Approximation Theory and Proof Assistants: Certified Computations

Nicolas Brisebarre and Damien Pous

Master 2 Informatique Fondamentale
École Normale Supérieure de Lyon, 2023-2024

Section 2.2. Best L^∞ (or minimax) approximation

Theorem 7 (Alternation Theorem. Chebyshev? Borel (1905)? Kirchberger (1902))

Let $\{\varphi_0, \ldots, \varphi_n\}$ *be a Chebyshev system over* $[a, b]$*. Let* $f \in \mathcal{C}([a, b])$*. A generalized polynomial* $p = \sum_{k=0}^{n} \alpha_k \varphi_k$ *is the best approximation (or minimax approximation) of* f *iff there exist* $n+2$ *points* $x_0, \ldots, x_{n+1}$*,* $a \leqslant x_0 < x_1 < \cdots < x_{n+1} \leqslant b$ *such that, for all* k*,*

$$f(x_k) - p(x_k) = (-1)^k (f(x_0) - p(x_0)) = \pm \|f - p\|_\infty .$$

Section 2.2. Best L^∞ (or minimax) approximation

Theorem 7 (Alternation Theorem. Chebyshev? Borel (1905)? Kirchberger (1902))

Let $\{\varphi_0, \ldots, \varphi_n\}$ be a Chebyshev system over $[a,b]$. Let $f \in \mathcal{C}([a,b])$. A generalized polynomial $p = \sum_{k=0}^{n} \alpha_k \varphi_k$ is the best approximation (or minimax approximation) of f iff there exist $n+2$ points $x_0, \ldots, x_{n+1}$, $a \leqslant x_0 < x_1 < \cdots < x_{n+1} \leqslant b$ such that, for all k,

$$f(x_k) - p(x_k) = (-1)^k \left(f(x_0) - p(x_0)\right) = \pm \|f - p\|_\infty .$$

In other words, $p = \sum_{k=0}^{n} \alpha_k \varphi_k$ is the best approximation if and only if the error function $f - p$ has $n+2$ extrema, all global (of the same absolute value) and with alternating signs.

Section 2.2. Best L^∞ (or minimax) approximation

Theorem 8

(La Vallée Poussin) Let $f \in \mathcal{C}([a,b])$*. Let* $\{\varphi_0, \ldots, \varphi_n\}$ *be a Chebyshev system over* $[a,b]$*, and let* $p \in \operatorname{Span}_{\mathbb{R}} \{\varphi_0, \ldots, \varphi_n\}$*. If there exist* $x_0 < x_1 < \cdots < x_{n+1}$ *such that* $p - f$ *alternates at the* x_i*, then*

$$\min_i |f(x_i) - p(x_i)| \leqslant E_n(f) \leqslant \|f - p\|_\infty ,$$

where $E_n(f) = \inf_{q \in \operatorname{Span}_{\mathbb{R}}\{\varphi_i\}} \|f - q\|_\infty$.

Section 2.2. Best L^∞ (or minimax) approximation

Theorem 8

(La Vallée Poussin) Let $f \in \mathcal{C}([a,b])$*. Let* $\{\varphi_0, \ldots, \varphi_n\}$ *be a Chebyshev system over* $[a,b]$*, and let* $p \in \operatorname{Span}_{\mathbb{R}} \{\varphi_0, \ldots, \varphi_n\}$*. If there exist* $x_0 < x_1 < \cdots < x_{n+1}$ *such that* $p - f$ *alternates at the* x_i*, then*

$$\min_i |f(x_i) - p(x_i)| \leqslant E_n(f) \leqslant \|f - p\|_\infty ,$$

where $E_n(f) = \inf_{q \in \operatorname{Span}_{\mathbb{R}}\{\varphi_i\}} \|f - q\|_\infty$.

Remark

Let $\{\varphi_0, \ldots, \varphi_n\}$ *be a Chebyshev system over* $[a,b]$*. These statements remain valid if* $[a,b]$ *is replaced with any compact subset of* $\mathbb{R}$ *containing at least* $n+2$ *points.*

Section 2.2. Best L^∞ (or minimax) approximation - Remez' algorithm

Remez published in 1934 an algorithm to approximate the minimax polynomial.

Section 2.2. Best L^∞ (or minimax) approximation - Remez' algorithm

Input A segment $[a, b]$, a function $f \in \mathcal{C}([a, b])$, a Chebyshev system $\{\varphi_i\}$, a tolerance Δ.

Output An approximation of the best approximation of f on the system $\{\varphi_i\}$.

1. Choose $n + 2$ points $x_0 < x_1 < \cdots < x_{n+1}$ in $[a, b]$, $\delta \leftarrow 1, \varepsilon \leftarrow 0$.
2. While $\delta \geqslant \Delta |\varepsilon|$ do
   a. Solve for $a_0, \ldots, a_n$ and ε the linear system

$$\sum_{k=0}^{n} a_k \varphi_k(x_j) - f(x_j) = (-1)^j \varepsilon, \qquad j = 0, \ldots, n + 1.$$

Section 2.2. Best L^∞ (or minimax) approximation - Remez' algorithm

Input A segment $[a,b]$, a function $f \in \mathcal{C}([a,b])$, a Chebyshev system $\{\varphi_i\}$, a tolerance Δ.

Output An approximation of the best approximation of f on the system $\{\varphi_i\}$.

1. Choose $n+2$ points $x_0 < x_1 < \cdots < x_{n+1}$ in $[a,b]$, $\delta \leftarrow 1, \varepsilon \leftarrow 0$.
2. While $\delta \geqslant \Delta |\varepsilon|$ do
   a. Solve for $a_0, \ldots, a_n$ and ε the linear system

$$\sum_{k=0}^{n} a_k \varphi_k(x_j) - f(x_j) = (-1)^j \varepsilon, \qquad j = 0, \ldots, n+1.$$

   b. Choose $x_{\text{new}} \in [a,b]$ such that

$$\|p - f\|_\infty = |p(x_{\text{new}}) - f(x_{\text{new}})|, \text{ with } p = \sum_{k=0}^{n} a_k \varphi_k.$$

   Replace one of the x_i with x_{new}, s.t. $p - f$ alternates at $x_{0,\text{new}}, \ldots, x_{n+1,\text{new}}$. Set $\delta = |p(x_{\text{new}}) - f(x_{\text{new}})| - |\varepsilon|$.

Section 2.2. Best L^∞ (or minimax) approximation - Remez' algorithm

Input A segment $[a, b]$, a function $f \in \mathcal{C}([a, b])$, a Chebyshev system $\{\varphi_i\}$, a tolerance Δ.

Output An approximation of the best approximation of f on the system $\{\varphi_i\}$.

1. Choose $n+2$ points $x_0 < x_1 < \cdots < x_{n+1}$ in $[a, b]$, $\delta \leftarrow 1, \varepsilon \leftarrow 0$.
2. While $\delta \geqslant \Delta |\varepsilon|$ do
   a. Solve for $a_0, \ldots, a_n$ and ε the linear system

$$\sum_{k=0}^{n} a_k \varphi_k(x_j) - f(x_j) = (-1)^j \varepsilon, \qquad j = 0, \ldots, n+1.$$

   b. Choose $x_{\text{new}} \in [a, b]$ such that

$$\|p - f\|_\infty = |p(x_{\text{new}}) - f(x_{\text{new}})|, \text{ with } p = \sum_{k=0}^{n} a_k \varphi_k.$$

   Replace one of the x_i with x_{new}, s.t. $p - f$ alternates at $x_{0,\text{new}}, \ldots, x_{n+1,\text{new}}$. Set $\delta = |p(x_{\text{new}}) - f(x_{\text{new}})| - |\varepsilon|$.
3. Return p.

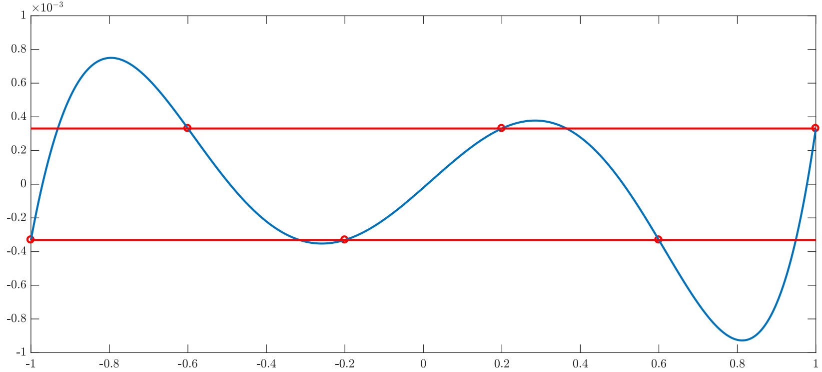
Section 2.2. Remez' algorithm, an example (Silviu-Ioan Filip)

Degree-4 minimax approximation to $\exp$ over $[-1, 1]$

First iteration: $x_j = -1 + 2j/5$, $j = 0, \ldots, 5$.

Section 2.2. Remez' algorithm, an example (Silviu-Ioan Filip)

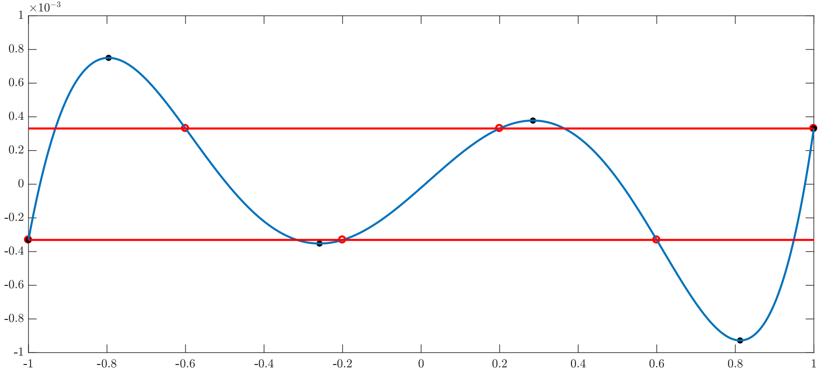
Degree-4 minimax approximation to $\exp$ over $[-1, 1]$



First iteration: $x_j = -1 + 2j/5$, $j = 0, \ldots, 5$.
leveled error $\varepsilon = 3.3083e - 04$, approximation error $= 9.2751e - 04$.

Section 2.2. Remez' algorithm, an example (Silviu-Ioan Filip)

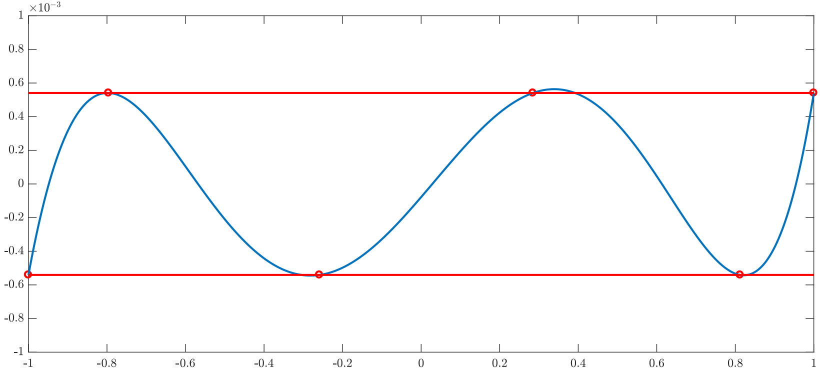
Degree-4 minimax approximation to $\exp$ over $[-1, 1]$



First iteration: $x_j = -1 + 2j/5$, $j = 0, \ldots, 5$.
leveled error $\varepsilon = 3.3083e - 04$, approximation error $= 9.2751e - 04$.

Section 2.2. Remez' algorithm, an example (Silviu-Ioan Filip)

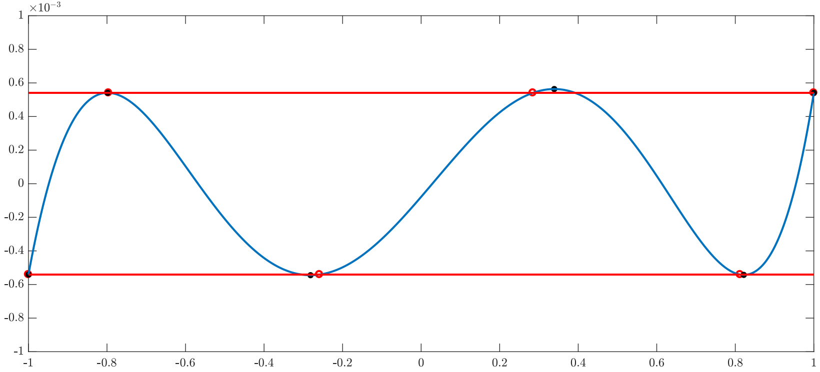
Degree-4 minimax approximation to $\exp$ over $[-1, 1]$



Second iteration:
leveled error $\varepsilon = 5.4083e - 04$, approximation error $= 5.6350e - 04$.

Section 2.2. Remez' algorithm, an example (Silviu-Ioan Filip)

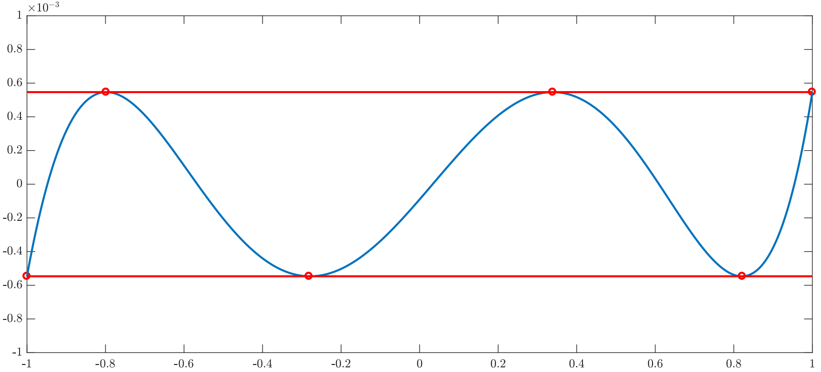
Degree-4 minimax approximation to $\exp$ over $[-1, 1]$



Second iteration:
leveled error $\varepsilon = 5.4083e - 04$, approximation error $= 5.6350e - 04$.

Section 2.2. Remez' algorithm, an example (Silviu-Ioan Filip)

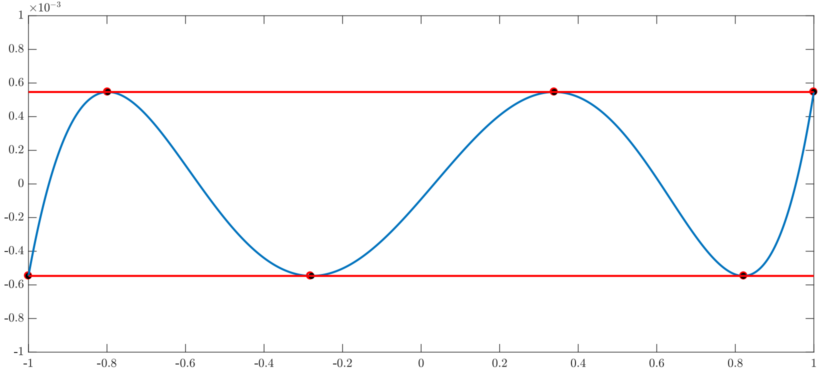
Degree-4 minimax approximation to $\exp$ over $[-1, 1]$



Third iteration:
leveled error $\varepsilon = 5.4665e - 04$, approximation error $= 5.4670e - 04$.

Section 2.2. Remez' algorithm, an example (Silviu-Ioan Filip)

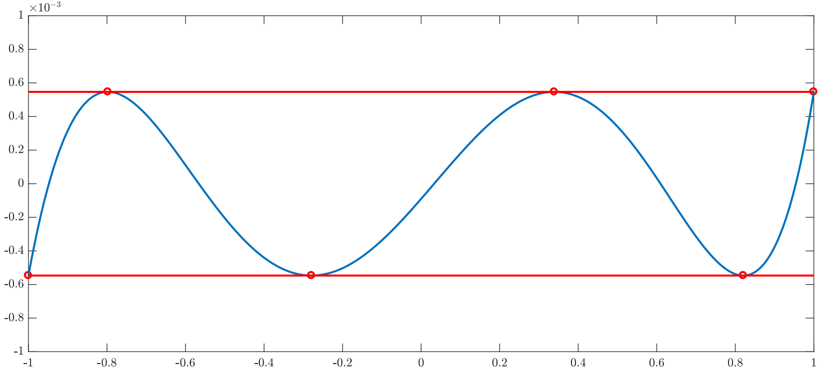
Degree-4 minimax approximation to $\exp$ over $[-1, 1]$



Third iteration:
leveled error $\varepsilon = 5.4665e - 04$, approximation error $= 5.4670e - 04$.

Section 2.2. Remez' algorithm, an example (Silviu-Ioan Filip)

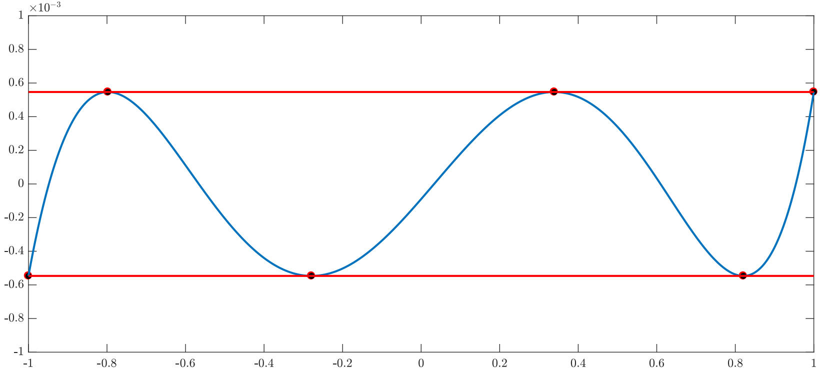
Degree-4 minimax approximation to $\exp$ over $[-1, 1]$



Fourth iteration:
leveled error $\varepsilon = 5.4667e - 04$, approximation error $= 5.4667e - 04$.

Section 2.2. Remez' algorithm, an example (Silviu-Ioan Filip)

Degree-4 minimax approximation to $\exp$ over $[-1, 1]$



Fourth iteration:
leveled error $\varepsilon = 5.4667e - 04$, approximation error $= 5.4667e - 04$.

Section 2.2. Best L^∞ (or minimax) approximation

Theorem 9

Let p_k denote the value of p after k loop turns, and let p^* be such that $E_n(f) = \|f - p^*\|$. There exists $\theta \in (0,1)$ such that $\|p_k - p^*\| = O\left(\theta^k\right)$.

Under mild regularity assumptions, the bound $O\left(\theta^k\right)$ can in fact be improved to $O(\theta^{2^k})$ (Veidinger, 1960).

Section 2.3. Polynomial Interpolation

Interpolation problem: given pairwise distinct $x_0, \ldots, x_n \in [a, b]$ and values $y_0, \ldots, y_n \in \mathbb{R}$, compute

$$p \in \mathbb{R}[x], \text{ s.t. } p\left(x_i\right) = y_i.$$

If $f \in \mathcal{C}\left([a, b]\right)$, consider $y_i = f(x_i)$ for $i = 0, \ldots, n$.

Section 2.3. Polynomial Interpolation

Interpolation problem: given pairwise distinct $x_0, \ldots, x_n \in [a, b]$ and values $y_0, \ldots, y_n \in \mathbb{R}$, compute

$$p \in \mathbb{R}[x], \text{ s.t. } p\left(x_i\right) = y_i.$$

If $f \in \mathcal{C}\left([a, b]\right)$, consider $y_i = f(x_i)$ for $i = 0, \ldots, n$.

Natural to focus on techniques for computing these interpolants:

- sometimes a finite number of values is the only information we have on the function,

Section 2.3. Polynomial Interpolation

Interpolation problem: given pairwise distinct $x_0, \ldots, x_n \in [a, b]$ and values $y_0, \ldots, y_n \in \mathbb{R}$, compute

$$p \in \mathbb{R}[x], \text{ s.t. } p\left(x_i\right) = y_i.$$

If $f \in \mathcal{C}\left([a, b]\right)$, consider $y_i = f(x_i)$ for $i = 0, \ldots, n$.

Natural to focus on techniques for computing these interpolants:

- sometimes a finite number of values is the only information we have on the function,
- Step 2.a of Remez' algorithm requires an efficient interpolation process,

Section 2.3. Polynomial Interpolation

Interpolation problem: given pairwise distinct $x_0, \ldots, x_n \in [a, b]$ and values $y_0, \ldots, y_n \in \mathbb{R}$, compute

$$p \in \mathbb{R}[x], \text{ s.t. } p(x_i) = y_i.$$

If $f \in \mathcal{C}([a, b])$, consider $y_i = f(x_i)$ for $i = 0, \ldots, n$.

Natural to focus on techniques for computing these interpolants:

- sometimes a finite number of values is the only information we have on the function,
- Step 2.a of Remez' algorithm requires an efficient interpolation process,
- Theorem 6 shows that, for all n, there exists $a \leqslant z_0 < z_1 < \cdots < z_n \leqslant b$ such that $f(z_i) = p^*(z_i)$ for $i = 0, \ldots, n$, where p^* is the minimax approximation of f: the polynomial p^* is an interpolation polynomial of f.

Section 2.3. Polynomial Interpolation

Let A be a commutative ring (with unity). Given pairwise distinct $x_0, \ldots, x_n \in A$ and $y_0, \ldots, y_n \in A$, find $p \in A_n[x]$ such that $p(x_i) = y_i$ for all i. Write $p = \sum_k a_k x^k$. It can be restated as

$$V \cdot \mathbf{a} = \mathbf{y}$$

where V is a Vandermonde matrix. If $\det V$ is invertible, there is a unique solution.

Section 2.3. Polynomial Interpolation

Let A be a commutative ring (with unity). Given pairwise distinct $x_0, \ldots, x_n \in A$ and $y_0, \ldots, y_n \in A$, find $p \in A_n[x]$ such that $p(x_i) = y_i$ for all i. Write $p = \sum_k a_k x^k$. It can be restated as

$$V \cdot \mathbf{a} = \mathbf{y}$$

where V is a Vandermonde matrix. If $\det V$ is invertible, there is a unique solution.

Here we assume $A = \mathbb{R}$. If the x_i are pairwise distinct, there is a unique solution.

Section 2.3. Polynomial Interpolation - Linear System Solving

Given pairwise distinct $x_0, \ldots, x_n \in \mathbb{R}$ and $y_0, \ldots, y_n \in \mathbb{R}$,
find $p = \sum_k a_k x^k \in \mathbb{R}_n[x]$ such that $p(x_i) = y_i$ for all i i.e.

$$V \cdot \mathbf{a} = \mathbf{y}$$

where V is a Vandermonde matrix.

Section 2.3. Polynomial Interpolation - Linear System Solving

Given pairwise distinct $x_0, \ldots, x_n \in \mathbb{R}$ and $y_0, \ldots, y_n \in \mathbb{R}$,
find $p = \sum_k a_k x^k \in \mathbb{R}_n[x]$ such that $p(x_i) = y_i$ for all i i.e.

$$V \cdot \mathbf{a} = \mathbf{y}$$

where V is a Vandermonde matrix.

We could invert this system using standard linear algebra algorithms.
This takes $O(n^3)$ operations using Gaussian elimination.

Section 2.3. Polynomial Interpolation - Linear System Solving

Given pairwise distinct $x_0, \ldots, x_n \in \mathbb{R}$ and $y_0, \ldots, y_n \in \mathbb{R}$,
find $p = \sum_k a_k x^k \in \mathbb{R}_n[x]$ such that $p(x_i) = y_i$ for all i i.e.

$$V \cdot \mathbf{a} = \mathbf{y}$$

where V is a Vandermonde matrix.

We could invert this system using standard linear algebra algorithms.
This takes $O\left(n^3\right)$ operations using Gaussian elimination.

In theory, best known complexity bound: $O\left(n^\theta\right)$ where $\theta \approx 2.3728596$
[Alman and Williams, 2021].

Section 2.3. Polynomial Interpolation - Linear System Solving

Given pairwise distinct $x_0, \ldots, x_n \in \mathbb{R}$ and $y_0, \ldots, y_n \in \mathbb{R}$,
find $p = \sum_k a_k x^k \in \mathbb{R}_n[x]$ such that $p(x_i) = y_i$ for all i i.e.

$$V \cdot \mathbf{a} = \mathbf{y}$$

where V is a Vandermonde matrix.

We could invert this system using standard linear algebra algorithms.
This takes $O\left(n^3\right)$ operations using Gaussian elimination.

In theory, best known complexity bound: $O\left(n^\theta\right)$ where $\theta \approx 2.3728596$
[Alman and Williams, 2021].

In practice, Strassen's algorithm: cost of $O\left(n^{\log_2 7}\right)$ operations,
$\log_2 7 \approx 2.8073$.

Section 2.3. Polynomial Interpolation - Linear System Solving

There are issues with this approach, though:

- the problem is ill-conditioned: a small perturbation on the y_i leads to a significant perturbation of the solution.

Section 2.3. Polynomial Interpolation - Linear System Solving

There are issues with this approach, though:

- the problem is ill-conditioned: a small perturbation on the y_i leads to a significant perturbation of the solution.
- we can do better from the complexity point of view: $O\left(n^2\right)$ or even $O(n \log^{O(1)} n)$ in general, $O\left(n \log n\right)$ if the x_i are so-called *Chebyshev nodes*;

Section 2.3. Polynomial interpolation. Evaluation in the monomial basis

Evaluation cost of $p(x) = \sum_{k=0}^{n} a_k x^k$.

Section 2.3. Polynomial interpolation. Evaluation in the monomial basis

Evaluation cost of $p(x) = \sum_{k=0}^{n} a_k x^k$.

Horner's method, which relies on the writing

$$p(x) = (\cdots(((a_n x + a_{n-1})x + a_{n-2})x + a_{n-3})\cdots)x + a_0,$$

yields a $O(n)$ complexity.

Section 2.3. Polynomial interpolation: divided differences

The divided-difference method.

Newton's *divided-difference method*: compute interpolation polynomials incrementally.

Section 2.3. Polynomial interpolation: divided differences

The divided-difference method.

Newton's *divided-difference method*: compute interpolation polynomials incrementally.

Let $p_k \in \mathbb{R}_k [x]$ be such that $p_k (x_i) = y_i$ for $0 \leqslant i \leqslant k < n$, and write

$$p_{k+1} (x) = p_k (x) + a_{k+1} (x - x_0) \cdots (x - x_k) .$$

Section 2.3. Polynomial interpolation: divided differences

The divided-difference method.

Newton's *divided-difference method*: compute interpolation polynomials incrementally.

Let $p_k \in \mathbb{R}_k[x]$ be such that $p_k(x_i) = y_i$ for $0 \leqslant i \leqslant k < n$, and write

$$p_{k+1}(x) = p_k(x) + a_{k+1}(x - x_0) \cdots (x - x_k).$$

Given $y_0, \ldots, y_k$, we denote by $[y_0, \ldots, y_k]$ the corresponding a_k: Then, we can compute a_k using the relation

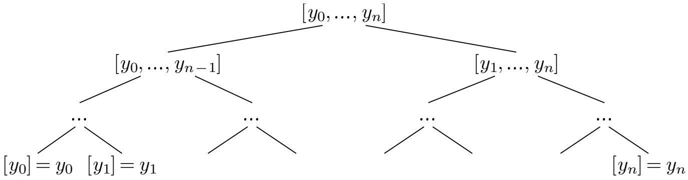
$$[y_0, \ldots, y_{k+1}] = \frac{[y_1, \ldots, y_{k+1}] - [y_0, \ldots, y_k]}{x_{k+1} - x_0}.$$

Section 2.3. Polynomial Interpolation: divided differences

Given $y_0, \ldots, y_k$, we denote by $[y_0, \ldots, y_k]$ the corresponding a_k: Then, we can compute a_k using the relation

$$[y_0, \ldots, y_{k+1}] = \frac{[y_1, \ldots, y_{k+1}] - [y_0, \ldots, y_k]}{x_{k+1} - x_0}.$$

This leads to a tree of the following shape.



Hence, the cost for computing the coefficients is in $O(n^2)$ operations.

The evaluation cost at a given point z is in $O(n)$ operations in $\mathbb{R}$: we can adapt Horner's scheme as

$$p(z) = (\cdots(((a_n(z - x_{n-1}) + a_{n-1})(z - x_{n-2}) + a_{n-2})(z - x_{n-3}) + a_{n-3}) \cdots)(z - x_0) + a_0.$$

Section 2.3. Polynomial interpolation: Lagrange interpolation

Lagrange's Formula.

For all $j = 0, \ldots, n$, let

$$\ell_j(x) = \prod_{k \neq j} \frac{x - x_k}{x_j - x_k}.$$

Then we have $\deg \ell_j = n$ and $\ell_j(x_i) = \delta_{i,j}$ for all $0 \leqslant i, j \leqslant n$.

Section 2.3. Polynomial interpolation: Lagrange interpolation

Lagrange's Formula.

For all $j = 0, \ldots, n$, let

$$\ell_j(x) = \prod_{k \neq j} \frac{x - x_k}{x_j - x_k}.$$

Then we have $\deg \ell_j = n$ and $\ell_j(x_i) = \delta_{i,j}$ for all $0 \leqslant i, j \leqslant n$.

$\{\ell_j\}_{0 \leqslant j \leqslant n}$ is basis of $\mathbb{R}_n[x]$.

Section 2.3. Polynomial interpolation: Lagrange interpolation

Lagrange's Formula.

For all $j = 0, \ldots, n$, let

$$\ell_j(x) = \prod_{k \neq j} \frac{x - x_k}{x_j - x_k}.$$

Then we have $\deg \ell_j = n$ and $\ell_j(x_i) = \delta_{i,j}$ for all $0 \leqslant i, j \leqslant n$.

$\{\ell_j\}_{0 \leqslant j \leqslant n}$ is basis of $\mathbb{R}_n[x]$.

The interpolation polynomial p:

$$p(x) = \sum_{i=0}^{n} y_i \ell_i(x).$$

Thus, writing the interpolation polynomial on the Lagrange basis is straightforward.

Section 2.3. Polynomial Interpolation - Lagrange Formula

Let $p(x) = \sum_{i=0}^{n} y_i \ell_i(x)$.

Evaluation cost?
Naively, computing $\ell_j(z)$ costs (say) $2n$ subtractions, $2n+1$ multiplications and 1 division.

The total cost is $O(n^2)$ operations in $\mathbb{R}$.

But we can also write

$$p(x) = W(x) \sum_{i=0}^{n} \frac{y_i}{(x - x_i)W'(x_i)}, \qquad W(x) = \prod_{i=0}^{n} (x - x_i).$$

Assuming the $W'(x_i)$ are precomputed, the cost of evaluating $p(z)$ using this formula is only $O(n)$ arithmetical operations.

But we can also write

$$p(x) = W(x) \sum_{i=0}^{n} \frac{y_i}{(x - x_i) W'(x_i)}, \qquad W(x) = \prod_{i=0}^{n} (x - x_i).$$

Assuming the $W'(x_i)$ are precomputed, the cost of evaluating $p(z)$ using this formula is only $O(n)$ arithmetical operations.

Evaluation can be a tricky issue: not only a problem of speed but also of numerical stability. The notion of "barycentric Lagrange interpolation" is quite relevant regarding these stability issues (see Trefethen's "Approximation Theory and Approximation Practice").

Section 2.4. Interpolation and approximation, Chebyshev polynomials

How useful is interpolation for our initial L^∞ approximation problem?

Section 2.4. Interpolation and approximation, Chebyshev polynomials

How useful is interpolation for our initial L^∞ approximation problem?

It turns out that the choice of the points is critical. The more points, the better?

Section 2.4. Interpolation and approximation, Chebyshev polynomials

Exercise

Using your computer algebra system of choice, interpolate the function

$$f : x \mapsto \frac{1}{1+5x^2}$$

at the points $-1 + \frac{2k}{n}$, $0 \leqslant k \leqslant n$, *for* $n = 10, 15, \dots, 30$. *Compare with* f *on* $[-1, 1]$.

Section 2.4. Interpolation and approximation, Chebyshev polynomials

Exercise

Using your computer algebra system of choice, interpolate the function

$$f : x \mapsto \frac{1}{1+5x^2}$$

at the points $-1 + \frac{2k}{n}$, $0 \leqslant k \leqslant n$, *for* $n = 10, 15, \ldots, 30$. *Compare with* f *on* $[-1, 1]$.

In short: never use equidistant points when approximating a function by interpolation!

Theorem

(Faber)

For each n*, let a system of* $n+1$ *distinct nodes* $\xi_0^{(n)}, \ldots, \xi_n^{(n)} \in [a,b]$.

Then for some $f \in \mathcal{C}([a,b])$*, the sequence of errors* $(||f - p_n||_\infty)_{n\in\mathbb{N}}$ *is unbounded, where* $p_n \in \mathbb{R}_n[x]$ *denote the polynomial which interpolates* f *at the* $\xi_0^{(n)}, \ldots, \xi_n^{(n)}$.

Section 2.4. Interpolation and approximation, Chebyshev polynomials

Theorem

(Faber)
For each n*, let a system of* $n+1$ *distinct nodes* $\xi_0^{(n)}, \ldots, \xi_n^{(n)} \in [a, b]$*.*

Then for some $f \in \mathcal{C}([a, b])$*, the sequence of errors* $(||f - p_n||_\infty)_{n \in \mathbb{N}}$ *is unbounded, where* $p_n \in \mathbb{R}_n[x]$ *denote the polynomial which interpolates* f *at the* $\xi_0^{(n)}, \ldots, \xi_n^{(n)}$*.*

How depressing!

Section 2.4. Interpolation and approximation, Chebyshev polynomials

Theorem

(Faber)
For each n, let a system of $n+1$ distinct nodes $\xi_0^{(n)}, \dots, \xi_n^{(n)} \in [a,b]$.

Then for some $f \in \mathcal{C}([a,b])$, the sequence of errors $(||f - p_n||_\infty)_{n\in\mathbb{N}}$ is unbounded, where $p_n \in \mathbb{R}_n[x]$ denote the polynomial which interpolates f at the $\xi_0^{(n)}, \dots, \xi_n^{(n)}$.

How depressing!
Hmmm... Really?

Theorem

(Faber)

For each n, let a system of $n+1$ distinct nodes $\xi_0^{(n)}, \dots, \xi_n^{(n)} \in [a,b]$.

Then for some $f \in \mathcal{C}([a,b])$, the sequence of errors $(||f - p_n||_\infty)_{n\in\mathbb{N}}$ is unbounded, where $p_n \in \mathbb{R}_n[x]$ denote the polynomial which interpolates f at the $\xi_0^{(n)}, \dots, \xi_n^{(n)}$.

How depressing!
Hmmm... Really?
There is always hope!

Section 2.4. Interpolation and approximation, Chebyshev polynomials

Theorem 10

Let $a < x_0 < \cdots < x_n < b$*, and let* $f \in \mathcal{C}^{n+1}([a,b])$*. Let* $p \in \mathbb{R}_n[x]$ *be such that* $f(x_i) = p(x_i)$ *for all* i*. Then, for all* $x \in [a,b]$*, there exists* $\xi_x \in (a,b)$ *such that*

$$f(x) - p(x) = \frac{f^{(n+1)}(\xi_x)}{(n+1)!} W(x), \qquad W(x) = \prod_{i=0}^{n} (x - x_i).$$

Section 2.4. Interpolation and approximation, Chebyshev polynomials

Search for families of x_i which make $\|W\|_\infty$ as small as possible.

Section 2.4. Interpolation and approximation, Chebyshev polynomials

Search for families of x_i which make $\|W\|_\infty$ as small as possible.

Assume $[a, b] = [-1, 1]$. The n-th Chebyshev polynomial of the first kind is defined by

$$T_n(\cos t) = \cos(nt), \forall t \in [0, 2\pi].$$

.

The T_n can also be defined by

$$T_0(x) = 1, T_1(x) = x, T_{n+2}(x) = 2xT_{n+1}(x) - T_n(x), \forall n \in \mathbb{N}.$$

Proposition

The minimum value of the set

$$\left\{ \|p\|_{\infty,[-1,1]} : p \in \mathbb{R}_n[x], \deg P = n, \mathrm{lc}(p) = 1 \right\}$$

is uniquely attained for $T_n/2^{n-1}$ and is therefore equal to 2^{-n+1}.

Forcing $W(x) = 2^{-n}T_{n+1}(x)$ leads to the interpolation points

$$\mu_k = \cos\left(\frac{(2k+1)\pi}{2(n+1)}\right), k = 0, \dots, n,$$

called the Chebyshev nodes of the first kind.

Another important family is that of Chebyshev polynomials of the second kind $U_n(x)$, defined by

$$U_n(\cos x) = \frac{\sin((n+1)x)}{\sin(x)}.$$

They can also be defined by

$$U_0(x) = 1, U_1(x) = 2x, U_{n+2}(x) = 2xU_{n+1}(x) - U_n(x), \forall n \in \mathbb{N}.$$

For all $n \geqslant 0$, we have $\frac{\mathrm{d}}{\mathrm{d}x} T_n = nU_{n-1}$.

So the extrema of T_{n+1} are -1, 1 and the zeros of U_n, that is,

$$\nu_k = \cos\left(\frac{i\pi}{n}\right), k = 0, \ldots, n,$$

called the Chebyshev nodes of the second kind.

With $W(x) = 2^{-n+1}\left(1 - x^2\right) U_{n-1}(x)$, we have $\|W\|_\infty \leqslant 2^{-n+1}$.

We have $\deg T_n = \deg U_n = n$ for all $n \in \mathbb{N}$.

Therefore, $(T_k)_{0 \leqslant k \leqslant n}$ is a basis of $\mathbb{R}_n[x]$.

We have $\deg T_n = \deg U_n = n$ for all $n \in \mathbb{N}$.

Therefore, $(T_k)_{0 \leqslant k \leqslant n}$ is a basis of $\mathbb{R}_n[x]$.

Now, we give results that allow for (fast) computing the coefficients of interpolation polynomials, at the Chebyshev nodes, expressed in the basis $(T_k)_{0 \leqslant k \leqslant n}$.