Elementary functions

Jean-Michel Muller

CNRS - Laboratoire LIP

http://perso.ens-lyon.fr/jean-michel.muller/

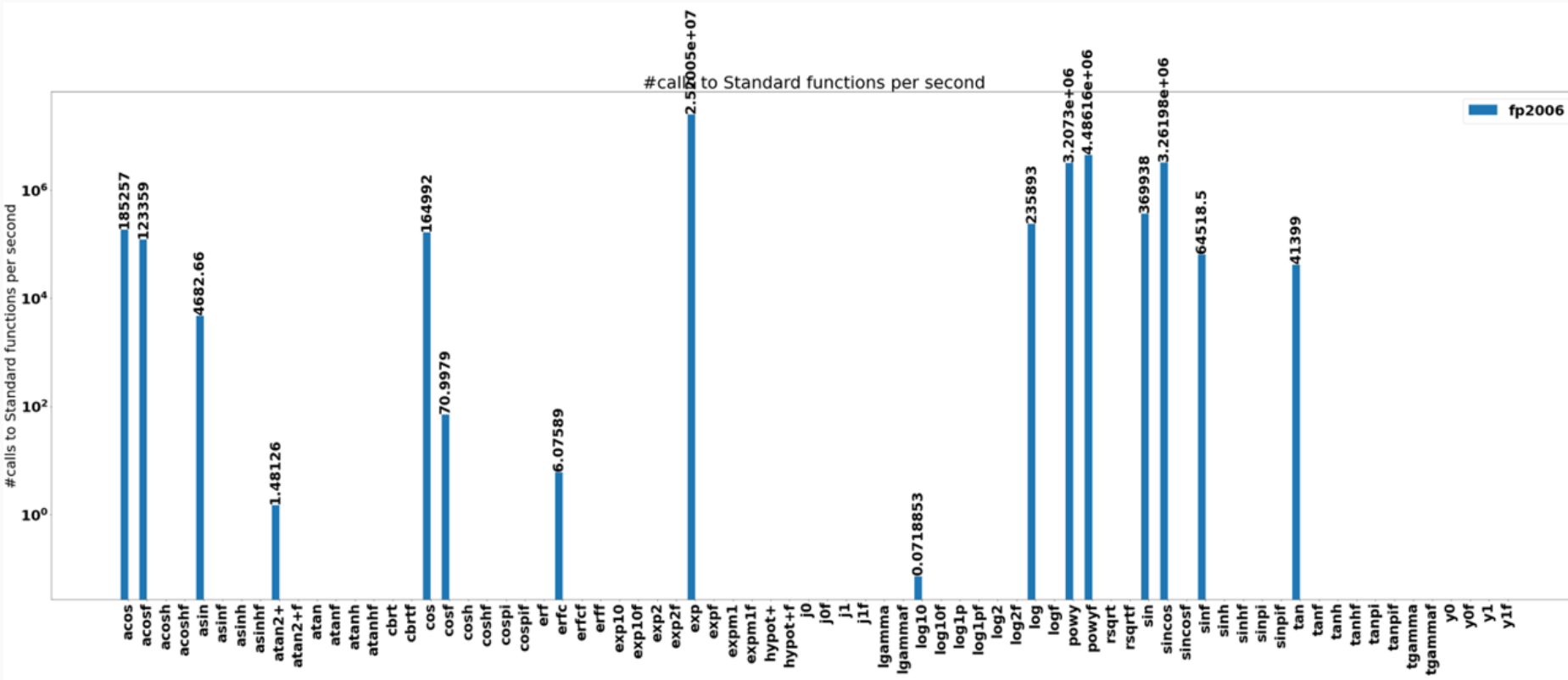
Elementary functions

- real name: elementary transcendental functions. The ones than can be built from the complex exponential and logarithm;
- the ones listed by IEEE-754 are:

$$e^x, e^x - 1, 2^x, 2^x - 1, 10^x, 10^x - 1,$$
$$\log(x), \log_2(x), \log_{10}(x), \log(1+x), \log_2(1+x), \log_{10}(1+x),$$
$$\sqrt{x^2+y^2}, 1/\sqrt{x}, (1+x)^n, x^n, x^{1/n} (n \text{ is an integer}), x^y,$$
$$\sin(\pi x), \cos(\pi x), \tan(\pi x), \arcsin(x)/\pi, \arccos(x)/\pi, \arctan(x)/\pi, \arctan(y/x)/\pi,$$
$$\sin(x), \cos(x), \tan(x), \arcsin(x), \arccos(x), \arctan(x), \arctan(y/x),$$
$$\sinh(x), \cosh(x), \tanh(x), \operatorname{arcsinh}(x), \operatorname{arccosh}(x), \operatorname{arctanh}(x).$$

- the C Standard defines a rather similar list;
- a few functions (sin, cos, exp, log, ...) are very frequently called, and are used as building blocks for implementing the other functions $\rightarrow$ efficient and accurate implementation of these functions is necessary.

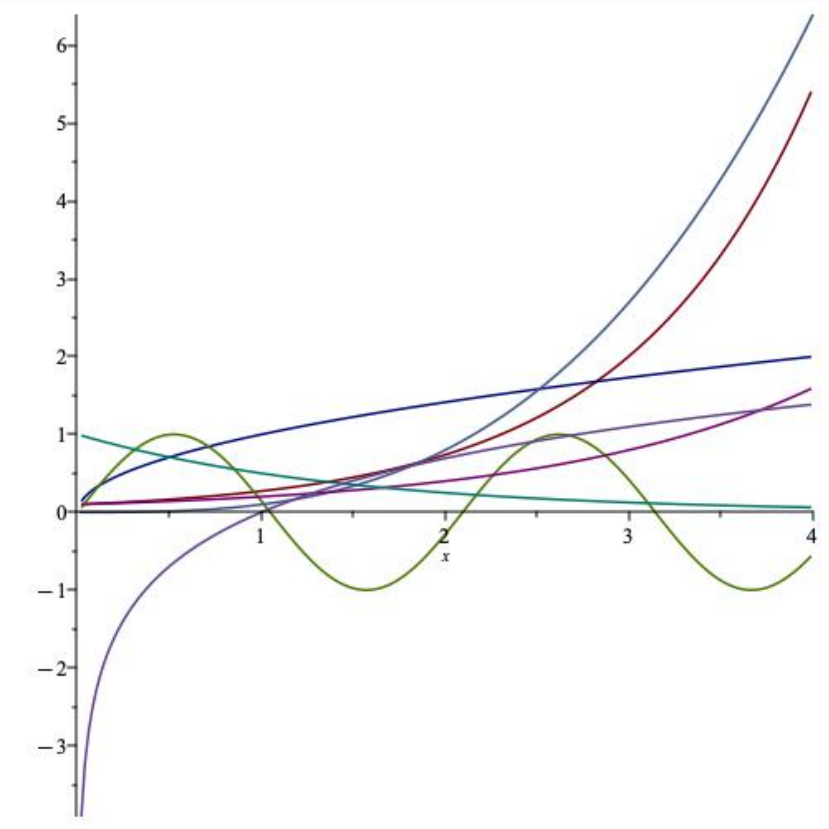
Elementary functions



Number of calls per second of various functions in a CERN proton collision application.

Elementary functions

- core set of “atomic functions”: exp, log, sin, . . .
- highest possible quality: reproducibility, proven error bounds, etc.
- best possible quality: correct rounding.



Work still needs to be done...

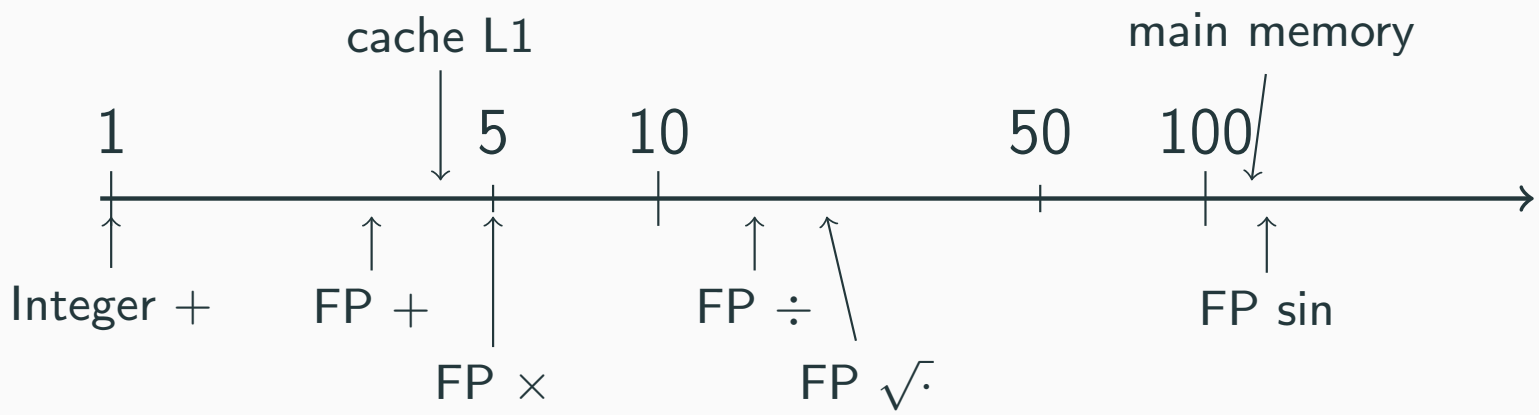
library	GNU libc	IML	AMD	Newlib	OpenLibm	Musl	Apple	LLVM	MSVC	FreeBSD	ArmPL	CUDA	ROCm
version	2.40	2024.0.2	4.2	4.4.0	0.8.3	1.2.5	14.5	18.1.8	2022	14.1	24.04	12.2.1	5.7.0
acos	**0.523**	0.531	1.36	0.930	0.930	0.930	1.06		0.934	0.930	1.52	1.53	0.772
acosh	2.25	**0.509**	1.32	2.25	2.25	2.25	2.25		3.22	2.25	2.66	2.52	0.661
asin	**0.516**	0.531	1.06	0.981	0.981	0.981	0.709		1.05	0.981	2.69	1.99	0.710
asinh	1.92	**0.507**	1.65	1.92	1.92	1.92	1.58		2.05	1.92	2.04	2.57	0.661
atan	**0.523**	0.528	0.863	0.861	0.861	0.861	0.876		0.863	0.861	2.24	1.77	1.73
atanh	1.78	**0.507**	1.04	1.81	1.81	1.80	2.01		2.50	1.81	3.00	2.50	0.664
cbrt	3.67	0.523	1.53e22	0.670	0.668	0.668	0.729		1.86	0.668	1.79	**0.501**	**0.501**
cos	**0.516**	0.518	0.919	0.887	0.834	0.834	0.948	Inf	0.897	0.834		1.52	0.797
cosh	1.93	**0.516**	1.85	2.67	1.47	1.04	0.523		1.91	1.47	1.93	1.40	0.563
erf	1.43	**0.773**	1.00	1.02	1.02	1.02	6.41		4.62	1.02	2.29	1.50	1.12
erfc	5.19	**0.826**		4.08	4.08	3.72	10.7		8.46	4.08	1.71	4.51	4.08
exp	0.511	0.530	1.01	0.949	0.949	0.511	0.521	**0.500**	1.50	0.949	0.511	0.928	0.929

Largest errors in ulps for double-precision calculation of some math functions. ulp (x) is the distance between two FP numbers in the neighborhood of x (so the largest values should be 0.5 – which is the case with $+$, $-$, $\times$, $\div$, and $\sqrt{\cdot}$).

(Extracted from Gladman, Innocente, Mather, and Zimmermann, *Accuracy of Mathematical Functions...*, Aug. 2024)

Elementary functions

- hardware arithmetic of our processors: $\pm$, $\times$, $ab+c$ (FMA), and $\div$ (costs more than $+$ and $\times$);

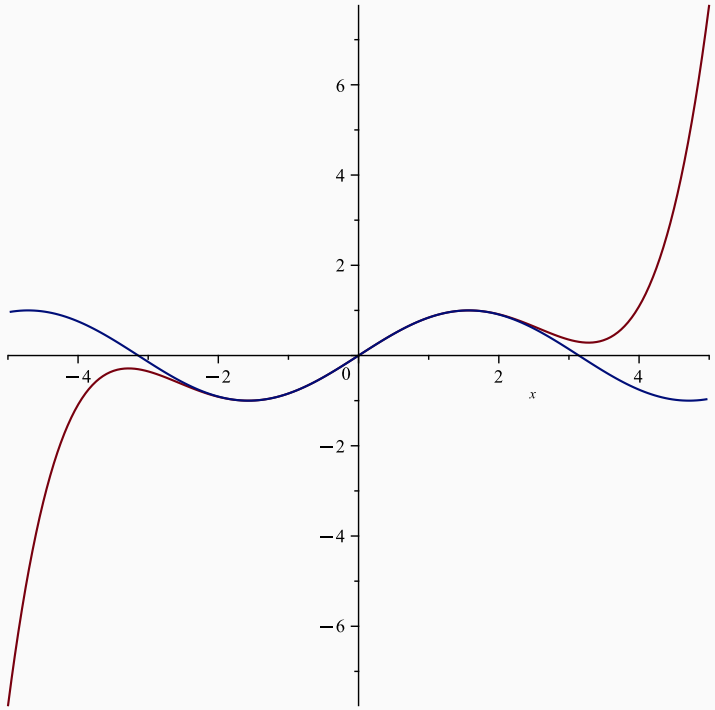


Typical current latencies in number of cycles (logarithmic scale). These figures vary from one processor to another, but the orders of magnitude remain similar.

- functions of one variable one can build from $\pm$ and $\times$: polynomials;
- No choice: approximate the functions by piecewise polynomials.

Approximating the functions by polynomials ?

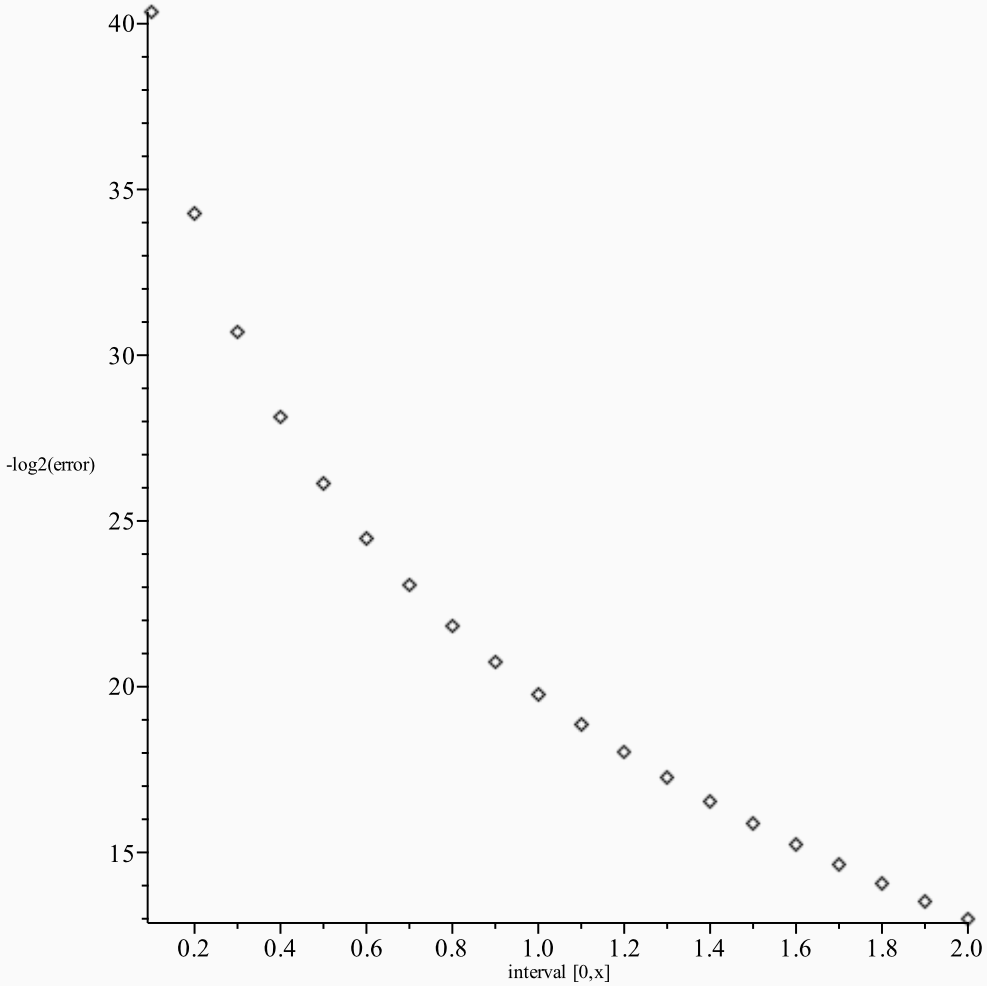
- polynomial approximation: valid in small interval only;
- need to reduce the initial argument to that interval;
- three steps:
  - range reduction;
  - polynomial evaluation;
  - reconstruction.



The sine function and its best degree-5 approximation in $[-\pi/2, \pi/2]$.

(clearly valid only in $[-\pi/2, \pi/2]$)

The advantage of small intervals



– $\log_2(error)$ of the best degree-5 approximation to e^t in $[0, x]$ as a function of x.

How do you compute an exponential?

I want to compute e^x, I have a polynomial approximation to exp in $[-\frac{\ln(2)}{2}, +\frac{\ln(2)}{2}]$.

- Range reduction: $x \to y = x - k\log(2)$, with $k \in \mathbb{Z}$ and $y \in [-\frac{\ln(2)}{2}, +\frac{\ln(2)}{2}]$
- Polynomial evaluation: get approximation z to e^y using the polynomial;
- Reconstruction: obtain $e^x = z \cdot 2^k$.

How do you compute a cosine?

I want to compute $\cos(x)$. I have polynomial approximations to sin and cos in $[-\frac{\pi}{4}, \frac{\pi}{4}]$.

Argument reduction: $x \to y = x - k\frac{\pi}{2}$, with $k \in \mathbb{Z}$ and $y \in [-\frac{\pi}{4}, \frac{\pi}{4}]$

Polynomial evaluation: compute $\begin{cases} c = \cos(y) \text{ (if } k \text{ is even), or} \\ s = \sin(y) \text{ (if } k \text{ is odd).} \end{cases}$

Reconstruction: obtain $\cos(x) = \begin{cases} c & \text{if} \quad k \bmod 4 = 0 \\ -s & \text{if} \quad k \bmod 4 = 1 \\ -c & \text{if} \quad k \bmod 4 = 2 \\ s & \text{if} \quad k \bmod 4 = 3; \end{cases}$

"Naive" range reduction: example with the constant $= \frac{\pi}{2}$

Naive reduction: we set $C = \mathrm{RN}(\pi/2)$, and we successively compute

- $k = \lceil x/C \rfloor$,
- $y = \mathrm{RN}(x - kC)$ with an FMA, or $\mathrm{RN}(x - \mathrm{RN}(kC))$ if no FMA..

Error analysis assuming FMA instruction is available

$$\begin{aligned} k = \lceil x/C \rfloor \quad &\Rightarrow \quad \tfrac{x}{C} - \tfrac{1}{2} \le k \le \tfrac{x}{C} + \tfrac{1}{2} \\ &\Rightarrow \quad |x - kC| \le \tfrac{C}{2}. \end{aligned}$$

$$\begin{aligned} \left| \mathrm{RN}\left(x - kC\right) - \left(x - k\tfrac{\pi}{2}\right) \right| \quad &\le \quad |\mathrm{RN}\left(x - kC\right) - (x - kC)| \\ &\qquad + \left|(x - kC) - \left(x - k\tfrac{\pi}{2}\right)\right| \\ &\le \quad u \cdot |x - kC| + k \cdot \left|C - \tfrac{\pi}{2}\right| \\ &\le \quad u\tfrac{C}{2} + ku. \end{aligned}$$

Hence:

- the absolute error is $\le u(k + \frac{C}{2})$ (the bound grows linearly with x, since k is proportional to x);
- the relative error depends on how small $\left|x - k\frac{\pi}{2}\right|$ can be.

$\rightarrow$ continued fractions.

In prectice the naive reduction can be very inaccurate

In binary64 arithmetic ($p = 53$):

- if $x = 355$:
  - y is 7230134.89 ulp away from the exact reduced argument if we do not use an FMA, and
  - 4084406.89 ulp away with an FMA.
- If $x = 37362253$, even with an FMA, y is 440183437673129 ulp away from the exact value.

(remember: for us good accuracy means final error not much larger than 0.5 ulp !)

Continued fraction convergents to $\frac{\pi}{2}$:

$$1, 2, \frac{3}{2}, \frac{11}{7}, \frac{344}{219}, \frac{355}{226}, \frac{51819}{32989}, \frac{52174}{33215}, \frac{260515}{165849}, \frac{573204}{364913}, \frac{4846147}{3085153}, \frac{5419351}{3450066}, \frac{37362253}{23785549}, \ldots$$

Cody and Waite reduction

Idea: approximate $\frac{\pi}{2}$ by two FP numbers C_1 and C_2 such that

- C_1 fits in $p - m$ bits, where m is the max. number of bits of k for which we want an accurate result $\rightarrow$ kC_1 will be a FP number
- $C_2 = RN(\frac{\pi}{2} - C_1)$, so that $C_1 + C_2$ represents $\frac{\pi}{2}$ with significantly more than p bits of precision.
- More precisely: $\left|\frac{\pi}{2} - C_1 - C_2\right| < 2^{-2p+m}$.

Then we compute (here with an FMA)

$$\mathrm{RN}\left(\mathrm{RN}\left(x - kC_1\right) - kC_2\right).$$

Cody and Waite reduction

We compute

$$\mathrm{RN}\,(\,\mathrm{RN}\,(x - kC_1) - kC_2).$$

- kC_1 is a FP number;
- as x and kC_1 are very near, their difference is a FP number (Sterbenz) $\rightarrow \mathrm{RN}\,(x - kC_1) = x - kC_1$. Hence we obtain

$$\mathrm{RN}\,(x - k(C_1 + C_2)).$$

- same error analysis as the naive reduction:

$$\left|\mathrm{RN}\,(x - k(C_1 + C_2)) - \left(x - k\frac{\pi}{2}\right)\right| < u\frac{C}{2} + ku^2 2^m.$$

- With $m = 26$, if $x = 355$, y is $0.108\,\mathrm{ulp}$ away from the exact result (still bad for $x = 37362253$, but not as much).

Generalizations, variants

- More than 2 constants: one may have C_2 fit in $p - m$ bits too, and approximate $\frac{\pi}{2}$ by a sum $C_1 + C_2 + C_3$. One then evaluates

$$\mathrm{RN}\left(((x - kC_1) - kC_2) - kC_3\right);$$

- systematic study of the numbers $\gamma \approx \frac{\pi}{2}$ such that $x - k\gamma$ is a FP number;
- express the reduced argument in double-word arithmetic.

After range reduction: using a polynomial approximation to the function

Two issues:

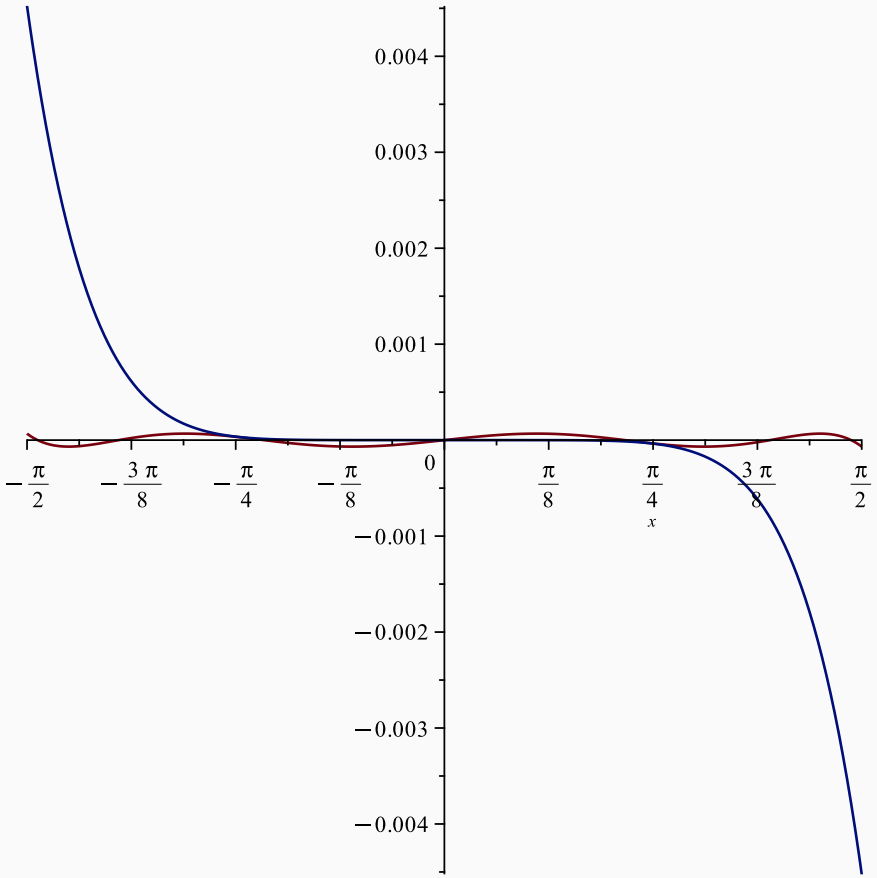
- choosing the right polynomial;
- evaluating it quickly and accurately.

Function f, approximated by $p(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_n x^n$; frequently evaluated as

$$a_0 + x \cdot (a_1 + x \cdot (a_2 + x \cdot (\cdots (a_{n-1} + x \cdot a_n))) \cdots)$$

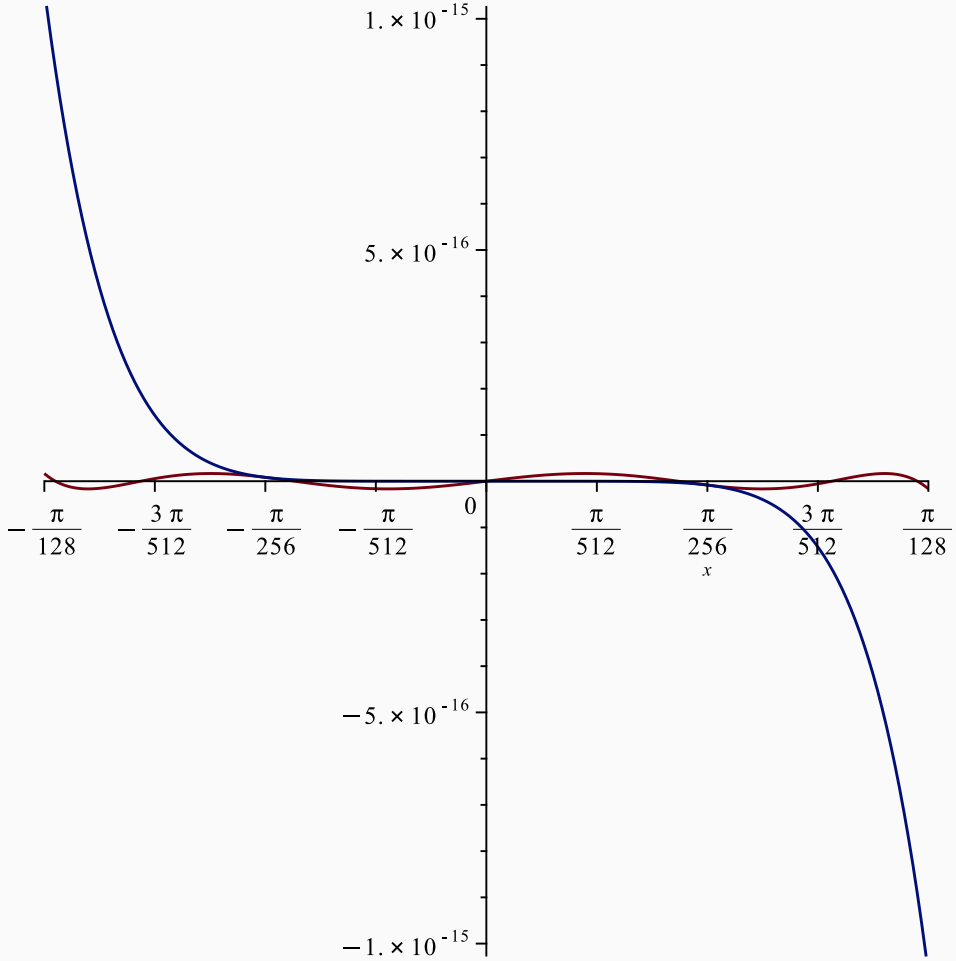
(called Horner's scheme (1819). Was already known by Newton, and probably by chinese mathematicians centuries before Horner)

Forget about Taylor series



- error of degree-5 Taylor series of sin vs error of best "minimax" degree-5 approximation in $[-\frac{\pi}{2}, +\frac{\pi}{2}]$;
- Taylor series are local best approximations: they cannot compete on a whole interval.

Taking a much smaller domain does not change the difference



Error of degree-5 Taylor series of sin vs error of best "minimax" degree-5 approximation in $[-\frac{\pi}{128}, +\frac{\pi}{128}]$.

Minimax approximation of functions by polynomials

- $\mathcal{P}_n = \{\text{polynomials of degree } \leq n \text{ with coefficients } \in \mathbb{R}\}$;
- L^∞ norm (also called supremum norm):

$$\|g\|_\infty = \sup_{x \in [a,b]} |g(x)|;$$

- function f, interval $[a, b]$;
- first, we look for $P^* \in \mathcal{P}_n$ such that

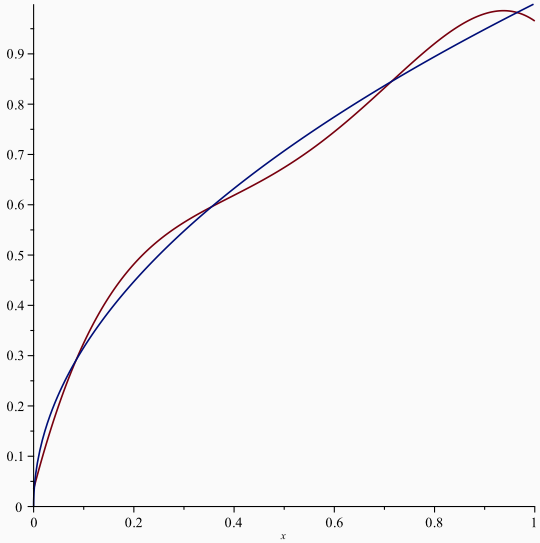
$$\|f - P^*\|_\infty = \min_{Q \in \mathcal{P}_n} \|f - Q\|_\infty.$$

Continuous world: everything was done in the 19th and early 20th centuries

Theorem 1 (Weierstrass, 1885)

Let f be a continuous function on $[a, b]$. For any $\epsilon > 0$ there exists a polynomial p such that $\|p - f\|_{\infty,[a,b]} \leq \epsilon$.

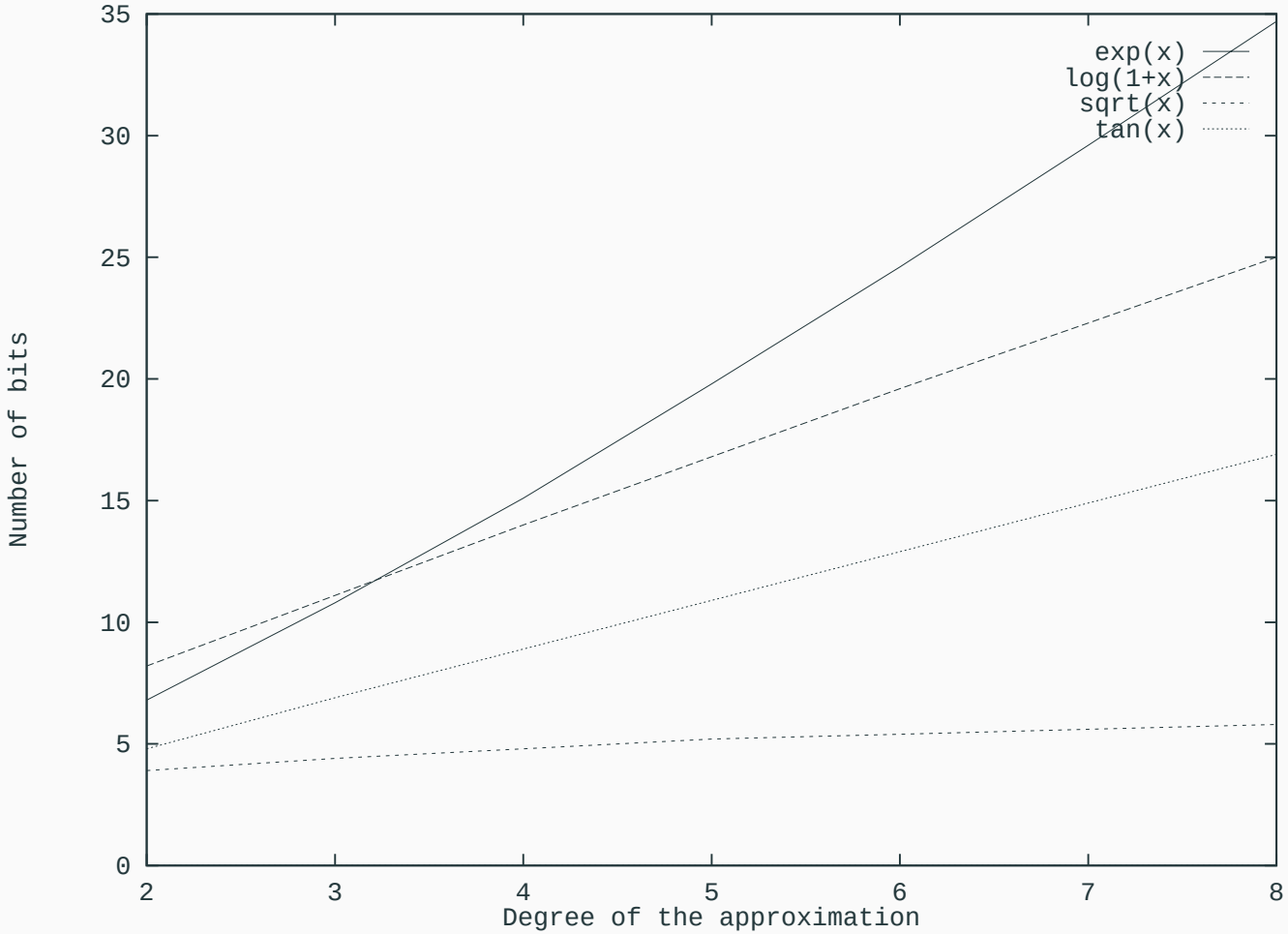
- $\rightarrow$ so it's not hopeless... but the theorem gives no clue on the necessary degree n to reach a given accuracy;
- for some functions it can be quite large.



The square root and its best degree-4 approximation in $[0, 1]$.

(with the exponential function, the two curves would be undistinguishable at this scale)

Some functions may need high-degree polynomials



$-\log_2(\mathit{error})$ of the minimax polynomial approximations to various functions on $[0, 1]$, as a function of the degree.

Continuous world: everything was done in the 19th and early 20th centuries

Theorem 2 (Kirchberger 1902 – frequently attributed to Chebyshev)

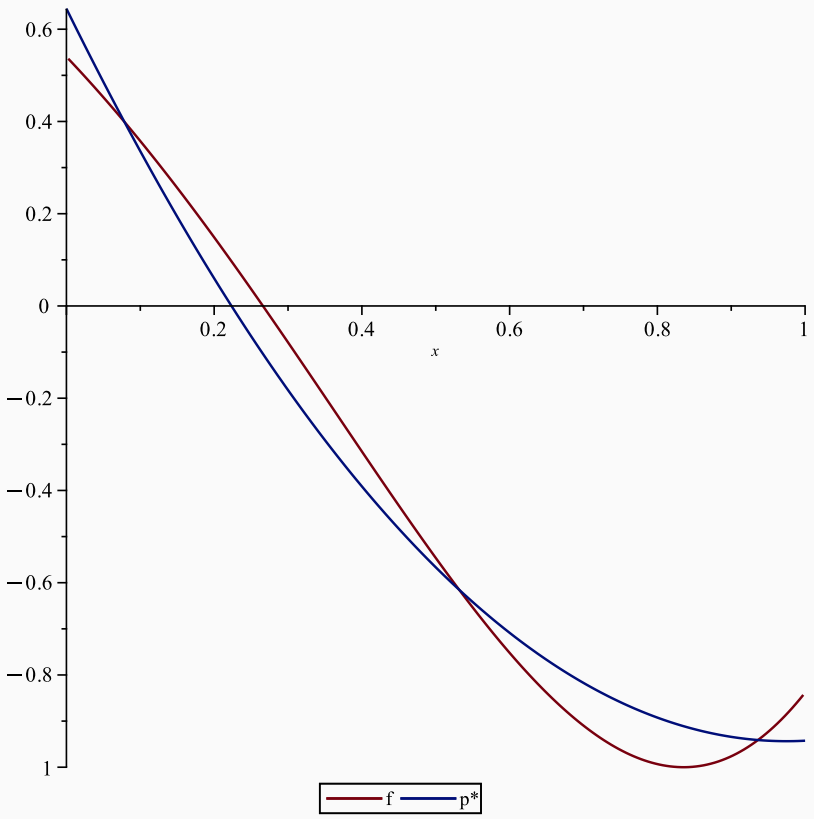
$P^* \in \mathcal{P}_n$ is the minimax degree-n approximation to f on $[a, b]$ if and only if there exist at least $n + 2$ points

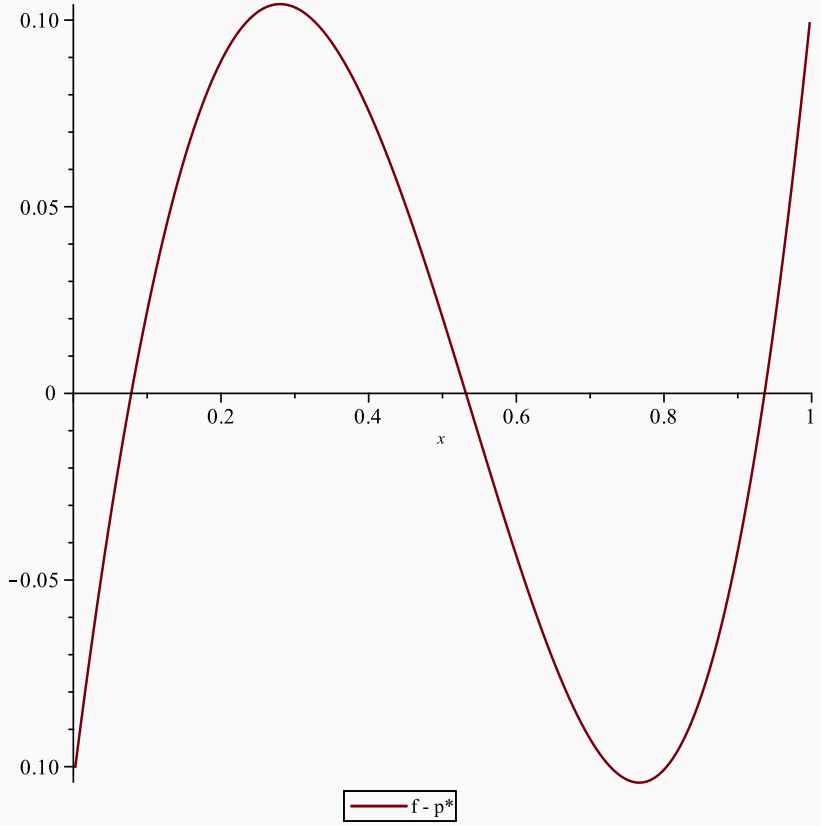
$$a \leq x_0 < x_1 < x_2 < \cdots < x_{n+1} \leq b$$

such that:

$$p^*(x_i) - f(x_i) = (-1)^i \left[p^*(x_0) - f(x_0)\right] = \pm \left\| f - p^* \right\|_\infty .$$

Example: $f(x) = \cos(x + e^x)$, degree 2, interval $[0, 1]$

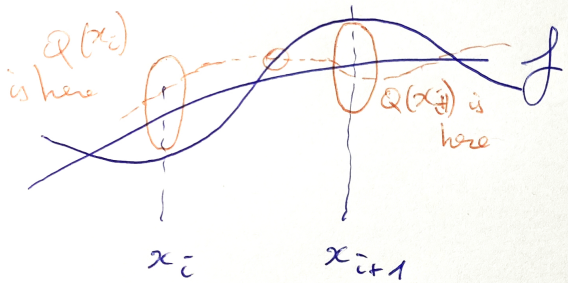




Proof that the condition is sufficient

Let $P \in \mathcal{P}_n$ that satisfies the condition

Imagine there is a $Q \in \mathcal{P}_n$ such that $\|Q-f\|_\infty < \|P-f\|_\infty$



We necessarily have $|Q(x_i)-f(x_i)| < |P(x_i)-f(x_i)|$

hence
- if $P(x_i) > f(x_i)$ then $Q(x_i) < P(x_i)$
- if $P(x_i) < f(x_i)$ then $Q(x_i) > P(x_i)$

as the sign of $P-f$ changes between x_i and x_{i+1},

so does the sign of $Q-f$

$\rightarrow$ $Q-P$ has a root between x_i and x_{i+1}

$n+2$ points $\rightarrow$ $n+1$ roots

which is impossible, since $Q-P$ is a nonzero polynomial of degree $\leq n$.

Continuous world: everything was done in the 19th and early 20th centuries

Remez algorithm (1934): iteratively builds the set of points $x_0, \ldots, x_{n+1}$ of Kirchberger's theorem.

- Start from an initial set of points $x_0, x_1, \ldots, x_{n+1}$ in $[a, b]$. (can be arbitrary, but $x_i = \frac{a+b}{2} + \frac{(b-a)}{2} \cos\left(\frac{i\pi}{n+1}\right), 0 \leq i \leq n+1$, called Chebyshev points, is in general a good choice)
- Consider the linear system of equations

$$\begin{cases} p_0 + p_1 x_0 + p_2 x_0^2 + \cdots + p_n x_0^n - f(x_0) &= +\epsilon \\ p_0 + p_1 x_1 + p_2 x_1^2 + \cdots + p_n x_1^n - f(x_1) &= -\epsilon \\ p_0 + p_1 x_2 + p_2 x_2^2 + \cdots + p_n x_2^n - f(x_2) &= +\epsilon \\ \cdots & \cdots \\ p_0 + p_1 x_{n+1} + \cdots + p_n x_{n+1}^n - f(x_{n+1}) &= (-1)^{n+1}\epsilon. \end{cases} \quad (1)$$

$n+2$ equations, with $n+2$ unknowns: $p_0, p_1, p_2, \ldots, p_n$ and ϵ. Nonzero determinant (Vandermonde matrix) $\rightarrow$ exactly one solution $(p_0, p_1, \ldots, p_n, \epsilon)$.

Continuous world: everything was done in the 19th and early 20th centuries

$\rightarrow$ gives a polynomial $P(x) = p_0 + p_1 x + \cdots + p_n x^n$.

- We now compute the set of points y_i in $[a, b]$ where $P - f$ has its extremes, and we start again (step 2), replacing the $x_i's$ by the y_i's.

In practice: extremely fast convergence.

Illustration: degree-4 approximation to $\sin(\exp(x))$ in $[0, 2]$

- we start with 0, 0.1909830057, 0.6909830062, 1.309016994, 1.809016994, 2 (heuristic: Chebyshev points);
- the corresponding linear system is

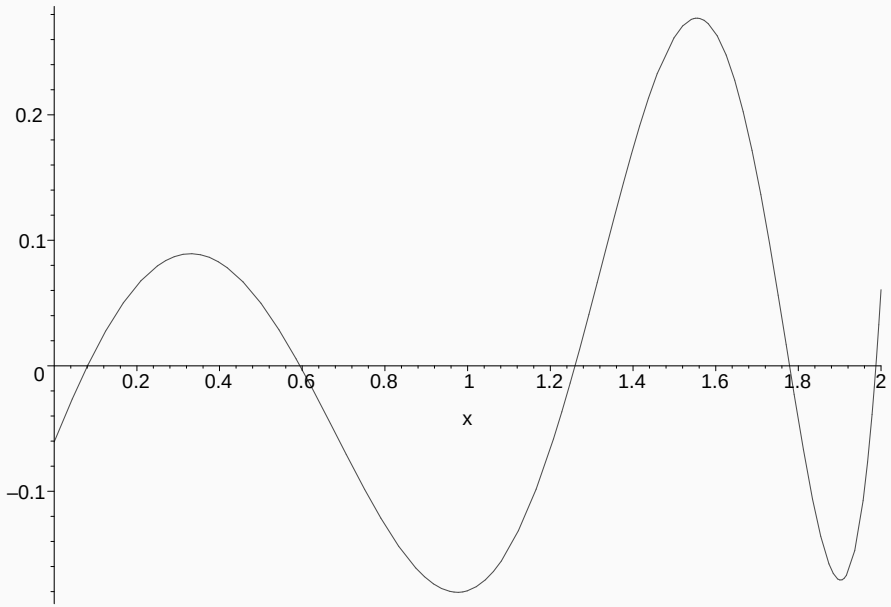
$$\begin{cases}
p_0 & -0.8414709848 & & & = & \epsilon \\
p_0 & +0.1909830057p_1 & +0.03647450847p_2 & +0.00696601126p_3 & & \\
 & & +0.00133038977p_4 & -0.9357708449 & = & -\epsilon \\
p_0 & +0.6909830062p_1 & +0.4774575149p_2 & +0.3299150289p_3 & & \\
 & & +0.2279656785p_4 & -0.9110882027 & = & \epsilon \\
p_0 & +1.309016994p_1 & +1.713525491p_2 & +2.243033987p_3 & & \\
 & & +2.936169607p_4 & +0.5319820928 & = & -\epsilon \\
p_0 & +1.809016994p_1 & +3.272542485p_2 & +5.920084968p_3 & & \\
 & & +10.70953431p_4 & +0.1777912944 & = & \epsilon \\
p_0 & +2p_1 & +4p_2 & +8p_3 & & \\
 & & +16p_4 & -0.8938549549 & = & -\epsilon.
\end{cases}$$

- solving this system gives the polynomial:

$$P^{(1)}(x) = 0.7808077493 + 1.357210937x$$
$$-0.7996276765x^2 - 2.295982186x^3 + 1.189103547x^4.$$

Illustration: degree-4 approximation to $\sin(\exp(x))$ in $[0, 2]$

- difference $P^{(1)}(x) - \sin(\exp(x))$:

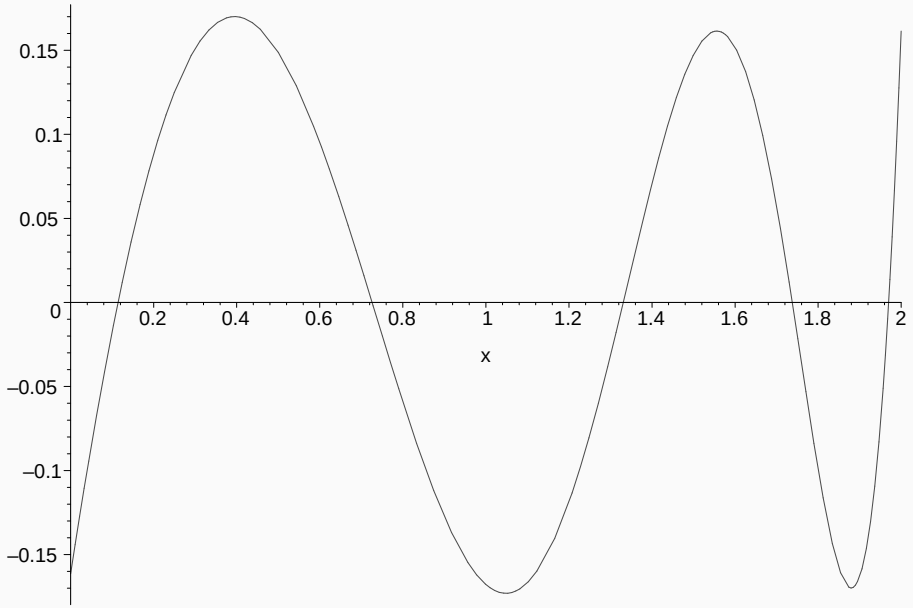


- extremes of $P^{(1)}(x) - \sin(\exp(x))$ in $[0, 2]$: 0, 0.3305112886, 0.9756471625, 1.554268282, 1.902075854, 2.
- Solving the linear system associated to this list of points gives the polynomial:

$$P^{(2)}(x) = 0.6800889007 + 2.144092090x$$
$$-1.631367834x^2 - 2.226220290x^3 + 1.276387351x^4.$$

Illustration: degree-4 approximation to $\sin(\exp(x))$ in $[0, 2]$

- difference $P^{(2)}(x) - \sin(\exp(x))$:



- the extreme values of $|P^{(2)}(x) - \sin(\exp(x))|$ are very close together: $P^{(2)}$ already "almost" satisfies the condition of Kirchberger's theorem;
- extremes of $P^{(2)}(x) - \sin(\exp(x))$ in $[0, 2]$: 0, 0.3949555564, 1.048154245, 1.556144609, 1.879537115, 2 $\rightarrow$ the linear system associated to these points gives a polynomial $P^{(3)}$, etc.

But the world of Floating-Point numbers is not continuous. . .

If we want ultimate accuracy, i.e., correct rounding (or even just an error not larger than $\approx 1\,\text{ulp}$), this will not work:

- by approximating the coefficients of the minimax polynomial by FP numbers, we already loose a significant amount of accuracy;
- need to take into account the error due to evaluating the polynomial.

Example: function $\log(1+x)$ in $[0,1]$

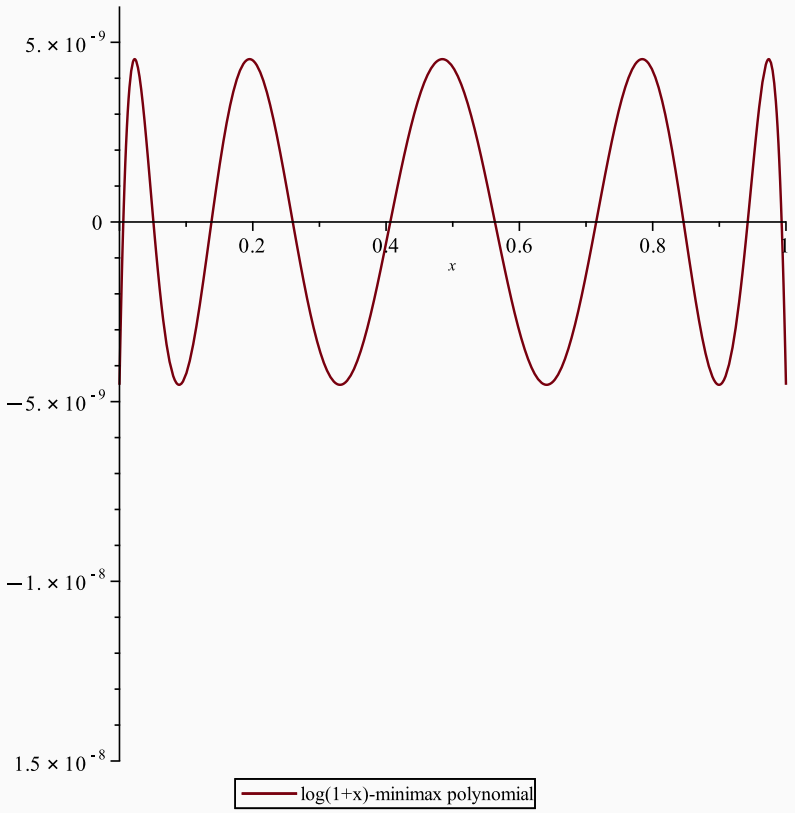
- Goal: binary32/single precision arithmetic ($\beta = 2$, $p = 24$);
- Withe the Remez algorithm, we get a degree-9 polynomial;

$$
\begin{aligned}
P^*(x) = {} & 4.531262676417885304508397507377211944683978417446495645952964327098 7 \cdot 10^{-9} \\
& + 0.999999025258539069369331280889443153540235373687738534675639041879165305 \cdot x \\
& - 0.499965301733306862646350263286494969422243084300307456583308608494202815 5 \cdot x^2 \\
& + 0.332849806189948672280944634938400232746236110583281184032142726147068820 \cdot x^3 \\
& - 0.246517965788047515383441886060448202153937598635915727063060602574084424 \cdot x^4 \\
& + 0.185154569966668361477975368769654445113328710403271118930768041265705015 \cdot x^5 \\
& - 0.126057455452415889884958065630540564789570519172717287039473166654700319 \cdot x^6 \\
& + 0.067294552999244180561977536465001023978793925662150291856930540995123804 2 \cdot x^7 \\
& - 0.023455350690510775380743240497604729489911455890469902806116859364420207 61 \cdot x^8 \\
& + 0.003845299809826069022496755870766178552568672023224498117158896293738444 90 \cdot x^9 .
\end{aligned}
$$

- error $\approx 4.53 \times 10^{-9}$.

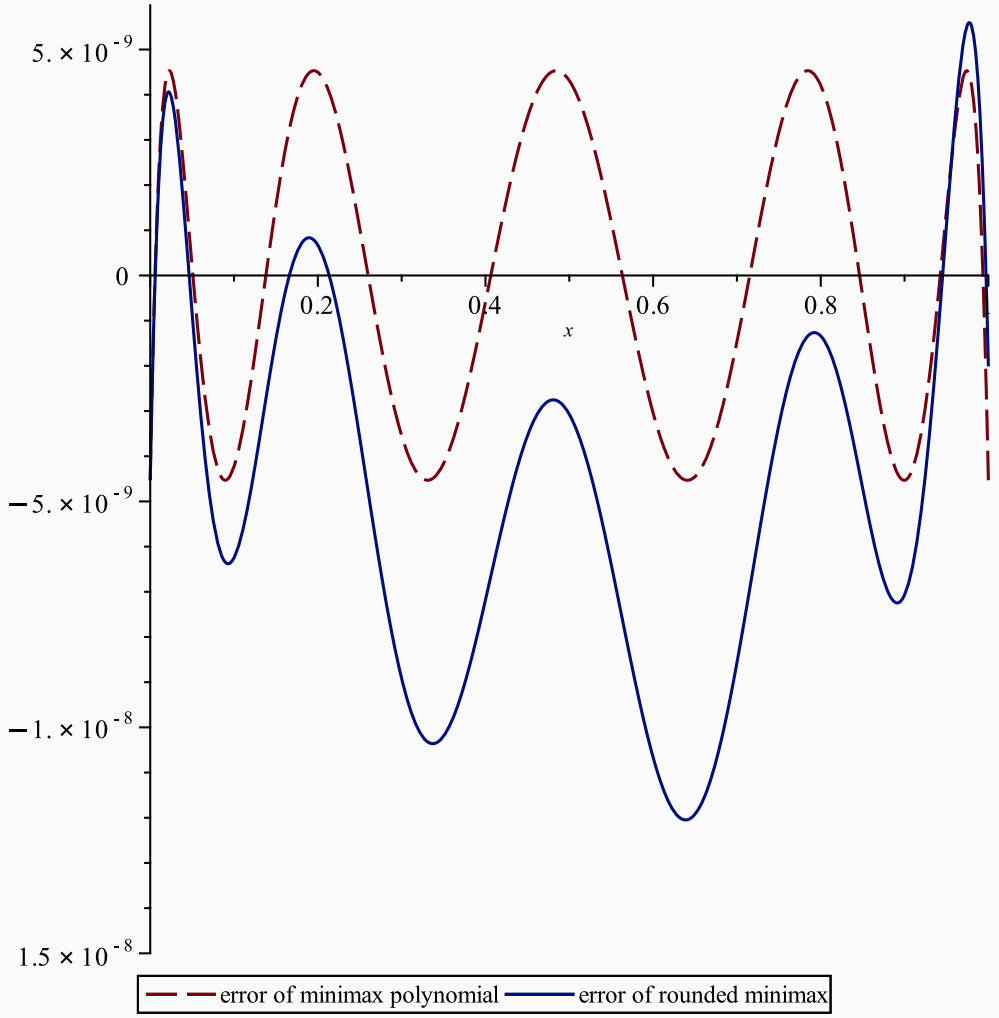
Example: function $\log(1+x)$ in $[0,1]$

- error curve $\log(1+x) - P^*(x)$: nice illustration of Kirchberger's theorem



- Problem: P^* has real coefficients;
- I want to implement the function in binary32 arithmetic ($\beta = 2$, $p = 24$). What happens if I round each coefficient of P^* to the nearest FP number?

The disaster...



Error multiplied by ≈ 2.66

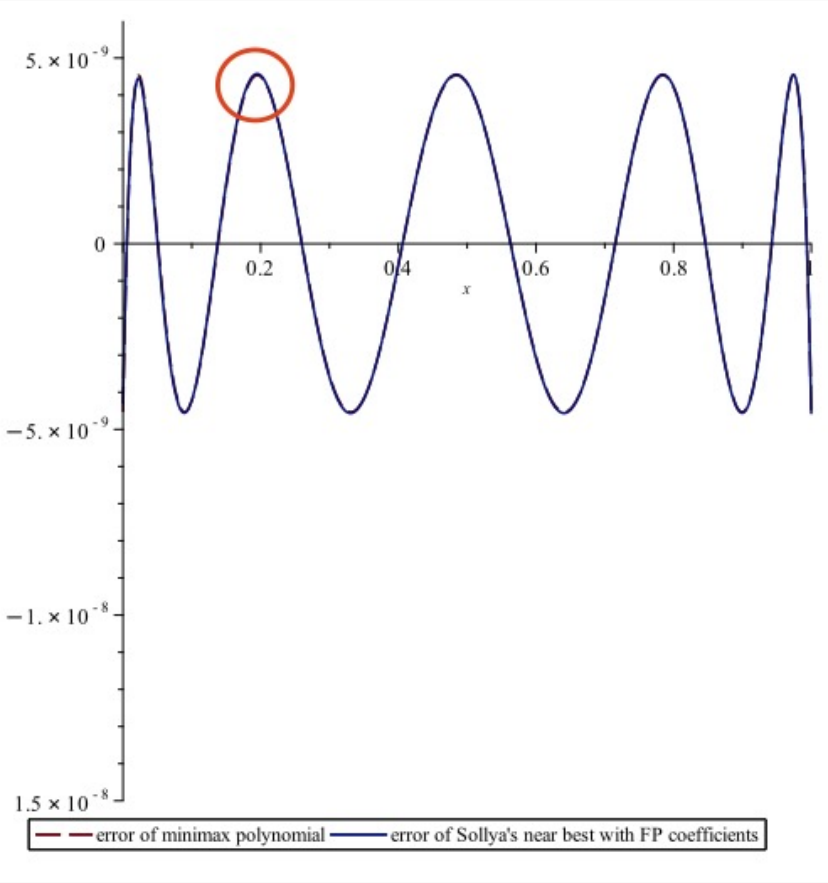
Near best polynomial approximations with FP coefficients

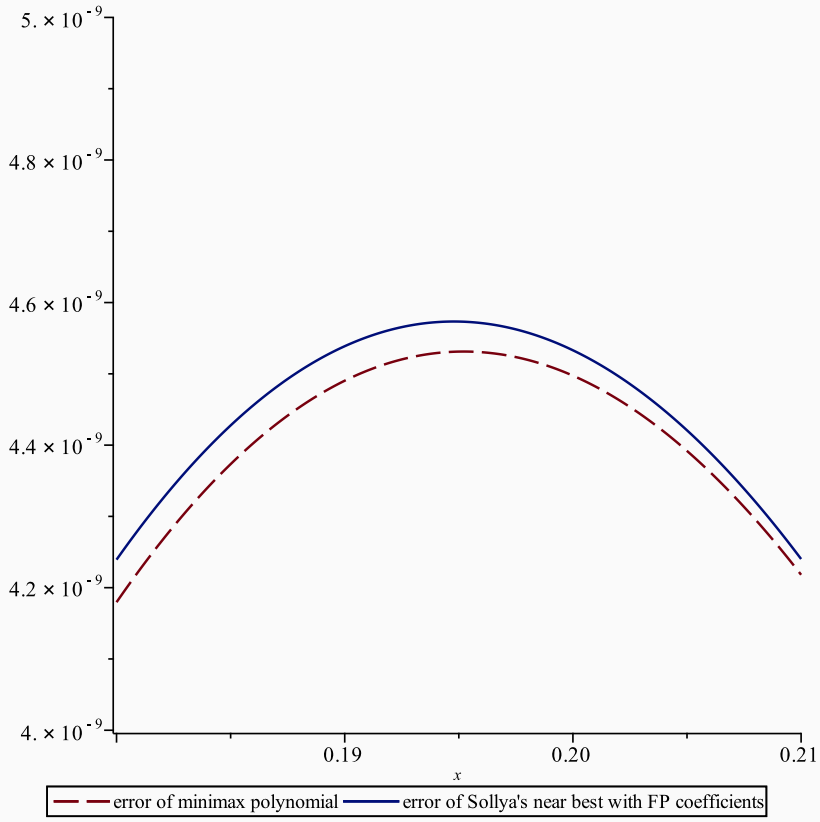
- algorithms that give near-best polynomials approximations under the constraint that the coefficients be FP numbers;
- more complex than Remez yet reasonable cost (and done once for all): *Brisebarre and Chevillard, Efficient Polynomial L^∞-Approximations, 18th IEEE Symposium on Computer Arithmetic, 2007. https://hal.inria.fr/inria-00119513*
- still very active domain: various other constraints on coefficients, norms other than L^∞, rational approximations, taking into account the evaluation error when choosing the polynomial...
- Sollya software (Chevillard, Joldes, Lauter). Widely used for designing function libraries.

  `https://www.sollya.org`

  It also provides a Certified supnorm: proven bound on $\|f - p\|_\infty$.

With our example of function $\log(1+x)$ in $[0,1]$





Sollya prompt: `P3 = fpminimax(log(1+x),9,[|24...|],[0;1],absolute);`

Error Sollya / Error minimax ≈ 1.012

$\rightarrow$ we recover almost all the loss due to the discretization of the polynomial.

Polynomial evaluation error

- Horner scheme
- use of interval arithmetic to know where each intermediate variable can lie;
- standard model to bound the error of each operation;
- can be very accurate if initial interval cut into many subintervals (a separate study for each subinterval);
- Gappa (`https://gappa.gitlabpages.inria.fr`), designed by G. Melquiond does this automatically and can generate a formal proof.

But what should be recommended?

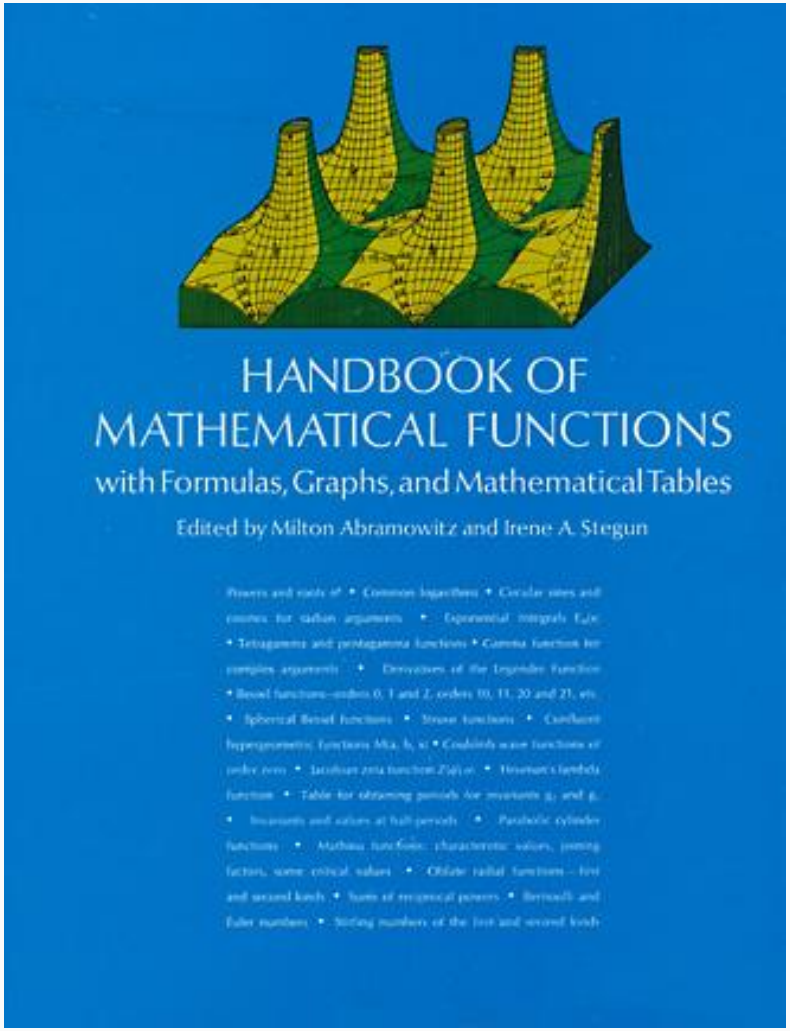
- reproducibility, portability $\rightarrow$ the result of $\sin(x)$, $\cos(x)$, $\exp(x)\ldots$ should be uniquely specified (no "fuzzy" specification of the form "error $< \epsilon$");
- specifying the result of an algorithm? Dangerous:
  - might make progress of elementary function algorithms difficult;
  - if somebody comes up with functions "better than the standard", the standard is dead

$\rightarrow$ what should be required is that we return the best possible result, i.e., correct rounding of the exact result.

However: difficult because of the table maker's dilemma.

The Table Maker's Dilemma

The Table Maker's Dilemma



The Table Maker's Dilemma

Table 4.2 NATURAL LOGARITHMS

x	$\ln x$	x	$\ln x$	x	$\ln x$
0.900	−0.10536 05156 578263	0.950	−0.05129 32943 875505	1.000	0.00000 00000 000000
0.901	−0.10425 00213 737991	0.951	−0.05024 12164 367467	1.001	0.00099 95003 330835
0.902	−0.10314 07589 195134	0.952	−0.04919 02441 907717	1.002	0.00199 80026 626731
0.903	−0.10203 27255 651516	0.953	−0.04814 03753 279349	1.003	0.00299 55089 797985
0.904	−0.10092 59185 899606	0.954	−0.04709 16075 338505	1.004	0.00399 20212 695375
0.905	−0.09982 03352 822109	0.955	−0.04604 39385 014068	1.005	0.00498 75415 110391
0.906	−0.09871 59729 391577	0.956	−0.04499 73659 307358	1.006	0.00598 20716 775475
0.907	−0.09761 28288 670004	0.957	−0.04395 18875 291828	1.007	0.00697 56137 364252
0.908	−0.09651 09003 808438	0.958	−0.04290 75010 112765	1.008	0.00796 81696 491769
0.909	−0.09541 01848 046582	0.959	−0.04186 42040 986988	1.009	0.00895 97413 714719
0.910	−0.09431 06794 712413	0.960	−0.04082 19945 202551	1.010	0.00995 03308 531681
0.911	−0.09321 23817 221787	0.961	−0.03978 08700 118446	1.011	0.01093 99400 383344
0.912	−0.09211 52889 078057	0.962	−0.03874 08283 164306	1.012	0.01192 85708 652738
0.913	−0.09101 93983 871686	0.963	−0.03770 18671 840115	1.013	0.01291 62252 665463
0.914	−0.08992 47075 279870	0.964	−0.03666 39843 715914	1.014	0.01390 29051 689914
0.915	−0.08883 12137 066157	0.965	−0.03562 71776 431511	1.015	0.01488 86124 937507
0.916	−0.08773 89143 080068	0.966	−0.03459 14447 696191	1.016	0.01587 33491 562901
0.917	−0.08664 78067 256722	0.967	−0.03355 67835 288427	1.017	0.01685 71170 664229
0.918	−0.08555 78883 616466	0.968	−0.03252 31917 055600	1.018	0.01783 99181 283310
0.919	−0.08446 91566 264500	0.969	−0.03149 06670 913708	1.019	0.01882 17542 405878
0.920	−0.08338 16089 390511	0.970	−0.03045 92074 847085	1.020	0.01980 26272 961797
0.921	−0.08229 52427 268302	0.971	−0.02942 88106 908121	1.021	0.02078 25391 825285
0.922	−0.08121 00554 255432	0.972	−0.02839 94745 216980	1.022	0.02176 14917 815127
0.923	−0.08012 60444 792849	0.973	−0.02737 11967 961320	1.023	0.02273 94869 694894
0.924	−0.07904 32073 404529	0.974	−0.02634 39753 396020	1.024	0.02371 65266 173160

The Table Maker's Dilemma

Consider the binary64 FP number ($\beta = 2, p = 53$)

$$x = \frac{8520761231538509}{2^{62}}$$

We have

$$2^x = \left(9018742077413030.9999999999999999805240837303\cdots\right) \times 2^{-53}$$

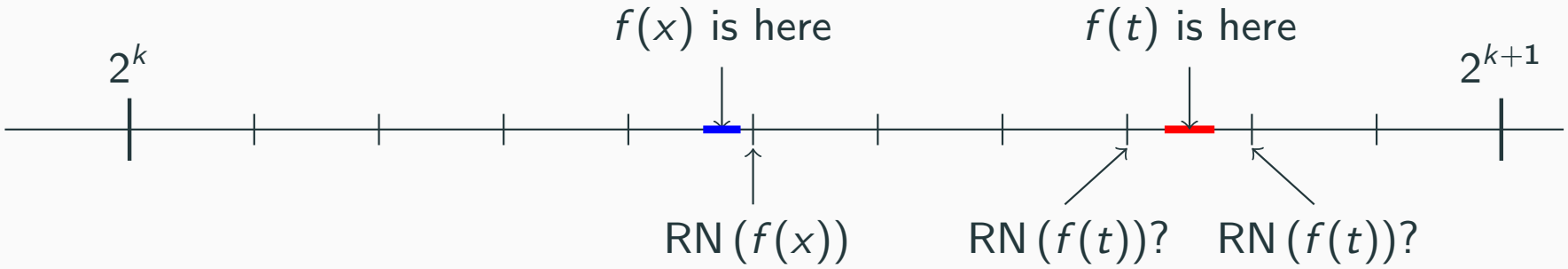
So what ?

Hardest-to-round case for function 2^x and binary64 FP numbers.

Correct rounding of the elementary functions

- radix 2, precision p;
- FP number x and integer m (with $m > p$) $\rightarrow$ one can compute an approximation y to $f(x)$ whose error on the significand is $\leq 2^{-m}$.
- can be done with a possible wider format, or using algorithms such as TwoSum, TwoMult, Double-Word (or triple-word) arithmetic, etc.
- getting a correct rounding of $f(x)$ from y: not possible if $f(x)$ is too close to a breakpoint: a point where the rounding function changes.

Correct rounding of the elementary functions



From the knowledge that $f(x)$ lies in the blue interval, we can deduce the value of $\mathrm{RN}(f(x))$. However, knowing that $f(t)$ lies in the red interval does not allow us to know if $\mathrm{RN}(f(t))$ is the FP number below $f(t)$ or the FP number above it.

We are in trouble when $f(x)$ has the form

- RN rounding function,

$$\overbrace{\underbrace{1.xxxxx\cdots xxx}_{p \text{ bits}}\,1000000\cdots 000000}^{m \text{ bits}}\,xxx\cdots$$

or

$$\overbrace{\underbrace{1.xxxxx\cdots xxx}_{p \text{ bits}}\,0111111\cdots 111111}^{m \text{ bits}}\,xxx\cdots\,;$$

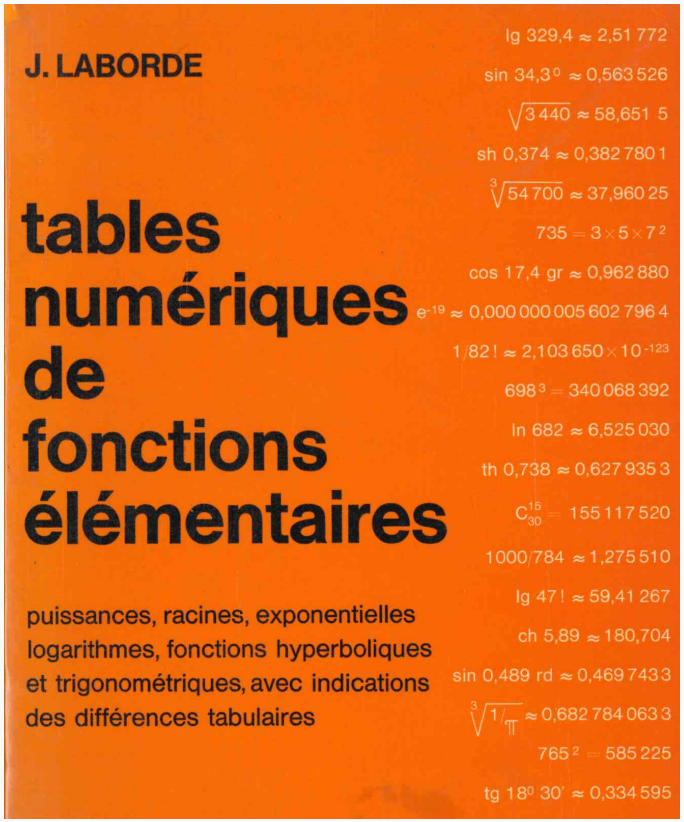
- other rounding functions,

$$\overbrace{\underbrace{1.xxxxx\cdots xxx}_{p \text{ bits}}\,0000000\cdots 000000}^{m \text{ bits}}\,xxx\cdots$$

or

$$\overbrace{\underbrace{1.xxxxx\cdots xxx}_{p \text{ bits}}\,1111111\cdots 111111}^{m \text{ bits}}\,xxx\cdots\,.$$

An example from the sixties



- no pocket calculators $\rightarrow$ tables of functions + interpolation were very useful;
- when designing his tables, Laborde was, for rare input values, unable to decide in which direction a result needed to be rounded;
- for these values, he used to choose a random rounding direction, and he would carefully record them to detect plagiarism of his tables.

Lindemann's theorem (1882)

- algebraic number: root of a nonzero polynomial with integer coefficients;
- the algebraic numbers are a field (proof not so easy); the $\sqrt{}$ of an algebraic number is algebraic (straightforward: $\sum a_i X^i \to \sum a_i X^{2i}$);
- transcendental number = not algebraic. Historical examples: $\sum_{k=0}^{\infty} 10^{-k!}$, π, e;
- the FP numbers are rational $\Rightarrow$ they are algebraic.

Theorem 3

If $z \in \mathbb{C}$ is a nonzero algebraic number then e^z is transcendental.

Consequences of Lindemann's theorem

- The sine, cosine of a nonzero algebraic number is transcendental.
  - x algebraic nonzero $\to$ ix algebraic nonzero $\to$ $u = e^{ix}$ transcendental;
  - if $\sin(x)$ was algebraic, then $w = 2i\sin(x) = u - 1/u$ would be algebraic too;

    (note that $w = u - 1/u$ implies that $u^2 - wu - 1 = 0$)

  - therefore $(w \pm \sqrt{w^2+4})/2$ would be algebraic too;
  - but $u \in (w \pm \sqrt{w^2+4})/2 \to$ contradiction;
- obviously similar for hyperbolic sine and cosine;
- true for the tangent function, since $\sin(x) = \tan(x)/\sqrt{\tan^2 x + 1}$;
- obviously true for their inverses: log, arcsin, arctan, ...

Finding m beyond which there is no problem ?

- function f: sin, cos, arcsin, arccos, tan, arctan, exp, log, sinh, cosh,
- Lindemann's theorem $\rightarrow$ except for straightforward cases (e^0, ln(1), sin(0), ...), if x is a FP number, there exists an m, say m_x, s.t. rounding the m_x-bit approximation $\Leftrightarrow$ rounding $f(x)$;
- finite number of FP numbers $\rightarrow \exists m_{\max} = \max_x(m_x)$ s.t. $\forall x$, rounding the $m_{\max}$-bit approximation to $f(x)$ is equivalent to rounding $f(x)$;
- this reasoning does not give any hint on the order of magnitude of $m_{\max}$. Could be huge.

A bound derived from a result due to Baker (1975)

- $\alpha = i/j$, $\beta = r/s$, with $i, j, r, s < 2^p$;
- $C = 16^{200}$;

$$|\alpha - \log(\beta)| > (p2^p)^{-Cp \log p}$$

Application: To evaluate ln et exp in double precision ($p = 53$) with correct rounding, it suffices to compute an approximation accurate to around

$$10^{244} \text{ bits}$$

oooops. . .

Some improvement

Definition 4 (Weil height)

Let α be an algebraic number of degree n and $P(x) = \sum_{i \leq n} p_i x^i$ be its minimal polynomial. Let $P(x) = p_n \cdot \prod_{i \leq n}(x - \alpha_i)$ be the factorization of P over the complex numbers. Then the Weil height of α is

$$H(\alpha) = \left(p_n \cdot \prod_{i \leq n} \max(1, |\alpha_i|) \right)^{\frac{1}{n}}.$$

Note: if $\alpha = b/a \in \mathbb{Q}$ with $\gcd(a, b) = 1$ then $H(\alpha) = \max\{|a|, |b|\}$.

Some improvement

Theorem 5 (Y. Nesterenko and M. Waldschmidt, specialized here to the rational numbers)

Let α and α' be rational numbers. Let θ be an arbitrary non-zero real number. Let A, A', and E be positive real numbers with[a]

$$E \geq e, \quad A \geq \max\left(H(\alpha), e\right), \quad A' \geq H(\alpha').$$

Then

$$\begin{aligned}&\left|e^{\theta} - \alpha\right| + |\theta - \alpha'| \geq \\ &\exp\Big\{-211 \cdot \Big(\ln A' + \ln\ln A + 2\ln(E \cdot \max\{1, |\theta|\}) + 10\Big) \\ &\cdot \Big(\ln A + 2E|\theta| + 6\ln E\Big) \cdot \Big(3.7 + \ln E\Big) \cdot \Big(\ln E\Big)^{-2}\Big\}.\end{aligned}$$

[a] Here, $e = 2.718\cdots$ is the base of the natural logarithms.

Some improvement

For the evaluation of exponentials in binary64 ($p = 53$), we find that $m = 7,290,678$ suffices.

Impossible $\rightarrow$ too expensive

In recent years, significant further improvements of the theoretical bounds, but they remain quite large...

Best current theoretical results

Estimates of current theoretical upper bounds or the hardness to round for exp, trigonometric and hyperbolic functions in the binary128 format. For each function f, we report the values θ_f such that, over a given binade, the hardness to round f is less than $\theta_f \cdot 113$.

Binade	exp	sin	cos	sinh	cosh	tan	cot	tanh	coth
$[1/8, 1/4)$	226	1371	1359	688	682	303	302	303	298
$[1/4, 1/2)$	297	2070	2058	1062	1057	410	410	409	405
$[1/2, 1)$	403	3288	3281	1698	1695	604	604	600	598
$[1, 2)$	593	5678	6481	2889	2889	1194	1196	931	929
$[2, 4)$	920	11408	10266	5285	5285	1854	1855	1507	1504
$[4, 8)$	1485	20395	20395	10155	10155	3361	3360	2634	2631

But in practice it's much less. Let us see why.

Approximation with error $\leq 2^{-p-k+1}$ on the significand

$\frac{f(x)}{2^{e_{f(x)}}} = \boxed{1.xxxx \cdots xxxxxxx}$

p bits, followed by:

$k=2$	$k=3$	$k=4$
00	000	0000
01	001	0001
10	010	0010
11	011	0011
	100	0100
	101	0101
	110	0110
	111	0111
		1000
		1001
		1010
		1011
		1100
		1101
		1110
		1111
Probability of failure: 1/2	1/4	1/8

k bits $\rightarrow$ probability of failure 2^{1-k}

Rule of thumb

If we approximate the significand of $f(x)$ with error $\leq 2^{-p-k+1}$ (roughly speaking, if we compute a $p+k$-bit approximation to $f(x)$), the probability of not being able to deduce $\mathrm{RN}(f(x))$ is around 2^{1-k}.

- exceptions to that rule: if x is tiny, not all bit strings are possible in $\sin(x)$, $\exp(x)$, etc. just after the first p bits. For instance,

$$\sin(1.xxxx\cdots x1 \times 2^{-p}) = 1.xxxx\cdots x01111111111\cdots \times 2^{-p}.$$

- in practice this is not a problem, just choose polynomial approximations where the lowest order term is exactly x for sin or sinh or $\log(1+x)$, 1 for $\exp(x)$, etc. They will automatically deliver correct rounding when x is tiny enough;
- the rule is essential for designing efficient algorithms;

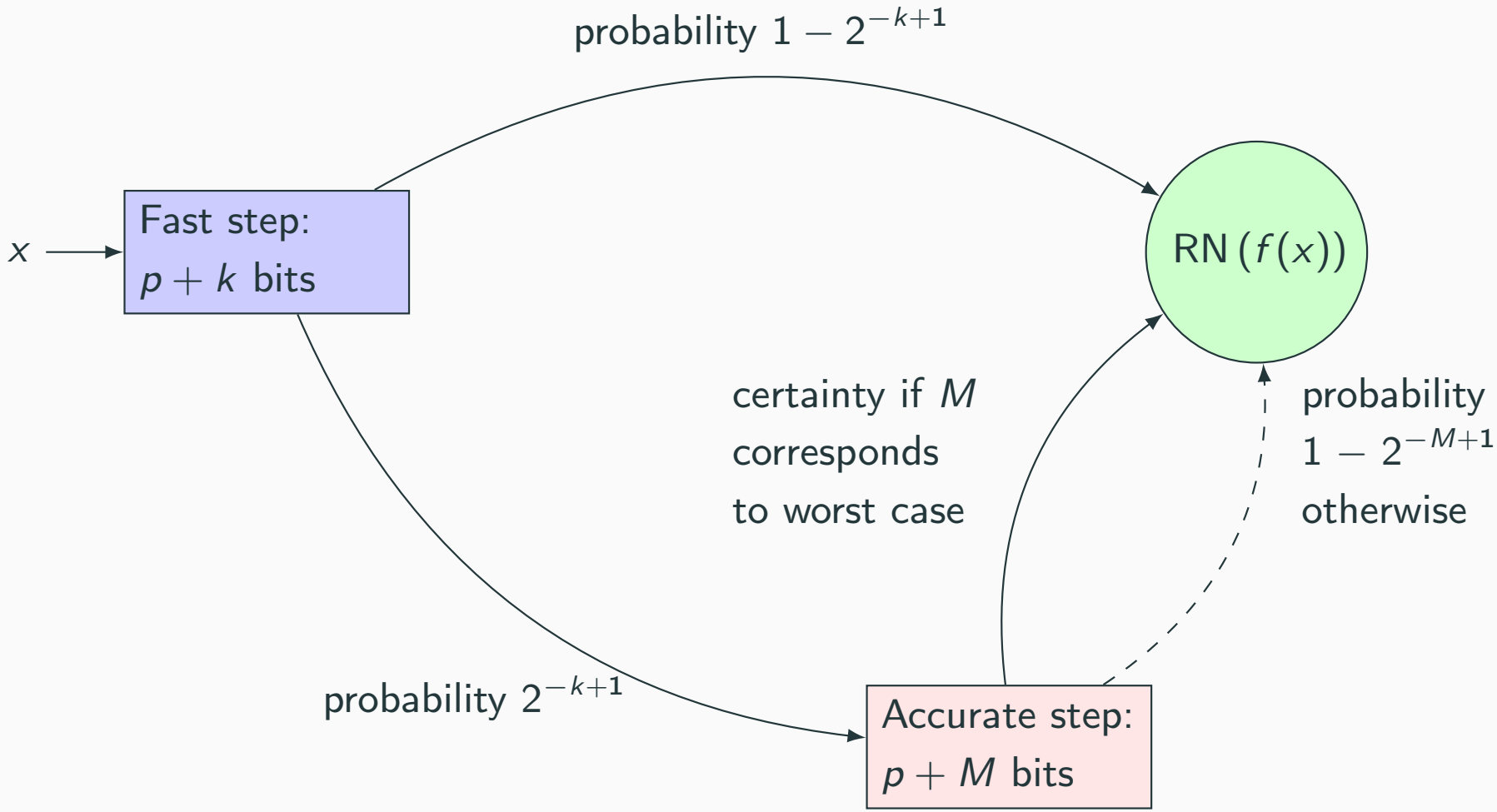
Expected vs actual worst cases

- Rule of thumb $\rightarrow$ if w is the word size, as there are $\approx 2^w$ FP numbers, with a $p+k$-bit approximation there is a total of 2^{w+1-k} failures $\rightarrow$ vanishes as soon as $k \approx w+1$.
- frequently less in practice: correlations (e.g. $\log_2$), function not defined in full range (exp because of overflow, arcsin, etc.);
- actual values (excluding tiny trivial input values):

format	$p+w+1$	exp	$\log_2$	arcsin
binary32	57	52	51	54 ($\lvert x\rvert > 2^{-23}$)
binary64	118	113	108	126 ($\lvert x\rvert > 2^{-25}$)

$\rightarrow$ the rule of thumb is not that bad.

2-step process (Ziv's strategy) – typically, $k \approx 10$



Algorithm that compute the hardest-to-round cases

- active domain since $\approx$ 2000;
- best algorithms based on lattice reduction;
- time of computation of hardest-to-round cases remains an exponential function of p;
- $p = 24$ (binary32) is easy, $p = 53$ (binary64) is feasible but very costly, larger formats out of reach.

Algorithm that compute the hardest-to-round cases

Table 1: *Worst cases for exponentials of binary64 FP numbers.*

Interval	worst case (binary)
$[-\infty, -2^{-30}]$	$\exp(-1.11101101001100011000111011111011011000100111111010101 \times 2^{-27})$ $= 1.11111111111111111111111100 \cdots 0111000100 \quad 1 \quad 1^{59}0001\ldots \times 2^{-1}$
$[-2^{-30}, 0)$	$\exp(-1.00000000000000000000000000000000000000000000000000001 \times 2^{-51})$ $= 1.111111111111111 \cdots 1111111111111100 \quad 0 \quad 0^{100}1010\ldots \times 2^{-1}$
$(0, +2^{-30}]$	$\exp(1.11111111111111111111111111111111111111111111111111111 \times 2^{-53})$ $= 1.00000000000000000000000000000000000000000000000000 \quad 1 \quad 1^{104}0101\ldots$
$[2^{-30}, +\infty]$	$\exp(1.01111111111111100111111111111101110000000000100100 \times 2^{-32})$ $= 1.00000000000000000000000000001011111111111111101000 \quad 0 \quad 0^{57}1101\ldots$ $\exp(1.10000000000000101111111111111101101111111111011100 \times 2^{-32})$ $= 1.00000000000000000000000000000110000000000000010111 \quad 1 \quad 1^{57}0010\ldots$ $\exp(1.10011110100111001011101111111101011000001000000001011 \times 2^{-31})$ $= 1.00000000000000000000000000001100111101001110010111 \quad 1 \quad 0^{57}1010\ldots$ $\exp(110.00001111010100101111001101111010111011001111110100)$ $= 110101100.01010000101101000000100111001000101011101110 \quad 0 \quad 0^{57}1000\ldots$

Results

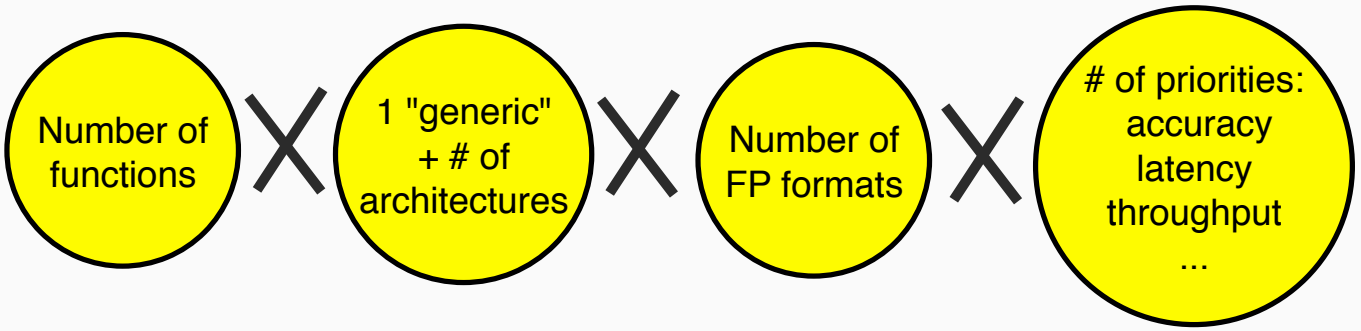
Table 2: *Worst cases for logarithms of binary64 FP numbers.*

Interval	worst case (binary)
$[2^{-\mathbf{1074}}, 1)$	$\log(1.1110101001110001110110000101110011101110000000100000 \times 2^{-\mathbf{509}})$ $= -101100000.00101001011010100110011010110100001011111111 \quad 1 \quad 1^{\mathbf{60}}0000\ldots$
	$\log(1.1001010001110110111000110000010011001101011111000111 \times 2^{-\mathbf{384}})$ $= -100001001.10110110000011001010111101000111101100110101 \quad 1 \quad 0^{\mathbf{60}}1010\ldots$
	$\log(1.0010011011101001110001001101001100100111100101100000 \times 2^{-\mathbf{232}})$ $= -10100000.10101011001011000010010111100110100001000010 \quad 0 \quad 0^{\mathbf{60}}1001\ldots$
	$\log(1.0110000100111001010101011101110010000000001011111000 \times 2^{-\mathbf{35}})$ $= -10111.11110000001011111001101110101111011000000110101 \quad 0 \quad 1^{\mathbf{60}}0011\ldots$
$(1, 2^{\mathbf{1024}}]$	$\log(1.0110001010101000100001100001001101100010100110110110 \times 2^{\mathbf{678}})$ $= 111010110.01000111100111101011101001111100100101110001 \quad 0 \quad 0^{\mathbf{64}}1110\ldots$

The CORE-Math library

- developed in Nancy by Paul Zimmermann and colleagues;
- all binary32 and binary64 functions from the C23 standard except compound and lgamma, with correct rounding;
- binary32 acos, acosh, asin, asinh, atan, atan2, atanh, cbrt, cosh, erf, erfc, expm1, exp2m1, exp10m1, tgamma, lgamma, log10, log1p, log2p1, log10p1, sinh, tan, tanh functions integrated into GNU libc
- code and an extensive bibliography are available from `https://core-math.gitlabpages.inria.fr/`

Libraries of math functions



thousands of function programs

- impossible to debug, maintain, keep consistent, improve...
- and physicists would like many other functions

First solution: computer-assisted library design

Metalibm project (`http://www.metalibm.org`), launched by Florent de Dinechin. Two versions

- fully automated for the end user;
- assistance for the specialist.

Metalibm builds upon tools such as Sollya and Gappa.

But this is not the ultimate goal

Generation of functions at compile-time

- take into account the exact context: underlying architecture, accuracy requirements, priorities (latency/throughput);
- possibly, information on input domain ($\rightarrow$ simplify/avoid range reduction), or special cases (e.g., infinities, NaNs known not to happen);
- compound functions: if you need

$$E_4(x) = \frac{x}{e^x - 1} - \ln(1 - e^{-x}),$$

  then you directly generate $E_4(x)$ instead of generating exp, ln and combining them.
- formal proof absolutely necessary (no library to heavily test beforehand);
- need collaboration of people from computer arithmetic, mathematics, computer algebra, compilation, formal proof...