Fundamental algorithms in Arb

Fredrik Johansson
LFANT, Inria Bordeaux & Institut de Mathématiques de Bordeaux

OSCAR : Antic Workshop
TU Kaiserslautern, Germany, 2017-08-02
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

- Started in 2012 to extend FLINT to $\mathbb{R}$ and $\mathbb{C}$
- Main types:
  - `arf_t` - arbitrary-precision floats
  - `mag_t` - unsigned floats with 30-bit precision
  - `arb_t` - real numbers $[\text{mid} \pm \text{rad}]$
  - `acb_t` - complex numbers $[a \pm r] + [b \pm s]i$
  - `arb_poly_t`, `acb_poly_t` - real and complex polynomials
  - `arb_mat_t`, `acb_mat_t` - real and complex matrices
- My main interest: special functions (analytic number theory), but intended for general purpose use
- Big rewrite in 2014 (2× speedup at low precision)
- Currently 140 000 lines of code (0.42 FLINTs)
- Notable recent feature: Dirichlet characters and Dirichlet L-functions (joint work with Pascal Molin)
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 $p(n) = 1, 1, 2, 3, 5, 7, 11, 15, 22, 30, 42\ldots$ 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!
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: $p(100) \in [190569292.00 \pm 0.39]$
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></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>553 s</td>
<td></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:

\[
\begin{align*}
[a \pm r] + [b \pm s] &\rightarrow [\text{round}(a + b) \pm (r + s + \varepsilon_{\text{round}})] \\
[a \pm r] \cdot [b \pm s] &\rightarrow [\text{round}(ab) \pm (|a|s + |b|r + rs + \varepsilon_{\text{round}})]
\end{align*}
\]

- More complicated operations generally involve doing several ball operations internally. The quality of enclosures reflects the algorithm!

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

\[
\sin(HUGE) \rightarrow [\pm 1] \text{ when more than } O(prec) \text{ bits needed for mod } \pi \text{ reduction}
\]
Content of the \texttt{arb\_t} type

| Exponent | \\
|---|---|
| Limb count + sign bit | \\
| Limb 0 | Allocation count |
| Limb 1 | Pointer to $\geq 3$ limbs |

Midpoint (\texttt{arf\_t}, 4 words)

\[ (-1)^s \cdot m \cdot 2^e, \quad \text{arbitrary-precision } \frac{1}{2} \leq m < 1 \quad \text{(or } 0, \pm\infty, \text{NaN)} \]

The mantissa \( m \) is an array of limbs, bit aligned like MPFR

Up to two limbs (128 bits), \( m \) is stored inline

Radius (\texttt{mag\_t}, 2 words)

\[ m \cdot 2^e, \quad \text{fixed 30-bit precision } \frac{1}{2} \leq m < 1 \quad \text{(or } 0, +\infty) \]

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: mpz, MPFI, MPFR, Arb]
Faster basic arithmetic (TOP SECRET WIP)

Squaring real numbers (arb_sqr)

![Graph showing time vs. precision for different arithmetics](image)
Polynomials in Arb

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

- Basic arithmetic, evaluation, composition
- Fast 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 $X \rightarrow 2^eX$
- 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 × 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: **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>
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>
The Durand-Kerner iteration gives numerical approximations of all $d$ complex roots simultaneously.

For any $z$, the ball $B(z, r)$ with $r = d|f(z)/f'(z)|$ contains at least one root of the polynomial.

If we get $d$ disjoint balls, we have found all roots (note: multiple roots will not work!)

User needs to write some wrapper code to increase precision, iterations.

New in Arb 2.11: arb_fmpz_poly_complex_roots
  - Increases precision, iterations automatically
  - Identifies all real roots and pairs complex conjugates
  - Implements power hack
Polynomial roots wishlist

- Do as much as possible with `double`
- Compute better initial values
- Use Aberth-Ehrlich method instead of Durand-Kerner
- Support close/clustered roots efficiently
- Parallel algorithm
- Newton iteration for high-precision refinement
- Dedicated algorithm for real roots
- Lazy interface + canonical root order:

```c
fmpz_poly_roots_t roots;
acb_t y;
...
fmpz_poly_roots_get_acb(y, roots, i, prec);
```
Linear algebra in Arb

- Multithreaded matrix multiplication

- Solving, LU decomposition, determinant, inverse (using Gaussian elimination)

- Cholesky and LDL decomposition and solving for real matrices (contributed by Alex Griffing)

- Characteristic polynomial \(O(n^4)\) algorithm

- Matrix exponential (fast algorithm using scaling + baby step giant step evaluation)
  - Improved error bounds for structured matrices by Alex Griffing
Linear algebra wishlist

- Linear solving using numerical approximation + posteriori certification
- Eigenvalues / eigenvectors
- Multiplication via \texttt{fmpz\_mat\_mul}
  - Using block + scaling strategy?
- Determinant via \texttt{fmpz\_mat\_det}
  - What about complex matrices?
- Sparse matrices
Special functions in Arb

The full complex domain for all parameters is supported

**Elementary:** \( \exp(z), \log(z), \sin(z), \tan(z), \expm1(z), \text{Lambert } W_k(z) \ldots \)

**Gamma, beta:** \( \Gamma(z), \log \Gamma(z), \psi^{(s)}(z), \Gamma(s, z), \gamma(s, z), B(z; a, b) \)

**Exponential integrals:** \( \text{erf}(z), \text{erfc}(z), \text{E}_s(z), \text{Ei}(z), \text{Si}(z), \text{Ci}(z), \text{Li}(z) \)

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

**Orthogonal:** \( P_{\mu}^{(a, b)}(z), Q_{\mu}^{(a, b)}(z), T_\nu(z), U_\nu(z), L_\nu(z), C_\nu^{{(a, b)}}(z), H_\nu(z), P_{\nu}^{(a, b)}(z) \)

**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:** \( \zeta(s), \zeta(s, z), \text{Li}_s(z), L(\chi, s) \)

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

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