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 $i = 0, \ldots, n$.

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,

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,

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 $O(n^3)$ operations using Gaussian elimination.

In theory, best known complexity bound: $O(n^\theta)$ where $\theta \approx 2.3728639$
(Le Gall, 2014).

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 $\theta \approx 2.3728639$
(Le Gall, 2014).

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

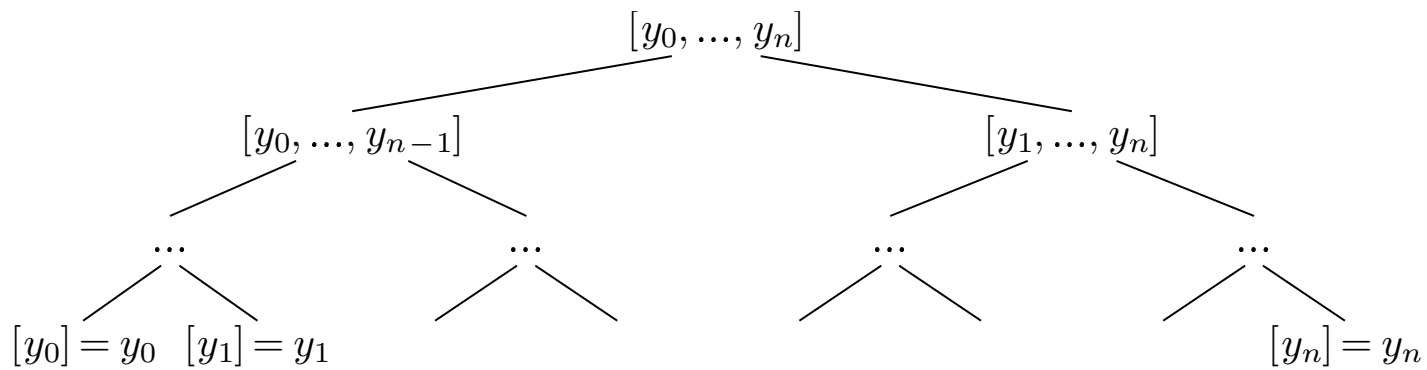
$$[y_0, \ldots, y_{k+1}] = \frac{[y_1, \ldots, y_{k+1}] - [y_0, \ldots, y_k]}{x_{k+1} - x_0}.$$

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.

Section 2.3. Polynomial Interpolation: divided differences

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 $O\left(n^2\right)$ operations in $\mathbb{R}$.

Section 2.3. Polynomial Interpolation - Lagrange Formula

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.

Section 2.3. Polynomial Interpolation - Lagrange Formula

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* $n = 10, 15, \ldots, 30$. *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!

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 $\xi_0^{(n)}, \ldots, \xi_n^{(n)}$.

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 $\xi_0^{(n)}, \ldots, \xi_n^{(n)} \in [a,b]$.

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!
Hmmm... Really?

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!
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

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

Section 2.4. Interpolation and approximation, Chebyshev polynomials

Proposition

The minimum value of the set

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

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

Section 2.4. Interpolation and approximation, Chebyshev polynomials

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

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

called the Chebyshev nodes of the first kind.

Section 2.4. Interpolation and approximation, Chebyshev polynomials

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 $\|W\|_\infty \leqslant 2^{-n+1}$.

Section 2.4. Interpolation and approximation, Chebyshev polynomials

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 $\mathbb{R}_n[x]$.

Section 2.4. Interpolation and approximation, Chebyshev polynomials

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 $(T_k)_{0 \leqslant k \leqslant n}$.

Section 2.4. Interpolation and approximation, Chebyshev polynomials

Proposition

(Discrete orthogonality.)

1. *We have*

$$\sum_{k=0}^{n} T_i(\mu_k) T_j(\mu_k) = \begin{cases} 0, & i \neq j, \\ n+1, & i = j = 0, \\ \frac{n+1}{2}, & i = j \neq 0. \end{cases}$$

2. *We have*

$$\sum_{k=0}^{n} T_i(\nu_k) T_j(\nu_k) = \begin{cases} 0, & i \neq j, \\ n, & i = j \in \{0, n\}, \\ \frac{n}{2}, & i = j \notin \{0, n\}. \end{cases}$$

Section 2.4. Interpolation and approximation, Chebyshev polynomials

$\sum'$denotes that the first term of the sum has to be halved, $\sum''$denotes that the first and the last terms of the sum have to be halved.

Proposition

1. *If* $p_{1,n} = \sum'_{0 \leqslant i \leqslant n} c_{1,i} T_i \in \mathbb{R}_n [x]$ *interpolates* f *on the set* $\{\mu_k : 0 \leqslant k \leqslant n\}$, *then*

$$c_{1,i} = \frac{2}{n+1} \sum_{k=0}^{n} f(\mu_k) T_i(\mu_k).$$

2. *Likewise, if* $p_{2,n} = \sum''_{0 \leqslant i \leqslant n} c_{2,i} T_i$ *interpolates* f *at* $\{\nu_k : 0 \leqslant k \leqslant n\}$, *then*

$$c_{2,i} = \frac{2}{n} \sum_{k=0}^{n} f(\nu_k) T_i(\nu_k).$$

Cost: $O(n^2)$ operations.

Section 2.5. Clenshaw's method for evaluating Chebyshev sums

Given coefficients $c_0, \ldots, c_N$ and a point t, we would like to compute the sum

$$\sum_{k=0}^{N} c_k T_k(t).$$

Recall that the polynomials T_k satisfy $T_{k+2}(x) = 2xT_{k+1}(x) - T_k(x)$.

Section 2.5. Clenshaw's method for evaluating Chebyshev sums

Given coefficients $c_0, \ldots, c_N$ and a point t, we would like to compute the sum

$$\sum_{k=0}^{N} c_k T_k(t).$$

Recall that the polynomials T_k satisfy $T_{k+2}(x) = 2xT_{k+1}(x) - T_k(x)$.

First idea: use this relation to compute the $T_k(t)$ that appear in the sum.

Section 2.5. Clenshaw's method for evaluating Chebyshev sums

Given coefficients $c_0, \ldots, c_N$ and a point t, we would like to compute the sum

$$\sum_{k=0}^{N} c_k T_k(t).$$

Recall that the polynomials T_k satisfy $T_{k+2}(x) = 2xT_{k+1}(x) - T_k(x)$.

First idea: use this relation to compute the $T_k(t)$ that appear in the sum.

Method is numerically unstable.
The $U_k(x)$ satisfy the same recurrence but grows faster: we have

$$\|T_k\|_\infty = 1, \qquad \|U_k\|_\infty = k+1.$$

Section 2.5. Clenshaw's method for Chebyshev sums

Algorithm 2

Input Chebyshev coefficients $c_0, \ldots, c_N$, a point t

Output $\sum_{k=0}^{N} c_k T_k(t)$

1. $b_{N+1} \leftarrow 0$, $b_N \leftarrow c_N$
2. for $k = N-1, N-2, \ldots, 1$
   1. $b_k \leftarrow 2tb_{k+1} - b_{k+2} + c_k$
3. return $c_0 + tb_1 - b_2$

This algorithm runs in $O(N)$ arithmetic operations.

Section 2.6. Computation of the Chebyshev coefficients

How do we compute the c_k?
Assume we use the Chebyshev nodes of the second kind and obtain the result on the Chebyshev basis.

Given $y_0, \ldots, y_N$, we are looking for $c_0, \ldots, c_N$ such that $p(x) = \sum_{j=0}^{N}{}'' c_j T_j(x)$ satisfies $p(\nu_k) = y_k$ for all k.
By discrete orthogonality, we have

$$c_j = \frac{2}{N} \sum_{k=0}^{N} y_k T_k(\nu_j).$$

Observe that we have

$$T_k(\nu_j) = \cos\left(jk\frac{\pi}{N}\right)$$

.

Section 2.6. Computation of the Chebyshev coefficients

How do we compute the c_k?
Assume we use the Chebyshev nodes of the second kind and obtain the result on the Chebyshev basis.

Given $y_0, \ldots, y_N$, we are looking for $c_0, \ldots, c_N$ such that $p(x) = \sum_{j=0}^{N}{}'' c_j T_j(x)$ satisfies $p(\nu_k) = y_k$ for all k.
By discrete orthogonality, we have

$$c_j = \frac{2}{N} \sum_{k=0}^{N} y_k T_k(\nu_j).$$

Observe that we have

$$T_k(\nu_j) = \cos\left(jk\frac{\pi}{N}\right)$$

.
DCT: Discrete Cosine Transform. JPEG, MPEG, MP3, etc.

Approximation Theory and Proof Assistants: Certified Computations

Nicolas Brisebarre and Damien Pous

Master 2 Informatique Fondamentale
École Normale Supérieure de Lyon, 2020-2021

Chapter 2. Orthogonal Polynomials - Chebyshev series

Section 2.1. Orthogonal Polynomials

Let $(a, b) \subset \mathbb{R}$ be an open interval, and let w be a weight function, that is to say $w : (a, b) \to (0, \infty)$ is a continuous function. We assume

$$\forall n \in \mathbb{N}, \quad \int_a^b |x|^n w(x) \mathrm{d}x < \infty.$$

Section 2.1. Orthogonal Polynomials

Let $(a, b) \subset \mathbb{R}$ be an open interval, and let w be a weight function, that is to say $w : (a, b) \to (0, \infty)$ is a continuous function. We assume

$$\forall n \in \mathbb{N}, \quad \int_a^b |x|^n w(x) \mathrm{d}x < \infty.$$

This is the case, for instance, if (a, b) is bounded and

$$\int_a^b w(x) \mathrm{d}x < \infty.$$

Section 2.1. Orthogonal Polynomials

Let

$$\mathcal{E}(\omega) = \left\{ f \in \mathcal{C}((a,b)) : \|f\|_2 := \left(\int_a^b f(x)^2 w(x) \mathrm{d}x \right)^{1/2} < \infty \right\}.$$

Section 2.1. Orthogonal Polynomials

Let

$$\mathcal{E}(\omega) = \left\{ f \in \mathcal{C}((a,b)) : \|f\|_2 := \left(\int_a^b f(x)^2 w(x) \mathrm{d}x \right)^{1/2} < \infty \right\}.$$

Observe that $\mathbb{R}[x] \subset \mathcal{E}(\omega)$. The space $\mathcal{E}(\omega)$ is equipped with an inner product

$$\langle f, g \rangle = \int_a^b f(x) g(x) w(x) \mathrm{d}x;$$

and $\| \cdot \|_2$ is the norm associated to this inner product.

Section 2.1. Orthogonal Polynomials

Definition 1

A family of orthogonal polynomials associated with w is a sequence $(p_n) \in \mathbb{R}[x]^{\mathbb{N}}$ where $\deg p_k = k$ for all k, and

$$i \neq j \quad \Rightarrow \quad \langle p_i, p_j \rangle = 0.$$

Section 2.1. Orthogonal Polynomials

Definition 1

A family of orthogonal polynomials associated with w is a sequence $(p_n) \in \mathbb{R}[x]^{\mathbb{N}}$ where $\deg p_k = k$ for all k, and

$$i \neq j \quad \Rightarrow \quad \langle p_i, p_j \rangle = 0.$$

Theorem 2

For any weight w, there exists a family of orthogonal polynomials associated with w. If additionally we request that the p_k are all monic, this family is unique.

Section 2.1. Orthogonal Polynomials

Definition 1

A family of orthogonal polynomials associated with w is a sequence $(p_n) \in \mathbb{R}[x]^{\mathbb{N}}$ where $\deg p_k = k$ for all k, and

$$i \neq j \quad \Rightarrow \quad \langle p_i, p_j \rangle = 0.$$

Theorem 2

For any weight w, there exists a family of orthogonal polynomials associated with w. If additionally we request that the p_k are all monic, this family is unique.

Gram-Schmidt orthogonalization process