Arb: efficient arbitrary-precision midpoint-radius interval arithmetic

Fredrik Johansson

LFANT, Inria Bordeaux & Institut de Mathématiques de Bordeaux

24th IEEE Symposium on Computer Arithmetic (ARITH 24)

Imperial College London, UK, 2017-07-24

Reliable arbitrary-precision arithmetic

Floating-point numbers (MPFR, MPC)

- $\pi \approx 3.14159265358979323850$
- Need error analysis – hard for nontrivial operations

Inf-sup intervals (MPFI, uses MPFR)

- $\pi \in [3.14159265358979323844, 3.14159265358979323850]$
- Twice as expensive

Mid-rad intervals / balls (iRRAM, Mathemagix, Arb)

- $\pi \in [3.14159265358979323850 \pm 4.15 \cdot 10^{-20}]$
- Better for precise intervals

Overview of Arb (`http://arblib.org`)

C library, licensed LGPL, depends on GMP, MPFR, FLINT
Portable, thread-safe, extensively tested and documented

Version 0.6 (presented at ISSAC 2013): 35 000 lines of code
Version 2.11 (July 2017): 2500 functions, 140 000 lines of code

Key features

- Efficient, flexible [mid ± rad] number format
- Complex numbers $[a \pm r] + [b \pm s]i$
- Polynomials, power series, matrices, special functions
- Use of asymptotically fast algorithms

Example: the integer partition function

Isolated values of $p(n) = 1, 1, 2, 3, 5, 7, 11, 15, 22, 30, 42\ldots$ can be computed by an infinite series:

$$p(n) = \frac{2\pi}{(24n-1)^{3/4}} \sum_{k=1}^{\infty} \frac{A_k(n)}{k} I_{3/2}\left(\frac{\pi}{k}\sqrt{\frac{2}{3}\left(n - \frac{1}{24}\right)}\right)$$

Example: the integer partition function

Isolated values of $p(n) = 1, 1, 2, 3, 5, 7, 11, 15, 22, 30, 42...$ can be computed by an infinite series:

$$p(n) = \frac{2\pi}{(24n-1)^{3/4}} \sum_{k=1}^{\infty} \frac{A_k(n)}{k} I_{3/2}\left(\frac{\pi}{k}\sqrt{\frac{2}{3}\left(n - \frac{1}{24}\right)}\right)$$

Old versions of Maple got $p(11269)$, $p(11566)$, $\ldots$ wrong!

Example: the integer partition function

Isolated values of $p(n) = 1, 1, 2, 3, 5, 7, 11, 15, 22, 30, 42...$ can be computed by an infinite series:

$$p(n) = \frac{2\pi}{(24n-1)^{3/4}} \sum_{k=1}^{\infty} \frac{A_k(n)}{k} I_{3/2}\left(\frac{\pi}{k}\sqrt{\frac{2}{3}\left(n - \frac{1}{24}\right)}\right)$$

Old versions of Maple got $p(11269)$, $p(11566)$, ... wrong!

Using ball arithmetic: $p(100) \in [190569292.00 \pm 0.39]$

Example: the integer partition function

Isolated values of $p(n) = 1, 1, 2, 3, 5, 7, 11, 15, 22, 30, 42...$ can be computed by an infinite series:

$$p(n) = \frac{2\pi}{(24n-1)^{3/4}} \sum_{k=1}^{\infty} \frac{A_k(n)}{k} \, I_{3/2}\left(\frac{\pi}{k}\sqrt{\frac{2}{3}\left(n - \frac{1}{24}\right)}\right)$$

Old versions of Maple got $p(11269)$, $p(11566)$, $\ldots$ wrong!

Using ball arithmetic: $p(100) \in [190569292.00 \pm 0.39]$

FJ (2012): algorithm for $p(n)$ with softly optimal complexity – requires tight control of the internal precision

	Digits	Mathematica	MPFR	Arb
$p(10^{10})$	111 391	60 s	0.4 s	0.3 s
$p(10^{15})$	35 228 031		828 s	553 s
$p(10^{20})$	11 140 086 260			100 hours

Example: accurate “black box” evaluation

Compute $\sin(\pi + e^{-10000})$ to a relative accuracy of 53 bits

```
#include "arb.h"
int main()
{
 arb_t x, y; long prec;
 arb_init(x); arb_init(y);

 for (prec = 64; ; prec *= 2)
 {
  arb_const_pi(x, prec);
  arb_set_si(y, -10000);
  arb_exp(y, y, prec);
  arb_add(x, x, y, prec);
  arb_sin(y, x, prec);

  arb_printn(y, 15, 0); printf("\n");
  if (arb_rel_accuracy_bits(y) >= 53)
    break;
 }
 arb_clear(x); arb_clear(y);
}
```

Output:

```
[+/- 6.01e-19]
[+/- 2.55e-38]
[+/- 8.01e-77]
[+/- 8.64e-154]
[+/- 5.37e-308]
[+/- 3.63e-616]
[+/- 1.07e-1232]
[+/- 9.27e-2466]
[-1.13548386531474e-4343 +/- 3.91e-4358]
```

Remark: `arb_printn` guarantees a correct decimal approximation (within 1 ulp) *and* a correct decimal enclosure

Precision and error bounds

- For simple operations, *prec* describes the floating-point precision for midpoint operations:

  $[a \pm r] \cdot [b \pm s] \;\rightarrow\; [\mathrm{round}(ab) \pm (|a|s + |b|r + rs + \varepsilon_{\mathrm{round}})]$

- Arb functions may try to achieve *prec* accurate bits, but will avoid doing more than $O(\mathrm{poly}(\mathit{prec}))$ work:

  $\sin(\mathit{HUGE}) \rightarrow [\pm 1]$ when more than $O(\mathit{prec})$ bits needed for mod π reduction

Content of the `arb_t` type

Exponent	
Limb count + sign bit	
Limb 0	Allocation count
Limb 1	Pointer to ≥ 3 limbs

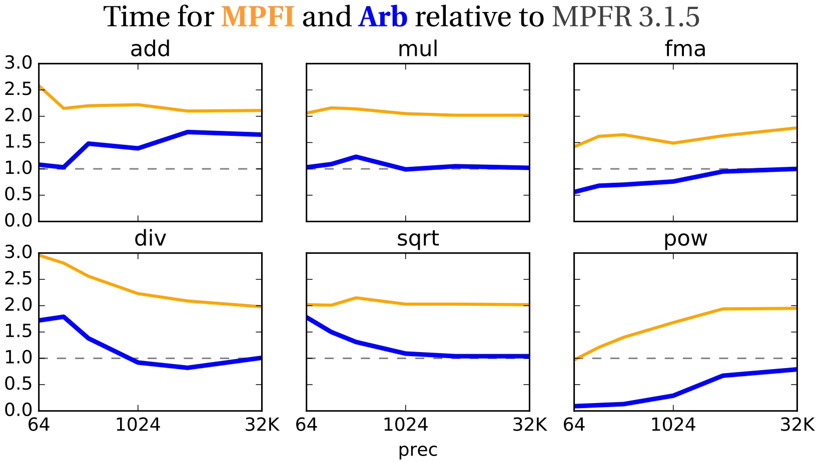
Exponent
Limb

Midpoint (`arf_t`, 4 words)
$(-1)^s \cdot m \cdot 2^e$, arbitrary-precision $\frac{1}{2} \leq m < 1$ (or $0, \pm\infty$, NaN)
The mantissa m is an array of limbs, bit aligned like MPFR
Up to two limbs (128 bits), m is stored inline

Radius (`mag_t`, 2 words)
$m \cdot 2^e$, fixed 30-bit precision $\frac{1}{2} \leq m < 1$ (or $0, +\infty$)

All exponents are unbounded (but stored inline up to 62 bits)

Performance for basic real operations



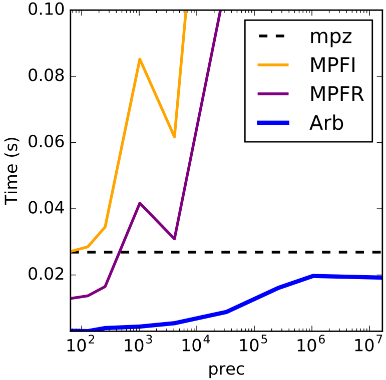
- Fast algorithm for pow (exp+log): see FJ, ARITH 2015
- MPFI does not have fma and pow (using mul+add and exp+log)
- MPFR 4 will be faster up to 128 bits; some speedup possible in Arb

Optimizing for numbers with short bit length

Trailing zero limbs are not stored: 0.1010 0000 → 0.1010
Heap space for used limbs is allocated dynamically

Example: $10^5!$ by binary splitting

```
fac(arb_t res, int a, int b, int prec)
{
  if (b - a == 1)
    arb_set_si(res, b);
  else {
    arb_t tmp1, tmp2;
    arb_init(tmp1); arb_init(tmp2);
    fac(tmp1, a, a+(b-a)/2, prec);
    fac(tmp2, a+(b-a)/2, b, prec);
    arb_mul(res, tmp1, tmp2, prec);
    arb_clear(tmp1); arb_clear(tmp2);
  }
}
```



Polynomials in Arb

Functionality for $\mathbb{R}[X]$ and $\mathbb{C}[X]$

- Basic arithmetic, evaluation, composition
- Multipoint evaluation, interpolation
- Power series arithmetic, composition, reversion
- Power series transcendental functions
- Complex root isolation (not asymptotically fast)

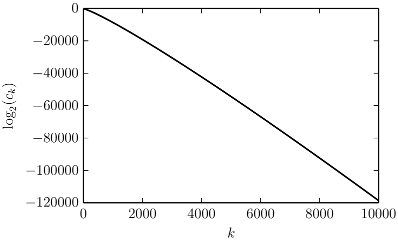
For high degree n, use polynomial multiplication as kernel

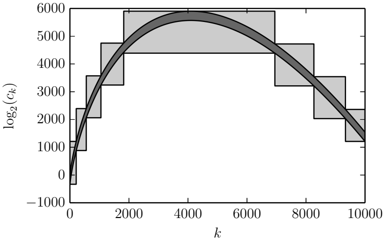
- FFT reduces complexity from $O(n^2)$ to $O(n \log n)$, but gives poor enclosures when numbers vary in magnitude
- Arb guarantees as good enclosures as $O(n^2)$ schoolbook multiplication, but with FFT performance when possible

Fast, numerically stable polynomial multiplication

Simplified version of algorithm by J. van der Hoeven (2008).

Transformation used to square $\sum_{k=0}^{10\,000} X^k/k!$ *at* 333 *bits precision*





- $(A+a)(B+b)$ via three multiplications AB, $|A|b$, $a(|B|+b)$
- The magnitude variation is reduced by scaling $X \to 2^e X$
- Coefficients are grouped into blocks of bounded height
- Blocks are multiplied exactly via FLINT's FFT over $\mathbb{Z}[X]$
- For blocks up to length 1000 in $|A|b$, $a(|B|+b)$, use `double`

Example: series expansion of Riemann zeta

Let $\xi(s) = (s-1)\pi^{-s/2}\Gamma\left(1+\frac{1}{2}s\right)\zeta(s)$, and define λ_n by

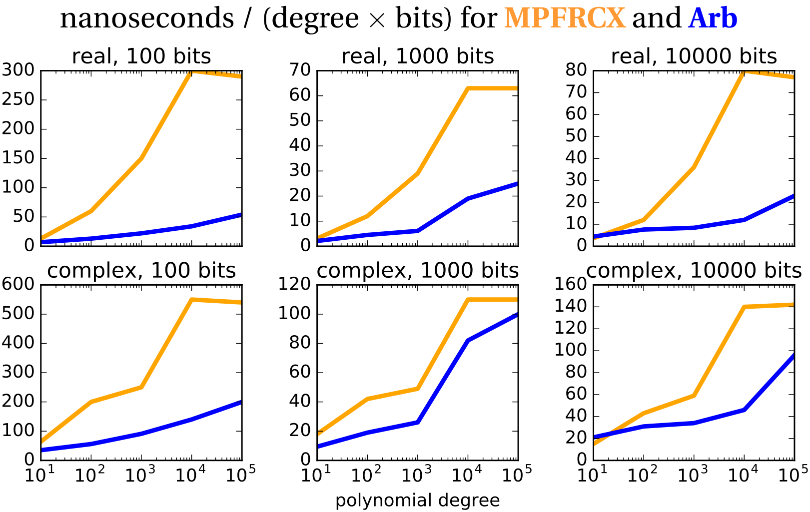
$$\log\left(\xi\left(\frac{X}{X-1}\right)\right) = \sum_{n=0}^{\infty} \lambda_n X^n.$$

The Riemann hypothesis is equivalent to $\lambda_n > 0$ for all $n > 0$.

Prove $\lambda_n > 0$ for all $0 < n \leq N$:

Multiplication algorithm	$N = 1000$	$N = 10000$
Slow, stable (schoolbook)	1.1 s	1813 s
Fast, stable	0.2 s	214 s
Fast, unstable (FFT used naively)	17.6 s	72000 s

Polynomial multiplication: uniform magnitude



MPFRCX uses floating-point Toom-Cook and FFT over MPFR and MPC coefficients, without error control

Example: constructing $f(X) \in \mathbb{Z}[X]$ from its roots

$$(X - \sqrt{3}i)(X + \sqrt{3}i) \quad \rightarrow \quad X^2 + [3.00 \pm 0.004] \quad \rightarrow \quad X^2 + 3$$

Two paradigms: **modular/p-adic** and **complex analytic**

Example: constructing $f(X) \in \mathbb{Z}[X]$ from its roots

$$(X - \sqrt{3}i)(X + \sqrt{3}i) \quad \rightarrow \quad X^2 + [3.00 \pm 0.004] \quad \rightarrow \quad X^2 + 3$$

Two paradigms: **modular/p-adic** and **complex analytic**

Constructing finite fields $GF(p^n)$ – need some $f(X)$ of degree n that is irreducible mod p – take roots to be certain sums of roots of unity

p	Degree (n)	Bits	Pari/GP	Arb
$2^{607} - 1$	729	502	0.03 s	0.02 s
$2^{607} - 1$	6561	7655	4.5 s	3.6 s
$2^{607} - 1$	59049	68937	944 s	566 s

Example: constructing $f(X) \in \mathbb{Z}[X]$ from its roots

$$(X - \sqrt{3}i)(X + \sqrt{3}i) \quad \to \quad X^2 + [3.00 \pm 0.004] \quad \to \quad X^2 + 3$$

Two paradigms: **modular/p-adic** and **complex analytic**

Constructing finite fields $GF(p^n)$ – need some $f(X)$ of degree n that is irreducible mod p – take roots to be certain sums of roots of unity

p	Degree (n)	Bits	Pari/GP	Arb
$2^{607} - 1$	729	502	0.03 s	0.02 s
$2^{607} - 1$	6561	7655	4.5 s	3.6 s
$2^{607} - 1$	59049	68937	944 s	566 s

Hilbert class polynomials $H_D(X)$ (used to construct elliptic curves with prescribed properties) – roots are values of the function $j(\tau)$

$-D$	Degree	Bits	Pari/GP	classpoly	CM	Arb
$10^6 + 3$	105	8527	12 s	0.8 s	0.4 s	0.2 s
$10^7 + 3$	706	50889	194 s	8 s	29 s	20 s
$10^8 + 3$	1702	153095	1855 s	82 s	436 s	287 s

Special functions in Arb

The full complex domain for all parameters is supported

Elementary: $\exp(z)$, $\log(z)$, $\sin(z)$, $\operatorname{atan}(z)$, $\operatorname{expm1}(z)$, Lambert $W_k(z)\ldots$
Gamma, beta: $\Gamma(z)$, $\log\Gamma(z)$, $\psi^{(s)}(z)$, $\Gamma(s,z)$, $\gamma(s,z)$, $B(z;a,b)$
Exponential integrals: $\operatorname{erf}(z)$, $\operatorname{erfc}(z)$, $\mathrm{E}_s(z)$, $\mathrm{Ei}(z)$, $\mathrm{Si}(z)$, $\mathrm{Ci}(z)$, $\mathrm{Li}(z)$
Bessel and Airy: $J_\nu(z)$, $Y_\nu(z)$, $I_\nu(z)$, $K_\nu(z)$, $\mathrm{Ai}(z)$, $\mathrm{Bi}(z)$
Orthogonal: $P_\nu^\mu(z)$, $Q_\nu^\mu(z)$, $T_\nu(z)$, $U_\nu(z)$, $L_\nu^\mu(z)$, $C_\nu^\mu(z)$, $H_\nu(z)$, $P_\nu^{(a,b)}(z)$
Hypergeometric: ${}_0F_1(a,z)$, ${}_1F_1(a,b,z)$, $U(a,b,z)$, ${}_2F_1(a,b,c,z)$
Zeta, polylogarithms and L-functions: $\zeta(s)$, $\zeta(s,z)$, $\mathrm{Li}_s(z)$, $L(\chi,s)$
Theta, elliptic and modular: $\theta_i(z,\tau)$, $\eta(\tau)$, $j(\tau)$, $\Delta(\tau)$, $G_{2k}(\tau)$, $\wp(z,\tau)$
Elliptic integrals: $\operatorname{agm}(x,y)$, $K(m)$, $E(m)$, $F(\phi,m)$, $E(\phi,m)$,
$\Pi(n,\phi,m)$, $R_F(x,y,z)$, $R_G(x,y,z)$, $R_J(x,y,z,p)$, $\wp^{-1}(z,\tau)$

Example: algorithm for $K_\nu(z)$

Large $|z|$: $K_\nu(z) = \sqrt{\frac{\pi}{2z}}\, e^{-z}\, {}_2F_0\left(\nu + \frac{1}{2}, \frac{1}{2} - \nu, -\frac{1}{2z}\right)$

Small $|z|$, $\nu \notin \mathbb{Z}$:

$$2K_\nu(2z) = z^\nu\, \Gamma(-\nu)\, {}_0F_1\left(1+\nu, z^2\right) + z^{-\nu}\, \Gamma(\nu)\, {}_0F_1\left(1-\nu, z^2\right)$$

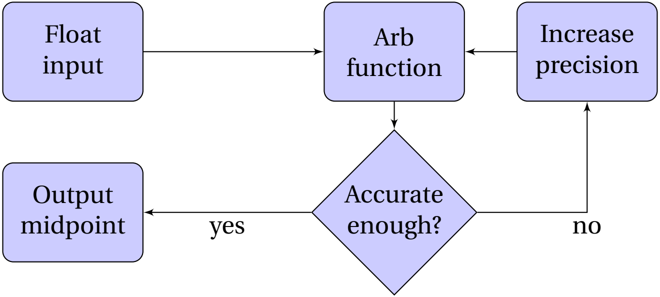
Small $|z|$, $\nu \in \mathbb{Z}$: $K_\nu(z) = \lim_{X \to 0} K_{\nu+X}(z)$ via $\mathbb{C}[[X]]/\langle X^2 \rangle$

The core building block is the hypergeometric series:

$${}_pF_q(a_1,\ldots,a_p;b_1,\ldots,b_q;z) = \underbrace{\sum_{k=0}^{N-1} \frac{(a_1)_k \ldots (a_p)_k}{(b_1)_k \ldots (b_q)_k} \frac{z^k}{k!}}_{\text{Compute using ball arithmetic}} + \underbrace{\varepsilon_N}_{\text{Bound}}$$

Summation uses fast techniques at high precision (binary splitting, rectangular splitting, polynomial multipoint evaluation)

Floating-point mathematical functions



- Can target any precision (53, 113, ...)
- Can ensure *correct rounding* if exact points are known
- Testing found wrong results computed by MPFR 3.1.3 (square roots, Bessel functions, Riemann zeta function)

Example code: C99 `double complex` math functions

`https://github.com/fredrik-johansson/arbcmath/`

Hypergeometric functions, 53-bit accuracy

	Code	Average	Median	Accuracy
${}_1F_1$	SciPy	2.7	0.76	18 good, 4 fair, 4 poor, 5 wrong, 2 NaN, 7 skipped
${}_2F_1$	SciPy	24	0.56	18 good, 1 fair, 1 poor, 3 wrong, 1 NaN, 6 skipped
${}_2F_1$	Michel & S.	7.7	2.1	22 good, 1 poor, 6 wrong, 1 NaN
${}_1F_1$	MMA (m)	1100	29	34 good, 2 poor, 4 wrong, 2 no significant digits out
${}_2F_1$	MMA (m)	30000	72	29 good, 1 fair
U	MMA (m)	4400	190	28 good, 4 fair, 2 wrong, 6 no significant digits out
Q	MMA (m)	4300	61	21 good, 3 fair, 2 poor, 1 wrong, 3 NaN
${}_1F_1$	MMA (a)	2100	170	39 good, 1 not good as claimed (actual error 2^{-40})
${}_2F_1$	MMA (a)	37000	540	30 good (2^{-53})
U	MMA (a)	25000	340	38 good, 2 not as claimed (2^{-40}, 2^{-45})
Q	MMA (a)	8300	780	28 good, 1 not as claimed (2^{-25}), 1 wrong
${}_1F_1$	Arb	200	32	40 good (correct rounding)
${}_2F_1$	Arb	930	160	30 good (correct rounding)
U	Arb	2000	93	40 good (correct rounding)
Q	Arb	3000	210	30 good (2^{-53})

40 test cases for ${}_1F_1/U$ and 30 for ${}_2F_1/Q$ from Pearson (2009)

Average and median time in microseconds

MMA = Mathematica, (m) machine, (a) arbitrary precision

Conclusion

Ball arithmetic **works in practice** for many applications where arbitrary-precision arithmetic is normally used

What needs further work?

- Tighter enclosures for many operations
- Make algorithms adaptive to the output error
- Reduce overhead at low precision
  - General optimizations, SIMD, `double` representations
  - Fusing operations e.g. J. van der Hoeven and G. Lecerf, "Evaluating straight-line programs over balls" (ARITH 2016)

Some more software using Arb

- SageMath - `RealBallField` and `ComplexBallField`
  `http://sagemath.org`
- Nemo.jl and Hecke.jl - computer algebra and algebraic number theory in Julia
  `http://nemocas.org`
- Marc Mezzarobba: rigorous evaluation of D-finite functions in SageMath
  `http://marc.mezzarobba.net/code/ore_algebra-analytic/`
- Pascal Molin, Christian Neurohr: rigorous computation of period matrices of superelliptic curves
  `https://github.com/pascalmolin/hcperiods`

Thank you!