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)
- $\pi \approx 3.1415926535897932385$
- Need error analysis – hard for nontrivial operations

Inf-sup intervals (MPFI, uses MPFR)
- $\pi \in [3.1415926535897932384, 3.1415926535897932385]$
- Twice as expensive

Mid-rad intervals / balls (iRRAM, Mathemagix, Arb)
- $\pi \in [3.1415926535897932385 \pm 4.15 \cdot 10^{-20}]$
- 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 ± r] + [b ± s]i
▶ Polynomials, power series, matrices, special functions
▶ Use of asymptotically fast algorithms
Example: the integer partition function

Isolated values of $p(n) = 1, 1, 2, 3, 5, 7, 11, 15, 22, 30, 42...$ 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 $p(11269), p(11566), \ldots$ wrong!

Using ball arithmetic: $p(100) \in [190569292.00 \pm 0.39]$
Example: the integer partition function

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

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

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

<table>
<thead>
<tr>
<th>( p(n) )</th>
<th>Digits</th>
<th>Mathematica</th>
<th>MPFR</th>
<th>Arb</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p(10^{10}) )</td>
<td>111 391</td>
<td>60 s</td>
<td>0.4 s</td>
<td>0.3 s</td>
</tr>
<tr>
<td>( p(10^{15}) )</td>
<td>35 228 031</td>
<td>828 s</td>
<td></td>
<td>553 s</td>
</tr>
<tr>
<td>( p(10^{20}) )</td>
<td>11 140 086 260</td>
<td>100 hours</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example: accurate “black box” evaluation

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

```c
#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:

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

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

$$\sin(HUGE) \rightarrow [\pm 1]$$
when more than $O(prec)$ bits needed for mod $\pi$ reduction
Content of the `arb_t` type

<table>
<thead>
<tr>
<th></th>
<th>Exponent</th>
<th>Limb 0</th>
<th>Allocation count</th>
<th>Limb 1</th>
<th>Pointer to ≥3 limbs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exponent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Midpoint (arf_t, 4 words)**

\((-1)^s \cdot m \cdot 2^e\), arbitrary-precision \(\frac{1}{2} \leq m < 1\) (or 0, ±\(\infty\), 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

Time for **MPFI** and **Arb** relative to MPFR 3.1.5

- 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 \rightarrow 0.1010
Heap space for used limbs is allocated dynamically

Example: $10^5!$ by binary splitting

```c
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);
    }
}
```

![Graph showing time vs. precision for different libraries](image)
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 \( \sum_{k=0}^{10000} X^k / k! \) at 333 bits precision

- \((A+a)(B+b)\) via three multiplications \(AB\), \(|A|b\), \(a(|B|+b)\)
- The magnitude variation is reduced by scaling \(X \rightarrow 2^e X\)
- 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 \( \lambda_n \) 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 \( 0 < n \leq N \):

<table>
<thead>
<tr>
<th>Multiplication algorithm</th>
<th>( N = 1000 )</th>
<th>( N = 10000 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slow, stable (schoolbook)</td>
<td>1.1 s</td>
<td>1813 s</td>
</tr>
<tr>
<td>Fast, stable</td>
<td>0.2 s</td>
<td>214 s</td>
</tr>
<tr>
<td>Fast, unstable (FFT used naively)</td>
<td>17.6 s</td>
<td>72000 s</td>
</tr>
</tbody>
</table>
Polynomial multiplication: uniform magnitude

nanoseconds / (degree \times bits) for MPFRCX and Arb

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

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

Two paradigms: **modular/p-adic** and **complex analytic**
Example: constructing \( f(X) \in \mathbb{Z}[X] \) from its roots

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

Two paradigms: \textbf{modular/p-adic} and \textbf{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

<table>
<thead>
<tr>
<th>( p )</th>
<th>Degree (( n ))</th>
<th>Bits</th>
<th>Pari/GP</th>
<th>Arb</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 2^{607} - 1 )</td>
<td>729</td>
<td>502</td>
<td>0.03 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>( 2^{607} - 1 )</td>
<td>6561</td>
<td>7655</td>
<td>4.5 s</td>
<td>3.6 s</td>
</tr>
<tr>
<td>( 2^{607} - 1 )</td>
<td>59049</td>
<td>68937</td>
<td>944 s</td>
<td>566 s</td>
</tr>
</tbody>
</table>
Example: constructing \( f(X) \in \mathbb{Z}[X] \) from its roots

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

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

<table>
<thead>
<tr>
<th>( p )</th>
<th>Degree ((n))</th>
<th>Bits</th>
<th>Pari/GP</th>
<th>Arb</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 2^{607} - 1 )</td>
<td>729</td>
<td>502</td>
<td>0.03 s</td>
<td>0.02 s</td>
</tr>
<tr>
<td>( 2^{607} - 1 )</td>
<td>6561</td>
<td>7655</td>
<td>4.5 s</td>
<td>3.6 s</td>
</tr>
<tr>
<td>( 2^{607} - 1 )</td>
<td>59049</td>
<td>68937</td>
<td>944 s</td>
<td>566 s</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>( -D )</th>
<th>Degree</th>
<th>Bits</th>
<th>Pari/GP</th>
<th>classpoly</th>
<th>CM</th>
<th>Arb</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 10^6 + 3 )</td>
<td>105</td>
<td>8527</td>
<td>12 s</td>
<td>0.8 s</td>
<td>0.4 s</td>
<td>0.2 s</td>
</tr>
<tr>
<td>( 10^7 + 3 )</td>
<td>706</td>
<td>50889</td>
<td>194 s</td>
<td>8 s</td>
<td>29 s</td>
<td>20 s</td>
</tr>
<tr>
<td>( 10^8 + 3 )</td>
<td>1702</td>
<td>153095</td>
<td>1855 s</td>
<td>82 s</td>
<td>436 s</td>
<td>287 s</td>
</tr>
</tbody>
</table>
Special functions in Arb

The full complex domain for all parameters is supported

Elementary: \( \exp(z) \), \( \log(z) \), \( \sin(z) \), \( \atan(z) \), \( \expm1(z) \), 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: \( \text{erf}(z) \), \( \text{erfc}(z) \), \( E_s(z) \), \( \text{Ei}(z) \), \( \text{Si}(z) \), \( \text{Ci}(z) \), \( \text{Li}(z) \)

Bessel and Airy: \( J_\nu(z) \), \( Y_\nu(z) \), \( I_\nu(z) \), \( K_\nu(z) \), \( \text{Ai}(z) \), \( \text{Bi}(z) \)

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: \( 0F1(a, z) \), \( 1F1(a, b, z) \), \( U(a, b, z) \), \( 2F1(a, b, c, z) \)

Zeta, polylogarithms and L-functions: \( \zeta(s) \), \( \zeta(s, z) \), \( \text{Li}_s(z) \), \( L(\chi, s) \)

Theta, elliptic and modular: \( \theta_i(z, \tau) \), \( \eta(\tau) \), \( j(\tau) \), \( \Delta(\tau) \), \( G_{2k}(\tau) \), \( \wp(z, \tau) \)

Elliptic integrals: \( \text{agm}(x, y) \), \( K(m) \), \( E(m) \), \( F(\phi, m) \), \( E(\phi, m) \), \( \Pi(n, \phi, m) \), \( R_F(x, y, z) \), \( R_G(x, y, z) \), \( R_f(x, y, z, p) \), \( \wp^{-1}(z, \tau) \)
Example: algorithm for $K_ν(z)$

Large $|z|$: \[ K_ν(z) = \sqrt{\frac{\pi}{2z}} e^{-z} 2 F_0 \left( \nu + \frac{1}{2}, \frac{1}{2} - \nu, -\frac{1}{2z} \right) \]

Small $|z|, \nu \not\in \mathbb{Z}$:

\[ 2K_ν(2z) = z^ν \Gamma(-ν) \, _0F_1(1 + ν, z^2) + z^{-ν} \Gamma(ν) \, _0F_1(1 - ν, z^2) \]

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

The core building block is the hypergeometric series:

\[ pF_q(a_1, \ldots, a_p; b_1, \ldots, b_q; z) = \sum_{k=0}^{N-1} \frac{(a_1)_k \cdots (a_p)_k z^k}{(b_1)_k \cdots (b_q)_k k!} + \varepsilon_N \]

Compute using ball arithmetic

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

<table>
<thead>
<tr>
<th>Code</th>
<th>Average</th>
<th>Median</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\frac{1}{F_1}$ SciPy</td>
<td>2.7</td>
<td>0.76</td>
<td>18 good, 4 fair, 4 poor, 5 wrong, 2 NaN, 7 skipped</td>
</tr>
<tr>
<td>$\frac{2}{F_1}$ SciPy</td>
<td>24</td>
<td>0.56</td>
<td>18 good, 1 fair, 1 poor, 3 wrong, 1 NaN, 6 skipped</td>
</tr>
<tr>
<td>$\frac{2}{F_1}$ Michel &amp; S.</td>
<td>7.7</td>
<td>2.1</td>
<td>22 good, 1 poor, 6 wrong, 1 NaN</td>
</tr>
<tr>
<td>$\frac{1}{F_1}$ MMA (m)</td>
<td>1100</td>
<td>29</td>
<td>34 good, 2 poor, 4 wrong, 2 no significant digits out</td>
</tr>
<tr>
<td>$\frac{2}{F_1}$ MMA (m)</td>
<td>30000</td>
<td>72</td>
<td>29 good, 1 fair</td>
</tr>
<tr>
<td>$U$ MMA (m)</td>
<td>4400</td>
<td>190</td>
<td>28 good, 4 fair, 2 wrong, 6 no significant digits out</td>
</tr>
<tr>
<td>$Q$ MMA (m)</td>
<td>4300</td>
<td>61</td>
<td>21 good, 3 fair, 2 poor, 1 wrong, 3 NaN</td>
</tr>
<tr>
<td>$\frac{1}{F_1}$ MMA (a)</td>
<td>2100</td>
<td>170</td>
<td>39 good, 1 not good as claimed (actual error $2^{-40}$)</td>
</tr>
<tr>
<td>$\frac{2}{F_1}$ MMA (a)</td>
<td>37000</td>
<td>540</td>
<td>30 good ($2^{-53}$)</td>
</tr>
<tr>
<td>$U$ MMA (a)</td>
<td>25000</td>
<td>340</td>
<td>38 good, 2 not as claimed ($2^{-40}$, $2^{-45}$)</td>
</tr>
<tr>
<td>$Q$ MMA (a)</td>
<td>8300</td>
<td>780</td>
<td>28 good, 1 not as claimed ($2^{-25}$), 1 wrong</td>
</tr>
<tr>
<td>$\frac{1}{F_1}$ Arb</td>
<td>200</td>
<td>32</td>
<td>40 good (correct rounding)</td>
</tr>
<tr>
<td>$\frac{2}{F_1}$ Arb</td>
<td>930</td>
<td>160</td>
<td>30 good (correct rounding)</td>
</tr>
<tr>
<td>$U$ Arb</td>
<td>2000</td>
<td>93</td>
<td>40 good (correct rounding)</td>
</tr>
<tr>
<td>$Q$ Arb</td>
<td>3000</td>
<td>210</td>
<td>30 good ($2^{-53}$)</td>
</tr>
</tbody>
</table>

40 test cases for $\frac{1}{F_1} / U$ and 30 for $\frac{2}{F_1} / Q$ 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](http://sagemath.org)

- **Nemo.jl** and **Hecke.jl** - computer algebra and algebraic number theory in Julia
  [http://nemocas.org](http://nemocas.org)

- **Marc Mezzarobba**: rigorous evaluation of D-finite functions in SageMath

- **Pascal Molin, Christian Neurohr**: rigorous computation of period matrices of superelliptic curves
  [https://github.com/pascalmolin/hcperiods](https://github.com/pascalmolin/hcperiods)
Thank you!