Numerical Computations and Formal Proofs

Guillaume Melquiond

Université Paris-Saclay, CNRS, ENS Paris-Saclay, Inria
Laboratoire Méthodes Formelles

May 22, 2023

Approximating Real Numbers on Computers

Floating-point arithmetic in a nutshell

binary64 $= \{m \cdot 2^e \mid |m| < 2^{53} \wedge e \in [-1074; 971]\} \cup \{\pm\infty, \mathrm{NaN}\}$.

Operations: $+$, $-$, $\times$, $\div$, $\sqrt{\cdot}$.

Rounding: $u \oplus v = \circ(u + v)$ with $\circ : \mathbb{R} \to$ binary64.

Approximating Real Numbers on Computers

Floating-point arithmetic in a nutshell

$\text{binary64} = \{m \cdot 2^e \mid |m| < 2^{53} \wedge e \in [-1074; 971]\} \cup \{\pm\infty, \text{NaN}\}$.

Operations: $+$, $-$, $\times$, $\div$, $\sqrt{\cdot}$.

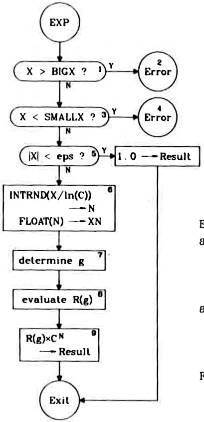
Rounding: $u \oplus v = \circ(u + v)$ with $\circ : \mathbb{R} \to \text{binary64}$.

Approximating a mathematical function, the wrong way

$$\exp x \simeq \begin{cases} +0 & \text{if } x \leq -746, \\ +\infty & \text{if } x \geq 710, \\ 1 + x + \frac{x^2}{2} + \ldots + \frac{x^{1200}}{1200!} & \text{otherwise.} \end{cases}$$

Cody & Waite, 1979: Implementing Exponential

b. Flow Chart for EXP(X)



for 30 ≤ b ≤ 42

p0 = 0.24999 99999 9992
p1 = 0.00595 04254 9776
q0 = 0.5
q1 = 0.05356 75176 4522
q2 = 0.00029 72936 3682

for 43 ≤ b ≤ 56

p0 = 0.24999 99999 99999 993
p1 = 0.00694 36000 15117 929
p2 = 0.00001 65203 30026 828
q0 = 0.5
q1 = 0.05555 38666 96900 119
q2 = 0.00049 58628 84905 441

Evaluate $R(g)$ in fixed point. First form $z = g^2$. Then form $g \cdot P(z)$ and $Q(z)$ using nested multiplication. For example, for $43 \le b \le 56$,

$$g \cdot P(z) = ((p2 * z + p1) * z + p0) * g$$

and

$$Q(z) = (q2 * z + q1) * z + q0.$$

Finally, form

$$r = .5 + g \cdot P(z)/[Q(z) - g \cdot P(z)]$$

in fixed point and convert back to floating point with $R(g)$ = REFLOAT(r) (see Chapter 2).

Cody & Waite, 1979: Implementing Exponential

Approximating $\exp x$

1. Argument reduction: $t = x - k \log 2$.
2. Rational approximation $f(t)$ of $\exp t$.
3. Result reconstruction: $\exp x = 2^k \exp t$.

Cody & Waite, 1979: Implementing Exponential

Approximating $\exp x$

1. Argument reduction: $t = x - k \log 2$.
2. Rational approximation $f(t)$ of $\exp t$.
3. Result reconstruction: $\exp x = 2^k \exp t$.

Source of errors

- Rounding errors: $\tilde{t} \simeq x - k \log 2$ and $\tilde{f}(\tilde{t}) \simeq f(\tilde{t})$.
- Method error: $f(\tilde{t}) \simeq \exp \tilde{t}$.

Cody & Waite, 1979: Implementing Exponential

Approximating $\exp x$

1. Argument reduction: $t = x - k \log 2$.
2. Rational approximation $f(t)$ of $\exp t$.
3. Result reconstruction: $\exp x = 2^k \exp t$.

Source of errors

- Rounding errors: $\tilde{t} \simeq x - k \log 2$ and $\tilde{f}(\tilde{t}) \simeq f(\tilde{t})$.
- Method error: $f(\tilde{t}) \simeq \exp \tilde{t}$.

Verifying a mathematical library is tedious and error-prone.

Using a Computer Algebra System

Bounding the method error $\frac{f(t) - \exp t}{\exp t}$ for $|t| \leq 0.35$.

About SageMathCell

SageMathCell

Type some Sage code below and press Evaluate.

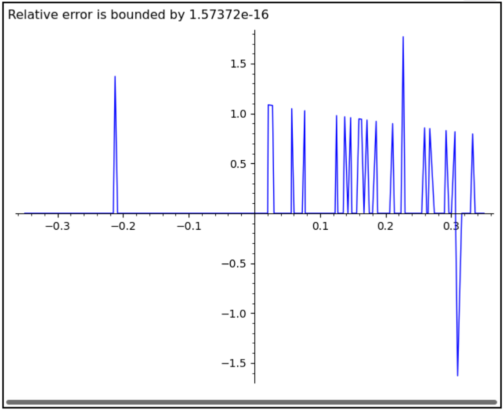
```
q2 = 4573527866750985 * 2**(-63)

t = SR.var('t')
t2 = t * t
p = p0 + t2 * (p1 + t2 * p2)
q = q0 + t2 * (q1 + t2 * q2)
f = 2 * ((t * p) / (q - t * p) + 1/2)
err(x) = (f(t = x) - exp(x)) / exp(x)
pretty_print(html(r'Relative error is bounded by %g' %
                  find_local_maximum(abs(err), -0.35, 0.35)[0]))
show(plot(err, -0.35, 0.35))
```

Evaluate

Language: Sage

Share



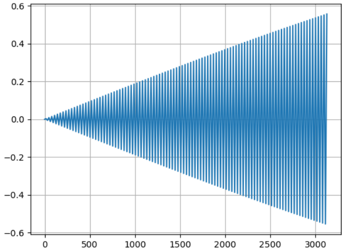
Help | Powered by SageMath

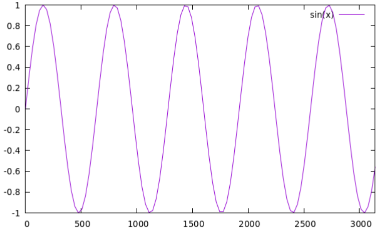
Plotting Gone Wrong

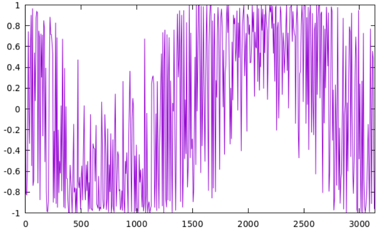
$\sin(x)$ for $x \in [0; 3141]$

How to sample?

- Gnuplot: 150 points
- Matplotlib: 200 points
- Sollya: 501 points + noise



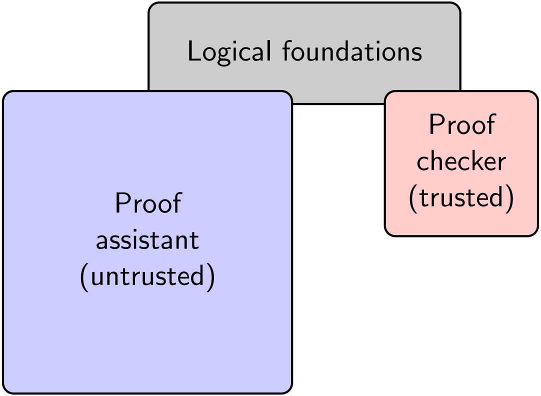




Outline

Outline

Formal Verification



Numbers in the Axiomatic World

$\mathcal{R} = (0, 1, +, \times, \leq, \ldots)$

- $u + 0 = u$,
- $(u + v) + w = u + (v + w)$,
- $(u + v) \times w = u \times w + u \times w$,
- $u + v \leq u + w \Leftrightarrow v \leq w$,
- $\ldots$

Numbers in the Axiomatic World

$\mathcal{R} = (0, 1, +, \times, \leq, \ldots)$

- $u + 0 = u$,
- $(u + v) + w = u + (v + w)$,
- $(u + v) \times w = u \times w + u \times w$,
- $u + v \leq u + w \Leftrightarrow v \leq w$,
- $\ldots$

Example: $(1 + 1) \times (1 + 1 + 1) \leq 1 + 1 + 1 + 1 + 1 + 1 + 1$

More than 10 proof steps.

Can we make it faster / more mechanical?

Numbers in the Computational World

Directed rewriting rules, provable from the axioms

- $u + 0 \to u$,
- $u + (v + 1) \to (u + v) + 1$,
- $u \times (v + 1) \to u \times v + u$,
- $(u + 1 \leq v + 1) \to (u \leq v)$,
- ...

Numbers in the Computational World

Directed rewriting rules, provable from the axioms

- $u + 0 \to u$,
- $u + (v + 1) \to (u + v) + 1$,
- $u \times (v + 1) \to u \times v + u$,
- $(u + 1 \leq v + 1) \to (u \leq v)$,
- ...

Example: $(1 + 1) \times (1 + 1 + 1) \leq 1 + 1 + 1 + 1 + 1 + 1 + 1$

Mindlessly apply the rules as much as possible. Result: $0 \leq 1$.
Remark: the number of proof steps stays about the same.

The unary representation will not scale to larger computations.

Binary Numbers in the Computational World

Integer literals, with $2 \equiv 1 + 1$

Horner-like representation: $21 \equiv 2 \times (2 \times (2 \times (2 \times 1) + 1)) + 1$.

Directed rewriting rules, provable from the axioms

- $2 \times u + 2 \times v \to 2 \times (u + v)$,
- $2 \times u + (2 \times v + 1) \to 2 \times (u + v) + 1$,
- $(2 \times u + 1) + (2 \times v + 1) \to 2 \times (u + v + 1)$,
- $(2 \times u) \times v \to 2 \times (u \times v)$,
- $(2 \times u + 1) \times v \to 2 \times (u \times v) + v$,
- ...

Checking that the rules are applied in the correct places is costly.

Reflection, the True Computational World

Step 1: Representing integers by lists of bits

1. Algebraic datatype $\mathcal{P} \equiv$ `xH | xO of` $\mathcal{P}$ `| xI of` $\mathcal{P}$.
2. Interpretation $\varphi : \mathcal{P} \to \mathcal{R}$.
   $\varphi(\texttt{xH}) \equiv 1$, $\varphi(\texttt{xO}\ p) \equiv 2 \times \varphi(p)$, $\varphi(\texttt{xI}\ p) \equiv 2 \times \varphi(p) + 1$.
3. Operations: $\texttt{plus} : \mathcal{P} \to \mathcal{P} \to \mathcal{P}$, $\texttt{mult} : \mathcal{P} \to \mathcal{P} \to \mathcal{P}$.
   - $\texttt{plus xH}\ (\texttt{xO}\ q) \equiv \texttt{xI}\ q$,
   - $\texttt{plus}\ (\texttt{xO}\ p)\ (\texttt{xI}\ q) \equiv \texttt{xI}\ (\texttt{plus}\ p\ q)$,

Reflection, the True Computational World

Step 1: Representing integers by lists of bits

1. Algebraic datatype $\mathcal{P} \equiv$ `xH | xO of` $\mathcal{P}$ `| xI of` $\mathcal{P}$.
2. Interpretation $\varphi : \mathcal{P} \to \mathcal{R}$.
   $\varphi(\mathtt{xH}) \equiv 1$, $\varphi(\mathtt{xO}\ p) \equiv 2 \times \varphi(p)$, $\varphi(\mathtt{xI}\ p) \equiv 2 \times \varphi(p) + 1$.
3. Operations: $\mathtt{plus} : \mathcal{P} \to \mathcal{P} \to \mathcal{P}$, $\mathtt{mult} : \mathcal{P} \to \mathcal{P} \to \mathcal{P}$.
   - $\mathtt{plus}\ \mathtt{xH}\ (\mathtt{xO}\ q) \equiv \mathtt{xI}\ q$,
   - $\mathtt{plus}\ (\mathtt{xO}\ p)\ (\mathtt{xI}\ q) \equiv \mathtt{xI}\ (\mathtt{plus}\ p\ q)$,

Step 2: Proving correctness lemmas

- $\varphi(\mathtt{plus}\ p\ q) = \varphi(p) + \varphi(q)$,
- $\varphi(\mathtt{mult}\ p\ q) = \varphi(p) \times \varphi(q)$.

$$2 \times 3 = \varphi(\mathtt{xO\ xH}) \times \varphi(\mathtt{xI\ xH}) = \varphi(\mathtt{mult\ (xO\ xH)\ (xI\ xH)}) = \varphi(\mathtt{xO\ (xI\ xH)}) = 6.$$

Going Faster: Large Integers

Multiplication, division, square root, over lists of bits, have quadratic complexity. How to go faster?

Perfect binary trees (Grégoire, Théry, 06)

- Algebraic datatype $\mathcal{T}_{k+1} \equiv \texttt{WW of } \mathcal{T}_k \times \mathcal{T}_k$.
- Interpretation $\varphi_{k+1}(\texttt{WW } u\ v) \equiv B^{2^k} \times \varphi_k(u) + \varphi_k(v)$.
- Operations $\texttt{mult} : \mathcal{T}_k \to \mathcal{T}_k \to \mathcal{T}_{k+1}$, etc.
- Lemmas: $\varphi_{k+1}(\texttt{mult } p\ q) = \varphi_k(p) \times \varphi_k(q)$, etc.

Going Even Faster: Hardware Integers

Processors have ALUs and we have to trust them anyway.
How to leverage them?

63-bit integers (Armand *et al*, 10)

- Abstract datatype $\mathcal{T}_0$.
- Interpretation $\varphi : \mathcal{T}_0 \to \mathcal{R}$.
- Operations $\texttt{plus} : \mathcal{T}_0 \to \mathcal{T}_0 \to \mathcal{T}_0 \times \mathbb{B}$,
  $\texttt{mult} : \mathcal{T}_0 \to \mathcal{T}_0 \to \mathcal{T}_0 \times \mathcal{T}_0$.
- Axioms:
  - $\varphi(p) + \varphi(q) = \varphi(r) + 2^{63}c$ with $(r, c) = \texttt{plus}\ p\ q$,
  - $\varphi(p) \times \varphi(q) = \varphi(r) + 2^{63}\varphi(s)$ with $(r, s) = \texttt{mult}\ p\ q$.

Hardware Floating-Point Numbers

Binary64 with a single NaN (Bertholon *et al*, 19)

- Abstract datatype $\mathcal{F}$.
- Interpretation $\varphi : \mathcal{F} \to \mathbb{F}$.
- Operations $\texttt{plus} : \mathcal{F} \to \mathcal{F} \to \mathcal{F}$, etc
- Axioms: $\varphi(\texttt{plus}\ u\ v) = \varphi(u) \oplus \varphi(v)$, etc.

But what is $\mathbb{F}$? What are $\oplus$, $\otimes$, etc?
Remark: If $\oplus$, $\otimes$, etc do not match, the system is inconsistent.

Software Floating-Point Numbers

Naive floating-point arithmetic (Boldo *et al*, 13)

- Algebraic datatype $\mathbb{F} \equiv \{\pm 0, \pm\infty, \mathrm{NaN}\} \cup \{(m,e) \in \mathbb{Z}^2 \mid \ldots\}$.
- Interpretation $\varphi : \mathbb{F} \to \mathbb{R}$, $\varphi(m,e) \equiv m \cdot 2^e$.
- Naive algorithms, rounded to nearest even: $\oplus$, $\otimes$, etc.
- Other operations: predecessor and successor, conversions, etc.

Software Floating-Point Numbers

Naive floating-point arithmetic (Boldo *et al*, 13)

- Algebraic datatype $\mathbb{F} \equiv \{\pm 0, \pm\infty, \mathrm{NaN}\} \cup \{(m, e) \in \mathbb{Z}^2 \mid \ldots\}$.
- Interpretation $\varphi : \mathbb{F} \to \mathbb{R}$, $\varphi(m, e) \equiv m \cdot 2^e$.
- Naive algorithms, rounded to nearest even: $\oplus$, $\otimes$, etc.
- Other operations: predecessor and successor, conversions, etc.

Correctness statement? IEEE-754 to the rescue

"Every operation shall be performed as if it first produced an intermediate result correct to infinite precision and with unbounded range, and then rounded that result."

Lemma: $\varphi(u \oplus v) = \circ(\varphi(u) + \varphi(v))$ if there are no overflow.

Outline

Handling Approximate Computations

Interval arithmetic

- Algebraic datatype $\mathcal{I} \equiv \mathcal{F} \times \mathcal{F}$.
- Interpretation $\varphi : \mathcal{I} \to \mathcal{P}(\mathbb{R})$, $\varphi(\underline{u}, \overline{u}) \equiv [\underline{u}; \overline{u}]$.
- Operations $\texttt{plus} : \mathcal{I} \to \mathcal{I} \to \mathcal{I}$, etc.

Lemma: containment property

$\varphi(\mathbf{u}) + \varphi(\mathbf{v}) \subseteq \varphi(\texttt{plus}\ \mathbf{u}\ \mathbf{v})$,

i.e., $\forall u, v \in \mathbb{R},\ u \in \varphi(\mathbf{u}) \wedge v \in \varphi(\mathbf{v}) \Rightarrow u + v \in \varphi(\texttt{plus}\ \mathbf{u}\ \mathbf{v})$.

Interval Arithmetic in a Nutshell

Interval extensions of $+$, $-$, $\times$

If $u \in [\underline{u}, \overline{u}]$ and $v \in [\underline{v}, \overline{v}]$, then

$$\begin{aligned}
u + v &\in [\nabla(\underline{u} + \underline{v}); \triangle(\overline{u} + \overline{v})],\\
u - v &\in [\nabla(\underline{u} - \overline{v}); \triangle(\overline{u} - \underline{v})],\\
u \times v &\in [\min(\nabla(\underline{u} \cdot \underline{v}), \nabla(\underline{u} \cdot \overline{v}), \nabla(\overline{u} \cdot \underline{v}), \nabla(\overline{u} \cdot \overline{v}));\\
&\qquad \max(\triangle(\underline{u} \cdot \underline{v}), \triangle(\underline{u} \cdot \overline{v}), \triangle(\overline{u} \cdot \underline{v}), \triangle(\overline{u} \cdot \overline{v}))].
\end{aligned}$$

Proof by monotony.

Intervals in Coq

Example: Number of decimal digits of 500!

```
Definition stirling x eps :=
  sqrt (2 * PI * x) * exp (x * (ln x - 1))
    * exp (1 / (12 * x + eps)).
Definition digits x :=
  IZR (Ztrunc (ln x / ln 10)) + 1.

Goal forall eps, 0 <= eps <= 1 ->
  digits (stirling 500 eps) = 1135.
Proof.
  intros eps Heps. apply eq_sym, Rle_le_eq.
  interval with (i_prec 30).
Qed.
```

Dependency/Wrapping Effect

Interval arithmetic works best when intervals are narrow or variables occur only once each.

Example: $x - x^2$ when $x \in [0; 1]$

$x - x^2 \in [0; 1] - [0^2; 1^2] = [0 - 1; 1 - 0] = [-1; 1]$.

Yet $x - x^2 \in [0; \frac{1}{4}]$.

Dependency/Wrapping Effect

Interval arithmetic works best when intervals are narrow
or variables occur only once each.

Example: $x - x^2$ when $x \in [0; 1]$

$x - x^2 \in [0; 1] - [0^2; 1^2] = [0 - 1; 1 - 0] = [-1; 1]$.

Yet $x - x^2 \in [0; \frac{1}{4}]$.

Brute-force approach: bisection

For $x \in [0; \frac{1}{2}]$, $x - x^2 \in [0; \frac{1}{2}] - [0; \frac{1}{4}] = [-\frac{1}{4}; \frac{1}{2}]$.
For $x \in [\frac{1}{2}; 1]$, $x - x^2 \in [\frac{1}{2}; 1] - [\frac{1}{4}; 1] = [-\frac{1}{2}; \frac{3}{4}]$.

So, $x - x^2 \in [-\frac{1}{2}; \frac{3}{4}]$.

Automatic Differentiation

Mean-value theorem

$\forall x \in \mathbf{x},\ \exists \xi \in \mathbf{x},\ f(x) = f(x_0) + (x - x_0) \cdot f'(\xi).$

Corollary: $\forall x \in \mathbf{x},\ f(x) \in (\mathbf{f}(x_0) + (\mathbf{x} - x_0) \cdot \mathbf{f}'(\mathbf{x})) \cap \mathbf{f}(\mathbf{x}).$

Special case: If $0 \notin \mathbf{f}'(\mathbf{x})$, then $f(x) \in \mathrm{hull}(\mathbf{f}(\underline{x}) \cup \mathbf{f}(\overline{x}))$.

Automatic Differentiation

Mean-value theorem

$\forall x \in \mathbf{x},\ \exists \xi \in \mathbf{x},\ f(x) = f(x_0) + (x - x_0) \cdot f'(\xi).$

Corollary: $\forall x \in \mathbf{x},\ f(x) \in (\mathbf{f}(x_0) + (\mathbf{x} - x_0) \cdot \mathbf{f}'(\mathbf{x})) \cap \mathbf{f}(\mathbf{x}).$

Special case: If $0 \notin \mathbf{f}'(\mathbf{x})$, then $f(x) \in \text{hull}(\mathbf{f}(\underline{x}) \cup \mathbf{f}(\overline{x})).$

Automatic differentiation

$$
\begin{aligned}
(\mathbf{u}, \mathbf{u}') + (\mathbf{v}, \mathbf{v}') &\equiv (\mathbf{u} + \mathbf{v}, \mathbf{u}' + \mathbf{v}'),\\
(\mathbf{u}, \mathbf{u}') \times (\mathbf{v}, \mathbf{v}') &\equiv (\mathbf{u} \cdot \mathbf{v}, \mathbf{u}' \cdot \mathbf{v} + \mathbf{u} \cdot \mathbf{v}'),\\
\exp(\mathbf{u}, \mathbf{u}') &\equiv (\exp(\mathbf{u}), \mathbf{u}' \cdot \exp(\mathbf{u})).
\end{aligned}
$$

Application: Root Finding

Interval-based Newton method: $f(x) = 0$

If $x \in \mathbf{x}$ and $f(x) = 0$, then $x \in m(\mathbf{x}) - \dfrac{\mathbf{f}(m(\mathbf{x}))}{\mathbf{f}'(\mathbf{x})}$.

Proof: $0 = f(x) = f(m(\mathbf{x})) + (x - m(\mathbf{x})) \cdot f'(\xi)$ with $\xi \in \mathbf{x}$.

Example: Solution of a quintic equation

```
Goal forall x, x^5 - x = 1 ->
  x = 1.1673039782614185 ± 1e-14.
Proof.
  intros x H.
  root H.
Qed.
```

Polynomial Approximations

Taylor-Lagrange Formula

$\forall x \in \mathbf{x},\ \exists \xi \in \mathbf{x},$

$$f(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k + \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)^{n+1}.$$

Polynomial Approximations

Taylor-Lagrange Formula

$\forall x \in \mathbf{x},\ \exists \xi \in \mathbf{x},$

$$f(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k + \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)^{n+1}.$$

Polynomial approximation (Brisebarre *et al*, 12)

$(\vec{\mathbf{p}}, \mathbf{\Delta})$ encloses f over $\mathbf{x} \ni x_0$ if

$$\exists p \in \mathbb{R}[X], \quad (\forall i,\ p_i \in \mathbf{p}_i)\ \wedge\ \forall x \in \mathbf{x},\ f(x) - p(x - x_0) \in \mathbf{\Delta}.$$

- Taylor-Lagrange formula for elementary functions,
- polynomial arithmetic for composite expressions.

Example: Cody & Waite's Exponential

Bounding the relative method error

```
Definition f t :=
  let t2 := t * t in
  let p := p0 + t2 * (p1 + t2 * p2) in
  let q := q0 + t2 * (q1 + t2 * q2) in
  2 * ((t * p) / (q - t * p) + 1/2).

Lemma method_error :
  forall t : R, Rabs t <= 0.35 ->
  Rabs ((f t - exp t) / exp t) <= 5e-18.
Proof.
  intros t Ht.
  interval with (i_bisect t, i_taylor t, i_prec 80).
Qed.
```

Proper Definite Integrals

Naive approach

If f is continuous over $[u; v]$, then

$$\int_u^v f \in (\mathbf{v} - \mathbf{u}) \cdot \mathbf{f}(\mathrm{hull}(\mathbf{u}, \mathbf{v})).$$

Proper Definite Integrals

Naive approach

If f is continuous over $[u; v]$, then

$$\int_u^v f \in (\mathbf{v} - \mathbf{u}) \cdot \mathbf{f}(\mathrm{hull}(\mathbf{u}, \mathbf{v})).$$

Using polynomials (Mahboubi *et al*, 16)

If $(p, \mathbf{\Delta})$ encloses f over $[u; v]$, and if P is a primitive of p, then

$$\int_u^v f \in P(\mathbf{v}) - P(\mathbf{u}) + (\mathbf{v} - \mathbf{u}) \cdot \mathbf{\Delta}.$$

Improper Definite Integrals

Naive approach

Assume that f is bounded, f and g are continuous, and g has a constant sign, over $[u; +\infty)$.

If $\int_u^{+\infty} g$ exists and is enclosed in $\mathbf{G}$, then $\int_u^{+\infty} fg$ exists, and

$$\int_u^{+\infty} fg \in \mathbf{f}(\mathrm{hull}(\mathbf{u}, +\infty)) \cdot \mathbf{G}.$$

Example: Helfgott's Proof of Ternary Goldbach Conjecture

Every odd number greater than 5 is the sum of three primes.

$$\int_{-\infty}^{\infty} \frac{(0.5 \cdot \ln(\tau^2 + 2.25) + 4.1396 + \ln \pi)^2}{0.25 + \tau^2} \, d\tau \leq 226.844.$$

```
Goal RInt (fun tau =>
       (0.5 * ln(tau^2 + 2.25) + 4.1396 + ln PI)^2
       / (0.25 + tau^2))
     (-100000) 100000
  = 226.8435 ± 2e-4.
Proof. integral. Qed.

Goal RInt_gen (fun tau =>
       ... * (powerRZ tau (-2) * (ln tau)^2))
     100000 p_infty
  = 0.00317742 ± 1e-5.
Proof. integral. Qed.
```

Function Plots

A function plot is...

- correct if blank pixels are not traversed by the function; ✓
- complete if filled pixels are traversed by the function. ✗

Function Plots

A function plot is. . .

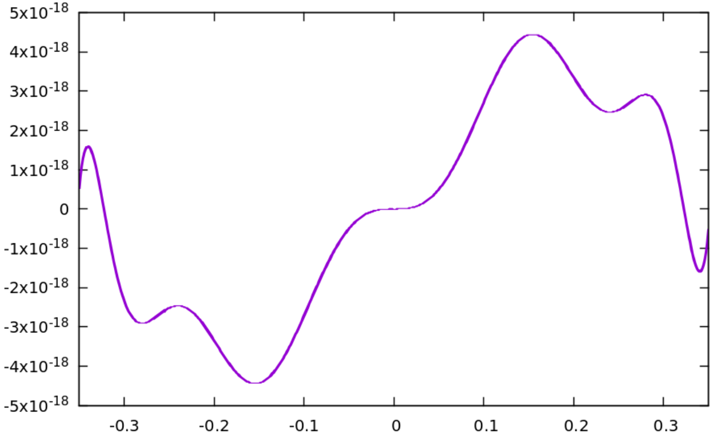
- correct if blank pixels are not traversed by the function; ✓
- complete if filled pixels are traversed by the function. ✗

Plotting is no harder than integrating

1. Split $[u; v]$ into smaller subintervals W_k.
2. Compute a polynomial approximation (p_k, Δ_k) of f over W_k.
3. If Δ_k is not thin enough, go back to step 1.
4. Do something over W_k using interval arithmetic:
   - Integrate $p_k + \Delta_k$, then accumulate.
   - Plot $p_k + \Delta_k$, one horizontal pixel at a time.

Example: Cody & Waite's Exponential

```
Plot ltac:(plot (fun t => (f t - exp t) / exp t)
    (-0.35) 0.35 with (i_prec 80)).
```



Outline

Pocket Calculator

30 digits of $\pi^2/6$

```
Definition zeta_2 := ltac:(interval (PI^2 / 6)
   with (i_prec 100, i_decimal)).
About zeta_2.
(* zeta_2 :   1.64493406684822643647241516664 <=
  PI^2 / 6 <= 1.64493406684822643647241516666 *)
```

$\pi^2/6$ again, but harder

```
Definition zeta_2 := ltac:(integral (RInt_gen
    (fun x => 1/(1+x)^2 * (ln x)^2)
    (at_right 0) (at_point 1)
  ) with (i_relwidth 30, i_decimal)).
About zeta_2.
(* zeta_2 :       1.644934066432123 <=
  RInt_gen ... <= 1.644934067350727 *)
```

What About Rounding Errors? Flocq & Gappa

Accuracy of Cody & Waite's exponential

```
Definition cw_exp (x : R) :=
  let k := nearbyint (mul x InvLog2) in
  let t := sub (sub x (mul k Log2h)) (mul k Log2l) in
  ...

Theorem exp_correct : forall x : R,
  generic_format radix2 (FLT_exp (-1074) 53) x ->
  -746 <= x <= 710 ->
  Rabs ((cw_exp x - exp x) / exp x) <= pow2 (-51).
Proof.
... generalize (method_error t Bt).
... gappa.
Qed.
```

Perspectives

What is next?

1. Improve the usability of Coq to verify floating-point code.
2. Turn Coq into a tool suitable for experimental mathematics.

Where to find the tools?

https://coqinterval.gitlabpages.inria.fr/
https://flocq.gitlabpages.inria.fr/
https://gappa.gitlabpages.inria.fr/