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

- \((v_1, e_1) + (v_2, e_2) = (v_1 + v_2, e_1 + e_2)\)
- \((v_1, e_1) - (v_2, e_2) = (v_1 - v_2, e_1 + e_2)\)
- \((v_1, e_1) \times (v_2, e_2) = (v_1v_2, |v_1|e_2 + |v_2|e_1 + e_1e_2)\)
- \(\frac{(v_1, e_1)}{(v_2, e_2)} = \left(\frac{v_1}{v_2}, \frac{e_1 + \frac{v_1}{v_2} + e_2}{|v_2| - e_2}\right)\)

Problems:

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

Dependency problems...
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.

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.

Asymptotically 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

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

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

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 $\Rightarrow$ generalized method have $O(nT(n))$ time complexity

---

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

\[(x_1 = x_1^0, \ldots, x_n = x_n^0)\];

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

\[
\vdots
\]

\[x_l := x_{i_l} \circ_l x_{j_l};\]

\[
\vdots
\]

\[x_m := x_{i_m} \circ_m x_{j_m};\]

\[z_1 := 0;\]

\[
\vdots
\]

\[z_{m-1} := 0;\]

\[z_m := 1;\]

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 \(\circ_l = \ast\), append

\[z_i := z_i + z_l \ast x_j;\]

\[z_j := z_j + z_l \ast x_i;\]

if \(\circ_l = /\), append

\[z_i := z_i + z_l / x_j;\]

\[z_j := z_j - z_l \ast x_l / x_j^2;\]
For $l = n$ we have unique solution and $z_i = \frac{\partial y(0)}{\partial x_i}$ for $i = 1...n$

Final error value

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

The method was proposed by Yu. Matijasevich

---

Error improvement scheme

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

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

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

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
Why use arb? Frederik link.

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

Interval libraries
Dynamic implementation

Three-address 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

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:
- $x_4 = [2.000 \pm 0.0924]$
- $x_5 = [2.000 \pm 0.0513]$
- $x_3/x_2 = x_1/x_0 = [2.000 \pm 0.0506]$
Example of program – determinant

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

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;
}

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

    IntervalResult d;
    d = det(A);

    cout << d << endl;
}
```
Static implementation

Straight line programs — same as dynamic

Arbitrary programs $^1$:
- cycles
- conditional statements
- re-assignment
- procedures
- ...

Program inversion example

If we have program with cycle

\[
\text{for } i := L \text{ to } U \text{ do } S,
\]

then inverted program

\[
\text{for } i := U \text{ downto } L \text{ do } S^{-1}
\]

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

\[
p := 1;
\]
\[
\text{for } i := 1 \text{ to } n \text{ do }
\]
\[
p := p \times x;
\]
\[
dx := 0; \ dp := 1;
\]
\[
\text{for } i := n \text{ downto } 1 \text{ do }
\]
\[
p := p \div x;
\]
\[
dx := dx + dp \times p;
\]
\[
dp := dp \times x;
\]
Theoretical complexity

Table: Determinant computation

<table>
<thead>
<tr>
<th>Method</th>
<th>Time complexity</th>
<th>Spatial complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional</td>
<td>$O(n^3)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Dynamic</td>
<td>$O(n^3)$</td>
<td>$O(n^3)$</td>
</tr>
<tr>
<td>Static</td>
<td>$O(n^3)$</td>
<td>$O(n^2)$</td>
</tr>
</tbody>
</table>

Table: System of linear equations

<table>
<thead>
<tr>
<th>Method</th>
<th>Time Complexity</th>
<th>Spatial complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional</td>
<td>$O(n^3)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Dynamic</td>
<td>$O(n^4)$</td>
<td>$O(n^3)$</td>
</tr>
<tr>
<td>Static</td>
<td>$O(n^4)$</td>
<td>$O(n^2)$</td>
</tr>
</tbody>
</table>
Example — det computation

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

Error 0.01

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td>-118</td>
<td>2.1</td>
</tr>
<tr>
<td>Traditional</td>
<td>-118</td>
<td>10.9</td>
</tr>
<tr>
<td>Dynamic</td>
<td>-118</td>
<td>2.56</td>
</tr>
<tr>
<td>Static</td>
<td>-118</td>
<td>4.49</td>
</tr>
</tbody>
</table>

Table: Errors comparison
Experiments — accuracy

Figure: Determinant

Figure: System of linear equations
Experiment — timings

**Figure:** Determinant

**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