Real numbers: a computational perspective

Fredrik Johansson

Inria

Fifteenth Algorithmic Number Theory Symposium (ANTS-XV)
University of Bristol
August 8 - 12, 2022

Question

Can we make computation over the fields $\mathbf{R}$ and $\mathbf{C}$ as easy and reliable as computation over, say, $\mathbf{Q}$ or $\mathbf{F}_q$?

Hint: there will not be a single, universal, optimal solution!

Three levels of real arithmetic

1. **Heuristic** (approximations)

$$\sin(\pi) \approx 1.2246 \cdot 10^{-16}$$

2. **Rigorous** (enclosures)

$$\sin(\pi) \in [0 \pm 3.46 \cdot 10^{-16}]$$

3. **Exact** (symbolic and algebraic expressions)

$$\sin(\pi) = 0$$

Three levels of real arithmetic

1. **Heuristic** (approximations)

$$\sin(\pi) \approx 1.2246 \cdot 10^{-16}$$

2. **Rigorous** (enclosures)

$$\sin(\pi) \in [0 \pm 3.46 \cdot 10^{-16}]$$

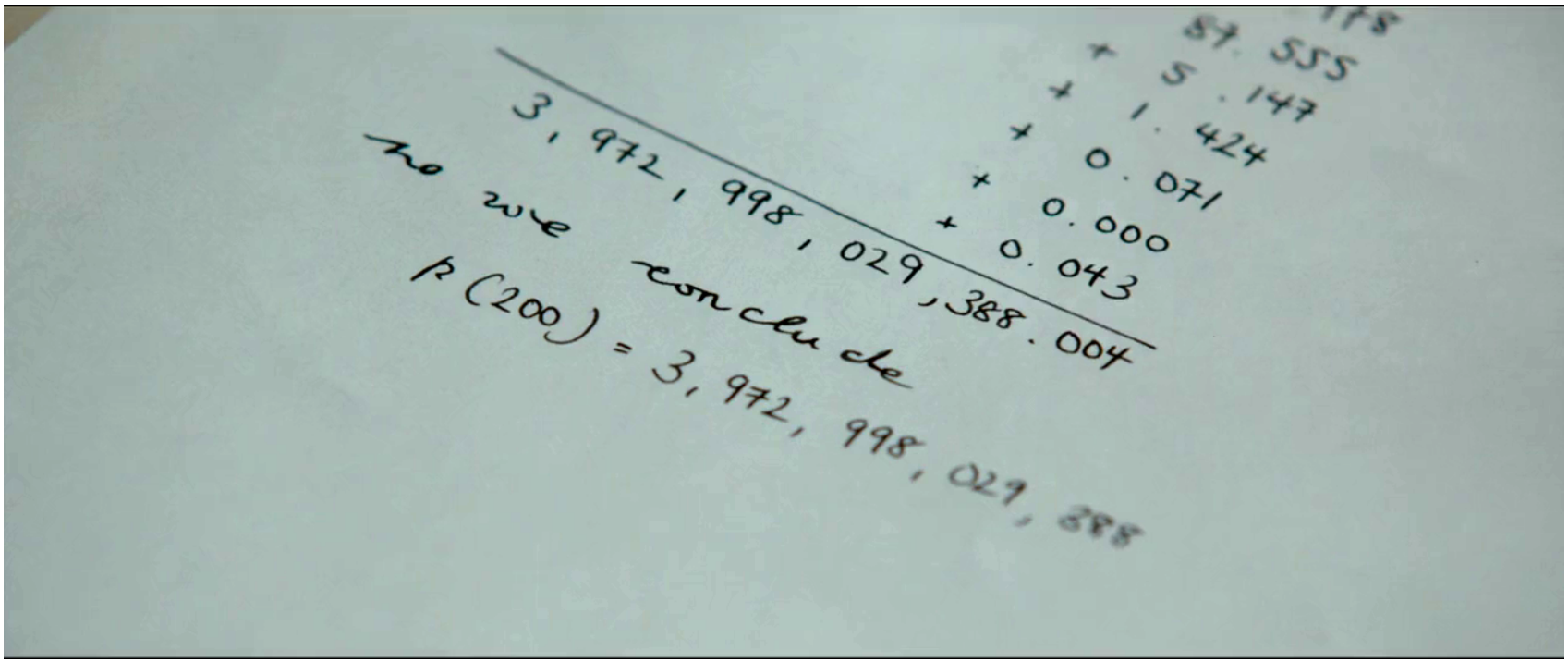
3. **Exact** (symbolic and algebraic expressions)

$$\sin(\pi) = 0$$

Example software:

- SageMath: 1. `RealField`, 2. `RealBallField`, 3. `QQbar`, `SymbolicRing`
- My libraries: 1. mpmath, 2. Arb, 3. Calcium

Example: computing the partition function $p(n)$



Scene from: Brown, *The Man Who Knew Infinity*, 2015

Example: computing the partition function $p(n)$

THE ON–LINE ENCYCLOPEDIA OF INTEGER SEQUENCES ®

founded in 1964 by N. J. A. Sloane

A110375 Numbers n such that Maple 9.5, Maple 10, Maple 11 and Maple 12 give the wrong answers for the number of partitions of n. [5]

11269, 11566, 12376, 12430, 12700, 12754, 15013, 17589, 17797, 18181, 18421, 18453, 18549, 18597, 18885, 18949, 18997, 20865, 21531, 21721, 21963, 22683, 23421, 23457, 23547, 23691, 23729, 23853, 24015, 24087, 24231, 24339, 24519, 24591, 24627, 24681, 24825, 24933, 25005, 25023, 25059, 25185, 25293, 27020 (list; graph; refs; listen; history; text; internal format)

OFFSET 1,1

COMMENTS Based on various postings on the Web, sent to N. J. A. Sloane by R. J. Mathar. Thanks to several correspondents who sent information about other versions of Maple.
Mathematica 6.0, DrScheme and pari-2.3.3 all give the correct answers.
Ramanujan's congruence says that numbpart(5*k+4)==0 mod 5, so numbpart(11269)=...851==1 mod 5 can't be correct. [Robert Gerbicz, May 13 2008]

LINKS Table of n, a(n) for n=1..44.
Author?, Concerning this sequence

EXAMPLE From PARI, the correct answer:
numbpart(11269)
231139177231303975514411787649455628959060199360109972557851519105155176 1\
803182158917958749053182741632480330718 50
From Maple 11, incorrect:
combinat[numbpart](11269);
231139177231303975514411787649455628959060199360109972557851519105155176 1\
803182158917958749053182741632480330718 51
On the other hand, the old Maple 6 gives the correct answer.

Example: "exact" symbolic arithmetic

$$a = 2\log(\sqrt{2} + \sqrt{3}) - \log(5 + 2\sqrt{6})$$

Example: "exact" symbolic arithmetic

$$a = 2\log(\sqrt{2}+\sqrt{3}) - \log(5+2\sqrt{6}) \qquad (a = 0)$$

Example: "exact" symbolic arithmetic

$$a = 2\log(\sqrt{2}+\sqrt{3}) - \log(5+2\sqrt{6}) \qquad (a = 0)$$

$$A = \begin{pmatrix} 0 & a \\ 0 & 0 \end{pmatrix} \qquad B = \begin{pmatrix} 0 & a + e^{-1000} \\ 0 & 0 \end{pmatrix}$$

Example: "exact" symbolic arithmetic

$$a = 2\log(\sqrt{2}+\sqrt{3}) - \log(5+2\sqrt{6}) \qquad (a = 0)$$

$$A = \begin{pmatrix} 0 & a \\ 0 & 0 \end{pmatrix} \qquad B = \begin{pmatrix} 0 & a + e^{-1000} \\ 0 & 0 \end{pmatrix}$$

Maple 2020, SageMath 9.6's `SymbolicRing`: $\operatorname{rank}(A) = 1$

Mathematica 12.2: $\operatorname{rank}(B) = 0$

Example: "exact" symbolic arithmetic

$$a = 2\log(\sqrt{2}+\sqrt{3}) - \log(5+2\sqrt{6}) \qquad (a = 0)$$

$$A = \begin{pmatrix} 0 & a \\ 0 & 0 \end{pmatrix} \qquad B = \begin{pmatrix} 0 & a + e^{-1000} \\ 0 & 0 \end{pmatrix}$$

Maple 2020, SageMath 9.6's `SymbolicRing`: $\operatorname{rank}(A) = 1$

Mathematica 12.2: $\operatorname{rank}(B) = 0$

Calcium:

```
>>> a = 2*log(sqrt(2)+sqrt(3)) - log(5+2*sqrt(6))
>>> A = ca_mat([[0,a],[0,0]]); A.rank()
0
>>> B = ca_mat([[0,a+exp(-1000)],[0,0]]); B.rank()
1
```

Wishful thinking in real computation

Fallacy 1:

$$\mathbf{Q}(a, b, c) \cong \mathbf{Q}(X, Y, Z).$$

Counterexamples:

- $\mathbf{Q}(\sqrt{2}, \sqrt{3}, \sqrt{6}) \cong \mathbf{Q}(X, Y, Z)/\langle X^2 - 2, Y^2 - 3, XY - Z\rangle$[1]
- $\mathbf{Q}(\log(\sqrt{2} + \sqrt{3}), \log(5 + 2\sqrt{6})) \cong \mathbf{Q}(X, Y)/\langle 2X - Y\rangle$

- However $\mathbf{Q}(\pi) \cong \mathbf{Q}(X)$
- Is $\mathbf{Q}(\pi, e) \cong \mathbf{Q}(X, Y)$?

[1] Read $\mathbf{Q}(X)/I$ as $\mathrm{Frac}(\mathbf{Q}[X]/I)$

Wishful thinking in real computation

Fallacy 2:

$$(|x - y| < \varepsilon) \implies x = y.$$

Counterexamples:

- $0 + \exp(-1000) \neq 0$
- $\dfrac{(\exp(\pi\sqrt{163}) - 744)^{1/3}}{640320} \neq 1 \quad (\varepsilon < 10^{-30})$
- $\displaystyle\int_0^\infty \cos(2x) \prod_{n=1}^{\infty} \cos(x/n)dx \neq \frac{\pi}{8} \quad (\varepsilon < 10^{-40})$

OK if we have a rigorous lower separation bound (e.g. Mahler-Mignotte for $x, y \in \overline{\mathbf{Q}}$).

Wishful thinking in real computation

Fallacy 3:

"I implemented the algorithm from the book and the code compiles. Therefore it must be correct."

Meta-algorithm to decide whether $x = 0$

1. Use **rigorous numerical enclosures** to attempt to prove $x \neq 0$.
   (If a separation bound is available, we can also prove $x = 0$ here.)

2. Express x in a field $K = \mathbf{Q}(\alpha_1, \ldots, \alpha_n)$.

   Guess (e.g. with LLL) an ideal I of relations on $\alpha_1, \ldots, \alpha_n$.
   Prove the relations (recursing into simpler fields).
   Reduce (using Gröbner bases) x in $K' = \mathbf{Q}(X_1, \ldots, X_n)/I$.

   If $x = 0$ in K', we have proved that $x = 0$ in K.
   (If we can show that $K \cong K'$, we can also prove $x \neq 0$ here.)

3. Repeat with increased precision.

Meta-algorithm to decide whether $x = 0$

1. Use **rigorous numerical enclosures** to attempt to prove $x \neq 0$.
   (If a separation bound is available, we can also prove $x = 0$ here.)

2. Express x in a field $K = \mathbf{Q}(\alpha_1, \ldots, \alpha_n)$.

   Guess (e.g. with LLL) an ideal I of relations on $\alpha_1, \ldots, \alpha_n$.
   Prove the relations (recursing into simpler fields).
   Reduce (using Gröbner bases) x in $K' = \mathbf{Q}(X_1, \ldots, X_n)/I$.

   If $x = 0$ in K', we have proved that $x = 0$ in K.
   (If we can show that $K \cong K'$, we can also prove $x \neq 0$ here.)

3. Repeat with increased precision.

Theorem (Richardson): assuming Schanuel's conjecture, a version of this algorithm always terminates for *elementary numbers* (complex numbers constructed from $\mathbf{Q}$ via field operations, exp, and log).

Several practical issues

- Eager or lazy algebraic evaluation and simplification?
- How to choose generators of $\mathbf{Q}(\alpha_1, \ldots, \alpha_n)$? How to handle (or avoid) extensions of high degree / large coefficients?
  - Example: $\mathbf{Q}(\sqrt{2}, \sqrt{3})$ vs $\mathbf{Q}(\sqrt{2} + \sqrt{3})$
  - Example: $\mathbf{Q}(\cos(x), \sin(x))$ vs $\mathbf{Q}(\tan(x/2))$ vs $\mathbf{Q}(i, \exp(ix))$
  - Example: $\mathbf{Q}((\pi + 1)^{1000})$ vs $\mathbf{Q}(\pi)$
- How to handle very large, small or close numbers numerically?
  - Example: $\exp(\exp(\exp(-100))) = 1$?
  - Formal series expansions, level-index arithmetic
- Simplification of extensions (e.g. $\exp(\log(x)/2) \to \sqrt{x}$)
- Multivariate fraction field arithmetic, Gröbner bases

Sample benchmark: exact algebraic computation

Time to prove $\mathbf{x} - \mathrm{DFT}^{-1}(\mathrm{DFT}(\mathbf{x})) = \mathbf{0}$ where $\mathbf{x} = (x_n)_{n=0}^{N-1}$

x_{n-2}	N	Sage `QQbar`	Sage `SR`	SymPy	Maple	Mathematica	**Calcium**
n	8	0.018	0.11	1.1	0.0060	0.057	0.00016
	20	0.14	172	fail	0.13	0.96	0.00045
	100	8.2	fail	fail	9.1	> 60	0.044
$\sqrt{n}$	20	$> 10^3$	208	fail	1.1	2.3	0.064
	100	$> 10^3$	fail	fail	$> 10^3$	> 60	17
$\log(n)$	20	-	188	fail	0.74	45	0.043
	100	-	fail	fail	$> 10^3$	> 60	26
$e^{2\pi i/n}$	20	$> 10^3$	329	fail	fail	> 60	0.24
	100	$> 10^3$	fail	fail	$> 10^3$	> 60	86*
$\frac{1}{1+n\pi}$	20	-	219	fail	2.4	> 60	0.12
	100	-	fail	fail	$> 10^3$	> 60	202
$\frac{1}{1+\sqrt{n}\pi}$	8	-	0.76	22	0.074	2.6	0.072
	20	-	fail	fail	$> 10^3$	> 60	62

Numerical algebraic computation over $\mathbf{R}$ and $\mathbf{C}$

Computations in domains like $\mathbf{R}[x]$ and in $\mathbf{R}^{n \times n}$ *usually* work quite well in interval arithmetic.

We have quite good kernel operations (e.g. polynomial and matrix multiplication) in both high precision[2] and low/medium precision[3].

Problems:

- Numerical instability in higher operations (e.g. inversion, multipoint evaluation) often results in loss of $O(n)$ digits.

  Throw more precision at it or find a better algorithm?
- Handling ill-posed problems (e.g. $\operatorname{rank}(A)$) without falling back on exact computation.

[2] J. *Faster arbitrary-precision dot product and matrix multiplication*, 2019

[3] Example: van der Hoeven & Lecerf, *Faster FFTs in medium precision*, 2014

Certification & mixed-precision methods

Useful principle:

$$\textit{approximate computation} + \textit{posteriori certification}$$

often performs better than a direct computation in interval arithmetic.

Example: solving $Ax = b$ in 53-bit precision where A is a well-conditioned real $n \times n$ matrix.

```
n    FP Gauss           Interval Gauss                FP + posteriori cert.
-------------------------------------------------------------------------
     x_1 =              x_1 =                         x_1 =
3    1.69270534084004   [1.69270534084004 ± 7e-15]    [1.69270534084004 ± 7e-15]
10   2.93381472087850   [2.93381472088 ± 1e-11]       [2.93381472087850 ± 5e-15]
30   4.98358363678481   [5.0 ± 0.05]                  [4.9835836367848 ± 2e-14]
100  9.03226736256870   [± inf]                       [9.0322673625687 ± 3e-14]
```

Certification & mixed-precision methods

Mixed-precision arithmetic: start the computation in low precision and finish in higher precision, e.g.

$$\text{16-bit} \to \text{32-bit} \to \text{64-bit}$$
$$\text{64-bit} \to \text{128-bit} \to \text{arbitrary-precision}$$

This is classical e.g. in polynomial root-finding, and nowadays all the rage in HPC linear algebra.

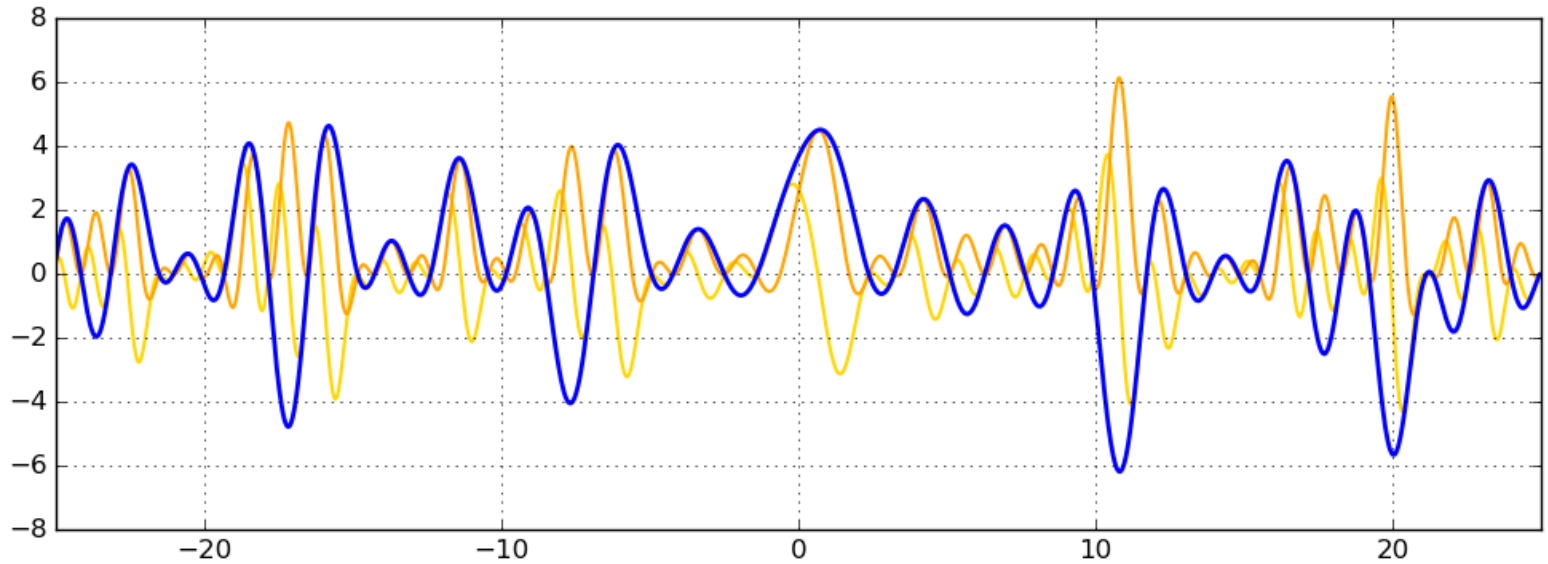
A recent application in number theory: computing modular forms on noncongruence subgroups.

$$\text{Linear system} \to \text{Numerical approximations of Fourier coefficients} \to \text{LLL} \to \text{Algebraic expressions}$$

Replacing *direct arbitrary-precision solving* with *iterative solving using a machine-precision preconditioner* yields a 100× speedup.[4]

[4] Berghaus, Monien & Radchenko, *On the computation of modular forms on noncongruence subgroups*, 2022

Real and complex analysis



How can we compute things like $f'(x)$, $\int_a^b f(x)dx$ or $\sum_{n=0}^{\infty} f(n)$?

How to represent a function?

General functions

- Black box evaluation: $z \mapsto f(z)$
- Taylor/Chebyshev/Fourier models: $f(z) = 1 + z + 0.5z^2 + \varepsilon(z)$
- Generalized series expansions:

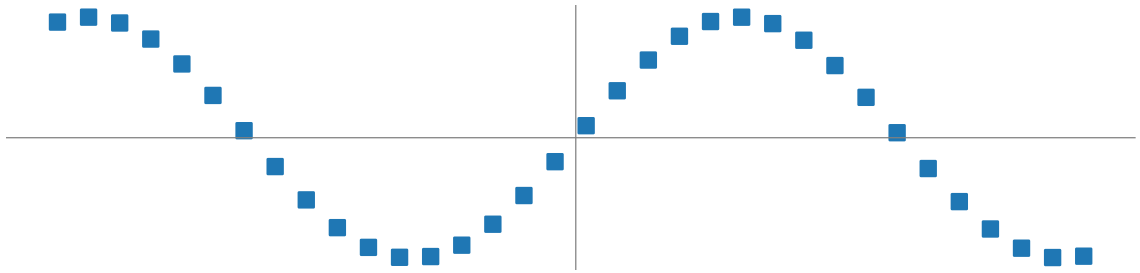
$$f(z) = \sqrt{2\pi z}\left(\frac{z}{e}\right)^z \left(1 + \frac{1}{12z} + \varepsilon(z)\right)$$

Special functions

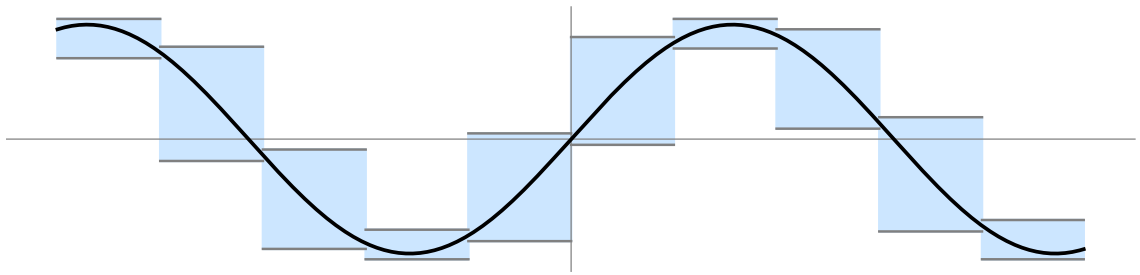
- Functions with exact descriptions (for example, using *differential, difference, functional or algebraic equations* + initial conditions)
- Potentially much more efficient than general methods
- Can have exact descriptions of singularities, special values

Computing in the black box model

Point sampling:

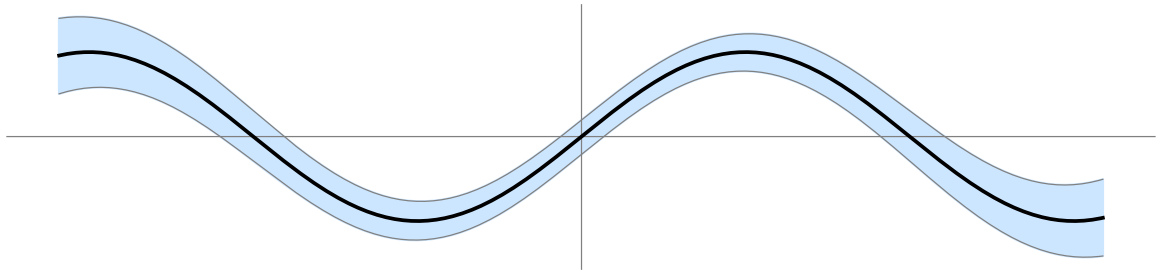


Enclosures:



Interval subdivision is powerful, but can be exponentially slow. Point sampling with error bounds (e.g. based on $f^{(n)}$) is usually better.

Polynomial approximants/models



Advantages:

- Manipulating polynomials is cheap
- High order ("spectral") convergence
- Reduces overestimation for enclosures

Often used *ad hoc*. Example: the Booker-Molin method for computing L-functions (using Fourier series).

Can also be a basis for general-purpose "computer analysis systems", heuristic (e.g. Chebyshev approximants in Chebfun[5]) or rigorous (e.g. Taylor models in CoqInterval[6]).

[5] Trefethen et al.

[6] Melquiond et al.

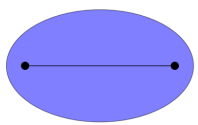
Black box analysis with analytic functions

In Holomorphic World, we can get good bounds for point sampling *using only black box enclosures.*

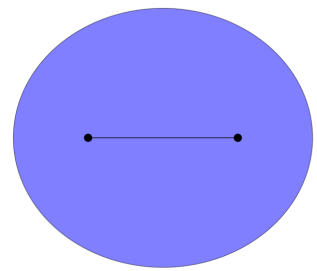
This even works for piecewise functions like $\lfloor \exp(x) \rfloor$ if the black box can check "$f(z)$ is analytic on the given complex interval for z".

Example: integration using adaptive Gauss-Legendre + subdivision.

$$\left| \int_a^b f(x)dx - \sum_{k=1}^{n} w_k f(x_k) \right| \leq \frac{M}{\rho^{2n}} \cdot |b-a| C_\rho, \quad |f(z)| \leq M$$



$\rho = 2.00$



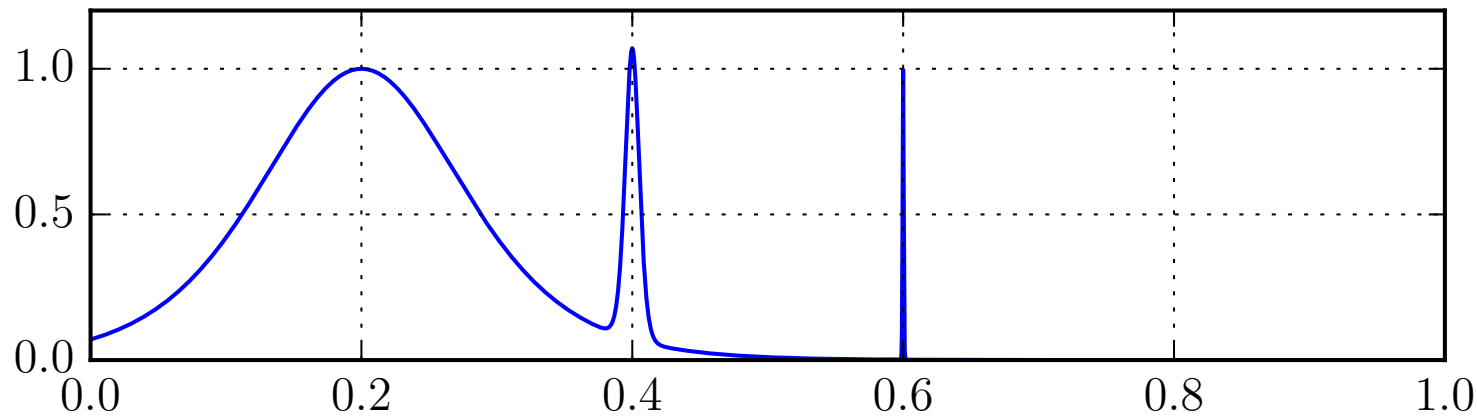
$\rho = 3.73$

If there are singularities too close to $[a, b]$, bisect, falling back to a naive enclosure $(b-a)f([a, b])$ when $[a, b]$ is narrow.[7]

[7] Petras, 1998 and 2002; J., 2018

Example: the spike integral

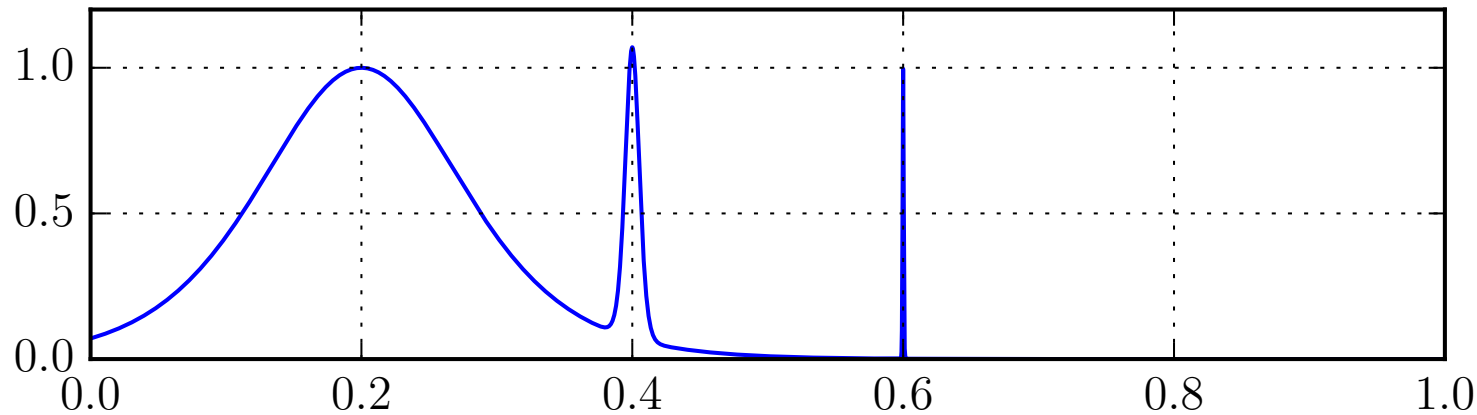
$$\int_0^1 \operatorname{sech}^2(10(x-0.2)) + \operatorname{sech}^4(100(x-0.4)) + \operatorname{sech}^6(1000(x-0.6))\ dx$$



Mathematica `NIntegrate`:	0.209736
Octave `quad`:	0.209736, error estimate 10^{-9}
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

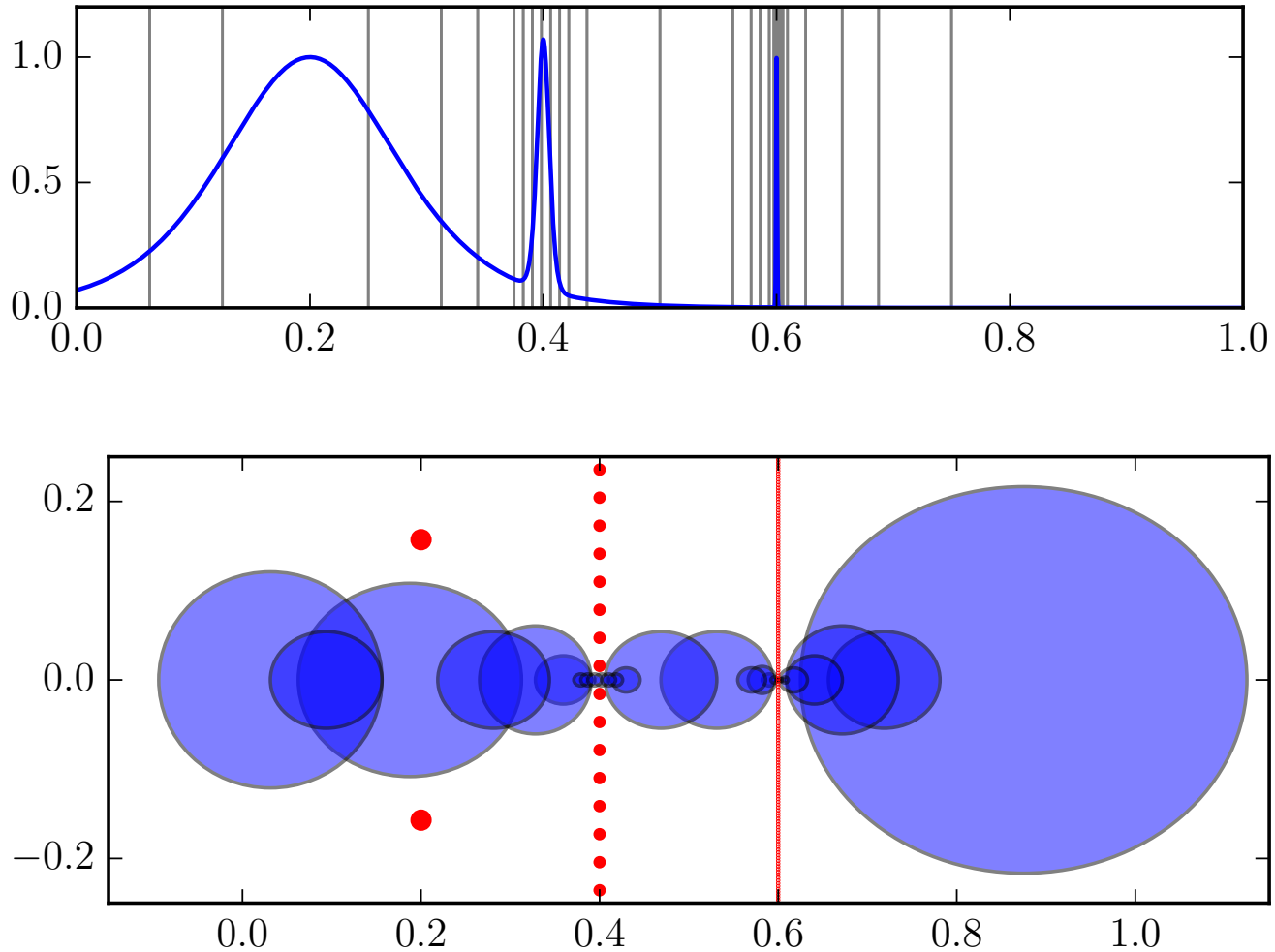
Example: the spike integral

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



Mathematica `NIntegrate`:	0.209736
Octave `quad`:	0.209736, error estimate 10^{-9}
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
Chebfun:	0.210803
Arb (rigorous):	0.210803

Example: the spike integral



On my laptop, Arb computes 1000 digits in 0.5 seconds (using 8 cores), evaluating the integrand 30,000 times.

Example: the spike integral

If the integral is written as

$$\int_0^1 \left(\frac{1}{\cosh(10(x-0.2))}\right)^2 + \left(\frac{1}{\cosh(100(x-0.4))}\right)^4 + \left(\frac{1}{\cosh(1000(x-0.6))}\right)^6 dx$$

Arb takes 32 seconds, performing 2 million evaluations.

In the form

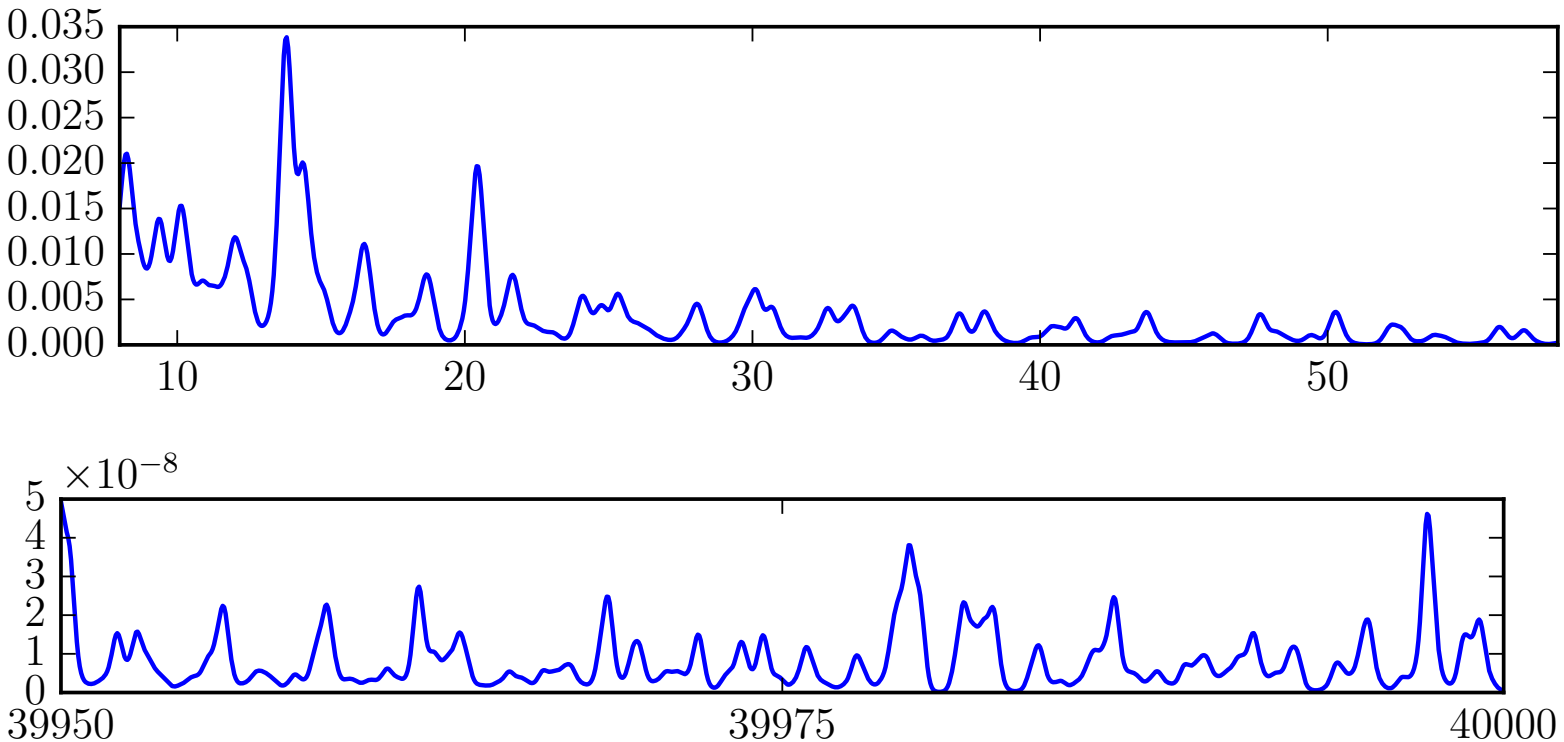
$$\int_0^1 \frac{1}{\cosh^2(10(x-0.2))} + \frac{1}{\cosh^4(100(x-0.4))} + \frac{1}{\cosh^6(1000(x-0.6))} dx$$

Arb takes 260 seconds, performing 17 million evaluations.

This kind of sensitivity is a major drawback of interval arithmetic. Symbolic preprocessing would be very useful.

Example: $|\zeta(s)|$-integrals (from Harald Helfgott)

$$\int_{-1/4+8i}^{-1/4+40000i} \left| \frac{F_{19}(s+1/2)F_{19}(s+1)}{s^2} \right| |ds|, \quad F_N(s) = \zeta(s) \prod_{p \le N} (1-p^{-s})$$



We compute Taylor models $f(s) = g(s) + h(s)i + \varepsilon$ on subintervals $[a, a + 0.5]$, and integrate $\sqrt{g^2(s) + h^2(s)}$. Time: a few hours.

Singularity analysis: improper integrals

To compute a finite limit where something goes to ∞, for example,

$$\int_0^\infty f(t)e^{-t}dt, \quad \int_0^1 \frac{f(t)}{\sqrt{t}}dt,$$

we need to handle the singular behavior.

Options:

- Special-purpose methods (e.g. Gauss-Jacobi quadrature)
- Domain truncation
- Change of variables or integration path
- Double exponential (DE) integration

Good error bounds exist when $f(t)$ is reasonably regular, but must be implemented manually for each integral.

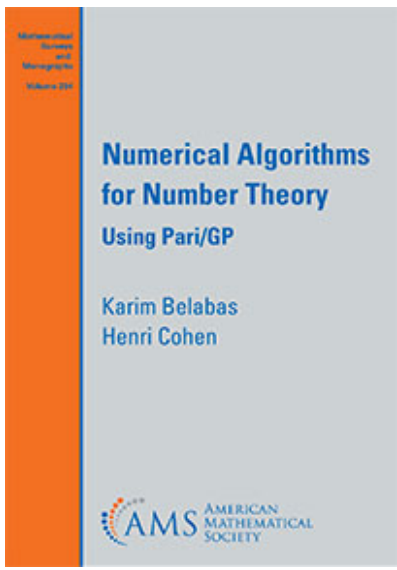
In many cases, this kind of analysis could be automated using symbolic computation + generalized series expansions.

Numerical limits and summation

How to compute slowly convergent series like $\sum_{n=0}^{\infty}(-1)^n/(2n+1)$?

```
? sum(n=0, 10^6, (-1)^n/(2*n+1.)) / (Pi/4)
%1 = 1.000000318309567874143220082156300751 2
? sumalt(n=0,(-1)^n/(2*n+1.)) / (Pi/4)
%2 = 1.000000000000000000000000000000000000
```

For an excellent review of convergence acceleration techniques, see →



- Euler-Maclaurin formula
- CVZ method
- Monien summation
- Lagrange extrapolation
- ...

Rigorous error bounds: similar difficulty to improper integrals.

Computing functions using integral representations

Many special functions admit elementary definite integral representations. Random example:

$$\Phi(z, s, a) = \sum_{n=0}^{\infty} \frac{z^n}{(n+a)^s} = \frac{1}{\Gamma(s)} \int_0^{\infty} \frac{t^{s-1} e^{-at}}{1 - z e^{-t}} dt.$$

Direct numerical integration is usually slower than well-designed methods based on series expansions. However, it is usually the easiest way to cover the whole domain.

Problems:

- Choosing valid or numerically satisfactory integration paths
- Tail bounds
- Good enclosures with large parameters (may need saddle-point analysis and series expansions)

Special functions based on ODEs

A particularly important family of special functions: *holonomic functions* (in one variable) are solutions of linear ODEs

$$a_r f^{(r)}(z) + \cdots + a_1 f'(z) + a_0 f(z) = 0$$

with polynomial coefficients $a_r, \ldots, a_0 \in \mathbf{C}[z],\ a_r \neq 0$.

Examples:

- $\exp(z)$, $\log(z)$, $J_n(z)$, $\Gamma(s,z)$, ${}_pF_q(\cdots, z)$
- Inverse Mellin transforms $K(z) = \frac{1}{2\pi i}\int_L z^{-s}\gamma(s)ds$ of gamma products

Closure properties:

- $\alpha f(z) + \beta g(z)$, $f(z)g(z)$, $f'(z)$, $\int_0^z f(t)dt$
- $f(h(z))$ where $h(z)$ is algebraic

Non-examples: $f(z)/g(z)$ (in general), $\tan(z)$, $\Gamma(z), \zeta(z)$

Special functions based on ODEs

Algorithmic tools for holonomic functions:[8]

- Closure properties are computable
- Function equality: $f = g$ is decidable if we can decide equality of constants
- Singularity analysis (locations, generalized series expansions)
- Rigorous numerical evaluation with low asymptotic complexity

Applications:

- Analysis of sequences via generating functions
- Symbolic definite integration and summation
- Convergence acceleration and analytic continuation
- Many techniques for computing with non-holonomic functions (e.g. L-functions) rely on holonomic functions

[8] See work by Stanley, Zeilberger, Chyzak, Salvy, van der Hoeven, Mezzarobba, and others. There are several implementations, including ore_algebra in SageMath.

Tangent: numerical evaluation of $\Gamma(s)$ and $\Gamma(s, z)$

Gamma function:[9]

- Stirling series + re-expansion in terms of hypergeometric series
- Efficient subroutines for rising factorials, Bernoulli numbers
- Taylor series
- Approximation by $\Gamma(s) \approx \Gamma(s, N)$
- Functional equations

Incomplete gamma function:

- Series expansions at $z_0 = 0$ and $z_0 = \infty$
- Bit-burst analytic continuation $0 \to \ldots \to z$ or $\infty \to \ldots \to z$
- Numerical integration of $\Gamma(s, z) = z^s e^{-z} \int_0^\infty e^{-zt}(1+t)^{s-1} dt$
- Functional equations
- Application: reduced-complexity computation of $L(s, \chi)$ [10]

More than 20,000 lines of code for these functions alone in Arb. Time to consider code generation?

---

[9] J., *Arbitrary-precision computation of the gamma function*, 2021

[10] J., *Rapid computation of special values of Dirichlet L-functions*, 2021

Tangent: evaluation of elementary functions

Schönhage's idea: perform argument reduction using

$$\exp(x) = 2^m 3^n \exp(\varepsilon), \quad \varepsilon = x - m\log(2) - n\log(3).$$

This can be improved by using many primes.[11] Example computation: $\exp(\sqrt{2}-1)$ to 10,000 digits. The 31-smooth approximation

$$e^{\sqrt{2}-1} = \frac{13^{651} \cdot 19^{463} \cdot 37^{634}}{2^{274} \cdot 3^{414} \cdot 5^{187} \cdot 7^{314} \cdot 11^{211} \cdot 17^{392} \cdot 23^{36} \cdot 29^{369} \cdot 31^{231}} \exp(\varepsilon)$$

gives $\varepsilon \approx -1.57 \cdot 10^{-32}$; we finish with holonomic methods for $\exp(\varepsilon)$. The entire evaluation costs roughly 25 full multiplications.

Arb now uses this method for all elementary functions above 1000 digits. Empirically, it is about twice as fast as previous methods, including the AGM.

[11] J., *Computing elementary functions using multi-prime argument reduction*, 2022

Conclusion

There are several useful approaches to computing with real and complex numbers (and functions). We should use all these methods where they make the most sense.

Apart from all the usual fun algorithm optimizations for particular operations/functions, there is a lot of room for improving integration and accessibility of the tools.

Where we are: the user (e.g. the working mathematician) does a lot of low-level coding to work around assorted bugs and numerical issues.

Where we should be: the user inputs a high-level mathematical statement involving real numbers. The software automatically chooses the appropriate low-level representation and algorithm, or at least makes it easy to choose.