# Probabilistic Floating-Point Round-Off Analysis via Concentration Inequalities

YICHEN TAO, University of Michigan, USA
HONGFEI FU*, Shanghai Jiao Tong University, China
JIAWEI CHEN, University of Michigan, USA
JEAN-BAPTISTE JEANNIN, University of Michigan, USA

Floating-point round-off errors are ubiquitous in numerically intensive programs arising in fields such as scientific computing and optimization. As floating-point errors potentially lead to unexpected and catastrophic program failures, one must derive guaranteed round-off thresholds to ensure the correctness of these programs. However, deterministic round-off thresholds tend to be too conservative to be usable in practice, since they often involve large round-off errors that occur with small probability. Probabilistic thresholds relax deterministic ones by specifying that the probability of the round-off error exceeding a threshold is below a given confidence.

In this work, we propose a novel approach to probabilistic round-off analysis, by applying concentration inequalities over the Taylor expansion from FPTaylor (TOPLAS 2018). A major obstacle in applying concentration inequalities is that the Taylor expansion involves absolute value operators that make the calculation of the expected values of the first order partial differential terms difficult. Our first step to overcome this obstacle is a sound over-approximation that removes the absolute value operators in polynomial expressions. Then, we show how to handle fractional expressions by a transformation into polynomial case. Finally, we show how to improve our approach with range partitioning. Our approach is scalable since the key computational part is the calculation of expected values of polynomial expressions with independent variables, for which the linear and independence properties of expectation boost the computation. Experimental results show that our approach is orders of magnitude more time efficient, while producing thresholds with comparable precision against the state of the art.

**ACM Reference Format:**
Yichen Tao, Hongfei Fu, Jiawei Chen, and Jean-Baptiste Jeannin. 2026. Probabilistic Floating-Point Round-Off Analysis via Concentration Inequalities. 1, 1 (May 2026), 38 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 Introduction

Most non-integer numerical computations in computers are performed using floating-point arithmetic. However, floating-point arithmetic is inherently imprecise due to its finite precision [19], which introduces round-off errors at almost every execution step of a floating-point program. While these errors are typically small in a single floating-point operation, the accumulation of round-off errors in numerically intensive programs can be non-negligible, potentially leading to catastrophic outcomes. Ensuring that round-off errors in numerically intensive programs are under control necessitates rigorous round-off analysis, i.e., deriving guaranteed thresholds for the floating-point error incurred in numerical computation.

A typical example for the round-off analysis problem is as follows. Consider the computation of $f(x_1, x_2, x_3) = x_1x_2 + x_3$, where the input variables $x_1$, $x_2$ and $x_3$ are floats confined within the interval $[-1, 1]$. Due to the finite precision nature of floating-point representation in calculating

*Corresponding author

Authors' Contact Information: Yichen Tao, ychtao@umich.edu, University of Michigan, Ann Arbor, Michigan, USA; Hongfei Fu, jt002845@sjtu.edu.cn, Shanghai Jiao Tong University, Shanghai, China; Jiawei Chen, chenjw@umich.edu, University of Michigan, Ann Arbor, Michigan, USA; Jean-Baptiste Jeannin, jeannin@umich.edu, University of Michigan, Ann Arbor, Michigan, USA.

2026. ACM XXXX-XXXX/2026/5-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

the product $x_1x_2$ and the sum $x_1x_2 + x_3$, the computed result deviates from the mathematically exact value obtained under real-number arithmetic. A typical task of deterministic floating-point round-off analysis is to determine an upper bound on this deviation that holds for any valid input.

Rigorous analysis of floating-point round-off errors has been extensively studied in the literature. Deterministic analysis targets guaranteed thresholds for the numerical error under any program input, and has been studied in multiple existing works. Tools like Gappa [16] and PRECiSA [33] adopt an abstraction-based approach, while FPTaylor [30] and Real2Float [26] formulate it as an optimization problem. Alternatively, probabilistic analysis tightens these sometimes overly-conservative thresholds, limiting them to only the most likely input ranges.

In this paper, we consider the *probabilistic* analysis of floating-point errors: its objective is to derive a threshold such that the probability that the round-off error does not exceed the threshold is at least a given confidence level close to 1. Probabilistic analysis is particularly motivated by the overly pessimistic nature of deterministic analysis, since it accounts for worst-case inputs, even those with negligible probability [32, Section 4.7]. In contrast, probabilistic analysis is less impacted by these rare input scenarios, providing a more informative and practical insight when input distributions are known. Recall the previous example of computing $x_1x_2 + x_3$ in floating-point arithmetic. If additional information about the distribution of the input variables is provided, such as the assumption that they are uniformly distributed on $[-1, 1]$, it is possible to obtain a "probabilistic threshold" for the round-off error, which is a threshold that is not exceeded with probability of at least a given confidence value (e.g., 0.99). A typical application motivating probabilistic analysis is the numerical computation in GPS sensor data. Such data are often modeled by normal distributions with unbounded or large support, which renders worst-case analysis too conservative, and therefore, uninformative. In contrast, probabilistic analysis focuses on the bulk of probability mass, yielding thresholds that are more useful in practice, especially for distributions with large support.

The PAF tool [11] is the current state of the art in probabilistic round-off analysis. PAF traverses the abstract syntax tree of a floating-point expression, calculating probabilistic ranges and error distributions for intermediate results, and using Dempster-Shafer structures (DS-structures) [18], an interval-based data structure to represent the probability distributions. It uses symbolic affine arithmetic for error propagation and computes conditional round-off errors by maximizing symbolic error forms over the input variable ranges. Although interval abstraction achieves arbitrary accuracy as the number of intervals approaches infinity, the combinatorial explosion from maintaining a large number of intervals hinders the efficiency of the round-off analysis. For instance, in our experimental evaluation, it takes PAF more than two hours to analyze an expression with seven operations on an 18-core machine. An earlier tool, PrAn [25], discretizes the input distribution into subdomains, applies probabilistic affine arithmetic to analyze each subdomain independently, and then merges the resulting error distributions to compute a global probabilistic error bound, which is extracted as a refined probabilistic error guarantee. In general, PrAn is faster than PAF, but at the cost of considerably less accurate thresholds. Hence, conducting accurate and efficient probabilistic round-off analysis remains a challenge.

We address the aforementioned challenge over arithmetic expressions with addition, subtraction, multiplication and division by applying concentration inequalities to the probabilistic round-off analysis. Our detailed contributions are as follows:

- We introduce a novel approach to probabilistic round-off analysis for polynomial arithmetic expressions with addition, subtraction, multiplication, and division with constant denominators, given specified distributions of input variables. The main idea is to apply concentration inequalities to the Taylor expansion used by FPTaylor. A key technical contribution in our

approach is a sound relaxation (called *positive-negative decomposition*) of polynomial expressions with absolute values into those without absolute values to avoid expensive computer algebra for computing positive and negative regions of polynomials.
- We show how to transform the probabilistic round-off analysis of fractional expressions into one of polynomials, so that our approach handles division with non-constant denominators.
- We show how one can use range partition to further improve the accuracy of our approach, and prove the correctness of our range partition refinement.
- We implement our algorithm in a prototype named ProbTaylor in OCaml. Our prototype handles uniform, truncated normal, and Laplace distributions for input variables. A technical novelty here is a symbolic approach that efficiently and accurately computes expectation.

Evaluation over benchmarks from PAF, FPBench and two realistic examples shows that: (a) ProbTaylor is significantly more time efficient compared to PAF and PrAn; (b) ProbTaylor provides thresholds whose accuracy is comparable to or better than those by PAF and PrAn, and in some cases at least an order of magnitude better; (c) With a moderate range for input variables, the probabilistic thresholds derived from ProbTaylor are significantly tighter than the deterministic ones generated by the deterministic tool FPTaylor; (d) Our approach is potentially scalable in handling floating point expressions with many variables and operations.

## 2 Preliminaries

In this Section, we present some necessary background regarding floating-point arithmetic, first-order Taylor expansions and probability theory.

### 2.1 Floating-Point Arithmetic

In this work, we limit our focus to floating-point arithmetic specified by the IEEE 754 standard [20]. A binary floating-point number is defined by a triple consisting of a sign bit (sgn), significand bits (sig) and exponent bits (exp), whose numerical value can be expressed as $(-1)^{\mathtt{sgn}} \times \mathtt{sig} \times 2^{\mathtt{exp}}$. When operations are "correctly rounded" and there is no overflow or exception, we have the following model for floating-point operations:

$$x \circ_{fl} y = (x \circ y)(1 + e) + d, \text{ with } |e| \leq \epsilon \text{ and } |d| \leq \delta$$

where $\circ \in \{+, -, \times, /\}$ is a basic operation in real numbers, and $\circ_{fl}$ is its floating-point counterpart. The error variables $e$ and $d$ account for relative error due to rounding, and absolute error due to underflow, respectively. Their upper bounds $\epsilon$ and $\delta$ depend on the floating-point format under which the computation is carried out and the choice of rounding operator. For instance, if the computation is done in single precision, where sig has 23 bits and exp has 8 bits, by rounding to nearest, we have $\epsilon = 2^{-24}$ and $\delta = 2^{-150}$. Beyond these bounds, the exact values of $e$ and $d$ are typically considered unknown in round-off analysis.

Given a multivariate function $f(\mathbf{x})$ with variables in a vector $\mathbf{x}$ (bold symbols like $\mathbf{x}$ are used to denote vectors), its floating-point model $\tilde{f}(\mathbf{x}, \mathbf{e}, \mathbf{d})$ can be derived by replacing all operations by the floating-point model mentioned above, and $\mathbf{e}$ and $\mathbf{d}$ are vectors of the error variables, i.e., $\mathbf{e} = (e_1, \ldots, e_k)$ and $\mathbf{d} = (d_1, \ldots, d_k)$. The length $k$ of $\mathbf{e}$ and $\mathbf{d}$, is equal to the number of operations in $f(\mathbf{x})$, i.e., each floating-point operation corresponds to exactly one $(e_i, d_i)$ from $\mathbf{e}$ and $\mathbf{d}$. Note that $|e_i| \leq \epsilon$ and $|d_i| \leq \delta$ for $i = 1, \ldots, k$. For example, the floating-point version of $x \times y + z$ is given by $((x \times y)(1 + e_1) + d_1 + z)(1 + e_2) + d_2$, where $(e_1, d_1)$ arises from the inner multiplication and $(e_2, d_2)$ from the outer addition. It is guaranteed that for some actual error values $(\tilde{\mathbf{e}}, \tilde{\mathbf{d}})$ for $(\mathbf{e}, \mathbf{d})$, the deviated value $\tilde{f}(\mathbf{x}, \tilde{\mathbf{e}}, \tilde{\mathbf{d}})$ is equal to the actual floating-point computation result of $f(\mathbf{x})$. Additionally, it holds that $\tilde{f}(\mathbf{x}, \mathbf{0}, \mathbf{0}) = f(\mathbf{x})$. The magnitude of this deviation at a particular program

input $\mathbf{x}$ can be formalized by the following absolute round-off function $\mathrm{err}(f, \mathbf{x})$:

$$\mathrm{err}(f, \mathbf{x}) := \left| \tilde{f}(\mathbf{x}, \tilde{\mathbf{e}}, \tilde{\mathbf{d}}) - f(\mathbf{x}) \right|.$$

### 2.2 First-Order Taylor Expansion

A sufficiently smooth function can be approximated by its *Taylor expansion* [29], which is a polynomial expression in terms of the function's derivatives at a given point. For a $k$-ary function $f(w_1, \ldots, w_k)$ that is at least twice continuously differentiable on its domain $D \subset \mathbb{R}^k$, its first-order Taylor expansion around a particular point $(a_1, \ldots, a_k) \in D$ can be expressed as

$$f(\mathbf{w}) = f(\mathbf{a}) + \sum_{i=1}^{k} \frac{\partial f}{\partial w_i}(\mathbf{a})(w_i - a_i) + \frac{1}{2} \sum_{i,j=1}^{k} \frac{\partial^2 f}{\partial w_i \partial w_j}(\mathbf{z})(w_i - a_i)(w_j - a_j)$$

for some $\mathbf{z} \in D$, where $\mathbf{w} = (w_1, \ldots, w_k)$ and $\mathbf{a} = (a_1, \ldots, a_k)$.

We follow FPTaylor [30] and apply a first-order Taylor expansion to the floating-point version $\tilde{f}(\mathbf{x}, \mathbf{e}, \mathbf{d})$ from Section 2.1. Henceforth, applying Taylor's theorem to $\tilde{f}(\mathbf{x}, \mathbf{e}, \mathbf{d})$ w.r.t $e_1, \ldots, e_k, d_1, \ldots, d_k$ around zero gives us

$$\tilde{f}(\mathbf{x}, \mathbf{e}, \mathbf{d}) = \tilde{f}(\mathbf{x}, \mathbf{0}, \mathbf{0}) + \sum_{i=1}^{k} \frac{\partial \tilde{f}}{\partial e_i}(\mathbf{x}, \mathbf{0}, \mathbf{0}) e_i + R_2(\mathbf{x}, \mathbf{e}, \mathbf{d}).$$

For sake of brevity, set $y_i \triangleq e_i$ for $i = 1, \ldots, k$ and $y_i \triangleq d_{i-k}$ for $i = k+1, \ldots, 2k$. Then, we have

$$R_2(\mathbf{x}, \mathbf{e}, \mathbf{d}) = \frac{1}{2} \sum_{i,j=1}^{2k} \frac{\partial^2 \tilde{f}}{\partial y_i \partial y_j}(\mathbf{x}, \mathbf{e}', \mathbf{d}') y_i y_j + \sum_{i=1}^{k} \frac{\partial \tilde{f}}{\partial d_i}(\mathbf{x}, \mathbf{0}, \mathbf{0}) d_i$$

for some $\mathbf{e}', \mathbf{d}' \in \mathbb{R}^k$ satisfying $|e_i'| \le \epsilon$ and $|d_i'| \le \delta$ for $i = 1, \ldots, k$. The first-order terms $\frac{\partial \tilde{f}}{\partial d_i}(\mathbf{x}, \mathbf{0}, \mathbf{0}) d_i$ is added to $R_2$ since $\delta \ll \epsilon^2$ for common floating-point formats. As is mentioned before, $\tilde{f}(\mathbf{x}, \mathbf{0}, \mathbf{0}) = f(\mathbf{x})$. Therefore, we have a new expression for $\mathrm{err}(f, \mathbf{x})$, which we may further relax using the triangular inequality:

$$\mathrm{err}(f, D) \le \left| \sum_{i=1}^{k} \frac{\partial \tilde{f}}{\partial e_i}(\mathbf{x}, \mathbf{0}, \mathbf{0}) e_i + R_2(\mathbf{x}, \mathbf{e}, \mathbf{d}) \right| \le \sum_{i=1}^{k} \left| \frac{\partial \tilde{f}}{\partial e_i}(\mathbf{x}, \mathbf{0}, \mathbf{0}) e_i \right| + \left| R_2(\mathbf{x}, \mathbf{e}, \mathbf{d}) \right|.$$

Throughout the paper, we denote the "first-order error term" $\sum_{i=1}^{k} \left| \frac{\partial \tilde{f}}{\partial e_i}(\mathbf{x}, \mathbf{0}, \mathbf{0}) e_i \right|$ by $F(\mathbf{x}, \mathbf{e})$. We call $|R_2(\mathbf{x}, \mathbf{e}, \mathbf{d})|$ the "second-order error term".

### 2.3 Probability Theory

Given a *probability space* $(\Omega, \mathcal{F}, \mathbb{P})$ with sample space $\Omega$, set of events $\mathcal{F}$ and *probability measure* $\mathbb{P} : \mathcal{F} \to [0, 1]$, the *probability* that an event $E \in \mathcal{F}$ happens is denoted $\mathbb{P}[E]$. A *random variable* $X : \Omega \to \mathbb{R}$ is such that the subset $\{\omega \in \Omega \mid X(\omega) \le d\} \in \mathcal{F}$ for any $d \in \mathbb{R}$, and its *expected value* $\mathbb{E}[X]$ is defined as the integral $\int X \, \mathrm{d}\mathbb{P}$. Given a *continuous* random variable $X$, its probability density function (PDF) is a Lebesgue-measurable function $g : \mathbb{R} \to [0, \infty)$ such that $c(X)(t) = \int_{-\infty}^{t} g(x) \, \mathrm{d}x$ for all reals $t$. Given a PDF $g$ of a continuous random variable $X$, we have that the expected value $\mathbb{E}[X]$ is equal to $\int_{-\infty}^{\infty} x \cdot g(x) \, \mathrm{d}x$. Given two events $E_1, E_2 \in \mathcal{F}$ with $\mathbb{P}[E_2] > 0$, the *conditional probability* of $E_1$ given $E_2$ is defined as $\mathbb{P}[E_1 \mid E_2] \triangleq \mathbb{P}[E_1 \cap E_2] / \mathbb{P}[E_2]$. Similarly, for a continuous random variable $X$ and an event $E \in \mathcal{F}$ with $\mathbb{P}[E] > 0$, the *conditional expectation* of $X$ given $E$ is defined by $\mathbb{E}[X \mid E] \triangleq \left( \int_E X \mathrm{d}\mathbb{P} \right) / \mathbb{P}[E]$. See [2, 10, 36] for a formal treatment of these concepts.

*Concentration inequalities* are used for bounding the probability that a random variable deviates largely from its majority part of probability mass. Here, we use Markov's inequality as stated below.

Theorem 2.1 (Markov's Inequality). *If $X$ is a non-negative random variable (i.e., a random variable that always takes non-negative values), then for any constant $a > 0$, $\mathbb{P}[X \ge a] \le \mathbb{E}[X]/a$.*

In this work, we take the $n$-th order higher moment $X^n$ of a random variable $X$ and apply the Markov's inequality to the random variable $X^n$ to obtain $\mathbb{P}[|X| \geq a] = \mathbb{P}[|X|^n \geq a^n] \leq \mathbb{E}[|X|^n]/a^n$. We refer to the order $n$ above as the *analysis order*.

*Problem Statement.* We consider the following probabilistic round-off analysis problem:

- **Input:** (i) an arithmetic expression $f(\mathbf{x})$ that consists of addition, multiplication, subtraction and division over a vector $\mathbf{x}$ of variables, (ii) a map $\mathcal{D}$ that assigns a probability distribution $\mathcal{D}(x)$ to every variable $x$ in the vector $\mathbf{x}$, and (iii) a target confidence level $c \in (0, 1)$.
- **Output:** a threshold $U_c > 0$ such that $\mathbb{P}_{\mathcal{D}}[\mathrm{err}(f, \mathbf{x}) < U_c] > c$, for which the probability measure $\mathbb{P}_{\mathcal{D}}$ corresponds to the independent joint distribution of the distributions $\mathcal{D}(x)$ for every variable $x$ in the vector $\mathbf{x}$.

*Example 2.2.* Consider $f(\mathbf{x}) = x_1 \times x_2 + x_3$, where $x_1, x_2, x_3$ are all uniformly distributed on $[-1, 1]$. Floating-point computations are done in single precision, and thus $\epsilon = 2^{-24}$ and $\delta = 2^{-150}$. We aim to derive a round-off error threshold $U_{0.99}$ for $f(\mathbf{x})$ that holds with probability at least 99%.

*Example 2.3.* Consider $g(\mathbf{x}) = (x_1 \times x_2)/(x_3 + 5)$, where $x_1, x_2, x_3$ are all uniformly distributed on the interval $[-1, 1]$. Floating-point computations are carried out in single precision, and thus $\epsilon = 2^{-24}$ and $\delta = 2^{-150}$. We aim to derive a round-off error threshold $U_{0.99}$ for $g(\mathbf{x})$.

## 3 Overview of Our Results

In this section, we perform probabilistic round-off error analysis on Example 2.2 and Example 2.3 to illustrate our approach to the problem, for polynomials and fractional expressions, respectively. Note that, for the sake of simplicity, the method presented in this Section has some minor differences with our algorithm described in Section 4.

At a high level, our approach to the probabilistic round-off analysis is divided into three steps. The first step is to apply a Taylor expansion to obtain the first-order and second-order error terms as in Section 2.2. The second step is to relax the absolute values in the first-order term to obtain a sound over-approximation without absolute values. The final step is to apply Markov's inequality (Theorem 2.1) to obtain the probabilistic threshold for the round-off error.

### 3.1 Illustration of Example 2.2

Below we detail the steps for performing probabilistic round-off error analysis on Example 2.2.

**Step 1: Applying Taylor Expansion.** We first apply a Taylor expansion to the floating-point model $\tilde{f}(\mathbf{x}, \mathbf{e}, \mathbf{d}) = (x_1x_2(1 + e_1) + d_1 + x_3)(1 + e_2) + d_2$, as shown in Section 2.2. Here $e_1, e_2 \in [-\epsilon, \epsilon]$ correspond to the relative errors associated with normal number results arising from multiplication (resp. addition), and $d_1, d_2 \in [-\delta, \delta]$ correspond to the absolute errors associated with the subnormal number results. Recall that the Taylor expansion gives two error terms: the first-order error term $F(\mathbf{x}, \mathbf{e})$ and the second-order error term $|R_2(\mathbf{x}, \mathbf{e}, \mathbf{d})|$. For Example 2.2, by computing the partial derivatives and summing them up, we derive that $F(\mathbf{x}, \mathbf{e}) = |(x_1x_2) \cdot e_1| + |(x_1x_2 + x_3) \cdot e_2|$, and $|R_2(\mathbf{x}, \mathbf{e}, \mathbf{d})| = d_1 + d_2 + e_1e_2x_1x_2 + e_2d_1$.

Since the round-off error is dominated by first-order term $F(\mathbf{x}, \mathbf{e})$ (typically by several orders of magnitude), we bound the second-order error deterministically with an existing global optimization tool (GELPIA [1] in our implementation), and focus mainly on the first-order error term.

**Step 2: Over-approximation for Absolute Values.** Then, we analyze the probabilistic threshold of the first-order error term $F(\mathbf{x}, \mathbf{e})$. A major obstacle here is the probabilistic evaluation of the absolute values in the first-order term. To overcome this difficulty, we propose a sound over-approximation by introducing fresh variables to separate the positive and negative parts that an input variable can take, which we term *positive-negative (PN) decomposition*.

*PN Decomposition.* For each input variable $x$ in the vector $\mathbf{x}$, we introduce two fresh variables, $x^+$ and $x^-$, which represent the positive part $x^+ := \max\{x, 0\}$ and the negative part $x^- := \max\{-x, 0\}$, respectively. The probability distributions of the new variables $x^+, x^-$ are accordingly calculated as those of $\max\{x, 0\}, \max\{-x, 0\}$ from the distribution $\mathcal{D}(x)$ of the variable $x$. We then replace each variable $x$ with $x = x^+ - x^-$ in every expression within the absolute-value operators from $F(\mathbf{x}, \mathbf{e})$ and expand these expressions into a summation of products of variables. After the expansion, we soundly remove the absolute values by identifying the PN parts in the expanded expressions. A key simplification here is that during the expansion, we can use the equality $x^+ \cdot x^- = 0$ to substantially simplify the resultant expression.

For Example 2.2, we have the relaxation of the term $|(x_1x_2) \cdot e_1|$ in $F(\mathbf{x}, \mathbf{e})$ by:

$$\begin{aligned}|(x_1x_2) \cdot e_1| &= |(x_1^+x_2^+ + x_1^-x_2^-)e_1 - (x_1^+x_2^- + x_1^-x_2^+)e_1| \le |(x_1^+x_2^+ + x_1^-x_2^-)e_1| + |(x_1^+x_2^- + x_1^-x_2^+)e_1| \\ &= (x_1^+x_2^+ + x_1^-x_2^-)|e_1| + (x_1^+x_2^- + x_1^-x_2^+)|e_1| \le (x_1^+x_2^+ + x_1^-x_2^- + x_1^+x_2^- + x_1^-x_2^+)\epsilon.\end{aligned}$$

In the relaxation above, the first equality comes from substituting each $x_i$ with $x_i^+ - x_i^-$, the second inequality is obtained via applying the triangular inequality, the third equality is by safely removing the absolute-value operator due to the PN decomposition, and the final inequality results from choosing $e_i$'s as the maximum value $\epsilon$. The term $|(x_1x_2 + x_3) \cdot e_2|$ can be handled similarly:

$$|(x_1x_2 + x_3) \cdot e_2| \le (x_1^+x_2^+ + x_1^-x_2^- + x_3^+ + x_1^+x_2^- + x_1^-x_2^+ + x_3^-)\epsilon.$$

Combining the two yields the following sound over-approximation for $F(\mathbf{x}, \mathbf{e})$:

$$F(\mathbf{x}, \mathbf{e}) \le (2(x_1^+x_2^+ + x_1^-x_2^- + x_1^+x_2^- + x_1^-x_2^+) + x_3^+ + x_3^-)\epsilon \triangleq p(\mathbf{x}) \cdot \epsilon.$$

Note that we write $p(\mathbf{x})$ instead of $p(x_1^+, x_1^-, x_2^+, x_2^-, x_3^+, x_3^-)$ for brevity.

**Step 3: Applying Markov's inequality.** Finally, we apply Markov's inequality (Theorem 2.1). For a given error threshold $u > 0$, we bound the "threshold violation probability" for the first-order term $F(\mathbf{x}, \mathbf{e})$, i.e., $\mathbb{P}[F(\mathbf{x}, \mathbf{e}) \ge u]$, with analysis order $n = 2$:

$$\mathbb{P}[F(\mathbf{x}, \mathbf{e}) \ge u] = \mathbb{P}[(F(\mathbf{x}, \mathbf{e}))^2 \ge u^2] \le \mathbb{E}[(F(\mathbf{x}, \mathbf{e}))^2]/u^2 \le \epsilon^2 \cdot \mathbb{E}[(p(\mathbf{x}))^2]/u^2.$$

Therefore, to derive a threshold $u$ that guarantees a threshold violation probability below $1 - c$, it suffices to have $\frac{\epsilon^2 \cdot \mathbb{E}[(p(\mathbf{x}))^2]}{u^2} \le 1 - c$, which indicates that $u = \epsilon \cdot \sqrt{\mathbb{E}[(p(\mathbf{x}))^2]/(1-c)}$ is a valid threshold. Setting $c = 0.99$ for Example 2.2, we obtain $u \approx 6.74 \times 10^{-7}$ so that $\mathbb{P}[F(\mathbf{x}, \mathbf{e}) \ge u] \le 0.01$.

Note that our approach simplifies the calculation of the expected values related to the over-approximation $p(\mathbf{x})$ by the linear and independence properties of expectation. That is, our approach calculates the expected value of each finite product $\prod_i x_i^{a_i}$ as $\prod \mathbb{E}(x_i^{a_i})$ and takes a summation over these expected values (possibly multiplied with corresponding coefficients).

For the second-order term, we apply an existing global optimization tool, GELPIA [1], to obtain a deterministic upper bound. In this example, $R_2(\mathbf{x}, \mathbf{e}, \mathbf{d}) = d_1 + d_2 + e_1e_2x_1x_2 + e_2d_1$. By feeding the input ranges of the variables to GELPIA, we get the upper bound $|R_2(\mathbf{x}, \mathbf{e}, \mathbf{d})| \le 3.56 \times 10^{-15} \triangleq \theta$, which is neglectable compared with the first-order threshold $u$ for $c = 0.01$ determined as above. Finally, we conclude that for Example 2.2, $\mathbb{P}[\mathrm{err}(f, \mathbf{x}) \ge 6.74 \times 10^{-7} + \theta] \le 0.01$. A refined version of the analysis of this example with partition of value ranges is presented in Section 4.4.

### 3.2 Illustration of Example 2.3

Below we demonstrate how the three-step procedure can be applied to the fractional expression $g(\mathbf{x}) = (x_1 \times x_2)/(x_3 + 5)$ in Example 2.3. Most details are analogous to those employed in the analysis of Example 2.2, with the exception of certain adjustments in Step 2 to handle denominators.

**Step 1: Applying Taylor Expansion.** The floating-point model of $g(\mathbf{x})$ is derived by replacing each operation with its floating-point model: $\tilde{g}(\mathbf{x}, \mathbf{e}, \mathbf{d}) = \frac{x_1x_2(1+e_1)+d_1}{(x_3+5)(1+e_2)+d_2}(1 + e_3) + d_3$. Subsequently,

the first-order and second-order error terms can be computed by applying Taylor expansion. Specifically, the first-order term is $F(\mathbf{x}, \mathbf{e}) = \frac{|x_1x_2e_1|+|x_1x_2e_2|+|x_1x_2e_3|}{x_3+5}$. The absolute value around $x_3 + 5$ can be dropped since it is positive, because we know that $x_3 \in [-1, 1]$. The second-order term $|R_2(\mathbf{x}, \mathbf{e}, \mathbf{d})|$ is omitted here for brevity. As before, a deterministic upper bound for $|R_2(\mathbf{x}, \mathbf{e}, \mathbf{d})|$ is obtained using off-the-shelf tools.

**Step 2: Over-approximation for Absolute Values.** We apply the relaxation technique for polynomials employed over Example 2.2 to the expressions $|x_1x_2e_1|$, $|x_1x_2e_2|$, $|x_1x_2e_3|$ in the numerator of $F(\mathbf{x}, \mathbf{e})$. We obtain $|x_1x_2e_1| \le (x_1^+x_2^+ + x_1^-x_2^- + x_1^+x_2^- + x_1^-x_2^+)\epsilon$, and analogous bounds for the other two expressions. Consequently, we derive the following over-approximation for $F(\mathbf{x}, \mathbf{e})$:

$$F(\mathbf{x}, \mathbf{e}) \le 3(x_1^+x_2^+ + x_1^-x_2^- + x_1^+x_2^- + x_1^-x_2^+)\epsilon/(x_3 + 5).$$

Define $p(\mathbf{x}) \triangleq 3(x_1^+x_2^+ + x_1^-x_2^- + x_1^+x_2^- + x_1^-x_2^+)$ and $Q(\mathbf{x}) \triangleq x_3 + 5$, we have $F(\mathbf{x}, \mathbf{e}) \le p(\mathbf{x}) \cdot \epsilon/Q(\mathbf{x})$.

**Step 3: Applying Markov's inequality.** Before applying Markov's inequality, we first perform the following transformation. For a fixed threshold $u$, since $Q(\mathbf{x}) > 0$, we have

$$\mathbb{P}[F(\mathbf{x}, \mathbf{e}) \ge u] \le \mathbb{P}\left[p(\mathbf{x}) \cdot \epsilon/Q(\mathbf{x}) \ge u\right] = \mathbb{P}[p(\mathbf{x}) \cdot \epsilon - Q(\mathbf{x}) \cdot u > 0].$$

Denote $p(\mathbf{x}) \cdot \epsilon - Q(\mathbf{x}) \cdot u$ by $K_u(\mathbf{x})$ and define $\mu = \mathbb{E}[K_u(\mathbf{x})]$. When $\mu < 0$ [1] and for analysis order $n = 2$, the relaxation proceeds as

$$\mathbb{P}[K_u(\mathbf{x}) \ge 0] = \mathbb{P}[K_u(\mathbf{x}) - \mu \ge -\mu] \le [(K_u(\mathbf{x}) - \mu)^2 \ge \mu^2] \le \mathbb{E}[(K_u(\mathbf{x}) - \mu)^2]/\mu^2,$$

where the final inequality follows by an application of Markov's inequality. The validity of the preceding steps is established in more detail in Theorem 4.6.

Therefore, it suffices to have a threshold $u$ where $\mathbb{E}[(K_u(\mathbf{x}) - \mu)^2]/\mu^2 \le 1 - c$. Plugging in the previous definitions and values, we have $\mu = 0.75\epsilon - 5u$, the inequality can be simplified into $(u^2/3 - 11\epsilon^2/48)/(9\epsilon^2/16 - 15u\epsilon/2 + 25u^2) < 0.01$. The largest value of $u$ satisfying this inequality would be $u^* \approx 7.68 \times 10^{-8}$.

By providing the second-order error term $|R_2(\mathbf{x}, \mathbf{e}, \mathbf{d})|$ along with the variable ranges to GELPIA, we obtain a deterministic upper bound: $|R_2(\mathbf{x}, \mathbf{e}, \mathbf{d})| \le 1.47 \times 10^{-14} \triangleq \delta$. Consequently, for Example 2.3, we conclude that the total round-off error satisfies $\mathbb{P}[\mathrm{err}(g, \mathbf{x}) \ge 7.68 \times 10^{-8} + \delta] \le 0.01$.

## 4 Threshold Synthesis Algorithms

In this section, we present our algorithms to address the probabilistic floating-point analysis problem (cf. the end of Section 2). A sequence of algorithms are presented, each building on its predecessor to either incorporate a wider range of inputs, or improve the precision of the analysis. We first demonstrate the positive-negative (PN) decomposition technique (Section 4.1), which relaxes the absolute values and eliminates explicit occurrences of error variables. We then describe two algorithms for handling division-free input functions (functions that do not involve division by non-constant expressions) in Section 4.2: the *Naive Markov* (NM) algorithm and the *Central-Moment-Based* (CMB) algorithm. The NM algorithm (Section 4.2.1) performs a direct application of Markov's inequality following the PN decomposition, offering a lightweight analysis. The CMB algorithm (Section 4.2.2) improves upon the NM algorithm by applying Markov's inequality with the central moment. This algorithm is extended to handle fractional expressions involving non-constant denominators (Section 4.3). Moreover, the CMB algorithm enables further refinement via range partition (Section 4.4).

We introduce some terminologies for this section. A *term* is a product of the form $cx_{j_1}^{a_1} \dots x_{j_m}^{a_m}$, where $c$ is a constant that acts as the coefficient, the $a_i$'s are non-negative integers that act as

[1] The condition that $\mu < 0$ can be guaranteed by the values of $u$ we have. This will be explained in more details in Section 4.3.

the exponents, and each $x_{j_i}$ is a variable in the vector $\mathbf{x}$. Given a term $cx_{j_1}^{a_1} \dots x_{j_m}^{a_m}$, the product $x_{j_1}^{a_1} \dots x_{j_m}^{a_m}$ is called the *monomial part* of the term, and each $x_{j_i}^{a_i}$ in the monomial is called a *factor*. By convention, terms have non-zero coefficients. An arithmetic expression over the variables $\mathbf{x}$ is in *polynomial form* if it is a finite sum $\sum_i t_i$ where each $t_i$ is a term.

### 4.1 Positive-Negative Decomposition

Given $k$ polynomial functions $h_1(\mathbf{x}), \dots, h_k(\mathbf{x})$, the objective of the PN decomposition algorithm is to derive a relaxation of the sum $\sum_{i=1}^{k} |h_i(\mathbf{x})e_i|$ that eliminates both the absolute value operations and the explicit appearance of error variables $e_i$ (the relative error terms defined in Section 2.1). This relaxation of the summation is used in subsequent steps; a precise computation would otherwise necessitate expensive computer algebra for identifying the individual positive and negative regions of each absolute value's argument.

The central idea of our PN decomposition algorithm is to introduce a pair of fresh variables $x^+, x^-$ for each existing variable $x$, defined by $x^+ := \max\{x, 0\}$ and $x^- := \max\{-x, 0\}$ (here we abuse the notation so that $x, x^+, x^-$ also refer to the random variables they indicate). Two obvious properties follow: (a) $x = x^+ - x^-$; (b) $x^+ \geq 0$ and $x^- \geq 0$. We refer to $x$ as the *original variable*, $x^+$ as the *positive component*, and $x^-$ the *negative component*. In addition, the identity $x^+ \cdot x^- = 0$ holds for all $x$, which serves as a useful simplification property throughout the algorithm.

At a high level, the PN decomposition targets variables $x$ whose distribution spans both positive and negative values, i.e., variables that are not always non-negative or non-positive. For such variables, odd powers $x^n$ are rewritten using the identity $x^n = x^{n-1}(x^+ - x^-)$, which can be further simplified to $(x^+)^n - (x^-)^n$, leveraging the non-negativity of $x^{n-1}$. For $x$ always non-negative, no transformation is needed; for $x$ always non-positive, $x^n$ is replaced by $(-x^-)^n$.

The pseudocode of our PN decomposition algorithm is given in Algorithm 1. The algorithm takes as input a list of $k$ polynomial functions $h_1(\mathbf{x}), \dots, h_k(\mathbf{x})$, and outputs a polynomial expression $p(\mathbf{x})$ that satisfies $\sum_{i=1}^{k} |h_i(\mathbf{x})e_i| \leq p(\mathbf{x}) \cdot \epsilon$ for all possible values of $\mathbf{x}$ and $\mathbf{e}$. Note that the polynomial $p$ explicitly depends on the positive and negative components of the variables in the vector $\mathbf{x}$, but since they depend on $\mathbf{x}$, we use $p(\mathbf{x})$ as our notation for simplicity.

Algorithm 1 first accepts the input polynomials and initializes the output $p(\mathbf{x})$. Then, the for loop at line 2 processes each $h_i$. During each loop iteration, lines 4 – 10 replace each factor $x^a$ in $h_i$ by its PN decomposition as mentioned previously. Lines 12 – 20 collect the positive parts of the $h_i$'s into $h_i^+$ and the negative parts into $h_i^-$. Finally, line 21 takes the over-approximation of the original $|h_i|$ as $h_i^+ + h_i^-$ and adds it into the current over-approximation $p(\mathbf{x})$. After the loop, the returned polynomial $p(\mathbf{x})$ over-approximates $\sum_{i=1}^{k} |h_i(\mathbf{x})e_i|$. The correctness is given in Theorem 4.1.

Theorem 4.1. *Let $p(\mathbf{x})$ be the output of Algorithm 1. Then for all possible values of $\mathbf{x}$ and $\mathbf{e}$, $\sum_{i=1}^{k} |h_i(\mathbf{x})e_i| \leq p(\mathbf{x}) \cdot \epsilon$ holds.*

Proof Sketch (full proof in Appendix B). After the execution reaches line 12, $h_i(\mathbf{x})$ remains equivalent to the original input expression, as substituting $x_j^a$ with $x_j^{a-1}(x_j^+ - x_j^-)$ preserves the equality. Besides, in each term in $h_i(\mathbf{x})$, all factors ($x_j^+$, $x_j^-$, or even powers) are non-negative, ensuring non-negative monomial parts, and thus, all terms in $h_i(\mathbf{x})$ have the same sign as its coefficient.

The decomposition of $h_i(\mathbf{x})$ into $h_i^+(\mathbf{x})$ and $h_i^-(\mathbf{x})$ after the for loop at line 14 can be proved to have two properties similar to the decomposition of individual variables: (a) both $h_i^+(\mathbf{x})$ and $h_i^-(\mathbf{x})$ are non-negative; (b) $h_i(\mathbf{x}) = h_i^+(\mathbf{x}) - h_i^-(\mathbf{x})$.

**Algorithm 1** Positive-Negative Decomposition

**Input:** A list of $k$ polynomial functions $h_1(\mathbf{x}), \ldots, h_k(\mathbf{x})$
**Output:** A polynomial expression $p(\mathbf{x})$ such that $\forall \mathbf{x}, \mathbf{e}, \sum_{i=1}^{k} |h_i(\mathbf{x})e_i| \leq p(\mathbf{x}) \cdot \epsilon$

$p(\mathbf{x}) \leftarrow 0$
**for** $i \leftarrow 1$ to $k$ **do**
  Expand $h_i(\mathbf{x})$ into polynomial form
  **for** each factor $x_j^a$ in each term in $h_i(\mathbf{x})$ **do**
    **if** $a$ is odd and $x_j$'s range spans over positive and negative values **then**
      Replace $x_j^a$ by $(x_j^+)^a - (x_j^-)^a$
    **end if**
    **if** $a$ is odd and $x_j$ is always non-positive **then**
      Replace $x_j^a$ by $-(x_j^-)^a$
    **end if**
  **end for**
  Expand $h_i(\mathbf{x})$ again into polynomial form
  Initialize $h_i^+(\mathbf{x}) \leftarrow 0$; $h_i^-(\mathbf{x}) \leftarrow 0$
  **for** each term $t$ in $h_i(\mathbf{x})$ **do**
    **if** term $t$ has positive coefficient **then**
      $h_i^+(\mathbf{x}) \leftarrow h_i^+(\mathbf{x}) + t$
    **else**
      $h_i^-(\mathbf{x}) \leftarrow h_i^-(\mathbf{x}) - t$
    **end if**
  **end for**
  $p(\mathbf{x}) \leftarrow p(\mathbf{x}) + h_i^+(\mathbf{x}) + h_i^-(\mathbf{x})$
**end for**
**Return** $p(\mathbf{x})$

Using the triangular inequality, $|h_i(\mathbf{x})e_i| = |h_i^+(\mathbf{x})e_i - h_i^-(\mathbf{x})e_i| \leq |h_i^+(\mathbf{x})e_i| + |h_i^-(\mathbf{x})e_i|$. Since $h_i^+(\mathbf{x})$ and $h_i^-(\mathbf{x})$ are non-negative, this simplifies to $|h_i(\mathbf{x})e_i| \leq (h_i^+(\mathbf{x}) + h_i^-(\mathbf{x}))\epsilon$. Summing over all $i$, $F(\mathbf{x}, \mathbf{e}) = \sum_{i=1}^{k} |h_i(\mathbf{x})e_i| \leq \sum_{i=1}^{k} (h_i^+(\mathbf{x}) + h_i^-(\mathbf{x}))\epsilon = p(\mathbf{x}) \cdot \epsilon$, as required. □

Remark 1. *We choose to restrict the input functions to the PN decomposition algorithm to division-free expressions only. The algorithm expands each of the input functions into polynomial form, and subsequent steps (such as selectively transforming only odd-degree factors) operate on the granularity of individual terms and do not naturally extend to more general algebraic structures.*

### 4.2 Algorithms for Division-Free Expressions

By feeding polynomials $h_i(\mathbf{x}) = \frac{\partial \tilde{f}}{\partial e_i}(\mathbf{x}, \mathbf{0}, \mathbf{0})$ $(1 \leq i \leq k)$ from a division-free first-order error term $F(\mathbf{x}, \mathbf{e}) = \sum_{i=1}^{k} \left| \frac{\partial \tilde{f}}{\partial e_i}(\mathbf{x}, \mathbf{0}, \mathbf{0}) e_i \right|$ as input to Algorithm 1, one obtains a sound polynomial over-approximation $p(\mathbf{x})$ for $F(\mathbf{x}, \mathbf{e}) = \sum_{i=1}^{k} \left| \frac{\partial \tilde{f}}{\partial e_i}(\mathbf{x}, \mathbf{0}, \mathbf{0}) e_i \right|$. By using Algorithm 1 as a subroutine, we further develop two algorithms below that solve the probabilistic round-off analysis problem by synthesizing a threshold that fulfills $\mathbb{P}[\mathrm{err}(f, \mathbf{x}) < U_c] > c$.

*4.2.1 Naive Markov (NM) Algorithm.* The Naive Markov Algorithm, presented in Algorithm 2, computes an error threshold by utilizing an over-approximation produced by Algorithm 1 and a direct application of Markov's inequality in a higher-moment setting.

Given a confidence level $c$, an analysis order $n$ [2], and a division-free arithmetic expression $f(\mathbf{x})$ to be analyzed, the algorithm first computes the partial derivatives $\frac{\partial \tilde{f}}{\partial e_i}(\mathbf{x}, \mathbf{0}, \mathbf{0})$ (lines 1 – 2) and passes it to Algorithm 1 to generate an over-approximation $p(\mathbf{x})$ of $F(\mathbf{x}, \mathbf{e})$ (line 3). Then, the algorithm gets a probabilistic threshold $U_c^{(1)}$ for first-order error term $F(\mathbf{x}, \mathbf{e})$ by applying Markov's inequality with analysis order $n$ over $p(\mathbf{x})$ (line 4, see also Theorem 2.1). Next, the algorithm employs a sound global optimization tool (e.g. GELPIA) to get a deterministic upper bound $U^{(2)}$ for the second-order error term (line 6). Finally, the sum of the two thresholds is returned as the overall threshold $U_c^*$. The soundness of the algorithm is given in Theorem 4.2.

**Algorithm 2** The Naive Markov Algorithm

**Input:** A division-free expression $f(\mathbf{x})$, distribution of the input variables, confidence level $c$, unit round-off $\epsilon$, smallest representable number $\delta$, analysis order $n$

**Output:** A probabilistic upper bound $U_c^*$ for the round-off error $\mathrm{err}(f, \mathbf{x})$

1: Compute the floating-point version of the expression $\tilde{f}(\mathbf{x}, \mathbf{e}, \mathbf{d})$.
2: Compute the partial derivatives $h_i(\mathbf{x}) = \frac{\partial \tilde{f}}{\partial e_i}(\mathbf{x}, \mathbf{0}, \mathbf{0})$.
3: Input the partial derivatives $h_i(\mathbf{x})$ into Algorithm 1 and get the expression for $p(\mathbf{x})$
4: $U_c^{(1)} \leftarrow \epsilon \cdot \sqrt[n]{\mathbb{E}[(p(\mathbf{x}))^n]/(1-c)}$
5: Compute the expression for second-order error term $|R_2(\mathbf{x}, \mathbf{e}, \mathbf{d})|$.
6: Get a deterministic upper bound $U^{(2)}$ for $|R_2(\mathbf{x}, \mathbf{e}, \mathbf{d})|$ using a sound global optimization tool
7: **Return** $U_c^* \leftarrow U_c^{(1)} + U^{(2)}$

Theorem 4.2. *Let $U_c^*$ be the output of Algorithm 2, then $\mathbb{P}[\mathrm{err}(f, \mathbf{x}) \geq U_c^*] \leq 1 - c$.*

Proof. By Theorem 4.1, we have $F(\mathbf{x}, \mathbf{e}) = \sum_{i=1}^{k} \left| \frac{\partial \tilde{f}}{\partial e_i}(\mathbf{x}, \mathbf{0}, \mathbf{0}) e_i \right| \leq p(\mathbf{x}) \cdot \epsilon$. Thus, for any value of $U_c^{(1)} \in \mathbb{R}$, we can relax the first-order error threshold violation probability $\mathbb{P}[F(\mathbf{x}, \mathbf{e}) \geq U_c^{(1)}] \leq \mathbb{P}[p(\mathbf{x}) \cdot \epsilon \geq U_c^{(1)}] = \mathbb{P}[(p(\mathbf{x}))^n \cdot \epsilon^n \geq (U_c^{(1)})^n]$. Since $F(\mathbf{x}, \mathbf{e})$ is non-negative, its over-approximation $(p(\mathbf{x}))^n \cdot \epsilon^n$ is also non-negative, and thus Markov's inequality is applicable. Therefore, $\mathbb{P}[(p(\mathbf{x}))^n \cdot \epsilon^n \geq (U_c^{(1)})^n] \leq \mathbb{E}[(p(\mathbf{x}))^n] \cdot \epsilon^n / (U_c^{(1)})^n$. Subsequently, we substitute $U_c^{(1)} = \epsilon \cdot \sqrt[n]{\mathbb{E}[(p(\mathbf{x}))^n]/(1-c)}$ in the formula to get: $\mathbb{E}[(p(\mathbf{x}))^n] \cdot \epsilon^n / (U_c^{(1)})^n = \mathbb{E}[(p(\mathbf{x}))^n] \cdot \epsilon^n / (\mathbb{E}[(p(\mathbf{x}))^n] \cdot \epsilon^n / (1-c)) = 1 - c$. Combining the derivations above, we arrive at the following inequality $\mathbb{P}[F(\mathbf{x}, \mathbf{e}) \geq U_c^{(1)}] \leq 1 - c$. Since the global optimization tool gives a sound and deterministic upper bound of the second-order error, $|R_2(\mathbf{x}, \mathbf{e}, \mathbf{d})| \leq U^{(2)}$ holds. Therefore, $\mathbb{P}[\mathrm{err}(f, \mathbf{x}) \geq U_c^*] = \mathbb{P}[F(\mathbf{x}, \mathbf{e}) + |R_2(\mathbf{x}, \mathbf{e}, \mathbf{d})| \geq U_c^{(1)} + U^{(2)}] \leq \mathbb{P}[F(\mathbf{x}, \mathbf{e}) \geq U_c^{(1)}] \leq 1 - c$. □

*4.2.2 Central-Moment-Based (CMB) Algorithm.* The NM algorithm determines an optimal threshold $U_c^*$ for a specified confidence level $c$. In contrast, the CMB algorithm adopts an inverse approach – for a fixed threshold $u$, it computes an upper bound $\mathrm{flag}_u$ on the first-order violation probability $\mathbb{P}[F(\mathbf{x}, \mathbf{e}) \geq u]$. The CMB algorithm then seeks a close-to-minimum value of $u$ such that $\mathrm{flag}_u \leq 1 - c$, thereby ensuring the desired probabilistic guarantee. To achieve this, the CMB algorithm employs a binary search over an interval $(\ell, r)$, whose endpoints will be formally introduced later. It is worth noting that $\mathrm{flag}_u$ is monotonically decreasing with respect to $u$ once $u$ exceeds the lower endpoint $\ell$ [3], which justifies the correctness of the binary search procedure.

[2]The analysis order $n$ is a hyper-parameter. Analyses with a larger $n$ usually generate more accurate thresholds at the cost of increased analysis time. The parameter $n$ in Section 4.2.2 and 4.3 serves the same role.

[3]A proof of this monotonicity property is provided in Appendix C.

We first introduce the definition of $\text{flag}_u$ and prove the validity of $\text{flag}_u$ as an upper bound in Theorem 4.3, and then present the complete CMB algorithm in Algorithm 3. The correctness for Algorithm 3 is proved in Theorem 4.4.

Theorem 4.3. *Let $u$ be a fixed threshold. Let $p(\mathbf{x})$ be a function satisfying $F(\mathbf{x}, \mathbf{e}) \le p(\mathbf{x}) \cdot \epsilon$ for all possible values of $\mathbf{x}$ and $\mathbf{e}$. Let $K_u(\mathbf{x}) \triangleq p(\mathbf{x}) \cdot \epsilon - u$, and $\mu \triangleq \mathbb{E}[K_u(\mathbf{x})]$. Suppose that $\mu < 0$. For an even analysis order $n$, if we define $\text{flag}_u \triangleq \mathbb{E}[(K_u(\mathbf{x}) - \mu)^n]/\mu^n$, then the first-order violation probability admits the following upper bound: $\mathbb{P}[F(\mathbf{x}, \mathbf{e}) \ge u] \le \text{flag}_u$.*

Proof. By assumption, we have $F(\mathbf{x}, \mathbf{e}) \le p(\mathbf{x}) \cdot \epsilon$. Therefore, for a fixed value of $u$, $\mathbb{P}[F(\mathbf{x}, e) \ge u] \le \mathbb{P}[p(\mathbf{x}) \cdot \epsilon \ge u] = \mathbb{P}[K_u(\mathbf{x}) \ge 0]$. Denote the event $K_u(\mathbf{x}) \ge 0$ by $E_1$, and the event $|K_u(\mathbf{x}) - \mu| \ge |\mu|$ by $E_2$. We claim that the occurrence of event $E_1$ implies the occurrence of event $E_2$, i.e., $E_1 \subseteq E_2$, based on the following reasoning: whenever $K_u(\mathbf{x}) \ge 0$ ($E_1$ occurs) and given that $\mu < 0$, it holds that $K_u(\mathbf{x}) - \mu \ge 0$, and therefore, $|K_u(\mathbf{x}) - \mu| = K_u(\mathbf{x}) - \mu \ge -\mu = |\mu|$, which satisfies the condition defining event $E_2$. Hence, $\mathbb{P}[K_u(\mathbf{x}) \ge 0] = \mathbb{P}[E_1] \le \mathbb{P}[E_2] = \mathbb{P}[|K_u(\mathbf{x}) - \mu| \ge \mu] = \mathbb{P}[(K_u(\mathbf{x}) - \mu)^n \ge \mu^n]$. Then we apply Markov's inequality (applicable since $n$ is even and thus $\mu^n > 0$) and get $\mathbb{P}[(K_u(\mathbf{x}) - \mu)^n \ge \mu^n] \le \mathbb{E}[(K_u(\mathbf{x}) - \mu)^n]/\mu^n = \text{flag}_u$. Combining the derivations above would lead us to the desired conclusion. □

**Algorithm 3** The Central-Moment-Based Algorithm

**Input:** A division-free expression $f(\mathbf{x})$, distribution of the input variables, confidence level $c$, unit round-off $\epsilon$, smallest representable number $\delta$, and even analysis order $n$

**Output:** A probabilistic upper bound $U_c^*$ for the round-off error $\text{err}(f, \mathbf{x})$

1: Compute the floating-point version of the expression $\tilde{f}(\mathbf{x}, \mathbf{e}, \mathbf{d})$.
2: Compute the partial derivatives $h_i(\mathbf{x}) = \frac{\partial \tilde{f}}{\partial e_i}(\mathbf{x}, \mathbf{0}, \mathbf{0})$.
3: Input the partial derivatives $h_i(\mathbf{x})$ into Algorithm 1 and get the expression for $p(\mathbf{x})$
4: $\ell \leftarrow \mathbb{E}[p(\mathbf{x})] \cdot \epsilon$
5: $r \leftarrow \epsilon \cdot \sqrt[n]{\mathbb{E}[(p(\mathbf{x}))^n]/(1-c)}$
6: **if** $\text{flag}_r < 1 - c$ **then**
7: Use binary search to find the close-to-minimum $U_c^{(1)} \in (\ell, r)$ such that $\text{flag}_{U_c^{(1)}} \le 1 - c$
8: **else**
9: $U_c^{(1)} \leftarrow r$
10: **end if**
11: Get a deterministic upper bound $U^{(2)}$ for $|R_2(\mathbf{x}, \mathbf{e}, \mathbf{d})|$ using a sound global optimization tool
12: **Return** $U_c^* \leftarrow U_c^{(1)} + U^{(2)}$

Theorem 4.4. *Let $U_c^*$ be the output of Algorithm 3, then $\mathbb{P}[\text{err}(f, \mathbf{x}) \ge U_c^*] \le 1 - c$.*

Proof. As established in the proof of Theorem 4.2, $p(\mathbf{x})$ computed in line 3 should satisfy $F(\mathbf{x}, \mathbf{e}) \le p(\mathbf{x}) \cdot \epsilon$. Note that the initial value of $r$ coincides with the value of $U_c^{(1)}$ produced by Algorithm 2. To verify that $U_c^{(1)}$ is a valid threshold, we take a closer look at the two branches of the conditional statement in line 6:

- When $\text{flag}_r < 1 - c$, the `then` branch is taken. In this case, the value of $\ell = \mathbb{E}[p(\mathbf{x})] \cdot \epsilon$ is chosen so that for any candidate threshold $u \in (\ell, r)$, we have $\mathbb{E}[K_u(\mathbf{x})] = \mathbb{E}[p(\mathbf{x}) \cdot \epsilon - u] = \ell - u \le 0$. Consequently, the conditions of Theorem 4.3 is satisfied, and we conclude that $\mathbb{P}[F(\mathbf{x}, \mathbf{e}) \ge U_c^{(1)}] \le \text{flag}_{U_c^{(1)}} \le 1 - c$.

- When the condition $\text{flag}_r < 1 - c$ is not satisfied, the algorithm directly sets $U_c^{(1)} := r$, which, by construction, is identical to the first-order threshold computed by Algorithm 2. By Theorem 4.2, it follows that $\mathbb{P}[F(\mathbf{x}, \mathbf{e}) \geq U_c^{(1)}] \leq 1 - c$.

In both cases, the value assigned to $U_c^{(1)}$ ensures that $\mathbb{P}[F(\mathbf{x}, \mathbf{e}) \geq U_c^{(1)}] \leq 1 - c$, making it a valid threshold on the first-order error term. Since the second-order error term is handled identically to Algorithm 2, it follows that the final threshold $U_c^*$, including both first- and second-order error, satisfies $\mathbb{P}[\text{err}(f, \mathbf{x}) \geq U_c^*] \leq 1 - c$. □

### 4.3 Algorithm for Fractional Expressions

In Section 4.2, two algorithms are introduced to address division-free expressions. However, when dealing with expressions with division by non-constant expressions, the partial derivatives $\frac{\partial \tilde{f}}{\partial e_i}(\mathbf{x}, \mathbf{0}, \mathbf{0})$ generally contain divisions, rendering them incompatible with the PN decomposition algorithm, which assumes division-free inputs. In this subsection, we explain how the CMB algorithm (Section 4.2.2) can be extended to handle fractional expressions in the form of $N(\mathbf{x})/Q(\mathbf{x})$ where both the numerator $N(\mathbf{x})$ and the denominator $Q(\mathbf{x})$ are division-free.

Similar to Section 4.2.2, we first state and prove the applicability of PN decomposition to our modified inputs in Lemma 4.5, then prove a theorem for bounding the first-order violation probability of a fixed threshold $u$, denoted by $\text{flag}_u$, in Theorem 4.6. Finally we introduce the complete algorithm in Algorithm 4 and prove its soundness in Theorem 4.7.

Lemma 4.5. *Suppose the input function $f(\mathbf{x}) = N(\mathbf{x})/Q(\mathbf{x})$ is a fractional expression, where both the numerator $N(\mathbf{x})$ and the denominator $Q(\mathbf{x})$ are division-free, and $Q(\mathbf{x}) \neq 0$ for all possible values of $\mathbf{x}$ in the specified distribution. Let $h_i(\mathbf{x}) = \frac{\partial \tilde{f}}{\partial e_i}(\mathbf{x}, \mathbf{0}, \mathbf{0}) \cdot (Q(\mathbf{x}))^2$, then $h_i(\mathbf{x})$ can be simplified to a polynomial expression.*

Proof Sketch (full proof in Appendix B). Given the structure of the input function $f(\mathbf{x})$, the floating-point model should have the form $\tilde{f}(\mathbf{x}, \mathbf{e}, \mathbf{d}) = \frac{\tilde{N}(\mathbf{x},\mathbf{e},\mathbf{d})}{\tilde{Q}(\mathbf{x},\mathbf{e},\mathbf{d})}(1 + e_k) + d_k$, where $k$ is the total number of arithmetic operations in $f(\mathbf{x})$ and $e_k, d_k$ are the error variables for the outmost division. Each $e_i$ should appear exactly once. We consider the three possibilities where $e_i$ appears, and describe how the corresponding $h_i(\mathbf{x})$ should be reducible to polynomials: (a) $e_i$ appears in $\tilde{N}(\mathbf{x}, \mathbf{e}, \mathbf{d})$: differentiation yields a factor $1/Q(\mathbf{x})$; multiplying by $(Q(\mathbf{x}))^2$ produces a division-free term; (b) $e_i$ appears in $\tilde{Q}(\mathbf{x}, \mathbf{e}, \mathbf{d})$: differentiation gives $1/(Q(\mathbf{x}))^2$, which is cancelled by multiplying $(Q(\mathbf{x}))^2$; (c) $e_i = e_k$: differentiation gives $N(\mathbf{x})/Q(\mathbf{x})$, and again multiplying $(Q(\mathbf{x}))^2$ yields $N(\mathbf{x})Q(\mathbf{x})$. In all cases, the result involves only sums and products of division-free expressions, so each $h_i(\mathbf{x})$ reduces to a polynomial. □

Theorem 4.6. *Let $u$ be a fixed threshold. Suppose the input function $f(\mathbf{x}) = N(\mathbf{x})/Q(\mathbf{x})$ where both the numerator $N(\mathbf{x})$ and the denominator $Q(\mathbf{x})$ are division-free, and $Q(\mathbf{x}) \neq 0$ for all possible values of $\mathbf{x}$ in the specified distributions. Let $p(\mathbf{x})$ be a polynomial*[4] *satisfying $\sum_{i=1}^{k} \left|\frac{\partial \tilde{f}}{\partial e_i}(\mathbf{x}, \mathbf{0}, \mathbf{0}) \cdot (Q(\mathbf{x}))^2 \cdot e_i\right| \leq p(\mathbf{x}) \cdot \epsilon$ for all possible values of $\mathbf{x}$ and $\mathbf{e}$. Let $K_u(\mathbf{x}) \triangleq p(\mathbf{x}) \cdot \epsilon - (Q(\mathbf{x}))^2 \cdot u$, and $\mu \triangleq \mathbb{E}[K_u(\mathbf{x})]$. Suppose that $\mu < 0$. For an even analysis order $n$, if we define $\text{flag}_u \triangleq \mathbb{E}[(K_u(\mathbf{x}) - \mu)^n]/\mu^n$, then the first-order violation probability admits the following upper bound: $\mathbb{P}[F(\mathbf{x}, \mathbf{e}) \geq u] \leq \text{flag}_u$.*

Proof. We first exploit the property of $p(\mathbf{x})$ to relax $F(\mathbf{x}, \mathbf{e})$ and eliminate explicit occurrences of error variables $\mathbf{e}$: $F(\mathbf{x}, \mathbf{e}) = \sum_{i=1}^{k} \left|\frac{\partial \tilde{f}}{\partial e_i}(\mathbf{x}, \mathbf{0}, \mathbf{0}) \cdot e_i\right| = \frac{1}{(Q(\mathbf{x}))^2} \sum_{i=1}^{k} \left|\frac{\partial \tilde{f}}{\partial e_i}(\mathbf{x}, \mathbf{0}, \mathbf{0}) \cdot (Q(\mathbf{x}))^2 \cdot e_i\right| \leq p(\mathbf{x}) \cdot \epsilon/(Q(\mathbf{x}))^2$. Accordingly, the violation probability can be relaxed by $\mathbb{P}[F(\mathbf{x}, \mathbf{e}) \geq u] \leq$

[4] It will be described after the proof of Theorem 4.6 how to derive such a polynomial function $p(\mathbf{x})$.

$\mathbb{P}\left[p(\mathbf{x}) \cdot \epsilon/(Q(\mathbf{x}))^2 \geq u\right] = \mathbb{P}[p(\mathbf{x}) \cdot \epsilon - (Q(\mathbf{x}))^2 \cdot u \geq 0] = \mathbb{P}[K_u(\mathbf{x}) \geq 0]$. The rest of the proof is analogous to that in Theorem 4.3. The event $K_u(\mathbf{x}) \geq 0$ would imply the event $|K_u(\mathbf{x}) - \mu| \geq |\mu|$, and therefore, $\mathbb{P}[K_u(\mathbf{x}) \geq 0] \leq \mathbb{P}[|K_u(\mathbf{x}) - \mu| \geq \mu| = \mathbb{P}[(K_u(\mathbf{x}) - \mu)^n \geq \mu^n] \leq \mathbb{E}[(K_u(\mathbf{x}) - \mu)^n]/\mu^n = \text{flag}_u$. Combining the inequalities, we conclude $\mathbb{P}[F(\mathbf{x}, \mathbf{e}) \geq u] \leq \text{flag}_u$. □

We now describe how PN decomposition can be employed to produce the $p(\mathbf{x})$ as required in Theorem 4.6. As is proved in Lemma 4.5, $h_i(\mathbf{x}) = \frac{\partial \tilde{f}}{\partial e_i}(\mathbf{x}, \mathbf{0}, \mathbf{0}) \cdot (Q(\mathbf{x}))^2$ can always be simplified to a polynomial expression. We may thus provide $h_1(\mathbf{x}), \ldots, h_k(\mathbf{x})$ as inputs to the PN decomposition algorithm (Algorithm 1), and by Theorem 4.1, the algorithm returns a function $p(\mathbf{x})$ such that $\sum_{i=1}^{k} \left|\frac{\partial \tilde{f}}{\partial e_i}(\mathbf{x}, \mathbf{0}, \mathbf{0}) \cdot (Q(\mathbf{x}))^2 \cdot e_i\right| \leq p(\mathbf{x}) \cdot \epsilon$, which precisely satisfies the requirement of Theorem 4.6.

**Algorithm 4** The Extended Central-Moment-Based Algorithm for Fractional Expressions

**Input:** A fractional expression $f(\mathbf{x}) = N(\mathbf{x})/Q(\mathbf{x})$ (where $N(\mathbf{x})$ and $(Q(\mathbf{x})$ are division-free), distribution of the input variables, confidence level $c$, unit round-off $\epsilon$, smallest representable number $\delta$, and even analysis order $n$

**Output:** A probabilistic upper bound $U_c^*$ for the round-off error $\text{err}(f, \mathbf{x})$

1: Compute the floating-point version of the expression $\tilde{f}(\mathbf{x}, \mathbf{e}, \mathbf{d})$.
2: Compute the expressions for $h_i(\mathbf{x}) = \frac{\partial \tilde{f}}{\partial e_i}(\mathbf{x}, \mathbf{0}, \mathbf{0}) \cdot (Q(\mathbf{x}))^2$.
3: Input the list of $h_i(\mathbf{x})$ into Algorithm 1 and get the expression for $p(\mathbf{x})$
4: $\ell \leftarrow \mathbb{E}[p(\mathbf{x})] \cdot \epsilon/\mathbb{E}[(Q(\mathbf{x}))^2]$
5: $r \leftarrow \ell \times 10^5$ ▷ $10^5$ is the large multiple mentioned in the description of the algorithm.
6: Use binary search to find the close-to-minimum $U_c^{(1)} \in (\ell, r)$ such that $\text{flag}_{U_c^{(1)}} \leq 1 - c$
7: Get a deterministic upper bound $U^{(2)}$ for $|R_2(\mathbf{x}, \mathbf{e}, \mathbf{d})|$ using a sound global optimization tool
8: **Return** $U_c^* \leftarrow U_c^{(1)} + U^{(2)}$

We adopt a strategy analogous to the CMB algorithm (Section 4.2.2, Algorithm 3), employing binary search [5] to find the close-to-minimum $U_c^{(1)}$ such that $\text{flag}_{U_c^{(1)}} \leq 1 - c$ w.r.t Theorem 4.6. To ensure that $\mu = \mathbb{E}[p(\mathbf{x}) \cdot \epsilon - (Q(\mathbf{x}))^2 \cdot u]$ remains negative throughout the search, we initialize the left endpoint of the binary search to $\mathbb{E}[p(\mathbf{x})] \cdot \epsilon/\mathbb{E}[(Q(\mathbf{x}))^2]$, and the right endpoint to a large multiple of the left endpoint. The detailed algorithm is presented in Algorithm 4.

Theorem 4.7. *Let $U_c^*$ be the output of Algorithm 4, then $\mathbb{P}[\text{err}(f, \mathbf{x}) \geq U_c^*] \leq 1 - c$.*

Proof. We follow a similar line of reasoning as in the proof of Theorem 4.4. As proved in Lemma 4.5, each expression $h_i(\mathbf{x})$ computed at line 2 of the algorithm can be simplified to a polynomial expression, and hence qualifies as input to Algorithm 1. Applying Algorithm 1 to the list $h_i(\mathbf{x})$ and invoking Theorem 4.1, we obtain an output polynomial $p(\mathbf{x})$ such that $\sum_{i=1}^{k} \left|\frac{\partial \tilde{f}}{\partial e_i}(\mathbf{x}, \mathbf{0}, \mathbf{0}) \cdot (Q(\mathbf{x}))^2 \cdot e_i\right| \leq p(\mathbf{x}) \cdot \epsilon$. Besides, all possible values of $U_c^{(1)}$ in the search space are greater than the left endpoint of our binary search $\ell = \mathbb{E}[p(\mathbf{x})] \cdot \epsilon/\mathbb{E}[(Q(\mathbf{x}))^2]$, and hence $\mu = \mathbb{E}[p(\mathbf{x}) \cdot \epsilon - (Q(\mathbf{x}))^2 \cdot U_c^{(1)}] < 0$. Thus the preconditions of Theorem 4.6 are met, and by Theorem 4.6, $\text{flag}_u$ serves as a sound over-approximation of the first-order violation probability.

[5] Since the monotonicity of $\text{flag}_u$ in the case of fractional expressions is not formally proved, binary search is a tentative plan here. It currently works across all evaluated benchmarks. As a fallback, if binary search fails to find a solution, we can instead enumerate candidate values of $u$ at each order of magnitude to identify a suitable threshold.

Therefore, when the binary search at line 6 is able to output a value of $U_c^{(1)}$, it satisfies $\mathbb{P}[F(\mathbf{x},\mathbf{e}) \geq U_c^{(1)}] \leq \text{flag}_{U_c^{(1)}} \leq 1-c$. Since the second order error term is handled identically to Algorithm 2, it follows that the final threshold $U_c^*$, including both first- and second-order error, satisfies $\mathbb{P}[\text{err}(f,\mathbf{x}) \geq U_c^*] \leq 1-c$. □

Remark 2. *Our approach can be easily extended for general expressions with division, by reducing* $\tilde{f}(\mathbf{x},\mathbf{e},\mathbf{d})$ *to a common denominator* $\tilde{Q}(\mathbf{x},\mathbf{e},\mathbf{d})$*, and then taking partial derivatives* $\frac{\partial \tilde{f}}{\partial e_i}(\mathbf{x},\mathbf{0},\mathbf{0})$*. Though some* $e_i$ *may occur in both the numerator and the denominator, in all partial derivatives* $\frac{\partial \tilde{f}}{\partial e_i}(\mathbf{x},\mathbf{0},\mathbf{0})$ *the denominator divides* $(\tilde{Q}(\mathbf{x},\mathbf{0},\mathbf{0}))^2$*, and thus* $h_i(\mathbf{x}) = \frac{\partial \tilde{f}}{\partial e_i}(\mathbf{x},\mathbf{0},\mathbf{0}) \cdot (\tilde{Q}(\mathbf{x},\mathbf{0},\mathbf{0}))^2$ *remains a polynomial, thereby ensuring the proposed algorithm is applicable in this setting.*

### 4.4 Refinement with Range Partition

A central component of the CMB algorithm, both in its original formulation and its extension to fractional expressions, is the computation of an upper bound on first-order violation probability, $\text{flag}_u$, for a given threshold $u$. In order to tighten this bound, we introduce a *range partition refinement* for the computation of $\text{flag}_u$. The key idea is to partition the domain of each variable in $\mathbf{x}$ into smaller sub-regions, compute a local upper bound on $\text{flag}_u$ within every resulting sub-region of the input space, and then aggregate these local bounds to obtain the global estimate of $\text{flag}_u$.

Several new notations and definitions related to range partition are introduced for readability. Assume there are $m$ input variables, $x_1,\ldots,x_m$, and the range of variable $x_i$ is $I_i \subset \mathbb{R}$. For each $i \in \{1,\ldots,m\}$, we partition the range $I_i$ into $b$ disjoint sub-ranges: $I_{i,1},\ldots,I_{i,b}$, where $I_i = \bigcup_{j=1}^{b} I_{i,j}$ and $I_{i,j} \cap I_{i,k} = \emptyset$ for $j \neq k$. The weight of each sub-range $I_{i,j}$ is defined as $w_{i,j} \triangleq \mathbb{P}[x_i \in I_{i,j}]$. Given a multi-index $(i_1,\ldots,i_m) \in \{1,\ldots,b\}^m$, a sub-region is defined as $B_{i_1,\ldots,i_m} \triangleq I_{1,i_1} \times \cdots \times I_{m,i_m}$. Assuming the independence of variables, the weight of the sub-region is given by the product $W_{i_1,\ldots,i_m} = \mathbb{P}[\mathbf{x} \in B_{i_1,\ldots,i_m}] = \prod_{j=1}^{m} w_{j,i_j}$.

*Illustrative Example.* We illustrate range partition using Example 2.2. Suppose the number of sub-ranges is $b = 2$, and the ranges are partitioned uniformly. Then $I_{i,1} = [-1,0], I_{i,2} = [0,1]$ for $i \in \{1,2,3\}$, each associated with an equal weight of 0.5. The eight sub-regions are different combinations of the sub-ranges. For example, sub-region $B_{1,1,1}$ is $[-1,0]^3$, and its weight is $W_{1,1,1} = 0.5^3 = 0.125$. For a given value of $u$, we compute the local probability bounds $\text{flag}_{u,B_\mathbf{j}}$ for all $\mathbf{j} \in \{1,2\}^3$ and sub-region $B_\mathbf{j}$, where the expectations are taken w.r.t the normalized distribution on the sub-region, and then get the global bound by aggregating across all sub-regions $\text{flag}_u = \sum_{\mathbf{i}\in\{1,2\}^3} \text{flag}_{u,B_\mathbf{i}}$.

We now present the general refined computation of $\text{flag}_u$ in Algorithm 5. The algorithm begins by partitioning the ranges into sub-ranges, and computing the corresponding weights (lines 1–4). The main loop (lines 6–16) iterates over each sub-region $B$: it first computes the weight of the sub-region, $W$ (line 7), then derives the local upper bound $\text{flag}_{u,B}$ (lines 8–14), and finally accumulates the weighted local upper bound onto the global bound (line 15). We provide a proof of correctness in Theorem 4.8, and then explain why the range partition refinement could result in a tighter bound.

Theorem 4.8. *Let* $\text{flag}_u$ *be the output of Algorithm 5, then* $\mathbb{P}[K_u(\mathbf{x}) \geq 0] \leq \text{flag}_u$.

Proof. We first establish the correctness of each iteration of the for-loop (line 6 – 16) by showing that the $\text{flag}_{u,B_\mathbf{i}}$ generated in line 14 is a sound local upper bound $\mathbb{P}[K_u \geq 0 \mid \mathbf{x} \in B_\mathbf{i}] \leq \text{flag}_{u,B_\mathbf{i}}$. If $\mu > 0$ or the value computed in line 10 exceeds 1, the algorithm sets $\text{flag}_{u,B_\mathbf{i}}$ to 1, and the desired inequality holds trivially. Otherwise, the validity of the bound follows by an argument analogous

**Algorithm 5** Computing $\text{flag}_u$ with range partition

**Input:** Input variables $x_1, \ldots, x_m$ and their ranges $I_1, \ldots, I_m$, threshold $u$, number of partitions $b$, expression of $K_u(\mathbf{x})$ (either in polynomial or fractional case), analysis order $n$
**Output:** An upper bound for first-order violation probability $\text{flag}_u$

**for** $i \leftarrow 1$ to $m$ **do**
  Partition range $I_i$ into sub-ranges $I_{i,1}, \ldots, I_{i,b}$
  Compute weights: $w_{i,j} \leftarrow \mathbb{P}[x_i \in I_{i,j}]$ for $j = 1$ to $b$
**end for**
$\text{flag}_u \leftarrow 0$
**for** each multi-index $\mathbf{i} \in \{1, \ldots, b\}^m$ **do**
  Compute sub-region weight $W_\mathbf{i} \leftarrow \prod_{j=1}^m w_{j,i_j}$
  $\mu \leftarrow \mathbb{E}[K_u(\mathbf{x}) \mid \mathbf{x} \in B_\mathbf{i}]$
  **if** $\mu < 0$ **then**
    $\text{flag}_{u,B_\mathbf{i}} \leftarrow \mathbb{E}[(K_u(\mathbf{x}) - \mu)^n \mid \mathbf{x} \in B_\mathbf{i}]/\mu^n$
  **else**
    $\text{flag}_{u,B_\mathbf{i}} \leftarrow 1$
  **end if**
  Set $\text{flag}_{u,B_\mathbf{i}}$ to 1 if $\text{flag}_{u,B_\mathbf{i}} > 1$
  $\text{flag}_u \leftarrow \text{flag}_u + W_\mathbf{i} \cdot \text{flag}_{u,B_\mathbf{i}}$
**end for**
**Return** $\text{flag}_u$

to that in Theorem 4.3. Specifically, under the normalized distribution of $\mathbf{x}$ conditioned on $\mathbf{x} \in B_\mathbf{i}$,

$$\mathbb{P}[K_u(\mathbf{x}) \geq 0 \mid \mathbf{x} \in B_\mathbf{i}] \leq \mathbb{P}[|K_u(\mathbf{x}) - \mu| \geq |\mu| \mid \mathbf{x} \in B_\mathbf{i}] = \mathbb{P}[(K_u(\mathbf{x}) - \mu)^n \geq \mu^n \mid \mathbf{x} \in B_\mathbf{i}]$$
$$\leq \mathbb{E}[(K_u(\mathbf{x}) - \mu)^n \mid \mathbf{x} \in B_\mathbf{i}]/\mu^n = \text{flag}_{u,B_\mathbf{i}}.$$

This confirms that the local bound $\text{flag}_{u,B_\mathbf{i}}$ indeed bounds the probability.

Note that the returned $\text{flag}_u$ is equal to the weighted sum of all local bounds, i.e., $\text{flag}_u = \sum_{\mathbf{i} \in (1,\ldots,b)^m} \text{flag}_{u,B_\mathbf{i}} W_\mathbf{i}$. Then we extend the local guarantee to a global one. Since $\{B_\mathbf{i}\}_{\mathbf{i} \in (1,\ldots,b)^m}$ is a set of disjoint sub-regions covering the domain of $\mathbf{x}$, by the law of total probability,

$$\mathbb{P}[K_u(\mathbf{x}) \geq 0] = \textstyle\sum_{\mathbf{i} \in (1,\ldots,b)^m} \mathbb{P}[K_u(\mathbf{x}) \geq 0 \mid \mathbf{x} \in B_\mathbf{i}] \cdot \mathbb{P}[\mathbf{x} \in B_\mathbf{i}] \leq \sum_{\mathbf{i} \in (1,\ldots,b)^m} \text{flag}_{u,B_\mathbf{i}} \cdot W_\mathbf{i} = \text{flag}_u.$$

□

By the proofs in Theorem 4.3 and Theorem 4.6, we know that $\mathbb{P}[F(\mathbf{x}, \mathbf{e}) \geq u] \leq \mathbb{P}[K_u(\mathbf{x}) \geq 0]$ for the CMB algorithm, both in its original form and its extension for fractional expressions. Although the specific form of the auxiliary expression $K_u(\mathbf{x})$ differs between the two cases, the range partition refinement applies to both settings. Consequently, in either case, the first-order violation probability satisfies $\mathbb{P}[F(\mathbf{x}, \mathbf{e}) \geq u] \leq \mathbb{P}[K_u(\mathbf{x}) > 0] \leq \text{flag}_u$ for the $\text{flag}_u$ given by Algorithm 5.

Remark 3. *Range partition enhances the CMB method because the mean value of $K_u(\mathbf{x})$ differs across sub-regions, enabling the computation of CMB bounds with sub-region-specific means rather than one global mean. Since CMB bounds are fractional, these differences influence the final aggreagated result. By contract, partition offers no benefit for the NM method, since local bounds are combined by direct summation, which ultimately depends on the global mean over the entire range.* □

Remark 4. *Our current approach does not support transcendental functions. While our implementation can handle polynomial approximations of transcendental functions (such as benchmarks* `ksin` *and* `kcos`*), extending our approach to handle transcendental functions directly would require substantial*

*effort. In particular, this would require precise knowledge of the polynomial or piecewise-polynomial approximations used in specific languages, libraries, or hardware, as well as accounting for the additional error introduced by replacing the transcendental functions with its (piecewise) polynomial approximation. Handling this approximation error falls outside the scope of this paper.* □

## 5 Implementation and Evaluation

**Implementation.** We have implemented our algorithms (Algorithm 2, Algorithm 3 along with its extension for fractional expressions, Algorithm 4, and their refinement using Algorithm 5) in a prototype tool named ProbTaylor. The implementation comprises approximately 2000 lines of code in OCaml, relying only on standard libraries – `List`, `ocamllex`, and `ocamlyacc`. We obtain second-order errors using the off-the-shelf sound optimizer, GELPIA [1], to carry out global optimization. ProbTaylor offers three operational modes: `nm`, `cmb`, and `div`, corresponding to the Naive Markov algorithm, the Central-Moment-Based algorithm, and its extension for fractional expressions, respectively. ProbTaylor accepts as input a text file specifying the distribution of the input variables and the floating-point expression to be analyzed. Additionally, the user may also specify the desired precision parameters $\epsilon$ and $\delta$, the target confidence level $c$, the analysis order $n$, and the number of partitions $b$ (the latter applicable only to `cmb` and `div` modes). Three types of input distributions are currently supported: uniform, normal, and double exponential (a.k.a. Laplace distribution) with various variance values. The standard normal and double exponential distributions are truncated to a certain range specified by the input file. Four basic arithmetic operations are supported: addition, subtraction, multiplication, and division. The output of ProbTaylor is a valid threshold that is satisfied with probability of at least the given confidence level $c$.

It is worth mentioning that the expectation of given expressions is computed through exact symbolic derivation instead of numerical approximation. This not only enhances the computational efficiency, but also produces a more accurate result. See Appendix A for details.

**Benchmarks.** The benchmark set we use to evaluate our prototype includes three parts – polynomial benchmarks, fractional benchmarks, and scalability benchmarks. The polynomial benchmarks include all 24 polynomial benchmarks from PAF [11], which are originally adapted from FPBench [13], and 2 benchmarks adapted from `fdlibm` [6] (`ksin`, `kcos`). The number of operations in these polynomial benchmarks ranges from 1 to 31. The fractional benchmarks include all 3 fractional benchmarks from PAF, and 4 additional ones from FPBench (`nonlin1`, `nonlin2`, `predator`, `verhulst`). The number of operations in these fractional benchmarks ranges from 2 to 12. The scalability benchmarks are explained in Section 5.2. Additional benchmarks are derived by widening the input distributions of existing benchmarks. We assume single precision computation and 99% target confidence level for all experiments in this section.

**Evaluation Criteria.** In the evaluation, we consider the following four research questions:

- **RQ1:** How does the NM algorithm compare to the SOTA tools, PAF and PrAn, in terms of accuracy and time efficiency on polynomial benchmarks?
- **RQ2:** How well does our approach scale when analyzing expressions with a large number of variables and operations?
- **RQ3:** How does the CMB algorithm, equipped with range partition refinement, perform relative to the NM algorithm and the existing tools on polynomial benchmarks? How does its extension for fractional expressions perform compared to existing tools?
- **RQ4:** To what extent can ProbTaylor improve upon the results obtained from conservative deterministic analysis tools (such as FPTaylor)?

[6] `fdlibm`: freely distributable math library for C programming language originally developed by Sun Microsystems, available at http://www.netlib.org/fdlibm/.

**Experiment Setup.** The experiments were run on a 2017 iMac Pro with a 2.3 GHz 18-core Intel Xeon W processor and 128 GB of memory, running MacOS 14.4.1. Since PAF is only compatible with Ubuntu 18.04, all experiments, including those with PAF and PrAn, and with ProbTaylor, were run on a virtual machine configured with Ubuntu 18.04 hosted on the aforementioned iMac Pro.

**Underflow and Exceptions.** Due to numerical limits of OCaml and Python, underflow arises during the analysis in both ProbTaylor (e.g. on `classids` benchmarks with Laplace distribution with $\sigma = 0.01$), and in existing tools – PAF (e.g. on `nonlin2`) and PrAn (e.g. on `doppler`). When such underflow occurs in the denominator, it could potentially lead to division by zero, and produce NaN values. In these cases, the corresponding entries in the results are marked as "DZ". Following the same assumption of PAF, we assume no overflow or other exceptions during computations. This assumption is further validated since every benchmark passes the exception checker of FPTaylor.

### 5.1 Comparing the Naive Markov algorithm to PAF and PrAn (Answering RQ1)

The NM algorithm (Algorithm 2) allows customization of the analysis order $n$, thereby offering flexibility. It is intended as a light-weight static analysis algorithm capable of producing results with relatively low computational overhead. Therefore, in the evaluation, we execute the NM algorithm on orders 2, 4, 6, 8, 10, 12, 18, 24, 30, 36, with a timeout of 90 seconds for each individual analysis order, and terminate the execution upon a timeout. [7] We report the tightest probabilistic threshold generated among these configurations, and report the running time as the total time elapsed across analysis orders, plus the time spent computing and bounding second-order error via GELPIA [8].

Since the NM algorithm only applies to division-free expressions, the experiments in this subsection exclude benchmarks involving division by non-constants. We compare our thresholds and running time with those obtained using PAF and PrAn (settings C and D in [25]). We consider the case where all operations are performed in single precision and the target confidence level is 99%.

We document the experimental results on the benchmarks with the original distributions from PAF in Table 1 (uniform distribution), Table 2 (standard normal distribution), and Table 3 (Laplace distribution). In the case of Laplace distribution, we adopt a variance value of $\sigma = 0.01$ to maintain consistency with PAF, except for four benchmarks (`classids0`, `classids1`, `classids2`, and `solvecubic`) where numerical underflow occurs. For these four benchmarks, we instead set $\sigma = 1$. We impose a timeout of 3600 seconds per benchmark. For the PAF benchmarks where PAF exceeds this time limit, we report the output thresholds from PAF as documented in [11] and run PAF exhaustively to get thresholds for the $\sigma = 1$ double exponential benchmarks as mentioned above.

Due to page limits, the output thresholds and running times of PAF and PrAn are not presented in the main tables. These data can be found in Appendix E. The tables focus on comparing our method against PAF and PrAn. The columns labeled "Speedup" indicate how much faster (or, in a few cases, slower) the NM algorithm is compared to PAF/PrAn, expressed as the ratio of their respective running times. For those benchmarks where PAF fails to complete within the 3600-second timeout, we report a lower bound on the speedup. The columns labeled "Threshold ratio" provide a quantitative comparison of the thresholds, expressing the thresholds derived using the NM algorithm as a percentage of those obtained with PAF/PrAn. For uniform (Table 1) and truncated normal distribution (Table 2), we compare the NM method to both PAF and PrAn. For truncated Laplace distribution (Table 3), which is not supported by PrAn, we compare only to PAF.

[7] We are not aware of an existing method to determine the optimal analysis order in advance. Nonetheless, evaluating multiple values of $n$ remains computationally inexpensive.

[8] For benchmarks `ksin` and `kcos`, GELPIA crashes when analyzing the second-order error. Therefore, we developed a simple brute-force algorithm to bound the second-order error in those cases. The brute-force algorithm is explained in Appendix D

Table 1. Experimental results for uniform distribution. The thresholds are produced assuming computations done in single precision and 99% confidence level. The optimal $n$ is the analysis order with which the NM algorithm produces the tightest threshold. The threshold ratios are the ratios of thresholds given by the NM algorithm to those given by PAF/PrAn. A threshold ratio smaller than 100% (marked in bold) indicates that the NM algorithm produces a tighter threshold, and vice versa. "TO" indicates timeout for competing tool.

| | Uniform distribution | | | | | | |
|---|---|---|---|---|---|---|---|
| | Naive Markov algorithm | | | Comparison with PAF | | Comparison with PrAn | |
| Benchmark | Time (s) | Threshold | Optimal $n$ | Speedup | Threshold ratio | Speedup | Threshold ratio |
| bsplines0 | 0.84 | 4.85E-07 | 36 | 1,466x | 849.39% | 21x | 558.11% |
| bsplines1 | 0.74 | 5.61E-07 | 36 | 2,210x | 301.61% | 37x | 296.83% |
| bsplines2 | 1.47 | 5.60E-07 | 36 | 1,869x | 288.66% | 24x | 264.15% |
| bsplines3 | 0.26 | 3.96E-08 | 36 | 3,565x | **93.84%** | 58x | **69.35%** |
| classids0 | 98.12 | 1.17E-05 | 36 | >36x | 168.83% | 1.08x | 135.26% |
| classids1 | 99.76 | 6.56E-06 | 36 | >36x | 176.82% | 1.46x | 139.87% |
| classids2 | 98.29 | 1.02E-05 | 36 | >36x | 195.03% | 1.22x | 134.56% |
| filters1 | 0.19 | 8.58E-08 | 36 | 3,200x | **68.64%** | 18x | **42.27%** |
| filters2 | 0.75 | 8.07E-07 | 36 | 4,248x | 101.77% | 86x | **79.90%** |
| filters3 | 19.49 | 2.57E-06 | 36 | >184x | 109.83% | 10x | **89.86%** |
| filters4 | 168.95 | 5.62E-06 | 36 | >21x | 135.42% | 2.54x | 108.08% |
| rigidbody1 | 97.02 | 1.75E-04 | 18 | >37x | 100.57% | 0.28x | 101.16% |
| rigidbody2 | 137.21 | 1.55E-02 | 8 | >26x | **79.08%** | 0.33x | 159.79% |
| sine | 3.9 | 5.52E-07 | 36 | >923x | 232.91% | 34x | 229.05% |
| solvecubic | 31.03 | 1.91E-05 | 36 | >116x | 107.30% | 5.82x | **95.02%** |
| sqrt | 1.18 | 1.39E-04 | 36 | >3,050x | **90.26%** | 47x | **90.26%** |
| traincars1 | 12.82 | 2.47E-03 | 36 | 267x | 140.34% | 3.92x | 126.02% |
| traincars2 | 185.36 | 1.84E-03 | 18 | >19x | 176.92% | 1.10x | 138.35% |
| traincars3 | 138.94 | 2.80E-02 | 8 | >25x | 160.00% | 0.13x | 122.27% |
| traincars4 | 115.24 | 3.53E-01 | 6 | >31x | 195.03% | 0.22x | 153.48% |
| trid1 | 158.33 | 6.78E-03 | 18 | >22x | 112.81% | 0.43x | 110.78% |
| trid2 | 195.14 | 1.35E-02 | 10 | >18x | 131.07% | 0.30x | 115.38% |
| trid3 | 116.82 | 2.46E-02 | 6 | >30x | 140.57% | 2.67x | 126.15% |
| trid4 | 100.17 | 4.64E-02 | 4 | >35x | 172.49% | 10x | 161.11% |
| ksin | 8.09 | 7.78E-08 | 36 | >444x | TO | 253x | 112.27% |
| kcos | 6.72 | 2.05E-07 | 36 | >535x | TO | 273x | 170.83% |

The NM algorithm successfully completes all benchmarks within 200 seconds, whereas PAF exceeds the time limit on 56 of 78 benchmarks. This shows that our approach is significantly more time-efficient compared to PAF. Furthermore, PrAn fails to finish analysis on 2 of 52 benchmarks. Our approach outperforms PrAn in terms of runtime, achieving an average speedup of 49x and completing the analysis faster on 41 of 52 benchmarks. Importantly, the advantage in time efficiency does not sacrifice the accuracy of the derived threshold, as the NM algorithm produces probabilistic thresholds that are within the same order of magnitude as those produced by PrAn on all benchmarks, and by PAF on all but two. Moreover, the NM algorithm yields tighter thresholds than PAF on 30 of 72 benchmarks, and achieves thresholds more than an order of magnitude tighter on 6 of the benchmarks. In comparison to PrAn, the NM algorithm produces tighter thresholds on 25 of

Table 2. Experimental results for truncated standard normal distribution. The setup is the same as in Table 1.

| | Truncated standard normal distribution | | | | | | |
|---|---|---|---|---|---|---|---|
| | Naive Markov algorithm | | | Comparison with PAF | | Comparison with PrAn | |
| Benchmark | Time (s) | Threshold | Optimal $n$ | Speedup | Threshold ratio | Speedup | Threshold ratio |
| bsplines0 | 0.86 | 7.56E-07 | 8 | 1,426x | 1323.99% | 33x | 869.97% |
| bsplines1 | 0.74 | 7.56E-07 | 6 | 2,306x | 406.45% | 72x | 360.00% |
| bsplines2 | 1.47 | 7.59E-07 | 6 | 1,889x | 391.24% | 42x | 358.02% |
| bsplines3 | 0.26 | 6.18E-08 | 4 | 3,515x | 146.45% | 86x | 108.23% |
| classids0 | 98.77 | 4.54E-06 | 18 | >36x | 102.02% | 2.83x | **51.01%** |
| classids1 | 98.34 | 2.67E-06 | 18 | >36x | **99.63%** | 3.02x | **56.09%** |
| classids2 | 98.89 | 4.47E-06 | 18 | >36x | 116.10% | 2.74x | **58.82%** |
| filters1 | 0.18 | 8.37E-08 | 24 | 3,227x | **67.50%** | 30x | **41.23%** |
| filters2 | 0.76 | 7.48E-07 | 24 | 4,236x | 122.02% | 159x | **74.06%** |
| filters3 | 19.76 | 2.30E-06 | 18 | >182x | 112.20% | 14x | **80.14%** |
| filters4 | 168.12 | 4.89E-06 | 18 | >21x | 117.83% | 4.91x | **94.04%** |
| rigidbody1 | 87.27 | 4.19E-06 | 6 | >41x | **68.24%** | 0.52x | **2.42%** |
| rigidbody2 | 133.20 | 2.35E-05 | 4 | >27x | **39.23%** | 0.71x | **0.24%** |
| sine | 3.77 | 6.85E-07 | 6 | >954x | 289.03% | 126x | 284.23% |
| solvecubic | 31.37 | 4.94E-06 | 18 | >114x | **72.22%** | 7.34x | **24.46%** |
| sqrt | 1.19 | 2.98E-06 | 4 | >3,025x | 1211.38% | 97x | **1.94%** |
| traincars1 | 12.75 | 9.53E-04 | 18 | 235x | 115.38% | 7.32x | **48.62%** |
| traincars2 | 180.03 | 4.07E-04 | 12 | >19x | 112.43% | 2.64x | **30.60%** |
| traincars3 | 138.61 | 5.92E-03 | 8 | >25x | **61.92%** | 0.13x | **26.55%** |
| traincars4 | 114.32 | 6.95E-02 | 6 | >31x | **78.35%** | 0.23x | **30.22%** |
| trid1 | 158.87 | 7.32E-06 | 8 | >22x | **46.33%** | 1.59x | **0.12%** |
| trid2 | 195.4 | 1.21E-05 | 8 | >18x | **50.00%** | 0.71x | **0.10%** |
| trid3 | 116.59 | 1.88E-05 | 6 | >30x | **27.65%** | 4.60x | **0.10%** |
| trid4 | 100.29 | 3.34E-05 | 4 | >35x | **12.65%** | 15.71x | **0.11%** |
| ksin | 8.3 | 1.49E-07 | 4 | >433x | TO | >433x | TO |
| kcos | 7.18 | 4.69E-07 | 4 | >501x | TO | >501x | TO |

50 benchmarks, with improvements exceeding an order of magnitude on 7 benchmarks. Overall, these results indicate that the NM algorithm generally derives comparable or better thresholds in substantially less time. Additionally, the accuracy of the results can be further improved through the CMB algorithm and range partition.

### 5.2 Scalability Experiments (Answering RQ2)

To evaluate the scalability of the NM algorithm, we designed a set of large-scale benchmarks, constructed as the inner product of two vectors whose entries are independently and uniformly distributed over $(0, 1)$. The analysis is performed with analysis order $n = 2$. Due to GELPIA failing to bound the second-order errors on these large benchmarks, the brute-force method is employed instead. A 4-hour timeout is enforced on the analysis of second-order errors. We report the running time for analyzing first- and second-order error separately. We report the results in Table 4.

Table 3. Experimental results for truncated double exponential distribution. The setup is the same as in Table 1. For benchmarks `classids0`, `classids1`, `classids2`, and `solvecubic`, the variance value is set to 1. For all other benchmarks, the variance value is set to 0.01.

| | Truncated double exponential distribution | | | | |
|---|---|---|---|---|---|
| | Naive Markov algorithm | | | Comparison with PAF | |
| Benchmark | Time (s) | Threshold | Optimal $n$ | Speedup | Threshold ratio |
| bsplines0 | 0.84 | 7.41E-08 | 30 | 3,129x | 129.77% |
| bsplines1 | 0.74 | 9.04E-08 | 36 | 2,974x | 130.07% |
| bsplines2 | 1.47 | 2.96E-08 | 18 | >2,448x | 140.28% |
| bsplines3 | 0.25 | 1.07E-11 | 2 | 3,752x | 140.42% |
| classids0 | 381.09 | 6.76E-06 | 18 | 64x | 135.20% |
| classids1 | 382.22 | 3.90E-06 | 18 | 42x | 126.62% |
| classids2 | 382.39 | 6.23E-06 | 18 | 35x | 150.12% |
| filters1 | 0.19 | 2.69E-09 | 6 | 3,857x | **49.54%** |
| filters2 | 0.74 | 2.06E-08 | 6 | >4,864x | **71.03%** |
| filters3 | 19.88 | 5.54E-08 | 8 | >181x | **50.83%** |
| filters4 | 168.43 | 1.12E-07 | 8 | >21x | **24.30%** |
| rigidbody1 | 97.11 | 9.01E-09 | 6 | >37x | **1.88%** |
| rigidbody2 | 132.93 | 1.20E-08 | 4 | >27x | **1.26%** |
| sine | 3.82 | 1.16E-08 | 6 | >942x | **77.85%** |
| solvecubic | 116.66 | 1.52E-05 | 12 | 143x | 107.04% |
| sqrt | 1.2 | 2.91E-07 | 24 | >3,000x | 118.29% |
| traincars1 | 13.83 | 3.10E-04 | 36 | >260x | **68.89%** |
| traincars2 | 181.78 | 8.64E-06 | 10 | >19x | **30.53%** |
| traincars3 | 223.37 | 1.17E-04 | 8 | >16x | **13.07%** |
| traincars4 | 114.92 | 1.58E-03 | 6 | >31x | **21.56%** |
| trid1 | 158.35 | 5.55E-07 | 18 | >22x | **3.51%** |
| trid2 | 195.13 | 1.07E-06 | 10 | >18x | **4.40%** |
| trid3 | 117.04 | 2.25E-06 | 6 | >30x | **3.32%** |
| trid4 | 99.84 | 4.86E-06 | 4 | >36x | **1.84%** |
| ksin | 8.36 | 3.87E-09 | 6 | >430x | TO |
| kcos | 6.76 | 1.89E-07 | 10 | >532x | TO |

The experimental results demonstrate that the NM algorithm maintains reasonable computational performance on large-scale inputs – the first-order analysis completes within 0.1 to 76 seconds as the vector length increases from 25 to 300. However, the computation of the second-order error expression and the brute-force method for bounding second-order error are considerably more expensive, consuming significantly more time or even being infeasibly slow. Overall, these findings suggest that the NM algorithm scales effectively for first-order error analysis, while efficiency improvements for second-order error estimation remains an important direction for future work.

### 5.3 Evaluating the Central-Moment-Based Algorithm (Answering RQ3)

In this subsection, we compare the NM method with the CMB method (Algorithm 3) with range partition (Algorithm 5) on polynomial benchmarks. We also evaluate the extended CMB method for fractional expressions (Algorithm 4) by comparing with those obtained by PAF and PrAn.

Table 4. Scalability experiments. The benchmarks are constructed as the inner product of two vectors. All entries are independently and uniformly distributed over the interval $(0, 1)$. Analysis assumes single precision computation and 99% target confidence level. Computational times for first-order and second-order error threshold are reported separately in columns labeled "FO time (s)" and "SO time (s)". "TO" stands for timeout.

| Vector length | Threshold | FO time (s) | SO time (s) | Vector length | Threshold | FO time (s) | SO time(s) |
|---|---|---|---|---|---|---|---|
| 25 | 5.30E-05 | 0.09 | 53.64 | 175 | 2.33E-03 | 14.5 | TO |
| 50 | 1.99E-04 | 0.45 | 1779.53 | 200 | 3.03E-03 | 21.58 | TO |
| 75 | 4.39E-04 | 1.23 | 13965.87 | 225 | 3.83E-03 | 31.32 | TO |
| 100 | 7.71E-04 | 2.88 | TO | 250 | 4.72E-03 | 42.38 | TO |
| 125 | 1.20E-03 | 5.13 | TO | 275 | 5.71E-03 | 57.4 | TO |
| 150 | 1.72E-03 | 9.12 | TO | 300 | 6.78E-03 | 75.37 | TO |

Table 5. Comparison of the thresholds generated by the CMB algorithm and by the NM algorithm. The data reported are the ratio of the threshold generated by the Central-Moment-Based algorithm with range partition refinement to those by the Naive Markov Method. The experiments use an analysis order of $n = 4$. For the Central-Moment-Based algorithm, the number of partitions is $b = 8$ for benchmarks with fewer than four input variables and fewer than ten operations, and range partitioning is disabled otherwise.

| Benchmark | Uniform | Normal | Laplace | Benchmark | Uniform | Normal | Laplace |
|---|---|---|---|---|---|---|---|
| bsplines0 | 52.54% | 55.64% | 56.13% | sine | 59.60% | 18.68% | 100.00% |
| bsplines1 | 55.53% | 24.12% | 61.00% | solvecubic | 53.23% | 32.37% | 67.01% |
| bsplines2 | 57.03% | 22.66% | 63.58% | sqrt | 67.57% | 16.04% | 24.52% |
| bsplines3 | 62.39% | 14.13% | 70.29% | traincars1 | 47.88% | 30.64% | 58.57% |
| classids0 | 60.58% | 51.23% | 61.21% | traincars2 | 64.89% | 71.70% | 81.63% |
| classids1 | 62.33% | 54.88% | 62.93% | traincars3 | 64.79% | 69.86% | 76.64% |
| classids2 | 57.75% | 51.31% | 58.23% | traincars4 | 60.93% | 61.85% | 69.57% |
| filters1 | 51.53% | 21.67% | 59.38% | trid1 | 64.44% | 22.14% | 12.94% |
| filters2 | 51.06% | 25.04% | 62.06% | trid2 | 76.47% | 80.13% | 98.02% |
| filters3 | 51.67% | 27.04% | 59.59% | trid3 | 75.54% | 77.22% | 94.69% |
| filters4 | 65.14% | 69.46% | 71.03% | trid4 | 77.58% | 74.85% | 93.11% |
| rigidbody1 | 68.82% | 16.23% | 91.15% | ksin | 77.40% | 93.96% | 66.14% |
| rigidbody2 | 93.82% | 97.45% | 100.00% | kcos | 49.79% | 51.60% | 16.20% |

We run both the NM algorithm and the CMB algorithm on the polynomial benchmarks with analysis order $n = 4$. For the CMB algorithm, we configure the number of partitions to $b = 8$ for benchmarks with fewer than four input variables and ten operations, while disabling range partition for the others. To avoid numerical underflow with double exponential distributions, we set the variance to $\sigma = 1$ when comparing with the NM algorithm. The resulting output threshold ratios are documented in Table 5. The threshold values and running times can be found in Appendix E.

We use the results from the NM algorithm as a baseline for comparison to evaluate the amount of refinement that can be achieved with the CMB algorithm. From the experimental data, we conclude that the CMB algorithm with range partition refinement significantly improves the derived thresholds. Among the 78 polynomial benchmarks, the CMB algorithm yields thresholds less than half of the baseline values in 16 cases. Furthermore, the average threshold refinement, defined as the ratio of the refined threshold to the baseline, is observed to be 62.7% for uniform distribution benchmarks,

Table 6. Comparison between thresholds generated by the CMB algorithm and by existing tools. The setup is the same as in Table 5. The CMB algorithm is run with analysis order $n = 4$ and number of partitions $b = 8$ for benchmarks with fewer than four input variables and fewer than ten operations, and range partition is disabled otherwise. The columns marked "CMB/PAF" are ratios of thresholds given by the CMB algorithm to those given by PAF, and "CMB/PrAn" is defined similarly. Ratios below 100% indicate that the CMB algorithm yields a tighter threshold than the competing tool, and is highlighted in bold. "TO" indicates timeout for competing tool.

| | Uniform | | Normal | | Laplace |
|---|---|---|---|---|---|
| Benchmarks | CMB/PAF | CMB/PrAn | CMB/PAF | CMB/PrAn | CMB/PAF |
| bsplines0 | 868.65% | 570.77% | 872.15% | 573.07% | 112.61% |
| bsplines1 | 307.53% | 302.65% | 125.27% | 110.95% | 119.86% |
| bsplines2 | 296.91% | 271.70% | 110.82% | 101.42% | 104.74% |
| bsplines3 | **97.87%** | **72.33%** | **20.69%** | **15.29%** | **3.27%** |
| classids0 | 210.68% | 168.79% | 107.64% | **53.82%** | 142.00% |
| classids1 | 223.45% | 176.76% | 104.85% | **59.03%** | 131.17% |
| classids2 | 235.18% | 162.27% | 122.34% | **61.97%** | 158.55% |
| filters1 | **72.56%** | **44.68%** | **25.16%** | **15.37%** | **7.99%** |
| filters2 | 103.66% | **81.39%** | **51.88%** | **31.49%** | **14.90%** |
| filters3 | 112.39% | **91.96%** | **51.71%** | **36.93%** | **52.48%** |
| filters4 | 174.22% | 139.04% | 141.93% | 113.27% | **25.38%** |
| rigidbody1 | 104.02% | 104.62% | **12.05%** | **0.43%** | **0.41%** |
| rigidbody2 | **85.20%** | 172.16% | **38.23%** | **0.24%** | **0.92%** |
| sine | 254.01% | 249.79% | **67.93%** | **66.80%** | **12.48%** |
| solvecubic | 111.24% | **98.51%** | **45.91%** | **15.54%** | 114.08% |
| sqrt | **97.40%** | **97.40%** | 194.31% | **0.31%** | 102.03% |
| traincars1 | 140.91% | 126.53% | **63.80%** | **26.89%** | **63.33%** |
| traincars2 | 222.12% | 173.68% | 124.59% | **33.91%** | **30.99%** |
| traincars3 | 164.00% | 125.33% | **65.69%** | **28.16%** | **12.85%** |
| traincars4 | 166.30% | 130.87% | **67.08%** | **25.87%** | **17.33%** |
| trid1 | 115.81% | 113.73% | **12.59%** | **0.03%** | **2.84%** |
| trid2 | 151.46% | 133.33% | **51.65%** | **0.11%** | **3.13%** |
| trid3 | 139.43% | 125.13% | **26.91%** | **0.09%** | **1.67%** |
| trid4 | 128.62% | 120.14% | **9.47%** | **0.08%** | **0.63%** |
| ksin | TO | 163.06% | TO | TO | TO |
| kcos | TO | 197.50% | TO | TO | TO |

47.4% for normal distribution benchmarks, and 66.8% for double exponential benchmarks. The results underscore the effectiveness of the CMB algorithm and the range partition technique in deriving tighter probabilistic error thresholds. Besides, the CMB's time efficiency is not much worse than the NM's, with all but three benchmarks finishing within 90 seconds.

We then perform a comparison between the overall results produced by ProbTaylor and the other tools over all polynomial benchmarks. The threshold ratios are presented in Table 6 [9]. ProbTaylor

[9]Readers might have noticed that ProbTaylor exhibits relatively better performance on the normal and Laplace distributions. This behavior is expected, as our approach fundamentally relies on the application of "concentration" inequalities. Consequently, the approach is particularly effective for distributions that are more "concentrated" – namely, those with relatively small variance.

Table 7. Extended CMB algorithm for fractional expressions. The experiments use an analysis order of $n = 2$, and number of partitions $b = 16$. "DZ" indicates that the competing tool encounters underflow-related NaNs (PAF, PrAn). "N/A" indicates lack of support for the corresponding distribution (PrAn laplace).

| | | ProbTaylor | | Comparison with PAF | | Comparison with PrAn | |
|---|---|---|---|---|---|---|---|
| Benchmark | Distribution | Time (s) | Threshold | Speedup | Threshold ratio | Speedup | Threshold ratio |
| doppler1 | uniform | 758.96 | 7.01E-06 | >4.74x | **8.82%** | DZ | DZ |
| doppler2 | uniform | 760.92 | 1.86E-05 | >4.73x | **13.01%** | DZ | DZ |
| doppler3 | uniform | 1487.33 | 4.29E-06 | >2.42x | **9.43%** | DZ | DZ |
| nonlin1 | uniform | 2.5 | 6.33E-08 | 336x | **94.34%** | 9.33x | **83.95%** |
| nonlin2 | uniform | 26.19 | 7.20E-06 | DZ | DZ | 8.60x | 196.72% |
| predator | uniform | 23.22 | 4.42E-07 | 41x | 444.22% | 3.66x | 442.00% |
| verhulst | uniform | 5.9 | 2.43E-07 | 59x | 141.28% | 5.09x | 135.00% |
| nonlin1 | normal | 2.52 | 6.02E-08 | 334x | **89.72%** | 6.32x | **79.84%** |
| nonlin2 | normal | 25.74 | 7.94E-05 | DZ | DZ | 26.48x | 2577.92% |
| predator | normal | 22.26 | 4.42E-07 | 43x | 444.22% | 7.81x | 442.00% |
| verhulst | normal | 6.12 | 2.43E-07 | 57x | 141.28% | 7.97x | 135.00% |
| nonlin1 | laplace | 2.52 | 6.04E-08 | 333x | **90.01%** | N/A | N/A |
| nonlin2 | laplace | 27.81 | 1.72E-05 | DZ | DZ | N/A | N/A |
| predator | laplace | 22.9 | 4.42E-07 | 41x | 446.92% | N/A | N/A |
| verhulst | laplace | 6.42 | 2.43E-07 | 54x | 143.79% | N/A | N/A |

achieves more accurate thresholds than PAF on 35 out of 72 polynomial benchmarks (PAF times out on 6), improving by more than an order of magnitude in 9 instances. Compared to PrAn, ProbTaylor produces tighter thresholds on 26 out of 50 polynomial benchmarks (PrAn times out on 2), improving by more than an order of magnitude in 7 instances. In terms of threshold accuracy, ProbTaylor never underperforms PAF or PrAn by more than an order of magnitude. The CMB algorithm with the range partition refinement remains faster than competing tools, with all benchmarks except `rigidbody1` finishing within 2 minutes.

Next, we evaluate the extension of the CMB algorithm for handling fractional expressions and compare the results of our tool to PAF and PrAn. For these experiments, we set the analysis order to $n = 2$ and number of partitions to $b = 16$. We run the extended CMB algorithm, PAF, and PrAn on the three `doppler` benchmarks from PAF with uniform distribution due to underflow issues in OCaml with the normal and Laplace distributions. We run four additional benchmarks using all three distributions. We document the running time and resulting thresholds, as well as comparison with PAF and PrAn in Table 7. "Speedup" and "threshold ratio" are defined in the same way as in Section 5.1. Cases where PAF or PrAn encounters NaNs caused by underflow are indicated with "DZ". PrAn is not run in Laplace distribution cases due to its lack of support.

From the experimental results on fractional expressions, ProbTaylor's CMB algorithm is substantially more time-efficient than PAF, achieving an average speedup of 109x. Moreover, ProbTaylor produces tighter thresholds than PAF on 6 out of 12 benchmarks that PAF is able to analyze, with 2 of these improvements exceeding an order of magnitude. When compared to PrAn, ProbTaylor also exhibits significant time-efficiency advantage, with an average speedup of 9.4x. Across all 11 benchmarks evaluated against PrAn, ProbTaylor successfully produces result for 3 benchmarks where PrAn crashes, and yields tighter threshold on 2 of them.

### 5.4 Comparison to FPTaylor (Answering RQ4)

Table 8. Comparison with FPTaylor on benchmarks with widened input range. The column "original" thresholds given on benchmarks with original input variable range. Here the "threshold ratio" is defined as the ratio of threshold generated by ProbTaylor with optimal analysis order $n$ and 99% confidence on normally distributed input variables and the deterministic bound generated by FPTaylor. The column "doubled" and "quadrupled" correspond to the threshold ratios on benchmarks with doubled and quadrupled input ranges respectively. Lower numbers indicate better result by probabilistic analysis.

| Threshold ratio $\left(\frac{\text{ProbTaylor}}{\text{FPTaylor}}\right)$ on widened input ranges on truncated standard normal distribution | | | | | | | |
|---|---|---|---|---|---|---|---|
| benchmark | original | doubled | quadrupled | benchmark | original | doubled | quadrupled |
| bsplines0 | 1321.68% | 1910.24% | 245.03% | rigidbody2 | 0.12% | 0.01% | 0.00% |
| bsplines1 | 404.28% | 320.28% | 88.17% | sine | 287.82% | 234.48% | 14.50% |
| bsplines2 | 389.23% | 272.48% | 74.80% | solvecubic | 30.68% | DZ | DZ |
| bsplines3 | 146.45% | 95.56% | 43.70% | sqrt | 1.97% | 0.12% | 0.01% |
| classids0 | 66.28% | 45.69% | DZ | traincars1 | 54.77% | 26.62% | 13.63% |
| classids1 | 73.76% | 47.38% | DZ | traincars2 | 43.02% | 21.59% | 10.82% |
| classids2 | 86.80% | 59.42% | DZ | traincars3 | 32.89% | 16.52% | 8.25% |
| filters1 | 66.96% | 51.60% | 26.15% | traincars4 | 38.40% | 19.81% | 9.90% |
| filters2 | 94.33% | 65.41% | 33.44% | trid1 | 0.12% | 0.03% | 0.01% |
| filters3 | 103.14% | 67.42% | 33.89% | trid2 | 0.12% | 0.03% | 0.01% |
| filters4 | 128.35% | 82.18% | 41.24% | trid3 | 0.11% | 0.03% | 0.01% |
| rigidbody1 | 2.65% | 0.66% | 0.17% | trid4 | 0.13% | 0.03% | 0.01% |

To demonstrate that ProbTaylor is capable of producing thresholds in situations where deterministic analysis is too conservative, we adopt the standard deterministic analysis tool FPTaylor [30] as a baseline. We conduct experiments using benchmarks with original distribution ranges from Table 1, Table 2 and Table 3 (excluding two realistic benchmarks from `fdlibm`, which are designed specifically for certain input ranges). We derive additional benchmarks by doubling and quadrupling (the end-points of) the ranges of the distributions. For example, for a uniform distribution with range $(-2, 4)$, we double the range to get the uniform distribution over $(-4, 8)$, and quadruple the range to get the uniform distribution over $(-8, 16)$. The cases for truncated normal and double exponential distributions are similar: we widen the range, and keep mean and variance unchanged.

The experimental results for standard normal distribution are collected in Table 8. Here, "DZ" indicates NaN produced by ProbTaylor. The ProbTaylor configuration follows the same settings as described in Section 5.1, with single precision, target confidence level 99%, and the same strategy – reporting the optimal threshold among a set of analysis orders with a 90-second timeout per order.

We observe that ProbTaylor produces significantly tighter thresholds compared to FPTaylor with expanded input ranges. On average, doubling the input range results in a 45% decrease in threshold ratio, and quadrupling the range leads to a 82% decrease compared to original benchmarks. We focus only on comparing accuracy rather than runtime as (i) both tools run within 200 seconds for every benchmark and (ii) they have different objectives (deterministic vs. probabilistic analysis).

**Summary of Experimental Evaluations.** When employing the NM algorithm, ProbTaylor is significantly faster than both PAF and PrAn while producing thresholds of comparable precision. The NM algorithm itself scales well, capable of handling polynomial expressions with hundreds of operations within minutes. However, the analysis of second-order error remains a scalability

bottleneck in the overall toolchain. Nevertheless, considering the CMB algorithm and its extensions, ProbTaylor consistently remains faster than existing tools. In terms of accuracy, the extended variants yield tighter thresholds than PAF and PrAn on roughly half of the benchmarks. Moreover, comparisons against deterministic analysis show our probabilistic approach yields tighter thresholds under widened input ranges, demonstrating robustness with large input distribution's support.

## 6 Related Works

Static analysis of round-off errors is an active research area. Most existing approaches do not involve probabilistic inputs, but rather are deterministic methods and produce bounds valid even in worst-case scenarios. Many of these, including Gappa [16], Gappa++ [24], FLUCTUAT [17], RangeLab [27], Rosa [15] and Daisy [14], use abstract interpretation techniques [12], and employ abstract domains such as intervals [28], affine forms [22], or polyhedra [9] to analyze errors.

An alternative approach, used by PRECiSA [33] and Real2Float [26], formulates the bounding of round-off errors as an optimization problem. Lee et al. [23] also employ optimization-based techniques and deal with the interplay between floating-point and bit-level operations. FPTaylor [30] leverages symbolic Taylor expansion, and optimizes its lower-order terms. Our approach draws on FPTaylor's idea for the initial transformation and over-approximation of the target problem.

Recently, researchers have started to focus on probabilistic analysis of round-off error, aiming to produce bounds that are less pessimistic yet still valid with high probability. PrAn [25] is the first work we are aware of that provides probabilistic analysis of round-off errors with specified input distributions. It extends probabilistic affine arithmetic [3] and utilizes probabilistic interval subdivision techniques. PAF [11] is the current state-of-the-art, deriving tight bounds on most benchmarks, but is very slow even for moderate size benchmarks. There are also results on analyzing probabilistic programs via concentration inequalities [4–8, 21, 31, 34, 35]. Our result is orthogonal to them as we focus on floating-point arithmetic and deal with absolute values in the Taylor expansion.

## 7 Conclusion and Future Work

In this paper, we propose a novel approach to sound probabilistic analysis of floating-point round-off errors. Our method employs Taylor expansion and concentration inequalities, combined with a positive-negative decomposition technique, to compute rigorous probabilistic thresholds of round-off errors. We implemented our algorithms in a prototype tool, ProbTaylor, and evaluated its performance on a wide range of benchmarks. Experimental results demonstrate that ProbTaylor yields thresholds at least comparable to, and sometimes significantly more accurate than, those produced by existing tools such as PAF and PrAn, while substantially more time-efficient. Besides, it potentially scales well to large benchmarks and can rule out unlikely worst-case scenarios that often hinder deterministic analysis. For future work, we plan to expand our supported range of operations, including trigonometrics, logarithms, exponentials, etc. Another possible direction would be to use concentration bounds other than Markov's inequality, such as Chernoff bounds.

*Data Availability Statement.* Our artifact can be found at doi.org/10.5281/zenodo.18499134. It supports an implementation of ProbTaylor, and its comparison with PrAn (PAF is out of scope because it takes several days to run), and experiments for Sections 5.1, 5.2 and 5.3. Experiments in Section 5.4 is out of scope, but FPTaylor can be run at monadius.github.io/FPTaylorJS.

## Acknowledgments

We thank Karthik Duraisamy, Sahil Bhola and Daisuke Uchida for fruitful discussions, as well as the anonymous reviewers for helpful comments. This research was supported in part by NSF grants CCF-2219997, CCF-2348706 and CCF-2446214.

## References


[1] Mark Baranowski and Ian Briggs. [n. d.]. GELPIA: Global Extrema Locator Parallelization for Interval Arithmetic. https://github.com/soarlab/gelpia.

[2] Joseph K Blitzstein and Jessica Hwang. 2019. *Introduction to probability*. Chapman and Hall/CRC.

[3] Olivier Bouissou, Eric Goubault, Jean Goubault-Larrecq, and Sylvie Putot. 2012. A generalization of p-boxes to affine arithmetic. *Computing* 94 (2012), 189–201.

[4] Olivier Bouissou, Eric Goubault, Sylvie Putot, Aleksandar Chakarov, and Sriram Sankaranarayanan. 2016. Uncertainty propagation using probabilistic affine forms and concentration of measure inequalities. In *Tools and Algorithms for the Construction and Analysis of Systems: 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings 22*. Springer, 225–243.

[5] Aleksandar Chakarov and Sriram Sankaranarayanan. 2013. Probabilistic Program Analysis with Martingales. In *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 8044)*, Natasha Sharygina and Helmut Veith (Eds.). Springer, 511–526. doi:10.1007/978-3-642-39799-8_34

[6] Krishnendu Chatterjee, Hongfei Fu, Petr Novotný, and Rouzbeh Hasheminezhad. 2018. Algorithmic Analysis of Qualitative and Quantitative Termination Problems for Affine Probabilistic Programs. *ACM Trans. Program. Lang. Syst.* 40, 2 (2018), 7:1–7:45. doi:10.1145/3174800

[7] Krishnendu Chatterjee, Amir Kafshdar Goharshady, Tobias Meggendorfer, and Dorde Zikelic. 2024. Quantitative Bounds on Resource Usage of Probabilistic Programs. *Proc. ACM Program. Lang.* 8, OOPSLA1 (2024), 362–391. doi:10.1145/3649824

[8] Krishnendu Chatterjee, Petr Novotný, and Dorde Zikelic. 2017. Stochastic invariants for probabilistic termination. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, Giuseppe Castagna and Andrew D. Gordon (Eds.). ACM, 145–160. doi:10.1145/3009837.3009873

[9] Liqian Chen, Antoine Miné, and Patrick Cousot. 2008. A sound floating-point polyhedra abstract domain. In *Asian Symposium on Programming Languages and Systems*. Springer, 3–18.

[10] Erhan Çinlar. 2011. *Probability and stochastics*. Springer.

[11] George Constantinides, Fredrik Dahlqvist, Zvonimir Rakamarić, and Rocco Salvia. 2021. Rigorous roundoff error analysis of probabilistic floating-point computations. In *International Conference on Computer Aided Verification*. Springer, 626–650.

[12] Patrick Cousot and Radhia Cousot. 1977. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. 238–252.

[13] Nasrine Damouche, Matthieu Martel, Pavel Panchekha, Chen Qiu, Alexander Sanchez-Stern, and Zachary Tatlock. 2017. Toward a standard benchmark format and suite for floating-point analysis. In *Numerical Software Verification: 9th International Workshop, NSV 2016, Toronto, ON, Canada, July 17-18, 2016, Revised Selected Papers 9*. Springer, 63–77.

[14] Eva Darulova, Anastasiia Izycheva, Fariha Nasir, Fabian Ritter, Heiko Becker, and Robert Bastian. 2018. Daisy-framework for analysis and optimization of numerical programs (tool paper). In *Tools and Algorithms for the Construction and Analysis of Systems: 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part I 24*. Springer, 270–287.

[15] Eva Darulova and Viktor Kuncak. 2014. Sound compilation of reals. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. 235–248.

[16] Marc Daumas and Guillaume Melquiond. 2010. Certification of bounds on expressions involving rounded operators. *ACM Transactions on Mathematical Software (TOMS)* 37, 1 (2010), 1–20.

[17] David Delmas, Eric Goubault, Sylvie Putot, Jean Souyris, Karim Tekkal, and Franck Védrine. 2009. Towards an industrial use of FLUCTUAT on safety-critical avionics software. In *International Workshop on Formal Methods for Industrial Critical Systems*. Springer, 53–69.

[18] Scott Ferson, Vladik Kreinovich, Lev Ginzburg, Davis S Myers, and Kari Sentz. 2003. *Constructing probability boxes and Dempster-Shafer structures*. Number 4015. Sandia National Laboratories.

[19] David Goldberg. 1991. What every computer scientist should know about floating-point arithmetic. *ACM computing surveys (CSUR)* 23, 1 (1991), 5–48.

[20] William Kahan. 1996. IEEE standard 754 for binary floating-point arithmetic. *Lecture Notes on the Status of IEEE* 754, 94720-1776 (1996), 11.

[21] Satoshi Kura, Natsuki Urabe, and Ichiro Hasuo. 2019. Tail Probabilities for Randomized Program Runtimes via Martingales for Higher Moments. In *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 11428)*, Tomás

Vojnar and Lijun Zhang (Eds.). Springer, 135–153. doi:10.1007/978-3-030-17465-1_8

[22] COMBA Joao LD. 1993. Affine arithmetic and its applications to computer graphics. *SIBGRAPI'93, Recife, PE (Brazil), October* (1993).

[23] Wonyeol Lee, Rahul Sharma, and Alex Aiken. 2016. Verifying bit-manipulations of floating-point. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 70–84.

[24] Michael D Linderman, Matthew Ho, David L Dill, Teresa H Meng, and Garry P Nolan. 2010. Towards program optimization through automated analysis of numerical precision. In *Proceedings of the 8th annual IEEE/ACM international symposium on Code generation and optimization*. 230–237.

[25] Debasmita Lohar, Milos Prokop, and Eva Darulova. 2019. Sound probabilistic numerical error analysis. In *International Conference on Integrated Formal Methods*. Springer, 322–340.

[26] Victor Magron, George Constantinides, and Alastair Donaldson. 2017. Certified roundoff error bounds using semidefinite programming. *ACM Transactions on Mathematical Software (TOMS)* 43, 4 (2017), 1–31.

[27] Matthieu Martel. 2011. RangeLab: a static-analyzer to bound the accuracy of finite-precision computations. In *2011 13th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*. IEEE, 118–122.

[28] Ramon E Moore. 1966. *Interval analysis*. Vol. 4. prentice-Hall Englewood Cliffs.

[29] Walter Rudin et al. 1964. *Principles of mathematical analysis*. Vol. 3. McGraw-hill New York.

[30] Alexey Solovyev, Marek S Baranowski, Ian Briggs, Charles Jacobsen, Zvonimir Rakamarić, and Ganesh Gopalakrishnan. 2018. Rigorous estimation of floating-point round-off errors with symbolic taylor expansions. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 41, 1 (2018), 1–39.

[31] Yican Sun, Hongfei Fu, Krishnendu Chatterjee, and Amir Kafshdar Goharshady. 2023. Automated Tail Bound Analysis for Probabilistic Recurrence Relations. In *Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 13966)*, Constantin Enea and Akash Lal (Eds.). Springer, 16–39. doi:10.1007/978-3-031-37709-9_2

[32] Mohit Tekriwal. 2023. *A Mechanized Error Analysis Framework for End-to-End Verification of Numerical Programs*. Ph. D. Dissertation.

[33] Laura Titolo, Marco A Feliú, Mariano Moscato, and César A Muñoz. 2018. An abstract interpretation framework for the round-off error analysis of floating-point programs. In *Verification, Model Checking, and Abstract Interpretation: 19th International Conference, VMCAI 2018, Los Angeles, CA, USA, January 7-9, 2018, Proceedings 19*. Springer, 516–537.

[34] Di Wang, Jan Hoffmann, and Thomas W. Reps. 2021. Central moment analysis for cost accumulators in probabilistic programs. In *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021*, Stephen N. Freund and Eran Yahav (Eds.). ACM, 559–573. doi:10.1145/3453483.3454062

[35] Jinyi Wang, Yican Sun, Hongfei Fu, Krishnendu Chatterjee, and Amir Kafshdar Goharshady. 2021. Quantitative analysis of assertion violations in probabilistic programs. In *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021*, Stephen N. Freund and Eran Yahav (Eds.). ACM, 1171–1186. doi:10.1145/3453483.3454102

[36] David Williams. 1991. *Probability with Martingales*. Cambridge University Press.

## A Symbolic Computation of Expectations

As is mentioned in Section 5, ProbTaylor computes expectations symbolically rather than numerically. In this section, we provide a detailed explanation of the mathematical derivations underlying these symbolic computations.

Our symbolic expectation computation deals with expressions represented in polynomial form. We compute the expectation of each term individually, and sum them to obtain the expectation of the entire expression. Within each term, the factors are grouped based on their original variables (e.g., $x$ and $x^+$ has the same original variable, and thus belong to the same factor group). Each group can then be simplified to just one factor (or even zero). If the group involves more than one factor, it has one of the following four forms:

- $x^n(x^+)^m$. This can be simplified to $(x^+)^{n+m}$ since $x^n$ just gets multiplied by zero when $x < 0$.
- $x^n(x^-)^m$. This can be simplified to $(-1)^n(x^-)^{n+m}$ for similar reason, and the fact that $x^- = -x$ when $x < 0$ contributes to the $(-1)^n$ factor.
- $(x^+)^n(x^-)^m$. This term is always equal to zero, since at least one factor is zero no matter what value $x$ takes.
- $(x^+)^n(x^-)^m x^k$. This term is also always equal to zero, for the same reason as above.

Therefore, our task reduces to calculating the expectation or higher moments or a single variable and multiply them together to obtain the expectation of the whole term. The computation of higher moments for single factors following different distributions are presented in the following subsections.

### A.1 Uniform Distribution

When $x$ is uniformly distributed on the interval $[a, b]$, its expectation and higher moments can be computed as follows.

$$E[x^k] = \int_a^b \frac{x^k}{b-a} dx = \frac{b^{k+1} - a^{k+1}}{(k+1)(b-a)}.$$

If the range $[a, b]$ crosses zero, i.e., $a < 0 < b$, then we can compute the higher moments of the positive component and negative component as follows.

$$E[x_+^k] = \int_a^0 \frac{0}{b-a} dx + \int_0^b \frac{x^k}{b-a} dx = \frac{b^{k+1}}{(k+1)(b-a)}.$$

$$E[x_-^k] = \int_a^0 \frac{(-x)^k}{b-a} dx + \int_0^b \frac{0}{b-a} dx = \frac{(-a)^{k+1}}{(k+1)(b-a)}.$$

If the interval $[a, b]$ does not cross zero, then either $x = x^+$ or $x = -x^-$ holds, and the computation can be simplified accordingly.

### A.2 Truncated Standard Normal Distribution

When $x$ follows the standard normal distribution truncated to interval $[a, b]$, its probabilistic distribution function (PDF) is expressed as

$$f(x) = \begin{cases} \dfrac{\varphi(x)}{\Phi(b) - \Phi(a)}, & x \in [a, b] \\ 0, & \text{otherwise} \end{cases},$$

where $\varphi(x) = \frac{1}{\sqrt{2\pi}} \exp(-\frac{x^2}{2})$ is the probabilistic distribution function of standard normal distribution, and $\Phi(x) = \int_{-\infty}^{x} \varphi(t) dt$ is the cummulative distribution function (CDF) of the standard normal

distribution. It is noteworthy that the derivative of $\varphi(x)$ satisfies $\varphi'(x) = \frac{1}{\sqrt{2\pi}} \exp(-\frac{x^2}{2})(-x) = -x\varphi(x)$.

Let $I_k = \int_a^b x^k \varphi(x) dx$, then the $k$-th moment of $x$ is given by

$$\mathbb{E}[x^k] = \int_a^b x^k f(x) dx = \frac{1}{\Phi(b) - \Phi(a)} \int_a^b x^k \varphi(x) dx = \frac{I_k}{\Phi(b) - \Phi(a)}.$$

Using integration by parts, let $u = x^{k-1}$ and $dv = x\varphi(x)dx = -\varphi'(x)dx$, and we have $du = (k-1)x^{k-2}$ and $v = -\varphi(x)$. Apply the integration by parts formula, we derive

$$I_k = uv|_a^b - \int_a^b v du = \left[-x^{k-1}\varphi(x)\right]_a^b + (k-1)I_{k-2}.$$

Therefore, the $k$-th moment of $x$ can be computed recursively with

$$\mathbb{E}[x^k] = \frac{a^{k-1}\varphi(a) - b^{k-1}\varphi(b)}{\Phi(b) - \Phi(a)} + (k-1)\mathbb{E}[x^{k-2}].$$

The base cases to start with are as follows.

$$\mathbb{E}[x^0] = E[1] = 1.$$

$$\mathbb{E}[x^1] = \frac{1}{\Phi(b) - \Phi(a)} \int_a^b x\varphi(x) dx = \frac{1}{\Phi(b) - \Phi(a)} \left[-\varphi(x)\right]_a^b = \frac{\varphi(a) - \varphi(b)}{\Phi(b) - \Phi(a)}.$$

Using this recursive formula, the moments of $x$ for any order $k$ can be efficiently computed for the truncated standard normal distribution.

Next we turn to compute the higher moment for the positive components $x^+$ and negative components $x^-$. Only the cases where $a < 0 < b$ are discussed here, for otherwise, the computation of $\mathbb{E}[x^k]$ can be used directly. The derivations are similar as for computing $\mathbb{E}[x^k]$ and we only present the recursion formula and the base cases below.

$$\mathbb{E}[x_+^k] = \frac{-b^{k-1}\varphi(b)}{\Phi(b) - \Phi(a)} + (k-1)\mathbb{E}[x_+^{k-2}].$$

$$\mathbb{E}[x_+^0] = \frac{\Phi(b) - \Phi(0)}{\Phi(b) - \Phi(a)}, \quad \mathbb{E}[x_+^1] = \frac{-\varphi(b) + \varphi(0)}{\Phi(b) - \Phi(a)}.$$

$$\mathbb{E}[x_-^k] = \frac{(-1)^k a^{k-1}\varphi(a)}{\Phi(b) - \Phi(a)} + (k-1)\mathbb{E}[x_-^{k-2}].$$

$$\mathbb{E}[x_-^0] = \frac{\Phi(0) - \Phi(a)}{\Phi(b) - \Phi(a)}, \quad \mathbb{E}[x_-^1] = \frac{\varphi(0) - \varphi(a)}{\Phi(b) - \Phi(a)}.$$

### A.3 Truncated Double Exponential Distribution

When a random variable $x$ follows the double exponenetial distribution (also known as Laplace distribution) parameterized by $\sigma$ truncated to the interval $[a, b]$, its PDF is expressed as

$$f(x) = \begin{cases} \dfrac{\varphi(x)}{\Phi(b) - \Phi(a)}, & x \in [a, b] \\ 0, & \text{otherwise} \end{cases},$$

where $\varphi(x) = \frac{1}{2\sigma}\exp\left(-\frac{|x|}{\sigma}\right)$ is the PDF of double exponential distribution, and the CDF is

$$\Phi(x) = \begin{cases} \frac{1}{2}\exp\left(\frac{x}{b}\right), & x \le 0 \\ 1 - \frac{1}{2}\exp\left(-\frac{x}{b}\right), & x > 0 \end{cases}.$$

We first define a helper integration and study its computation. Define $I(c,k) = \int_0^c x^k e^{-x} dx$. Let $u = x^k$ and $dv = e^{-x}dx$, and therefore $du = kx^{k-1}$ and $v = -e^{-x}$. By applying integration by parts,

$$I(c,k) = \int_0^c u dv = [uv]_0^c - \int_0^c v du = kI(c,k-1) - c^k e^{-c}.$$

We have a recursive relation for computing $I(c,k)$. The base case can be solved as follows:

$$I(c,0) = \int_0^c e^{-x} dx = 1 - e^{-c}.$$

Then we use $I(c,k)$ as a tool for computing $\mathbb{E}[x^k]$. We first discuss the case when $a < 0 < b$. Let $I_1 = \int_a^0 x^k \varphi(x) dx$ and $I_2 = \int_0^b x^k \varphi(x) dx$, and the $k$-th moment of $x$ is given by

$$\mathbb{E}[x^k] = \frac{1}{\Phi(b) - \Phi(a)} \int_a^b x^k \varphi(x) dx = \frac{1}{\Phi(b) - \Phi(a)} (I_1 + I_2).$$

$I_1$ can be computed as

$$\begin{aligned} I_1 &= \int_a^0 x^k \varphi(x) dx = \frac{1}{2\sigma} \int_a^0 x^k \exp\left(\frac{x}{\sigma}\right) dx \\ &= \frac{1}{2}(-1)^{k+1}\sigma^k \int_{-a/\sigma}^0 u^k \exp(-u) du \quad (u \triangleq -\frac{x}{\sigma}) \\ &= \frac{1}{2}(-\sigma)^k \int_0^{-a/\sigma} u^k \exp(-u) du = \frac{1}{2}(-\sigma)^k \cdot I\left(-\frac{a}{\sigma}, k\right). \end{aligned}$$

$I_2$ can be similarly computed as

$$I_2 = \frac{1}{2}\sigma^k \cdot I\left(\frac{b}{\sigma}, k\right).$$

By computing the corresponding $I(c,k)$ values and plugging them back, we can obtain the result of $\mathbb{E}[x^k]$.

When $0 < a < b$, define and compute $I_1, I_2$ as follows.

- $I_1 \triangleq \int_0^b x^k \varphi(x) dx = \frac{1}{2}\sigma^k \cdot I\left(\frac{b}{\sigma}, k\right)$.
- $I_2 \triangleq \int_0^a x^k \varphi(x) dx = \frac{1}{2}\sigma^k \cdot I\left(\frac{a}{\sigma}, k\right)$.

Thus, $\mathbb{E}[x^k] = \frac{1}{\Phi(b)-\Phi(a)}(I_1 - I_2)$.

When $a < b < 0$, define and compute $I_1, I_2$ as follows.

- $I_1 \triangleq \int_a^0 x^k \varphi(x) dx = \frac{1}{2}(-\sigma)^k \cdot I\left(-\frac{a}{\sigma}, k\right)$.
- $I_2 \triangleq \int_b^0 x^k \varphi(x) dx = \frac{1}{2}(-\sigma)^k \cdot I\left(-\frac{b}{\sigma}, k\right)$.

Thus, $\mathbb{E}[x^k] = \frac{1}{\Phi(b)-\Phi(a)}(I_1 - I_2)$.

Again, we only talk about the computation of reasoning about positive and negative parts when $[a,b]$ crosses zero. The derivations are similar as for the original variable, and the result is as

follows.

$$E[x_+^k] = \frac{1}{\Phi(b)-\Phi(a)} \int_0^b x^k f(x)dx = \frac{1}{\Phi(b)-\Phi(a)} \cdot \left(\frac{1}{2}\sigma^k \cdot I\left(\frac{b}{\sigma}, k\right)\right).$$

$$E[x_-^k] = \frac{1}{\Phi(b)-\Phi(a)} \int_a^0 (-x)^k f(x)dx = \frac{1}{\Phi(b)-\Phi(a)} \cdot \left(\frac{1}{2}\sigma^k \cdot I\left(-\frac{a}{\sigma}, k\right)\right).$$

## B Full Proofs from Section 4

Full Proof of Theorem 4.1. We first establish the following two facts after the execution reaches line 11: (a) $h_i(\mathbf{x})$ is equivalent to the original input expression; (b) each term in $h_i(\mathbf{x})$ has the same sign as its coefficient (the constant factor of the term). Fact (a) is evident because the only two possible modifications to $h_i(\mathbf{x})$ are replacing $x_j^a$ by either $(x_j^+)^a - (x_j^-)^a$ or $(-x_j^-)^a$. Given that $x_j = x_j^+ - x_j^-$ and $x_j^{a-1}$ is non-negative for odd values of $a$, $x_j^a = x_j^{a-1}x_j^+ - x_j^{a-1}x_j^- = (x_j^+)^a - (x_j^-)^a$, and thus the first possible replacement preserves the value. When $x_j$ is always non-positive, $x_j = x_j^-$ holds, and thus the second possible replacement is an equivalent transformation. Then we examine the monomial part of each term. Note that all factors fall into one of the three categories: an even power of the original variable, the positive component, or the negative component. Each of the three is non-negative, and thus their product can be proved non-negative. Thus, the monomial part of all terms in $h_i(\mathbf{x})$ is non-negative, and fact (b) immediately follows.

Then we analyze the values of $h_i^+(\mathbf{x})$ and $h_i^-(\mathbf{x})$ after the execution of line 22, and show that it has the two properties similar to the decomposition of variables:

- Both $h_i^+(\mathbf{x})$ and $h_i^-(\mathbf{x})$ are non-negative. When a term $t$ has positive coefficient, indicating that $t$ is non-negative (by fact (b)), it is added to $h_i^+$, thereby preserving the non-negativity of $h_i^+$. On the other hand, when $t$ has negative coefficient, the term $t$ is non-positive. Subtracting such a term form $h_i^-$ ensures that its non-negativity is maintained.
- $h_i(\mathbf{x}) = h_i^+(\mathbf{x}) - h_i^-(\mathbf{x})$. Each term in $h_i(\mathbf{x})$ is assigned either to $h_i^+(\mathbf{x})$ as it appears, or to $h_i^-(\mathbf{x})$ as its negation. Thus, subtracting $h_i^-(\mathbf{x})$ from $h_i^+(\mathbf{x})$ reconstructs the original value of $h_i(\mathbf{x})$, thereby verifying the equality.

Therefore, we have the following relaxation:

$$\begin{aligned} |h_i(\mathbf{x})e_i| &= |h_i^+(\mathbf{x})e_i - h_i^-(\mathbf{x})e_i| \le |h_i^+(\mathbf{x})e_i| + |h_i^-(\mathbf{x})e_i| \\ &= (h_i^+(\mathbf{x}) + h_i^-(\mathbf{x}))|e_i| \le (h_i^+(\mathbf{x}) + h_i^-(\mathbf{x}))\epsilon. \end{aligned}$$

The second inequality is an application of the triangular inequality. The third equality is because $h_i^+(\mathbf{x})$ and $h_i^-(\mathbf{x})$ are non-negative. At the end of the algorithm, $p(\mathbf{x}) = \sum_{i=1}^k (h_i^+(\mathbf{x}) + h_i^-(\mathbf{x}))$. Thus, for the entire first-order term we have

$$\sum_{i=1}^k |h_i(\mathbf{x})e_i| \le \sum_{i=1}^k (h_i^+(\mathbf{x}) + h_i^-(\mathbf{x}))\epsilon = p(\mathbf{x}) \cdot \epsilon.$$

□

Full Proof of Lemma 4.5. Given that $f(\mathbf{x})$ has the form $N(\mathbf{x})/Q(\mathbf{x})$, its floating-point model should observe the following structure: $\tilde{f}(\mathbf{x}, \mathbf{e}, \mathbf{d}) = \tilde{N}(\mathbf{x}, \mathbf{e}, \mathbf{d})/\tilde{Q}(\mathbf{x}, \mathbf{e}, \mathbf{d}) \cdot (1 + e_k) + d_k$, where $k$ is the total number of arithmetic operations in $f(\mathbf{x})$ and $e_k, d_k$ are the error variables for the outmost division. We observe the fact that each relative error term $e_i$ appears exactly once in the symbolic expression of $\tilde{f}(\mathbf{x}, \mathbf{e}, \mathbf{d})$. More specifically, each $e_i$ contributes to one of the following components: (i) the floating-point model of the numerator $\tilde{N}(\mathbf{x}, \mathbf{e}, \mathbf{d})$; (ii) the floating-point model of the denominator $\tilde{Q}(\mathbf{x}, \mathbf{e}, \mathbf{d})$; or (iii) the error term $e_k$ related to division $N(\mathbf{x})/Q(\mathbf{x})$. To demonstrate

that for any index $i$, $\frac{\partial \tilde{f}}{\partial e_i}(\mathbf{x},\mathbf{0},\mathbf{0}) \cdot (Q(\mathbf{x}))^2$ can be reduced into polynomial form, we analyze the three cases. Throughout, we make frequent use of the identity $\tilde{f}(\mathbf{x},\mathbf{0},\mathbf{0}) = f(\mathbf{x})$.

(i) When $e_i$ appears in the numerator $\tilde{N}(\mathbf{x},\mathbf{e},\mathbf{d})$, we have $\frac{\partial \tilde{f}}{\partial e_i}(\mathbf{x},\mathbf{0},\mathbf{0}) \cdot (Q(\mathbf{x}))^2 = \frac{1}{Q(\mathbf{x})} \cdot \frac{\partial \tilde{N}}{\partial e_i}(\mathbf{x},\mathbf{0},\mathbf{0}) \cdot (Q(\mathbf{x}))^2 = \frac{\partial \tilde{N}}{\partial e_i}(\mathbf{x},\mathbf{0},\mathbf{0}) \cdot Q(\mathbf{x})$ which is division-free since $N(\mathbf{x})$ and $Q(\mathbf{x})$ are both division-free.

(ii) When $e_i$ appears in the denominator $\tilde{Q}(\mathbf{x},\mathbf{e},\mathbf{d})$, we have $\frac{\partial \tilde{f}}{\partial e_i}(\mathbf{x},\mathbf{0},\mathbf{0}) \cdot (Q(\mathbf{x}))^2 = \frac{-N(\mathbf{x})}{(Q(\mathbf{x}))^2} \cdot \frac{\partial \tilde{Q}}{\partial e_i}(\mathbf{x},\mathbf{0},\mathbf{0}) \cdot (Q(\mathbf{x}))^2 = -N(\mathbf{x}) \cdot \frac{\partial \tilde{Q}}{\partial e_i}(\mathbf{x},\mathbf{0},\mathbf{0})$, which is also division-free.

(iii) When $e_i$ is just $e_k$, then $\frac{\partial \tilde{f}}{\partial e_i}(\mathbf{x},\mathbf{0},\mathbf{0}) \cdot (Q(\mathbf{x}))^2 = \frac{N(\mathbf{x})}{Q(\mathbf{x})} \cdot (Q(\mathbf{x}))^2 = N(\mathbf{x}) \cdot Q(\mathbf{x})$, clearly division-free.

Therefore, in each of the three possible cases, $h_i(\mathbf{x}) = \frac{\partial \tilde{f}}{\partial e_i}(\mathbf{x},\mathbf{0},\mathbf{0}) \cdot (Q(\mathbf{x}))^2$ is division-free, and is thus reducible to a polynomial expression. □

## C Monotonicity of $\mathrm{flag}_u$ in Algorithm 3 with regard to $u$

As stated in Theorem 4.3, $\mathrm{flag}_u$ is defined by

$$\mathrm{flag}_u = \frac{\mathbb{E}[K_u(\mathbf{x}) - \mu)^n]}{\mu^n},$$

where $K_u(\mathbf{x}) = p(\mathbf{x}) \cdot \epsilon - u$ and $\mu = \mathbb{E}[K_u(\mathbf{x})]$. We perform the following transformations on $\mathrm{flag}_u$:

$$\begin{aligned}\mathrm{flag}_u &= \frac{\mathbb{E}[K_u(\mathbf{x}) - \mu)^n]}{\mu^n} = \frac{\mathbb{E}[(p(\mathbf{x}) \cdot \epsilon - \mathbb{E}[p(\mathbf{x}) \cdot \epsilon] + \mathbb{E}[u])^n]}{(\mathbb{E}[p(\mathbf{x}) \cdot \epsilon] - u)^n} \\ &= \frac{\mathbb{E}[(p(\mathbf{x}) \cdot \epsilon - \mathbb{E}[p(\mathbf{x}) \cdot \epsilon])^n]}{(\mathbb{E}[p(\mathbf{x}) \cdot \epsilon] - u)^n}.\end{aligned}$$

Observe that the numerator is constant with respect to $u$, non-negative since $n \geq 0$, while the denominator is monotonically increasing in $u$ when $u > \mathbb{E}[p(\mathbf{x}) \cdot \epsilon] = \ell$. Therefore, $\mathrm{flag}_u$ is monotonically decreasing when $u > \ell$.

## D Brute-Force algorithm for bounding the second-order error

The second-order error expressions we aim to bound do not involve division by non-constant terms. Therefore, for a given expression $f(\mathbf{x})$, we derive a deterministic upper bound $\mathrm{bound}(f)$ for $|f|$ via structural recursion and pattern matching on the syntactic form of $f$:

- If $f(\mathbf{x}) = c$, where $c$ is a constant, then $\mathrm{bound}(f) = |c|$.
- If $f(\mathbf{x}) = x$, where $x$ is a variable with range $[a, b]$, then $\mathrm{bound}(f) = \max\{|a|, |b|\}$.
- If $f(\mathbf{x}) = f_1(\mathbf{x}) + f_2(\mathbf{x})$, then $\mathrm{bound}(f) = \mathrm{bound}(f_1) + \mathrm{bound}(f_2)$.
- If $f(\mathbf{x}) = f_1(\mathbf{x}) - f_2(\mathbf{x})$, then $\mathrm{bound}(f) = \mathrm{bound}(f_1) + \mathrm{bound}(f_2)$.
- If $f(\mathbf{x}) = f_1(\mathbf{x}) \cdot f_2(\mathbf{x})$, then $\mathrm{bound}(f) = \mathrm{bound}(f_1) \cdot \mathrm{bound}(f_2)$.
- If $f(\mathbf{x}) = f_1(\mathbf{x})/c$, where $c$ is a constant, then $\mathrm{bound}(f) = \mathrm{bound}(f_1)/|c|$.

The soundness of this algorithm follows directly from applications of triangular inequality and basic properties of absolute values under arithmetic operations.

## E Full Experimental Results

Due to space limitations, full experimental results are recorded here. Tables 9, 10, and 11 report thresholds and running times of PAF and PrAn on benchmarks with uniform distribution, truncated standard normal distribution, and truncated double exponential distribution, respectively. Table 13 records the running time of ablation experiments on polynomial benchmarks. Table 14 records the thresholds and running times of PAF and PrAn on fractional benchmarks. PAF crashes on the

nonlin2 benchmarks. PrAn crashes on the doppler benchmarks, and does not support double exponential distribution.

Remark 5. *Further refinement of the algorithm can be achieved through the use of SMT solvers. Specifically, when analyzing each sub-region $B$, we may submit the logical query $\mathbf{x} \in B \Rightarrow K_u(\mathbf{x}) > 0$ to an SMT solver. If the solver returns* UNSAT*, this indicates that $K_u(\mathbf{x}) \leq 0$ holds within the sub-region $B$. In such cases, we can soundly conclude that the violation probability over $B$ is zero, and safely assign* $\text{flag}_{u,B} := 0$. □

Table 9. Experimental results on PAF and PrAn for uniform distribution. The thresholds are produced assuming computations are done in single precision and with a 99% target confidence level. A timeout of 1 hour is enforced for all benchmarks, except for ksin and kcos, where the timeout is extended to 4 hours.

| | Uniform Distribution | | | |
|---|---|---|---|---|
| | PAF | | PrAn | |
| Benchmark | Time (s) | Threshold | Time (s) | Threshold |
| bsplines0 | 1,232 | 5.71E-08 | 18.3 | 8.69E-08 |
| bsplines1 | 1,636 | 1.86E-07 | 27.91 | 1.89E-07 |
| bsplines2 | 2,748 | 1.94E-07 | 36.03 | 2.12E-07 |
| bsplines3 | 927 | 4.22E-08 | 15.32 | 5.71E-08 |
| classids0 | Timeout | 6.93E-06 | 106.36 | 8.65E-06 |
| classids1 | Timeout | 3.71E-06 | 145.9 | 4.69E-06 |
| classids2 | Timeout | 5.23E-06 | 120.77 | 7.58E-06 |
| filters1 | 608 | 1.25E-07 | 3.58 | 2.03E-07 |
| filters2 | 3,186 | 7.93E-07 | 65.21 | 1.01E-06 |
| filters3 | Timeout | 2.34E-06 | 212.15 | 2.86E-06 |
| filters4 | Timeout | 4.15E-06 | 429.3 | 5.20E-06 |
| rigidbody1 | Timeout | 1.74E-04 | 27.55 | 1.73E-04 |
| rigidbody2 | Timeout | 1.96E-02 | 46.5 | 9.70E-03 |
| sine | Timeout | 2.37E-07 | 132.95 | 2.41E-07 |
| solvecubic | Timeout | 1.78E-05 | 180.59 | 2.01E-05 |
| sqrt | Timeout | 1.54E-04 | 56.01 | 1.54E-04 |
| traincars1 | 3,434 | 1.76E-03 | 50.27 | 1.96E-03 |
| traincars2 | Timeout | 1.04E-03 | 205.4 | 1.33E-03 |
| traincars3 | Timeout | 1.75E-02 | 18.96 | 2.29E-02 |
| traincars4 | Timeout | 1.81E-01 | 26.36 | 2.30E-01 |
| trid1 | Timeout | 6.01E-03 | 69.03 | 6.12E-03 |
| trid2 | Timeout | 1.03E-02 | 59.48 | 1.17E-02 |
| trid3 | Timeout | 1.75E-02 | 312.22 | 1.95E-02 |
| trid4 | Timeout | 2.69E-02 | 1018.98 | 2.88E-02 |
| ksin | Timeout | N/A | 2049.07 | 6.93E-08 |
| kcos | Timeout | N/A | 1838.64 | 1.20E-07 |

Table 10. Experimental results on PAF and PrAn for truncated normal distribution. The thresholds are produced assuming computations are done in single precision and with a 99% target confidence level. A timeout of 1 hour is enforced for all benchmarks, except for `ksin` and `kcos`, where the timeout is extended to 4 hours.

| | Truncated standard normal distribution | | | |
|---|---|---|---|---|
| | PAF | | PrAn | |
| Benchmark | Time (s) | Threshold | Time (s) | Threshold |
| bsplines0 | 1,227 | 5.71E-08 | 28.99 | 8.69E-08 |
| bsplines1 | 1,707 | 1.86E-07 | 53.31 | 2.10E-07 |
| bsplines2 | 2,778 | 1.94E-07 | 62.68 | 2.12E-07 |
| bsplines3 | 914 | 4.22E-08 | 22.45 | 5.71E-08 |
| classids0 | Timeout | 4.45E-06 | 279.69 | 8.90E-06 |
| classids1 | Timeout | 2.68E-06 | 297.81 | 4.76E-06 |
| classids2 | Timeout | 3.85E-06 | 271.49 | 7.60E-06 |
| filters1 | 581 | 1.24E-07 | 5.52 | 2.03E-07 |
| filters2 | 3,220 | 6.13E-07 | 121.31 | 1.01E-06 |
| filters3 | Timeout | 2.05E-06 | 293.62 | 2.87E-06 |
| filters4 | Timeout | 4.15E-06 | 826.53 | 5.20E-06 |
| rigidbody1 | Timeout | 6.14E-06 | 45.43 | 1.73E-04 |
| rigidbody2 | Timeout | 5.99E-05 | 95.76 | 9.70E-03 |
| sine | Timeout | 2.37E-07 | 475.34 | 2.41E-07 |
| solvecubic | Timeout | 6.84E-06 | 230.44 | 2.02E-05 |
| sqrt | Timeout | 2.46E-07 | 115.91 | 1.54E-04 |
| traincars1 | 3,007 | 8.26E-04 | 93.36 | 1.96E-03 |
| traincars2 | Timeout | 3.62E-04 | 476.4 | 1.33E-03 |
| traincars3 | Timeout | 9.56E-03 | 19.18 | 2.23E-02 |
| traincars4 | Timeout | 8.87E-02 | 26.86 | 2.30E-01 |
| trid1 | Timeout | 1.58E-05 | 253.98 | 6.06E-03 |
| trid2 | Timeout | 2.42E-05 | 140.63 | 1.17E-02 |
| trid3 | Timeout | 6.80E-05 | 537.01 | 1.95E-02 |
| trid4 | Timeout | 2.64E-04 | 1576.52 | 3.03E-02 |
| ksin | Timeout | N/A | Timeout | N/A |
| kcos | Timeout | N/A | Timeout | N/A |

Table 11. Experimental results on PAF for truncated double exponential distribution. The thresholds are produced assuming computations are done in single precision and with a 99% target confidence level. A timeout of 1 hour is enforced for all benchmarks, except for `ksin` and `kcos`, where the timeout is extended to 4 hours. For benchmarks `classids0`, `classids1`, `classids2`, and `solvecubic`, the variance value is set to 1. For all other benchmarks, the variance value is set to 0.01.

| | Truncated double exponential distribution | |
|---|---|---|
| | PAF | |
| Benchmark | Time (s) | Threshold |
| bsplines0 | 2,629 | 5.71E-08 |
| bsplines1 | 2,201 | 6.95E-08 |
| bsplines2 | Timeout | 2.11E-08 |
| bsplines3 | 938 | 7.62E-12 |
| classids0 | 24,606 | 5.00E-06 |
| classids1 | 16,145 | 3.08E-06 |
| classids2 | 13,698 | 4.15E-06 |
| filters1 | 733 | 5.43E-09 |
| filters2 | Timeout | 2.90E-08 |
| filters3 | Timeout | 1.09E-07 |
| filters4 | Timeout | 4.61E-07 |
| rigidbody1 | Timeout | 4.80E-07 |
| rigidbody2 | Timeout | 9.55E-07 |
| sine | Timeout | 1.49E-08 |
| solvecubic | 16,795 | 1.42E-05 |
| sqrt | Timeout | 2.46E-07 |
| traincars1 | Timeout | 4.50E-04 |
| traincars2 | Timeout | 2.83E-05 |
| traincars3 | Timeout | 8.95E-04 |
| traincars4 | Timeout | 7.33E-03 |
| trid1 | Timeout | 1.58E-05 |
| trid2 | Timeout | 2.43E-05 |
| trid3 | Timeout | 6.77E-05 |
| trid4 | Timeout | 2.64E-04 |
| ksin | Timeout | N/A |
| kcos | Timeout | N/A |

Table 12. Ablation experiments on polynomial benchmarks. Columns labeled "NM" report thresholds generated by the Naive Markov method, while columns labeled "CMB" report thresholds generated by the Central-Moment-Based algorithm with range partition refinement. The experiments use an analysis order of $n = 4$. For the Central-Moment-Based algorithm, the number of partitions is $b = 8$ for benchmarks with fewer than four input variables and fewer than ten operations, and range partitioning is disabled otherwise.

| | Uniform | | Normal | | Laplace | |
|---|---|---|---|---|---|---|
| Benchmark | NM | CMB | NM | CMB | NM | CMB |
| bsplines0 | 9.44E-07 | 4.96E-07 | 8.95E-07 | 4.98E-07 | 8.57E-07 | 4.81E-07 |
| bsplines1 | 1.03E-06 | 5.72E-07 | 9.66E-07 | 2.33E-07 | 9.23E-07 | 5.63E-07 |
| bsplines2 | 1.01E-06 | 5.76E-07 | 9.49E-07 | 2.15E-07 | 9.06E-07 | 5.76E-07 |
| bsplines3 | 6.62E-08 | 4.13E-08 | 6.18E-08 | 8.73E-09 | 5.89E-08 | 4.14E-08 |
| classids0 | 2.41E-05 | 1.46E-05 | 9.35E-06 | 4.79E-06 | 1.16E-05 | 7.10E-06 |
| classids1 | 1.33E-05 | 8.29E-06 | 5.12E-06 | 2.81E-06 | 6.42E-06 | 4.04E-06 |
| classids2 | 2.13E-05 | 1.23E-05 | 9.18E-06 | 4.71E-06 | 1.13E-05 | 6.58E-06 |
| filters1 | 1.76E-07 | 9.07E-08 | 1.44E-07 | 3.12E-08 | 1.45E-07 | 8.61E-08 |
| filters2 | 1.61E-06 | 8.22E-07 | 1.27E-06 | 3.18E-07 | 1.26E-06 | 7.82E-07 |
| filters3 | 5.09E-06 | 2.63E-06 | 3.92E-06 | 1.06E-06 | 3.86E-06 | 2.30E-06 |
| filters4 | 1.11E-05 | 7.23E-06 | 8.48E-06 | 5.89E-06 | 8.32E-06 | 5.91E-06 |
| rigidbody1 | 2.63E-04 | 1.81E-04 | 4.56E-06 | 7.40E-07 | 1.13E-05 | 1.03E-05 |
| rigidbody2 | 1.78E-02 | 1.67E-02 | 2.35E-05 | 2.29E-05 | 1.24E-04 | 1.24E-04 |
| sine | 1.01E-06 | 6.02E-07 | 8.62E-07 | 1.61E-07 | 9.33E-07 | 9.33E-07 |
| solvecubic | 3.72E-05 | 1.98E-05 | 9.70E-06 | 3.14E-06 | 1.97E-05 | 1.32E-05 |
| sqrt | 2.22E-04 | 1.50E-04 | 2.98E-06 | 4.78E-07 | 3.98E-05 | 9.76E-06 |
| traincars1 | 5.18E-03 | 2.48E-03 | 1.72E-03 | 5.27E-04 | 2.10E-03 | 1.23E-03 |
| traincars2 | 3.56E-03 | 2.31E-03 | 6.29E-04 | 4.51E-04 | 9.09E-04 | 7.42E-04 |
| traincars3 | 4.43E-02 | 2.87E-02 | 8.99E-03 | 6.28E-03 | 1.22E-02 | 9.35E-03 |
| traincars4 | 4.94E-01 | 3.01E-01 | 9.62E-02 | 5.95E-02 | 1.16E-01 | 8.07E-02 |
| trid1 | 1.08E-02 | 6.96E-03 | 8.99E-06 | 1.99E-06 | 2.21E-05 | 2.86E-06 |
| trid2 | 2.04E-02 | 1.56E-02 | 1.56E-05 | 1.25E-05 | 3.54E-05 | 3.47E-05 |
| trid3 | 3.23E-02 | 2.44E-02 | 2.37E-05 | 1.83E-05 | 5.08E-05 | 4.81E-05 |
| trid4 | 4.46E-02 | 3.46E-02 | 3.34E-05 | 2.50E-05 | 6.82E-05 | 6.35E-05 |
| ksin | 1.46E-07 | 1.13E-07 | 1.49E-07 | 1.40E-07 | 5.02E-07 | 3.32E-07 |
| kcos | 4.76E-07 | 2.37E-07 | 4.69E-07 | 2.42E-07 | 1.42E-06 | 2.30E-07 |

Table 13. Running time of ablation experiments on polynomial benchmarks. Both the NM and CMB algorithm use an analysis order of $n = 4$. For the CMB algorithm, the number of partitions if $b = 8$ for benchmarks with fewer than four input variables and fewer than ten operations.

| | Uniform | | Normal | | Laplace | |
|---|---|---|---|---|---|---|
| Benchmark | NM time (s) | CMB time(s) | NM time (s) | CMB time (s) | NM time (s) | CMB time (s) |
| bsplines0 | 0.59 | 0.67 | 0.67 | 0.67 | 0.62 | 0.68 |
| bsplines1 | 0.54 | 0.61 | 0.56 | 0.63 | 0.55 | 0.63 |
| bsplines2 | 1.21 | 1.32 | 1.21 | 1.29 | 1.11 | 1.28 |
| bsplines3 | 0.25 | 0.3 | 0.25 | 0.27 | 0.25 | 0.28 |
| classids0 | 1.23 | 1.3 | 1.21 | 1.34 | 1.23 | 1.21 |
| classids1 | 1.13 | 1.25 | 1.12 | 1.27 | 1.23 | 1.3 |
| classids2 | 1.24 | 1.36 | 1.24 | 1.36 | 1.25 | 1.34 |
| filters1 | 0.18 | 0.23 | 0.17 | 0.23 | 0.17 | 0.22 |
| filters2 | 0.24 | 2.64 | 0.23 | 1.86 | 0.23 | 2.07 |
| filters3 | 1.08 | 48.62 | 1.07 | 41.58 | 1.06 | 75.33 |
| filters4 | 6.06 | 6.05 | 5.51 | 5.94 | 5.81 | 5.91 |
| rigidbody1 | 0.42 | 234.49 | 0.42 | 235.61 | 0.43 | 234.79 |
| rigidbody2 | 1.28 | 2.97 | 1.23 | 2.45 | 1.32 | 1.41 |
| sine | 2.35 | 3.03 | 2.26 | 3.06 | 2.28 | 2.39 |
| solvecubic | 1.03 | 27.28 | 1.01 | 26.74 | 0.99 | 26.97 |
| sqrt | 0.79 | 0.87 | 0.76 | 0.9 | 0.75 | 0.93 |
| traincars1 | 0.39 | 10.16 | 0.38 | 8.73 | 0.39 | 8.56 |
| traincars2 | 0.94 | 1.09 | 0.9 | 1.22 | 0.96 | 60.05 |
| traincars3 | 3.38 | 4.89 | 3.31 | 5.05 | 3.23 | 5.68 |
| traincars4 | 6.43 | 10.34 | 6.06 | 10.98 | 6.38 | 10.79 |
| trid1 | 0.49 | 25.22 | 0.47 | 22.04 | 0.46 | 21.82 |
| trid2 | 1.5 | 4.67 | 1.46 | 3.43 | 1.46 | 3.77 |
| trid3 | 4.28 | 13.02 | 4.16 | 16.24 | 4.15 | 15.68 |
| trid4 | 10.13 | 33.34 | 10.25 | 34.51 | 9.94 | 36.35 |
| ksin | 7.11 | 7.35 | 7.27 | 7.32 | 7.32 | 7.17 |
| kcos | 5.49 | 5.9 | 5.59 | 5.63 | 5.62 | 5.74 |

Table 14. Experimental results on PAF and PrAn on fractional benchmarks. The settings are the same as in Table 1. Entries marked "N/A" indicate that the corresponding tool either crashed or lacked support for the corresponding distribution (PrAn laplace).

| | | PAF | | PrAn | |
|---|---|---|---|---|---|
| Benchmark | Distribution | Time (s) | Threshold | Time (s) | Threshold |
| doppler1 | uniform | Timeout | 7.95E-05 | N/A | N/A |
| doppler2 | uniform | Timeout | 1.43E-04 | N/A | N/A |
| doppler3 | uniform | Timeout | 4.55E-05 | N/A | N/A |
| nonlin1 | uniform | 840.21 | 6.71E-08 | 23.32 | 7.54E-08 |
| nonlin2 | uniform | N/A | N/A | 225.24 | 3.66E-06 |
| predator | uniform | 972.16 | 9.95E-08 | 84.93 | 1.00E-07 |
| verhulst | uniform | 350.34 | 1.72E-07 | 30.03 | 1.80E-07 |
| nonlin1 | normal | 842.05 | 6.71E-08 | 15.92 | 7.54E-08 |
| nonlin2 | normal | N/A | N/A | 681.51 | 3.08E-06 |
| predator | normal | 965.61 | 9.95E-08 | 173.84 | 1.00E-07 |
| verhulst | normal | 353.44 | 1.72E-07 | 48.75 | 1.80E-07 |
| nonlin1 | laplace | 841.33 | 6.71E-08 | N/A | N/A |
| nonlin2 | laplace | N/A | N/A | N/A | N/A |
| predator | laplace | 957.49 | 9.89E-08 | N/A | N/A |
| verhulst | laplace | 346.78 | 1.69E-07 | N/A | N/A |