An implementation of a posteriori interval analysis technique and its application to linear algebra problems

Vladimir Glazachev

St. Petersburg State University, Russia
glazachev.vladimir@gmail.com

SWIM 2016, June 21, Lyon

Interval analysis

Interval — pair $(val, err) = [val - err, val + err]$

Arithmetic operations

- $(v_1, e_1) + (v_2, e_2) = (v_1 + v_2, e_1 + e_2)$
- $(v_1, e_1) - (v_2, e_2) = (v_1 - v_2, e_1 + e_2)$
- $(v_1, e_1) * (v_2, e_2) = (v_1 v_2, |v_1| e_2 + |v_2| e_1 + e_1 e_2)$
- $\dfrac{(v_1, e_1)}{(v_2, e_2)} = (\dfrac{v_1}{v_2}, \dfrac{e_1 + \frac{v_1}{v_2} + e_2}{|v_2| - e_2})$

Problems:

- $A(B + C) \subset AB + AC$ - subdistributivity
- $(val, err) - (val, err) = (0, 2err)$

Dependency problems...

Optimal solution

We want to compute $y = (v_y, e_y) = Y(x_1, x_2, ..., x_n)$, where $x_i = (v_i, e_i)$ and Y — rational function

Optimal solution

$$\Delta(\bar{v}, \bar{e}) = \max_{|r_i - v_i| < e_i} |Y(r_1, ..., r_n) - Y(v_1, ..., v_n)|$$

Find optimal solution — NP-hard for many interval algorithms (determinant[1], Linear equation systems[2]).

---

[1] A. A. Gaganov, "Computational complexity of the range of the polynomial in several variables", *Cybernetics*, 1985, pp. 418–421

[2] V. Kreinovich, A. V. Lakeyev, and S. I. Noskov, "Optimal solution of interval linear systems is intractable (NP-hard)." *Interval Computations*, 1993, No. 1, pp. 6–14

Asymptotically optimal solution

Asymptoticallly optimal solution

For small values e_i we have: $Y(\bar{r}) - Y(\bar{v}) \approx \sum_{i=1}^{n} \frac{\partial Y(\bar{v})}{\partial v_i}(r_i - v_i)$

Error estimation: $\Delta(\bar{v}, \bar{e}) \approx \sum_{i=1}^{n} \left|\frac{\partial Y(\bar{v})}{\partial v_i}\right| e_i = e_y$

e_y — *asymptotically optimal* if

$$\frac{\Delta(\bar{v}, \bar{e})}{e_y} \xrightarrow[\bar{e} \to \bar{0}]{} 1$$

for each v_i such that $Y(\bar{v})$ is defined and $\frac{\partial Y(\bar{v})}{\partial v_i} \neq 0$.

For traditional interval analysis we have only $\frac{\Delta(\bar{v}, \bar{e})}{e_y} = O(1)$

Generalized Interval arithmetic[1]

Represent each input variable $x_i = v_i + c_i$ where $c_i \in [-e_i, e_i]$

Generalized interval

$$X_i = Y_i + \sum_{j=1}^{n} c_j z_{ij}$$

- need to compute $z_{i+1,j}$ for $j = 1..n$ after each operation
- n inputs, $T(n)$ complexity $=>$ generalized method have $O(nT(n))$ time complexity

[1] E.R. Hansen, A Generalized Interval Arithmetic. *Lecture Notes in Computer Science*, 1975, Vol. 29, pp. 7-18

Aposteriori interval arithmetic

Suppose that the inputs $x_1, ..., x_n$ — functions of variable t.
During computation we have $x_{n+1}, ..., x_m = y$. So

$$y'(0) = \sum_{i=1}^{n} \frac{\partial y(0)}{\partial x_i} x_i'(0)$$

We can solve (for $n \leq k \leq m$):

$$y'(0) = \sum_{i=1}^{k} z_i x_i'(0)$$

- $k = m$ — trivial solution $z_m = 1, z_i = 0$ for $i < m$; append it to the end of the program
- build solution for $k = k - 1$ by induction

Aposteriori interval arithmetic (continue)

Program

$$(x_1 = x_1^0, \dots, x_n = x_n^0);$$

$$x_{n+1} := x_{i_{n+1}} \circ_{n+1} x_{j_{n+1}};$$

$$\dots$$

$$x_l := x_{i_l} \circ_l x_{j_l};$$

$$\dots$$

$$x_m := x_{i_m} \circ_m x_{j_m};$$

$$z_1 := 0;$$

$$\dots$$

$$z_{m-1} := 0;$$

$$z_m := 1;$$

For $l = k-1$ to n

if $\circ_l = +$, append

$$z_i := z_i + z_l;$$

$$z_j := z_j + z_l;$$

if $\circ_l = -$, append

$$z_i := z_i + z_l;$$

$$z_j := z_j - z_l;$$

if $\circ_l = *$, append

$$z_i := z_i + z_l * x_j;$$

$$z_j := z_j + z_l * x_i;$$

if $\circ_l = /$, append

$$z_i := z_i + z_l / x_j;$$

$$z_j := z_j - z_l * x_l / x_j^2;$$

Aposteriori interval arithmetic (continue)

For $l = n$ we have unique solution and $z_i = \frac{\partial y(0)}{\partial x_i}$ for $i = 1...n$

Final error value

$$e_y := \sum_{i=1}^{n} (|val(z_i)| + err(z_i))e_i$$

The method was proposed by Yu. Matijasevich [1]

[1] Yu. Matiyasevich, A posteriori interval analysis, *Lecture Notes in Computer Science*, 1985, Vol. 204, pp. 328–334

Error improvement scheme

Input: interval values $x_1, ..., x_n$
Output: interval values $y_1, ..., y_m$ — m may depend on n

Step 1: Compute $(y_1, ..., y_m)$
Step 2: Find $z_{i,j} = \frac{\partial y_i}{\partial x_j}$
Step 3: New error value $err(y_i) = \sum_{j=1}^{n} |z_{i,j}| * err(x_j)$

If the complexity of the initial algorithm $T(x_1, ..., x_n)$, then the new will be $O(mT(x_1, ..., x_n))$

Resulting error value — *asymptotically optimal*.

Auto differentiation

Dynamic generation — virtual machine, special controller

- easy to implement
- need to store commands and all data

Static — have second step code

- write code by hand or create special translator
- better spatial complexity
- faster
- may have larger result interval

Interval libraries

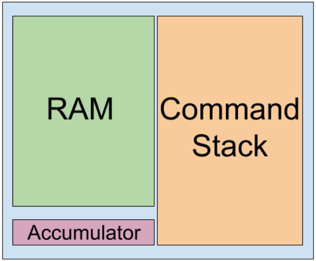
Why use arb? Frederik link.

- INTLAB
- Pascal-XSC, C-XSC
- **Arb**
- MPFI
- Boost interval
- libieeep1788
- ...

Dynamic implementation[1]

Three-adress commands ADD, SUB, MUL, DIV.

Second step commands — NULL, INULL и CORR.



ADD a1 a2 a3 : send to a1 sum of values in a2 и a3 and write in stack:

CORR a3 (1, 0)
CORR a2 (1, 0)
NULL a1

[1] Yu. Matiyasevich, Real numbers and computers (In Russian), *Cybernetics and Computing Machinery*, 1986, Vol.2, pp. 104–133

Example of program

```cpp
int main() {
    IntervalVar x0(ArbInterval(1, 0.01));
    IntervalVar x1(ArbInterval(2, 0.03));
    controller.init();

    IntervalVar x2 = x0 * x0;
    IntervalVar x3 = x1 * x0;
    IntervalVar x4 = x3 / x2;

    IntervalResult x5;
    x5 = x4;

    std::cout << x5 << std::endl;
}
```

Results:

- $x_4 = [2.000 \pm 0.0924]$
- $x_5 = [2.000 \pm 0.0513]$
- $x_3/x2 = x1/x0 = [2.000 \pm 0.0506]$

Example of program – determinant

```
template<class IntervalT>
IntervalT det(Matrix<IntervalT> matrix) {
    size_t n = matrix.nrow();

    gauss_elimination(matrix);

    IntervalT d = IntervalT(1);
    for (size_t i = 0; i < n; ++i)
        d = d * matrix.at(i, i);

    return d;
}

int main() {
    Matrix<IntervalVar> A;
    ... set A values ...
    controller.init();

    IntervalResult d;
    d = det(A);

    std::cout << d << std::endl;
}
```

$$\begin{bmatrix} 4 & 7 & 8 \\ 6 & 4 & 6 \\ 7 & 3 & 10 \end{bmatrix}$$

Error 0.01

Error value:

- Traditional — 10.9
- Dynamic — 2.56

Static implementation

Straight line programs — same as dynamic

Arbitary programs [1]:

- cycles
- conditional statements
- re-assignment
- procedures
- ...

[1] D. Shiriaev, Fast Automatic Differentiation for Vector Processors and Reduction of the Spatial Complexity in a Source Translation Environment, 1993, Ph.D. thesis, Karlsruhe University

Program inversion example

If we have program with cycle

`for i := L to U do S,`

then inverted program

for i := U downto L do S^{-1}

Example: compute $f(x) = x^n$

```
p := 1;
for i := 1 to n do
  p := p * x;
dx := 0; dp := 1;
for i := n downto 1 do
  p := p / x;
  dx := dx + dp * p;
  dp := dp * x;
```

Theoretical complexity

Table: Determinant computation

Method	Time complexity	Spatial complexity
Traditional	$O(n^3)$	$O(n^2)$
Dynamic	$O(n^3)$	$O(n^3)$
Static	$O(n^3)$	$O(n^2)$

Table: System of linear equations

Method	Time Complexity	Spatial complexity
Traditional	$O(n^3)$	$O(n^2)$
Dynamic	$O(n^4)$	$O(n^3)$
Static	$O(n^4)$	$O(n^2)$

Example — det computation

$$\begin{bmatrix} 4 & 7 & 8 \\ 6 & 4 & 6 \\ 7 & 3 & 10 \end{bmatrix}$$

Error 0.01

Type	Value	Error
Optimal	-118	2.1
Traditional	-118	10.9
Dynamic	-118	2.56
Static	-118	4.49

Table: Errors comparision

Experiments — accuracy

Figure: Detarminant

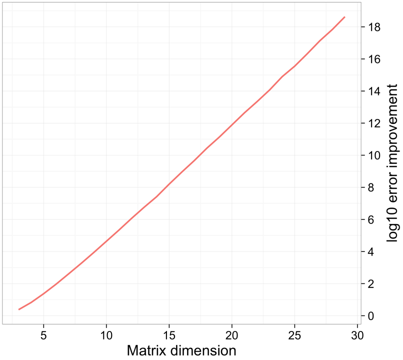
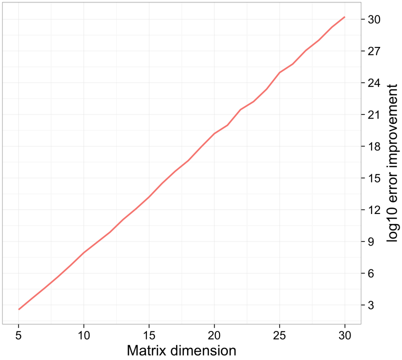


Figure: System of linear equations



Experement — timings

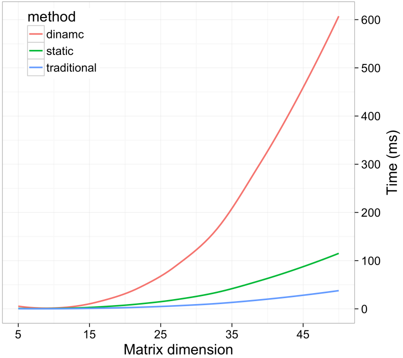


Figure: Detarminant

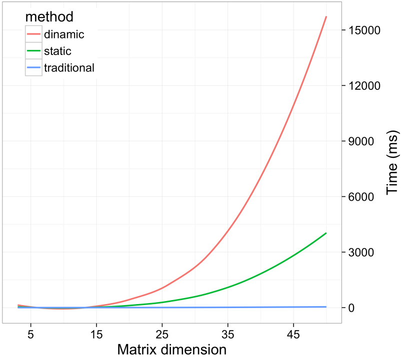


Figure: System of linear equations

Conclusion

- Proposed method give more accurate result but require more resources
- Have only linear slowdown for programs with one variable
- The library can be easily installed and used in an arbitrary program

---

Fork on github — `https://github.com/VladimirGl/apost`