Numerical integration in complex interval arithmetic

Fredrik Johansson

LFANT, Inria Bordeaux & Institut de Mathématiques de Bordeaux

MAX computer algebra seminar
LIX / Inria Saclay / École polytechnique
11 December 2017

Arb – http://arblib.org/

C library for arbitrary-precision ball arithmetic

- Real numbers $[m \pm r]$
- Complex numbers $[a \pm r] + [b \pm s]i$
- Polynomials, power series, matrices
- Special functions

Highlights in Arb 2.12:

- **Numerical integration (this talk)**
- Arbitrary-precision FFT (contributed by Pascal Molin)
- Faster sin/cos/exp at $>10^3$ digits
- Improved algorithms for elliptic functions

Goal

Numerical evaluation of

$$\int_a^b f(x)dx$$

with:

- Rigorous error bounds
- Possibility to obtain 100 or 10000 digits
- Support for complex numbers, special functions
- Support for badly behaved f (small, large, discontinuous)
- Minimal information required apart from black box evaluation of f in interval/ball arithmetic
- Sensible behavior when convergence is too slow

Applications: complex analysis

- (Inverse) Mellin/Laplace/Fourier transforms
- Computing Taylor/Laurent/Fourier series coefficients:

$$f(z) = \sum_{n=-\infty}^{\infty} c_n (z-a)^n, \quad c_n = \frac{1}{2\pi i} \oint_C \frac{f(z)}{(z-a)^{n+1}} dz$$

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{inx}, \quad c_n = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x) e^{-inx} dx$$

- Counting zeros and poles:

$$N - P = \frac{1}{2\pi i} \oint_C \frac{f'(z)}{f(z)} dz$$

- Acceleration of series (Euler-Maclaurin summation...)

Applications: computing special functions

Examples of integral representations:

$$\Gamma(s, z) = \int_z^\infty t^{s-1} e^{-t} dt$$

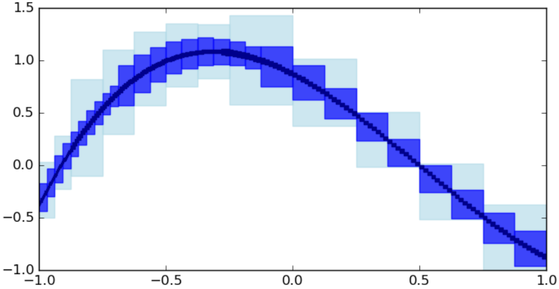
$$J_\nu(z) = \frac{1}{\pi} \int_0^\pi \cos(z \sin\theta - \nu\theta)\, d\theta - \frac{\sin(\nu\pi)}{\pi} \int_0^\infty e^{-z \sinh t - \nu t} dt$$

Benefits of direct integration:

- Useful especially with large parameters (faster convergence, less cancellation vs series expansions)
- Possibility to deform path (steepest descent method, analytic continuation)
- Automatic error bounds from integration algorithm

Brute force interval integration

$$\int_a^b f(x)dx \subseteq (b-a)f([a,b]) \quad + \quad \text{subdivision}$$



Pros: simple, only depends on direct interval evaluation of f

Cons: need $\sim 2^p$ evaluations for p-bit accuracy

Methods with high order convergence

For analytic f, we can use algorithms that give p-bit accuracy with $n = O(p)$ work:

- Taylor series of order n (via automatic differentiation)
- Quadrature rule with n evaluation points

Error bounds:

- Via derivatives $f^{(n)}$ on $[a, b]$
- Via $|f|$ on a complex domain around $[a, b]$

Quadrature rules

$$\int_{-1}^{1} f(x)dx \approx \sum_k w_k f(x_k)$$

Gauss-Legendre

- x_k = roots of Legendre polynomial $P_n(x)$, w_k from $P_n'(x_k)$

Clenshaw-Curtis

- x_k = Chebyshev nodes $\cos(\pi k/n)$, w_k from FFT
- need about 2 times as many points as Gauss-Legendre

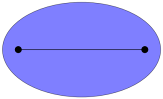
Double exponential

- x_k, w_k from change of variables $x = \tanh(\frac{1}{2}\pi \sinh t)$ and trapezoidal approximation $\int_{-\infty}^{\infty} g(t)dt \approx h \sum_{k=-n}^{n} g(hk)$
- need > 5 times as many points as Gauss-Legendre

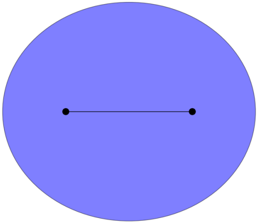
Error bounds using complex magnitudes

If f is analytic with $|f(z)| \leq M$ on an ellipse E with foci $-1, 1$ and semi-axes X, Y with $\rho = X + Y > 1$, then the error for n-point Gauss-Legendre quadrature satisfies

$$\left| \int_{-1}^{1} f(x) dx - \sum_{k=0}^{n-1} w_k f(x_k) \right| \leq \frac{M}{\rho^{2n}} \cdot \frac{64\rho}{15(\rho - 1)}$$



$X = 1.25, Y = 0.75, \rho = 2.00$



$X = 2.00, Y = 1.73, \rho = 3.73$

Adaptive integration algorithm

1. Compute $(b-a)f([a,b])$. If the error is $\leq \varepsilon$, done!
2. Compute $|f(z)|$ and check analyticity of f on some ellipse E around $[a,b]$. If the error of Gauss-Legendre quadrature is $\leq \varepsilon$, compute it – done!
3. Split at $m=(a+b)/2$ and integrate on $[a,m]$, $[m,b]$ recursively.

Knut Petras published a version of this algorithm in 2002 and pointed out that it guarantees rapid convergence for a large class of piecewise analytic functions.

Choosing the quadrature degree n for $[a, b]$

Strategy used by Arb's integration code:

Set $n_{\text{best}} = \infty$.
For a sequence of E_i around $[-1, 1]$ with $\rho_i = 3.73, \ldots \sim 2^{2^i}, 2^i < p$:

- Compute $M \geq |\frac{b-a}{2} f(\frac{b-a}{2} E_i + \frac{a+b}{2})|$. If $M = \infty$, break. (Here, also $M = \infty$ if analyticity fails.)
- Determine the smallest n such that the error bound is $\leq \varepsilon$ and set $n_{\text{best}} = \min(n_{\text{best}}, n)$, if such an n exists.

Proceed with Gauss-Legendre quadrature if $n_{\text{best}} < \infty$.

Constraints on the degree:

- $n \leq 0.5p + 10$ by default (can be changed by user)
- n is chosen among $1, 2, 4, 6, 8, 12, 16, 22, 32, 46, \ldots \approx 2^{j/2}$

Using the integration code

```
int acb_calc_integrate(acb_t res,       /* output */
  acb_calc_func_t func,                 /* integrand */
  void * param,                         /* parameters to func */
  const acb_t a,                        /* endpoints */
  const acb_t b,
  slong rel_goal,                       /* relative goal */
  const mag_t abs_tol,                  /* absolute goal */
  const acb_calc_integrate_opt_t opt, /* optional options */
  slong prec)                           /* working precision */
```

Documentation: `http://arblib.org/acb_calc.html`

Demonstration program, with code for all integrals in this talk:
`https://github.com/fredrik-johansson/arb/blob/master/examples/integrals.c`

Defining object functions

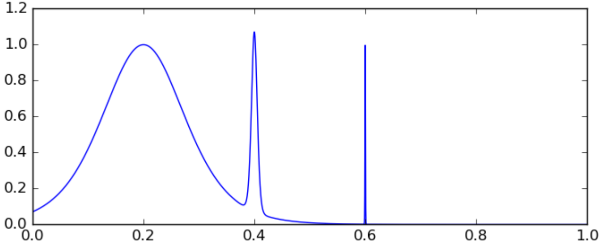
- $d = 0$: set res to $f(z)$
- $d = 1$: test analyticity + set res to $f(z)$
- $d > 1$: test analyticity + set res to d coeffs of Taylor series ($d > 1$ is not used by the integration code)

```
int f_tan_3z(acb_ptr res, const acb_t z, void * param, slong d, slong prec)
{
  acb_mul_ui(res, z, 3, prec);
  acb_tan(res, res, prec);
  return 0;
}

int f_sqrt(acb_ptr res, const acb_t z, void * param, slong d, slong prec)
{
  if (d > 0 &&       /* catch branch cut */
       arb_contains_zero(acb_imagref(z)) && !arb_is_positive(acb_realref(z)))
    acb_indeterminate(res);
  else
    acb_sqrt(res, z, prec);
  return 0;
}
```

An example integral (from the *Mathematica* docs)

$$\int_0^1 \operatorname{sech}^2(10(x-0.2)) + \operatorname{sech}^4(100(x-0.4)) + \operatorname{sech}^6(1000(x-0.6))\,dx$$



Some results (with default options):

Mathematica `NIntegrate`:	0.209736
Sage `numerical_integral`:	0.209736, error estimate 10^{-14}
SciPy `quad`:	0.209736, error estimate 10^{-9}
mpmath `quad`:	0.209819
Pari/GP `intnum`:	0.211316
Actual value:	0.2108027355...

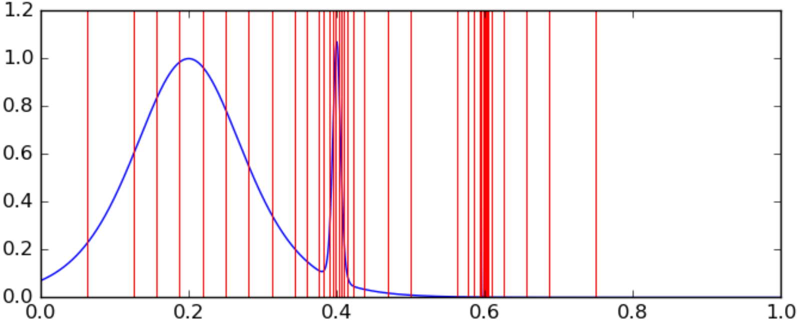
Results with the new integration code in Arb

p	Time (s)	Sub.	Eval.	Result
32	0.0030	49	809	`[0.2108027 +/- 4.21e-8]`
64	0.0051	49	1299	`[0.21080273550054928 +/- 4.44e-18]`
333	0.038	49	4929	`[0.2108027355...  +/- 3.72e-99]`
3333	8.7 (*)	49	48907	`[0.2108027355...  +/- 1.39e-1001]`

(*) with $p = 3333$, the time is 11 seconds on a first run due to nodes precomputation

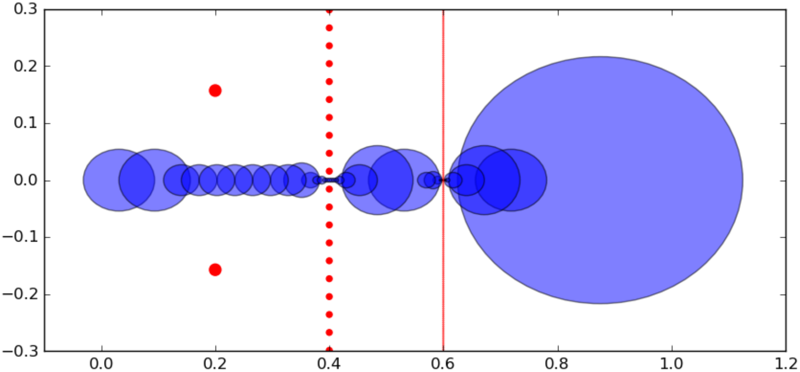
Sub. = total number of terminal subintervals
Eval. = total number of integrand evaluations

Adaptive subdivision performed by Arb



49 terminal subintervals (smallest width 2^{-12})

Adaptive subdivision, complex view

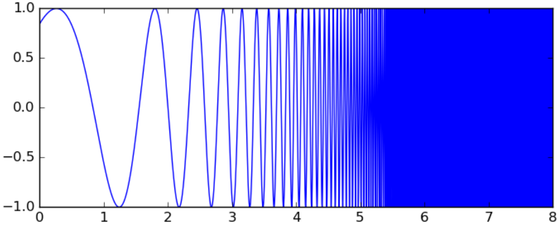


Blue ellipses used for error bounds on the subintervals

Red dots: poles of the integrand

Rump's oscillatory example

$$\int_0^8 \sin(x + e^x)\,dx$$



Siegfried Rump (2010) noticed that MATLAB's `quad` computes an incorrect result (after running for about 1 second).

Rump's INTLAB `verifyquad` computes the correct enclosure `[0.34740016, 0.34740018]` in 2 seconds.

This integral was also used by Mahboubi, Melquiond & Sibut-Pinote (2016) as an example for CoqInterval. CoqInterval computes 1 digit in 80 s and 4 digits in 277 s.

Results with the new integration code in Arb

p	Time (s)	Subint.	Eval.	Result
32	0.0067	110	3689	`[0.34740 +/- 7.80e-6]`
64	0.0085	96	4325	`[0.34740017265725 +/- 3.95e-15]`
333	0.021	39	5410	`[0.3474001726... +/- 5.97e-96]`
3333	1.2 (*)	8	10417	`[0.3474001726... +/- 2.95e-999]`

(*) 5.3 seconds on a first run due to nodes precomputation

For comparison, mpmath `quad`:

- 0.01 s, wrong result with 53-bit prec
- 0.12 s, correct result with 53-bit prec + maxdegree=10
- 12 s, correct result with 3333-bit prec

Pari/GP `intnum`:

- 0.01 s, wrong result with 38-digit prec
- 0.08 s, correct result with 38-digit prec + tab=5
- 3 s, wrong result with 1000-digit prec
- 14 s, correct result with 1000-digit prec + tab=2

Error tolerances

The user specifies:

- Absolute tolerance ε_{abs}
- Relative tolerance ε_{rel} (as $2^{-\text{rel\_goal}}$)
- Working precision p

Goal: error $\leq \max(\varepsilon_{\text{abs}}, M\varepsilon_{\text{rel}})$, where $M = |\int_a^b f(x)dx|$.

(This goal is just a guideline for the algorithm, and the width of the output interval can be larger.)

$\varepsilon_{\text{abs}} = \varepsilon_{\text{rel}} = 2^{-p}$ works well for most applications.

Can set $\varepsilon_{\text{abs}} = 0$ to force relative tolerance.

Relative tolerance (large or small M, or $\varepsilon_{\text{abs}} = 0$)

Problem: we don't know $M = |\int_a^b f(x)dx|$ in advance.

M has to be estimated while it is being computed.

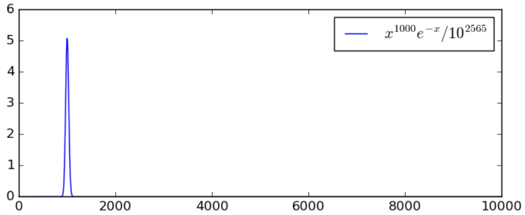
- Too large estimate: the final result will have a large error
- Too small estimate: we waste time on small parts

Current solution: use lower bounds (up to cancellation) for M. Every time we compute an enclosure for $I_k = \int_{a_k}^{b_k} f(x)dx$, we get $M_{\text{low}} \le |I_k| \le M_{\text{high}}$. Set $\varepsilon_{\text{abs}} \leftarrow \max(\varepsilon_{\text{abs}}, M_{\text{low}}\varepsilon_{\text{rel}})$.

If the user has a good guess for M, setting $\varepsilon_{\text{abs}} \approx \varepsilon_{\text{rel}} M$ is more efficient than $\varepsilon_{\text{abs}} = 0$.

A tall peak

$$\int_0^{10000} x^{1000} e^{-x} dx = \gamma(1001, 10000) \approx 4.0238726 \cdot 10^{2567}$$



With $p = 64$, $\varepsilon_{\text{rel}} = 2^{-64}$ and $\varepsilon_{\text{abs}} = 0$ or $\varepsilon_{\text{abs}} = 2^{-64}$:
`[4.023872600770938e+2567 +/- 8.39e+2551]` in 0.06 seconds

With $p = 64$, $\varepsilon_{\text{rel}} = 2^{-64}$ and $\varepsilon_{\text{abs}} = 10^{2551}$:
`[4.02387260077094e+2567 +/- 3.19e+2552]` in 0.006 seconds

With $p = 3333$, $\varepsilon_{\text{rel}} = 2^{-3333}$, $\ldots$: 1.5 seconds vs 0.6 seconds

A small magnitude

$$\int_{-1020}^{-1010} e^x dx \approx 2.304 \cdot 10^{-439}$$

With $p = 64$ and $\varepsilon_{\text{rel}} = \varepsilon_{\text{abs}} = 2^{-64}$:
`[+/- 2.31e-438]` in 10^{-6} seconds (1 function evaluation)

With $\varepsilon_{\text{abs}} = 0$:
`[2.304377150949363e-439 +/- 5.91e-455]` in 0.00015 seconds

With $\varepsilon_{\text{abs}} = 10^{-455}$:
`[2.304377150949363e-439 +/- 5.99e-455]` in 0.000028 seconds

Singularities and infinite intervals

Convergence requires $|a|, |b|, |f| < \infty$. Can use manual truncation, e.g. $\int_0^\infty f(x)dx \approx \int_\varepsilon^N f(x)dx$ otherwise.

Integral	Problem	Truncation	Evaluations
$\int_0^\infty \frac{xe^{-x}}{1+e^{-x}}$	Exponential decay	$N \approx p \log(2)$	$O(p \log p)$
$\int_0^1 \sqrt{1-x^2}dx$	Branch point (f finite)	Not needed	$O(p^2)$
$\int_0^\infty \frac{dx}{1+x^2}$	Algebraic decay	$N \approx 2^p$	$O(p^2)$
$\int_0^1 \frac{-\log(x)}{1+x}dx$	Branch point (f infinite)	$\varepsilon \approx 2^{-p}$	$O(p^2)$

- Manual truncation is not an ideal solution, but the algorithm is at least robust enough to work with large N or small ε
- $O(p^2)$ cost can be avoided with exponential change of variables
- Future improvement: automatic algorithm, provided that user supplies extra “global” information, e.g. $|f(x)| < x^\alpha e^{\beta x^\gamma}$

Timings

Integral	p	Time (s)	Subint.	Evaluations
$\int_0^1 \frac{dx}{1+x^2}$	333	0.00019	2	188
	3333	0.013	2	2056
$\int_0^\infty e^{-x^2} dx$ (*)	333	0.0012	4	551
	3333	0.22	4	3894
$\int_0^\infty \frac{xe^{-x}}{1+e^{-x}} dx$ (*)	333	0.0028	10	994
	3333	0.82	14	14097
$\int_0^1 \sqrt{1-x^2} dx$	333	0.014	223	12735
	3333	7.4	2223	1188115
$\int_0^\infty \frac{dx}{1+x^2}$ (*)	333	0.047	998	51907
	3333	27	9998	4711135
$\int_0^1 \frac{-\log(x)}{1+x} dx$ (*)	333	0.10	997	52024
	3333	335	9997	4720879

(*) by manual truncation + separate truncation error bound

Work limits

In practice, convergence might be too slow, or even impossible due to:

- Pole on the integration path
- Insufficient working precision
- Too much blowup in interval evaluation of integrand

If convergence looks too slow, abort gracefully!

Configurable work limits:

- Number of calls to the integrand (default: $O(p^2)$)
- Number of queued subintervals (default: $O(p)$)

(More sophisticated approaches are possible.)

Adaptive subdivision with work limits

$S \leftarrow 0, \quad Q \leftarrow [(a, b)]$.
While $Q = [(a_0, b_0), \ldots, (a_n, b_n)]$ is not empty:

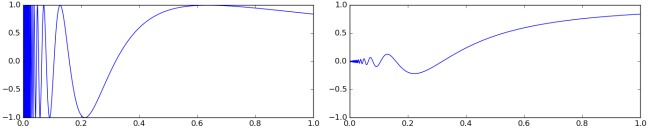
1. Pop $(\alpha, \beta) = (a_n, b_n)$ from Q
2. If integration on (α, β) meets the tolerance goal **or limits have been exceeded**, set $S \leftarrow S + \int_\alpha^\beta f(x)dx$
3. Otherwise, let $m = \frac{\alpha+\beta}{2}$ and extend Q with $(\alpha, m), (m, \beta)$

For (a_k, b_k), also store $v_k = (b_k - a_k)f([a_k, b_k])$.

- Local subdivision (default): in (3), append to Q and ensure $\mathrm{rad}(v_n) \leq \mathrm{rad}(v_{n+1})$.
- Global priority queue (using max-heap): in (3), ensure $\mathrm{rad}(v_0) \leq \ldots \leq \mathrm{rad}(v_{n+1})$.

Example: too much oscillation

$$I_1 = \int_0^1 \sin(1/x)\,dx, \qquad I_2 = \int_0^1 x\sin(1/x)\,dx$$



Default options, 64-bit precision, taking 0.2 seconds:

`[+/- 1.27], [+/- 1.12]`

With $\varepsilon_{\text{abs}} = 10^{-6}$, taking 0.01 and 0.0008 seconds:

`[0.504 +/- 2.68e-4], [0.37853 +/- 6.35e-6]`

With heap, taking 0.007 and 0.01 seconds:

`[0.504 +/- 7.88e-4], [0.3785300 +/- 3.17e-8]`

With heap, work limits bumped to 10^7, taking 17 seconds:

`[0.504067 +/- 2.78e-7], [0.3785300171242 +/- 5.75e-14]`

Why not use global priority queue by default?

Optimize for eventual convergence

- Local subdivision tends to complete one problematic area before moving on to the next one – Q is kept small
- Global algorithm tends to deal with all problematic areas simultaneously – Q can blow up

A better algorithm might:

- Combine global and local strategies
- Estimate the priority of a subinterval more intelligently than by looking at the error of $(b_k - a_k) f([a_k, b_k])$

Piecewise and discontinuous functions

Functions like $\lfloor x \rfloor$ and $|x|$ on $\mathbb{R}$ can be extended to piecewise holomorphic functions on $\mathbb{C}$.

$$f(x) = |x| \to f(x+yi) = \sqrt{(x+yi)^2} = \begin{cases} x+yi & x > 0 \\ -(x+yi) & x < 0 \end{cases}$$

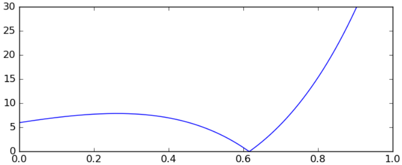
(discontinuous at $x = 0$)

$f(x) = \lfloor x \rfloor \to f(x+yi) = \lfloor x \rfloor + yi$ (discontinuous at $x \in \mathbb{Z}$)

Note: this trick does not work for $\int_a^b |f(z)| dz$ where f is a *complex* function. However, if we have a decomposition $f(z) = g(z) + h(z)i$, we can use $|f(z)| = \sqrt{g(z)^2 + h(z)^2}$. Taylor methods are more useful in such cases.

Examples

1: Helfgott's example $\int_0^1 |(x^4 + 10x^3 + 19x^2 - 6x - 6)| \exp(x)\, dx$

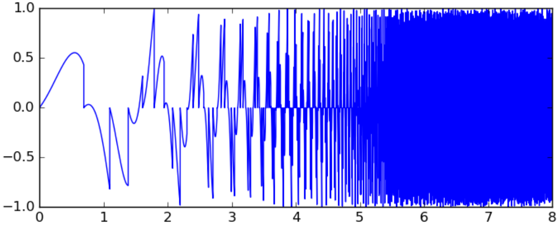


2: The "Gauss sum" $\int_1^{101} \lfloor x \rfloor\, dx = \sum_{n=1}^{100} n = 5050$

Integral	p	Time (s)	Subint.	Evaluations
$\int_0^1 \lvert \cdot \rvert \exp(x)\, dx$	64	0.0015	70	1093
	333	0.052	339	18137
	3333	108	3339	1624951
$\int_1^{101} \lfloor x \rfloor\, dx$	64	0.014	5536	16606
	333	0.11	33512	100534
	3333	1.5	345512	1036534

Rump's example revisited

$$\int_0^8 (e^x - \lfloor e^x \rfloor)\sin(x + e^x)\,dx$$



64-bit precision, default work limit:
`[+/- 7.32e+3]` in 0.18 seconds

64-bit precision, increased limit:
`[0.0986517044784 +/- 4.37e-14]` in 9.1 seconds

333-bit precision, increased limit:
`[0.0986517...0645824 +/- 5.99e-95]` in 548 seconds

Special functions

Special functions implemented in Arb work out of the box.

Integral	p	Time (s)	Subint.	Evaluations
$\int_0^{1000} W_0(x)dx$	64	0.00081	12	273
	333	0.0092	12	1109
	3333	1.3	12	12043
$\int_1^{1+1000i} \Gamma(x)dx$	64	0.023	64	1524
	333	0.46	69	6502
	3333	225	72	73423

W_k: Lambert W function

Caveats:

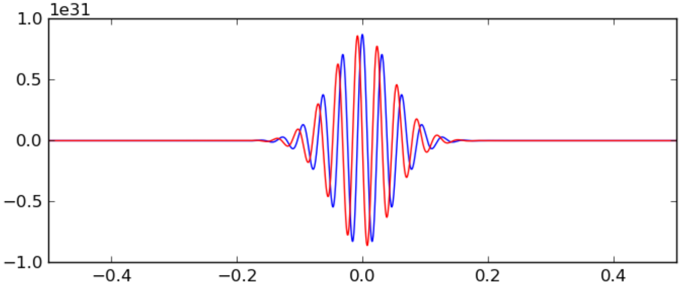
- The user must check for overlap with branch cuts in the evaluation of the integrand (e.g. $(-\infty, -1/e)$ for W_0)
- Many functions (e.g. $\Gamma(x)$) will currently output poor enclosures for wide input intervals

Example: Laurent series of elliptic functions

$$\wp(z;\tau) = \sum_{n=-2}^{\infty} a_n(\tau) z^n, \quad a_n = \frac{1}{2\pi i} \int_\gamma \frac{\wp(z)}{z^{n+1}} dz$$

Fix $\tau = i \Rightarrow \wp(z)$ has poles at $z = M + Ni \quad (M, N \in \mathbb{Z})$.

Pick $\gamma =$ square of width 1 centered on $z = 0$.
One segment ($n = 100$):



Example: Laurent series of elliptic functions

Time per integral ($n \leq 100$):
64 bits: 0.05 seconds
333 bits: 0.8 seconds
3333 bits: 120 seconds

Results with 333-bit precision:

```
a[-2] = [1.000000000000000...00000 +/- 3.57e-98] + [+/- 1.89e-98]*I
a[-1] = [+/- 4.11e-98] + [+/- 2.57e-98]*I
a[0] = [+/- 1.02e-97] + [+/- 5.39e-98]*I
a[1] = [+/- 1.41e-97] + [+/- 1.35e-97]*I
a[2] = [9.453636006461692...52235 +/- 4.44e-97] + [+/- 2.48e-97]*I
a[3] = [+/- 4.47e-97] + [+/- 4.60e-97]*I
...
a[94] = [380.0000000000013500...63746 +/- 9.24e-70] + [+/- 8.27e-70]*I
a[95] = [+/- 1.37e-69] + [+/- 1.37e-69]*I
a[96] = [+/- 2.93e-69] + [+/- 2.91e-69]*I
a[97] = [+/- 5.81e-69] + [+/- 5.82e-69]*I
a[98] = [395.9999999999996482...46383 +/- 2.90e-68] + [+/- 1.17e-68]*I
a[99] = [+/- 2.32e-68] + [+/- 2.32e-68]*I
a[100] = [+/- 4.95e-68] + [+/- 4.95e-68]*I
```

Example: counting zeros of Riemann's zeta function

How many zeros does the Riemann zeta function have on $R = [0,1] + [0,T]i$?

$$N(T) = 1 + \frac{1}{2\pi i}\int_\gamma \frac{\zeta'(s)}{\zeta(s)}\,ds$$

$\gamma =$ contour around R (plus small excursion around $s = 1$)

More useful version:

$$N(T) = 1 + \frac{\theta(T)}{\pi} + \frac{1}{\pi}\,\mathrm{Im}\left[\int_{1+\varepsilon}^{1+\varepsilon+Ti} \frac{\zeta'(s)}{\zeta(s)}\,ds + \int_{1+\varepsilon+Ti}^{\frac{1}{2}+Ti} \frac{\zeta'(s)}{\zeta(s)}\,ds\right]$$

Can take ε large, e.g. $\varepsilon = 100$.

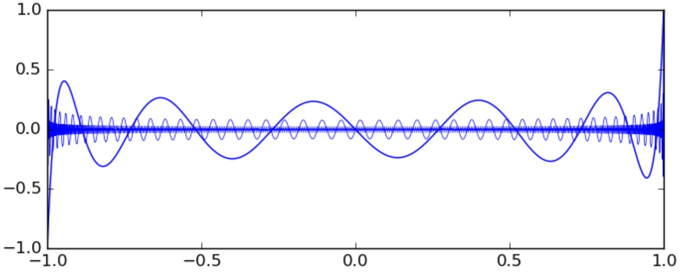
Example: counting zeros of Riemann's zeta function

T	Time (s)	Eval.	Subint.	$N(T)$
10^2	0.044	261	24	`[29.00000 +/- 1.94e-6]`
10^3	0.51	1219	109	`[649.00000 +/- 7.78e-6]`
10^4	13	6901	621	`[10142.0000 +/- 4.25e-5]`
10^5	12	4088	353	`[138069.000 +/- 3.10e-4]`
10^6	16	5326	440	`[1747146.00 +/- 4.06e-3]`
10^7	42	4500	391	`[21136125.0000 +/- 5.53e-5]`
10^8	210	6205	533	`[248008025.0000 +/- 8.09e-5]`
10^9	1590	8070	677	`[2846548032.000 +/- 1.95e-4]`

With tolerance 10^{-6}, prec = 32 bits ($T \le 10^6$), 48 bits ($T \ge 10^7$).

Legendre polynomials and Gauss-Legendre nodes

This is joint work with Marc Mezzarobba.



Goal: fast and rigorous evaluation of $P_n(x)$ on $[-1, 1]$ and computation of the Gauss-Legendre nodes and weights

$$P_n(x_k) = 0, \quad w_k = \frac{2}{\left(1 - x_k^2\right) [P'_n(x_k)]^2}, \quad 0 \le k < n$$

Overall strategy

- Newton iteration converges from initial approximations $x_k \approx \cos\left(\frac{4k+3}{4n+2}\pi\right)$ (error bounds by Petras, 1999)
- For high precision, use interval Newton method with doubling precision steps
- By symmetry, can assume $k < n/2$ and $x_k \in (0, 1)$
- For $x = [m \pm r]$, can evaluate at m and bound error using bounds for $|P_n'(x)|$ and $|P_n''(x)|$
- We can obtain $P_n'(x)$ from $(P_n(x), P_{n-1}(x))$ using contiguous relations
- The problem is now reduced to simultaneous computation of $P_n(x)$ and $P_{n-1}(x)$, with exact $x \in [0, 1]$

Remarks on complexity

What is the bit complexity of computing the n roots of P_n (and the corresponding weights) to p-bit accuracy?

- $\widetilde{O}(n^2 p)$ classically
- $\widetilde{O}(n)$ assuming $p = O(1)$, using asymptotic methods (fast methods for 53-bit IEEE 754 arithmetic recently by Townsend, Hale, Bogaert and others)

Assuming $p \sim n$ (which is most interesting for integration), the first bound can be improved from $\widetilde{O}(n^3)$ to $\widetilde{O}(n^2)$.

Algorithm: do Newton iteration for all roots simultaneously using fast multipoint evaluation of the expanded polynomials P_n, P_n'. Unfortunately, this method is slow in practice.

Hybrid evaluation algorithm

Three-term recurrence (n and p small):

$$(n+1)P_{n+1}(x) - (2n+1)xP_n(x) + nP_{n-1}(x) = 0$$

Hypergeometric series expansions:

$P_n(x) = \sum_{k=0}^{n} c_k x^k$ (truncated when x is near 0)
$P_n(x) = \sum_{k=0}^{n} d_k (x-1)^k$ (truncated when x is near 1)

Asymptotic expansion (large n, for x not too close to 1):

$$P_n(\cos(\theta)) \sim \sum_{k=0}^{\infty} \frac{a_k(n,\theta)}{\sin^k(\theta)}$$

Algorithm selection: for each series expansion, estimate cost = (number of terms) · (working precision), choose method with lowest cost.

Stability of the three-term recurrence

Example: $P_n(0.40625)$ via the three-term recurrence, using:

- 53-bit floating-point arithmetic
- 53-bit ball arithmetic

n	Floating-point error	Ball result
10	$6 \cdot 10^{-18}$	`[0.24468343638404 +/- 8.81e-17]`
20	$2 \cdot 10^{-17}$	`[0.07466174411982 +/- 8.44e-15]`
40	$4 \cdot 10^{-17}$	`[-0.1291065547 +/- 3.76e-11]`
100	$1 \cdot 10^{-18}$	`[+/- 0.239]`
200	$6 \cdot 10^{-17}$	`[+/- 1.72e+16]`
400	$5 \cdot 10^{-17}$	`[+/- 2.93e+50]`

With naive error bounds, we would need $O(n)$ extra precision.

Error bounds for the three-term recurrence

Exact version:

$$P_{n+1} = \frac{1}{(n+1)} \left((2n+1)xP_n - nP_{n-1}\right)$$

Approximate version:

$$\tilde{P}_{n+1} = \frac{1}{n+1}\Big((2n+1)x\tilde{P}_n - n\tilde{P}_{n-1}\Big) + \varepsilon_n, \quad |\varepsilon_n| \le \bar{\varepsilon}$$

We can show:

$$|\tilde{P}_n - P_n| \le \frac{(n+1)(n+2)}{4}\bar{\varepsilon}$$

Efficient implementation with `mpz_t` fixed-point arithmetic.

Proof sketch

The sequence of errors $\delta_n = \tilde{P}_n - P_n$ satisfies the recurrence

$$(n+1)\delta_{n+1} = (2n+1)x\delta_n - n\delta_{n-1} + \eta_n, \quad \eta_n = (n+1)\varepsilon_n.$$

This translates to a differential equation

$$\delta(z) = \sum_{n\geq 0} \delta_n z^n, \qquad \eta(z) = \sum_{n\geq 0} \eta_n z^n$$

$$(1 - 2xz + z^2)z\frac{d}{dz}\delta(z) = z(x-z)\delta(z) + z\eta(z)$$

with solution

$$\delta(z) = p(z)\int_0^z \eta(w)\,p(w)\,dw, \quad p(z) = \frac{1}{\sqrt{1 - 2xz + z^2}}.$$

Computing a majorant for $\delta(z)$ gives the result.

Hypergeometric series expansions

Close to 1:

$$P_n(1-x) = \sum_{k=0}^{n} \binom{n}{k} \binom{n+k}{k} \left(\frac{-x}{2}\right)^k$$

Close to 0 (and also at high precision):

$$P_{2d+j}(x) = \sum_{k=0}^{d} \frac{(-1)^{d+k}}{2^n} \binom{n}{d-k} \binom{n+2k+j}{n} x^{2k+j}, \quad j \in \{0,1\}$$

Truncation bounds: first omitted term $\times$ geometric series

Estimates for cancellation (needed working precision) via:

- $P_n(1+x) \approx \sum_{k=0}^{\infty} \frac{n^{2k}}{(k!)^2} \left(\frac{x}{2}\right)^k = I_0(2n\sqrt{x/2}) \approx e^{2n\sqrt{x/2}}$
- $|P_n(z)| \leq |P_n(i|z|)| \leq \left(|z| + \sqrt{1+|z|^2}\right)^n$

Fast evaluation of hypergeometric series

Compute

$$\sum_{k=0}^{K} c_k x^k, \quad c_k / c_{k-1} \in \mathbb{Q}(k)$$

using $2\sqrt{K}$ expensive multiplications + $O(K)$ cheap multiplications and divisions by small integers:

- Rectangular splitting:

$$(\square + \square x + \ldots + \square x^{m-1}) + x^m((\square + \square x + \ldots) + x^m(\ldots))$$

- Use c_k / c_{k-1} to get small coefficients (Smith, 1989)
- Collect denominators to skip most divisions (FJ, 2015)
- For P_n, P_n' simultaneously: recycle powers $x^2, \ldots, x^m$

Implementation using ball arithmetic for error bounds.

Asymptotic expansion

For large n and $x = \cos(\theta) < 1$:

$$P_n(\cos(\theta)) = \left(\frac{2}{\pi \sin(\theta)}\right)^{1/2} \sum_{k=0}^{K-1} C_{n,k} \frac{\cos(\alpha_{n,k}(\theta))}{\sin^k(\theta)} + \xi_{n,K}(\theta)$$

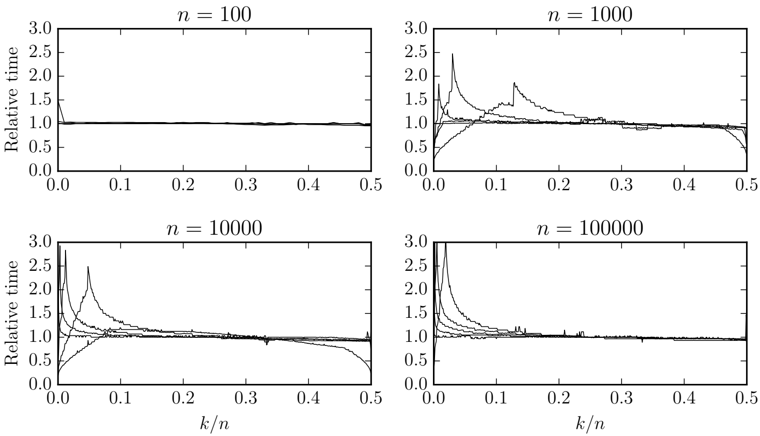
$$C_{n,k} = \frac{[\Gamma(k+\frac{1}{2})]^2 \Gamma(n+1)}{\pi 2^k \Gamma(n+k+\frac{3}{2})\Gamma(k+1)}, \quad |\xi_{n,K}(\theta)| < \sqrt{\frac{8}{\pi \sin(\theta)}} \frac{C_{n,K}}{\sin^K(\theta)}$$

Let $\omega = 1 - (x/y)i$, with $x = \cos(\theta)$ and $y = \sin(\theta)$. Then

$$P_n(x) = \sqrt{\pi y}\ \mathrm{Re}\left[(1-i)(x+yi)^{n+1/2} \sum_{k=0}^{K-1} C_{n,k} \omega^k\right] + \xi_{n,K}(\theta).$$

By working with complex numbers, the sum becomes a pure hypergeometric series and rectangular splitting can be used.

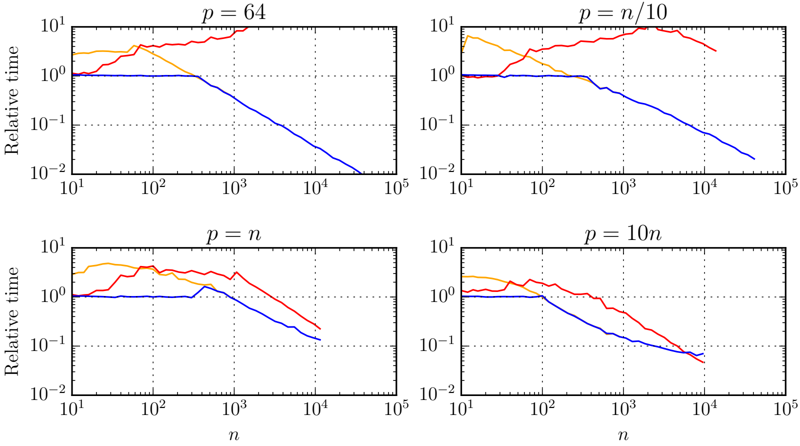
Algorithm selection profile



Normalized time to evaluate $(P_n(x), P_n'(x))$ for x near x_k, $0 \leq k < n/2$ ($x \approx 0$ near $k/n = 0.5$ and $x \approx 1$ near $k/n = 0$)

The separate curves show $p = 64, 256, 1024, 4096, 16384$

Time to evaluate (P_n, P'_n) at $n/2$ points



- Baseline (10^0): three-term recurrence
- Blue: our hybrid algorithm
- Orange: hybrid algorithm without three-term recurrence
- Red: fast multipoint evaluation

Time to compute nodes and weights

$n \setminus p$	64	256	1024	3333	33333
20	0.000149	0.000300	0.000660	0.00149	0.0217
50	0.000540	0.00119	0.00267	0.00590	0.0760
100	0.00181	0.00380	0.00900	0.0188	0.205
200	0.00660	0.0141	0.0310	0.0640	0.624
500	0.0289	0.0850	0.214	0.384	2.80
1000	0.0660	0.174	0.625	1.36	9.68
2000	0.106	0.362	1.20	4.52	34.3
5000	0.235	0.815	2.92	14.6	189
10000	0.480	1.63	5.49	27.3	694
100000	4.90	16.1	49.6	221	13755
1000000	73.0	195	512	2016	105705

Time in seconds to compute the degree-n Gauss-Legendre quadrature rule with p-bit precision.

Generating 1000-digit nodes: comparison

D. H. Bailey's ARPREC precomputes 3408-bit Gauss-Legendre rules of degree $n = 3 \cdot 2^{i+1}$, $1 \leq i \leq 10$ intended for 1000-digit integration.

n	ARPREC	Arb
12	0.00520	0.000735
24	0.0189	0.00197
48	0.0629	0.00574
96	0.251	0.0185
192	0.974	0.0611
384	3.83	0.231
768	15.2	0.875
1536	60.9	3.03
3072	241	9.75
6144	1013	18.4

Time in seconds.

Gauss vs Clenshaw-Curtis vs double exponential

Recall: number of points for equivalent accuracy

- Gauss-Legendre: n
- Clenshaw-Curtis: $\approx 2n$
- Double exponential: $> 5n$

Time to generate suitable quadrature rule:

1000 digits

- GL: 1 second
- CC: 0.1 seconds
- DE: 0.1 seconds

10000 digits

- GL: 10 minutes
- CC: 0.5 minutes
- DE: 2 minutes

Rough estimate: Gauss-Legendre is competitive if the integrand costs m elementary function (log, exp, ...) evaluations, or if the integration requires m subintervals, or m integrals will be computed, for $m \approx 10$.

Summary

Gauss-Legendre quadrature:

- Order of magnitude improvement for computing nodes
- Gauss-Legendre quadrature becomes practical even at very high precision (1000 or 10000 digits)
- Extension to Jacobi polynomials would be useful

Numerical integration:

- A Petras-style adaptive complex analytic algorithm implemented in ball arithmetic seems to work extremely well in practice for rigorous high precision integration
- Should be tested in more applications (likely requiring fine-tuning of the methods)
- Further comparison with Taylor methods would be useful
- Further work needed for improper integrals