

Exact Computation

Theory and Techniques

1. Evaluating One Determinant Expression

The expression is viewed as labelled directed acyclic graph. Each sink node (a node that has no outgoing edges) represents a number. Each internal node represents an operator. When the operator is binary the node has two incoming edges.

This helps to optimise the expression. For example when we need the square root of 2 twice in the expression we only have to compute it once.

2. Error-bounds Versus Precision-bounds

Error-bounds are computed bottom up: The precision of the result depends on the precision of the input and a global precision bound (without a global precision bound we would never stop while computing the square root of 2). The precision of the result and the algorithms are deterministic.

Example: Our input are two numbers a and b both with an absolute error of ± 0.1 . When we compute the expression $c = a + b$ c would have an absolute error of ± 0.2 .

Precision-bounds are computed top down. The user specifies the desired precision of the result. When c is supposed to have an error of ± 0.2 and we once again compute $c = a + b$ there are several possibilities for the errors of a and b. Both could have an error of ± 0.1 . If we already know $|a| < 0.1$ then b can have an error of ± 0.18 and $c = b$.

Therefore precision-bounds are not deterministic.

3. Error-Bounds

3.1 Number Representation (Notation)

BigFloats are used as an approximation to the number.

$$b = 1212/343434 \approx 0.35 * 10^{-2}$$

When bigFloats are used with error-bounds the following notation is used:

$$b' = (0.35 \pm 0.01) * 10^{-2} = (35, -2, 1)$$

In general, the triple (f, e, d) of integers represents the range $\langle f \pm d \rangle * B^e$.

The base B is usually 2 (for efficiency reasons) or 10 (which is easier to understand for human beings). $\langle n \rangle$ of any integer means that we place the radix point just before the first digit of n.

The number is exact when $d = 0$

3.2 Normalizing Error Digits

$$(35, -2, 1)^2 = (1225, -4, 71)$$

Storing many insignificant digits is a waste of space and time. A number is normalized when f doesn't start with 0 and $0 \leq d < B$. Delayed normalization can improve accuracy.

Un-normalized	Value-range	Normalized
(1000, 0, 10)	0.0990 - 0.1010	(100, 0, 1)
(1000, 0, 11)	0.0989 - 0.1011	(100, 0, 2)
(1001, 0, 10)	0.0991 - 0.1011	(100, 0, 2)
(1004, 0, 19)	0.0985-0.1023	(100, 0, 3)

3.3 Algorithms for Maintaining bigFloat Ranges

3.3.1 Addition

The following is a safe algorithm for addition:

1. Align the least significant bits. 2. Add the digits and error values. 3. Normalize
 $(123, 0, 1) + (246, -2, 3) = (12300, 0, 100) + (246, -2, 3) = (12546, 0, 103)$

Normalized: $(125, 0, 2)$

The problem is far more digits are computed than would be necessary. These digits will disappear with normalization.

One solution is the following algorithm:

-Find the least significant bit (global precision bound, magnitude of both summands) and don't use smaller bits.

$(123, 0, 1) + (246, -2, 3) = (123, 0, 1) + (2, -2, 1) = (125, 0, 2)$

3.3.2 Multiplication

$(f_1, e_1, d_1) * (f_2, e_2, d_2) = (f_1 * f_2, e_1 + e_2, f_1 * d_2 + f_2 * d_1 + e_1 * e_2) \quad f_1, f_2 \geq 0$

If f_1 and f_2 (both n digits) are not exact \Rightarrow the result doesn't have $2n$ but only n significant digits. \Rightarrow Don't compute more digits than the significant ones (similar like in Addition).

3.3.3 Square Root

$\alpha = \sqrt{\beta}$ the Newton iteration $\beta_{i+1} = \frac{1}{2}(\beta_i / (\alpha + \beta_i))$ β_1, β_2, \dots converges toward $\sqrt{\beta}$

β_i would be exact, but α might be not exact and operations can't be done accurate. But still β lies between β_i and β_{i+1} .

4. Precision-bounds

4.1 Composite Precision Bounds

X approximates x

absolute precision(AP): $|X - x| \leq 2^{-a}$

relative precision(RP): $|X - x| \leq |x| 2^{-r}$

composite precision(CP): $|X - x| \leq \max(2^{-a}, |x| 2^{-r})$

written: $X = x[a, r]$

$CP \Rightarrow AP \quad r = \infty \quad CP \Rightarrow RP \quad a = \infty$

4.2 Algorithms

Most significant bit (MSB) $m_X := \lfloor \log|x| \rfloor$

$m_X[a, r] = M_X \Leftrightarrow 2^{M_X} \leq |x| < 2^{1+M_X} + \max\{2^{e-r}, 2^a\}$

Computing M_X is expensive

Often a lower bound for M_X is sufficient

5. Theory of Exact Computation

Computation of semi-algebraic problems is theoretically possible.

An algebraic number (AN) is any complex root of a polynomial (with integer coefficients)

$\sqrt{2}$ is AN, since it is root of $x^2 - 2$

Isolating interval representation: $\sqrt{2} = (x^2 - 2, [1, 2])$

Any algebraic number can be represented as the root of a polynomial in a certain interval.

5.1 Semi-algebraic Predicates

$\Phi(x_1, \dots, x_n)$ is a Boolean combination of $F(x_1, \dots, x_n) \quad p \quad 0 \quad p \in \{=, <, >, \leq, \geq\}$

$\Phi_0(x, y) : (x^2 + y^2 - 1 < 0) \vee (x + y - 5 = 0)$ Is the union of an open disc and a straight line.

5.2 Tarski Formula

A first-order formula based on semi-algebraic predicates without free variables is a true Tarski sentence.

Example: $(\forall x) (\exists y) \Phi_0(x, y)$

Sets defined by Tarski formulas are semi-algebraic.

5.3 Practical Use?

Theorem: *A semi-algebraic problem can be solved in double-exponential time on a Turing-machine.*

Mainly of theoretical interest

Practical use if some instances can be computed quickly (double exponential time is worst case)

5.4 Theory of Root Bounds

Comparing values is a basic operation

It is sufficient to compare to zero \Leftrightarrow determine the sign of an expression

How can we know a number is 0 without looking at infinite signs and without using „some heuristic cut-off epsilon value“?

h is a height bound on a

The height of an algebraic number is the maximum value of the coefficients in its minimal polynomial

Cauchy's bound says:

a is non-zero iff $|a| > 1 / (1+h)$

Evaluate a to an AP of $1+\log(1+h)$ to determine the sign