Polynomial approximation with size-constrained coefficients

Tom Hubrecht,
with Nicolas Brisebarre, Sylvain Chevillard, Guillaume Hanrot, Serge Torres

Jeu. 26 juin 2025: NuSCAP Nantes

What is it about

- Polynomial approximation: Given $f : I \subset \mathbb{R} \longrightarrow \mathbb{R}$, find $P \in \mathbb{R}_n[X]$ “close” to f

What is it about

- Polynomial approximation: Given $f : I \subset \mathbb{R} \longrightarrow \mathbb{R}$, find $P \in \mathbb{R}_n[X]$ "close" to f

- Size-constrained coefficients: That can be represented on some finite amount of memory (e.g. 64 bits)

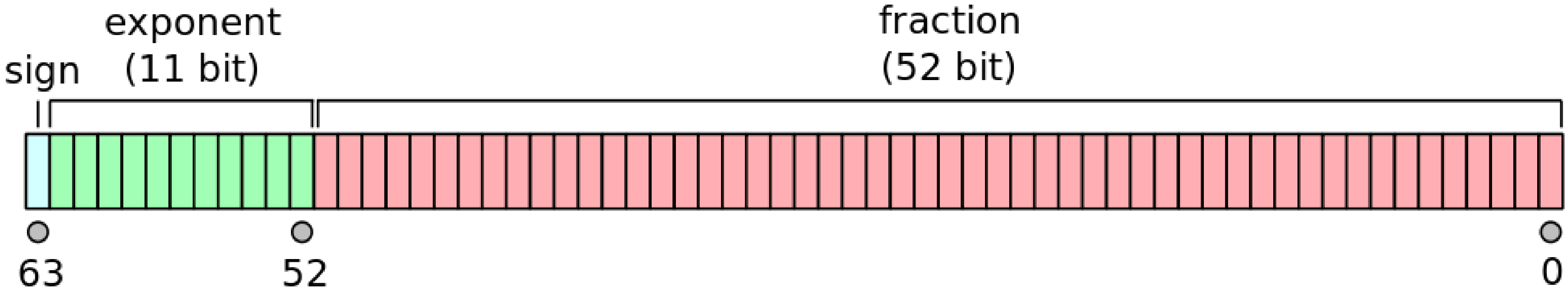
But why ?

Numerical evaluation of functions

We want:

- To evaluate numerically various mathematical functions
- Use computers to do the work

Limited precision of machine numbers



$$\hat{x} = (-1)^s \times 2^e \times 1.f$$

$$\log(2) \approx 0.693147180559945 \quad \mathrm{o}\big(\log(2)\big) = \texttt{0x1.62e42fefa39efp-1}$$

Where $\mathrm{o}(x)$ is the closest floating-point number to x

Some basic arithmetic operations

The operations at our disposal are: $+, \quad \times, \quad -, \quad \sqrt{x}, \quad /$

We need to use approximations to compute numerical values of functions.

In most cases, we work with polynomial approximations:

$$\exp(z) \approx a_0 + z \times \left(a_1 + z \times \left(a_2 + z \times \left(a_3 + z \times a_4\right)\right)\right)$$

Mathematical Libraries

- All programs use libraries: sets of (mostly) standard functions to avoid reinventing the wheel (and making mistakes).

- To compute mathematical functions, there are "libm"s implementing `exp`, `log`, `sin`, …
- List of mathematical functions defined in standards as IEEE754, ISO/IEC 9899
- Several of them coexist: `glibc`, `LLVM` math library, `CORE-MATH`, …

Libm constraints

- Speed is a big requirement, those functions will be used more than 100M times
- Accuracy varies and is not always defined

Correct Rounding

The evaluation $\hat{f}$ of a function f is correctly rounded is $\hat{f}(x)$ is the closest floating-point value to $f(x)$.

It is necessary in multiple domains :

- Distributed computations, HPC
- Any application requiring reproducible results

But, it is a much harder property to guarantee than, e.g., “52 bits of precision” out of the 53 bits of doubles

Building a libm function

Three steps are usually observed:

1. Range reduction: go from $\mathbb{R}$ to I a small segment for the inputs
   - Using various equalities: e.g. $\log\left(2^k x\right) = \log(x) + k \times \log(2)$
2. Use a polynomial approximation of f over I
3. Reconstruct the final result

- If Correct Rounding is required, this may be done several times with increasing precision

Example: x^y in CORE-MATH

- A “library” of correctly-rounded functions[1]
- Computed as $\exp\left(y \times \log(x)\right)$
- Three phases to attain Correct Rounding
- Requires 6 polynomial approximations in total

[1] https://core-math.gitlabpages.inria.fr/

Polynomial Approximation

Core Problem

In the end, it is the foundation of numerical evaluation, and needs to be:

- Fast, as it is in the critical path
- Accurate, to not have to redo computations

I.e. we want a polynomial with the smallest number of coefficients possible while maintaining a necessary accuracy.

Accuracy, i.e. relative error

What does “q bits of precision” mean ?

For an approximation P of f over $I = [a, b] \subset \mathbb{R}$:

- Absolute error: $\|P - f\|_\infty \coloneqq \max_{x \in I} \left|P(x) - f(x)\right|$
- Relative error: $\left\|\frac{P-f}{f}\right\|_\infty \coloneqq \max_{x \in I} \left|\frac{P(x)-f(x)}{f(x)}\right|$

Thus, “q bits of precision” means a relative error smaller than 2^{-q}

Generalized polynomials

- Real polynomial: $Q \coloneqq \sum a_i x^i$ with $a_i \in \mathbb{R}$, $\Longrightarrow$ used when minimizing absolute errors, but not enough for relative errors.
- Generalized polynomial: $G \coloneqq \sum a_i \varphi_i, \quad$ with $\varphi_i : \mathbb{R} \longrightarrow \mathbb{R}$
- Special case: $\sum a_i \frac{x^i}{f}$, $\Longrightarrow$ used when minimizing the relative error, with the target $x \mapsto 1$

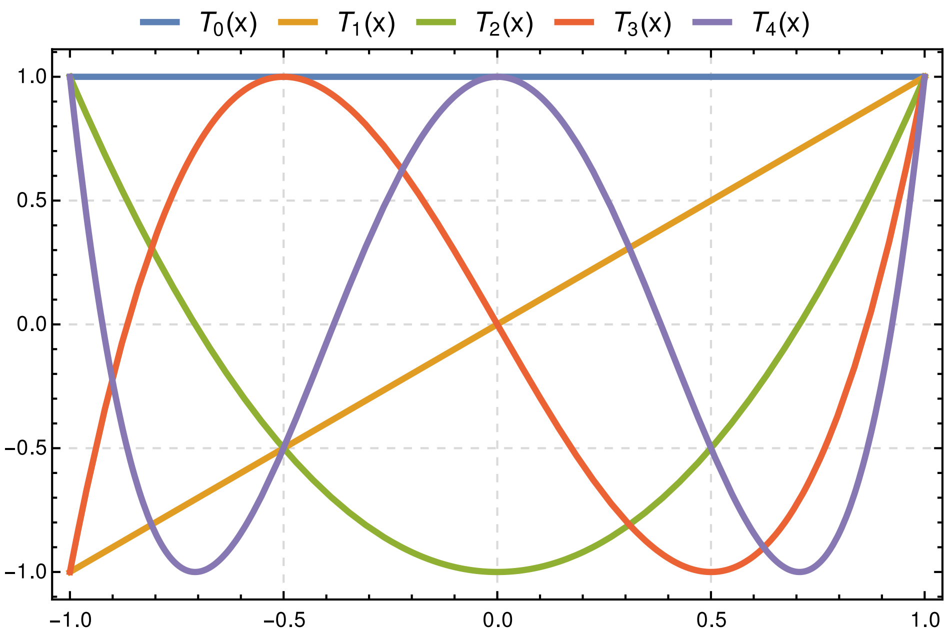
Best Approximation: Minimax

For real polynomials, a minimax approximation $p^\star$ of f over $I \subset \mathbb{R}$ of degree $n \in \mathbb{N}$ is the polynomial $P \in \mathbb{R}_n[x]$ that minimizes the absolute error.

Under some conditions, it is the same using generalized polynomials.

As a non-linear problem, we have an iterative algorithm to solve it.

Chebyshev polynomial of the first kind



- $T_n(\cos(\theta)) = \cos(n\theta)$
- Orthogonal family
- $T_n^{-1}(0) = \left\{\cos\left(\frac{2k+1}{2n}\pi\right) : k \in [\![0, n-1]\!]\right\}$

Non-linear minimax, with linear approximation of the problem

Computing the minimax is a non-linear problem, that can be approximated by linear ones.

- Optimal: minimax polynomial
- Truncated Chebyshev Series or Interpolation polynomial at the Chebyshev nodes of first kind are “good approximations”

Lebesgue constant

$$\Lambda(L) \coloneqq \sup_{f} \frac{\|Lf\|_\infty}{\|f\|_\infty}$$

- For a linear approximation L, and $p^\star$ the minimax, $\|f - Lf\|_\infty \leq (1 + \Lambda(L))\,\|f - p^\star\|_\infty$
- Truncated Chebyshev series of degree $n > 1$: $\frac{4}{\pi^2}\log(n-1) + 3 > \Lambda\left(\mathrm{TCS}_n\right) \geq \frac{4}{\pi^2}\log(n+1)$
- Interpolation of degree n : $\frac{2}{\pi}\log(n+1) + 1 \geq \Lambda\left(\mathrm{I}_n\right) \geq \frac{2}{\pi}\left(\log(n+1) + \gamma + \log\left(\frac{4}{\pi}\right)\right)$

L^2 projections

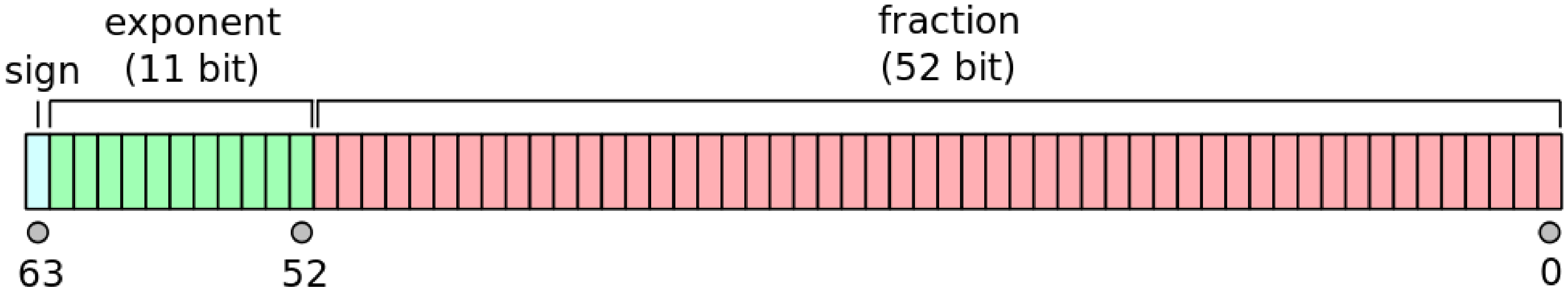
In the following, $I \coloneqq [-1, 1]$ (up to a linear change of variable)

The truncated Chebyshev series of degree n is the orthogonal projection of f onto the subspace $\mathrm{Span}(1, x, \ldots, x^n)$ for the inner product $\langle f, g \rangle \coloneqq \int_{-1}^{1} f g \frac{\mathrm{d}x}{\sqrt{1-x^2}}$

Therefore, we can approximate the non-linear minimization problem by a projection in some L^2 function space.

Machine-efficient polynomials

Recap on limited precision



Without a stroke of luck, real coefficients of polynomial approximations are not representable as floating-point numbers of fixed precision.

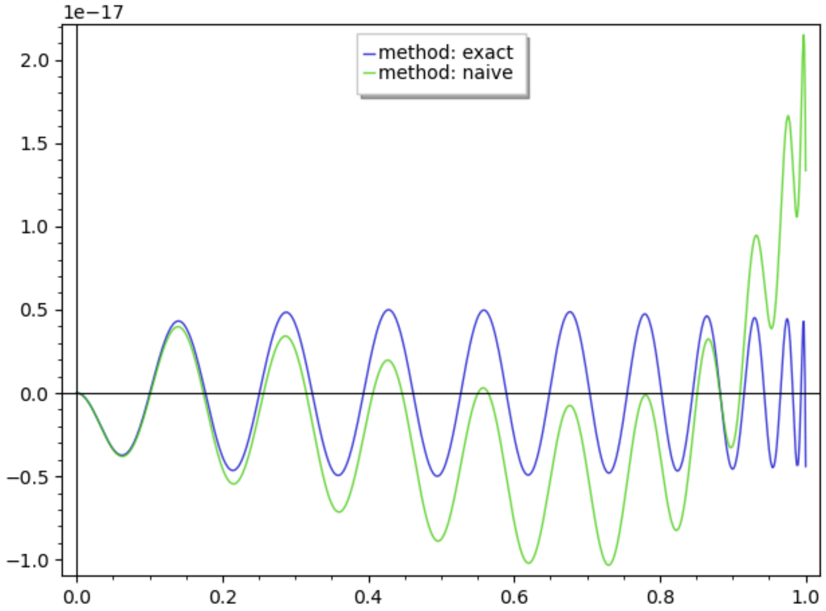
Practical example: $\arctan$ over $[-1, 1]$

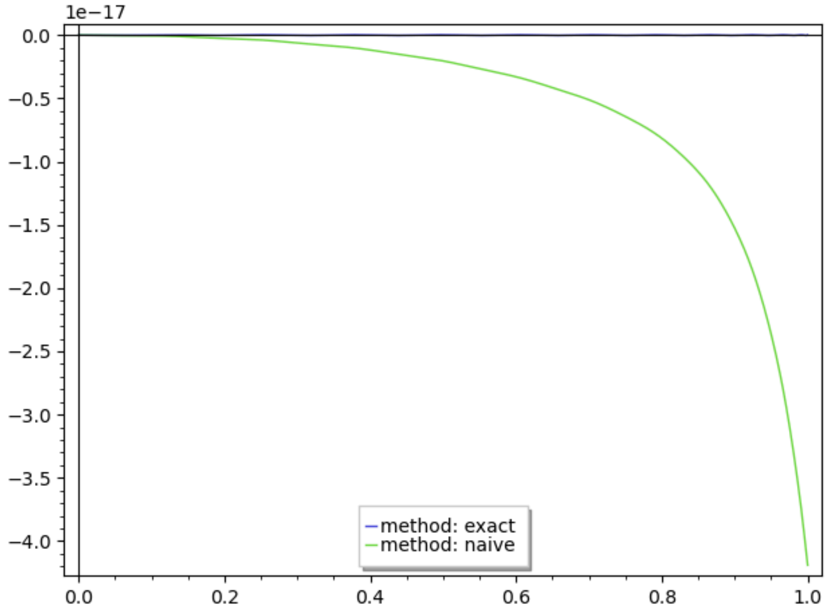
Taken from “Scientific Computing on Itanium-Based Systems”[2]

- Odd function $\implies$ only consider odd powers of x
- Pin the first coefficient to 1 (save a multiplication)
- Use the symmetry to approximate over $[0, 1]$ instead
- Minimizing the relative error

[2] M. Cornea and J. Harrison and P. T. P. Tang

- First idea: round each coefficient of the minimax (best approximation)





- Lose accuracy when increasing the degree (43 vs. 47)

Lack of a good structure for floating-point numbers

Floating-point numbers are of the form $2^{e_i} m_i$ with $m_i \in [\![2^{\boldsymbol{p-1}}, 2^p - 1]\!]$

- For the same exponent, the values are regularly placed on the reals
- But not when the exponent changes… $\Longrightarrow$ non linear set

Finding the best coefficients

For each coefficient, we need to find both e_i and m_i

- Finding both at the same time is tricky
- We first set e_i and then search for a corresponding m_i

Heuristically pinning the exponents

- Compute the projection with real coefficients $P = \sum a_i x^i$ and set $e_i \coloneqq \lfloor p_i - \log_2\left(|a_i|\right) \rfloor$
- Works when the precision is high enough (e.g. doubles)
- If it fails, adjust the exponents and start again

Closest Vector Problem

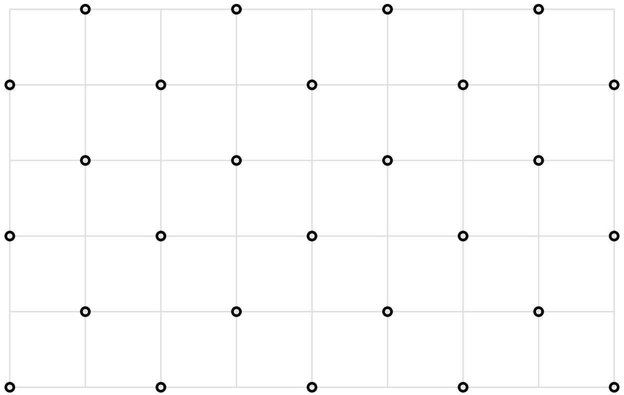
We look for an approximation of the form $P : x \mapsto \left(\sum_{i=0}^{n} m_i 2^{e_i} \cdot x^i\right), \quad |m_i| \in \mathbb{N} < 2^p - 1.$

- When e_i is set heuristically, we search for a vector of the lattice generated by $\left(2^{e_i} \cdot x^i\right)_{i \in [\![0,n]\!]}$ that is close to f.

- For relative error, use the basis $\left(2^{e_i} \cdot \frac{x^i}{f}\right)_{i \in [\![0,n]\!]}$ and the target $x \mapsto 1$

A Euclidean lattice is $L \coloneqq \mathrm{Span}_{\mathbb{Z}}\left(b_0, \ldots, b_n\right)$, for $\left(b_i\right)_{i \in [0,n]}$ a family of linearly independant vectors.

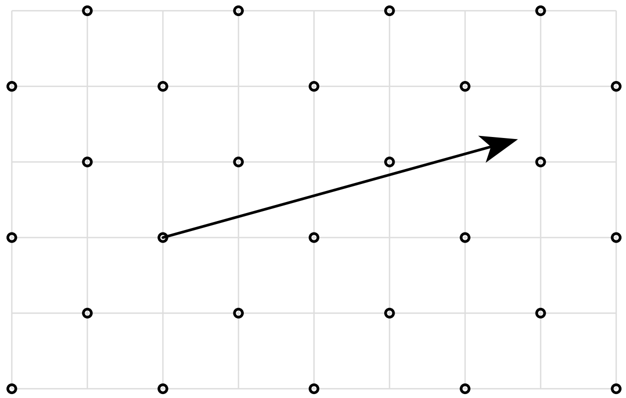
$E \supset L$ is a vector space.



The "Closest Vector Problem" is, for $x \in E$ and $\| \cdot \|$ a norm over E, to find $y \in L$ such that $\|x - y\|$ is small.

A Euclidean lattice is $L \coloneqq \mathrm{Span}_{\mathbb{Z}}\left(b_0, \ldots, b_n\right)$, for $\left(b_i\right)_{i \in [0,n]}$ a family of linearly independant vectors.

$E \supset L$ is a vector space.

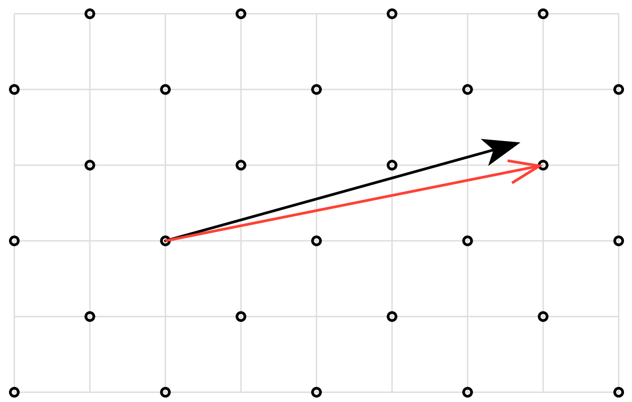


The "Closest Vector Problem" is, for $x \in E$ and $\|\cdot\|$ a norm over E, to find $y \in L$ such that $\|x - y\|$ is small.

Euclidean Lattices

A Euclidean lattice is $L \coloneqq \mathrm{Span}_{\mathbb{Z}}\left(b_0, \ldots, b_n\right)$, for $\left(b_i\right)_{i \in [0,n]}$ a family of linearly independant vectors.

$E \supset L$ is a vector space.



The “Closest Vector Problem” is, for $x \in E$ and $\|\cdot\|$ a norm over E, to find $y \in L$ such that $\|x - y\|$ is small.

The LLL algorithm (improving the basis)

For a generic basis, solving CVP or a polynomial approximation of it is hard.

[3] Nguyen, P.Q., Stehlé, D. (2006). LLL on the Average

The LLL algorithm (improving the basis)

For a generic basis, solving CVP or a polynomial approximation of it is hard.

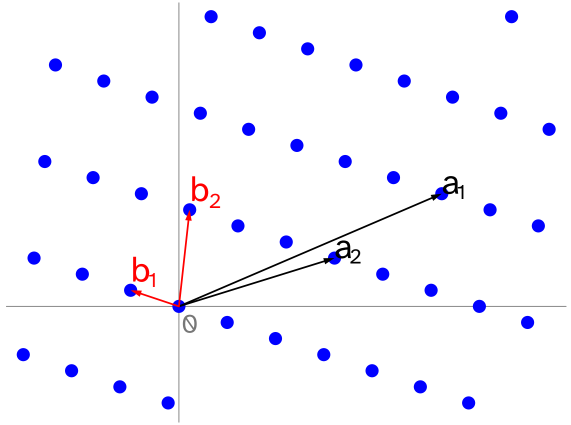
In a perfect world, $b_i = b_i^\star$ its orthogonalised vector.

[4]Nguyen, P.Q., Stehlé, D. (2006). LLL on the Average

The LLL algorithm (improving the basis)

For a generic basis, solving CVP or a polynomial approximation of it is hard.

In a perfect world, $b_i = b_i^\star$ its orthogonalised vector.



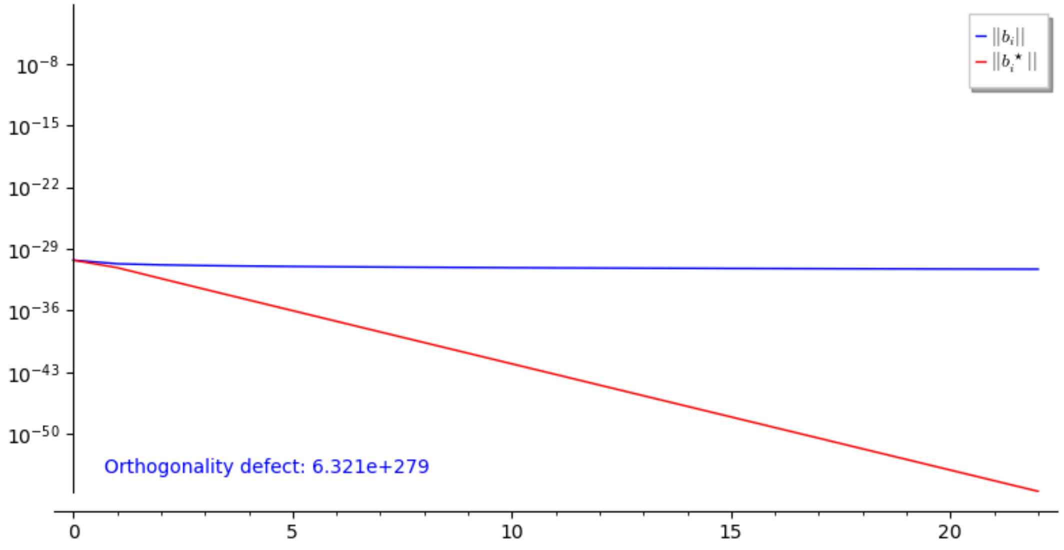
LLL algorithm: transforms $\left(a_0, \ldots, a_n\right)$ into $\left(b_0, \ldots, b_n\right)$ such that $\|b_1\| \leq 2^{\frac{n}{2}} \min_{x \in L}\left(\|x\|\right)$.

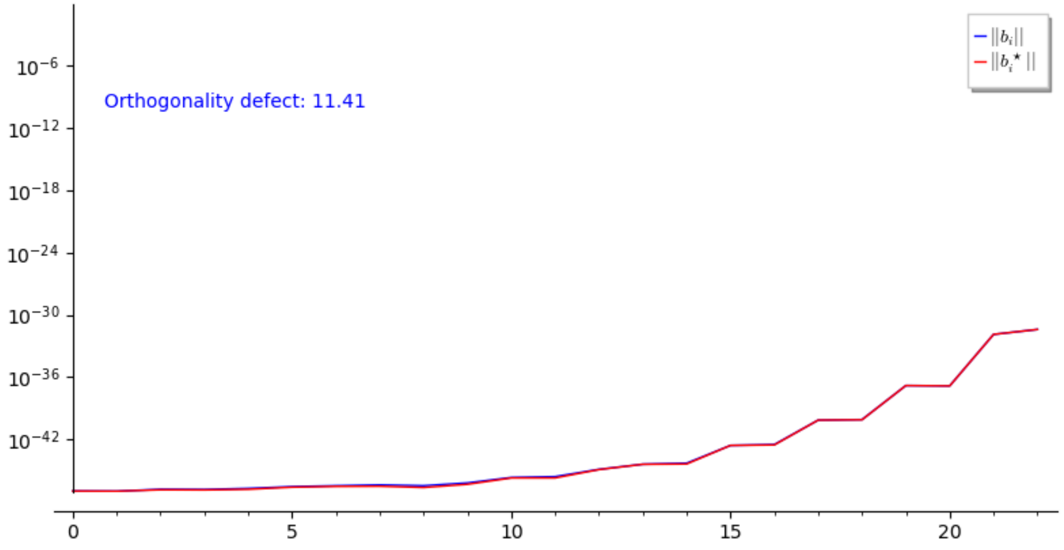
In our case, the basis is not average[5] and gives better results.

[5]Nguyen, P.Q., Stehlé, D. (2006). LLL on the Average

Polynomial bases are special

Starting Lattice basis $\underbrace{x^3}_{a_0}, \underbrace{x^5}_{a_1}, \dots \underbrace{x^{47}}_{a_{22}}$, and $a_0^\star, \dots a_{22}^\star$ the orthogonalized family, transformed into $(b_0, \dots, b_{22})$ and $(b_0^\star, \dots, b_{22}^\star)$





Orthogonality defect: measures how non-orthogonal the lattice basis is

Solving the Approximate CVP: Babai algorithms

When the basis is LLL-reduced, we have two algorithms at our disposal:

Solving the Approximate CVP: Babai algorithms

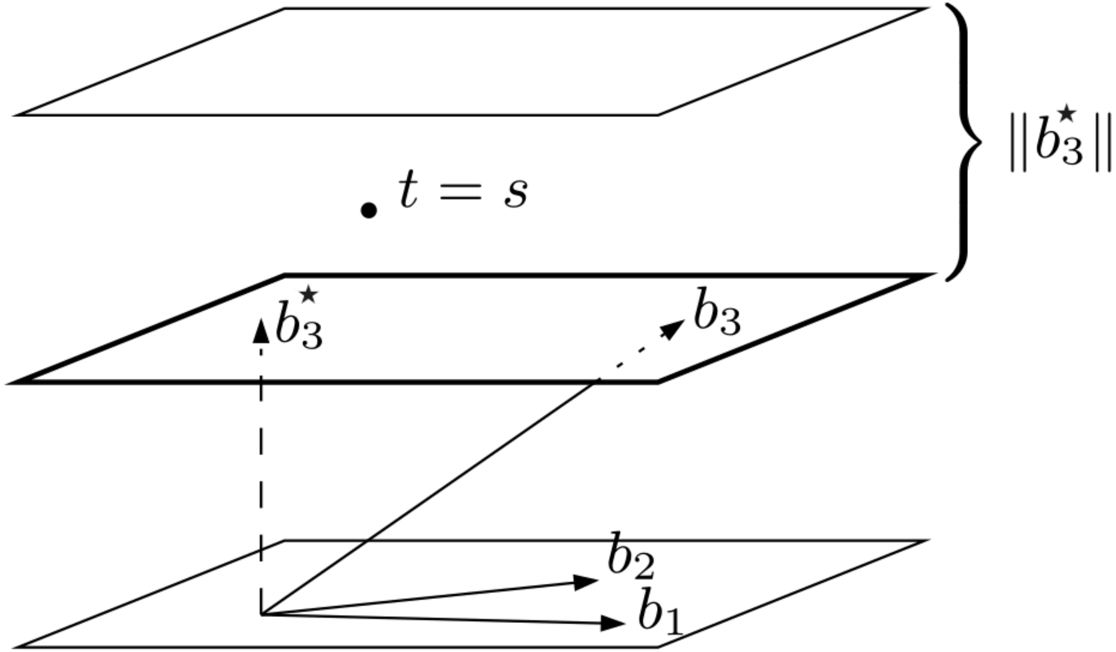
When the basis is LLL-reduced, we have two algorithms at our disposal:

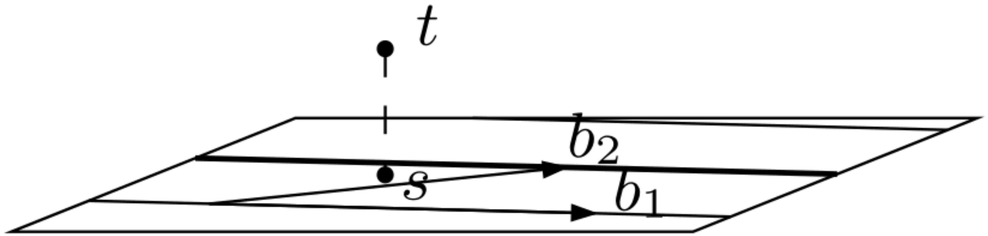
- Rounding Off: Express the vector in the new basis, and set each coordinate to its closest integer
-

Solving the Approximate CVP: Babai algorithms

When the basis is LLL-reduced, we have two algorithms at our disposal:

- Rounding Off: Express the vector in the new basis, and set each coordinate to its closest integer
- Nearest Plane: Iteratively project each coordinate, taking into account previous rounding errors





- In our case, both perform the same (reduced basis is near orthogonal)

Implementations

- State of the art: `fpminimax` in the Sollya[6] toolbox
- Newly revisited L^2 prototype

With the same global approach:

- Find a polynomial with real coefficients approximating f (minimax or projection)
- Explore the surroundings to find one with coefficients of the desired size

[6] https://sollya.org

Fpminimax: Discretization

- Take $d + 1$ points $x_0, \ldots, x_d$ in I such that $p^\star\left(x_i\right)$ (the minimax approximation) is as close as possible to $f\left(x_i\right)$
- We want to minimize $\left\|\sum_{i=0}^{d} m_i \begin{pmatrix} 2^{e_i} x_0^i \\ \ldots \\ 2^{e_i} x_d^i \end{pmatrix} - \begin{pmatrix} f(x_0) \\ \ldots \\ f(x_d) \end{pmatrix}\right\|_2$, which is an instance of the Closest Vector Problem.

L^2: A functional view

- Using a function space as the overall vector space: $\mathscr{F}(I, \mathbb{R})$ (and $\mathrm{Span}_{\mathbb{R}}\left(x^0, ..., x^n\right)$ as a subspace)
- Lattice basis are scaled monomials: $x \mapsto 2^{e_i} x^i$
- Euclidean norm as an integral computation

Integral inner product

- Weight function: $w : x \mapsto \sqrt{1 - x^2}^{-1}$
- Inner product: $\int_{-1}^{1} f(x)g(x)w(x)\,\mathrm{d}x$

$\Longrightarrow$ The projection gives the Chebyshev truncated series

Computing integrals

- Using ARB[7] for the intermediate computations
- High precision (1024-2048 bits) is required so the result is not just an error ball

$w : x \mapsto \frac{1}{\sqrt{1-x^2}}$ is ill-conditionned at the bounds of I

[7]https://flintlib.org/doc/index_arb.html

Computing integrals

- Using ARB[8] for the intermediate computations
- High precision (1024-2048 bits) is required so the result is not just an error ball

$w : x \mapsto \frac{1}{\sqrt{1-x^2}}$ is ill-conditionned at the bounds of I

- Change of variable: w disappears
  - Set $x = \cos(\theta)$, $\quad \langle f, g \rangle = \int_{-1}^{1} \big(f \times g\big)(x) w(x) \mathrm{d}x = \int_{0}^{\pi} \big(f \times g\big)(\cos(\theta)) \mathrm{d}\theta$

[8] https://flintlib.org/doc/index_arb.html

Chebyshev strikes again

- View it as a truncated Chebyshev series: $f \times g = \lim_{n \longrightarrow \infty} \sum_{k=0}^{n} h_{k,n} T_n$
-
-

[9]Trefethen, Lloyd N. and Weideman, J. A. C., The Exponentially Convergent Trapezoidal Rule

Chebyshev strikes again

- View it as a truncated Chebyshev series: $f \times g = \lim_{n \longrightarrow \infty} \sum_{k=0}^{n} h_{k,n} T_n$
- $h_{0,n} = \alpha \times \sum (f \times g)(\mu_k)$ where $\mu_k \coloneqq \cos\left(\frac{2k+1}{2n}\pi\right)$, the roots of T_{n+1}
-

[10] Trefethen, Lloyd N. and Weideman, J. A. C., The Exponentially Convergent Trapezoidal Rule

Chebyshev strikes again

- View it as a truncated Chebyshev series: $f \times g = \lim_{n \longrightarrow \infty} \sum_{k=0}^{n} h_{k,n} T_n$
- $h_{0,n} = \alpha \times \sum \left(f \times g\right) \left(\mu_k\right)$ where $\mu_k \coloneqq \cos\left(\frac{2k+1}{2n}\pi\right)$, the roots of T_{n+1}
- For $k \neq 0$, $\int_0^\pi T_k(\cos(\theta))\, \mathrm{d}\theta = \int_0^\pi \cos(k\theta)\, \mathrm{d}\theta = 0$

[11]Trefethen, Lloyd N. and Weideman, J. A. C., The Exponentially Convergent Trapezoidal Rule

Chebyshev strikes again

- View it as a truncated Chebyshev series: $f \times g = \lim_{n \longrightarrow \infty} \sum_{k=0}^{n} h_{k,n} T_n$
- $h_{0,n} = \alpha \times \sum \left(f \times g\right)\left(\mu_k\right)$ where $\mu_k \coloneqq \cos\left(\frac{2k+1}{2n}\pi\right)$, the roots of T_{n+1}
- For $k \neq 0$, $\int_0^\pi T_k(\cos(\theta))\, \mathrm{d}\theta = \int_0^\pi \cos(k\theta)\, \mathrm{d}\theta = 0$

Hence,

$$\langle f, g\rangle = \frac{\pi}{n} \lim_{n \longrightarrow \infty} \sum \left(f \times g\right)\left(\cos\left(\frac{2k+1}{2n}\pi\right)\right)$$

which converges exponentially fast[12].

[12] Trefethen, Lloyd N. and Weideman, J. A. C., The Exponentially Convergent Trapezoidal Rule

Fpminimax: Discretization

- Take $d + 1$ points $x_0, \ldots, x_d$ in I such that $p^\star(x_i)$ (the minimax approximation) is as close as possible to $f(x_i)$
- We want to minimize:

$$\left\| \sum_{i=0}^{d} m_i \begin{pmatrix} 2^{e_i} x_0^i \\ \ldots \\ 2^{e_i} x_d^i \end{pmatrix} - \begin{pmatrix} f(x_0) \\ \ldots \\ f(x_d) \end{pmatrix} \right\|_2$$

Fpminimax: a special case of L^2 ?

- Minimize $\sum_{j=0}^{d}\left(\sum_{i=0}^{d} m_i\left(2^{e_i}x_j^i\right) - f\left(x_j\right)\right)^2$
- When the $\left(x_j\right)$ are the Chebyshev nodes, it is the same computation as our integral
- The sum can be seen as an approximation of

$$\underbrace{\int_{-1}^{1}\left(\sum_{i=0}^{d} m_i\left(2^{e_i}x^i\right) - f(x)\right)^2 \mathrm{d}x}_{L^2} \sim \underbrace{\frac{1}{d+1}\sum_{j=0}^{d}\left(\sum_{i=0}^{d} m_i\left(2^{e_i}x_j^i\right) - f\left(x_j\right)\right)^2}_{\text{fpminimax}}$$

Closest Vector Problem: Gram form

- Vectors are functions $\Longrightarrow$ need of a basis to express them
- Use the same basis!
- Gram matrix: $G \coloneqq \left(\langle b_i, b_j \rangle\right)_{i,j \in [\![0,n]\!]}$
- Projection: $V \coloneqq \left(\langle f, b_i \rangle\right)_{i \in [\![0,n]\!]}$

$\Longrightarrow$ we have coordinates scaled by the norm of the b_i

Nearest Plane in Gram form

Input:

- G, the Gram matrix of the basis $(b_i)_{i \in [\![0,n]\!]}$
- V, the projection of f onto the space generated by $(b_i)_{i \in [\![0,n]\!]}$, in the form $(f|b_i)_{i \in [\![0,n]\!]}$

Output:

- $X \in \mathbb{N}^{n+1}$ the coordinates of an element of the lattice generated by $(b_i)_{i \in [\![0,n]\!]}$ close to f

begin

$D, B \coloneqq \mathrm{Gram\_Schmidt}(G)$, i.e. $G = B^t D B$

$W \leftarrow D^{-1} (B^t)^{-1} V$

for j **from** n **to** 0

$X[j] \leftarrow \lfloor W[j] \rceil$

for i **from** 0 **to** n

$W[i] \leftarrow W[i] - X[j] B[i, j]$

end

end

return X

end

Some results

Table

Degree	Minimax error	Naïve rounding error	fpminimax error	L^2 projection error	Babai error
7	2.5870e-4	2.9446e-4	2.5870e-4	2.9446e-4	2.9446e-4
25	9.9686e-12	1.2099e-11	9.9686e-12	1.2099e-11	1.2099e-11
37	1.7341e-16	2.1254e-16	1.7341e-16	2.1231e-16	2.1236e-16
47	2.0381e-20	9.2094e-18	2.6477e-20	2.4891e-20	2.5526e-20

Maximal relative errors betweens approximating polynomials and $\arctan$ over $[-1, 1]$

Conclusion

- More general view of the minimization problem
- Another tool, complementary to fpminimax, for polynomial approximation
- Trivial extension for multivariate functions (integrate over a n-dimensional cube)
- But it does not take into account the evaluation error due to rounding

  c.f. work by D. Arzelier, F. Bréhard and M. Joldes