Faster computation of elementary functions

Fredrik Johansson

MATHEXP–Polsys seminar, Inria Saclay

January 27, 2023

Paper: https://arxiv.org/abs/2207.02501

Introduction

Given $x \in \mathbb{R}$ and $B \geq 0$, we want to compute any of the elementary functions

- $\exp(x)$
- $\log(x)$
- $\sin(x)$, $\cos(x)$ (often simultaneously)
- $\operatorname{atan}(x)$

with error $\leq 2^{-B}$.

How can we make this fast (in practice) for "large" B?

In computational number theory, we typically care about B between 100 and 1,000,000.

Asymptotically fast algorithms (Brent, 1970s)

As usual, the problem is reduced to (fast) integer multiplication.[1] This can be achieved in quite different ways.

1. Taylor series + functional equations

$$O(M(B)\log^{2+\varepsilon}(B))$$

2. The arithmetic-geometric mean (AGM)

$$O(M(B)\log(B))$$

[1] Asymptotically $M(B) = O(B \log B)$ (Harvey – van der Hoeven). Up to a few thousand bits, it is more accurate to assume $M(B) = O(B^2)$ (classical) or $M(B) = O(B^{1.6})$ (Karatsuba).

Sketch of the Taylor series method

Consider $\exp(x)$. The other functions are analogous.

Step 1 (optional): argument reduction

$$\exp(x) = 2^m \exp(y), \quad y = x - m \log(2), \; |y| \leq \frac{\log(2)}{2}.$$

The constant $\log(2)$ only needs to be computed once. For trigonometric functions, π is used.

Step 2: second argument reduction

$$\exp(y) = \exp(t)^{2^r}, \quad t = y/2^r$$

ensuring $|t| \leq 2^{-r}$ for some tuning parameter $r \geq 0$.

Sketch of the Taylor series method

Step 3a (used up to $B \approx 10^4$)

$$\exp(t) = s + \sqrt{s^2+1}, \quad s = \sinh(t) \approx \sum_{n=0}^{N} \frac{t^{2n+1}}{(2n+1)!}$$

The sum is evaluated using $O(\sqrt{N})$ full-precision multiplications and $O(N)$ "scalar" operations.

Step 3b ("bit-burst algorithm", very high precision)
Write $\exp(t) = \exp(t_1) \cdot \exp(t_2) \cdots$ where t_j extracts 2^j bits in the binary expansion of t. Use binary splitting to evaluate

$$\exp(t_j) \approx \sum_{n=0}^{N_j} \frac{t^n}{n!}.$$

Sketch of the AGM method

The AGM iteration

$$\mathrm{agm}(x_0, y_0) = \lim_{n\to\infty} x_n, \quad x_{n+1} = (x_n + y_n)/2,\ y_{n+1} = \sqrt{x_n y_n}$$

converges to B-bit accuracy in $O(\log B)$ steps.

The AGM allows computing $\log(z)$ for $z \in \mathbb{C}$, and by extension any elementary function.[2]

MPFR implements real logarithms using

$$\log(x) \approx \frac{\pi}{2\,\mathrm{agm}(1, 4/s)} - m \log(2), \quad s = x \cdot 2^m > 2^{B/2}.$$

[2] E.g. using Newton iteration to obtain $\exp(z)$.

Taylor vs AGM

Surprising fact: in practice, Taylor series seem to beat the AGM for reasonable B (at least for $B \leq 10^9$).

What are the overheads in the AGM?

- One B-bit square root costs roughly 1-3 times a B-bit multiplication (the overhead depends on the precision), so each step of the AGM costs roughly 2-4 multiplications.
- Each iteration must be done with full precision.[3]
- There is more overhead (around $3\times$) for trigonometric functions, since we have to use complex arithmetic.

[3] We can save a bit of work in the last iterations, but this does not make a large difference.

Faster argument reduction

Efficient argument reduction is key to the performance of Taylor series methods. Note that evaluating

$$\exp(y) = \exp(t)^{2^r}, \quad t = y/2^r$$

costs r full B-bit squarings. In practice $r \approx 10$ to 100 is optimal.

Question: can we reduce the input to size 2^{-r} more quickly?

This is possible with precomputation. For example, we need just one multiplication if we have a table of $\exp(j/2^r)$, $0 \leq j < 2^r$, or m multiplications with an m-partite table of $m2^{r/m}$ entries.

This works extremely well in “medium precision” (up to about 1000 digits) (J. 2015), but eventually gives smaller returns / uses excessive memory.

Schönhage's argument reduction

Some years ago,[4] Arnold Schönhage presented a method to compute elementary functions **without large tables.**

The idea: use "diophantine combinations of incommensurable logarithms" for argument reduction.

$$\exp(x) = 2^c 3^d \exp(t), \quad t = x - c\log(2) - d\log(3)$$

- We can find $c, d \in \mathbb{Z}$ such that t is arbitrarily small.
- $2^c 3^d \in \mathbb{Q}$ is computed using binary powering.
- We only need to precompute $\log(2)$ and $\log(3)$, for any B.

[4] In talks given at Dagstuhl in 2006 and at RISC in 2011; there are published talk abstracts, but no paper with details.

Schönhage's method for trigonometric functions

For trigonometric functions, use pairs of Gaussian primes $a + bi$ instead of rational primes. The formula for one prime:

$$\cos(x) + i\sin(x) = \exp(ix) = \exp(i(x - c\alpha))\frac{(a+bi)^c}{(a-bi)^c}, \quad c \in \mathbb{Z}$$

where

$$\alpha = \frac{1}{i}\left[\log(a+bi) - \log(a-bi)\right] = 2\,\mathrm{atan}\left(\frac{b}{a}\right)$$

defines a rotation by $e^{i\alpha} = (a+bi)/(a-bi)$.

For example, we can use the pair $\mathrm{atan}(1)$ and $\mathrm{atan}(1/2)$, corresponding to the Gaussian primes $1 + i$ and $2 + i$,

Using many primes

Schönhage describes the method as useful for “medium precision”, with B in the range from around 50 to 3000 bits.

Problem: to achieve $|t| < 2^{-r}$, we will generally need coefficients (exponents) with $r/2$ bits.

Indeed, r should be at most $O(\log B)$ with this method. If r is too large, we will not save time over r-fold repeated squaring.

Idea for improvement: instead of using a pair of primes, use n primes for $n \geq 2$, giving coefficients around r/n bits.

Solving the inhomogeneous integer relation problem

Problem: given real numbers x and $\alpha_1, \ldots, \alpha_n$ and a tolerance 2^{-r}, find a small vector $(c_1, \ldots, c_n) \in \mathbb{Z}^n$ such that

$$x \approx c_1\alpha_1 + \ldots c_n\alpha_n$$

with error at most 2^{-r}.

When $P = \{p_1, \ldots p_n\}$ is a set of prime numbers and $\alpha_i = \log(p_i)$, a solution yields a P-smooth rational approximation

$$\exp(x) \approx p_1^{c_1} \cdots p_n^{c_n} \in \mathbb{Q}$$

with small numerator and denominator.

Solving the inhomogeneous integer relation problem

Idea: use LLL to solve

$$c_0 x + c_1 \alpha_1 + \ldots + c_n \alpha_n \approx 0.$$

Unfortunately, this will generally give a denominator $c_0 \neq \pm 1$.

Also, running LLL each time we want to evaluate an elementary function will be too slow!

Solving the inhomogeneous integer relation problem

Instead, use LLL to solve the homogeneous problem

$$c_1\alpha_1 + \ldots c_n\alpha_n \approx 0.$$

Do this with tolerance C^{-i}, for $i = 1, 2, \ldots$.[5] Each solution yields an approximate relation

$$\varepsilon_i = d_{i,1}\alpha_1 + \ldots d_{i,n}\alpha_n, \quad \varepsilon_i = O(C^{-i})$$

We store tables of the coefficients $d_{i,j}$ and floating-point approximations of the errors ε_i.

Given x, we now simply reduce with respect to $\varepsilon_1, \varepsilon_2, \varepsilon_3, \ldots$.

[5] Theoretically $C = e$ is optimal, but $C = 2$ or $C = 10$ work just as well.

Numerical example

We generate a relation table for the logarithms of the first $n = 13$ primes

$$P = \{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41\}$$

One line in Pari/GP can do the job:

```
? n=13; for(i=1, 32, localprec(i+10);
  P=vector(n,k,log(prime(k)));
  d=lindep(P,i)~; printf("%s %.5g\n", d, d * P~))
```

[0, 0, 0, 0, -1, 1, 0, 0, 0, 0, 0, 0, 0]	0.16705
[0, 0, 1, 0, -1, 0, -1, 0, 0, 0, 0, 1, 0]	-0.010753
[-1, 0, 0, 0, 0, -1, 1, -1, 0, 1, 0, 0, 0]	-0.0020263
[-1, 0, 0, 0, -1, 0, 1, -1, 1, -1, 1, 0, 0]	-8.2498 e-5
[1, 0, 1, -1, 0, 1, -1, 1, -1, 0, 0, -1, 1]	9.8746 e-6
[0, 1, 0, -1, -1, 0, 2, -1, 0, -1, -1, 1, 1]	1.5206 e-6
[1, -1, 0, 1, 1, 2, -1, 0, -2, 1, -1, -1, 1]	3.2315 e-8
[1, -1, 0, 1, 1, 2, -1, 0, -2, 1, -1, -1, 1]	3.2315 e-8
[1, 0, 4, -1, -2, 0, 0, 2, 0, -2, -2, 1, 1]	4.3825 e-9
[0, -2, 0, 0, -2, 0, 0, 2, -4, 4, -1, 1, 0]	-2.1170 e-10
[1, 1, 4, 1, -1, 1, -2, -3, 0, -4, 3, 1, 1]	-7.0743 e-11
[0, -2, -1, 0, 2, 4, 4, 0, 3, 1, -6, -1, -3]	3.3304 e-12
[3, 2, -1, -6, 2, 3, -2, -2, 3, 1, 5, -4, -2]	2.5427 e-13
[-4, -2, 4, -4, 3, 1, 7, 0, -3, -4, 4, -7, 3]	-9.9309 e-14
[1, -1, -7, -2, 5, 5, -6, 2, 0, -10, 5, 2, 3]	-9.5171 e-15
[3, -2, -7, -9, 6, 6, 3, 9, 1, 8, -15, -4, 0]	6.8069 e-16
[-1, 13, -5, -7, -3, -3, -13, 3, 0, -1, 6, -3, 12]	-7.1895 e-17
[-2, 3, -2, 2, -15, 16, 4, -7, 11, -15, 0, 9, -4]	8.1931 e-18
[2, 0, -9, -11, -5, -11, 21, 9, -9, -4, -1, -4, 13]	5.6466 e-19
[6, -9, 0, 9, 9, -2, -4, -22, 4, -7, 0, 5, 11]	4.6712 e-19
[1, -27, 22, -14, -2, 0, 0, -27, -3, -5, 18, 10, 9]	-1.0084 e-20
[1, 41, -2, 5, -42, 6, -2, 13, 5, 3, -5, 7, -9]	-1.3284 e-21
[4, -5, 8, -8, 6, -25, -38, -16, 24, 13, -10, 10, 24]	-8.5139 e-23
[4, -5, 8, -8, 6, -25, -38, -16, 24, 13, -10, 10, 24]	-8.5139 e-23
[-43, -2, 4, 9, 19, -26, 92, -30, -6, -24, 11, -4, -18]	-4.8807 e-24
[8, 38, -4, 34, -31, 60, -75, 31, 44, -32, -1, -43, 17]	2.7073 e-25
[48, -31, 21, -27, 34, -23, -29, 41, -50, -65, 33, 20, 40]	5.2061 e-26
[-41, 8, 67, -84, 7, -22, -58, -35, 17, 58, -18, 13, 40]	-7.9680 e-27
[20, 15, 50, -1, 48, 72, -67, -96, 75, 48, -38, -126, 68]	2.7161 e-28
[26, 20, -35, 16, -1, 75, -13, 2, -128, -100, 130, 46, -13]	-3.3314 e-29
[-26, -20, 35, -16, 1, -75, 13, -2, 128, 100, -130, -46, 13]	3.3314 e-29
[137, -26, 127, 45, -14, -73, -66, -166, 71, 76, 122, -154, 53]	-1.4227 e-31

Numerical example

We compute $\exp(\sqrt{2}-1)$ with precision $B = 33220$ (10^4 digits).

Reducing $x = \sqrt{2}-1$ by the table on the previous slide yields the 37-smooth approximation $\exp(\sqrt{2}-1) = (u/v)\exp(t)$ where

$$\frac{u}{v} = \frac{13^{651} \cdot 19^{463} \cdot 37^{634}}{2^{274} \cdot 3^{414} \cdot 5^{187} \cdot 7^{314} \cdot 11^{211} \cdot 17^{392} \cdot 23^{36} \cdot 29^{369} \cdot 31^{231}}$$

and $t \approx -1.57 \cdot 10^{-32}$.

Now 148 terms of the Taylor series for $\sinh(t)$ yield full accuracy. Evaluating this series costs $2\sqrt{148} \approx 24$ full B-bit multiplications. (The bit-burst algorithm is about as fast here.)

Empirically, the entire evaluation costs roughly 25 full multiplications. For comparison, the AGM requires 25 iterations.

Speedup for elementary functions

Arb 2.23 using the new method with $n = 13$ primes, vs Arb 2.22

	$\exp(x)$		$\log(x)$		$\cos(x), \sin(x)$		$\operatorname{atan}(x)$	
Digits	First	Repeat	First	Repeat	First	Repeat	First	Repeat
1000	0.16×	1.43×	0.77×	1.43×	0.18×	1.23×	1.00×	1.00×
2000	0.22×	2.06×	0.73×	1.81×	0.40×	1.25×	0.75×	2.21×
4000	0.33×	2.37×	0.93×	1.86×	0.43×	1.62×	0.74×	2.45×
10,000	0.48×	2.03×	1.05×	1.70×	0.53×	1.89×	0.70×	2.23×
100,000	0.51×	1.52×	1.25×	1.68×	0.68×	1.61×	0.68×	1.53×
1,000,000	0.51×	1.26×	1.23×	1.39×	0.59×	1.29×	0.67×	1.25×

exp, sin/cos: using Taylor series
log: previously using AGM, now using exp + Newton
atan: previously using Taylor series, now using sin/cos + Newton

Varying the number of primes n

B	n	Memory (logs)	Time (logs)	r	Time to evaluate $\exp(x)$
10^4	0				0.000202
	2	2.4 KiB	0.000238	11	0.000183
	4	4.9 KiB	0.000240	27	0.000137
	8	9.8 KiB	0.000335	52	0.000106
	16	19.5 KiB	0.000579	83	8.48e-05
	32	39.1 KiB	0.00123	86	8.75e-05
	64	78.1 KiB	0.00270	72	9.71e-05
10^5	0				0.00895
	2	24.4 KiB	0.00679	18	0.00747
	4	48.8 KiB	0.0068	44	0.00638
	8	97.7 KiB	0.00977	71	0.00565
	16	195.3 KiB	0.0164	106	0.00534
	32	390.6 KiB	0.0337	161	0.00445
	64	781.2 KiB	0.0755	240	0.00383
10^7	0				4.36
	2	2.4 MiB	3.02	18	3.89
	4	4.8 MiB	3.01	47	3.53
	8	9.5 MiB	4.14	110	3.18
	16	19.1 MiB	6.57	222	2.90
	32	38.1 MiB	13.8	338	2.61
	64	76.3 MiB	31.3	551	2.39

Precomputation of logs and arctangents

How can we efficiently compute $\log(2), \log(3), \ldots, \log(p_n)$ simultaneously to B-bit precision?

Similarly for $\mathtt{atan}(1), \mathtt{atan}(1/2), \ldots, \mathtt{atan}(b_n/a_n)$?

Using Machin-like formulas

Examples:

$$\operatorname{atan}(1) = \frac{\pi}{4} = 4\operatorname{atan}\left(\frac{1}{5}\right) - \operatorname{atan}\left(\frac{1}{239}\right)$$

$$\log(2) = 4\operatorname{atanh}\left(\frac{1}{7}\right) + 2\operatorname{atanh}\left(\frac{1}{17}\right)$$

Used together with binary splitting evaluation of the series:

$$\operatorname{atan}\left(\frac{1}{x}\right) = \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k+1)} \frac{1}{x^{2k+1}}, \quad \operatorname{atanh}\left(\frac{1}{x}\right) = \sum_{k=0}^{\infty} \frac{1}{(2k+1)} \frac{1}{x^{2k+1}}.$$

We want an argument basis $X \subset \mathbb{Z}$ with small *Lehmer measure*

$$\mu(X) = \sum_{x \in X} \frac{1}{\log_{10}(|x|)}.$$

Simultaneous Machin-like formulas

Given $P = \{p_1, \ldots, p_n\}$, find $X = \{x_1, \ldots, x_n\}$ such that

$$\begin{pmatrix} \log(p_1) \\ \vdots \\ \log(p_n) \end{pmatrix} = M \begin{pmatrix} 2\,\mathrm{atanh}(1/x_1) \\ \vdots \\ 2\,\mathrm{atanh}(1/x_n) \end{pmatrix}, \quad M \in \mathbb{Q}_{n \times n}$$

has a solution. Similarly, for $Q = \{a_1 + b_1 i, \ldots, a_n + b_n i\}$,

$$\begin{pmatrix} \mathrm{atan}(b_1/a_1) \\ \vdots \\ \mathrm{atan}(b_n/a_n) \end{pmatrix} = M \begin{pmatrix} \mathrm{atan}(1/x_1) \\ \vdots \\ \mathrm{atan}(1/x_n) \end{pmatrix}, \quad M \in \mathbb{Q}_{n \times n}.$$

Example: a solution for $P = \{2, 3\}$ is $X = \{7, 17\}, M = (2, 1; 3, 2)$:

$$\log(2) = 4\,\mathrm{atanh}(1/7) + 2\,\mathrm{atanh}(1/17)$$

$$\log(3) = 6\,\mathrm{atanh}(1/7) + 4\,\mathrm{atanh}(1/17)$$

Finding Machin-like formulas using Gauss's method

For a finite set of primes $p \in P$:[6]

$$X \subseteq Y, \quad Y = \{x : x^2 - 1 \text{ is } P\text{-smooth}\}$$

For a finite set of Gaussian primes with $a^2 + b^2 \in Q$:

$$X \subseteq Z, \quad Z = \{x : x^2 + 1 \text{ is } Q\text{-smooth}\}$$

Having Y or Z, we can find solutions X (and then M) using linear algebra.

Fact: the sets Y and Z are finite for each fixed set P or Q. Tabulations by Luca and Najman (2010, 2013):

- For the 25 primes $p < 100$, $\#Y = 16223$.
- For the 22 Gaussian primes with $a^2 + b^2 < 100$, $\#Z = 811$.

[6] Since $2\,\mathtt{atanh}(1/x) = \log((x+1)/(x-1))$, we try to write each $p \in P$ as a power-product of P-smooth rational numbers of the form $(x+1)/(x-1)$.

Optimal(?) n-term formulas for the first n primes

n	P	X	$\mu(X)$
1	2	3	2.09590
2	2, 3	7, 17	1.99601
3	2, 3, 5	31, 49, 161	1.71531
4	$2 \ldots 7$	251, 449, 4801, 8749	1.31908
5	$2 \ldots 11$	351, 1079, 4801, 8749, 19601	1.48088
6	$2 \ldots 13$	1574, 4801, 8749, 13311, 21295, 246401	1.49710
7	$2 \ldots 17$	8749, 21295, 24751, 28799, 74359, 388961, 672281	1.49235
8	$2 \ldots 19$	57799, 74359, 87361, 388961, 672281, 1419263, 11819521, 23718421	1.40768
$\vdots$			
13	$2 \ldots 41$	51744295, 170918749, 265326335, 287080366, 362074049, 587270881, 831409151, 2470954914, 3222617399, 6926399999, 9447152318, 90211378321, 127855050751	1.42585
$\vdots$			
25	$2 \ldots 97$	373632043520429, 386624124661501, 473599589105798, 478877529936961, 523367485875499, 543267330048757, 666173153712219, 1433006524150291, 1447605165402271, 1744315135589377, 1796745215731101, 1814660314218751, 2236100361188849, 2767427997467797, 2838712971108351, 3729784979457601, 4573663454608289, 9747977591754401, 11305332448031249, 17431549081705001, 21866103101518721, 34903240221563713, 99913980938200001, 332110803172167361, 19182937474703818751	1.60385

Optimal(?) n-term formulas for the first n Gaussian primes

n	Q	X	$\mu(X)$
1	2	1	∞
2	2, 5	3, 7	3.27920
3	2, 5, 13	18, 57, 239	1.78661
4	$2 \ldots 17$	38, 57, 239, 268	2.03480
5	$2 \ldots 29$	38, 157, 239, 268, 307	2.32275
6	$2 \ldots 37$	239, 268, 307, 327, 882, 18543	2.20584
7	$2 \ldots 41$	268, 378, 829, 882, 993, 2943, 18543	2.33820
8	$2 \ldots 53$	931, 1772, 2943, 6118, 34208, 44179, 85353, 485298	2.01152
$\vdots$			
13	$2 \ldots 101$	683982, 1984933, 2343692, 2809305, 3014557, 6225244, 6367252, 18975991, 22709274, 24208144, 193788912, 201229582, 2189376182	1.84765
$\vdots$			
22	$2 \ldots 197$	1479406293, 1892369318, 2112819717, 2189376182, 2701984943, 2971354082, 3558066693, 4038832337, 5271470807, 6829998457, 7959681215, 8193535810, 12139595709, 12185104420, 12957904393, 14033378718, 18710140581, 18986886768, 20746901917, 104279454193, 120563046313, 69971515635443	2.19850

Things to do

- Detailed complexity analysis.

  What is the theoretically optimal number of primes n as a function of the precision B? Is there a theoretical asymptotic (constant-factor?) speedup?

- Fine-tuning of various parameters.

- For $z \in \mathbb{C}$, it is better to reduce with respect to lattices instead of separating real and imaginary parts?

- A p-adic version (we can use LLL to precompute relations $\sum_{i=1}^{n} c_i \log(p_i) = O(p^i)$ for reduction).

- Tabulate more Machin-like formulas.