Fast and rigorous arbitrary-precision evaluation of Legendre polynomials and Gauss-Legendre quadrature nodes

Fredrik Johansson Marc Mezzarobba

Journées Nationales de Calcul Formel 2018
CIRM, Luminy
22 January 2018

Numerical integration

$$\int_{-1}^{1} f(x)dx \approx \sum_{i=1}^{n} w_i f(x_i)$$

n = degree
x_i = nodes
w_i = weights

Gauss-Legendre quadrature: $x_1, \ldots, x_n$ are the roots of the Legendre polynomial $P_n(x)$, defined by orthogonality

$$\int_{-1}^{1} P_n(x)P_m(x)dx = \begin{cases} 0 & m \neq n \\ \frac{2}{2n+1} & m = n. \end{cases}$$

The weights are

$$w_i = \frac{2}{\left(1 - x_i^2\right)\left[P_n'(x_i)\right]^2}.$$

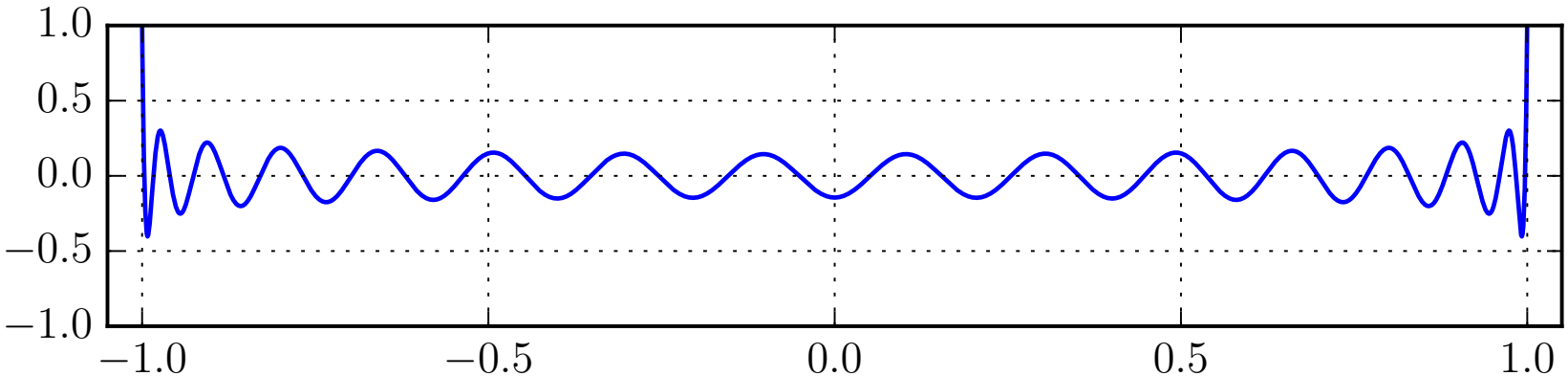
Legendre polynomials

Equivalent definitions:

$$P_n(x) = {}_2F_1\left(-n, n+1, 1, \tfrac{1}{2}(1-x)\right)$$

$$\frac{1}{\sqrt{1-2xt+t^2}} = \sum_{n=0}^{\infty} P_n(x)\,t^n$$

$$(1-x^2)P_n''(x) - 2xP_n'(x) + n(n+1)P_n(x) = 0 \quad \text{+ initial values}$$



$$P_{30}(x) = \frac{1}{67108864}\Big(7391536347803839x^{30} - 54496920530418135x^{28} + \ldots + 105294257318225x^6 - 347123925225x^4 + 4508102925x^2 - 9694845\Big)$$

Rapid convergence of GL quadrature

GL quadrature is exact when f is a polynomial of degree at most $2n-1$, and nearly optimal when f is well-approximated by polynomials (e.g. analytic with no poles close to $[-1,1]$).

Example: $\left|\int_{-1}^{1} f(x)dx - \sum_{i=1}^{n} w_i f(x_i)\right|$

n	$f(x)=\log(2+x)$	$f(x)=\mathrm{Ai}(10x)$
12	10^{-14}	10^{-1}
24	10^{-28}	10^{-9}
48	10^{-56}	10^{-34}
96	10^{-111}	10^{-105}
192	10^{-222}	10^{-284}
384	10^{-441}	10^{-721}
768	10^{-881}	

The drawback of Gaussian quadrature

> *High-precision [Gauss-Legendre] abscissas and weights, once computed, may be stored for future use. But for truly extreme-precision calculations – i.e., several thousand digits or more – the cost of computing them even once becomes prohibitive.*

– D. H. Bailey and J. M. Borwein, *High-precision numerical integration: Progress and challenges*, J. Symb. Comp., 2011

The drawback of Gaussian quadrature

> *High-precision [Gauss-Legendre] abscissas and weights, once computed, may be stored for future use. But for truly extreme-precision calculations – i.e., several thousand digits or more – the cost of computing them even once becomes prohibitive.*

– D. H. Bailey and J. M. Borwein, *High-precision numerical integration: Progress and challenges*, J. Symb. Comp., 2011

Alternatives with simpler nodes:

- Clenshaw-Curtis – nodes $\cos(\pi k/n)$, weights by FFT ($\approx 2n$ points for same accuracy as n-point GL)
- Double exponential – nodes+weights by exponentials ($> 5n$ points for same accuracy)

What is the complexity of computing GL rules?

Can use Newton iteration from initial values $x_k \approx \cos\left(\frac{4k+3}{4n+2}\pi\right)$

$O(n)$ – number of roots
$O(n)$ – number of operations to evaluate P_n, P'_n
$O(p)$ – precision in bits

Time complexity (ignoring log factors):

- $O(n^2 p)$ – basic algorithm

What is the complexity of computing GL rules?

Can use Newton iteration from initial values $x_k \approx \cos\left(\frac{4k+3}{4n+2}\pi\right)$

$O(n)$ – number of roots
$O(n)$ – number of operations to evaluate P_n, P_n'
$O(p)$ – precision in bits

Time complexity (ignoring log factors):

- $O(n^2 p)$ – basic algorithm
- $O(n^2)$ when $p = O(1)$
  - Can be improved to $O(n)$ using asymptotic expansions. Fast `double` ($p = 53$) implementations by Hale, Townsend, Bogaert and others (without rigorous error bounds).

What is the complexity of computing GL rules?

Can use Newton iteration from initial values $x_k \approx \cos\left(\frac{4k+3}{4n+2}\pi\right)$

$O(n)$ – number of roots
$O(n)$ – number of operations to evaluate P_n, P_n'
$O(p)$ – precision in bits

Time complexity (ignoring log factors):

- $O(n^2 p)$ – basic algorithm
- $O(n^2)$ when $p = O(1)$
  - Can be improved to $O(n)$ using asymptotic expansions. Fast `double` ($p = 53$) implementations by Hale, Townsend, Bogaert and others (without rigorous error bounds).
- $O(n^3) = O(p^3)$ when $n \sim p$
  - Can be improved to $O(n^2)$ by computing all roots simultaneously, using fast multipoint evaluation.

What's new

Efficient (in practice) algorithm for evaluating $P_n(x)$ on $[-1, 1]$ and computing roots and Gauss-Legendre quadrature rules in arbitrary precision.

Implementation with rigorous error bounds in the Arb library (`http://arblib.org`) using ball arithmetic $[m \pm r]$.

Performance for computing quadrature rules:

- $O(n)$ complexity when $p = O(1)$.
- $O(n^3)$ complexity when $n \sim p$, but in practice better than fast multipoint evaluation for realistic 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 for $P_n(x)$ and $P'_n(x)$ 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 $x \in [0, 1]$

Our strategy for evaluating $(P_n(x), P_{n-1}(x))$

Three-term recurrence (n and p small, $\lesssim 1000$):

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

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):

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

Hybrid method: for each method/series, estimate

$$\text{cost} = (\text{number of terms}) \cdot (\text{working precision}),$$

choose method with lowest cost.

Stability of the three-term recurrence

Example: use $(n+1)P_{n+1}(x) = (2n+1)xP_n(x) - nP_{n-1}(x)$ starting from $P_0(x) = 1, P_1(x) = x$ to evaluate $P_n(0.40625)$, in:

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

Stability of the three-term recurrence

Example: use $(n+1)P_{n+1}(x) = (2n+1)xP_n(x) - nP_{n-1}(x)$ starting from $P_0(x) = 1, P_1(x) = x$ to evaluate $P_n(0.40625)$, in:

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

n	Error in 53-bit FP	Result in 53-bit ball arithmetic
10	$6 \cdot 10^{-18}$	`[0.244683436384045 +/- 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:

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

Approximate version:

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

We can show:

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

This permits a fast, rigorous implementation with `mpz_t` fixed-point arithmetic, using only $O(\log n)$ extra precision.

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

For x close to 1:

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

For x close to 0 (also, for general x 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

Hypergeometric series expansions

For x close to 1:

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

For x close to 0 (also, for general x 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 (extra working precision) via:

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

$$|P_n(z)| \leq |P_n(i|z|)| \leq \left(|z| + \sqrt{1+|z|^2}\right)^n$$

(Here we really need $O(n)$ extra bits of precision.)

Fast evaluation of hypergeometric series

We evaluate hypergeometric series

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

using rectangular splitting

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

with $m \sim \sqrt{N}$, costing $O(\sqrt{N})$ expensive + $O(N)$ cheap ops.

- Exploit c_k/c_{k-1} to get small coefficients (Smith, 1989)
- Collect denominators to skip most divisions
- For P_n, P_{n-1} or P_n, P_n' simultaneously: recycle $x^2, \ldots, x^m$
- Implementation using ball arithmetic for error bounds

Asymptotic expansion

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

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

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

Asymptotic expansion

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

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

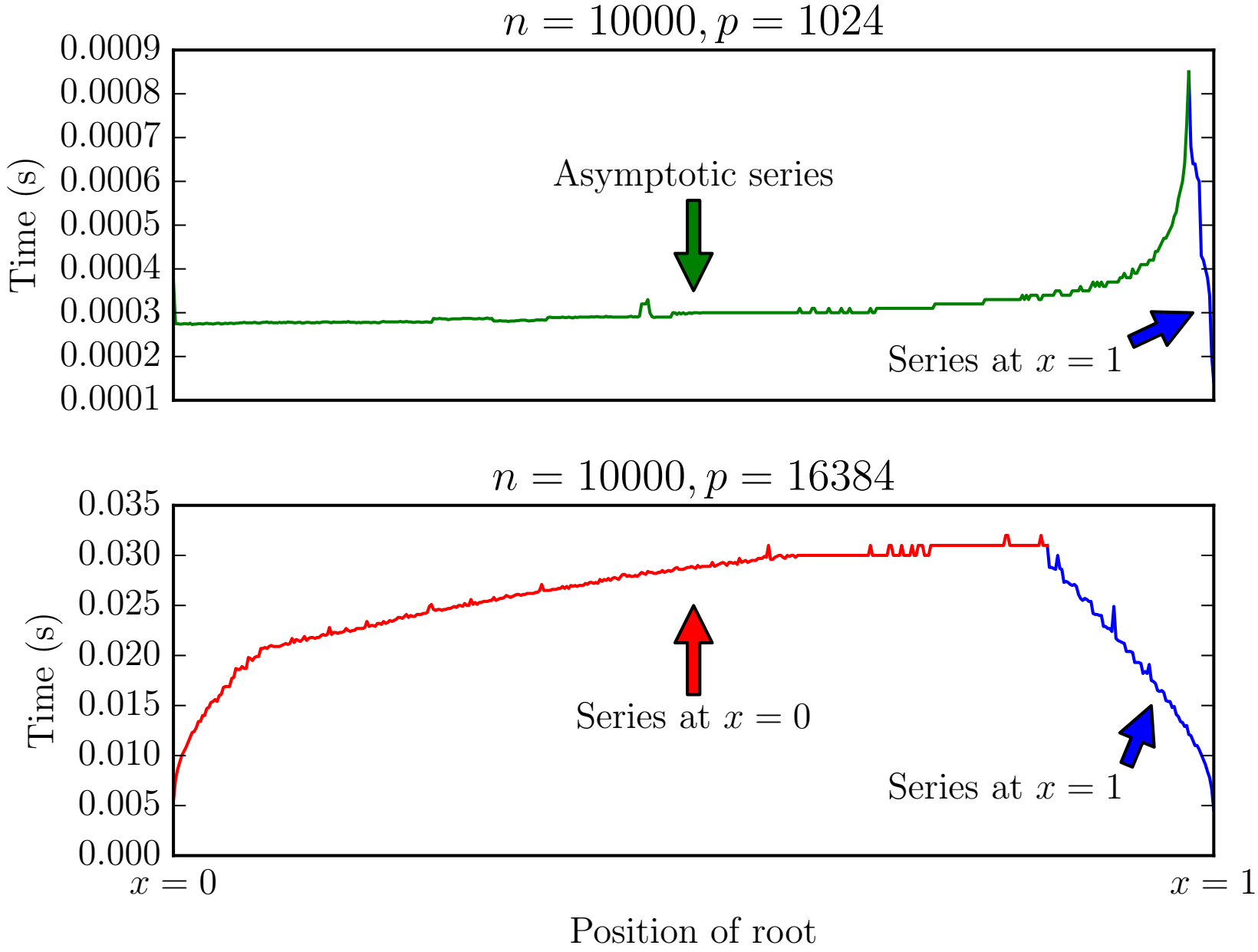
$$C_{n,k} = \frac{[\Gamma(k+\frac{1}{2})]^2 \Gamma(n+1)}{\pi 2^k \Gamma(n+k+\frac{3}{2})\Gamma(k+1)}, \quad |\xi_{n,K}(\theta)| < \sqrt{\frac{8}{\pi \sin(\theta)}} \frac{C_{n,K}}{\sin^K(\theta)}$$

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

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

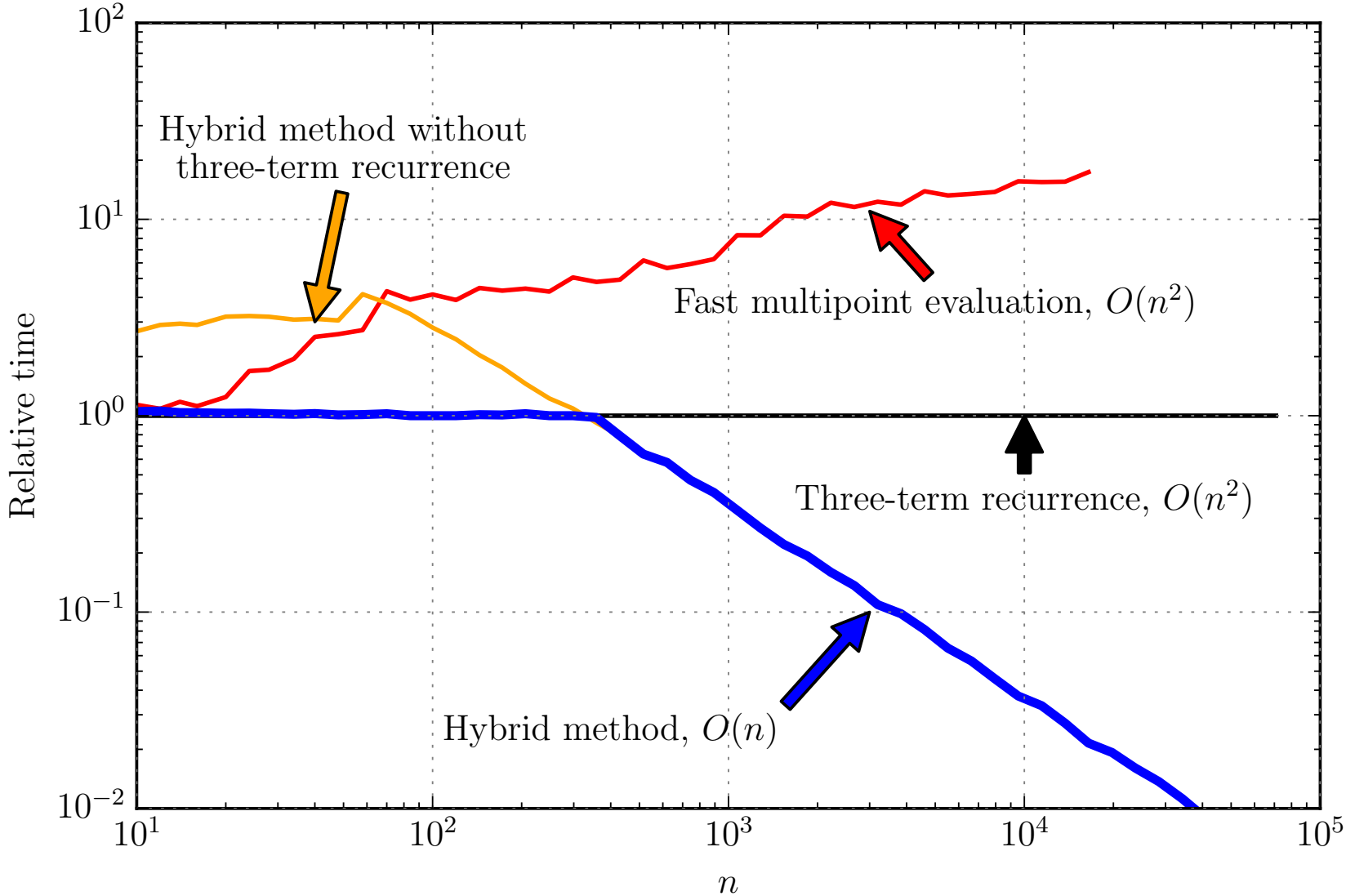
By working with complex numbers, the sum becomes a pure hypergeometric series and rectangular splitting can be used.

Time to evaluate (P_n, P'_n) for varying x



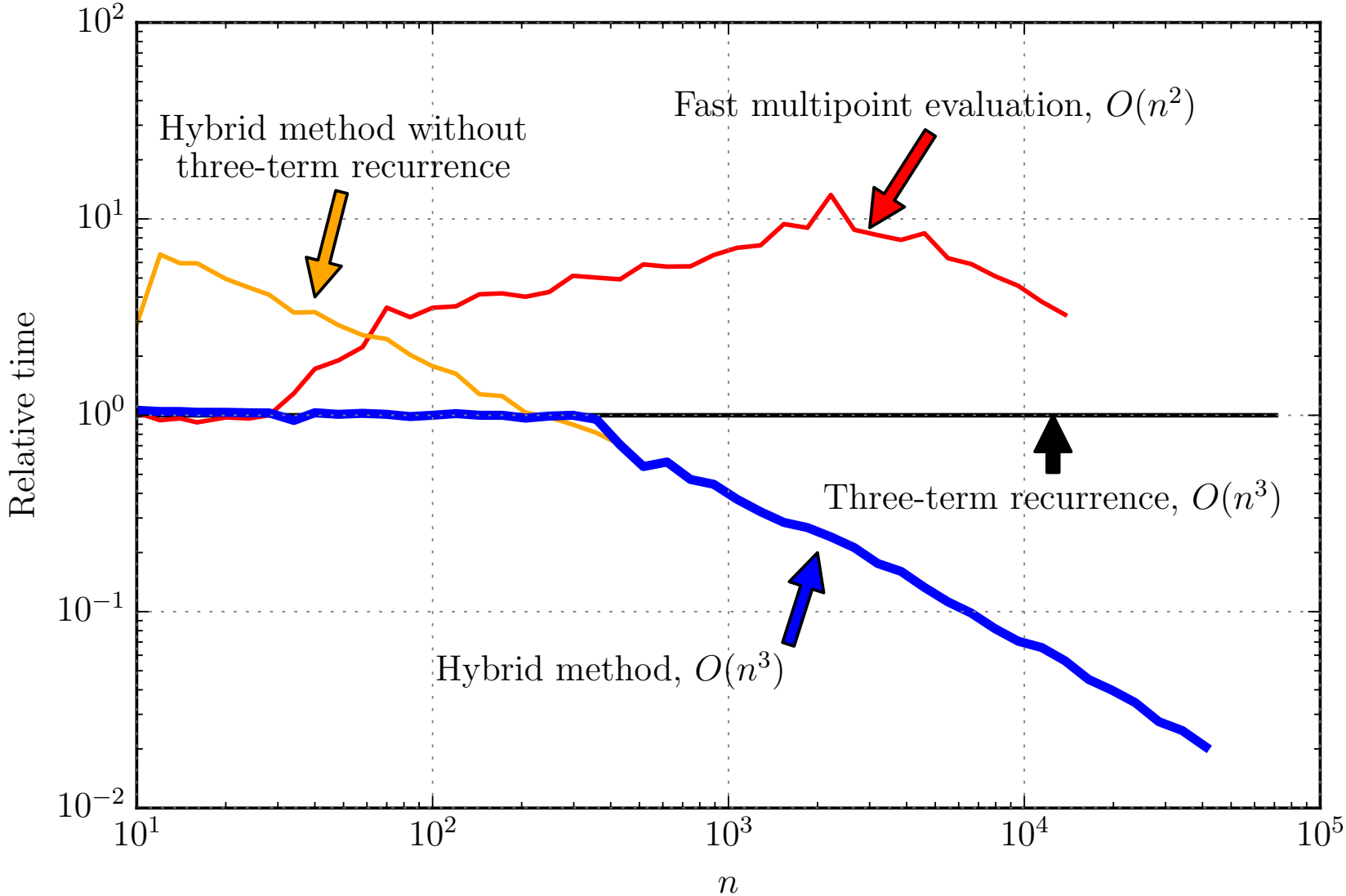
Time to evaluate (P_n, P_n') at $n/2$ points

Constant precision, $p = 64$



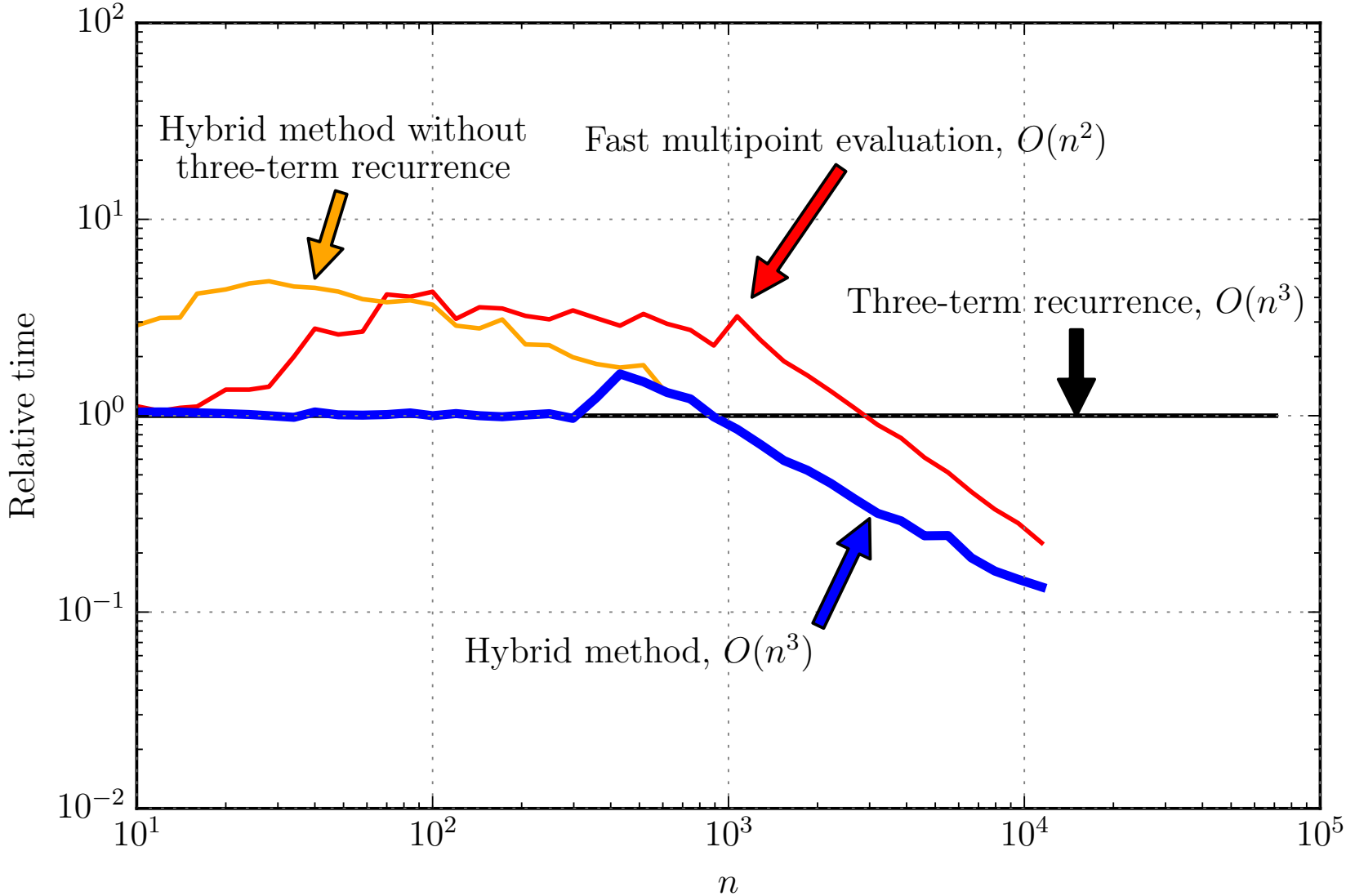
Time to evaluate (P_n, P'_n) at $n/2$ points

Increasing precision, $p = n/10$



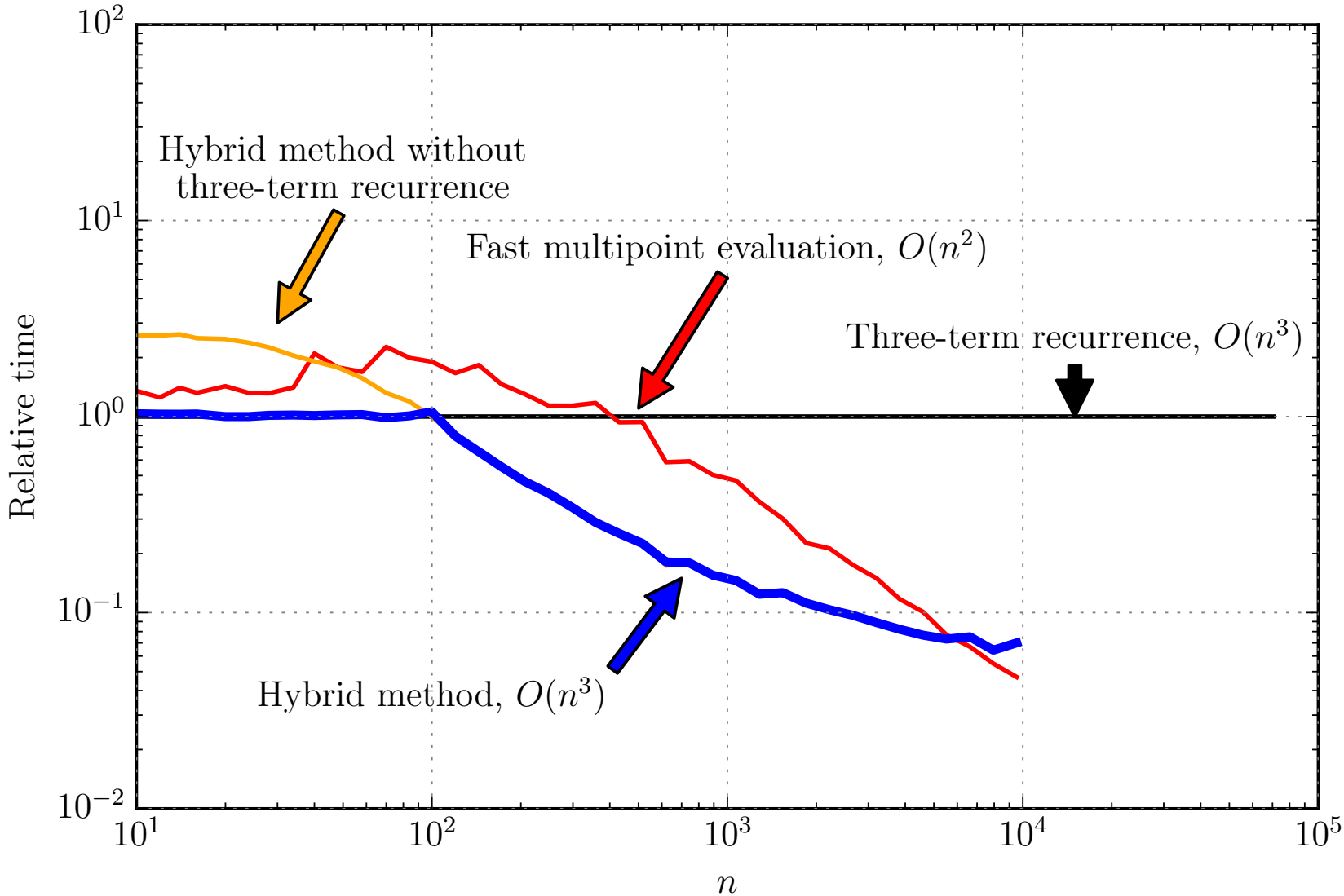
Time to evaluate (P_n, P'_n) at $n/2$ points

Increasing precision, $p = n$



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

Increasing precision, $p = 10n$



Time to generate Gauss-Legendre quadrature rules

$n \setminus p$	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.

Time to generate Gauss-Legendre quadrature rules

$n \setminus p$	64	256	1024	3333	33333
20	3.9	2.3	2.3	2.3	3.6
50	8.0	4.2	3.9	3.6	5.6
100	11	6.0	5.0	4.7	7.5
200	17	8.9	7.8	7.0	10
500	71	25	14	13	21
1000	383	150	46	29	31
2000	4680	1320	395	118	55
5000					
10000					
100000					
1000000					

Speedup compared to Pari/GP `intnumgaussinit`.

Another comparison: 1000-digit quadrature

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 integration with up to 1000 digit accuracy.

n	ARPREC (s)	Our code (s)	Speedup
12	0.00520	0.000735	7.1
24	0.0189	0.00197	9.6
48	0.0629	0.00574	11.0
96	0.251	0.0185	13.6
192	0.974	0.0611	16.0
384	3.83	0.231	16.6
768	15.2	0.875	17.4
1536	60.9	3.03	20.0
3072	241	9.75	24.7
6144	1013	18.4	55.0

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

Gauss-Legendre is competitive for ≈ 10 integrals, or single integral when integrand costs ≈ 10 elementary functions

New rigorous integration code in Arb

Adaptive subdivision + degree-adaptive Gauss-Legendre quadrature + error bounds based on complex magnitudes (Petras algorithm)

Gauss-Legendre quadrature rules are automatically cached

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

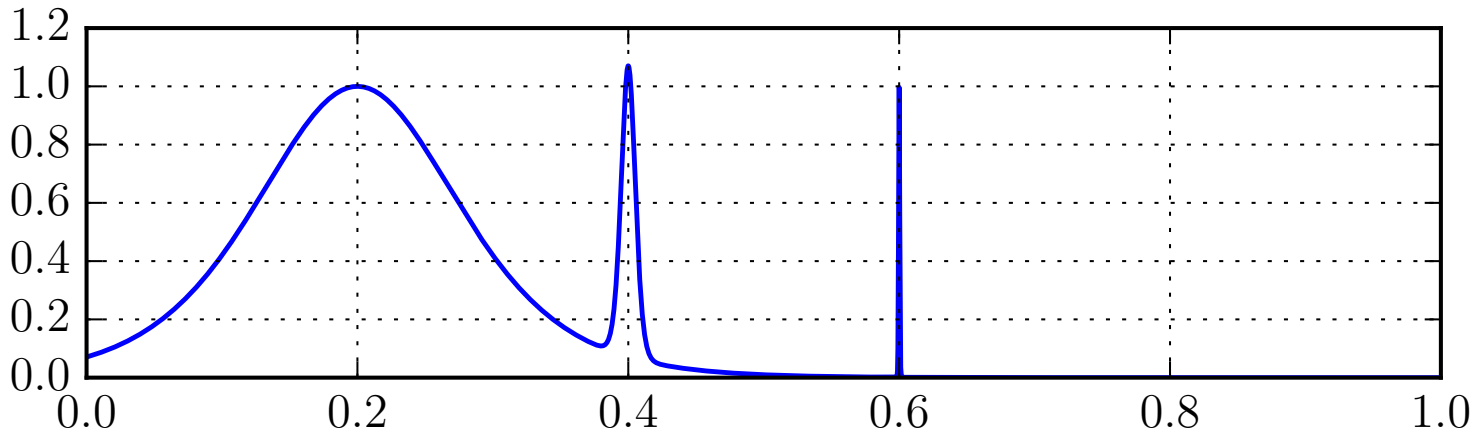
Blog post: `http://fredrikj.net/blog/2017/11/new-rigorous-numerical-integration-in-arb/`

Integration examples

An example from the Mathematica documentation:

$$\int_0^1 \operatorname{sech}^2(10(x-0.2)) + \operatorname{sech}^4(100(x-0.4)) + \operatorname{sech}^6(1000(x-0.6))\,dx = 0.2108027\ldots$$

With default settings, many numerical integrators (Mathematica, Sage, SciPy, mpmath, Pari/GP, ...) return an incorrect estimate!



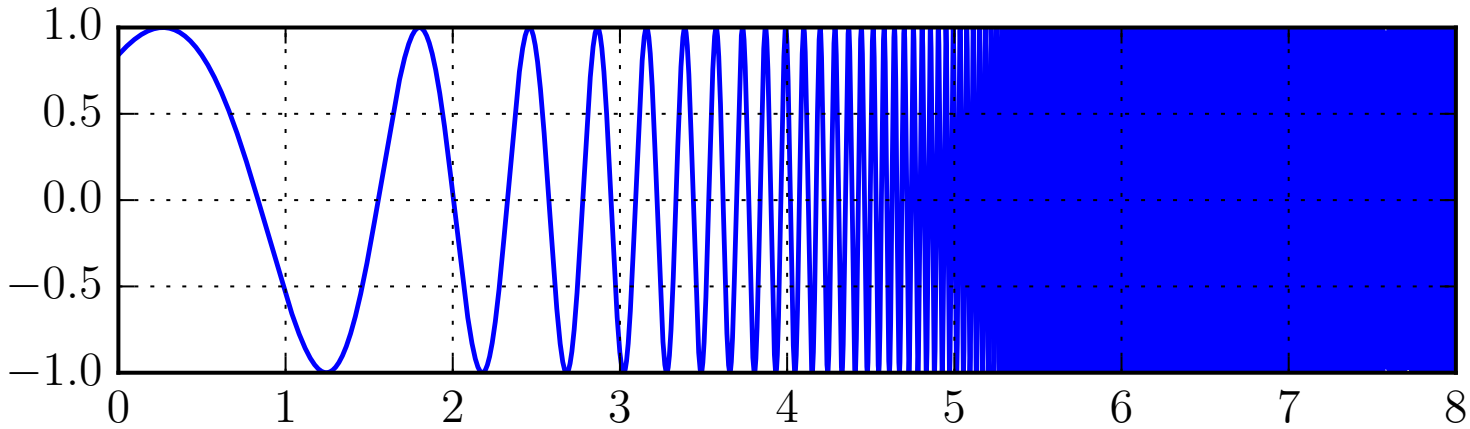
Arb with 100 digits: 0.04 s (+ 0.02 s nodes computation)
Arb with 1000 digits: 8.7 s (+ 2.3 s nodes computation)

mpmath with 100 digits: 0.7 s (must split domain manually)
mpmath with 1000 digits: 32 s (must split domain manually)

Integration examples

Rump's oscillatory example

$$\int_0^8 \sin(x + e^x) = 0.347400172657\ldots$$



Arb with 100 digits: 0.02 s (+ 0.01 s nodes computation)
Arb with 1000 digits: 1.2 s (+ 4.1 s nodes computation)

mpmath with 100 digits: 0.6 s (must increase degree manually)
mpmath with 1000 digits: 12 s
Pari/GP with 100 digits: 0.2 s (must split domain manually)
Pari/GP with 1000 digits: 14 s (must split domain manually)

Conclusion

- Error analysis for three-term recurrence + fast hybrid algorithm in ball arithmetic to evaluate Legendre polynomials on $[-1, 1]$ for any combination of n, p.
- Order-of-magnitude speedup for computing high-precision Gauss-Legendre quadrature rules. GL quadrature becomes practical even at 10^3 or 10^4 digits.
- We also tested fast multipoint evaluation and found that our algorithm performs better in practice.
- Extension to other Gaussian quadrature rules

$$\int_a^b w(x) f(x) dx \approx \sum_{i=1}^{n} w_i f(x_i)$$

(Jacobi, Laguerre, Hermite, ... polynomials) would be useful and could be done using similar techniques.