Evaluating straight-line programs over balls

JORIS VAN DER HOEVEN

AND

GRÉGOIRE LECERF

LABORATOIRE D'INFORMATIQUE

ÉCOLE POLYTECHNIQUE

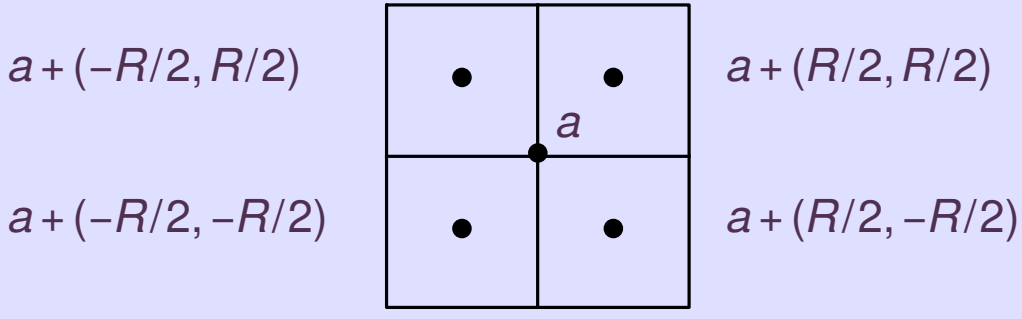
→ Compute fast with intervals and balls, subdivision methods, certified path tracking...

Example of subdivision method: draw $f > 0$,

where $f\colon \mathbb{R}^2 \to \mathbb{R}$ is continuous in $a + [-R, R]^2$.

We evaluate $B(b,s) = f(B(a, \sqrt{2}\,R)) \in B(\mathbb{R}^2, \mathbb{R})$, and distinguish three cases:

- All the points in $B(b,s)$ are positive, then draw all points in $a + [-R, R]^2$.
- All the points in $B(b,s)$ are negative, then return.
- Otherwise $0 \in B(b,s)$, subdivide $[-R, R]^2$ into 4 squares of size $[-R/2, R/2]^2$, and call the algorithm recursively on each smaller square.



⇨ Very simple robust technique. / Balls have “large” radii in low recursion depth.

Representation

$\mathfrak{R}$ is the set of machine floating point numbers, with $p \geqslant 16$ bits of precision.

$E_{\min}$ and $E_{\max}$ are the minimal and maximal exponents (included).

For IEEE-754 double precision numbers: $p = 53$, $E_{\min} = -1022$ and $E_{\max} = 1023$.

$\mathfrak{R}$ is enlarged with symbols $-\infty$, $+\infty$, and NaN.

Rounding modes

$\downarrow$ downwards $\uparrow$ upwards $\natural$ nearest $\wr$ unspecified

Let $\circ \in \{\downarrow, \natural, \uparrow, \wr\}$, $x \in \mathbb{R}$, and $* \in \{+, -, \times, \ldots\}$.

$x_\circ \in \mathfrak{R}$ is the approximation of x in $\mathfrak{R}$ with the specified rounding mode.

$x *_\circ y$ is a shorthand for $(x * y)_\circ$, $\circ[x\,y + a^2\,b]$ is a shorthand for $x_\circ \times_\circ y_\circ +_\circ (a_\circ \times_\circ a_\circ) \times_\circ b_\circ$.

Errors

$\varepsilon_\circ(x) := |x_\circ - x|$ stands for the rounding error, that may be $+\infty$.

$\bar{\varepsilon}_\circ$ is any upper bound function for $\varepsilon_\circ$ that is easy to compute.

With IEEE-754, in absence of underflows/overflows, we may take

$\bar{\varepsilon}_\circ(x) = |x_\circ|\,\epsilon_\circ$, with $\epsilon_\circ = 2^{-p+1}$ for $\circ \neq \natural$ and $\epsilon_\natural = 2^{-p}$.

Closed balls

Let $\mathbb{A}$ denote $\mathbb{R}$ or $\mathbb{C}$.

Let $a \in \mathbb{A}$ and $r \in \mathbb{R}^{\geqslant} := \{x \in \mathbb{R} : x \geqslant 0\}$.

$$B(a,r) := \{x \in \mathbb{A} : |x-a| \leqslant r\}.$$

The set of such balls is denoted by $B(\mathbb{A},\mathbb{R})$.

Ball operations

$$\begin{aligned} B(a,r) \pm B(b,s) &:= B(a \pm b, r+s), \\ B(a,r) \times B(b,s) &:= B(ab, (|a|+r)s + |b|r). \end{aligned}$$

Inclusion principle

given $* \in \{+,-,\times\}$, $x \in B(a,r)$ and $y \in B(b,s)$, we have $x * y \in B(a,r) * B(b,s)$.

In fact, if $a' \in B(a,r)$ and $b' \in B(b,s)$, then

$$|a'b' - ab| \leqslant |a'(b'-b) + b(a'-a)| \leqslant (|a|+r)s + |b|r.$$

Usual certified arithmetic for balls with centers in $\mathfrak{R}$ or $\mathfrak{R}[\mathrm{i}]$ and radii in $\mathfrak{R}$:

$$\begin{aligned} B(a,r) \pm_\circ B(b,s) &:= B(a \pm_\circ b, \uparrow[r+s+\bar{\varepsilon}_\circ(a \pm b)]) \\ B(a,r) \times_\circ B(b,s) &:= B(a \times_\circ b, \uparrow[(\|a\|+r)s+\|b\|r+\bar{\varepsilon}_\circ(ab)]). \end{aligned}$$

- The inclusion principle is preserved.
- Usually, the rounding mode $\circ$ for centers is set to the nearest.

- Matrix products over real balls: series of papers by Rump, Ogita, Oishi, Ozaki, from 1999.
  - ⇨ Mostly perform operations on centers, and then rely on fast bounds on radii.
  - ⇨ Exploit HPC solutions for numeric types.
- Matrix products over intervalls: Hong Diep Nguyen (2011), Revol and Théveny (2014).
- Use of SIMD instructions for intervals: Gudenberg (2002).
  - ⇨ Rather modest speed-up.
  - ⇨ Changing the rounding mode is expensive on old hardware such as x87.
- Stolte (2005): use x87 and SSE units with independent rounding modes for interval arithmetic.
- Use of the opposite trick to minimize rounding mode changes in interval arithmetic: Gudenberg (2002), Lambov (2008), Goualard (2008).

⇨ Nowadays, switching rounding modes is fast with AVX technologies.

⇨ Rounding modes are even integrated into AVX-512 instructions.

A straight-line program (SLP) Γ over a ring $\mathbb{A}$ is a sequence $\Gamma_1, \ldots, \Gamma_l$ of instructions

$$\begin{aligned} \Gamma_k &\equiv X_k := C_k && \text{or} \\ \Gamma_k &\equiv X_k := Y_k * Z_k, && \text{where} \end{aligned}$$

X_k, Y_k, Z_k are variables in a finite ordered set $\mathscr{V}$,

C_k are constants in $\mathbb{A}$, and $* \in \{+, -, \times\}$.

Input variables: $I_1, \ldots, I_m$, are those that appear for the first time in the sequence in the right-hand side of an instruction.

Output variables: a distinguished subset $O_1, \ldots, O_n$ of the variables.

The length $l_\Gamma = l$ of the sequence is called the length of Γ.

Evaluation function: $E_\Gamma : \mathbb{A}^m \to \mathbb{A}^n$

given $(a_1, \ldots, a_m) \in \mathbb{A}^m$, we assign a_i to I_i for $i = 1, \ldots, m$,

then evaluate the instructions of Γ in sequence, and

finally read off the values of $O_1, \ldots, O_n$, which determine $E_\Gamma(a_1, \ldots, a_m)$.

Let us consider Γ, of length $l = 4$:

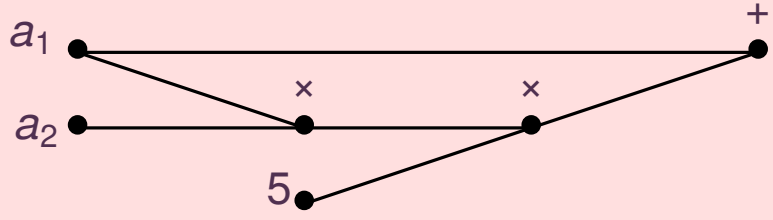
$$x_1 := 5, \qquad x_2 := a_1 \times a_2, \qquad x_1 := x_1 \times x_2, \qquad x_3 := x_1 + a_1.$$

The input variables are a_1 and a_2.

We distinguish x_3 as the sole output variable.

This SLP thus computes the function $5\,a_1\,a_2 + a_1$.

The associated computation graph:



Let us fix a rounding mode $\circ \in \{\downarrow, \updownarrow, \uparrow, \lozenge\}$.

⇨ Taking errors on the centers into account for certified ball arithmetic is expensive.

Transient ball arithmetic

$\tilde{\circ}$ denotes the corresponding "rounding mode" for :

$$\begin{aligned} B(a,r) \pm_{\tilde{\circ}} B(b,s) &:= B(a \pm_\circ b, \circ[r+s]), \\ B(a,r) \times_{\tilde{\circ}} B(b,s) &:= B(a \times_\circ b, \circ[(|a|+r)\,s + |b|\,r]). \end{aligned}$$

⇨ These formulas do not satisfy the inclusion principle.

Semi-exact ball arithmetic

with centers in $\mathfrak{A} = \mathfrak{R}$ or $\mathfrak{A} = \mathfrak{R}[\mathrm{i}]$, and radii in $\mathbb{R}$:

All computations on centers are done using a given rounding mode $\circ$.

All computations on radii are exact.

$$\begin{aligned} B(a, r^*) \pm^* B(b, s^*) &= B(a +_\circ b, r^* + s^* + \bar{\varepsilon}_\circ(a+b)), \\ B(a, r^*) \times^* B(b, s^*) &= B(a \times_\circ b, (|a| + r^*)\, s^* + |a|\, s^* + \bar{\varepsilon}_\circ(ab)), \end{aligned}$$

for any $a, b \in \mathfrak{A}$ and $r^*, s^* \in \mathbb{R}^{\geqslant}$.

⇨ It satisfies the inclusion principle.

⇨ We investigate how far the transient arithmetic deviates from this semi-exact arithmetic.

Let Γ be a SLP of length l_Γ, and depth q_Γ, and let $\alpha > 0$ be such that $1+\alpha > (1+\epsilon)^{4q_\Gamma}$.

Consider two evaluations of Γ with two different ball arithmetics:

- the first evaluation uses the semi-exact arithmetic with $\bar{\varepsilon}_\circ(x) = |x_\circ|\,\epsilon$;
- the second evaluation uses transient ball arithmetic with $\circ$.

Any input or constant ball $B(a, r^*)$ is replaced by a larger ball $B(a, r)$ with

$$r \geqslant \max\left(|a|\left((1+\epsilon)^{\beta q_\Gamma} - 1\right), (1+\alpha)\, r^*\right), \text{ where}$$

$$\beta \geqslant \max\left(3, \frac{1+\alpha}{\alpha}\gamma\right), \qquad \gamma \geqslant \left(1 + \frac{1}{2} + \cdots + \frac{1}{q_\Gamma}\right)(1+\epsilon)^{4q_\Gamma}\frac{\alpha}{1+\alpha}\left(1 - \frac{(1+\epsilon)^{4q_\Gamma}}{1+\alpha}\right)^{-1}.$$

Assume: no underflow or overflow occurs during the second evaluation.

Conclusion: for all $B(c, t^*)$ in the output of the first evaluation with corresponding entry $B(c, t)$ for the second evaluation, we have $t^* \leqslant t$.

$\Rightarrow$ $\alpha \simeq 1$, $\gamma \simeq \frac{\log q_\Gamma}{4\, q_\Gamma}$, $\beta q_\Gamma \simeq \log q_\Gamma$, so $r \gtrsim \epsilon\,|a| \log q_\Gamma$.

$\Rightarrow$ The loss of relative precision grows with the depth q_Γ of the SLP.

MATHEMAGIX

- C++ libraries:

  NUMERIX: `double`, complex, intervall, balls, modular integers, arbitrary large integers (GMP) and floating point numbers (MPFR), optimized fixed medium precision.

  MULTIMIX: SLPs, naive interpreted evaluation, compilation into dynamic libraries loaded *via* `dlopen`, and fast JIT compilation (restricted to double and SSE2). Plus other structures for multivariate polynomials and series.

- Compiled MATHEMAGIX language:

  RUNTIME: basic JIT compilation (*just in time*) facilities from assembly language.

  JUSTINLINE: templated SLP data type with additional JIT facilities. This includes *common subexpression simplification*, *constant simplification*, *register allocation*, and *vectorization*.

- Import/exports between C++ and MATHEMAGIX.

http://www.mathemagix.org

Platform

INTEL(R) CORE(TM) i7-4770 CPU at 3.40 GHz and 8 GB of 1600 MHz DDR3 memory.

Features AVX2 and FMA technologies.

JESSIE GNU DEBIAN operating system with a 64 bit LINUX kernel version 3.14.

GCC version 4.9.2 with options `-O3 -mavx2 -mfma -mfpmath=sse`.

Benchmark

A multivariate polynomial over `double`, with 10 variables, made of 100 terms,

built from random monomials of partial degrees at most 10.

The evaluation of this SLP takes 1169 products and 100 sums.

Timings in (µs)	`double`	`complex<double>`
MULTIMIX, interpreted, naive	2.1	3.4
MULTIMIX, compiled in 260ms	0.29	1.3
MULTIMIX, JIT, compiled in 50µs, no optimization	0.84	N/A
JUSTINLINE, JIT, compiled in 8ms, optimized	0.43	1.4

Early versions of the MATHEMAGIX libraries already contained a C99 portable implementation of ball arithmetic in the NUMERIX library.

⇨ Serious overhead due to function calls to libc, libm.

We carefully tuned the AVX2 + FMA assembly code generated by our SLP compiler:

For instance, if `ymm0` contains -0.0 and if `ymm1` contains a center a, then $-a$ is obtained as `vxorpd ymm0 ymm1 ymm2`, and $|a|$ as `vandnpd ymm0 ymm1 ymm2`.

⇨ latency = throughput = 1 cycle to compute $|a|$.

Timings in (µs)	Ball over `double`
Naive, exact, C99	62
Naive, transient	14
Compiled, transient	2.0
JIT, exact	3.2
JIT, transient	1.8

⇨ Transient arithmetic is just about 4.2 times slower than numeric arithmetic.

⇨ This turns out to be competitive with interval arithmetic, where each interval product usually requires 8 machine multiplications and 6 min/max operations.

The computation of norms is expensive in this case, because the scalar square root instruction takes 13 CPU cycles.

In order to reduce this cost, we rewrite SPLs so that norms of products are computed as products of norms.

Timings in (µs)	Ball over `complex<double>`
Naive, exacte, C99	130
Naive, transient	18
Compiled, transient	4.0
JIT, exact	4.5
JIT, transient	3.1

Transient arithmetic strategy is about 2.4 slower than numeric arithmetic.

Conclusion

- Robust numeric arithmetic finally involves a small overhead *via* AVX & FMA technologies.

More in the paper

- Precision analysis;
- Managing underflows and overflows;
- Vectorization.

Work in progress

- Significant speed-ups in our numerical solvers;
- Robust numerical integration;
- Development of specific JIT compilers for SLPs;
- Adapt present results to standard intervals.