FPE : Fast Polynomial Evaluation

Ramona Anton — UMR 7586

Nicolae Mihalache — UMR 8050

François Vigneron — UMR 9008

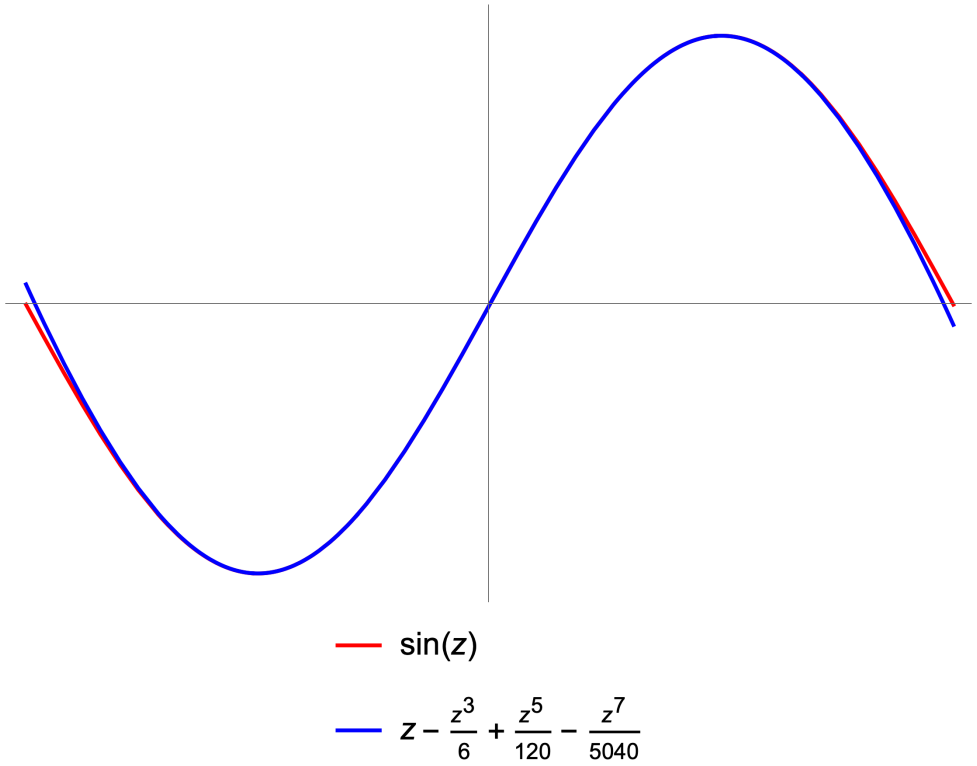
ramona.anton@imj-prg.fr
nicolae.mihalache@u-pec.fr
francois.vigneron@univ-reims.fr

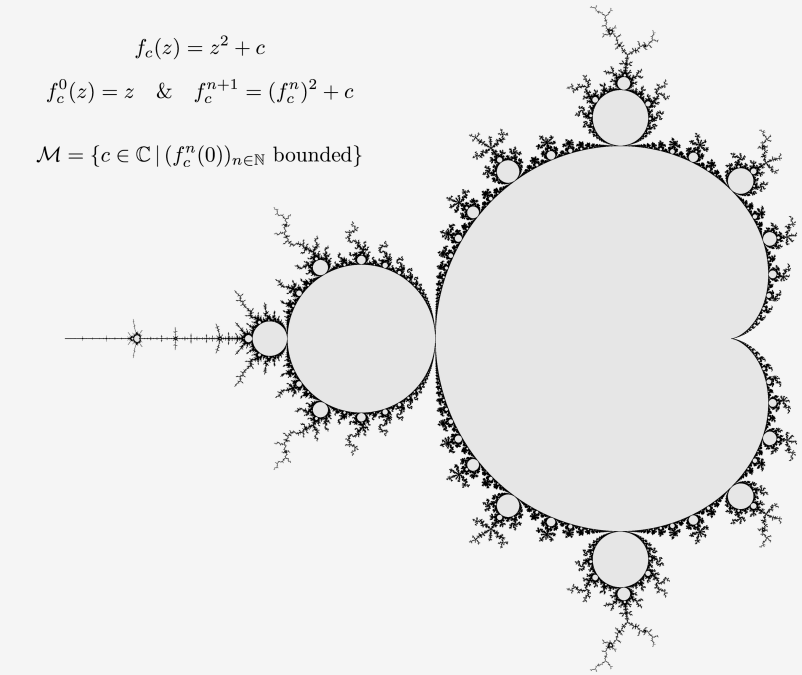
Journées Calcul & Données 2023

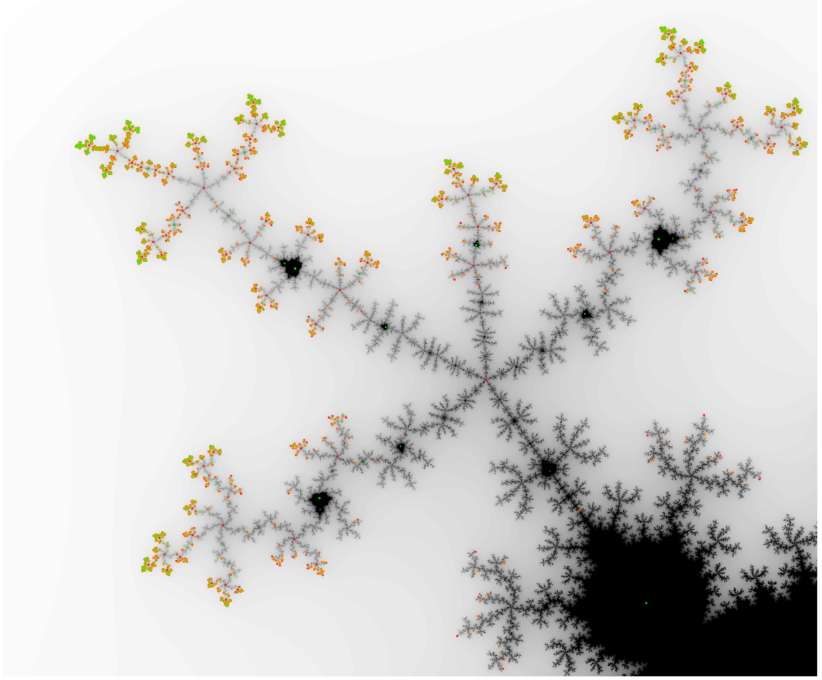
1. Polynomials

Polynomials are **ubiquitous**:

$$P(z) = a_0 + a_1 z + a_2 z^2 + \ldots + a_d z^d$$





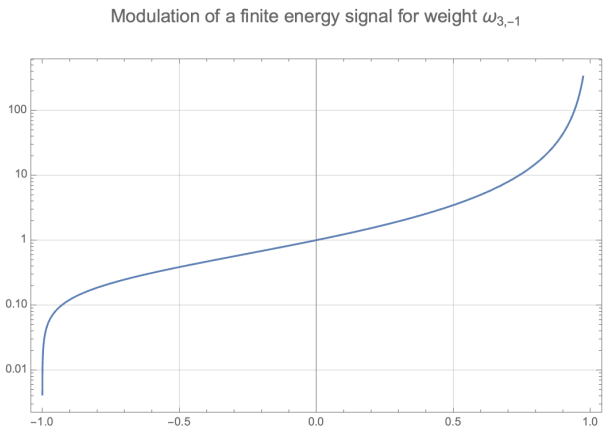


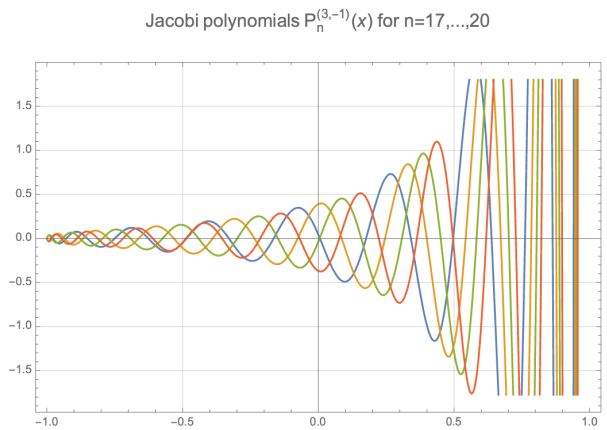
Tera-polynomial (deg $= 10^{12}$) slain on Roméo, 2021.

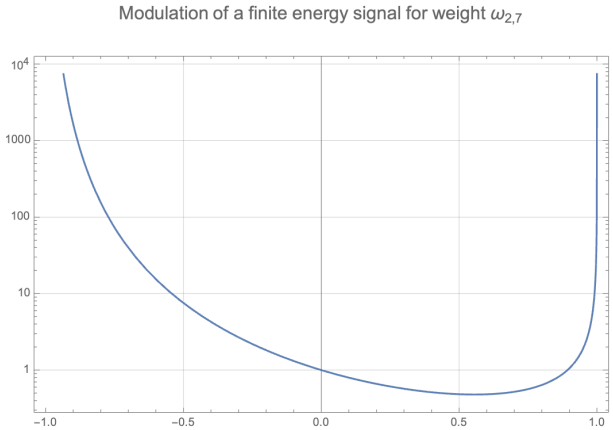
Pre-periodic point (deg $= 10^{10}$, $\times 100$) on Roméo, 2023.

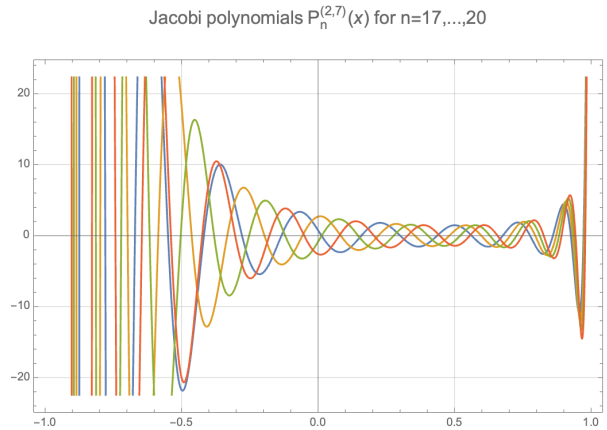
The Jacobi polynomials $P_n^{(\alpha,\beta)}$

$$\int_{-1}^{+1} P_m^{(\alpha,\beta)}(x) P_n^{(\alpha,\beta)}(x) \cdot (1-x)^\alpha (1+x)^\beta dx = \delta_{m,n}.$$

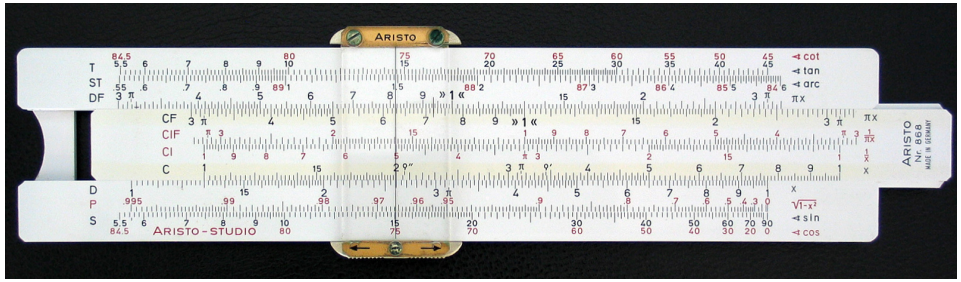


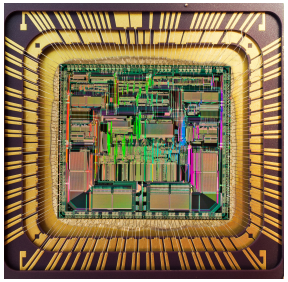






2. Floating point arithmetic

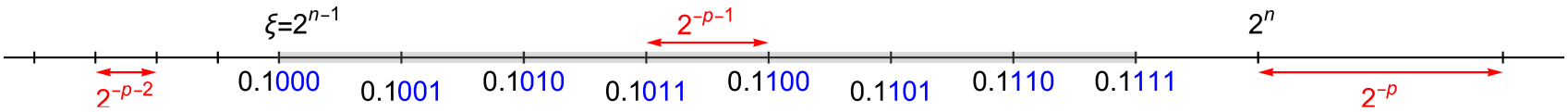




Floating point numbers (in base 2) with precision p bits:

$$\xi = \pm 2^n \times 0.1\underbrace{\xi_1 \dots \xi_p}_{p \text{ bits}} \qquad \text{with} \qquad \xi_1, \dots, \xi_p \in \{0, 1\}.$$

The **scale** (order of magnitude) of ξ is $n = s(\xi) = 1 + \lfloor \mathbf{log}_2 |\xi| \rfloor$.



Range of numbers with scale n.

Cost of multiplication in precision p:

$$M(p) = \begin{cases} \leq 10 \text{ cycles} & \texttt{FP64 (double, } p = 52) \rightsquigarrow \text{hardware} \\ \hline O(p^2) & \text{schoolbook expansion} \\ O(p^{1.585}) & \text{Karatsuba} \\ O(p^{1.465}) & \text{Toom-Cook} \\ O(p \log p \log\log p) & \text{Schönhage-Strassen} \end{cases}$$

$$(Pz^k + R)(Qz^k + M) = RM + ((P + R)(Q + M) - RM - PQ)\, z^k + PQz^{2k}$$

3. Polynomial evaluation

Classical methods

How to compute efficiently

$$P(z) = a_0 + a_1 z + a_2 z^2 + \ldots + a_d z^d.$$

1. **Monomial**: compute z^d recursively in time $O(M(p) \log d)$

$$z^{10} = z^{2+2^3} = \left(z \times (z^2)^2\right)^2 \qquad z \mapsto z^2 \mapsto z^4 \mapsto z^5 \mapsto z^{10}$$

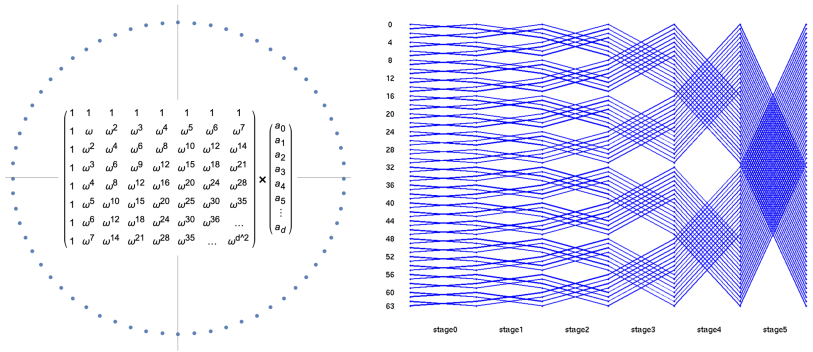
2. **Hörner's method**: compute $P(z)$ in time $O(M(p)d)$

$$2 - 3z + 4z^2 + 5z^3 = 2 + z\left(-3 + z\left(4 + 5z\right)\right)$$

$$P(z) = a_0 + z\underbrace{\left(a_1 + z\left(a_2 + \ldots\right)\right)}_{\text{deg. } d-1}$$

Multipoint: compute $P(z_j)$ on a mesh $\mathcal{Z} = (z_j)_{0 \le j \le d}$.

3. **FFT method**: $O(M(p)d \log d)$ if $\mathcal{Z}$ is a regular circular mesh



4. **Algebraic methods**: $O(M(p)d \log^2 d)$ on general mesh, with memory $O(d^2 \log d)$.

3. Polynomial evaluation

Lazy addition

In finite precision ($p = 4$ decimals), scale considerations can spare us the actual computation:

$$\begin{array}{rcrcrcl} z_L & \simeq_4 & 1.234 \times 10^{+2} & = & 12.340000 & & \\ z_S & \simeq_4 & 5.678 \times 10^{-3} & = & 0.005678 & & \\ \hline & & & + & 12.345678 & \simeq_4 & 1.234 \times 10^{+2} \end{array}$$

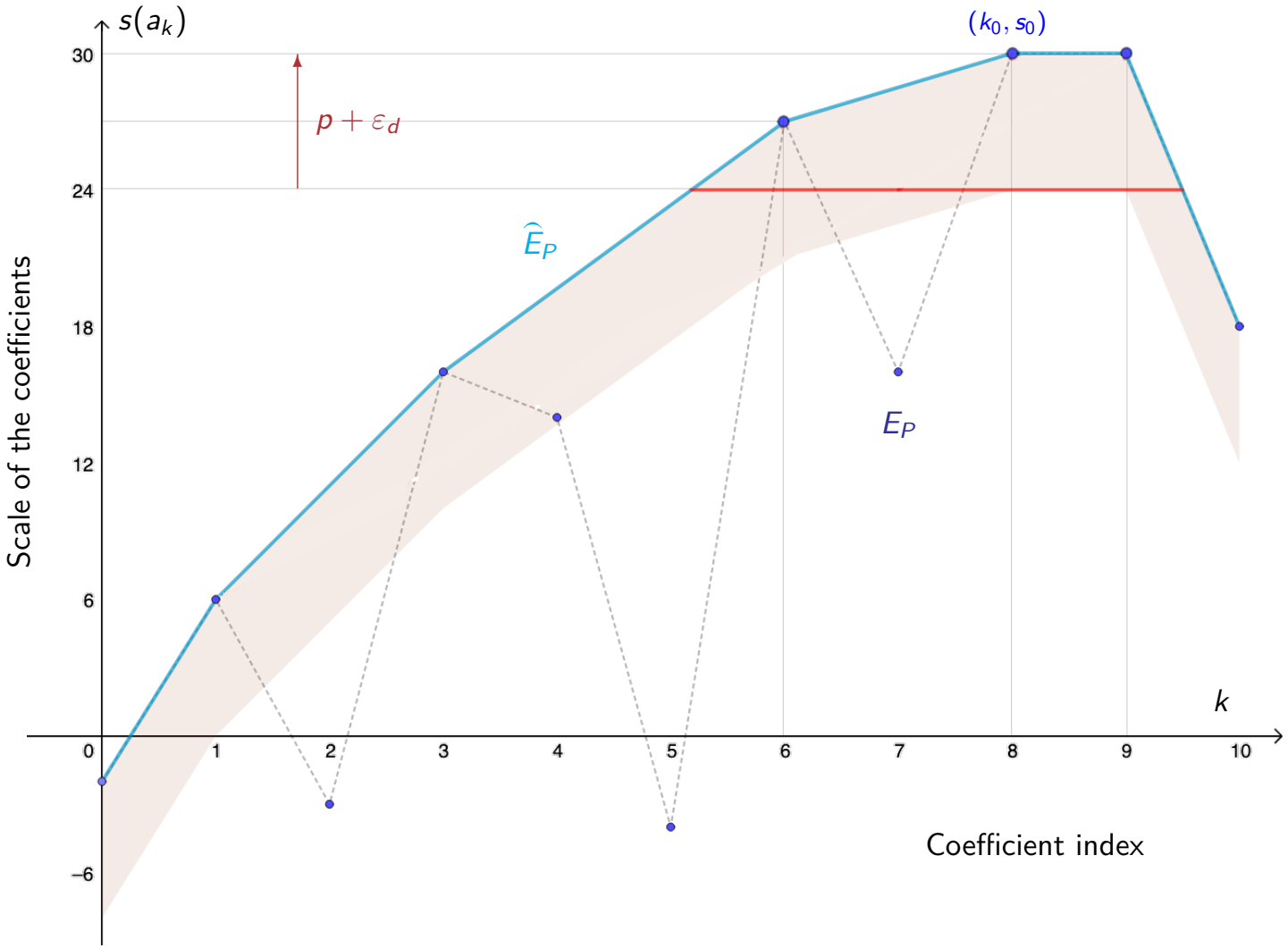
Practical rule: **start with the biggest monomial** and stop when the next operation has no possible influence on the result.

Problems with unsorted coefficients:

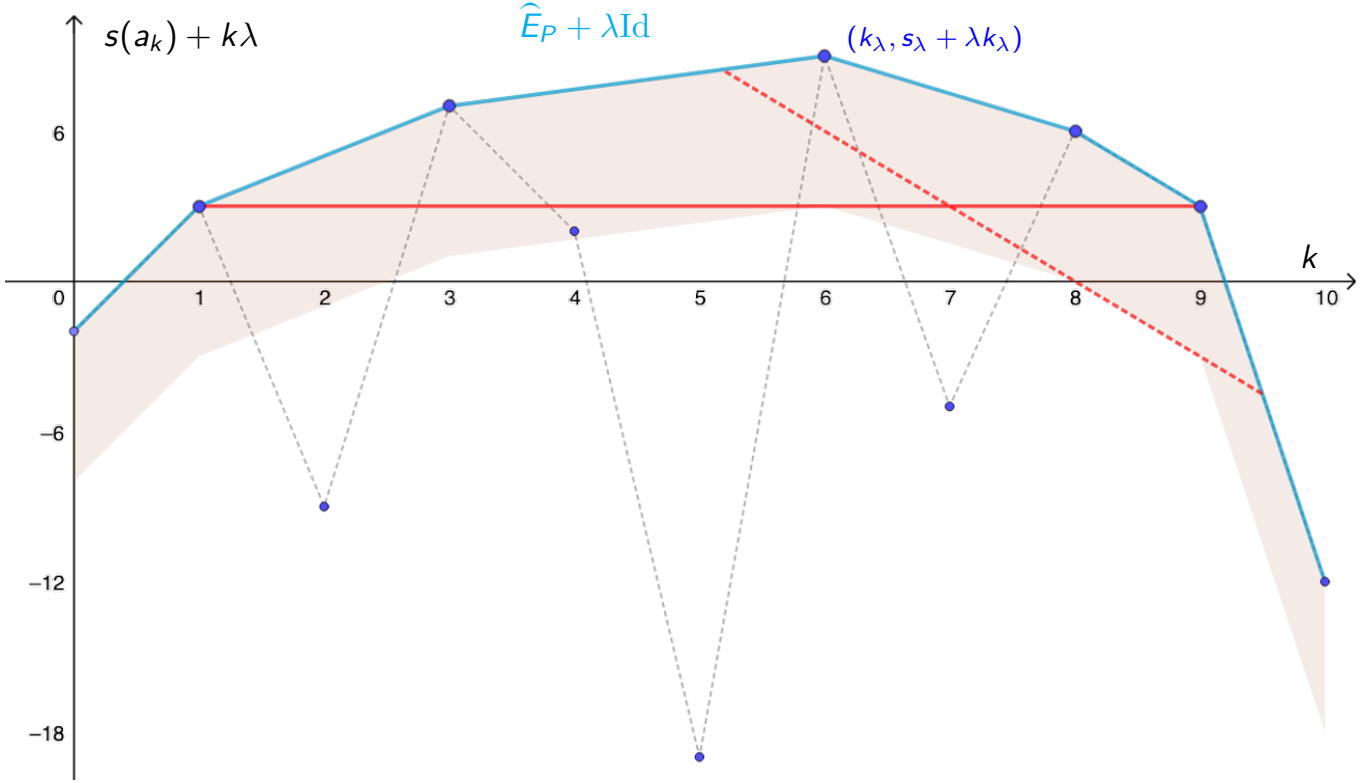
- Lazy evaluation requires sorting d numbers and costs $O(d \log d)$ operations $\gg O(d)$.
- Absolute precision threshold ? $s(z_L) \pm 10^{-p} = 0.001 \times 10^{+2}$ requires knowledge of the biggest input.

FPE algorithm (Fast Polynomial Evaluation)

$$\text{if } |z| = 1, \quad \sum a_k z^k \simeq a_6 + a_8 + a_9$$

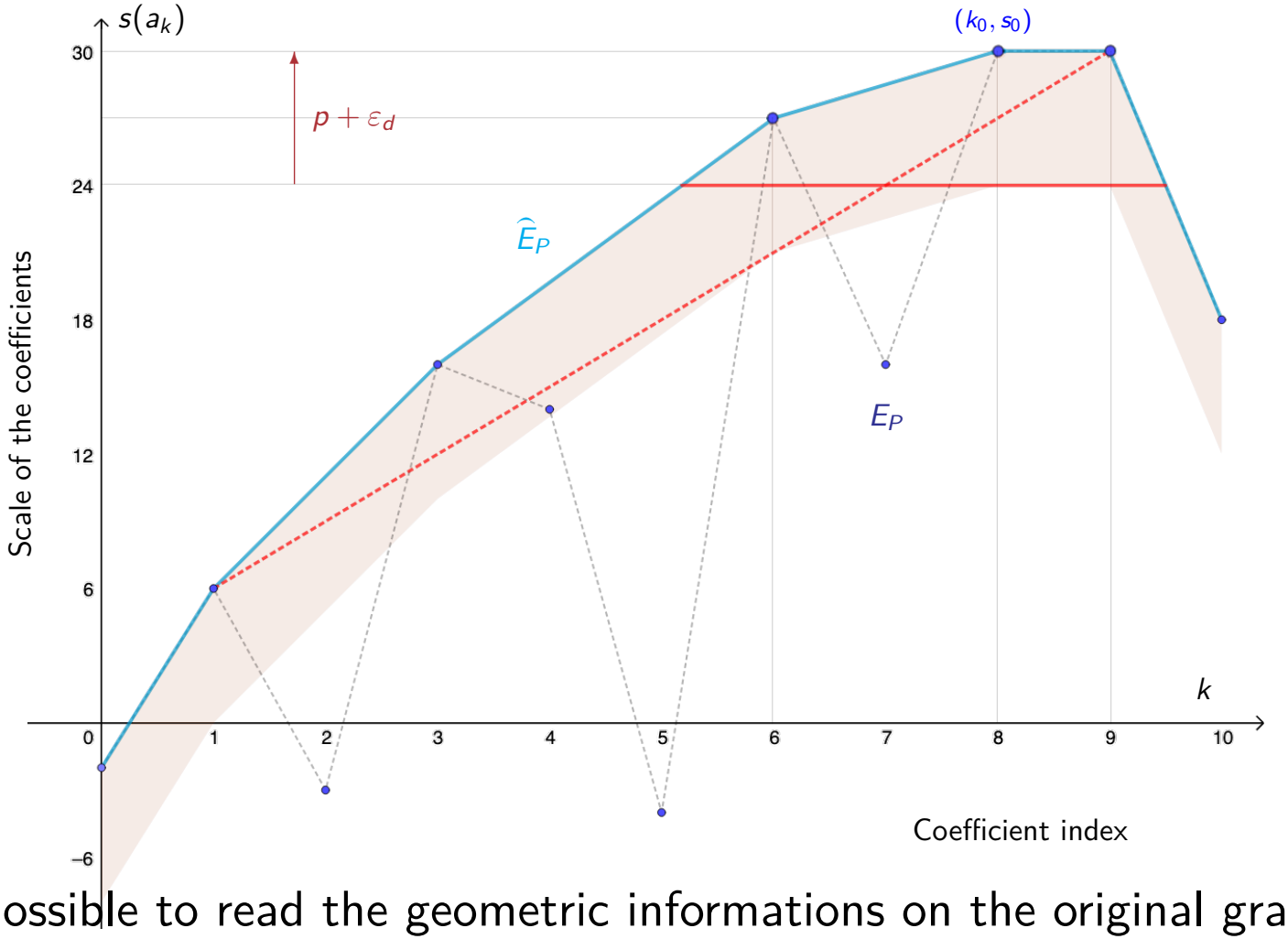


For $\log_2 |z| = \lambda$, $\sum a_k z^k \simeq a_1 z + a_3 z^3 + a_6 z^6 + a_8 z^8 + a_9 z^9$



It is possible to read the geometric informations on the original graph.

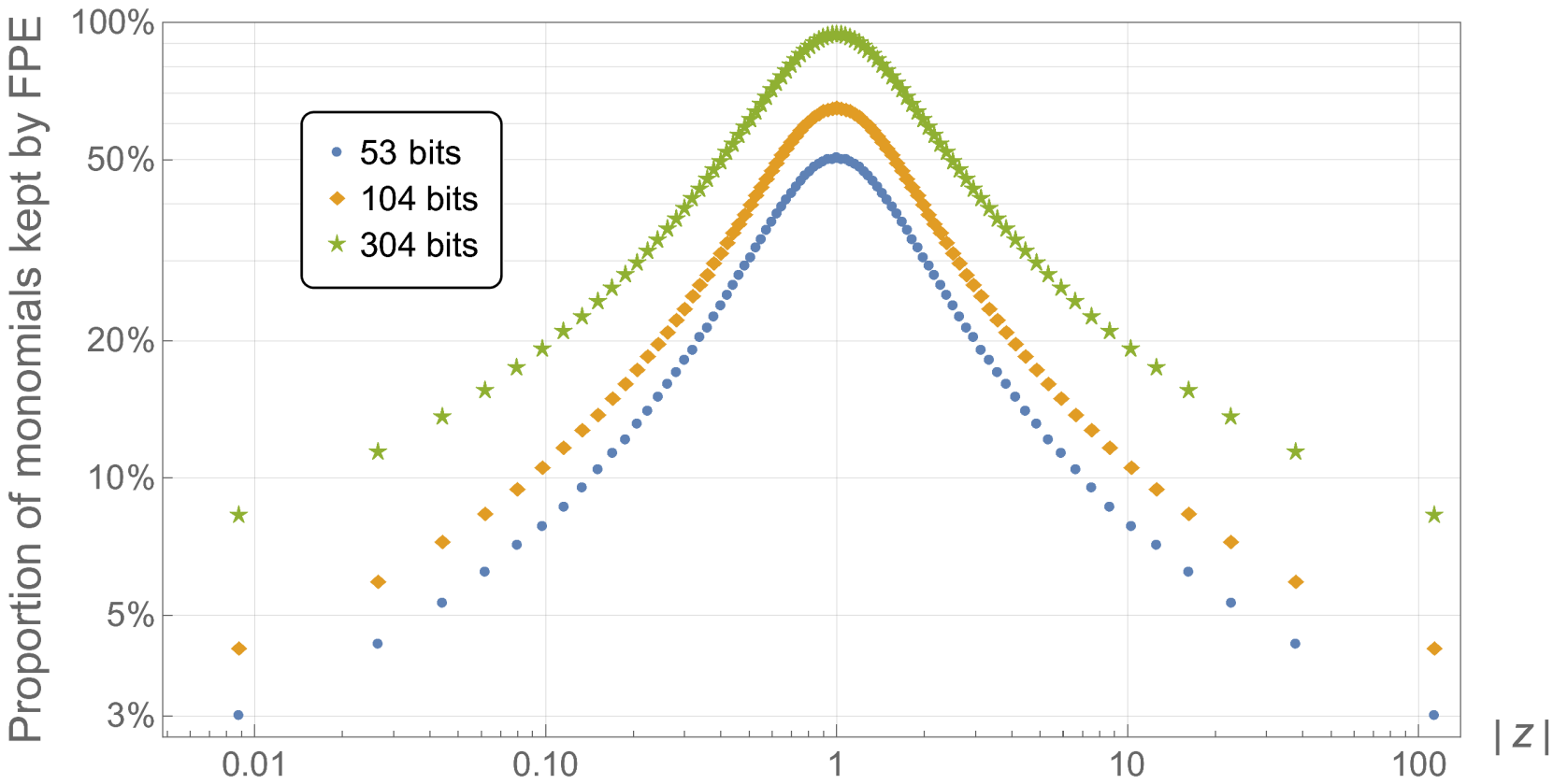
For $\log_2 |z| = \lambda$, $\sum a_k z^k \simeq a_1 z + a_3 z^3 + a_6 z^6 + a_8 z^8 + a_9 z^9$



It is possible to read the geometric informations on the original graph.

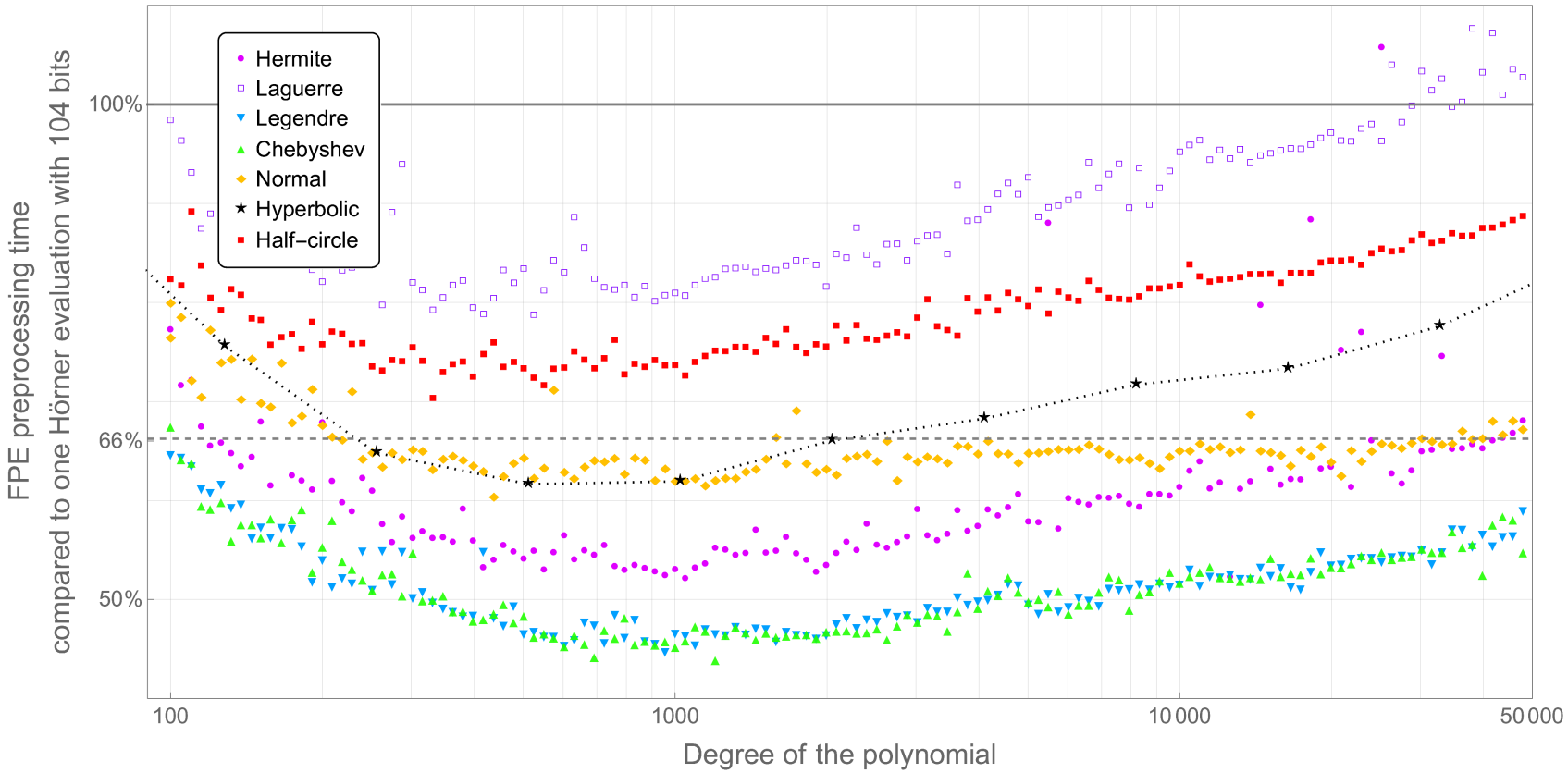
Reduction theorem (based on average width of a thin concave cap):

$$\sum_{k=0}^{d} a_k z^k \simeq_p \underbrace{\sum_{k \in K_p(z)} a_k z^k}_{\text{Hörner}} \qquad \text{with} \qquad |K_p(z)| = \underbrace{O\left(\sqrt{d(p+\log d)}\right)}_{\text{on } z\text{-average}}$$

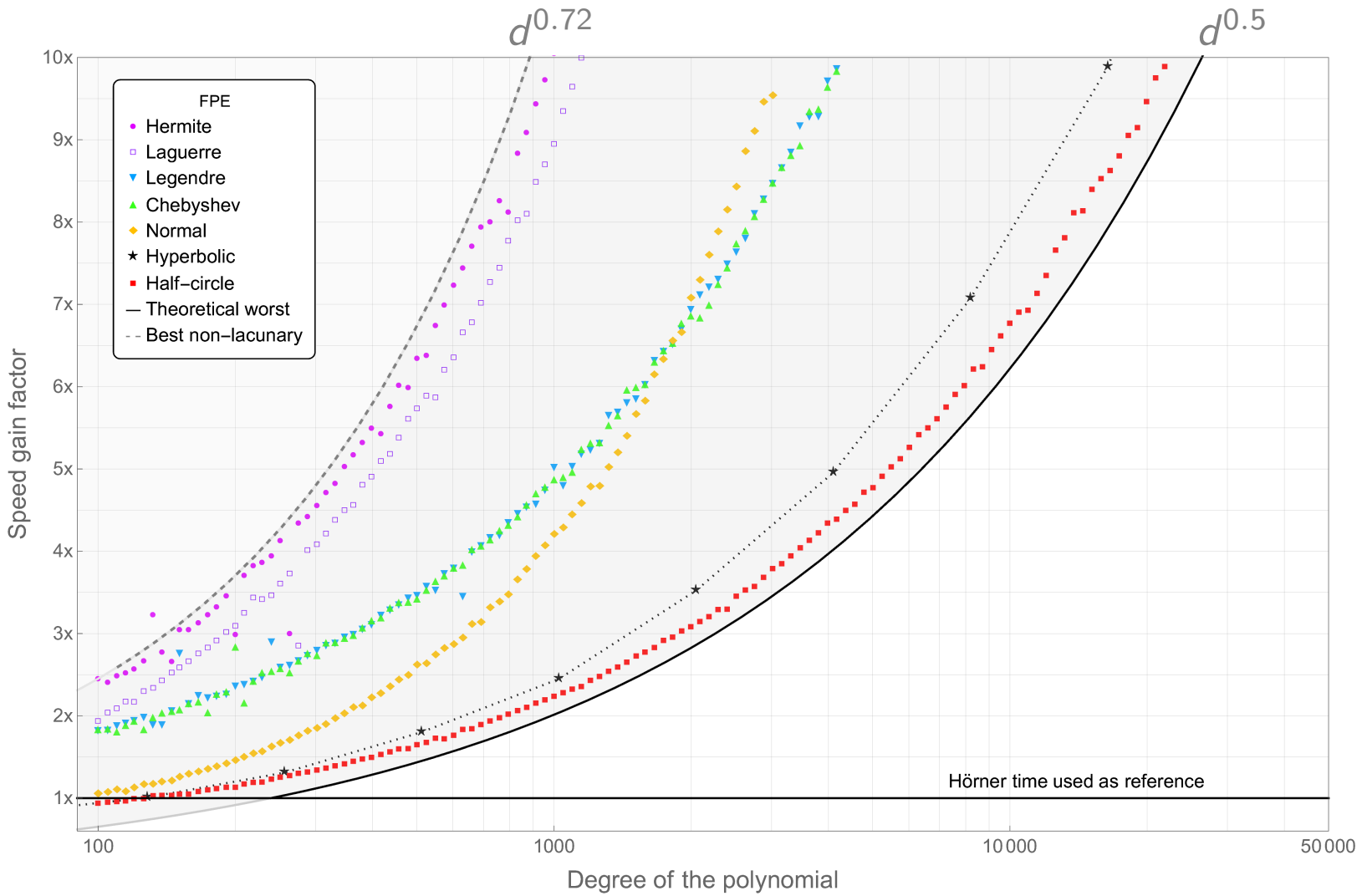


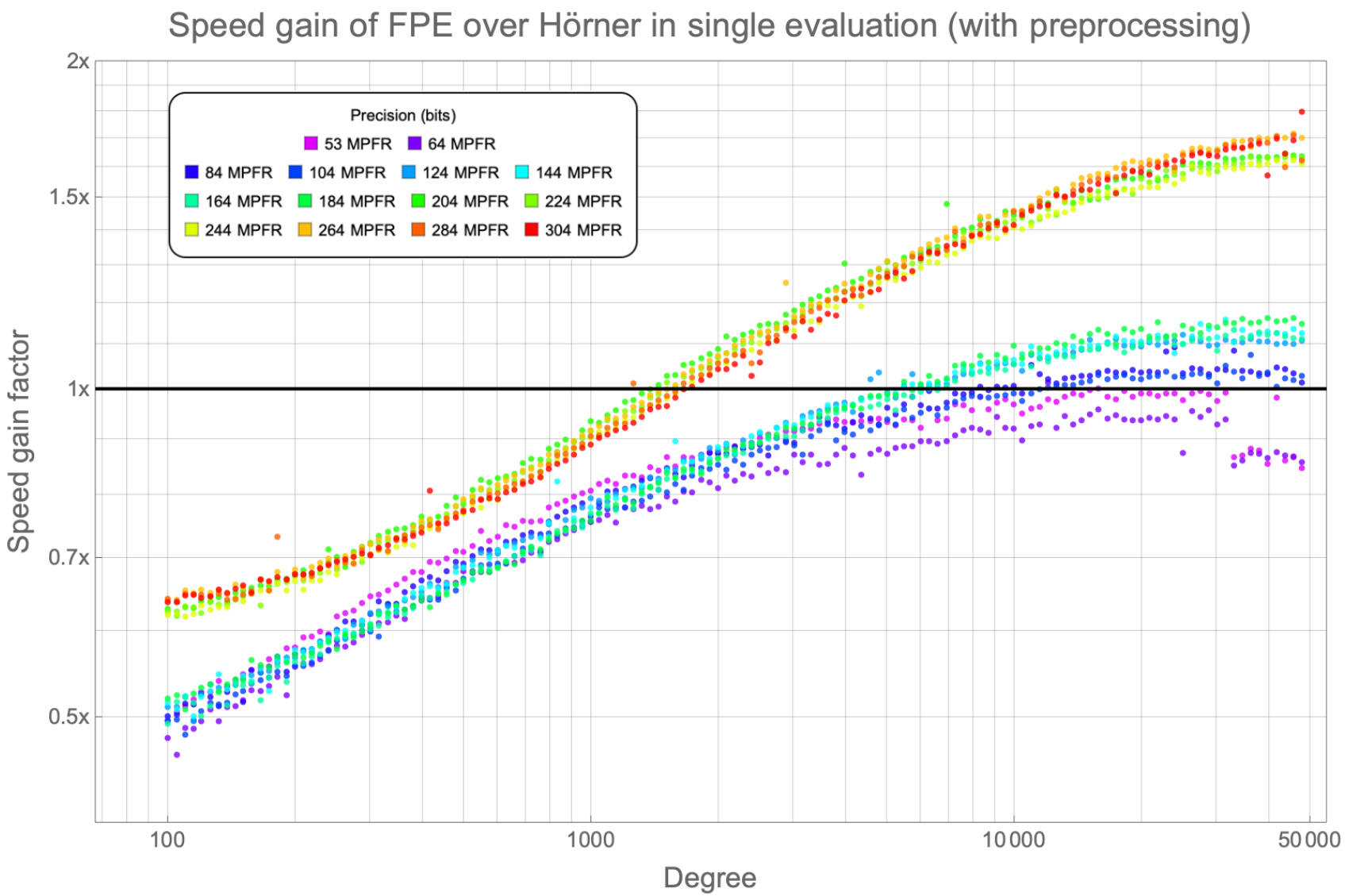
Benchmark on Roméo

Preprocessing in $d + 2d \log d$ using only scales (independent of p).

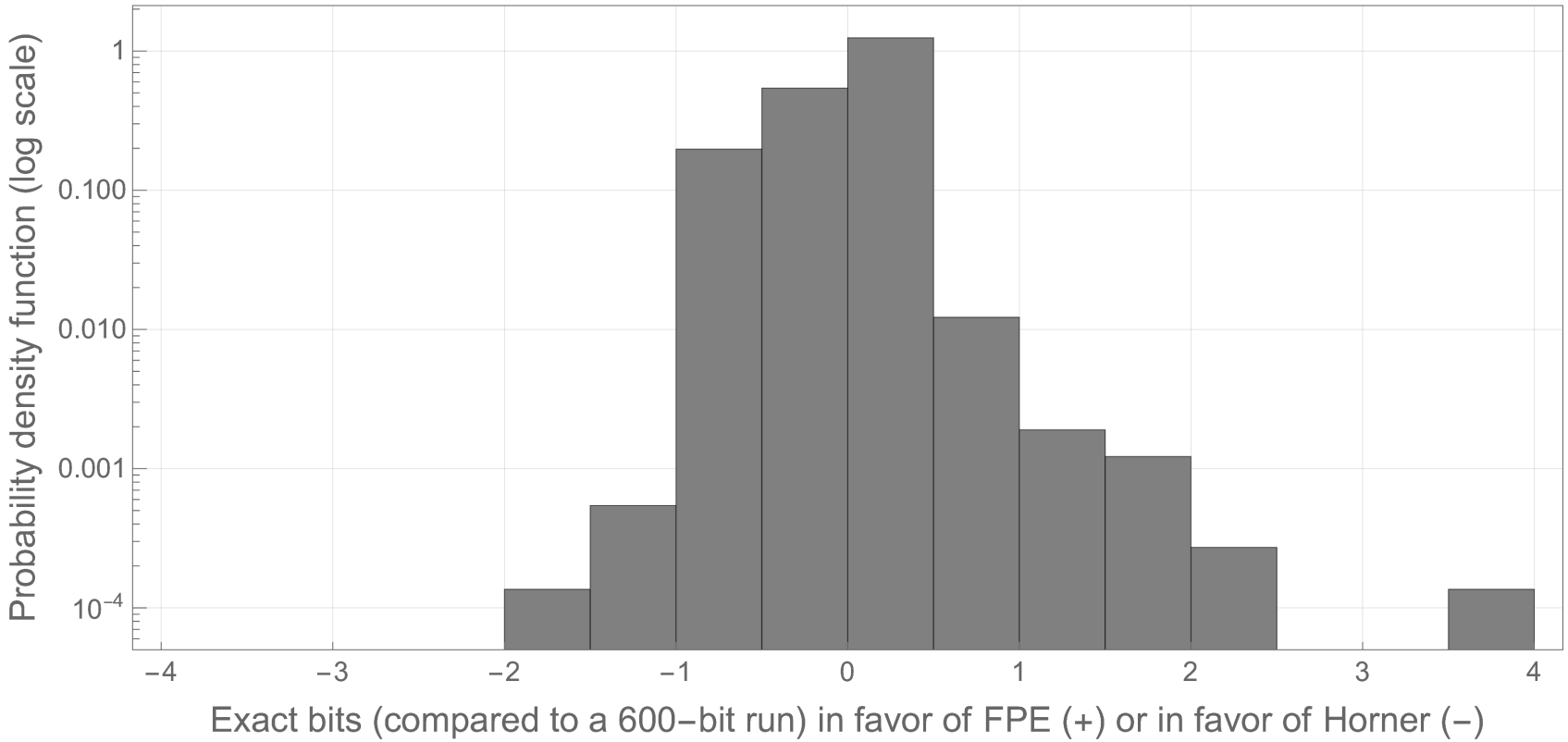


Average cost $\leq M(p)\sqrt{d(p+\log d)}$ per eval. & embarrassingly parallel.

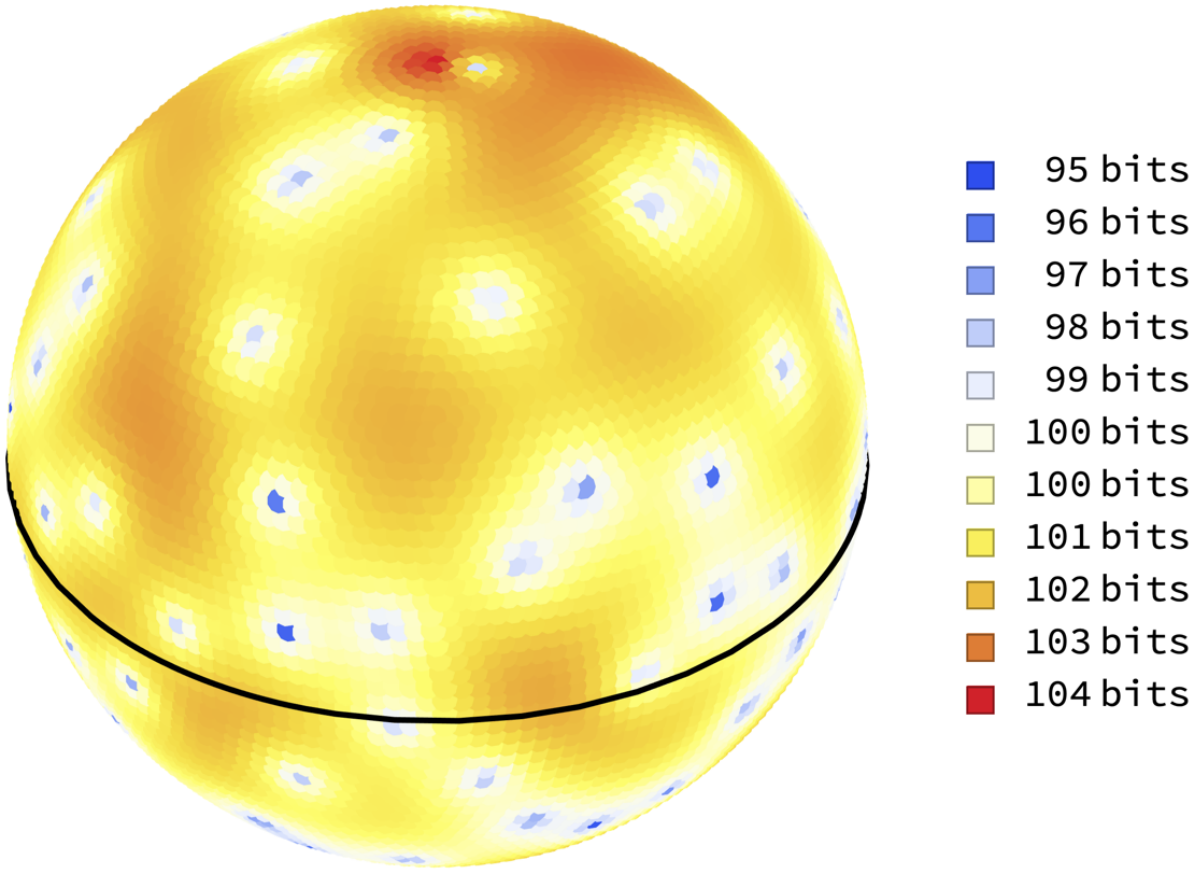



Speed gain of FPE over Hörner in single evaluation (with preprocessing)
Precision (bits)
53 MPFR
64 MPFR
84 MPFR
104 MPFR
124 MPFR
144 MPFR
164 MPFR
184 MPFR
204 MPFR
224 MPFR
244 MPFR
264 MPFR
284 MPFR
304 MPFR
Speed gain factor
2x
1.5x
1x
0.7x
0.5x
100
1000
10 000
50 000
Degree

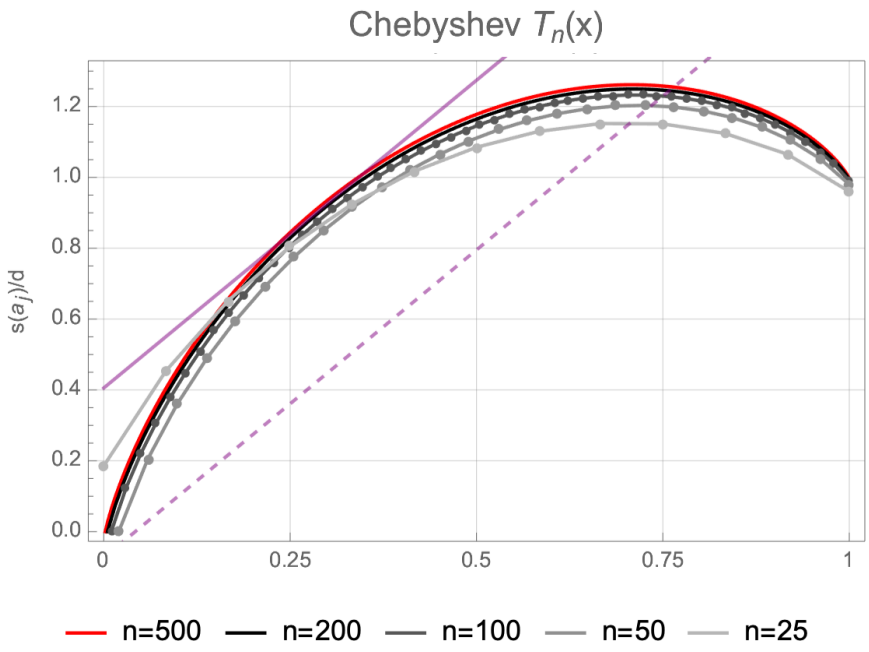
Result guarantied to be adjacent to the exact value, up to cancelled bits.

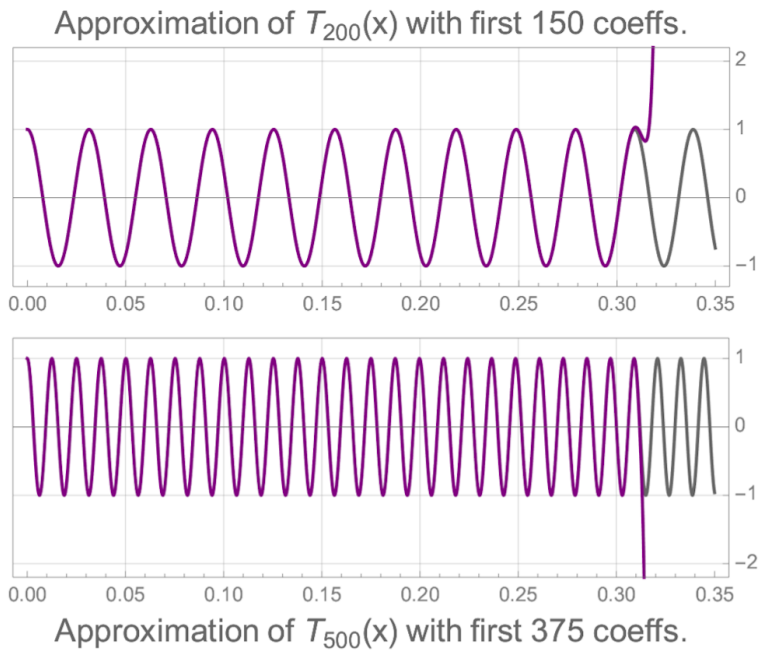


The number of canceled bits (precision loss) is free and explicit.



Predict which terms are necessary.





$\rightsquigarrow$ universal law for Jacobi polynomials (work with Clémentine Courtes).

https://github.com/fvigneron/FastPolyEval

```
git clone fvigneron/FastPolyEval FPE
make fpe

[vigneron@Francoiss-MacBook-Pro ~/FPE]
$ ./bin/FastPolyEval -help

The following tasks can be performed with the corresponding command line arguments below:

    Task            Parameters                                          Short description

    -hyperbolic     prec period outFile                                 writes the coefficients of the hyperbolic polynomial to a CSV file
    -Chebyshev      prec degree outFile                                 writes the coefficients of the Chebyshev polynomial to a CSV file
    -Legendre       prec degree outFile                                 writes the coefficients of the Legendre polynomial to a CSV file
    -Hermite        prec degree outFile                                 writes the coefficients of the Hermite polynomial to a CSV file
    -Laguerre       prec degree outFile                                 writes the coefficients of the Laguerre polynomial to a CSV file
    -sum            prec inFile1 inFile2 outFile                        computes the sum of two polynomials and writes the result to a CSV file
    -diff           prec inFile1 inFile2 outFile                        computes the difference of two polynomials and writes the result to a CSV file
    -prod           prec inFile1 inFile2 outFile                        computes the product of two polynomials and writes the result to a CSV file
    -cat            prec inFile1 inFile2 outFile                        concatenates two CSV files containing complex numbers
    -join           prec inFile1 inFile2 outFile                        joins the real part of two sequnces into one sequence of complex numbers
    -grid           prec inFile1 inFile2 outFile                        computes the set product of the real parts of two sequences
    -exp            prec inFile outFile                                 computes the complex exponential of a list of points and writes the results to a CSV file
    -rot            prec inFile outFile                                 maps complex numbers (a, b) to a*exp(ib)
    -polar          prec inFile outFile                                 computes the points given by polar coordinates on the sphere and writes the result to a CSV file
    -roots          prec inFile outFile                                 computes the polynomial with a given list of roots and writes the result to a CSV file
    -der            prec inFile outFile                                 computes the derivative of a polynomials and writes the result to a CSV file
    -unif           prec count outFile [start] [end]                    writes count real numbers in arithmetic progression to a CSV file
    -rand           prec count outFile [start] [end]                    writes count real random numbers uniformly distributed in an interval to a CSV file
    -normal         prec count outFile [center] [var]                   writes count real random numbers with Gaussian distibution to a CSV file
    -sphere         prec count outFile                                  writes polar coordinates approximating an uniform distribution on the sphere to a CSV file
    -eval           prec polyFile pointsFile outFile [checkFile] [count]  quickly evaluates a polynomial on a set of points
    -evalD          prec polyFile pointsFile outFile [checkFile] [count]  quickly evaluates the derivative of a polynomial on a set of points
    -evalN          prec polyFile pointsFile outFile [checkFile] [count]  quickly evaluates one Newton step of a polynomial on a set of points
    -iterN          prec polyFile pointsFile outFile [checkFile] [count]  quickly iterates and searches for limits of the Newton method (roots of the polynomial)
    -comp           prec inFile1 inFile2 [outFile]                      compares two lists of points
    -re             prec inFile outFile                                 writes the real part of the list of complex numbers
    -im             prec inFile outFile                                 writes the imaginary part of the list of complex numbers
    -conj           prec inFile outFile                                 writes the conjugates of the list of complex numbers
    -tensor         prec inFile1 inFile2 outFile                        computes the tensorial product of the two lists of numbers (c_i = a_i * b_i)
    -analyse        prec polyFile outFile1 [outFile2]                   computes the concave cover and the intervals of |z| for which the evaluation strategy changes


For each of the tasks above, use -task -help for more detailed information.
If the precision is at most 53, machine floating numbers are used (FP64, double),
otherwise MPFR floating numbers are used.
```

Thank you