynomial interpolation problem

Given the **simple nodes** $t_0, \dots, t_n, n \in \mathbb{N}, -\infty < t_0 < t_1 < \dots < t_n < \infty$ and the values $y_0, \dots, y_n \in \mathbb{K}$ compute $p \in \mathcal{P}_n$ such that

$$p(t_j) = y_j \text{ for } j = 0, \dots, n. \quad (7.2.1)$$

General polynomial interpolation problem

Given the **(eventually multiple) nodes** $t_0, \dots, t_n, n \in \mathbb{N}, -\infty < t_0 \leq t_1 \leq \dots \leq t_n < \infty$ and the values $y_0, \dots, y_n \in \mathbb{K}$ compute $p \in \mathcal{P}_n$ such that

$$\frac{d^k}{dt^k} p(t_j) = y_j \text{ for } k = 0, \dots, l_j \text{ and } j = 0, \dots, n, \quad (7.2.2)$$

where $l_j := \max\{i - i' : t_j = t_i = t_{i'}, i, i' = 0, \dots, n\}$ is the multiplicity of the nodes t_j .

When all the multiplicities are equal to 2: **Hermite interpolation** (or osculatory interpolation)

$t_0 = t_1 < t_2 = t_3 < \dots < t_{n-1} = t_n$ → $p(t_{2j}) = y_{2j}, p'(t_{2j}) = y_{2j+1},$ (**double nodes**).

7.2.1 Lagrange polynomials

For nodes $t_0 < t_1 < \dots < t_n$ (→ Lagrange interpolation) consider

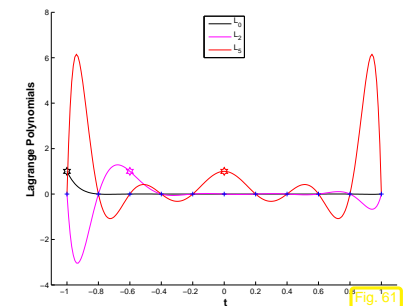
$$\text{Lagrange polynomials } L_i(t) := \prod_{\substack{j=0 \\ j \neq i}}^n \frac{t - t_j}{t_i - t_j}, \quad i = 0, \dots, n. \quad (7.2.3)$$

→ $L_i \in \mathcal{P}_n$ and $L_i(t_j) = \delta_{ij}$

Example 7.2.1. Lagrange polynomials for uniformly spaced nodes

$$\mathcal{T} := \left\{ t_j = -1 + \frac{2}{n} j \right\}, \quad j = 0, \dots, n.$$

Plot $n = 10, j = 0, 2, 5$ →



Matlab function: `lagrangeplot.m`

The Lagrange interpolation polynomial p for data $(t_i, y_i)_{i=0}^n$ has the representation:

$$p(t) = \sum_{i=0}^n y_i L_i(t), \quad \Rightarrow \quad p \in \mathcal{P}_n \text{ and } p(t_i) = y_i. \quad (7.2.4)$$

Theorem 7.2.1 (Existence & uniqueness of Lagrange interpolation polynomial).
 The general polynomial interpolation problem (7.2.1) admits a unique solution $p \in \mathcal{P}_n$.

Proof. Consider the linear evaluation operator

$$\text{eval}_{\mathcal{T}} : \begin{cases} \mathcal{P}_n \mapsto \mathbb{R}^{n+1} \\ p \mapsto (p(t_i))_{i=0}^n, \end{cases}$$

which maps between finite-dimensional vector spaces of the same dimension, see Thm. 7.1.1.

Representation (7.2.4) \Rightarrow existence of interpolating polynomial
 \Rightarrow $\text{eval}_{\mathcal{T}}$ is **surjective**

Known from linear algebra: for a linear mapping $T : V \mapsto W$ between finite-dimensional vector spaces with $\dim V = \dim W$ holds the equivalence

$$T \text{ surjective} \Leftrightarrow T \text{ bijective} \Leftrightarrow T \text{ injective.}$$

Applying this equivalence to $\text{eval}_{\mathcal{T}}$ yields the assertion of the theorem □

Lagrangian polynomial interpolation leads to linear systems of equations:

$$p(t_j) = y_j \iff \sum_{i=0}^n a_i t_j^i = y_j, j = 0, \dots, n$$

\iff solution of $(n+1) \times (n+1)$ linear system $\mathbf{V}\mathbf{a} = \mathbf{y}$ with matrix

$$\mathbf{V} = \begin{pmatrix} 1 & t_0 & t_0^2 & \cdots & t_0^n \\ 1 & t_1 & t_1^2 & \cdots & t_1^n \\ 1 & t_2 & t_2^2 & \cdots & t_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & t_n & t_n^2 & \cdots & t_n^n \end{pmatrix}. \tag{7.2.5}$$

Existence of a solution for a square system gives uniqueness. □

A matrix in the form of \mathbf{V} is called **Vandermonde matrix**.

Given a column vector \mathbf{t} , the corresponding Vandermonde matrix can be generated by

```
for j = 1 : length(t); V(j,:) = t(j).^[0 : length(t) - 1]; end;
```

or

```
for j = 1 : length(t); V(:,j) = t.^(j - 1); end;
```

Theorem 7.2.2 (Lagrange interpolation as linear mapping).
 The polynomial interpolation in the nodes $\mathcal{T} := \{t_j\}_{j=0}^n$ defines a linear operator

$$l_{\mathcal{T}} : \begin{cases} \mathbb{K}^{n+1} \rightarrow \mathcal{P}_n, \\ (y_0, \dots, y_n)^T \mapsto \text{interpolating polynomial } p. \end{cases} \tag{7.2.6}$$

Remark 7.2.2 (Matrix representation of interpolation operator).

In the case of Lagrange interpolation:

- if Lagrange polynomials are chosen as basis for \mathcal{P}_n , $\rightarrow l_{\mathcal{T}}$ is represented by the identity matrix;
- if monomials are chosen as basis for \mathcal{P}_n , $\rightarrow l_{\mathcal{T}}$ is represented by the inverse of the Vandermonde matrix \mathbf{V} , see (7.2.5).



Definition 7.2.3 (Generalized Lagrange polynomials).
 The **generalized Lagrange polynomials** on the nodes $\mathcal{T} = \{t_j\}_{j=0}^n \subset \mathbb{R}$ are $L_i := l_{\mathcal{T}}(\mathbf{e}_{i+1})$, $i = 0, \dots, n$, where $\mathbf{e}_i = (0, \dots, 0, 1, 0, \dots, 0)^T \in \mathbb{R}^{n+1}$ are the unit vectors.

Theorem 7.2.4 (Existence & uniqueness of generalized Lagrange interpolation polynomials).
 The general polynomial interpolation problem (7.2.2) admits a unique solution $p \in \mathcal{P}_n$.

Example 7.2.3. (Generalized Lagrange polynomials for Hermite Interpolation)
 double nodes
 $t_0 = 0, t_1 = 0, t_2 = 1, t_3 = 1 \Rightarrow n = 3$
 (cubic Hermite interpolation).

Explicit formulas for the polynomials \rightarrow see (??).

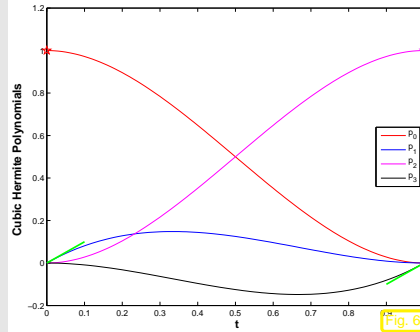


Fig. 62

7.2.2 Conditioning of polynomial interpolation

Necessary for studying the conditioning: norms on vector space of continuous functions $C(I), I \subset \mathbb{R}$

supremum norm $\|f\|_{L^\infty(I)} := \sup\{|f(t)| : t \in I\}$, (7.2.7)

L^2 -norm $\|f\|_{L^2(I)}^2 := \int_I |f(t)|^2 dt$, (7.2.8)

L^1 -norm $\|f\|_{L^1(I)} := \int_I |f(t)| dt$. (7.2.9)

Lemma 7.2.5 (Absolute conditioning of polynomial interpolation).

Given a mesh $\mathcal{T} \subset \mathbb{R}$ with generalized Lagrange polynomials $L_i, i = 0, \dots, n$, and fixed $I \subset \mathbb{R}$, the norm of the interpolation operator satisfies

$$\|I_{\mathcal{T}}\|_{\infty \rightarrow \infty} := \sup_{\mathbf{y} \in \mathbb{K}^{n+1} \setminus \{0\}} \frac{\|I_{\mathcal{T}}(\mathbf{y})\|_{L^\infty(I)}}{\|\mathbf{y}\|_\infty} = \left\| \sum_{i=0}^n |L_i| \right\|_{L^\infty(I)}, \quad (7.2.10)$$

$$\|I_{\mathcal{T}}\|_{2 \rightarrow 2} := \sup_{\mathbf{y} \in \mathbb{K}^{n+1} \setminus \{0\}} \frac{\|I_{\mathcal{T}}(\mathbf{y})\|_{L^2(I)}}{\|\mathbf{y}\|_2} \leq \left(\sum_{i=0}^n \|L_i\|_{L^2(I)}^2 \right)^{\frac{1}{2}}. \quad (7.2.11)$$

Proof. (for the L^∞ -Norm) By Δ -inequality

$$\|I_{\mathcal{T}}(\mathbf{y})\|_{L^\infty(I)} = \left\| \sum_{j=0}^n y_j L_j \right\|_{L^\infty(I)} \leq \sup_{t \in I} \sum_{j=0}^n |y_j| |L_j(t)| \leq \|\mathbf{y}\|_\infty \left\| \sum_{i=0}^n |L_i| \right\|_{L^\infty(I)},$$

equality in (7.2.10) for $\mathbf{y} := (\text{sgn}(L_j(t^*)))_{j=0}^n, t^* := \text{argmax}_{t \in I} \sum_{i=0}^n |L_i(t)|$. \square

Proof. (for the L^2 -Norm) By Δ -inequality and Cauchy-Schwarz inequality

$$\|I_{\mathcal{T}}(\mathbf{y})\|_{L^2(I)} \leq \sum_{j=0}^n |y_j| \|L_j\|_{L^2(I)} \leq \left(\sum_{j=0}^n |y_j|^2 \right)^{\frac{1}{2}} \left(\sum_{j=0}^n \|L_j\|_{L^2(I)}^2 \right)^{\frac{1}{2}}. \quad \square$$

Terminology: Lebesgue constant of \mathcal{T} : $\lambda_{\mathcal{T}} := \left\| \sum_{i=0}^n |L_i| \right\|_{L^\infty(I)}$

Example 7.2.4 (Computation of the Lebesgue constant).

$$I = [-1, 1], \quad \mathcal{T} = \left\{ -1 + \frac{2k}{n} \right\}_{k=0}^n \quad (\text{uniformly spaced nodes})$$

Asymptotic estimate (with (7.2.3) and Stirling formula): for $n = 2m$

$$\left| L_m \left(1 - \frac{1}{n} \right) \right| = \frac{\frac{1}{n} \cdot \frac{1}{n} \cdot \frac{3}{n} \dots \frac{n-3}{n} \cdot \frac{n+1}{n} \dots \frac{2n-1}{n}}{\left(\frac{2}{n} \cdot \frac{4}{n} \dots \frac{n-2}{n} \cdot 1 \right)^2} = \frac{(2n)!}{(n-1)2^{2n}((n/2)!)^2 n!} \sim \frac{2^{n+3/2}}{\pi(n-1)n}$$

Theory [8]: for uniformly spaced nodes $\lambda_{\mathcal{T}} \geq C e^{n/2}$ for $C > 0$ independent of n .

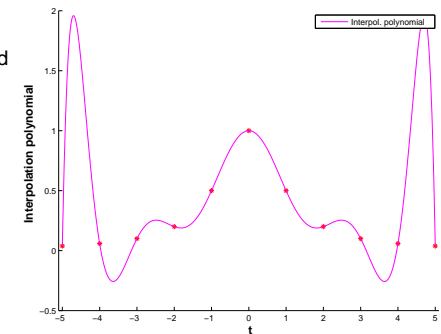
Example 7.2.5 (Oscillating interpolation polynomial: Runge's counterexample).

Between the nodes the interpolation polynomial can oscillate excessively and overestimate the changes in the values: bad approximation of functions!

Interpolation polynomial with uniformly spaced nodes:

$$\mathcal{T} := \left\{ -5 + \frac{10}{n} j \right\}_{j=0}^n, \quad y_j = \frac{1}{1 + t_j^2}, \quad j = 0, \dots, n.$$

Plot $n = 10 \rightarrow$



See example 7.4.3.

Attention:
 strong oscillations of the interpolation polynomials of high degree on uniformly spaced nodes!

7.3 Polynomial Interpolation: Algorithms

Given: nodes $\mathcal{T} := \{-\infty < t_0 < t_1 < \dots < t_n < \infty\}$,
 values $\mathbf{y} := \{y_0, y_1, \dots, y_n\}$,

define: $p := I_{\mathcal{T}}(\mathbf{y})$ as the unique Lagrange interpolation polynomial given by Theorem 7.2.1.

7.3.1 Newton basis and divided differences

Drawback of the Lagrange basis: adding another data point affects *all* basis polynomials!

Alternative, "update friendly" method: **Newton basis** for \mathcal{P}_n

$$N_0(t) := 1, \quad N_1(t) := (t - t_0), \quad \dots, \quad N_n(t) := \prod_{i=0}^{n-1} (t - t_i). \quad (7.3.1)$$

➤ LSE for polynomial interpolation problem in Newton basis:

$$a_j \in \mathbb{R}: \quad a_0 N_0(t_j) + a_1 N_1(t_j) + \dots + a_n N_n(t_j) = y_j, \quad j = 0, \dots, n.$$

⇔ triangular linear system

$$\begin{pmatrix} 1 & 0 & \dots & 0 \\ 1 & (t_1 - t_0) & \dots & \vdots \\ \vdots & \vdots & \dots & 0 \\ 1 & (t_n - t_0) & \dots & \prod_{i=0}^{n-1} (t_n - t_i) \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}.$$

Solution of the system with forward substitution:

$$\begin{aligned} a_0 &= y_0, \\ a_1 &= \frac{y_1 - a_0}{t_1 - t_0} = \frac{y_1 - y_0}{t_1 - t_0}, \\ a_2 &= \frac{y_2 - a_0 - (t_2 - t_0)a_1}{(t_2 - t_0)(t_2 - t_1)} = \frac{y_2 - y_0 - (t_2 - t_0)\frac{y_1 - y_0}{t_1 - t_0}}{(t_2 - t_0)(t_2 - t_1)}, \\ &\vdots \end{aligned}$$

Observation: same quantities computed again and again !

In order to find a better algorithm, we turn to a new interpretation of the coefficients a_j of the interpolating polynomials in Newton basis.

Newton basis polynomial $N_j(t)$: degree j and leading coefficient 1
 ⇒ a_j is the leading coefficient of the interpolating polynomial $p_{0,\dots,j}$

(notation $p_{0,\dots,j}$ introduced in Sect. ??, see (??))

➤ Recursion (??) implies recursion for leading coefficients $a_{\ell,\dots,m}$ of interpolating polynomials $p_{\ell,\dots,m}$, $0 \leq \ell \leq m \leq n$:

$$a_{\ell,\dots,m} = \frac{a_{\ell+1,\dots,m} - a_{\ell,\dots,m-1}}{t_m - t_\ell}.$$

Simpler and more efficient algorithm using **divided differences**:

$$\begin{aligned} y[t_i] &= y_i \\ y[t_i, \dots, t_{i+k}] &= \frac{y[t_{i+1}, \dots, t_{i+k}] - y[t_i, \dots, t_{i+k-1}]}{t_{i+k} - t_i} \end{aligned} \quad (\text{recursion}) \quad (7.3.2)$$

7.3

p. 385

Recursive calculation by **divided differences scheme**, cf. Aitken-Neville scheme, Code ??:

$$\begin{array}{l} t_0 \left| \begin{array}{l} y[t_0] \\ > y[t_0, t_1] \\ t_1 \left| \begin{array}{l} y[t_1] \\ > y[t_0, t_1, t_2] \\ > y[t_1, t_2] \\ t_2 \left| \begin{array}{l} y[t_2] \\ > y[t_1, t_2, t_3] \\ > y[t_2, t_3] \\ t_3 \left| \begin{array}{l} y[t_3] \end{array} \right. \end{array} \right. \end{array} \right. \end{array} > y[t_0, t_1, t_2, t_3], \end{array}$$

the elements are computed from left to right, every ">" means recursion (7.3.2).

If a new data (t_{n+1}, y_{n+1}) is added, it is enough to compute $n + 2$ new terms

$$y[t_{n+1}], y[t_n, t_{n+1}], \dots, y[t_0, \dots, t_{n+1}].$$

7.3

p. 386

Numeric
Methods
401-0654

V.
Gradinaru
D-ITET,
D-MATL

7.3

p. 38

Numeric
Methods
401-0654

V.
Gradinaru
D-ITET,
D-MATL

7.3

p. 38

Code 7.3.1: Divided differences, recursive implementation

```

1 function y = divdiff(t,y)
2 n = length(y)-1;
3 if (n > 0)
4     y(1:n) = divdiff(t(1:n),y(1:n));
5     for j=0:n-1
6         y(n+1) = (y(n+1)-y(j+1))/(t(n+1)-t(j+1));
7     end
8 end

```

By derivation: computed finite differences are the coefficients of interpolating polynomials in Newton basis:

$$p(t) = a_0 + a_1(t-t_0) + a_2(t-t_0)(t-t_1) + \dots + a_n \prod_{j=0}^{n-1} (t-t_j) \quad (7.3.3)$$

$$a_0 = y[t_0], \quad a_1 = y[t_0, t_1], \quad a_2 = y[t_0, t_1, t_2], \quad \dots$$

"Backward evaluation" of $p(t)$ in the spirit of Horner's scheme:

$$p \leftarrow a_n, \quad p \leftarrow (t-t_{n-1})p + a_{n-1}, \quad p \leftarrow (t-t_{n-2})p + a_{n-2}, \quad \dots$$

Code 7.3.2: Divided differences evaluation by modified Horner scheme

```

1 function p = evaldivdiff(t,y,x)
2 n = length(y)-1;
3 dd=divdiff(t,y);
4 p=dd(n+1);
5 for j=n:-1:1
6     p = (x-t(j)).*p+dd(j);
7 end

```

Computational effort: $\bullet O(n^2)$ for computation of divided differences,
 $\bullet O(n)$ for every single evaluation of $p(t)$.

Remark 7.3.3 (Divided differences and derivatives).

If y_0, \dots, y_n are the values of a smooth function f in the points t_0, \dots, t_n , that is, $y_j := f(t_j)$, then

$$y[t_i, \dots, t_{i+k}] = \frac{f^{(k)}(\xi)}{k!}$$

for a certain $\xi \in [t_i, t_{i+k}]$.

△

7.3.2 Extrapolation to zero

Extrapolation is the same as interpolation but the evaluation point t is outside the interval $[\inf_{j=0, \dots, n} t_j, \sup_{j=0, \dots, n} t_j]$. Assume $t = 0$.

Problem: compute $\lim_{t \rightarrow 0} f(t)$ with prescribed precision, when the evaluation of the function $y = f(t)$ is unstable for $|t| \ll 1$.

Known: existence of an asymptotic expansion in h^2

$$f(h) = f(0) + A_1 h^2 + A_2 h^4 + \dots + A_n h^{2n} + R(h), \quad A_k \in \mathbb{K},$$

with remainder estimate $|R(h)| = O(h^{2n+2})$ for $h \rightarrow 0$.

Idea:



- ① evaluation of $f(t_i)$ for different $t_i, i = 0, \dots, n, |t_i| > 0$.
- ② $f(0) \approx p(0)$ with interpolation polynomial $p \in \mathcal{P}_n, p(t_i) = f(t_i)$.

Example 7.3.4 (Numeric differentiation through extrapolation).

For a $(2n+1)$ -times continuously differentiable function $f: D \subset \mathbb{R} \mapsto \mathbb{R}, x \in D$ (Taylor sum in x with Lagrange residual)

$$T(h) := \frac{f(x+h) - f(x-h)}{2h} \sim f'(x) + \sum_{k=1}^n \frac{1}{(2k)!} \frac{d^{2k}f}{dx^{2k}}(x) h^{2k} + \frac{1}{(2n+2)!} f^{(2n+2)}(\xi(x)).$$

Since $\lim_{h \rightarrow 0} T(h) = f'(x) \rightarrow$ estimate of $f'(x)$ by interpolation of T in points h_i .

MATLAB-CODE: Numeric differentiation through interpolation: nodes $x \pm h_0/x$.

```

function d = diffex(f,x,h0,tol)
h = h0;
y(1) = (f(x+h0)-f(x-h0))/(2*h0);
for i=2:10
    h(i) = h(i-1)/2;
    y(i) = (f(x+h(i))-f(x-h(i)))/h(i-1);
    for k=i-1:-1:1
        y(k) = y(k+1)-(y(k+1)-y(k))*h(i)/(h(i)-h(k));
    end
    if (abs(y(2)-y(1)) < tol*abs(y(1))), break; end
end
d = y(1);

```

A posteriori error estimate

diffex2(@atan,1.1,0.5) diffex2(@sqrt,1.1,0.5) diffex2(@exp,1.1,0.5)

Degree	Relative error	Degree	Relative error	Degree	Relative error
0	0.04262829970946	0	0.02849215135713	0	0.04219061098749
1	0.02044767428982	1	0.01527790811946	1	0.02129207652215
2	0.00051308519253	2	0.00061205284652	2	0.00011487434095
3	0.00004087236665	3	0.00004936258481	3	0.00000825582406
4	0.00000048930018	4	0.00000067201034	4	0.00000000589624
5	0.0000000746031	5	0.00000001253250	5	0.0000000009546
6	0.0000000001224	6	0.00000000004816	6	0.0000000000002
		7	0.00000000000021		

advantage: guaranteed accuracy → efficiency

Comparison: numeric differentiation with finite (forward) differences
 → cancellation → smaller accuracy.

MATLAB-CODE : Numeric differentiation through finite differences & relative errors.

```
x=1.1; h=2.^[-1:-5:-36];
atanerr = abs(dirnumdiff(atan,x,h)-1/(1+x^2))*(1+x^2);
sqrtterr = abs(dirnumdiff(sqrt,x,h)-1/(2*sqrt(x)))*(2*sqrt(x));
experr = abs(dirnumdiff(exp,x,h)-exp(x))/exp(x);

function[df]=dirnumdiff(f,x,h)
df=(f(x+h)-f(x))./h;
end
```

$f(x) = \arctan(x)$

h	Relative error
2 ⁻¹	0.20786640808609
2 ⁻⁶	0.00773341103991
2 ⁻¹¹	0.00024299312415
2 ⁻¹⁶	0.0000759482296
2 ⁻²¹	0.00000023712637
2 ⁻²⁶	0.00000001020730
2 ⁻³¹	0.00000005960464
2 ⁻³⁶	0.00000679016113

$f(x) = \sqrt{x}$

h	Relative error
2 ⁻¹	0.09340033543136
2 ⁻⁶	0.00352613693103
2 ⁻¹¹	0.00011094838842
2 ⁻¹⁶	0.00000346787667
2 ⁻²¹	0.00000010812198
2 ⁻²⁶	0.00000001923506
2 ⁻³¹	0.00000001202188
2 ⁻³⁶	0.00000198842224

$f(x) = \exp(x)$

h	Relative error
2 ⁻¹	0.29744254140026
2 ⁻⁶	0.00785334954789
2 ⁻¹¹	0.00024418036620
2 ⁻¹⁶	0.00000762943394
2 ⁻²¹	0.00000023835113
2 ⁻²⁶	0.0000000429331
2 ⁻³¹	0.00000012467100
2 ⁻³⁶	0.00000495453865

7.4 Interpolation Error Estimates

Perspective approximation of a function by polynomial interpolation

Remark 7.4.1 (Approximation by polynomials).

? Is it always possible to approximate a continuous function by polynomials?

✓ Yes! Recall the **Weierstrass theorem**:
 A continuous function f on the interval $[a, b] \subset \mathbb{R}$ can be uniformly approximated by polynomials.

! But not by the interpolation on a fixed mesh [42, pag. 331]:
 Given a sequence of meshes of increasing size $\{\mathcal{T}_j\}_{j=1}^\infty$, $\mathcal{T}_j = \{x_1^{(j)}, \dots, x_j^{(j)}\} \subset [a, b]$,
 $a \leq x_1^{(j)} < x_2^{(j)} < \dots < x_j^{(j)} \leq b$, there exists a continuous function f such that
 the sequence interpolating polynomials of f on \mathcal{T}_j does not converge uniformly to f as
 $j \rightarrow \infty$.

We consider Lagrangian polynomial interpolation on node set

$$\mathcal{T} := \{t_0, \dots, t_n\} \subset I, I \subset \mathbb{R}, \text{ interval of length } |I|.$$

Notation: For a continuous function $f : I \mapsto \mathbb{K}$ we define the polynomial interpolation operator, see Thm. 7.2.2

$$l_{\mathcal{T}}(f) := l_{\mathcal{T}}(\mathbf{y}) \in \mathcal{P}_n \quad \text{with} \quad \mathbf{y} := (f(t_0), \dots, f(t_n))^T \in \mathbb{K}^{n+1}.$$

Goal: estimate of the interpolation error norm $\|f - l_{\mathcal{T}}f\|$ (for some norm on $C(I)$).

Focus: asymptotic behavior of interpolation error

Example 7.4.2 (Asymptotic behavior of polynomial interpolation error).

Interpolation of $f(t) = \sin t$ on equispaced nodes in $I = [0, \pi]$: $\mathcal{T} = \{j\pi/n\}_{j=0}^n$.

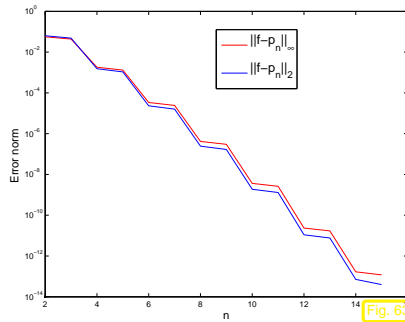
Interpolation polynomial $p := l_{\mathcal{T}}f \in \mathcal{P}_n$.

By Thm. 7.4.2:

$$\|f^{(k)}\|_{L^\infty(I)} \leq 1, \quad \Rightarrow \quad \|f - p\|_{L^\infty(I)} \leq \frac{1}{(1+n)!} \max_{t \in I} \left| (t-0)\left(t-\frac{\pi}{n}\right)\left(t-\frac{2\pi}{n}\right) \cdots (t-\pi) \right|$$

$$\forall k \in \mathbb{N}_0 \quad \leq \frac{1}{n+1} \left(\frac{\pi}{n}\right)^{n+1}$$

→ **Uniform exponential convergence** of the interpolation polynomials
(It holds for every mesh of nodes \mathcal{T})



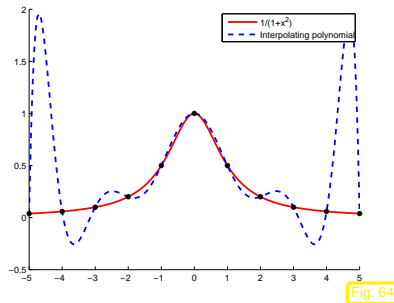
MATLAB-experiment: computation of the norms.

- L^∞ -norm: sampling on a grid of meshsize $\pi/1000$.
- L^2 -norm: numeric quadrature (→ Chapter 10) with trapezoidal rule on a grid of meshsize $\pi/1000$.

Example 7.4.3 (Runge's example).

Polynomial interpolation of $f(t) = \frac{1}{1+t^2}$ with equispaced nodes:

$$\mathcal{T} := \left\{ t_j := -5 + \frac{10}{n} j \right\}_{j=0}^n, \quad y_j = \frac{1}{1+t_j^2}, \quad j = 0, \dots, n.$$



Interpolating polynomial, $n = 10$

Observations: Strong oscillations of $\mathcal{I}_{\mathcal{T}} f$ near the endpoints of the interval:

$$\|f - \mathcal{I}_{\mathcal{T}} f\|_{L^\infty([-5,5])} \xrightarrow{n \rightarrow \infty} \infty.$$

How can this be reconciled with Thm. 7.4.2 ?

Numerical Methods
401-0654

Here $f(t) = \frac{1}{1+t^2}$ implies $|f^{(n)}(t)| = 2^n n! \cdot O(|t|^{-2-n})$.

→ The error bound from Thm. 7.4.1 $\rightarrow \infty$ for $n \rightarrow \infty$.

Numerical Methods
401-0654

◇

$$\exists C \neq C(n): \|f - \mathcal{I}_{\mathcal{T}} f\| \leq CT(n) \quad \text{for } n \rightarrow \infty. \quad (7.4.1)$$

Classification (best bound for $T(n)$):

$$\exists p > 0: \quad T(n) \leq n^{-p} \quad : \quad \text{algebraic convergence, with rate } p > 0,$$

$$\exists 0 < q < 1: \quad T(n) \leq q^n \quad : \quad \text{exponential convergence}.$$

V.

Gradinaru
D-ITET,
D-MATL

V.

Gradinaru
D-ITET,
D-MATL

Remark 7.4.4 (Exploring convergence).

Given: pairs (n_i, ϵ_i) , $i = 1, 2, 3, \dots$, $n_i \hat{=}$ polynomial degrees, $\epsilon_i \hat{=}$ norms of interpolation error

① Conjectured: **algebraic convergence**: $\epsilon_i \approx Cn_i^{-p}$

7.4

p. 397

$$\log(\epsilon_i) \approx \log(C) - p \log n_i \quad (\text{affine linear in log-log scale}).$$

7.4

p. 35

Numerical Methods
401-0654

Apply linear regression (MATLAB `polyfit`) to points $(\log n_i, \log \epsilon_i)$ > estimate for rate p .

Numerical Methods
401-0654

$$\text{exponential convergence} \quad \epsilon_i \approx C \exp(-\beta n_i)$$

$$\log \epsilon_i \approx \log(C) - \beta n_i \quad (\text{affine linear in lin-log scale}).$$

Apply linear regression (MATLAB `polyfit`) to points $(n_i, \log \epsilon_i)$ > estimate for $q := \exp(-\beta)$.

V.
Gradinaru
D-ITET,
D-MATL

Beware: same concept ↔ different meanings:

V.
Gradinaru
D-ITET,
D-MATL

- **convergence** of a sequence (e.g. of iterates $\mathbf{x}^{(k)} \rightarrow$ Sect. 1.1)
- **convergence** of an approximation (dependent on an approximation parameter, e.g. n)

7.4

p. 398

7.4

p. 40

Theorem 7.4.1 (Representation of interpolation error).

$f \in C^{n+1}(I): \forall t \in I: \exists \tau_t \in]\min\{t, t_0, \dots, t_n\}, \max\{t, t_0, \dots, t_n\}[$:

$$f(t) - \mathcal{I}_{\mathcal{T}}(f)(t) = \frac{f^{(n+1)}(\tau_t)}{(n+1)!} \cdot \prod_{j=0}^n (t - t_j) . \quad (7.4.2)$$

The theorem can also be proved using the following lemma.

Lemma 7.4.2 (Error of the polynomial interpolation). For $f \in C^{n+1}(I): \forall t \in I$:

$$f(t) - \mathcal{I}_{\mathcal{T}}(f)(t) = \int_0^1 \int_0^{\tau_1} \dots \int_0^{\tau_{n-1}} \int_0^{\tau_n} f^{(n+1)}(t_0 + \tau_1(t_1 - t_0) + \dots + \tau_n(t_n - t_{n-1}) + \tau(t - t_n)) d\tau d\tau_n \dots d\tau_1 \cdot \prod_{j=0}^n (t - t_j) .$$

Proof. By induction on n , use (??) and the fundamental theorem of calculus [44, Sect. 3.1]:

Remark 7.4.5. Lemma 7.4.2 holds also for Hermite Interpolation. △

Interpolation error estimate requires smoothness!

Remark 7.4.6 (L^2 -error estimates).

Thm. 7.4.1 gives error estimates for the L^∞ -Norm. And the other norms?

From Lemma. 7.4.2 using Cauchy-Schwarz inequality:

$$\begin{aligned} \|f - \mathcal{I}_{\mathcal{T}}(f)\|_{L^2(I)}^2 &= \int_I \left| \int_0^1 \int_0^{\tau_1} \dots \int_0^{\tau_{n-1}} \int_0^{\tau_n} f^{(n+1)}(\dots) d\tau d\tau_n \dots d\tau_1 \cdot \underbrace{\prod_{j=0}^n (t - t_j)}_{|t-t_j| \leq |I|} \right|^2 dt \\ &\leq \int_I |I|^{2n+2} \underbrace{\text{vol}_{(n+1)}(S_{n+1})}_{=1/(n+1)!} \int_{S_{n+1}} |f^{(n+1)}(\dots)|^2 d\tau dt \\ &= \int_I \frac{|I|^{2n+2}}{(n+1)!} \underbrace{\text{vol}_{(n)}(C_{t,\tau})}_{\leq 2^{(n-1)/2}/n!} |f^{(n+1)}(\tau)|^2 d\tau dt , \end{aligned}$$

where

$$S_{n+1} := \{\mathbf{x} \in \mathbb{R}^{n+1}: 0 \leq x_n \leq x_{n-1} \leq \dots \leq x_1 \leq 1\} \quad (\text{unit simplex}) ,$$

$$C_{t,\tau} := \{\mathbf{x} \in S_{n+1}: t_0 + x_1(t_1 - t_0) + \dots + x_n(t_n - t_{n-1}) + x_{n+1}(t - t_n) = \tau\} .$$

This gives the bound for the L^2 -norm of the error:

$$\Rightarrow \|f - \mathcal{I}_{\mathcal{T}}(f)\|_{L^2(I)} \leq \frac{2^{(n-1)/4} |I|^{n+1}}{\sqrt{(n+1)!n!}} \left(\int_I |f^{(n+1)}(\tau)|^2 d\tau \right)^{1/2} . \quad (7.4.3)$$

Notice:

$f \mapsto \|f^{(n)}\|_{L^2(I)}$ defines a **seminorm** on $C^{n+1}(I)$
(**Sobolev-seminorm**, measure of the smoothness of a function).

7.5 Chebyshev Interpolation

Perspective:

function **approximation** by polynomial interpolation

>

Freedom to choose interpolation nodes judiciously

7.5.1 Motivation and definition

Mesh of nodes: $\mathcal{T} := \{t_0 < t_1 < \dots < t_{n-1} < t_n\}, n \in \mathbb{N}$,

function $f: I \rightarrow \mathbb{R}$ continuous; without loss of generality $I = [-1, 1]$.

Thm. 7.4.1:

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

$$w(t) := (t - t_0) \dots (t - t_n) .$$



Idea: choose nodes t_0, \dots, t_n such that $\|w\|_{L^\infty(I)}$ is minimal!

Equivalent to finding $q \in \mathcal{P}_{n+1}$, with leading coefficient = 1, such that $\|q\|_{L^\infty(I)}$ is minimal.

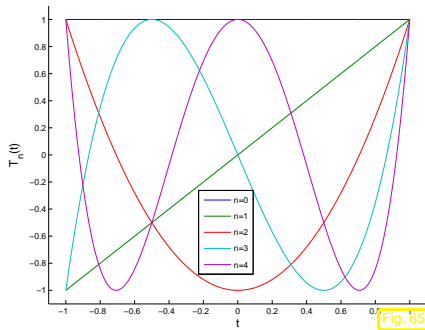
Choice of $t_0, \dots, t_n =$ zeros of q (caution: t_j must belong to I).

Heuristic:

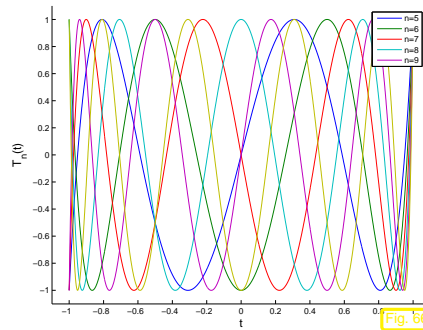
- t^* extremal point of $q \rightarrow |q(t^*)| = \|q\|_{L^\infty(I)}$,
- q has $n + 1$ zeros in I ,
- $|q(-1)| = |q(1)| = \|q\|_{L^\infty(I)}$.

Definition 7.5.1 (Chebyshev polynomial).

The n^{th} Chebyshev polynomial is $T_n(t) := \cos(n \arccos t)$, $-1 \leq t \leq 1$.



Chebyshev polynomials T_0, \dots, T_4



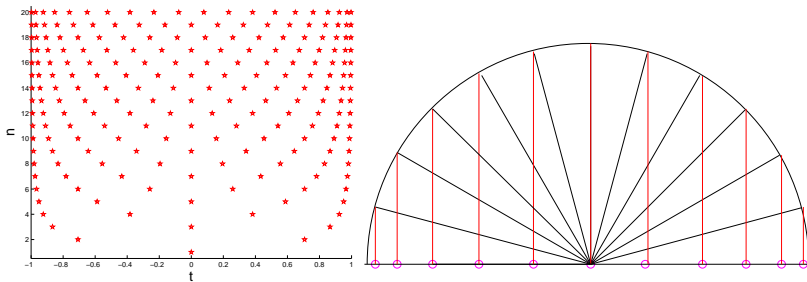
Chebyshev polynomials T_5, \dots, T_9

Zeros of T_n : $t_k = \cos\left(\frac{2k-1}{2n}\pi\right)$, $k = 1, \dots, n$. (7.5.1)

Extrema (alternating signs) of T_n :

$$|T_n(\bar{t}_k)| = 1 \Leftrightarrow \exists k = 0, \dots, n: \bar{t}_k = \cos\frac{k\pi}{n}, \quad \|T_n\|_{L^\infty([-1,1])} = 1.$$

Chebyshev nodes t_k from (7.5.1):



Remark 7.5.1 (3-term recursion for Chebyshev polynomial).

3-term recursion by $\cos(n+1)x = 2 \cos nx \cos x - \cos(n-1)x$ with $\cos x = t$:

$$T_{n+1}(t) = 2tT_n(t) - T_{n-1}(t), \quad T_0 \equiv 1, \quad T_1(t) = t, \quad n \in \mathbb{N}. \quad (7.5.2)$$

- This implies:
- $T_n \in \mathcal{P}_n$,
 - leading coefficients equal to 2^{n-1} ,
 - T_n linearly independent,
 - T_n basis of $\mathcal{P}_n = \text{Span}\{T_0, \dots, T_n\}$, $n \in \mathbb{N}_0$.

Theorem 7.5.2 (Minimax property of the Chebyshev polynomials).

$$\|T_n\|_{L^\infty([-1,1])} = \inf\{\|p\|_{L^\infty([-1,1])} : p \in \mathcal{P}_n, p(t) = 2^{n-1}t^n + \dots\}, \quad \forall n \in \mathbb{N}.$$

Proof. See [14, Section 7.1.4.] □

Application to approximation by polynomial interpolation:

- For $I = [-1, 1]$
- “optimal” interpolation nodes $\mathcal{T} = \left\{ \cos\left(\frac{2k+1}{2(n+1)}\pi\right), k = 0, \dots, n \right\}$,
 - $w(t) = (t-t_0)\dots(t-t_{n+1}) = 2^{-n}T_{n+1}(t)$, $\|w\|_{L^\infty(I)} = 2^{-n}$, with leading coefficient 1.

Then, by Thm. 7.4.1,

$$\|f - l_{\mathcal{T}}(f)\|_{L^\infty([-1,1])} \leq \frac{2^{-n}}{(n+1)!} \|f^{(n+1)}\|_{L^\infty([-1,1])}. \quad (7.5.3)$$

Remark 7.5.2 (Chebyshev polynomials on arbitrary interval).

How to use Chebyshev polynomial interpolation on an arbitrary interval?

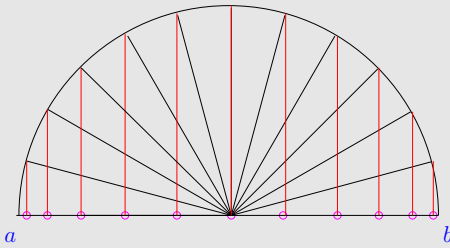
Scaling argument: interval transformation requires the transport of the functions

$$[-1, 1] \xrightarrow{\hat{t} \mapsto t := a + \frac{1}{2}(\hat{t}+1)(b-a)} [a, b] \leftrightarrow \hat{f}(\hat{t}) := f(t).$$

$$p \in \mathcal{P}_n \wedge p(t_j) = f(t_j) \Leftrightarrow \hat{p} \in \mathcal{P}_n \wedge \hat{p}(\hat{t}_j) = \hat{f}(\hat{t}_j).$$

With transformation formula for the integrals & $\frac{d^n \hat{f}}{d\hat{t}^n}(\hat{t}) = \left(\frac{1}{2}|I|\right)^n \frac{d^n f}{dt^n}(t)$:

$$\|f - l_{\mathcal{T}}(f)\|_{L^\infty(I)} = \|\hat{f} - l_{\hat{\mathcal{T}}}(\hat{f})\|_{L^\infty([-1,1])} \leq \frac{2^{-n}}{(n+1)!} \left\| \frac{d^{n+1} \hat{f}}{d\hat{t}^{n+1}} \right\|_{L^\infty([-1,1])} \\ \leq \frac{2^{-2n-1}}{(n+1)!} |I|^{n+1} \|f^{(n+1)}\|_{L^\infty(I)}. \quad (7.5.4)$$



The **Chebyshev nodes** in the interval $I = [a, b]$ are

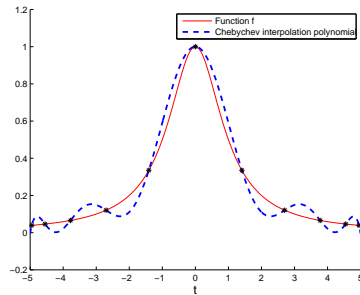
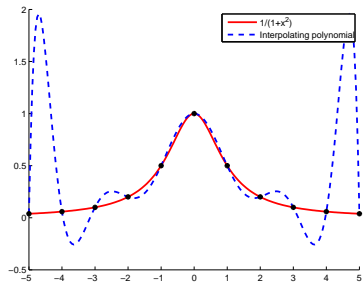
$$t_k := a + \frac{1}{2}(b-a) \left(\cos\left(\frac{2k+1}{2(n+1)}\pi\right) + 1 \right), \quad (7.5.5)$$

$$k = 0, \dots, n.$$

7.5.2 Chebyshev interpolation error estimates

Example 7.5.3 (Polynomial interpolation: Chebyshev nodes versus equidistant nodes).

Runge's function $f(t) = \frac{1}{1+t^2}$, see Ex. 7.4.3, polynomial interpolation based on uniformly spaced nodes and Chebyshev nodes:

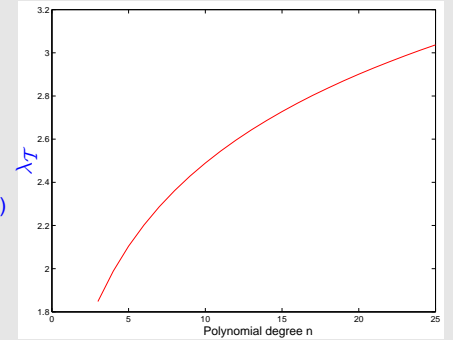


Remark 7.5.4 (Lebesgue Constant for Chebyshev nodes).

Theory [7, 56, 55]:

$$\lambda_T \sim \frac{2}{\pi} \log(1+n) + o(1),$$

$$\lambda_T \leq \frac{2}{\pi} \log(1+n) + 1. \quad (7.5.6)$$



Example 7.5.5 (Chebyshev interpolation error).

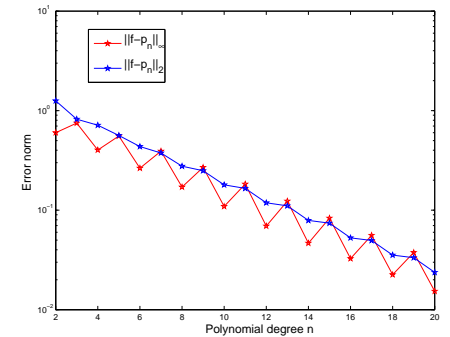
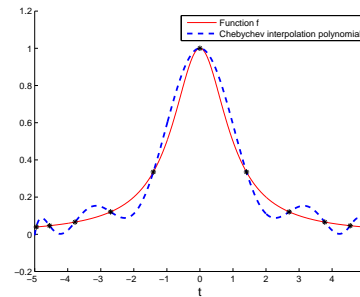
For $I = [a, b]$ let $x_l := a + \frac{b-a}{N}l$, $l = 0, \dots, N$, $N = 1000$ we approximate the norms of the error

$$\|f - p\|_\infty \approx \max_{0 \leq l \leq N} |f(x_l) - p(x_l)|$$

$$\|f - p\|_2^2 \approx \frac{b-a}{2N} \sum_{0 \leq l < N} (|f(x_l) - p(x_l)|^2 + |f(x_{l+1}) - p(x_{l+1})|^2)$$

① $f(t) = (1+t^2)^{-1}$, $I = [-5, 5]$ (see Ex. 7.4.3)

Interpolation with $n = 10$ Chebyshev nodes (plot on the left).



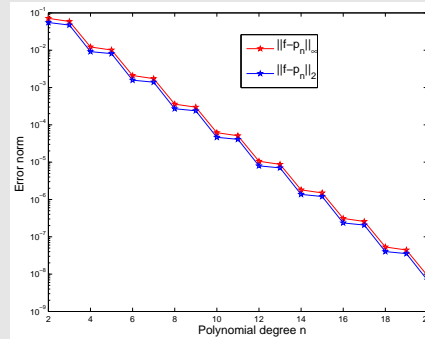
Notice: exponential convergence of the Chebychev interpolation:

$$p_n \rightarrow f, \quad \|f - I_n f\|_{L^2([-5,5])} \approx 0.8^n$$

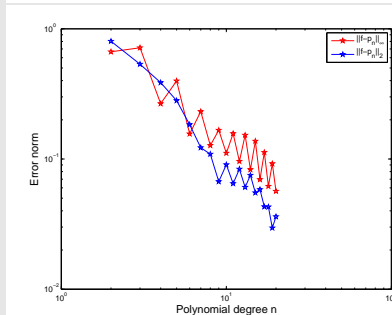
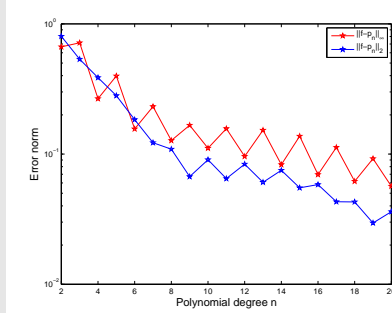
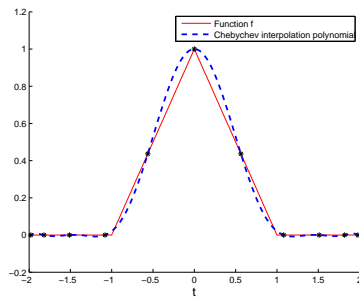
Now: the same function $f(t) = (1+t^2)^{-1}$ on a smaller interval $I = [-1, 1]$.

(Faster) exponential convergence:

$$\|f - I_n f\|_{L^2([-1,1])} \approx 0.42^n.$$



② $f(t) = \max\{1 - |t|, 0\}$, $I = [-2, 2]$, $n = 10$ nodes (plot on the left).
 $f \in C^0(I)$ but $f \notin C^1(I)$.

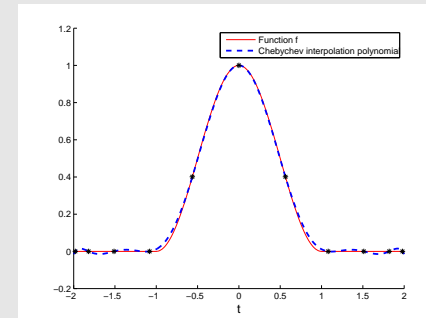


From the double logarithmic plot, notice

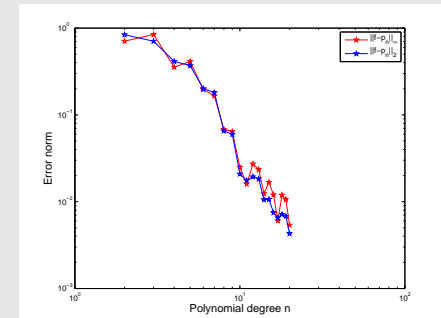
- no exponential convergence
- algebraic convergence (?)



③ $f(t) = \begin{cases} \frac{1}{2}(1 + \cos \pi t) & |t| < 1 \\ 0 & 1 \leq |t| \leq 2 \end{cases}$ $I = [-2, 2]$, $n = 10$ (plot on the left).



Notice: only algebraic convergence.



7.5.3 Chebychev interpolation: computational aspects

Task: Given: given degree $n \in \mathbb{N}$, continuous function $f : [-1, 1] \mapsto \mathbb{R}$

Sought: **efficient** representation/evaluation of interpolating polynomial $p \in \mathcal{P}_n$ in Chebychev nodes (7.5.5) on $[-1, 1]$

```
class ChebInterp {
private:
    // various internal data describing Chebychev interpolating polynomial p
public:
    // Constructor taking function f and degree n as arguments
    PolyInterp(const Function &f, unsigned int n);
    // Evaluation operator: y_j = p(x_j), j = 1, ..., m (m "large")
    double eval(const vector<double> &x, vector<double> &y) const;
};
```



Trick: expand p into Chebychev polynomials

$$p = \sum_{j=0}^n \alpha_j T_j, \quad \alpha_j \in \mathbb{R}. \quad (7.5.7)$$

The representation (7.5.7) is always possible, because $\{T_0, \dots, T_n\}$ is a *basis* of \mathcal{P}_n

Remark 7.5.6 (Fast evaluation of Chebychev expansion). → [29, Alg. 32.1]

Use the 3-term recurrence (7.5.2)

$$T_j(x) = 2xT_{j-1}(x) - T_{j-2}(x), \quad j = 2, 3, \dots,$$

to rewrite (7.5.7) as

$$p(x) = \sum_{j=0}^{n-1} \tilde{\alpha}_j T_j \quad \text{with} \quad \tilde{\alpha}_j = \begin{cases} \alpha_j + 2x\alpha_{j+1} & , \text{ if } j = n-1, \\ \alpha_j - \alpha_{j+2} & , \text{ if } j = n-2, \\ \alpha_j & \text{ else.} \end{cases} \quad (7.5.8)$$

▶ recursive algorithm, see Code 7.5.6.

Code 7.5.7: Recursive evaluation of Chebychev expansion (7.5.7)

```

1 function y = recclenshaw(a,x)
2 % Recursive evaluation of a polynomial  $p = \sum_{j=1}^{n+1} a_j T_{j-1}$  at point x, see (7.5.8)
3 % The coefficients  $a_j$  have to be passed in a row vector.
4 n = length(a)-1;
5 if (n<2), y = a(1)+x*a(2);
6 else
7     y = recclenshaw([a(1:n-2), a(n-1)-a(n+1), a(n)+2*x*a(n+1)],x);
8 end
    
```

Non-recursive version: **Clenshaw algorithm**

Code 7.5.8: Clenshaw algorithm for evaluation of Chebychev expansion (7.5.7)

```

1 function y = clenshaw(a,x)
2 % Clenshaw algorithm for evaluating  $p = \sum_{j=1}^{n+1} a_j T_{j-1}$  at points passed in vector x
3 n = length(a)-1; % degree of polynomial
4 d = repmat(reshape(a,n+1,1),1,length(x));
5 for j=n:-1:2
6     d(j,:) = d(j,:) + 2*x.*d(j+1,:); % see (7.5.8)
7     d(j-1,:) = d(j-1,:) - d(j+1,:);
8 end
9 y = d(1,:) + x.*d(2,:);
    
```

▶ Computational effort: $O(nm)$ for evaluation at m points

Numerical
Methods
401-0654

Issue: How to compute the Chebychev expansion coefficients α_j in (7.5.7) efficiently from the interpolation conditions

$$p(t_k) = f(t_k), \quad k = 0, \dots, n, \quad \text{for } t_k := \cos\left(\frac{2k+1}{2(n+1)}\pi\right). \quad (7.5.9)$$

Chebychev nodes

Trick: transformation of p into a 1-periodic function:

$$\begin{aligned}
 q(s) &:= p(\cos 2\pi s) = \sum_{j=0}^n \alpha_j T_j(\cos 2\pi s) \exp(-2\pi i n s) \stackrel{\text{Def. 7.5.1}}{=} \sum_{j=0}^n \alpha_j \cos(2\pi j s) \\
 &= \sum_{j=0}^n \frac{1}{2} \alpha_j (\exp(2\pi i j s) + \exp(-2\pi i j s)) \\
 &= \sum_{j=-n}^{n+1} \beta_j \exp(-2\pi i j s), \quad \beta_j := \begin{cases} 0 & , \text{ for } j = n+1, \\ \frac{1}{2} \alpha_j & , \text{ for } j = 1, \dots, n, \\ \alpha_0 & , \text{ for } j = 0, \\ \frac{1}{2} \alpha_{-n-j} & , \text{ for } j = -n, \dots, -1. \end{cases}
 \end{aligned} \quad (7.5.10)$$

7.5
p. 417

V.
Gradina
D-ITET,
D-MATL

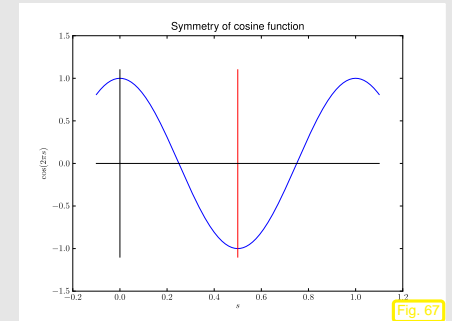
7.5
p. 41

Transformed interpolation conditions (7.5.9):

$$t = \cos(2\pi s) \stackrel{(7.5.9)}{\implies} q\left(\frac{2k+1}{4(n+1)}\right) = f(t_k), \quad k = 0, \dots, n. \quad (7.5.11)$$

Observe **symmetry**

$$\begin{aligned}
 q(s) &= q(1-s) \\
 &\downarrow \leftarrow (7.5.11) \\
 q\left(1 - \frac{2k+1}{4(n+1)}\right) &= y_k, \quad k = 0, \dots, n.
 \end{aligned}$$



V.
Gradina
D-ITET,
D-MATL

V.
Gradina
D-ITET,
D-MATL

Extend interpolation conditions (7.5.11) by symmetry, see Fig. 68

$$q\left(\frac{k}{2(n+1)} + \frac{1}{4(n+1)}\right) = z_k := \begin{cases} y_k & , \text{ for } k = 0, \dots, n, \\ y_{2n+1-k} & , \text{ for } k = n+1, \dots, 2n+1. \end{cases} \quad (7.5.12)$$

7.5
p. 418

7.5
p. 42

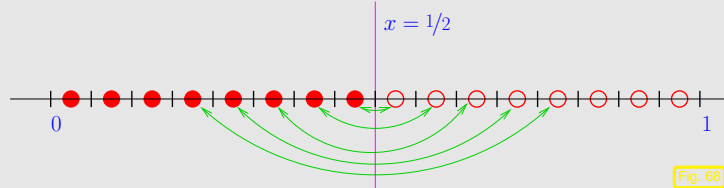


Fig. 68

⇔ linear system of equations (at last):

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

⇕

$$\sum_{j=0}^{2n+1} \left(\beta_j \exp\left(-\frac{2\pi i(j-n)}{4(n+1)}\right) \right) \underbrace{\exp\left(-\frac{2\pi i}{2n+1} k j\right)}_{\omega_{2(n+1)}^{kj}} = \exp\left(-\pi i \frac{nk}{n+1}\right) z_k, \quad k = 0, \dots, 2n+1 .$$

⇕

$$\mathbf{F}_{2(n+1)} \mathbf{c} = \mathbf{z} \quad \text{with} \quad \mathbf{c} = \left(\beta_j \exp\left(-\frac{2\pi i j}{4(n+1)}\right) \right)_{j=0}^{2n+1} . \quad (7.5.13)$$

$(2n+2) \times (2n+2)$ Fourier matrix, see (6.2.6)

► solve (7.5.13) with inverse discrete Fourier transform, see 6.2:

asymptotic complexity $O(n \log n)$ (→ Sect. 6.3)

Note: by symmetry of \mathbf{z} → $\beta_{2n+1} = 0$, cf. (7.5.10)!

Code 7.5.9: Efficient computation of Chebyshev expansion coefficient of Chebyshev interpolant

```

1 function a = chebexp(y)
2 % Efficiently compute coefficients  $\alpha_j$  in the Chebyshev expansion
3 %  $p = \sum_{j=0}^n \alpha_j T_j$  of  $p \in \mathcal{P}_n$  based on values  $y_k$ .
4 %  $k = 0, \dots, n$ , in Chebyshev nodes  $t_k$ ,  $k = 0, \dots, n$ . These values are
5 % passed in the row vector  $y$ .
6 n = length(y) - 1; % degree of polynomial
7 % create vector  $\mathbf{z}$  by wrapping and componentwise scaling
8 z = exp(-pi*i*n/(n+1)*(0:2*n+1)) .* [y, y(end:-1:1)]; % r.h.s. vector
9 c = fft(z); % Solve linear system (7.5.13) with effort  $O(n \log n)$ 
10 b = real(exp(0.5*pi*i/(n+1)*(-n:n+1)) .* c); % recover  $\beta_j$ , see (7.5.13)
11 a = [b(n+1), 2*b(n+2:n+1)]; % recover  $\alpha_j$ , see (7.5.10)

```

Remark 7.5.10 (Chebyshev representation of built-in functions).

Computers use approximation by sums of Chebyshev polynomials in the computation of functions like $\log, \exp, \sin, \cos, \dots$. The evaluation by means of Clenshaw algorithm according to Code 7.5.7 is more efficient and stable than for approximation by Taylor polynomials.



8

Trigonometric Interpolation

Is there something else than polynomials and Taylor approximation in the world?



Idea (J. Fourier 1822): Approximation of a function not by usual polynomials but by trigonometrical polynomials = partial sum of a Fourier series

We call **trigonometrical polynomial** of degree $\leq 2m$ the function

$$T_{2m}(t) := t \mapsto \sum_{j=-m}^m \gamma_j e^{-2\pi i j t}, \quad \gamma_j \in \mathbb{C}, t \in \mathbb{R} .$$

Remark 8.0.1. $T_{2m} : \mathbb{R} \rightarrow \mathbb{C}$ is periodic of period 1. Moreover, if $\gamma_{-j} = \overline{\gamma_j}$ for all $j = 0, \dots, 2m$, then $T_{2m}(t)$ takes only real values and may be written as

$$T_{2m}(t) = \frac{a_0}{2} + \sum_{j=1}^{2m} (a_j \cos(2\pi jt) + b_j \sin(2\pi jt))$$

with $a_0 = 2\gamma_0$ and $a_j = 2 \operatorname{Re} \gamma_j$, $b_j = -2 \operatorname{Im} \gamma_j$ for all $j = 1, \dots, 2m$.

Remark 8.0.2.

The functions $w_j(t) = e^{-2\pi i j t}$ are orthogonal with respect to the $L^2(]0, 1[)$ scalar product.

Let us take as granted (or known from lectures in Analysis, Mathematical Methods of Physics, etc.):

Theorem 8.0.1 (L^2 -convergence of the Fourier series). Every squared integrable function $f \in L^2(]0, 1[) := \{f :]0, 1[\rightarrow \mathbb{C} : \|f\|_{L^2(]0, 1[)} < \infty\}$ is the $L^2(]0, 1[)$ -limit of its Fourier series

$$f(t) = \sum_{k=-\infty}^{\infty} \widehat{f}(k) e^{2\pi i k t} \quad \text{in } L^2(]0, 1[),$$

with Fourier coefficients defined by

$$\widehat{f}(k) = \int_0^1 f(t) e^{-2\pi i k t} dt, \quad k \in \mathbb{Z}.$$

Remark 8.0.3. In view of this theorem, we may think of one function in two ways: once in the time (or space) domain $t \rightarrow f(t)$ and once in the frequency domain $k \rightarrow \widehat{f}(k)$.

$$\text{Isometry property (Parseval):} \quad \sum_{k=-\infty}^{\infty} |\widehat{f}(k)|^2 = \|f\|_{L^2(]0, 1[)}^2 \quad (8.0.1)$$

Remark 8.0.4. A real function f may be equally represented as

$$\frac{a_0}{2} + \sum_{j=1}^{\infty} (a_j \cos(2\pi jt) + b_j \sin(2\pi jt)) \quad \text{in } L^2(]0, 1[),$$

with

$$a_j = 2 \int_0^1 f(t) \cos(2\pi jt) dt, \quad j \geq 0,$$

$$b_j = 2 \int_0^1 f(t) \sin(2\pi jt) dt, \quad j \geq 1.$$

Lemma 8.0.2 (Derivative and Fourier coefficients).

$$f \in L^1(]0, 1[) \ \& \ f' \in L^1(]0, 1[) \Rightarrow \widehat{f}'(k) = 2\pi i k \widehat{f}(k), \quad k \in \mathbb{Z}.$$

Remark 8.0.5.

$$\|f^{(n)}\|_{L^2(]0, 1[)}^2 = (2\pi)^{2n} \sum_{k=-\infty}^{\infty} k^{2n} |\widehat{f}(k)|^2. \quad (8.0.2)$$

$$\text{If } |\widehat{f^{(n)}}(k)| \leq \int_0^1 |f^{(n)}(t)| dt < \infty \Rightarrow \widehat{f}(k) = O(k^{-n}) \text{ for } |k| \rightarrow \infty.$$

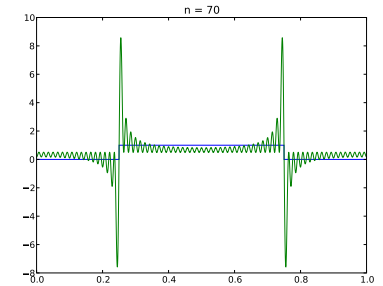
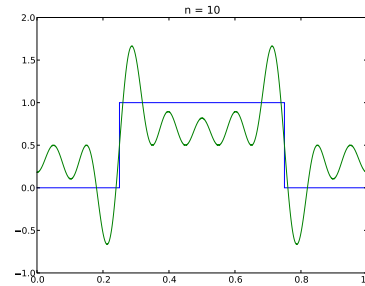
The smoothness of a function directly reflects in the quick decay of its Fourier coefficients.

Example 8.0.6. The Fourier series associated with the characteristic function of an interval $[a, b] \subset]0, 1[$ may be computed analytically as

$$b - a + \frac{1}{\pi} \sum_{|k|=1}^{\infty} e^{-ikc} \frac{\sin kd}{k} e^{i2\pi kt}, \quad t \in [0, 1],$$

with $c = \pi(a + b)$ and $d = \pi(b - a)$.

Note the slow decay of the Fourier coefficients, and hence expect slow convergence of the series. Moreover, observe in the pictures below the Gibbs phenomenon: the ripples move closer to the discontinuities and increase with larger n . Explanation: we have L^2 -convergence but no uniform convergence of the series!



Remark 8.0.7. Usually one cannot compute analytically $\widehat{f}(k)$ or one has to rely only on discrete values at nodes $x_\ell = \frac{\ell}{N}$ for $\ell = 0, 1, \dots, N$; the trapezoidal rule gives

$$\widehat{f}(k) \approx \frac{1}{N} \sum_{\ell=0}^{N-1} f(x_\ell) e^{-2\pi i k x_\ell} =: \widehat{f}_N(k) \quad (8.0.3)$$

8.1 Trigonometric Polynomials

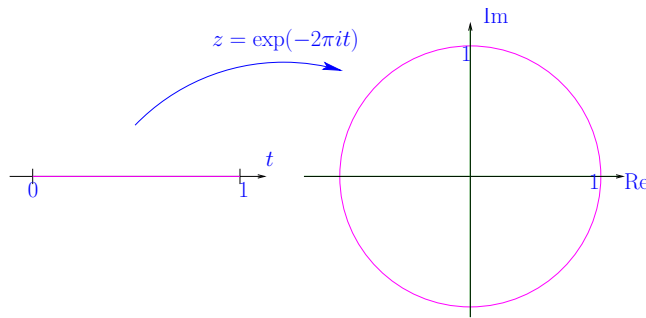
Definition 8.1.1 (Trigonometric polynomial). The space \mathcal{P}_n^T of *trigonometric polynomials* of degree $\leq n$ is

$$\mathcal{P}_n^T := \begin{cases} \{t \mapsto \sum_{j=-m}^m \gamma_j e^{-2\pi i j t}, \gamma_j \in \mathbb{C}\} & , \text{ if } n = 2m, m \in \mathbb{N}, \\ \{t \mapsto \sum_{j=-m+1}^m \gamma_j e^{-2\pi i j t}, \gamma_j \in \mathbb{C}\} & , \text{ if } n = 2m - 1, m \in \mathbb{N}. \end{cases}$$

Why do we call them trigonometric Polynomials ?

$$p(t) := \sum_{j=-m[+1]}^m \gamma_j e^{-2\pi i j t} = \sum_{j=-m[+1]}^m \gamma_j z^j \hat{=} \text{(Laurent)-polynomial on } \mathbb{S}^1$$

Parametrisation: $[0, 1[\xrightarrow[t \mapsto z]{z = \exp(-2\pi i t)} \mathbb{S}^1 := \{z \in \mathbb{C} : |z| = 1\}.$



Definition 8.1.2 (Laurent-polynomial). The space \mathcal{P}_n^L of *Laurent-polynomials* of degree $\leq n$ on \mathbb{K} is

$$\mathcal{P}_n^L := \begin{cases} \{z \mapsto \sum_{j=-m}^m \gamma_j z^j, \gamma_j \in \mathbb{K}\} & , \text{ if } n = 2m, m \in \mathbb{N}, \\ \{z \mapsto \sum_{j=-m+1}^m \gamma_j z^j, \gamma_j \in \mathbb{K}\} & , \text{ if } n = 2m - 1, m \in \mathbb{N}. \end{cases}$$

Note: trigonometric polynomials \subset 1-periodic functions on \mathbb{R} ($\leftrightarrow f(t) = f(t + 1)$)

Problem of trigonometric polynomial interpolation

Given *equidistant nodes* $\mathcal{T} := \{j/n, j = 0, \dots, n - 1\}$ and values $y_0, \dots, y_{n-1} \in \mathbb{C}$ find $p \in \mathcal{P}_{n-1}^T$, such that

$$p(j/n) = y_j, \quad j = 0, \dots, n - 1.$$

Theorem 8.1.3. The problem of trigonometric polynomial interpolation has an (unique) solution.

Note: 1-periodic functions on $\mathbb{R} \leftrightarrow$ functions $\mathbb{S}^1 \mapsto \mathbb{K}$

$$f(t) = f(t + 1) \quad \forall t \in \mathbb{R} \quad \leftrightarrow \quad g(z) = g(e^{2\pi i t}) := f(t) \quad \forall z \in \mathbb{S}^1.$$

► trigonometric polynomial interpolation \leftrightarrow equidistant polynomial interpolation on \mathbb{S}^1

► For nodes $\mathcal{T}_n = \{\omega_n^j\}_{j=0}^{n-1}$ (complex roots of unity, section 6.2) and values y_0, \dots, y_{n-1} find $p \in \mathcal{P}_{n-1}^L$ (complex coefficients), such that

$$p(\omega_n^k) = y_k, \quad k = 0, \dots, n - 1.$$

Reformulation of the problem of trigonometric polynomial interpolation:

• if $n = 2m, m \in \mathbb{N}$, then $p(t) = \sum_{j=-m+1}^m \gamma_j e^{-2\pi i j t} \in \mathcal{P}_{n-1}^L$

$$p(\frac{k}{n}) = y_k, \quad k = 0, \dots, n - 1.$$

$$\sum_{j=-m+1}^m \gamma_j \omega_n^{jk} = \sum_{j=0}^m \gamma_j \omega_n^{jk} + \sum_{j=1}^{m-1} \gamma_{-j} \omega_n^{(n-j)k} = \sum_{j=0}^m \gamma_j \omega_n^{jk} + \sum_{j=m+1}^{n-1} \gamma_{j-n} \omega_n^{jk} = y_k,$$

$\omega_n \hat{=}$ complex roots of unity, section 6.2.

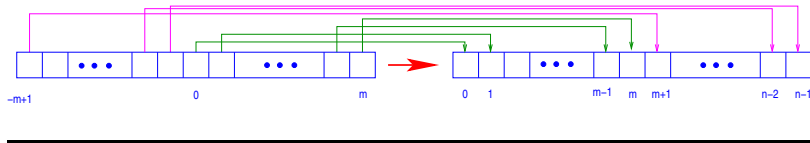
• if $n = 2m + 1, m \in \mathbb{N}$, then $p(t) = \sum_{j=-m}^m \gamma_j e^{-2\pi i j t} \in \mathcal{P}_{n-1}^L$

$$p(\frac{k}{n}) = y_k, \quad k = 0, \dots, n - 1.$$

$$\sum_{j=-m}^m \gamma_j \omega_n^{jk} = \sum_{j=0}^m \gamma_j \omega_n^{jk} + \sum_{j=1}^m \gamma_{-j} \omega_n^{(n-j)k} = \sum_{j=0}^m \gamma_j \omega_n^{jk} + \sum_{j=m+1}^{n-1} \gamma_{j-n} \omega_n^{jk} = y_k.$$

$\square \hat{=} \mathbf{F}_n \mathbf{g} = \mathbf{y}, \quad \mathbf{g} = (\gamma_0 \cdots \gamma_m \gamma_{m+1-n} \cdots \gamma_{n-1}), \quad \mathbf{y} = (y_0 \cdots y_{n-1})$

8.1 Fourier-matrix (6.2.6) hence we get γ_j by inverse **discrete Fourier transform** (\rightarrow Def. 6.2.3) + rearrangement of coefficients: p. 43



Aim: Evaluation of a trigonometric polynomial $p \in \mathcal{P}_{n-1}^T$ (\rightarrow Def. 8.1.1) at equidistant nodes $\frac{l}{N}$, $l = 0, \dots, N-1$, $N > n$, in $[0, 1[$:

For $n = 2m$:

$$p\left(\frac{l}{N}\right) = \sum_{j=-m+1}^m \gamma_j \omega_N^{lj} = \sum_{j=0}^m \gamma_j \omega_N^{lj} + \sum_{j=1}^{m-1} \gamma_{-j} \omega_N^{(N-j)l} = \sum_{j=0}^{N-1} \tilde{\gamma}_j \omega_N^{lj},$$

with $\tilde{\gamma}_j := \begin{cases} \gamma_j & , \text{if } 0 \leq j \leq m-1, \\ \gamma_{j-N} & , \text{if } N-m+1 \leq j \leq N-1, \\ 0 & \text{otherwise.} \end{cases}$

hence use **discrete Fourier transform** \rightarrow
 Def. 6.2.3
 of length N
 coefficients in c
 are rearranged before storage !

```

MATLAB-CODE Auswertung eines
function v= evaliptrig(y,N)
n = length(y);
if (mod(n,2) ~= 0), error; end
c = ifft(y);
c = [c(1:n/2), zeros(1,N-n), ...
     c(n/2+1:n)];
v = fft(c);
trigonometrischen Interpolationspolynom
  
```

8.2 Trigonometric Interpolation: Error Estimates

Linear trigonometric interpolation operator: for 1-periodic $f : \mathbb{R} \mapsto \mathbb{C}$

$$T_n(f) := p \in \mathcal{P}_{n-1}^T \quad \text{with} \quad p\left(\frac{j}{n}\right) = y_j, \quad j = 0, \dots, n-1.$$

Example 8.2.1 (Trigonometric interpolation).

#1 step function: $f(t) = 0$ for $|t - \frac{1}{2}| > \frac{1}{4}$, $f(t) = 1$ for $|t - \frac{1}{2}| \leq \frac{1}{4}$

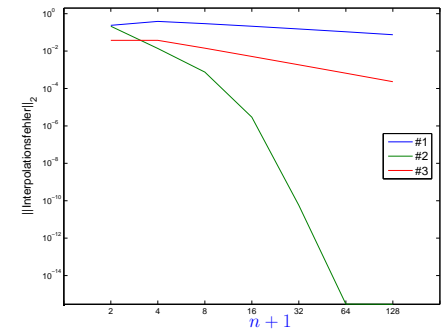
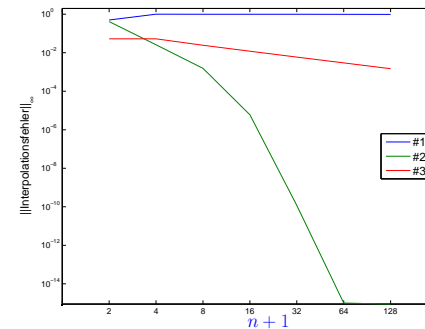
#2 smooth periodic function: $f(t) = \frac{1}{\sqrt{1 + \frac{1}{2} \sin(2\pi t)}}$

#3 hat function: $f(t) = |t - \frac{1}{2}|$

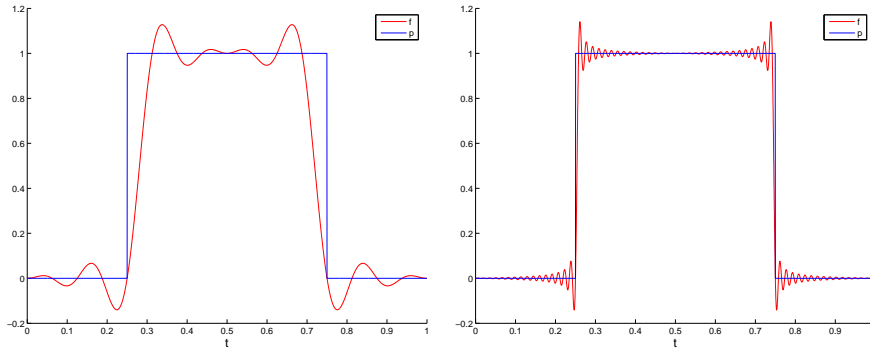
Note: computation of the norms of the interpolation error:

```

t = 0:1/n:1; y = feval(f,t(1:end-1));
v = real(evaliptrig(y,4096));
v = [v,v(1)]; fv = feval(f,0:1/4096:1);
d = abs(fv-v); linf = max(d); l2 = 1/64*norm(d(1:end-1));
  
```



Note: Cases #1, #3: algebraic convergence
 Case #2: exponential convergence

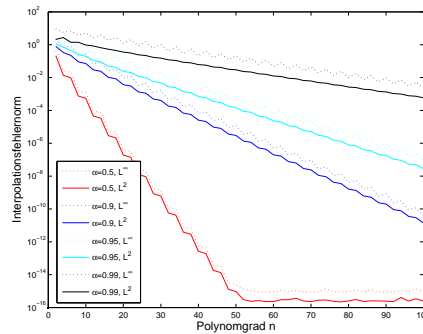


$n = 16$ **Gibbs'phenomenon:** ripples near discontinuities $n = 128$

Example 8.2.2 (Trigonometric interpolation of analytic functions).

$$f(t) = \frac{1}{\sqrt{1 - \alpha \sin(2\pi t)}} \text{ auf } I = [0, 1].$$

approximation of the norms: "oversampling" in 4096 points



Note: exponential convergence in degree n , quicker for smaller α

Analysis uses: trigonometric polynomials = partial sums of **Fourier series**

Theorem 8.2.1 (Error of DFT; aliasing formula). If $\sum_{k \in \mathbb{Z}} \hat{f}(k)$ absolut converges, then

$$\hat{f}_N(k) - \hat{f}(k) = \sum_{\substack{j \in \mathbb{Z} \\ j \neq 0}} \hat{f}(k + jN)$$

Corollary 8.2.2. Let $f \in C^p$ with $p \geq 2$ and 1-periodic. Then it holds:

$$\hat{f}_N(k) - \hat{f}(k) = O(N^{-p}) \text{ for } |k| \leq \frac{N}{2}$$

$$\text{for: } h = \frac{1}{N}, \quad h \sum_{j=0}^{N-1} f(x_j) - \int_0^1 f(x) dx = O(h^p).$$

Remark 8.2.3. $\hat{f}_N(k)$ is N -periodic, while $\hat{f}(k) \rightarrow 0$ quickly

$\hat{f}_N(k)$ is a bad approximation of $\hat{f}(k)$ for k large ($k \approx N$), but for $|k| \leq \frac{N}{2}$ it is good enough.

Theorem 8.2.3 (Trigonometric interpolant). Let N even and $z = \mathcal{F}_N y \in \mathbb{C}^N$. The trigonometric interpolant

$$p_N(x) := \sum_{k=-N/2}^{N/2} z_k e^{2\pi i k x} := \frac{1}{2} \left(z_{-N/2} e^{-2\pi i x N/2} + z_{N/2} e^{2\pi i x N/2} \right) + \sum_{|k| \leq N/2} z_k e^{2\pi i k x}$$

has the property that $p_N(x_\ell) = y_\ell$, where $x_\ell = \ell/N$.

Theorem 8.2.4 (Error of trigonometric interpolation). If f is 1-periodic and $\sum_{k \in \mathbb{Z}} \hat{f}(k)$ absolut converges, then

$$|p_N(x) - f(x)| \leq 2 \sum_{|k| \geq N/2} |\hat{f}(k)| \quad \forall x \in \mathbb{R}$$

Corollary 8.2.5 (Sampling-Theorem). Let f 1-periodic with maximum frequency M : $\hat{f}(k) = 0$ for all $|k| > M$. Then $p_N(x) = f(x)$ for all x , if $N > 2M$

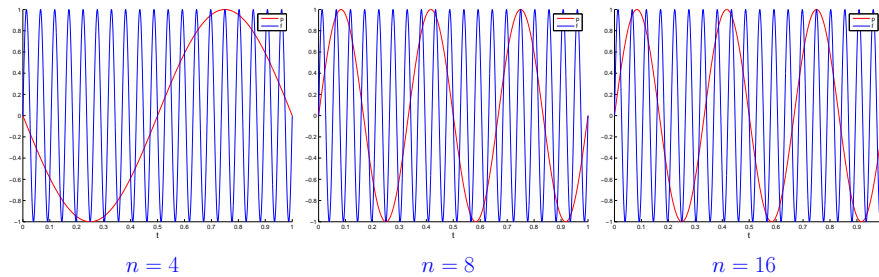
data compression

Remark 8.2.4 (Aliasing).

$$\mathcal{T} = \left\{ \frac{j}{n} \right\}_{j=0}^{n-1} \blacktriangleright e^{\frac{2\pi i j}{n} t} = e^{\frac{2\pi i (N-n)j}{N} t} \blacktriangleright T_n(e^{2\pi i \cdot}) = T_n(e^{2\pi i (N \bmod n) \cdot}).$$

hence: The trigonometric interpolation of $t \rightarrow e^{2\pi i N t}$ and $t \rightarrow e^{2\pi i (N \bmod n) t}$ of degree n give the same trigonometric interpolation polynomial! \rightarrow **Aliasing**

Example for $f(t) = \sin(2\pi \cdot 19t)$ $\triangleright N = 38$:



[Linearity of trigonometric interpolation]

For $f \in C([0, 1])$ 1-periodic, $n = 2m$:

$$f(t) = \sum_{j=-\infty}^{\infty} \hat{f}(j) e^{2\pi i j t} \Rightarrow T_n(f)(t) = \sum_{j=-m+1}^m \gamma_j e^{2\pi i j t}, \quad \gamma_j = \sum_{l=-\infty}^{\infty} \hat{f}(j + ln).$$

► Fourier coefficients of the error function $e = f - T_n(f)$:

$$\hat{e}(j) = \begin{cases} -\sum_{|l|=1}^{\infty} \hat{f}(j + ln) & , \text{ if } -m + 1 \leq j \leq m, \\ \hat{f}(j) & , \text{ if } j \leq -m \vee j > m. \end{cases}$$

$$\stackrel{(8.0.1)}{\Rightarrow} \|f - T_n(f)\|_{L^2([0,1])}^2 \leq \left| \sum_{|l|=1}^{\infty} \hat{f}(j + ln) \right|^2 + \sum_{|j| \geq m} |\hat{f}(j)|^2. \quad (8.2.1)$$

► may be estimated, if we know the decay of the Fourier coefficients $\hat{f}(j)$ \Leftrightarrow smoothness of f , see (8.0.2)

Theorem 8.2.6 (Error estimate for trigonometric interpolation). For $k \in \mathbb{N}$:

$$f^{(k)} \in L^2([0, 1]) \Rightarrow \|f - T_n f\|_{L^2([0,1])} \leq \sqrt{1 + c_k} n^{-k} \|f^{(k)}\|_{L^2([0,1])},$$

with $c_k = 2 \sum_{l=1}^{\infty} (2l - 1)^{-2k}$.

9

Filtering and Wavelets

We focus on elementary tools from the signal theory: filters and filter banks. They yield to wavelets and to the more general idea of approximation at multiple scales.

Note that in this chapter the Fourier transform is thought on $[-\pi, \pi]$, according to the common conventions in signal theory,

9.1 Discrete Filters

Definition 9.1.1. A **filter** is a linear time invariant system that acts on input sequences x producing an output sequence y as a **convolution** with a fixed sequence h

$$y_n = \sum_k h_k x_{n-k}$$