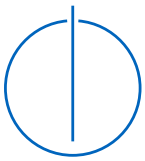
DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

On Automized Error Analysis for Numerical Solvers of Differential Equations

Torben Soennecken



DEPARTMENT OF INFORMATICS

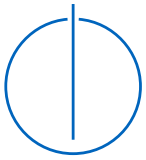
TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

On Automized Error Analysis for Numerical Solvers of Differential Equations

Zur automatisierten Fehleranalyse für numerische Löser von Differentialgleichungen

Author:	Torben Soennecken
Supervisor:	Prof. Dr. rer. nat. Martin Schulz
Advisor:	Prof. Dr. rer. nat. Martin Schreiber
Submission Date:	15.09.2021



I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.09.2021 Torben Soennecken

Acknowledgments

I want to express my gratitude to Prof. Dr. rer. nat. Martin Schreiber for supporting me throughout my and encouraging me on a journey through numerical mathematics. I would not have pursued this path and learned such invaluable lessons without his doing.

Furthermore, I want to thank Prof. Dr. rer. nat. Martin Schulz for allowing me to write my bachelor's thesis at the chair of Computer Architecture and Parallel Systems and supervising such interesting and modern topics.

Finally, I am grateful to my family and friends for their ongoing love and support.

Abstract

Ordinary differential equations are an essential building block of our modern-day life, but rarely inherit an analytic solution. The computers tasked with calculating estimations of the solution introduce additional errors with approximate representations of the infinite and uncountable set of real numbers. To provide guarantees for the correctness of computational calculations, it is required to understand and rigorously control these errors. Existing research determined combinations of less and more precise calculations to yield *accurate enough* results while significantly boosting the computational performance. Combining this idea with rigorousness, this thesis is going to define different analysis methods that calculate rigorous symbolic bounds for the errors in terms of the variable precision of calculation. This focus on symbolic bounds and understanding the error of arbitrary precision within calculation contrasts currently available analysis tools. We extend existing methods to allow for the necessary support of symbolic expressions and analysis over a range of possible computational precision. We also provide a proof-of-concept implementation for the analysis methods with the Python project vodes. This thesis demonstrates the achieved bounding accuracy to be close to the currently most prominent and accurate error analysis tools.

Contents

1. Introduction

An important foundation for our modern-day life are different kinds of simulations of our environment, giving us valuable information in a wide-ranging field of applications including climate simulations, population development, disease spread, and many more. The essential building blocks for these simulations are differential equations since they allow for the modeling of change relative to space and time. As important as differential equations are, as difficult are they to solve, often missing an analytic solution. The computers tasked with approximating the solutions, possibly in many steps of evaluation, work within a discrete and bound space and are therefore only able to represent a finite set of numbers exactly. This introduces a further approximation, as the numbers during the solving of the differential equation may not be able to be exactly represented by the computer. When the amount of exactly representable numbers is increased, a potential gain in accuracy is achieved based on higher computational cost. Mixed precision algorithms reduce the amount of exactly representable numbers, boosting the overall performance of calculations [1] while still obtaining *accurate enough* results. However, presented applications are often limited to problem statements of iterative refinement. An initial, costly approximation is provided with lower and refined using higher precision [2, 3, 1]. These refinement methods were validated empirically, challenging the value of the obtained solutions. As Thomas Carlyle once said :

"*The greatest of faults, I should say, is to be conscious of none.*" [4]

Further methods for approximating errors based on modeling higher precision calculations as exact solutions are readily available [5]. However, imagine a is close to but greater than 0 and neither exactly representable in lower nor higher precision. The solution obtained for $a \cdot b$ is going to wrongfully be 0, allowing for astronomically high errors. Therefore, it is necessary to understand, formalize and rigorously control the error arising from using certain precision in computations. Combining rigorous error control with the ability to alter the exactly representable numbers would allow for increased performance, which may be especially valuable in cases of e.g. long-running climate simulations, while still assuring the value of the obtained solutions. In this thesis, the idea of varying the computational precision is combined with the analysis tools presented in earlier research [1, 6, 7, 8, 9, 10] to obtain an automatic symbolic analysis framework, that exposes the precision of the computation as a free variable. This can be used to determine the necessary precision to obtain a guaranteed accuracy.

2. Ordinary Differential Equations

Differential equations relate an unknown function to its derivatives. In this thesis, ordinary differential equations (ODEs) are discussed, which are limited to one independent variable e.g. $t \in \mathbb{R}$, instead of multiple independent variables, as given within partial differential equations (PDEs). ODEs of order n can be written as

$$\frac{d^n}{dt^n}y(t) = f\left(t, y(t), \ldots, \frac{d^{n-1}}{dt^{n-1}}y(t)\right) \tag{2.1}$$

with f being a known function. We are using the abbreviated form

$$\frac{d^n y}{dt^n} = f\left(t, y, \ldots, \frac{d^{n-1}y}{dt^{n-1}}\right) \tag{2.2}$$

for the representation of ODEs of order n. An ODE in itself does not usually specify an unique solution and additional constraints have to be provided. In this thesis, constraints on the initial condition are given, resulting in an initial value problem.

$$\frac{d^n y}{dt^n} = f\left(t, y, \ldots, \frac{d^{n-1}y}{dt^{n-1}}\right) \qquad y(t_0) = y_0 \tag{2.3}$$

The order of an ODE can be reduced to first-order by transforming the equation into a system of first-order ODEs [11]. We start by defining two vectors $\overrightarrow{Y}$, $\overrightarrow{F}$.

$$\begin{aligned} \overrightarrow{Y} &:= \left(y, \frac{dy}{dt}, \ldots, \frac{d^{n-1}y}{dt^{n-1}}\right) \\ &:= (y_1, \ldots, y_n) \\ \overrightarrow{F} &:= (f_1, \ldots, f_n) \\ \forall_{i<n}\ f_i\left(t, \overrightarrow{Y}\right) &:= y_{i+1} \\ f_n\left(t, \overrightarrow{Y}\right) &:= f\left((t, y_1, \ldots, y_n)\right) \end{aligned} \tag{2.4}$$

Using these two vectors, we can then proceed to formulate a first-order ODE.

$$\frac{d\overrightarrow{Y}}{dt} = \overrightarrow{F}\left(t, \overrightarrow{Y}\right) = \begin{pmatrix} y_2 \\ y_3 \\ \ldots \\ f(t, y_1, \ldots, y_n) \end{pmatrix} \tag{2.5}$$

Example. Given the oscillating pendulum $\frac{d^2y}{dt^2} = -y$, we obtain the first-order ODE from Equation (2.6).

$$\frac{d\overrightarrow{Y}}{dt} = \begin{pmatrix} y_2 \\ -y_2 \end{pmatrix} \tag{2.6}$$

Given a differential equation of the form $\frac{dy}{dt} = f(t)g(y)$ one may be able to obtain an analytic solution using the separation of variables. [11]

$$\int \frac{dy}{g(y)} = \int f(t)\ dt\ + C \tag{2.7}$$

While mathematicians have found further analytic solutions for special cases of differential equations (e.g. linear equations with constant coefficients), retrieving analytic solutions for more complex differential equations is seldom possible [11]. Based on this, numerical methods play an important role in the solving of differential equations. In this thesis, we are using the Runge-Kutta family of integration methods. Given an initial value problem for an ODE of first-order, let

$$\tilde{y}_{i+1} \approx y\left(t_{i+1}\right) \tag{2.8}$$

denote the numerical approximation at the time t_{i+1}. To obtain this approximation, the numerical method determines an approximation for the slope of y at t_{i+1} using the known function f and intermediate points (stages) k_j in the neighborhood of the local solution $t_i + c_j \cdot h \in [t_i, t_{i+1}]$. These methods work iteratively and directly benefit from a reduced distance between the steps of evaluation t_i as well as multiple stages k_j, if selected appropriately. Let h denote the equidistance between all time steps. Runge-Kutta methods obey the following general formula, with $a_{j,l}$ and b_j being real coefficients and s being the stages of the integration method. [11, 12]

$$\begin{aligned} \tilde{y}_{i+1} &= y_i + h\left(b_1 k_1 + \cdots + b_s k_s\right) \\ k_j &= f\left(t_i + c_j h, y_i + h\sum_{l=1}^{s} a_{j,l} k_l\right) \\ k_3 &= f\left(t_i + c_3 h, y_i + h\left(a_{3,1} k_1 + a_{3,2} k_2\right)\right) \end{aligned} \tag{2.9}$$

Using the tabular notation introduced by Butcher for Runge-Kutta methods, as shown in Table 2.1, the Euler (RK-1), the Heun (RK-2), and the classic Runge-Kutta 4 integration methods are formalized in Table 2.2 with their respective real coefficients and stages. The numbers associated with the integration methods denote their order p. An integration method is defined to be of an order p, given that $\mathcal{O}(h^p)$ holds for the global error and therefore, the difference between the actual and approximated solution.

c_1	$a_{1,1}$	$a_{1,2}$	$\dots$	$a_{1,s}$
c_2	$a_{2,1}$	$a_{2,2}$	$\dots$	$a_{2,s}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
c_s	$a_{s,1}$	$a_{s,2}$	$\dots$	$a_{s,s}$
	b_1	b_2	$\dots$	b_s

(a) Full

c	$\mathcal{A}$
	b^T

(b) Compact

Table 2.1.: Tabular notation introduced by Butcher for defining Runge-Kutta methods in a compact and precise manner [11, 12]

0	
	1

(a) RK-1

0		
1	1	
	$\frac{1}{2}$	$\frac{1}{2}$

(b) RK-2

0				
$\frac{1}{2}$	$\frac{1}{2}$			
$\frac{1}{2}$	0	$\frac{1}{2}$		
1	0	0	1	
	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

(c) RK-4

Table 2.2.: Butcher tableaus for the basic Runge-Kutta methods

3. Related Work

The set of real numbers $\mathbb{R}$ is both unbound and continuous, meaning that between two distinct real numbers $r_1 < r_2$ it is always possible to find another real number $r_1 < r_3 < r_2$. This set is uncountable and therefore from even greater magnitude than the set of natural numbers $\mathbb{N}$. Faced with the challenge of representing these numbers within a discrete space, an approximation for each real number $r \in \mathbb{R}$ is given by a machine number $m \in \mathbb{M}$ obtained from the rounding function $rd : \mathbb{R} \to \mathbb{M}$. [13]

Definition 1. The absolute rounding error δ_r is defined as the difference between the exact number $r \in \mathbb{R}$ and its approximate representation $m \in \mathbb{R}$ obtained from applying the rounding function. [13]

$$\delta_r := r - rd(r) \tag{3.1}$$

Let m_1 and m_2 be two machine numbers for which the result of applying basic arithmetic operations $\circ \in \{+, -, \cdot, /\}$ is to be represented. Because the set of machine numbers $\mathbb{M}$ is not closed on $\circ$, the result may be a real number that is not exactly representable $r \in \mathbb{R} \setminus \mathbb{M}$. This introduces an additional error with the approximate representation of the result. [13]

$$\delta_r = (m_1 \circ m_2) - rd(m_1 \circ m_2) \tag{3.2}$$

Under the assumption of control over the set of machine numbers, these errors can be influenced by adjusting the machine numbers to include r and not introduce an additional rounding error.

$$r \in \mathbb{M} \Rightarrow \delta_r = r - rd(r) = r - r = 0 \tag{3.3}$$

3.1. Floating-Point Numbers

A floating-point number $f \in \mathbb{F}$ is defined to be composed of a mantissa M multiplied by a basis B to the power of a given exponent E.

$$f := M \cdot B^E \tag{3.4}$$

Using the Equation (3.5), we obtain the definition for machine numbers used within this thesis. Every machine number $m \in \mathbb{M}$ is composed of a sign bit s, a p-Bit mantissa r and a n-Bit exponent e. Importantly, these machine numbers are normalized. The mantissa r always starts with a 1 in the first digit. [13]

$$\begin{aligned} m &:= (-1)^s \cdot \sum_{i=0}^{p-1} r_i \cdot 2^{-i} \cdot 2^e \\ e &:= \sum_{i=0}^{n-1} e_i \cdot 2^i \\ e_i &\in \{0,1\} \\ r_0 &= 1 \\ r_i &\in \{0,1\} \\ s &\in \{0,1\} \end{aligned} \tag{3.5}$$

Definition 2. The relative rounding error ϵ_r defines errors in terms of relative deviation. For a real number $r \neq 0$ it is defined as the absolute rounding error δ_r divided by r. [13]

$$\epsilon_r := \frac{\delta_r}{r},\ r \neq 0 \tag{3.6}$$

A common standard for floating-point numbers is given in IEEE 754 [14], further adding the constraint that a biased representation is used for the exponent. This means, that if 127 is stored, the actual value of the exponent e for the floating-point number f is obtained by subtracting the bias 2^{n-1} from the exponent [15]. The binary format for the IEEE 754 floating-point numbers is given in Equation (3.7).

$$f = s_0 | e_{0\ldots n-1} | r_{1\ldots p-1} \tag{3.7}$$

Name	p	n	e_{min}	e_{max}
Half Precision	11	5	$(2^0 - 2^{n-1}) + 1 = -14$	$(2^n - 2^{n-1}) - 1 = 15$
Single Precision	24	8	-126	127
Double Precision	53	11	-1022	1023
Quadruple Precision	113	15	-16382	16833

Table 3.1.: Basic floating-point formats

Example. Given the number -13.75, we store it in the binary format using single precision.

$$1|10000010|10111000000000000000000 = -1 \cdot 2^3 \cdot 1.10111$$

IEEE 754 defines special quantities that are expressed using a specific combination of exponent and mantissa bits. These deviate in their meaning from the value that would be obtained from Equation (3.5) [15]. The quantities used within this thesis are listed in Table 3.2.

Name	s	r	e
$\overset{+}{-}\infty$	$\overset{+}{-}$	$1+0$	$e_{max}+1$
NaN	$\overset{+}{-}$	$1+c,\ c \neq 0$	$e_{min}-1$
0	$\overset{+}{-}$	$1+0$	$e_{max}+1$

Table 3.2.: IEEE 754 Specific Quantities

Definition 3. We define the boundaries of floating point representation in terms of the biggest positive and smallest normalized positive representable machine numbers λ and σ.

$$\lambda := \left(2 - 2^{-(p-1)}\right) \cdot 2^{e_{max}} \tag{3.8}$$

$$\sigma := 2^{e_{min}} \tag{3.9}$$

Any real number $r \in \mathbb{R}$ that is not included in the set $[-\lambda, -\sigma] \cup [\sigma, \lambda]$ defines either an overflow $|r| > \lambda$ or underflow $|r| < \sigma$ and results in special treatment within the rounding function.

Remark. It is important to state, that IEEE 754 also includes the notion of denormalized (subnormal) numbers. These are numbers that use the exponent $e_{min} - 1$ without the normalization $r_0 = 1$. They can therefore be used to allow for a more gradual transition to an underflow. [15]

Definition 4. Following, rounding is defined to map any real number $r \in \mathbb{R}$ to its closest machine number representation left m_l or right m_r to its actual value. The ambiguous case of equal distance to m_l and m_r is left open, as it has no impact on the rounding error analysis presented in this thesis. This rounding mode is referred to as round-to-nearest. [16]

$$rd(r) := \begin{cases} 0 & |r| < \sigma \\ \text{sign}(r) \cdot \infty & |r| > \lambda \\ m_l & r \leq \frac{m_l + m_r}{2} \\ m_r & r \geq \frac{m_l + m_r}{2} \end{cases} \tag{3.10}$$

Remark. It is to be noted, that further rounding modes like rounding to the closest machine number left m_l (round-to-$-\infty$) or right m_r (round-to-$+\infty$) of the actual value r exist and would alter the rounding errors. However, as a majority of programs use the default rounding mode round-to-nearest, this thesis assumes its usage. [16]

Theorem 5. *Assuming the absence of over- and underflow, the maximum absolute rounding error $|\delta_r|$ is bound by 2^{e-p} when using round-to-nearest.*

Proof. We go through a detailed proof, as this is a central assumption of this thesis and the automatic analysis framework. Let $r \in \mathbb{R}$ be any real number with $rd(r)$ not over- or underflowing. The maximum absolute rounding error is given, when the number lies exactly between two neighboring machine numbers $r = \frac{m_l+m_r}{2}$. Without loss of generality, we assume $rd(r) = m_l > 0$.

$$\begin{aligned} |\delta_r| &= |r - rd(r)| \\ &\leq \left| \frac{m_l + m_r}{2} - m_l \right| \\ &= \frac{m_r - m_l}{2} \end{aligned} \tag{3.11}$$

We now divide the proof in two cases that may occur for the neighboring m_l and m_r. In the first case, m_r is the first floating point number with the higher exponent $e+1$ and m_l the last floating-point number with the lower exponent e. We can therefore provide the structure given in Equation (3.12) for these machine numbers.

$$\begin{aligned} m_l &= \left(\sum_{i=0}^{p-1} 2^{-i} \right) \cdot 2^e \\ m_r &= 1 \cdot 2^{e+1} \end{aligned} \tag{3.12}$$

Substituting these values within the obtained bound for δ_r yields a series that can be simplified using the closed-form formula of the geometric series and finally compacted into the inequality $|\delta_r| \leq 2^{e-p}$.

$$\begin{aligned} |\delta_r| &\leq \left(2^{e+1} - \left(\sum_{i=0}^{p-1} 2^{-i} \right) \cdot 2^e \right) \cdot \frac{1}{2} \\ &= 2^e \cdot \left(1 - \sum_{i=1}^{p-1} 2^{-i} \right) \cdot \frac{1}{2} \\ &= 2^e \cdot \left(1 - \left(\frac{1 - 2^{-(p+1)}}{1 - 2^{-1}} - 2^{-p} - 1 \right) \right) \cdot \frac{1}{2} \\ &= 2^e \cdot \left(1 - \left(1 - 2^{-(p-1)} \right) \right) \cdot \frac{1}{2} \\ &= 2^e \cdot 2^{-p} \end{aligned} \tag{3.13}$$

In the second case, m_r and m_l have the same exponent. Because they neighbor each other, the difference between the two mantissa r_r and r_l is $2^{-(p-1)}$.

$$\begin{aligned} |\delta_r| &\leq 2^e \cdot (r_r - r_l) \cdot \frac{1}{2} \\ &= 2^e \cdot 2^{-p} \end{aligned} \tag{3.14}$$

□

Corollary 6. *The maximum relative rounding error ϵ_r for a real number $r \in \mathbb{R}$ with $rd(r)$ not over- or underflowing and r being unequal to 0 is bound by 2^{-p}.*

Proof. Using Definition 2 and Theorem 5, we obtain the chain of inequalities given in Equation (3.15).

$$\begin{aligned} |\epsilon_r| &= \left| \frac{\delta_r}{r} \right| \\ &\leq \left| \frac{2^{e-p}}{r} \right| \\ &\leq \left| \frac{2^{e-p}}{1 \cdot 2^e} \right| \\ &= 2^{-p} \end{aligned} \tag{3.15}$$

□

Definition 7. We summarize this section by defining the machine accuracy $\bar{\epsilon}$ as the maximum relative rounding error for a real number $r \in \mathbb{R}$ with $rd(r)$ not over- or underflowing and r being unequal to 0.

$$\bar{\epsilon} := 2^{-p} \tag{3.16}$$

3.2. Range Analysis

After Section 3.1 introduced the essential error definitions, we now take a look at the available methods of bounding them a posteriori. A well known tool for bounding the error of operations within finite number representation is the usage of interval analysis.

Definition 8. The interval $X := \left[\underline{X}, \overline{X}\right]$ is the set of real numbers contained within the lower and upper boundaries $\underline{X}, \overline{X}$. An interval X is degenerate if $\underline{X} = \overline{X}$, leading to the notation $x = [x, x]$. [17]

$$X := \left[\underline{X}, \overline{X}\right] = \left\{x \in \mathbb{R} : \underline{X} \leq x \leq \overline{X}\right\} \tag{3.17}$$

The general idea of interval analysis is to preserve the inclusion of the exact result. As this exact result can only be obtained from exact calculations and exact inputs, that are seldom possible within the finite set of machine numbers, the interval operations are defined to guarantee inclusion even when all calculations are done with finite precision [18]. A real number $r \in \mathbb{R}$ is represented as an interval of floating-point numbers $m_l, m_r \in \mathbb{M}$.

$$r \mapsto [m_l, m_r] \tag{3.18}$$

Interval analysis requires to define interval extensions for a real-valued function to be supported. As the goal is to ensure the inclusion of the exact result and with the knowledge of an interval fundamentally being a set, these extensions therefore have to include the set image.

$$f(X) = \{f(x) : x \in X\} \tag{3.19}$$

All basic arithmetic operations $(+, -, \cdot,)$ follow the same general form given in Equation (3.20).

$$X \odot Y = \{x \odot y : x \in X, y \in Y\} \tag{3.20}$$

Importantly, the rounding for the extended functions has to be done outwards to preserve the rigorous nature of interval analysis [17]. For the boundaries of the resulting interval, the round-to-$-\infty$ ($rd_\downarrow$) and round-to-$+\infty$ ($rd_\uparrow$) methods are used.

$$\begin{aligned} X + Y &= \left[rd_\downarrow(\underline{X} + \underline{Y}), rd_\uparrow(\overline{X} + \overline{Y})\right] \\ X - Y &= \left[rd_\downarrow(\underline{X} - \overline{Y}), rd_\uparrow(\overline{X} - \underline{Y})\right] \\ X \cdot S &= \left[rd_\downarrow(\min(S)), rd_\uparrow(\max(S))\right], \; S = \left\{\underline{X}\underline{Y}, \underline{X}\overline{Y}, \overline{X}\underline{Y}, \overline{X}\overline{Y}\right\} \\ \frac{X}{Y} &= X \cdot \frac{1}{Y} \\ &= X \cdot \left[rd_\downarrow(1/\overline{Y}), rd_\uparrow(1/\underline{Y})\right], \; 0 \notin Y \end{aligned} \tag{3.21}$$

For the application of interval analysis in the given context, multiple issues arise. A central problem is the phenomenon of interval dependency [17, 19].

$$f(X) = \{x^2 : x \in X\} = \begin{cases} \left[\underline{X}^2, \overline{X}^2\right] & 0 \leq \underline{X} \leq \overline{X} \\ \left[\overline{X}^2, \underline{X}^2\right] & \underline{X} \leq \overline{X} \leq 0 \\ \left[0, \max\{\underline{X}^2, \overline{X}^2\}\right] & \underline{X} \leq 0 \leq \overline{X} \end{cases} \tag{3.22}$$

With Equation (3.22) defining the extension of the power-2 function $f(x) = x^2$, an earlier equivalent product function $f_2(x) = x \cdot x$ would result in an in-equivalence and overestimation within interval analysis. [17]

$$[-10, 10]^2 = [0, 100] \neq [-100, 100] = [-10, 10] \cdot [-10, 10] \tag{3.23}$$

As interval analysis has no possibility to track correlated variables, the over approximation can accumulate along the operations and may result in an *"exponential range explosion"*[20]. Furthermore, interval analysis was initially created to allow rigorous computations within finite precision, with wide output intervals indicating the necessity of higher precision computations. This contrasts the requirement of exposing the precision as a free variable and a priori round-off error analysis. To overcome the interval dependency phenomenon, an extension in the form of affine analysis was created.

Definition 9. An affine expression is composed of a central value x_0 with independent sources of uncertainty (noise symbols) $n_i \in [-1, +1]$. [19]

$$\hat{x} = x_0 + \sum_{i=1}^{n} x_i n_i \tag{3.24}$$

Any affine expression $\hat{x}$ can be converted to an interval by applying Equation (3.25).

$$x \in \left[x_0 - \sum_{i=1}^{n} |x_i| \,,\; x_0 + \sum_{i=1}^{n} |x_i| \right]. \tag{3.25}$$

The correlation now present is important for the tightness of the obtained bound. This can be seen in the simple example of calculating $a - a$, as it is shown in Equation (3.26).

$$\begin{aligned} rd(a-a) &\overset{AA}{=} ((1 + \bar{\epsilon} n_a) \cdot a - (1 + \bar{\epsilon} n_a) \cdot a) \cdot (1 + \bar{\epsilon} n_{a-a}) \\ &\overset{AA}{=} 0 \\ &\overset{IA}{=} ([a\bar{\epsilon} - \bar{\epsilon}, a\bar{\epsilon} + \bar{\epsilon}] - [a\bar{\epsilon} - \bar{\epsilon}, a\bar{\epsilon} + \bar{\epsilon}]) \cdot [1 - \bar{\epsilon}, 1 + \bar{\epsilon}] \\ &\overset{IA}{=} [-2\bar{\epsilon}, 2\bar{\epsilon}] \cdot [1 - \bar{\epsilon}, 1 + \bar{\epsilon}] \\ &\overset{IA}{\neq} 0 \end{aligned} \tag{3.26}$$

Similar to interval analysis, affine analysis is meant to be used a posteriori to guarantee the inclusion of the exact result. It is also necessary to define (rigorous) approximations for non-affine operations. This includes the multiplication of two affine expressions, as the resulting quadratic terms are not allowed within a valid affine expression.

$$(a_0 + a_1 n_a) \cdot (b_0 + b_1 n_b) = a_0 b_0 + a_0 b_1 n_b + a_1 b_0 n_a + a_1 b_1 n_a n_b \tag{3.27}$$

Let $\hat{x}, \hat{y}$ denote two affine expressions. The multiplication $\hat{x} \cdot \hat{y}$ can be bound to an affine expression by adding δn_k, with n_k being a unique noise symbol, and δ being a bound for the remainder. [19, 21]

$$\hat{x} \cdot \hat{y} = x_0 y_0 + \sum_{i>0} (x_0 y_i + x_i y_0) \cdot n_i + \left(\sum_{i>0} |x_i| \cdot \sum_{i>0} |y_i| \right) \cdot n_k \tag{3.28}$$

A posteriori analysis based on affine analysis is available [22] by expressing any quantity u in an affine expression in terms of its input variations v_i and errors w_i introduced by input quantization, rounding and affine approximation.

$$\begin{aligned} \hat{u} &= u_0 + \sum v_i n_i + \sum w_i \delta_i \\ n_i, \delta_i &\in [-1, 1] \end{aligned} \tag{3.29}$$

Steps towards static analysis are given in existing literature [20] without obtaining rigorousness, as the entailing quadratic terms are dropped within the multiplication.

3.3. Taylor Analysis

As range analysis ensures the inclusion of the exact result and can provide rigorous bounds, it finds wide application as an a posteriori analysis tool within self-validated numerical libraries. To improve the obtained bounds, especially in non-linear cases, further extensions were made on the basis of Taylor approximations.

Definition 10. Let $f : D \subseteq \mathbb{R}^v \to \mathbb{R}$ be a function that is $n+1$ times continuously partially differentiable on D. The inclusion of $f(x)$ can be expressed in terms of a Taylor model which is composed of a nth order Taylor expansion of f around an expansion point $a \in D$ and an interval remainder I_R bounding all errors (truncation, rounding, ...). [23]

$$\forall_{x \in D}\ f(x) \in T_{n;f}(x) + I_R \tag{3.30}$$

Using Definition 10, enclosures for any (*sufficiently smooth*) function can be computed. The methods presented until now can be summarized to merge worst-case inclusions for every sub-expression to obtain a rigorous result. The Taylor analysis presented in [24] tries to shift from local worst-cases to globally solving for the maximum error. Let $e := (e_0, \ldots, e_k)$ be an error vector composed of independent error variables $e_i \in [-\bar{\epsilon}, \bar{\epsilon}]$ and x the input for an exact function f. Let $\widetilde{f_f}$ denote the model of calculating f using floating-point arithmetic. The absolute error is given in Equation (3.31).

$$\begin{aligned} \left|\delta_{f(x)}\right| &= \left|\tilde{f}(x) - f(x)\right| \\ &\leq \left|\tilde{f_f}(x, e) - f(x)\right| \end{aligned} \tag{3.31}$$

The most important step within the Taylor analysis is the construction of a first order Taylor polynomial around $e = (0, \ldots, 0)$. Based on the equality of calculating the exact function f and the floating point model $\tilde{f_f}$ without any errors $\tilde{f_f}(x, 0) = f(x)$, we obtain the Equation (3.32) for the absolute rounding error.

$$\begin{aligned} \left|\delta_{f(x)}\right| &\leq \left|\tilde{f_f}(x, e) - f(x)\right| \\ &= \left|\tilde{f_f}(x, 0) + T_{1;\tilde{f_f}}(x, 0) + R_{1;\tilde{f_f}}(x, \zeta) - f(x)\right| \\ &= \left|\sum_{i=0}^{k} \frac{\partial \tilde{f_f}}{\partial e_i}(x, 0) \cdot e_i + R_{1;\tilde{f_f}}(x, \zeta)\right| \\ &\leq \left|\sum_{i=0}^{k} \frac{\partial \tilde{f_f}}{\partial e_i}(x, 0) \cdot e_i\right| + \left|R_{1;\tilde{f_f}}(x, \zeta)\right| \\ &\leq \bar{\epsilon} \cdot \left|\sum_{i=0}^{k} \frac{\partial \tilde{f_f}}{\partial e_i}(x, 0)\right| + M \\ \forall_{x,\zeta \in [-\bar{\epsilon}, \bar{\epsilon}]^{k+1}}\ M &\geq \left|R_{1;\tilde{f_f}}(x, \zeta)\right| \end{aligned} \tag{3.32}$$

Taylor analysis is the first method allowing for actual symbolic computation, which in turn reduces the necessity of outward rounding methods and similar. The main contribution of error within the computation is the truncation error of the Taylor expansion and the bounding of the remainder with global optimization techniques. It is required for the global optimization to return rigorous results and therefore guarantee inclusion of the exact remainder bound. The method presented in [23] additionally accounts for subnormal computations by introducing an additional absolute deviation from the exact result. We leave these deviations out within this thesis, as we assume no over- or underflow as defined in Definition 3. Furthermore, the chosen values for the absolute deviations were not explained sufficiently and only correlated to the IEEE precisions. However, the over- and underflow of numbers is also depending on the available exponent bits. As this correlation was not presented within the paper and is required for supporting artificial floating-point types with artificial exponent sizes and precisions, this is left open for future work.

4. Rigorous A Priori Analysis of Round-Off Errors

In Chapter 3 we explored currently available methods for analyzing the error introduced by rounding and discussed potential shortcomings. Based on them, we are now presenting different methods for the a priori analysis of absolute round-off errors. We start by defining the underlying error model, that was already used implicitly in previous sections, when dealing with floating-point arithmetic by applying Corollary 6. The relative rounding error ϵ_r is limited to $\epsilon_r \in [-\bar{\epsilon}, \bar{\epsilon}]$.

Definition 11. Let $\tilde{x}$ denote the floating-point model of x. Furthermore, let ϵ_x be limited to $\epsilon_x \in [-\bar{\epsilon}, \bar{\epsilon}]$. We define the model using Equation (4.1). It assumes no over- or underflow, as defined in Definition 3.

$$\begin{aligned} \tilde{x} &= x,\ x \in \mathbb{M} \\ \tilde{x} &= rd(x),\ x \in \mathbb{R} \setminus \mathbb{M} \\ x_1 \circ x_2 &= rd(\tilde{x_1} \circ \tilde{x_2}) \\ rd(x) &= \begin{cases} x & x \in \{2^i, i \in [e_{min}, e_{max}]\} \\ x \cdot (1+\epsilon) & \text{otherwise} \end{cases} \end{aligned} \tag{4.1}$$

This basic error model could be further extended by defining special cases for e.g. multiplication or division with a non-negative power of 2 [23]. The absolute error is obtained from the difference of exact calculation f to floating-point calculation $\tilde{f}$.

$$|\delta_f| = |f - \tilde{f}| \tag{4.2}$$

For the following methods, it is mandatory to expose the machine accuracy $\bar{\epsilon}$ as a free variable. The methods therefore have to include support for symbolic expressions. Furthermore, they have to support constraining the values of free variables to a domain D. This may result in the calculations diverging, based on the possible values, and the result being composed of differentiating results that are valid within a given subdomain $D_i \subseteq D$. For the machine accuracy $\bar{\epsilon}$ the domain is based on the available precision.

$$\begin{aligned} \bar{\epsilon} \in D &= (2^{-p_{max}}, 2^{-p_{min}}) \\ p_{max} &> 0 \\ p_{max} &\geq p_{min} \end{aligned} \tag{4.3}$$

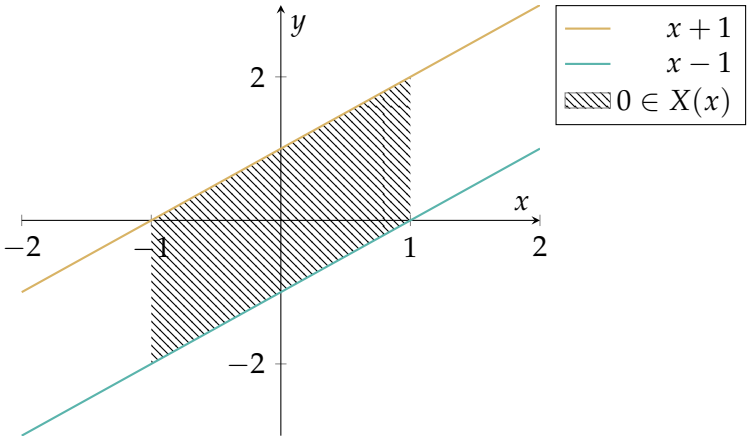


Figure 4.1.: Symbolic interval $X = [x-1, x+1]$ with x bound to $[-2, 2]$ visualizing the difference in value inclusion

4.1. Symbolic Range Analysis

Symbolic range analysis is based on the methods presented throughout Section 3.2. The relative error ϵ_r translates to the interval $[-\bar{\epsilon}, +\bar{\epsilon}]$. As interval analysis ensures the inclusion of the exact result, the absolute error is contained within the interval extension of calculating δ_f.

$$|\delta_f| \leq |f - \tilde{f}| \tag{4.4}$$

The interval operators presented in the previous sections are only defined for real-valued intervals. It is therefore required to introduce an extension to support symbolic boundaries while still ensuring the inclusion property given in Equation (3.19). In contrast to the previous operators having constant bounds $\underline{X}, \overline{X} \in \mathbb{R}$, the bounds now have to depend on symbolic expressions.

Definition 12. A symbolic interval X of order n defines a function $X : D \subseteq \mathbb{R}^n \mapsto M \subseteq \mathbb{R}$ that returns a real-valued interval $M \subseteq \mathbb{R}$ upon evaluation.

Example. Let $X(x) = \left[\overline{X(x)}, \underline{X(x)}\right] = [x-1, x+1]$ denote a symbolic interval with $x \in [-2, 2]$. We may obtain a valid real-valued interval by substituting x with a concise value from its domain. This symbolic interval is shown in Figure 4.1.

Remark. Within this thesis, only symbolic intervals of order 1 are taken into account. The term symbolic interval is from now on used equivalently to the term symbolic interval of order 1. In the case of round-off error analysis, we are only interested in boundaries depending on the free variable $\bar{\epsilon}$.

Let f be a real-valued function and X be a real-valued interval. We can obtain an interval extension from a monotonic characteristic of $f(x)$ with $x \in X$. Given that f is monotonically increasing or decreasing, only the functional values of the bounds have

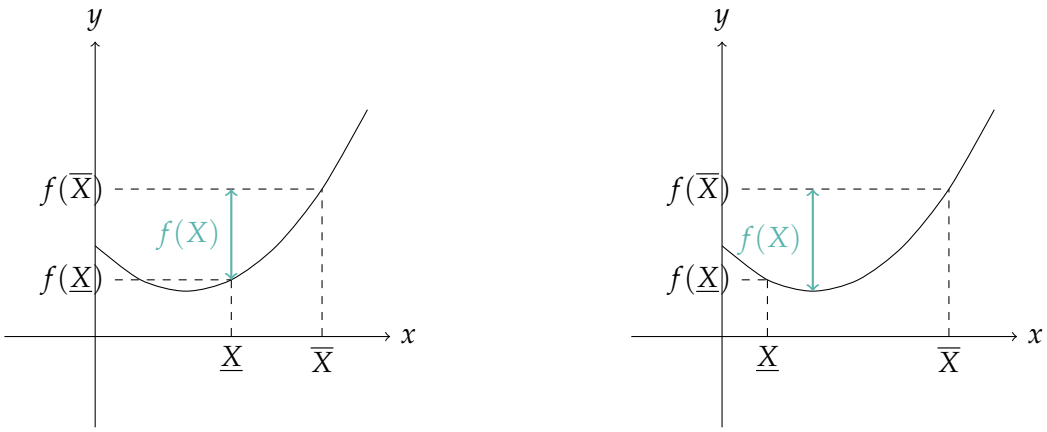


Figure 4.2.: Interval extension $f(X)$ of a real-valued function f in a monotonic and non-monotonic setting

to be calculated [17]. As Figure 4.2 shows, the same does not apply for non-monotonic functions. We seek to find a definition allowing for support of a wide variety of functions f. For this, we introduce the assumption of f being once continuously differentiable within its domain. This allows for the construction of the resulting interval solely based on the interval bounds and the possible extrema of f given within the bounds of the input interval. Instead of determining the extrema repeatedly for every possible function, they can be provided for common functions e.g. $x^{a \in \mathbb{Z}}, x^{\frac{1}{n \in \mathbb{Z}}}$. This allows for a speedup while still ensuring support for all once continuously differentiable functions.

Definition 13. Given the arithmetic operations $+, -, \cdot, /$, log, exp or in general a function $f \in C^1$ the interval extensions are defined in Equation (4.5). [17]

$$\begin{aligned}
X + Y &= \left[\underline{X} + \underline{Y}, \overline{X} + \overline{Y}\right] \\
X - Y &= \left[\underline{X} - \overline{Y}, \overline{X} - \underline{Y}\right] \\
\exp(X) &= \left[\exp(\underline{X}), \exp(\overline{X})\right] \\
\frac{X}{Y} &= X \cdot \frac{1}{Y} \\
&= X \cdot \left[1/\overline{Y}, 1/\underline{Y}\right], \ 0 \notin Y \\
\log(X) &= \left[\log(\underline{X}), \log(\overline{X})\right], \ X \cap (-\infty, 0] = \emptyset \\
X \cdot Y &= \left[\min(S), \max(S)\right], \ S = \left\{\underline{X}\underline{Y}, \underline{X}\overline{Y}, \overline{X}\underline{Y}, \overline{X}\overline{Y}\right\} \\
f(X) &= \left[\min(S), \max(S)\right], \ S = \left\{f(\underline{X}), f(\overline{X})\right\} \cup \left\{x \in X : f'(x) = 0\right\}
\end{aligned} \tag{4.5}$$

Using Definition 13, a wide range of functions can be supported within real-valued interval arithmetic.

Example. Let $X = [x - 1, x + 1]$ and $Y = [2, 2]$ be two symbolic intervals with x bound on the domain $[-2, 2]$. We want to determine the result of the exponentiation X^Y. We know that for $f(x) = x^2$ a global extremum is given when x is zero.

However, as shown within Figure 4.1, this extremum is not contained for all evaluations of the symbolic interval.

To construct subdomains for a symbolic interval Y - let it be because of domain restrictions or extrema inclusion - we have to determine when values are included within Y. Given a domain G, we want to calculate for which y the symbolic interval Y evaluates to a real-valued interval containing any value from G. For this, we use the comparisons given in Table 4.1 and determine for which values of y they are fulfilled.

D	$\overline{Y(y)}$	$\underline{Y(y)}$
$[a,b]$	$\geq b$	$\leq a$
$(a,b]$	$\geq b$	$\leq a$
$(-\infty,b]$		$\leq b$
$(-\infty,b)$		$\leq b$
$[a,\infty)$	$\geq a$	
(a,∞)	$\geq a$	

Table 4.1.: Comparisons required for determining the (partial) inclusion of a given domain D within a symbolic interval Y

Example. Using the inclusion of $[0,0]$, we obtain the subdomains $[-2,-1)$, $[-1,1]$, $(1,2]$ that are associated with the inclusion or non-inclusion of the extremum 0, as shown below.

$$S(x) = \begin{cases} \{\underline{X}^2, \overline{X}^2\} & x \in [-2,-1) \\ \{\underline{X}^2, \overline{X}^2, 0^2\} & x \in [-1,1] \\ \{\underline{X}^2, \overline{X}^2\} & x \in (1,2] \end{cases} \tag{4.6}$$
$$X^Y = [\min(S(x)), \max(S(x))]$$

After determining the sectioning of the domain based on the extrema inclusion, it is still not possible to calculate the interval extension. Thus, re-definitions of the min and max operators for symbolic expressions have to be defined. Let $S = [s_0, \ldots, s_{n-1}]$ denote a list of symbolic expressions. We want to determine the minimum and maximum within a given domain D, requiring us to specify the ordering of S. However, depending on the currently considered subdomain $D_i \subseteq D$ the ordering may vary or not be total.

Example. Let $S = \left[(x-1)^2, (x+1)^2\right]$ and $D = [-2,2]$. We want to identify the ordering of S. For the subdomain $[-2,0)$ a strict total order is given with $(x-1)^2 > (x+1)^2$. The reverse applies for the subdomain $(0,2]$ with the strict total order $(x+1)^2 > (x-1)^2$. For the complete domain D no total order can be obtained, as neither $s_0 \leq s_1$ nor $s_1 \geq s_0$ holds.

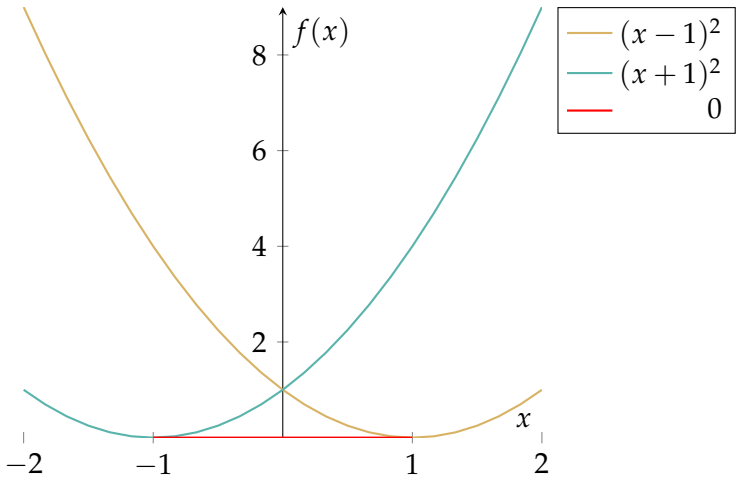


Figure 4.3.: Visualization of the changing interval boundaries and necessity to include evaluation of extrema, given the exponentiation X^Y with $X = [x-1, x+1]$ and $Y = [2, 2]$

The evaluation of minimum and maximum is composed of two main steps. In the first step, it is to be computed for every possible pair (s_i, s_j) with $i \neq j$ and $i, j \in \{0, \ldots, n-1\}$ for which subdomain the statement $s_i \leq s_j$ or $s_i \geq s_j$ respectively holds. The second step consists of the association of subdomains to their respective minimum or maximum expression. To simplify these steps, we define the comparison matrix C.

$$C := \begin{pmatrix} D & D \setminus C_{1,0} & D \setminus C_{2,0} & \ldots & D \setminus C_{n-1,0} \\ s_0 \leq s_1 & D & D \setminus C_{2,1} & \ldots & D \setminus C_{n-1,1} \\ s_0 \leq s_2 & s_1 \leq s_2 & D & \ldots & D \setminus C_{n-1,2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ s_0 \leq s_{n-1} & s_1 \leq s_{n-1} & s_2 \leq s_{n-1} & \ldots & D \setminus C_{n-1,n-2} \end{pmatrix} \tag{4.7}$$

Remark. The entries $D \setminus C_{i,j}$ no longer describe the subdomains where $s_i \leq s_j$ but rather $s_i < s_j$ holds. This has to be accounted for, but is rather unproblematic, as for the subdomain where both $s_i \leq s_j$ and $s_j \leq s_i$ hold s_i and s_j are equivalent.

After constructing matrix C for the respective comparison operator, the association of subdomains to extremum is a simple chain of set intersection operations. We obtain a vector E in which each element E_i describes for which subdomain the expression s_i is the extremum.

$$E^T := \begin{pmatrix} \bigcap_{i \in \{0,n-1\}} C_{i,0} \\ \vdots \\ \bigcap_{i \in \{0,n-1\}} C_{i,n-1} \end{pmatrix} \tag{4.8}$$

As we defined the upper triangular matrix of C by using the set difference and therefore the strict comparison, we also obtain the following properties for our vector E.

$$\begin{aligned} \bigcup_{i\in\{0,n-1\}} E_i &= D \\ \forall_{i,j\in\{0,n-1\}:i\neq j} E_i \cap E_j &= \emptyset \end{aligned} \tag{4.9}$$

After we defined the min and max operators to work for symbolic expressions, we now have to determine how to construct valid symbolic intervals. Given the list of sub-expressions S and the vectors E_{min} and E_{max} obtained from $\min(S)$ and $\max(S)$ respectively, we define the interval extension for f in Equation (4.10).

$$\begin{aligned} M &:= \left\{ \left(E_{min,i} \cap E_{max,j}, [s_i, s_j] \right) : i,j \in \{0,\dots,n-1\} \right\} \\ f(X) &= \{ (D_i, I_i) : (D_i, I_i) \in M, D_i \neq \emptyset \} \end{aligned} \tag{4.10}$$

Example. We finalize our example X^Y by obtaining the resulting interval extension. For the domains $[-2,-1),(1,2]$ provided by the non-inclusion of the global minimum zero, the minima and maxima can be determined visually using Figure 4.3. Within the domain $[-1,1]$ we calculate the matrices C_{min}, C_{max} and vectors E_{min}, E_{max} for $S = \left\{0,(x-1)^2,(x+1)^2\right\}$.

$$\begin{aligned} C_{min} &= \begin{pmatrix} D & \emptyset & \emptyset \\ [-1,1] & D & [-1,0) \\ [-1,1] & [0,1] & D \end{pmatrix} \\ E_{min} &= (D,\emptyset,\emptyset) \\ C_{max} &= \begin{pmatrix} D & [-1,1) & (-1,1] \\ [1,1] & D & (0,1] \\ [-1,-1] & [-1,0] & D \end{pmatrix} \\ E_{max} &= (\emptyset,[-1,0],(0,1]) \\ X^Y &= \begin{cases} \left[(x+1)^2,(x-1)^2\right] & x\in[-2,-1) \\ \left[0,(x-1)^2\right] & x\in[-1,0] \\ \left[0,(x+1)^2\right] & x\in(0,1] \\ \left[(x-1)^2,(x+1)^2\right] & x\in(1,2] \end{cases} \end{aligned} \tag{4.11}$$

Problem. Using Definition 13, we encounter a problem for dividing an interval $1/Y$. How should a division be defined, if the divisor contains a zero $0 \in Y$? It is possible to return the correct, but useless, interval containing all numbers $[-\infty,\infty] = \mathbb{R}$ or perform case distinction on the divisor and dividend. This allows for a tighter enclosure that is however, still defined by infinity. [17]

$$1/Y = \begin{cases} [1/\underline{Y},\infty) & \underline{Y} = 0 < \overline{Y} \\ (-\infty,1/\underline{Y}] \cup [1/\overline{Y},\infty) & \underline{Y} < 0 < \overline{Y} \\ (-\infty,1/\underline{Y}] & \underline{Y} < \overline{Y} = 0 \end{cases} \tag{4.12}$$

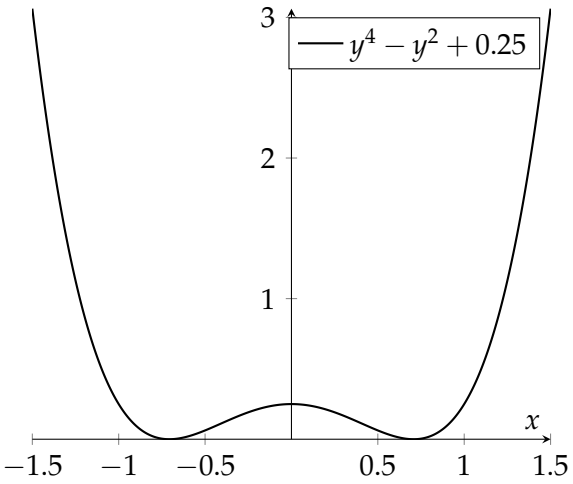


Figure 4.4.: Polynomial expression equaling zero at two distinct points. When this expression describes a lower symbolic interval boundary the division would not be defined at those points.

Because the introduction of infinity into the calculations adds further complexity and problems, like the division of intervals bound by infinity, the multiplication of infinity with 0 and subtraction of intervals bound by infinity, without offering a great value for the given context, we define the division for symbolic intervals to be undefined in cases, where a 0 may be contained within the divisor.

Definition 14. Let Y be a symbolic interval with y in the domain D. The expression $1/Y$ is defined by excluding subdomains, in which Y contains a zero.

$$\begin{aligned} 1/Y &= \left[1/\overline{Y}, 1/\underline{Y}\right],\ y \in \bigcup_{i=1}^{n} D_i \\ \forall_{y \in D_i} 0 \notin Y &\wedge \bigcap_{i=1}^{n} D_i = \emptyset \end{aligned} \tag{4.13}$$

Example. Let $Y = \left[y^4 - y^2 + 0.25, y^4 - y^2 + 0.25\right]$ be a symbolic interval with $y \in [-1, 1)$. We calculate $1/Y$ by applying Definition 14 and determine those subdomains, in which Y does and does not contain 0. The result provided below can be obtained from the visualized interval boundary in Figure 4.4.

$$\begin{aligned} 1/Y &= [1/y^4 - y^2 + 0.25, 1/y^4 - y^2 + 0.25] \\ y &\in \left[-1, -\frac{\sqrt{2}}{2}\right) \cup \left(\frac{-\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\right) \cup \left(\frac{\sqrt{2}}{2}, 1\right) \end{aligned} \tag{4.14}$$

Definition 14 can now be generalized for all real-valued functions that are taken into consideration within the symbolic range analysis.

Definition 15. Let Y be a symbolic interval with y in the domain D. The interval extension $f(Y)$ is limited to those subdomains of D, in which the function $f(x)$ is defined for every possible value x of the symbolic interval.

$$\begin{gathered} f(Y) = \left[\underline{f(Y)}, \overline{f(Y)}\right],\ y \in \bigcup_{i=1}^{n} D_i \\ \forall_{y \in D_i} \forall_{x \in Y} f(x) \text{ defined} \wedge \bigcap_{i=1}^{n} D_i = \emptyset \end{gathered} \tag{4.15}$$

Example. Let $Y = [y-1, y+1]$ be a symbolic interval with $y \in [-1, 3]$. We apply the operation $\sqrt{Y}$. It is known, that the real-valued square root $\sqrt{x}$ is only defined for non-negative numbers $D = [0, \infty)$. The domain of y is therefore restricted to $[1, 3]$, as shown within Figure 4.5.

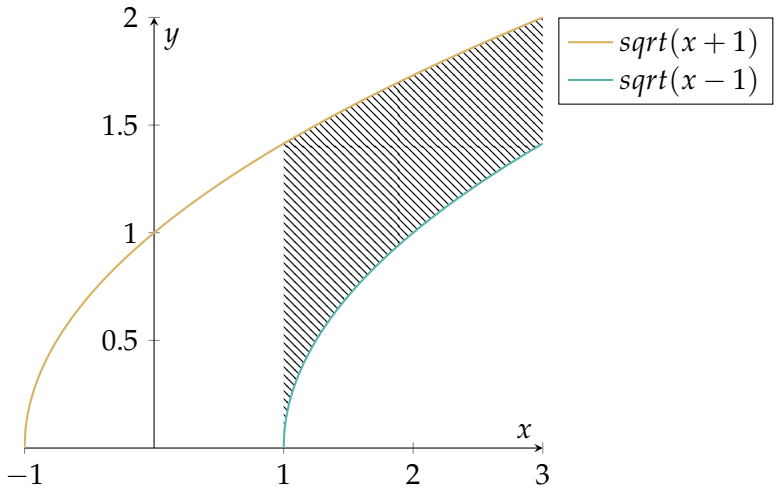


Figure 4.5.: The domain of x is limited to guarantee the function to be defined for all possible elements within all possible evaluations of the symbolic interval to a real-valued interval.

Problem. Using the presented interval extensions and procedures, we are able to rigorously calculate the symbolic interval expression $|f - \tilde{f}|$. However, we have to add another layer of approximation to guarantee decidability for computing the minimum and maximum. Because we rely on the inequalities $\geq, \leq$ for determining the order, decidability is not guaranteed for all inputs f [25]. As we know that these inequalities are decidable for polynomials [26], we convert the input f to a polynomial using Taylor expansion. To provide rigorous results, we bound the remainder based on its Lagrange form [27] using either an analytic or global optimization approach. Based on the idea of Taylor models presented in Section 4.2, we obtain an inclusion of $\tilde{f}$ using Equation (4.16).

The only difference is given by the interval remainder I_R being symbolic.

$$\begin{aligned} C &:= M\frac{|x-a|^{n+1}}{(n+1)!} \\ M &\geq \max_x \left| f^{(n+1)}(x) \right| \\ &\Rightarrow C \geq \left| R_{f;n}(x) \right| \\ &\Rightarrow \tilde{f} \in \left[T_{\tilde{f};n}(x), T_{\tilde{f};n} \right] + [-C, C] \end{aligned} \tag{4.16}$$

4.2. Taylor Analysis

From Section 4.1, we obtained an analysis method based on the idea of bounding sub-expressions locally and then merging them. We now define a method of locally modeling and then bounding the error globally. Precisely, the goal of this section is to extend the method described in Section 3.3, initially developed within [24]. Based on Equation (3.32), we obtain the Equation (4.17) for the absolute error, given a nth-order Taylor expansion.

$$\begin{aligned} \left|\delta_{f(x)}\right| &\leq \left(\sum_{i=1}^{n} \bar{\epsilon}^n \left| \sum_{I \in \{0,\dots,k\}^i} \frac{\partial \tilde{f_f}}{\partial e_{I_0} \dots \partial e_{I_{i-1}}}(x,0) \right|\right) + M \\ \forall_{x,\zeta \in [-\bar{\epsilon},\bar{\epsilon}]^{k+1}}\ M &\geq \left| R_{n;\tilde{f_f}}(x,\zeta) \right| \end{aligned} \tag{4.17}$$

The method $\tilde{f_f}$ represents our model of the floating-point calculation of f. It is expressed using independent error variables $e_i \in [-\bar{\epsilon},\bar{\epsilon}]$, which are introduced by the different computational operations as specified in Definition 11. Following, we restrict the Taylor expansions of our method $\tilde{f_f}$

$$\begin{aligned} \tilde{f_f}(x) &= f(x) + \sum_{i=1}^{n} \sum_{I \in \{0,\dots,k\}^i} \frac{\partial \tilde{f_f}}{\partial e_{I_0} \dots \partial e_{I_{i-1}}}(x,0) + R_{n;\tilde{f_f}}(x,\zeta) \\ \zeta &\in [-\bar{\epsilon},\bar{\epsilon}]^{k+1} \end{aligned} \tag{4.18}$$

to order two. The definitions and proofs could be extended to support any order. However, in the form given below, this would require intensively nested structures for which resource requirements get increasingly problematic.

Definition 16. Let $e = (e_0, \dots, e_k)$ be an error vector. We define a vector C to inherit the structure given in Equation (4.19).

$$C := \left(\begin{bmatrix} C_{0_0} \\ \vdots \\ C_{0_k} \end{bmatrix}, \begin{bmatrix} C_{1_{0,0}} & \dots & C_{1_{0,k}} \\ \vdots & \ddots & \vdots \\ C_{1_{k,0}} & \dots & C_{1_{k,k}} \end{bmatrix} \right) \tag{4.19}$$

Let I denote a chain of indices $I \in [0,k]^n$. We introduce an alternating index notation $C[I]$ for the vector.

$$\begin{aligned} C_{[I]} &:= \begin{cases} C_{0_{I_0}} & n = 1 \\ C_{n-1_{I_0}} & n > 1 \\ \qquad\quad \ddots_{I_{n-2},I_{n-1}} & \end{cases} \\ C_{[0,2]} &= C_{1_{0,2}} \end{aligned} \tag{4.20}$$

Using vector C, we define the tuple (t, C) to be a 2nd-order Taylor form for the function h, given that the following statement holds.

$$\begin{aligned} \forall_x \exists_e \; h(x) &= t(x) + \sum_{i=0}^{k} \left(C_{[i]}(x) \cdot e_i \right) + \sum_{i=0,j=0}^{k} \left(C_{[i,j]}(x) \cdot e_i e_j \right) \\ e_i &\in [-\bar{\epsilon}, \bar{\epsilon}] \end{aligned} \tag{4.21}$$

This equality is denoted using $(t, C) \sim h$ or $h \sim (t, C)$. [24]

Based on the expansion given in Equation (4.18), we can define a tth order term, indexed by an index chain $I \in [0, k]^t$, in the following way.

$$C[I] = \frac{\partial \tilde{t_f}}{\partial e_{I_0} \dots \partial e_{I_{t-1}}}(x, 0) \tag{4.22}$$

Remark. It is important to note that based on the chosen expansion point $e = (0, \dots, 0)$ no e_i can be present within $C[I]$. Following, we use the term Taylor form equivalently to the term 2nd-order Taylor form.

To obtain a valid Taylor form, we have to bound the remainder $R_{n;\tilde{f_f}}(x, \zeta),\ \zeta \in [-\bar{\epsilon}, \bar{\epsilon}]^{k+1}$. Let M be a constant for which $\forall_{x,\zeta \in [-\bar{\epsilon}, \bar{\epsilon}]^{k+1}} \; M \geq \left| R_{n;\tilde{f_f}}(x, \zeta) \right|$ holds. We extend our noise vector from $e = (e_0, \dots, e_k)$ to $e = (e_0, \dots, e_{k+1})$ and extend C as shown in Equation (4.23) to obtain a valid Taylor form. [24]

$$\begin{aligned} C_{[k+1]} &= \frac{M}{\bar{\epsilon}} \\ C_{[k+1,i]} &= C_{[i,k+1]} \\ &= 0 \end{aligned} \tag{4.23}$$

This is a result of the absolute difference between the remainder and the newly introduced term being less than or equal to the machine epsilon. Therefore, it is possible to determine a noise variable e_{k+1} for every initial noise vector $e = (e_0, \dots e_k)$ that satisfies the required equality within Equation (4.21).

$$\begin{aligned} \forall_{x,\zeta \in [-\bar{\epsilon}, \bar{\epsilon}]^{k+1}} \; \left| \frac{R_{n;\tilde{t_f}}(x, \zeta)}{C_{[k+1]}(x)} \right| &\leq \frac{M}{C_{[k+1]}(x)} \\ &= \bar{\epsilon} \\ \Rightarrow \forall_x \forall_e \exists_{e_{k+1} : e_{k+1} \in [-\bar{\epsilon}, \bar{\epsilon}]} \; R_{n;\tilde{t_f}}(x, e) &= C_{[k+1]}(x) \cdot e_{k+1} \end{aligned} \tag{4.24}$$

After we have defined the equality in terms of Equation (4.21) between a function h and the Taylor form (t, C), we combine this with the conversion from a Taylor expansion to a Taylor form, to obtain derivation rules for our floating-point operations.

Conjecture 17. Let $c \in \mathbb{R}$ be any real number. We can obtain a valid Taylor form for $rd(c)$ from Equation (4.25).

$$rd(c) \sim \begin{cases} (c, ([], [])) & c \in \{2^i, i \in [e_{min}, e_{max}]\} \\ (c, ([c], [0]) \end{cases} \tag{4.25}$$

Proof. Using the error model given in Equation (4.1), we obtain the cases listed above for the second order Taylor expansion. □

Conjecture 18. Given $(s, B) \sim h_1$ and $(t, C) \sim h_2$ with the respective error vectors $e_S = (e_{S_0}, \ldots, e_{S_k})$ and $e_T = (e_{T_0}, \ldots, e_{T_l})$ a valid Taylor form for $h_1 + h_2$ can be obtained from Equation (4.26).

$$h_1 + h_2 \sim \left(s + t, \left(\begin{bmatrix} B_{[0]} \\ \vdots \\ B_{[k]} \\ C_{[0]} \\ \vdots \\ C_{[l]} \end{bmatrix}, \begin{bmatrix} B_{[0,0]} & \ldots & B_{[0,k]} & 0 & \ldots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ B_{[k,0]} & \ldots & B_{[k,k]} & 0 & \ldots & 0 \\ 0 & \ldots & 0 & C_{[0,0]} & \ldots & C_{[0,l]} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \ldots & 0) & C_{[l,0]} & \ldots & C_{[l,l]} \end{bmatrix} \right) \right) \tag{4.26}$$

Proof. We start by substituting the equality given in Equation (4.21) for the functions h_1 and h_2. The desired structure can then be determined by grouping the terms of identical order.

$$\begin{aligned} \forall_x \exists_{e_S} \exists_{e_T} \; h_1(x) + h_2(x) &= s(x) + \sum_{i=0}^{k} \left(B_{[i]}(x) \cdot e_{S_i} \right) + \sum_{i=0,j=0}^{k} \left(B_{[i,j]}(x) \cdot e_{S_i} e_{S_j} \right) \\ &\quad + t(x) + \sum_{i=0}^{l} \left(C_{[i]}(x) \cdot e_{T_i} \right) + \sum_{i=0,j=0}^{l} \left(C_{[i,j]}(x) \cdot e_{T_i} e_{T_j} \right) \\ &= (s(x) + t(x)) \\ &\quad + \sum_{i=0}^{k} \left(B_{[i]}(x) \cdot e_{S_i} \right) + \sum_{i=0}^{l} \left(C_{[i]}(x) \cdot e_{T_i} \right) \\ &\quad + \sum_{i=0,j=0}^{k} \left(B_{[i,j]}(x) \cdot e_{S_i} e_{S_j} \right) + \sum_{i=0,j=0}^{l} \left(C_{[i,j]}(x) \cdot e_{T_i} e_{T_j} \right) \end{aligned} \tag{4.27}$$

□

Conjecture 19. Given $(s, B) \sim h_1$ and $(t, C) \sim h_2$ with the respective error vectors $e_S = (e_{S_0}, \ldots, e_{S_k})$ and $e_T = (e_{T_0}, \ldots, e_{T_l})$ a valid Taylor form for $h_1 - h_2$ can be obtained from Equation (4.28).

$$h_1 - h_2 \sim \left(s - t, \left(\begin{bmatrix} B_{[0]} \\ \vdots \\ B_{[k]} \\ -C_{[0]} \\ \vdots \\ -C_{[l]} \end{bmatrix}, \begin{bmatrix} B_{[0,0]} & \ldots & B_{[0,k]} & 0 & \ldots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ B_{[k,0]} & \ldots & B_{[k,k]} & 0 & \ldots & 0 \\ 0 & \ldots & 0 & -C_{[0,0]} & \ldots & -C_{[0,l]} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \ldots & 0) & -C_{[l,0]} & \ldots & -C_{[l,l]} \end{bmatrix} \right) \right) \tag{4.28}$$

Proof. The proof is analogous to the proof for $h_1 + h_2$. □

Conjecture 20. Given $(t, C) \sim h$ with the error vector $e = (e_0, \ldots, e_k)$ a valid Taylor form for $rd(h)$ can be obtained from Equation (4.29).

$$\begin{aligned} rd(h) &\sim \left(t, \begin{bmatrix} C_{[0]} \\ \vdots \\ C_{[k]} \\ t \end{bmatrix}, \begin{bmatrix} C_{[0,0]} & \ldots & C_{[0,k]} & C_{[0]} \\ \vdots & \ddots & \vdots & \vdots \\ C_{[k,0]} & \ldots & C_{[k,k]} & C_{[k]} \\ C_{[0]} & \ldots & C_{[k]} & 0 \end{bmatrix} \right) + (R_F, R_C) \\ R_F &= 0 \\ R_C &= \left(\begin{bmatrix} \bar{\epsilon}^2 M \end{bmatrix}, \begin{bmatrix} 0 \end{bmatrix} \right) \\ M &\geq \max_x \sum_{i=0,j=0}^{k} \left| C_{[i,j]}(x) \right| \end{aligned} \tag{4.29}$$

Proof. We start by applying the error model defined in Equation (4.1) to the rounding operation and then substituting the Taylor form equality given in Equation (4.21) for the function h. We obtain an equation, that can be split into the addition of two Taylor forms and a remainder.

$$\begin{aligned} \forall_x \exists_{e_{k+1}} \; rd\left(h\left(x\right)\right) &= h\left(x\right) \cdot \left(1 + e_{k+1}\right) \\ \forall_x \exists_e \exists_{e_{k+1}} \; rd\left(h\left(x\right)\right) &= t\left(x\right) + \sum_{i=0}^{k} \left(C_{[i]}\left(x\right) \cdot e_i \right) + \sum_{i=0,j=0}^{k} \left(C_{[i,j]}\left(x\right) \cdot e_i e_j \right) \\ &\quad + e_{k+1} \cdot t\left(x\right) + \sum_{i=0}^{k} \left(C_{[i]}\left(x\right) \cdot e_i e_{k+1} \right) + R\left(x, e, e_{k+1}\right) \\ &= (t, C) + \left(0, \begin{bmatrix} 0 \\ \vdots \\ 0 \\ t \end{bmatrix}, \begin{bmatrix} 0 & \ldots & 0 & C_{[0]} \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \ldots & 0 & C_{[k]} \\ C_{[0]} & \ldots & C_{[k]} & 0 \end{bmatrix} \right) \\ &\quad + R(x, e, e_{k+1}) \end{aligned} \tag{4.30}$$

The rounding introduces the remainder $R(x, e, e_{k+1})$ that is composed of terms of order 3 and is to be bound to obtain rigorous results.

$$\begin{aligned} R\left(x, e, e_{k+1}\right) &= e_{k+1} \cdot \sum_{i=0,j=0}^{k} \left(C_{[i,j]}\left(x\right) \cdot e_i e_j \right) \\ \left| \frac{R}{R_{C_{[0]}}} \right| &\leq \frac{\bar{\epsilon} \cdot \sum_{i=0,j=0}^{k} \left| C_{[i,j]} \cdot e_i e_j \right|}{\bar{\epsilon}^2 M} \\ &\leq \frac{\bar{\epsilon}^2 \cdot \sum_{i=0,j=0}^{k} \left| C_{[i,j]} \right|}{\bar{\epsilon} M} \leq \bar{\epsilon} \end{aligned} \tag{4.31}$$

□

Conjecture 21. Given $(s, B) \sim h_1$ and (t, C) with the respective error vectors $e_S = (e_{S_0}, \ldots, e_{S_k})$ and $e_T = (e_{T_0}, \ldots, e_{T_l})$ a valid Taylor form for $h_1 \cdot h_2$ can be obtained from Equation (4.32).

$$\begin{aligned} h_1 \cdot h_2 \sim & \left(s \cdot t, \left(\begin{bmatrix} t \cdot B_{[0]} \\ \vdots \\ t \cdot B_{[k]} \\ s \cdot C_{[0]} \\ \vdots \\ s \cdot C_{[l]} \end{bmatrix}, \begin{bmatrix} t \cdot B_{[0,0]} & \ldots & t \cdot B_{[0,k]} & B_{[0]} \cdot C_{[0]} & \ldots & B_{[0]} \cdot C_{[l]} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ t \cdot B_{[k,0]} & \ldots & t \cdot B_{[k,k]} & B_{[k]} \cdot C_{[0]} & \ldots & B_{[k]} \cdot C_{[l]} \\ C_{[0]} \cdot B_{[0]} & \ldots & C_{[0]} \cdot B_{[k]} & s \cdot C_{[0,0]} & \ldots & s \cdot C_{[0,l]} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ C_{[l]} \cdot B_{[0]} & \ldots & C_{[l]} \cdot B_{[k]} & s \cdot C_{[l,0]} & \ldots & s \cdot C_{[l,l]} \end{bmatrix} \right) \right) \\ & + (R_F, R_C) \\ R_F &= 0 \\ R_C &= \left(\left[\ \bar{\epsilon}^2 (M_1 + M_2) + \bar{\epsilon}^3 M_2 \ \right], [0] \right) \\ M_1 &\geq \max_x \sum_{i=0}^{k} \sum_{m=0,g=0}^{l} \left| B_{[i]}(x) \cdot C_{[m,g]}(x) \right| \\ M_2 &\geq \max_x \sum_{i=0,j=0}^{k} \sum_{m=0}^{l} \left| B_{[i,j]}(x) \cdot C_{[m]}(x) \right| \\ M_3 &\geq \max_x \sum_{i=0,j=0}^{k} \sum_{m=0,g=0}^{l} \left| B_{[i,j]}(x) \cdot C_{[m,g]}(x) \right| \end{aligned} \tag{4.32}$$

Proof. We start by substituting the equality given in Equation (4.21) for the functions h_1 and h_2. The multiplication introduces the remainder $R(x, e_S, e_T)$, as given within Equation (4.34). Because the different components of the remainder $R(x, e_S, e_T)$ contain terms of order 3 and 4, it is required to bound them to obtain rigorous results. For this, in Equation (4.35) we use the triangle inequality in combination with the bounding constants defined in Equation (4.32).

A valid Taylor form is obtained, as the term $R_{C_{[0]}}$ has a relative deviation from R that is less than or equal to the machine epsilon, allowing for an equality satisfying e_{k+1} to exist for every initial error vector e.

Substitution

$$\begin{aligned}
\forall_x \exists_{e_S} \exists_{e_T}\ h1 \cdot h2 = s\left(x\right) + \sum_{i=0}^{k} \left(B_{[i]}\left(x\right) \cdot e_{S_i} \right) + \sum_{i=0,j=0}^{k} \left(B_{[i,j]}\left(x\right) \cdot e_{S_i} e_{S_j} \right) \\
\cdot t\left(x\right) + \sum_{i=0}^{l} \left(C_{[i]}\left(x\right) \cdot e_{T_i} \right) + \sum_{i=0,j=0}^{l} \left(C_{[i,j]}\left(x\right) \cdot e_{T_i} e_{T_j} \right) \\
= \left(s\left(x\right) \cdot t\left(x\right)\right) \\
+ \sum_{i=0}^{l} \left(s\left(x\right) \cdot C_{[i]}\left(x\right) \cdot e_{T_i} \right) + \sum_{i=0}^{k} \left(t\left(x\right) \cdot B_{[i]}\left(x\right) \cdot e_{S_i} \right) \\
+ \sum_{i=0}^{k} \sum_{j=0}^{l} B_{[i]}\left(x\right) \cdot C_{[j]}\left(x\right) \cdot e_{S_i} e_{T_j} \\
+ \sum_{j=0}^{k} t\left(x\right) \cdot B_{[i,j]}\left(x\right) \cdot e_{S_i} e_{S_j} \\
+ \sum_{i=0,j=0}^{l} \left(s\left(x\right) \cdot C_{[i,j]}\left(x\right) \cdot e_{T_i} e_{T_j} \right) + R\left(x, e_S, e_T\right)
\end{aligned} \tag{4.33}$$

Remainder

$$\begin{aligned}
R(x, e_S, e_T) &= R_1(x, e_S, e_T) + R_2(x, e_S, e_T) + R_3(x, e_S, e_T) \\
R_1(x, e_S, e_T) &= \sum_{i=0}^{k} \sum_{m=0,g=0}^{l} B_{[i]}(x) \cdot C_{[m,g]}(x) \cdot e_{S_i} e_{T_m} e_{T_g} \\
R_2(x, e_S, e_T) &= \sum_{i=0,j=0}^{k} \sum_{m=0}^{l} B_{[i,j]}(x) \cdot C_{[m]}(x) \cdot e_{S_i} e_{S_j} e_{T_m} \\
R_3(x, e_S, e_T) &= \sum_{i=0,j=0}^{k} \sum_{m=0,g=0}^{l} B_{[i,j]}(x) \cdot C_{[m,g]}(x) \cdot e_{S_i} e_{S_j} e_{T_m} e_{T_g}
\end{aligned} \tag{4.34}$$

Bounding

$$
\begin{aligned}
|R_1| &\leq \bar{\epsilon}^3 \cdot \sum_{i=0}^{k} \sum_{m=0,g=0}^{l} \left| B_{[i]} \cdot C_{[m,g]} \right| \leq \bar{\epsilon}^3 M_1 \\
|R_2| &\leq \bar{\epsilon}^3 \cdot \sum_{i=0,j=0}^{k} \sum_{m=0}^{l} \left| B_{[i,j]} \cdot C_{[m]} \right| \leq \bar{\epsilon}^3 M_2 \\
|R_3| &\leq \bar{\epsilon}^4 \cdot \sum_{i=0,j=0}^{k} \sum_{m=0,g=0}^{l} \left| B_{[i,j]} \cdot C_{[m,g]} \right| \leq \bar{\epsilon}^4 M_3 \\
\left| \frac{R}{R_{C_{[0]}}} \right| &\leq \frac{|R_1| + |R_2| + |R_3|}{\bar{\epsilon}^2(M_1 + M_2) + \bar{\epsilon}^3 M_3} \leq \frac{\bar{\epsilon}^3 M_1 + \bar{\epsilon}^3 M_2 + \bar{\epsilon}^4 M_3}{\bar{\epsilon}^2(M_1 + M_2) + \bar{\epsilon}^3 M_3} = \bar{\epsilon}
\end{aligned}
\tag{4.35}
$$

□

Conjecture 22. Given $(t, C) \sim h$ with the error vector $e = (e_0, \ldots, e_k)$ a valid Taylor form for $g(h)$ can be obtained from Equation (4.36). Let h_T denote a compact form for writing the expanded Taylor form $h_T := t(x) + \sum_{i=0}^{k} \left(C_{[i]}(x) \cdot e_i \right) + \sum_{i=0,j=0}^{k} \left(C_{[i,j]}(x) \cdot e_i e_j \right)$.

$$
\begin{aligned}
g(h) &\sim \left(g(t), \left(\left[\; g'(t) \cdot C_0 \; \right], C_1 \right) \right) + (R_F, R_C) \\
C_{[i,j]} &= \frac{1}{2} \cdot \left(g''(t) \cdot C_{[i]} + g'(t) \cdot \left(C_{[j,i]} + C_{[i,j]} \right) \right) \\
R_F &= 0 \\
R_C &= \left(\left[\frac{1}{6} \cdot \left(\bar{\epsilon}^2 \cdot M_a + \bar{\epsilon}^2 \cdot M_b + \bar{\epsilon}^3 \cdot M_c \right) \right], \left[\; 0 \; \right] \right) \\
M_0 &\geq \max_x \sum_{i=0}^{k} \left| C_{[i]} \right| \\
M_1 &\geq \max_x \sum_{i=0,j=0}^{k} \left| \left(C_{[j,i]} + C_{[i,j]} \right)(x) \right| \\
M_2 &\geq \max_{x, |e_i| \leq \bar{\epsilon}} \left| g''(h_t(x,e)) \right| \\
M_3 &\geq \max_{x, |e_i| \leq \bar{\epsilon}} \left| g'''(h_t(x,e)) \right| \\
M_a &:= 2 \cdot k \cdot M_1 \cdot M_2 \\
M_b &:= k^2 \cdot M_0 \cdot M_3 \\
M_c &:= k^2 \cdot M_1 \cdot M_3
\end{aligned}
\tag{4.36}
$$

Proof. We start by substituting the Taylor form equality given in Equation (4.21) for the function h. In Equation (4.37), we obtain the second order Taylor expansion for g at $e = (0, \ldots, 0)$. The Taylor expansion introduces the remainder $R(x)$ given in Equation (4.38). We proceed to bound the remainder $R(x)$, as it contains terms of order 3 and 4, in Equation (4.2) analogous to Equation (4.35).

Substitution and Taylor expansion

$$\begin{aligned}
\forall_x \exists_e \; g\left(h\left(x\right)\right) &= g\left(t\left(x\right) + \sum_{i=0}^{k}\left(C_{[i]}\left(x\right)\cdot e_i\right) + \sum_{i=0,j=0}^{k}\left(C_{[i,j]}\left(x\right)\cdot e_i e_j\right)\right) \\
&= g\left(h_T\right)\left(x,0\right) + \sum_{i=0}^{k}\frac{g\left(h_T\right)}{\partial e_i}\left(x,0\right)\cdot e_i \\
&\quad + \sum_{i=0,j=0}^{k}\frac{g\left(h_T\right)}{\partial e_i \partial e_j}\left(x,0\right)\cdot\frac{e_i e_j}{2} + R\left(x\right) \\
&= g\left(h_T\right)\left(x,0\right) \\
&\quad + \sum_{i=0}^{k}\left(g'\left(h_T\right)\cdot\left(C_{[i]} + \sum_{l=0}^{k}\left(C_{[l,i]} + C_{[i,l]}\right)\cdot e_l\right)\right)\left(x,0\right)\cdot e_i \\
&\quad + \frac{1}{2}\cdot\sum_{i=0,j=0}^{k}\left(g''\left(h_T\right)\cdot\left(C_{[i]} + \sum_{l=0}^{k}\left(C_{[l,i]} + C_{[i,l]}\right)\cdot e_l\right)\right. \\
&\quad \left. + g'\left(h_T\right)\cdot\left(C_{[j,i]} + C_{[i,j]}\right)\left(x,0\right)\right)\cdot e_i e_j \\
&\quad + R\left(x\right) \\
&= g\left(t\left(x\right)\right) + \sum_{i=0}^{k}\left(g'\left(t\left(x\right)\right)\cdot C_{[i]}\right)\cdot e_i \\
&\quad + \frac{1}{2}\cdot\sum_{i=0,j=0}^{k}\left(g''\left(t\left(x\right)\right)\cdot C_{[i]} + g'\left(t\left(x\right)\right)\cdot\left(C_{[j,i]} + C_{[i,j]}\right)\right)\cdot e_i e_j \\
&\quad + R\left(x\right)
\end{aligned} \tag{4.37}$$

Remainder

$$\begin{aligned}
R\left(x\right) &= \frac{1}{6}\cdot\sum_{g=1}^{3} R_g\left(x,\zeta\right) \\
R_1\left(x,e\right) &= \sum_{i=0,j=0,m=0}^{k} g''\left(h_T\left(x,e\right)\right)\cdot\left(C_{[j,i]} + C_{[i,j]}\right)\cdot e_i e_j e_m \\
R_2\left(x,e\right) &= \sum_{i=0,j=0,m=0}^{k} g''\left(h_T\left(x,e\right)\right)\cdot\left(C_{[m,i]} + C_{[i,m]}\right)\left(x\right)\cdot e_i e_j e_m \\
R_3\left(x,e\right) &= \sum_{i=0,j=0,m=0}^{k} g'''\left(h_T\left(x,e\right)\right) \\
&\qquad \cdot\left(C_{[i]}\left(x\right) + \sum_{l=0}^{k}\left(C_{[l,i]} + C_{[i,l]}\right)\left(x\right)\cdot e_l\right)\cdot e_i e_j e_m \\
&\quad \zeta \in \left[-\bar{\epsilon},\bar{\epsilon}\right]^{k+1}
\end{aligned} \tag{4.38}$$

Bounding

$$
\begin{aligned}
|R_1| &\leq \sum_{i=0,j=0,m=0}^{k} \left| g''(h_T) \cdot \left(C_{[j,i]} + C_{[i,j]} \right) \cdot e_i e_j e_m \right| \\
&\leq \bar{\epsilon}^3 \cdot k \cdot \left| g''(h_T) \right| \cdot \sum_{i=0,j=0}^{k} \left| C_{[j,i]} + C_{[i,j]} \right| \leq \frac{\bar{\epsilon}^3}{2} \cdot M_a \\
|R_2| &\leq \frac{\bar{\epsilon}^3}{2} \cdot M_a \\
|R_3| &\leq \sum_{i=0,j=0,m=0}^{k} \left| g'''(h_T) \cdot \left(C_{[i]} + \sum_{l=0}^{k} \left(C_{[l,i]} + C_{[i,l]} \right) \cdot e_l \right) \cdot e_i e_j e_m \right| \\
&\leq \bar{\epsilon}^3 \cdot \left| g'''(h_T) \right| \cdot \sum_{i=0,j=0,m=0}^{k} \left| C_{[i]} \right| + \sum_{l=0}^{k} \left| \left(C_{[l,i]} + C_{[i,l]} \right) \cdot e_l \right| \\
&= \bar{\epsilon}^3 \cdot \left| g'''(h_T) \right| \cdot \left(k^2 \cdot \sum_{i=0}^{k} \left| C_{[i]} \right| + \bar{\epsilon} \cdot k^2 \cdot \sum_{i=0,l=0}^{k} \left| \left(C_{[l,i]} + C_{[i,l]} \right) \right| \right) \\
&\leq \bar{\epsilon}^3 \cdot M_b + \bar{\epsilon}^4 \cdot M_c \\
\left| \frac{R}{R_{C_{[0]}}} \right| &\leq \frac{\frac{1}{6} \cdot (|R_1| + |R_2| + |R_3|)}{R_{C_{[0]}}} \\
&\leq \frac{\bar{\epsilon}^3 \cdot M_a + \bar{\epsilon}^3 \cdot M_b + \bar{\epsilon}^4 \cdot M_c}{\bar{\epsilon}^2 \cdot M_a + \bar{\epsilon}^2 \cdot M_b + \bar{\epsilon}^3 \cdot M_c} = \bar{\epsilon}
\end{aligned}
\tag{4.39}
$$

□

To summarize, we have shown how to transform a wide range of correctly rounded operations to valid Taylor forms. Let (t, C) be the Taylor form for the floating-point model $\tilde{f_f}$. We can now rigorously bound the absolute round-off error using Equation (4.40).

$$
\begin{aligned}
\forall_x \exists_e \ |\delta_{f(x)}| &\leq \left| \sum_{i=0}^{k} \left(C_{[i]} \cdot e_i \right) + \sum_{i=0,j=0}^{k} \left(C_{[i,j]} \cdot e_i e_j \right) \right| \\
&\leq \bar{\epsilon} \cdot \sum_{i=0}^{k} \left| C_{[i]} \right| + \bar{\epsilon}^2 \cdot \sum_{i=0,j=0}^{k} \left| C_{[i,j]} \right|
\end{aligned}
\tag{4.40}
$$

Remark. For all bounds of remainders introduced, one may omit their addition within the matrices, given that the bounding constant is zero.

To finish up this section, we go through an example of converting a chain of operations to valid Taylor forms.

Example. Let $x \in [-10, 10]$ and $y \in [10, 20]$, we want to calculate the maximum absolute roundoff-error for $(x+1) \cdot y$ using Taylor forms. We obtain $\bar{\epsilon}^2 \cdot 400$ as bound for the multiplication remainder. This is because both M_3 and M_1 are zero and we only have to calculate $M_2 \geq \max_{x,y} |x \cdot y| + |x \cdot y|$.

$$\begin{aligned}
(x+1) \cdot y &\mapsto rd\left(rd\left(rd(x) + rd(1)\right) \cdot rd(y)\right) \\
rd(x) &\mapsto (x, [x][0]) \\
rd(1) &\mapsto (1, [][]) \\
(x, [x][]) + (1, [][]) &\mapsto (x+1, [x][0]) \\
&\overset{rd}{\mapsto} \left(x+1, \begin{bmatrix} x \\ x+1 \end{bmatrix} \begin{bmatrix} 0 & x \\ x & 0 \end{bmatrix}\right) \\
rd(y) &\mapsto (y, [y][0])
\end{aligned} \tag{4.41}$$

$$\left(x+1, \begin{bmatrix} x \\ x+1 \end{bmatrix} \begin{bmatrix} 0 & x \\ x & 0 \end{bmatrix}\right) \cdot (y, [y][0])$$

$$\mapsto \left(xy + y, \begin{bmatrix} xy \\ xy+y \\ xy+y \\ 400 \cdot \bar{\epsilon}^2 \end{bmatrix}, \begin{bmatrix} 0 & xy & xy \\ xy & 0 & xy+y \\ xy & xy+y & 0 \end{bmatrix}\right)$$

$$\overset{rd}{\mapsto} \left(xy + y, \begin{bmatrix} xy \\ xy+y \\ xy+y \\ 400 \cdot \bar{\epsilon}^2 \\ xy+y \\ 1240 \cdot \bar{\epsilon}^2 \end{bmatrix}, \begin{bmatrix} 0 & xy & xy & 0 & xy & 0 \\ xy & 0 & xy+y & 0 & xy+y & 0 \\ xy & xy+y & 0 & 0 & xy+y & 0 \\ 0 & 0 & 0 & 0 & 400 \cdot \bar{\epsilon}^2 & 0 \\ xy & xy+y & xy+y & 400 \cdot \bar{\epsilon}^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}\right)$$

Using Equation (4.41), we obtain the error bound given in Equation (4.2) for $(x+1) \cdot y$.

$$\begin{aligned}
|\delta| \leq &\left(|x \cdot y| + 3 \cdot |xy + y| + |+|1640 \cdot \bar{\epsilon}^2|\right) \cdot \bar{\epsilon} \\
&+ \left(|6 \cdot |x \cdot y| + 6 \cdot |xy + y| + 800 \cdot |\bar{\epsilon}^2|\right) \cdot \bar{\epsilon}^2
\end{aligned} \tag{4.42}$$

5. Implementation

In this chapter, the automatic analysis framework vodes, which was developed as a result of this thesis, is presented. The purpose of the project vodes is to provide a framework that allows for the computation of rigorous symbolic bounds of the absolute rounding error. It is intended to be a proof-of-concept for the theoretical framework provided in Chapter 4. Based on vodes, it should be easily achievable to extend the supported operations, optimize the error models and even include further analysis methods.

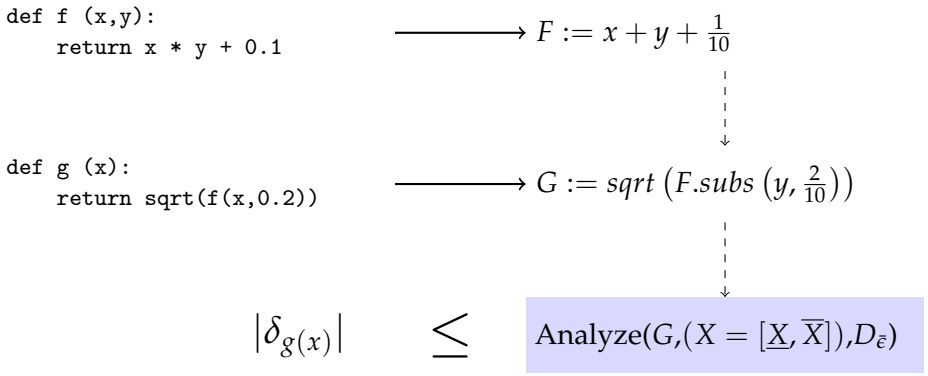


Figure 5.1.: An abstract overview of the automatic analysis framework vodes. The function `subs` replaces a free variable with a concise value. Note that 0.1 is being expressed using the quotient $\frac{1}{10}$.

We define the following governing requirements for vodes :

- **Problem Formalization** : It is required to provide the user with the ability to define the desired chain of computation using symbolic expressions.
- **Method Configuration** : The user needs to have the possibility to choose between the methods provided in Chapter 4. Furthermore, the user has to be able to configure the methods, if they provide configuration values.
- **Error Calculation** : Based on the provided symbolic expression, the selected method and its configuration, the user has to be able to retrieve a rigorous and symbolic bound for the absolute rounding error of floating-point calculation.
- **Visualization** : Because the main focus lies within the proof-of-concept, the user needs to have the ability to visualize the calculated errors from different methods and configurations to evaluate their accuracy.

- **Extensibility** : It must be possible for a developer to easily extend the functionality of vodes in terms of supported operations, error model and further analysis methods.

Based on these requirements, the next sections are dealing with the reasoning behind the used software dependencies, the general architecture of vodes and the concise implementation of the different analysis methods.

5.1. Software Basis

A step towards achieving extensibility for vodes is the usage of Python [28] as programming language, as it is not only one of the most used languages among developers [29], but also offers rich support for scientific computation. To allow the usage of symbolic expressions, the project relies heavily on the two existing Python libraries pymbolic [30] and SymPy [31]. SymPy is a Python library for symbolic computation providing a wide-ranging set of computer algebra functionalities. Features include the calculation of derivatives, basic arithmetic calculations and solving equations composed of equalities or inequalities within a given domain. While SymPy is incredibly feature-rich, it does not provide easy methods of (reliably) manipulating its expression tree. An essential problem of SymPy lies within its implicit simplification of expressions. A simple example is given in Equation (5.1).

$$x + 0.1 + 0.2 = x + 0.3 \tag{5.1}$$

While Equation (5.1) describes a correct algebraic equivalence, this simplification would result in incorrect computations for the absolute error because arithmetic and rounding operations are omitted. To support problem statements in the form of symbolic expressions, without alternating the calculated error bounds, vodes does not use SymPy but rather relies on pymbolic. This library separates structure from logic using symbolic expression trees that are only evaluated or compacted in any form using *mappers*. These mappers define their logic by the type of the currently observed expression. Let E be a symbolic expression with the type E_T and n children E_C being evaluated using very basic arithmetic operations, as defined in Equation (5.2).

$$f(E) := \begin{cases} E & E_T = \mathbb{R} \\ f(E_{C_0}) + \cdots + f(E_{C_{n-1}}) & E_T = + \\ f(E_{C_0}) \cdot \cdots \cdot f(E_{C_{n-1}}) & E_T = \cdot \end{cases} \tag{5.2}$$

The logic defined in Equation (5.2) can then easily be translated to a pymbolic mapper using the corresponding mapping functions. An exemplary implementation is given with the `BasicMapper` provided below.

```
1  from pymbolic.mapper import RecursiveMapper
2  class BasicMapper(RecursiveMapper):
3     def map_constant(self, expr):
4        return expr
5
6     def map_product(self, expr):
7        from pytools import product
8        return product(
9           self.rec(child) for child in expr.children
10       )
```

```
11
12    def map_sum(self, expr):
13        return sum(
14            self.rec(child) for child in expr.children
15        )
```

Based on these mappers, pymbolic created an easy and intuitive way for defining custom symbolic evaluation logic. They are extendable to custom expressions using custom mapping methods.

Example. Let $E = (2+5+8+9) \cdot 3 \cdot 10 \cdot 5 + 4$ be the expression that is to be evaluated by the `BasicMapper`. The symbolic expression tree of E is evaluated to 3604, as visualized in Figure 5.2.

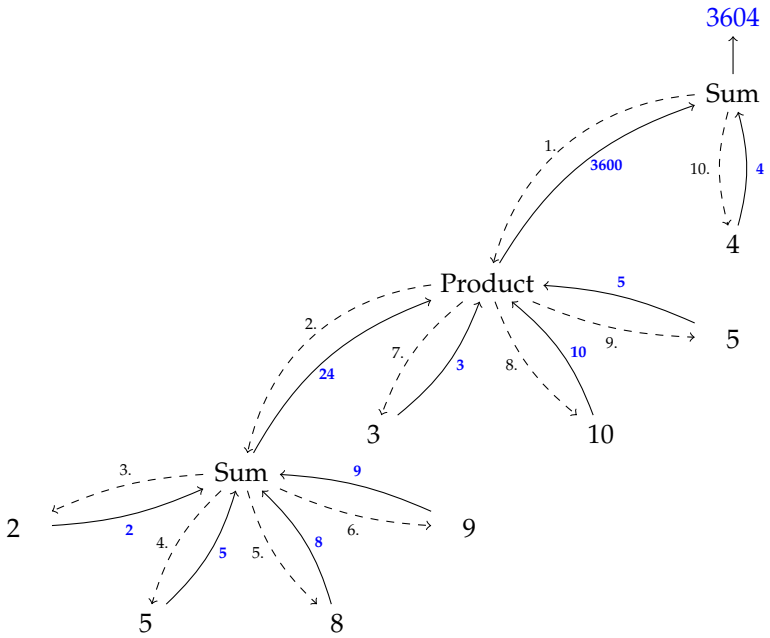


Figure 5.2.: Evaluation of a simple expression tree using a pymbolic mapper. The enumeration lists the order of the mapper's `self.rec(expr)` recursion calls with the blue numbers providing the result.

While pymbolic is used for the manipulation of symbolic expressions, SymPy is required to solve the inequalities given in Section 4.1 and allows for the computation of derivatives.

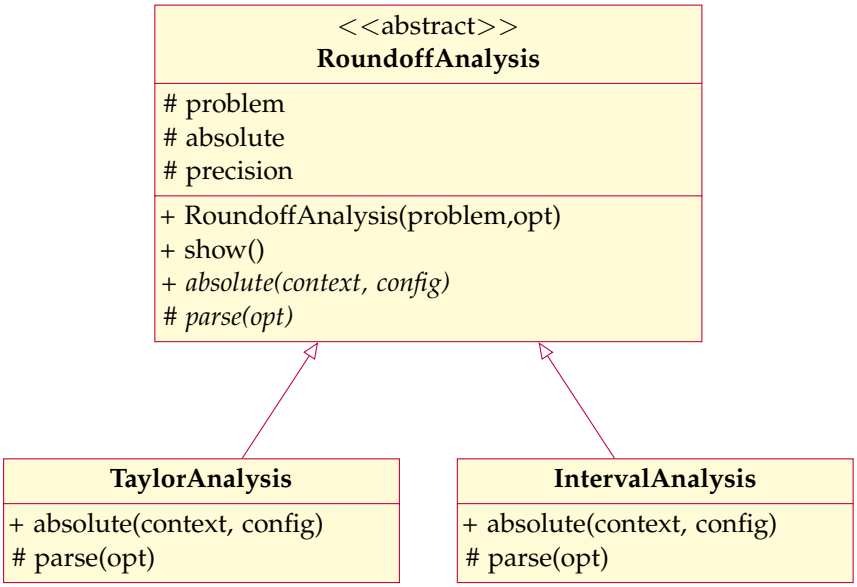


Figure 5.3.: Class diagram for the central analysis functionality.

5.2. Design

The essential component of the automatic analysis framework is the `RoundoffAnalysis` class. The user can choose between different analysis methods by instantiating the corresponding subclass. Additional to the formal problem statement provided as a pymbolic expression, the user may supply additional configuration parameters to the constructor. These are in turn read by the implementing class using its `parse` method. The `RoundoffAnalysis` class guarantees within its constructor to convert the initial problem statement to a binary problem statement using the custom `BinaryMapper`. This mapper ensures every operation to be either binary or unary and nests the operations, if necessary.

Example. We want to convert the expression $E = (2+5+8+9)\cdot 3\cdot 10\cdot 5+4$ to an equivalent expression with a binary expression tree. Using the `BinaryMapper`, we obtain the result $E_B = ((((((2+5)+8)+9)\cdot 3)\cdot 10)\cdot 5)+4$ visualized in Figure 5.4.

After instantiation, the user can calculate a bound for the absolute round-off error using the `absolute` method. Here, essential parameters for the analysis method are supplied. These include the minimum and maximum floating-point precision bits and therefore the domain of the machine precision variable. Additionally, the user can supply the minimum and maximum exponent bits, which could be used in future versions to detect over- and underflow. The user is then also presented with the possibility to visualize the calculated error bound using the `show` method. This architecture can be extended with new round-off analysis methods by inheriting from the `RoundoffAnalysis` class.

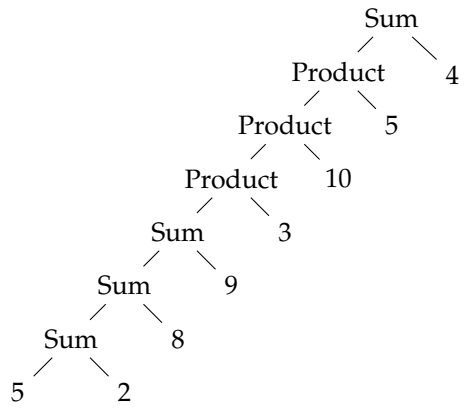


Figure 5.4.: An example conversion from the non-binary expression tree $E = (2 + 5 + 8 + 9) \cdot 3 \cdot 10 \cdot 5 + 4$ to a binary expression tree using vode's `BinaryMapper`.

5.3. Implementation

With the dependencies and the design of the analysis framework presented, we now take a look at the concise implementations of the analysis methods from Chapter 4. The Taylor analysis can be fully implemented within a single mapper, concisely the `TaylorMapper`, using the derivation rules presented within Section 4.2. The bounding constants for the remainder are obtained using the `ScalarEvaluator`, which is implementing the interval methods for constant expressions.

Remark. The usage of interval analysis for obtaining bounds on the remainder is a crucial optimization point for both accuracy and performance. Future versions should preferably switch to rigorous global optimizers or at least to interval libraries with greater performance.

The symbolic range analysis, on the other hand, is split into two parts. First, the `IntervalMapper` maps the sub-expressions to their floating-point model and represents them as an `Interval` expression. The expression $\left|f - \tilde{f}\right|$ is then evaluated using an implementation of the `IntervalEvaluator`. An essential concept of the evaluators shown in Figure 5.5 are their assumptions. Section 4.1 described the need to verify that an expression inherits certain properties and transform it, if necessary. As shown in Figure 5.6, the properties and translations implemented within this thesis mainly focus on the assertion of a polynomial expression for a given degree. This guarantees decidability for the solving of inequalities. The classes inheriting from the `IntervalEvaluator` define their assumptions for certain interval methods within the constructor. These are in turn checked by the `IntervalEvaluator` before calling the interval methods listed in Figure 5.5. This association is visualized by the following snippet from the `TaylorEvaluator` constructor.

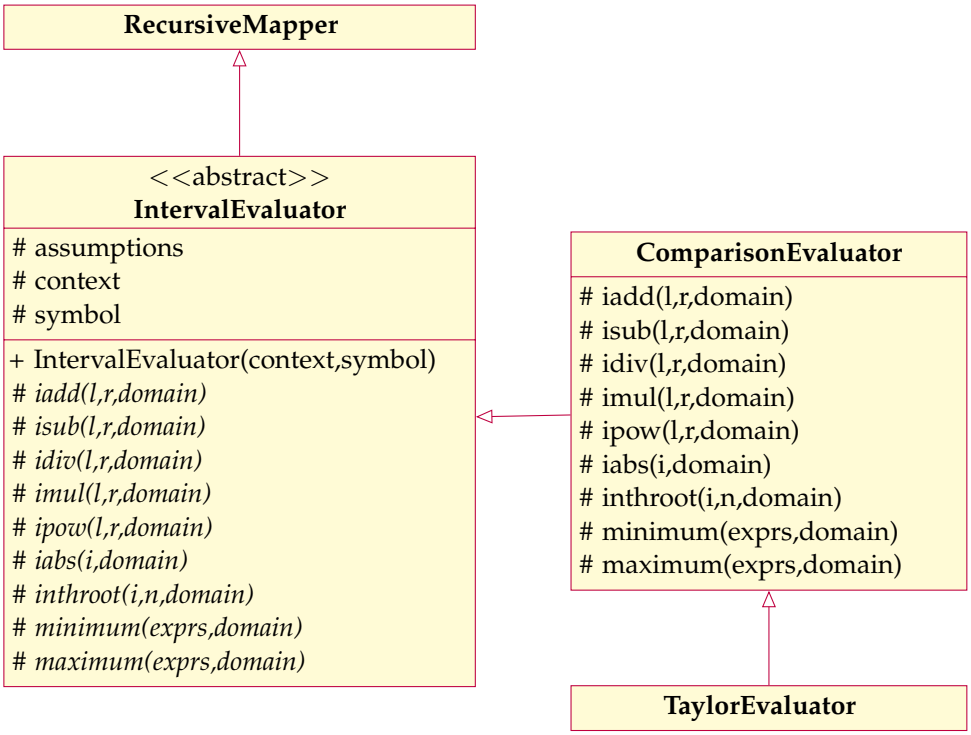


Figure 5.5.: Class diagram for the realization of symbolic interval evaluation. To simplify the diagram, the mapper methods (`map_sum...`) for all supported expressions are not explicitly mentioned. The `ScalarEvaluator` is not listed because it is used for constant expressions. Importantly, the domain limits the currently allowed values for the bound symbol.

```
self._assumptions["_minimum"] = [
   Assumption(
      property=IsPolynomial(n=2),
      translation=ToTaylor(n=2,limit=2**5)
   )
]
self._assumptions["_maximum"] = [
   Assumption(
      property=IsPolynomial(n=2),
      translation=ToTaylor(n=2,limit=2**5)
   )
]
```

It ensures that for the computation of the minimum or maximum, the expression is a polynomial of degree two or less, translating it to a Taylor polynomial using a Taylor expansion at 0, if necessary. Furthermore, real coefficients of the remainder bound are converted to rational coefficients with a maximum denominator of size 2^5.

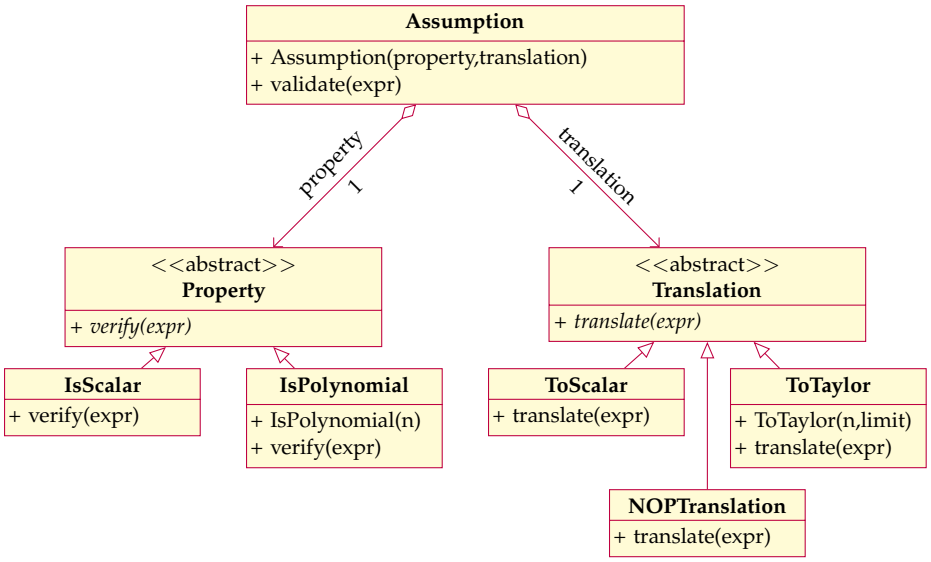


Figure 5.6.: Class diagram of the assumptions concept. The property is used for the verification of an expression. A failing expression is translated using the provided translation.

6. Experimental Results

In this chapter, we take a look at the achieved accuracy of the presented methods on the basis of empirical evaluation. To represent the bounds relative to existing methods, some of the most recent and most accurate libraries for the rigorous bounding of round-off errors are taken into consideration. Importantly, these methods do not allow for specifying the minimum and maximum precision to be analyzed, but rather adhere to a subset of the floating-point data types given in Table 3.1.

6.1. Tools

The libraries used for calculating comparative results are PRECiSA [32] and FPTaylor [24]) in the versions listed in Table 6.1. They can be seen as the current de facto standard within round-off error analysis as they achieve desirable accuracy [24] for a wide-ranging set of expressions. Additionally, they allow for formal verification of their results by emitting per-instance analysis certificates. These can be verified using external proof assistants. PRECiSA models a floating-point operation $\tilde{f}$ in terms of lemmas stating that the error from executing $\tilde{f}$ relative to f is bound by the calculated error approximation, under given conditions. Based on the theorem prover PVS and the floating-point formalization it is then possible to obtain rigorous error bounds. Furthermore, PRECiSA offers support for analyzing entire programs and automatic code generation for real-valued functions [32]. As described in Section 3.3, FPTaylor models floating-point programs using Taylor expansions. It incorporates a multitude of improvements to a very basic error model, like it is given in Definition 11, and combines them with rigorous global optimizers to allow for comparatively tight bounds. Importantly, it does not support loops or conditionals like PRECiSA and is therefore not developed to analyze complete floating-point programs [24]. Within the evaluation, we differentiate between two FPTaylor methods of modeling the rounding of a constant. FPTaylor (Improved) calculates $\frac{rnd(c)-c}{\bar{e}}$ using infinite precision rational arithmetic while FPTaylor denotes the usage of a more basic model. The concise configurations are listed in Table 6.2. We use the arbitrary precision floating-point arithmetic library mpmath [33] to calculate pseudo-exact solutions for the round-off error. Let f_p denote the calculation of f using p-Bits of precision. The pseudo-exact solution is obtained from $\left|f_{256} -_{256} f_p\right|$.

Remark. An important remark is to be given about the performance of the implementations provided within this thesis. It is **neither comparable** to FPTaylor nor PRECiSA. This is especially true for the symbolic range analysis, due to its dependency on function analysis.

The bad performance for the Taylor analysis is a result of the inefficient bounding of the remainders and the complete absence of expression compaction or sharing of noise variables.

Tool	Version
PRECiSA	master (commit 91e1e7543c5888ad5fb123d3462f71d085b99741)
FPTaylorJS	master (commit b0be0fa804a273177685113842b0356a8a345a1f)

Table 6.1.: Tools used within the Experiments

Configuration	Value
Precision of Results	20
Optimization	exact, 10000000, 0.01, 0.01, 0, 0.01, 0.0001
Advanced	optimized models
Improved Rounding Model	True : FPTaylor (Improved), False : FPTaylor

Table 6.2.: Configuration of FPTaylorJS

Name	Expression	Values
Point	$a := x_1 \cdot x_1 + x_2 - 11$ $b := x_1 + x_2 \cdot x_2 - 7$ $p := a \cdot a + b \cdot b$	$x_1 = 0.1$ $x_2 = 0.3$
Ranged		$x_1 \in [1, 5]$ $x_2 \in [1, 5]$

Table 6.3.: The two definitions of values for the himmilbeau [34] benchmark.

6.2. Benchmarks

Using the tools and techniques presented within Section 6.1, this section defines different kinds of benchmarks for evaluation and visualization.

Himmilbeau

We define the benchmark *himmilbeau* in Table 6.3 and use it as an indicator for the accuracy of bounding the round-off error for basic expressions. Importantly, we differentiate between a pointed and ranged variant. In the first case, the variables are limited to a single number, while the ranged variant constraints the real-variables within a domain. This is because the current implementation of the symbolic range analysis in vodes does not allow for the computation of the ranged variant. From the Figure 6.2 and Figure 6.3 we are able to deduce that the analysis methods within vodes are staying within close proximity of the error bounds calculated from FPTaylor and PRECiSA. However, while all tools obtain relatively accurate bounds, they are still noticeably pessimistic. It is to be noted, that the implementation of the method presented in Section 4.2 is able to obtain similar bounds to FPTaylor, as it can be seen in Figure 6.1, without using rigorous global optimizers and while relying on a very basic error model. This is an indication that an extension and refinement of the error model would result in superior accuracy.

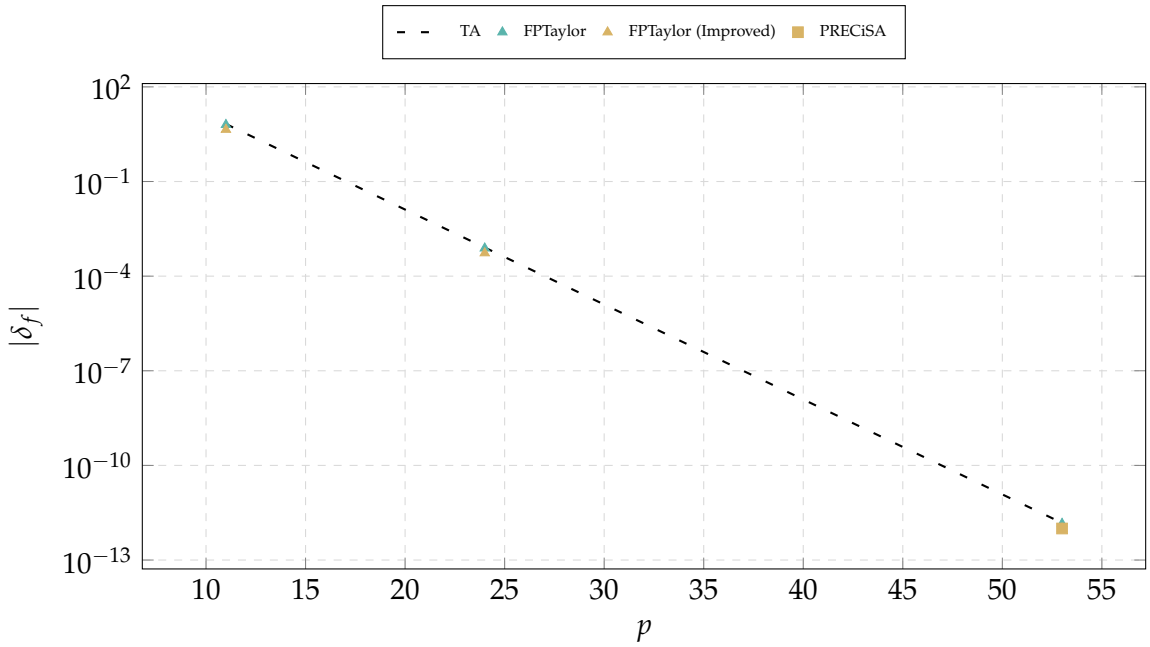


Figure 6.1.: Results of the comparative evaluation of the *himmilbeau* benchmark with the input variables being constrained to a domain. The symbolic range analysis is not contained, as it does not support constrained input variables.

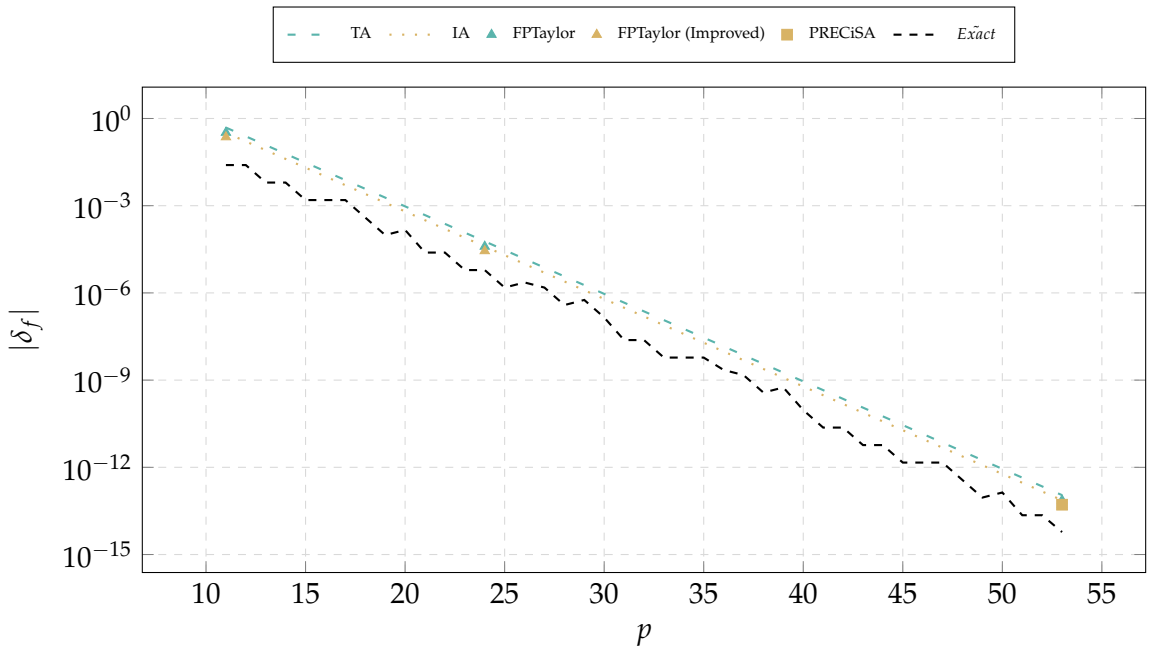


Figure 6.2.: Absolute error bounds for the *himmilbeau* benchmark. The analysis is performed for a floating-point precision between 11 and 53 and the result represented using a logarithmic scale.

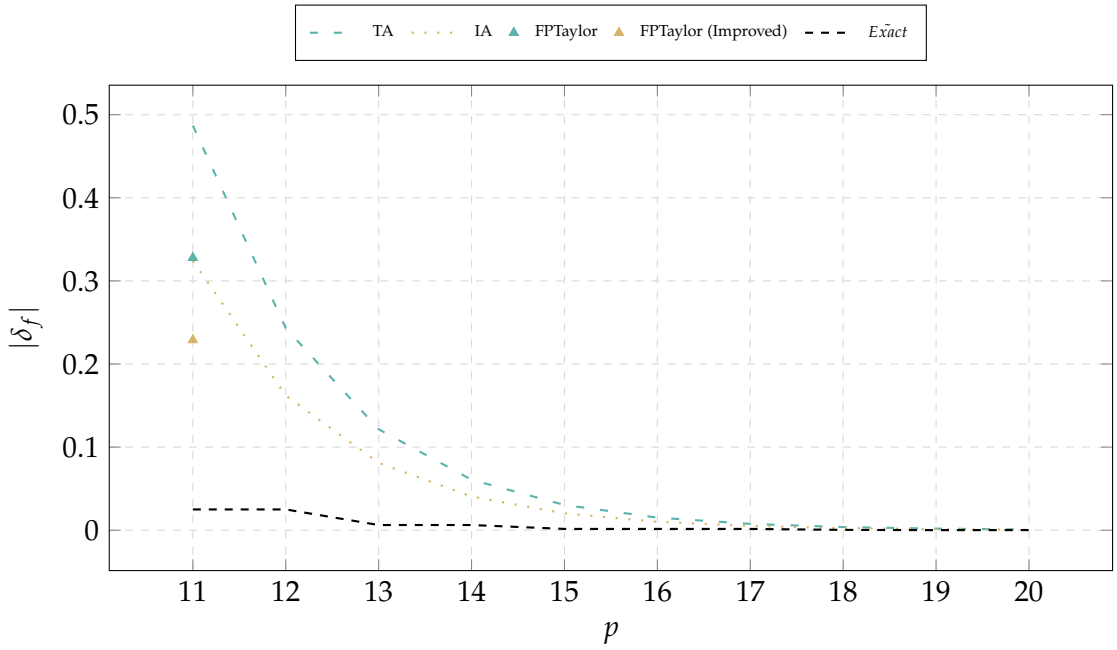


Figure 6.3.: The results visualized in Figure 6.2 are limited to floating-point precisions between 11 and 20 to allow for a linear scale.

Stabilizing Expressions

With the benchmarks listed in Table 6.4 we want to achieve two goals. First, we want to verify the calculated boundaries for the round-off error when comparing an unstable to a stable algorithm. We expect the tools to provide noticeably smaller bounds for the stable algorithms. The examples chosen are taken from [18] and can be summarized in replacing the subtraction to improve stability. Secondly, we want to give an exemplary indication for the impact of implementing unstable algorithms. Both Figure 6.4 and Figure 6.5 fulfill the expectations for the evaluated tools.

Name	Expression	Values
Stabilizing sqrt-Subtraction	$p_1 := \sqrt{x+1} - \sqrt{x}$ $p_2 := \frac{1}{\sqrt{x+1} + \sqrt{x}}$	$x = 127.7$
Stabilizing root-Finding	$p_1 := \sqrt{p^2+q} + p$ $p_2 := \frac{q}{\sqrt{p^2+q} + p}$	$p = 121.3$ $q = 0.99$

Table 6.4.: Benchmarks for verifying the tools within the process of expression stabilization.

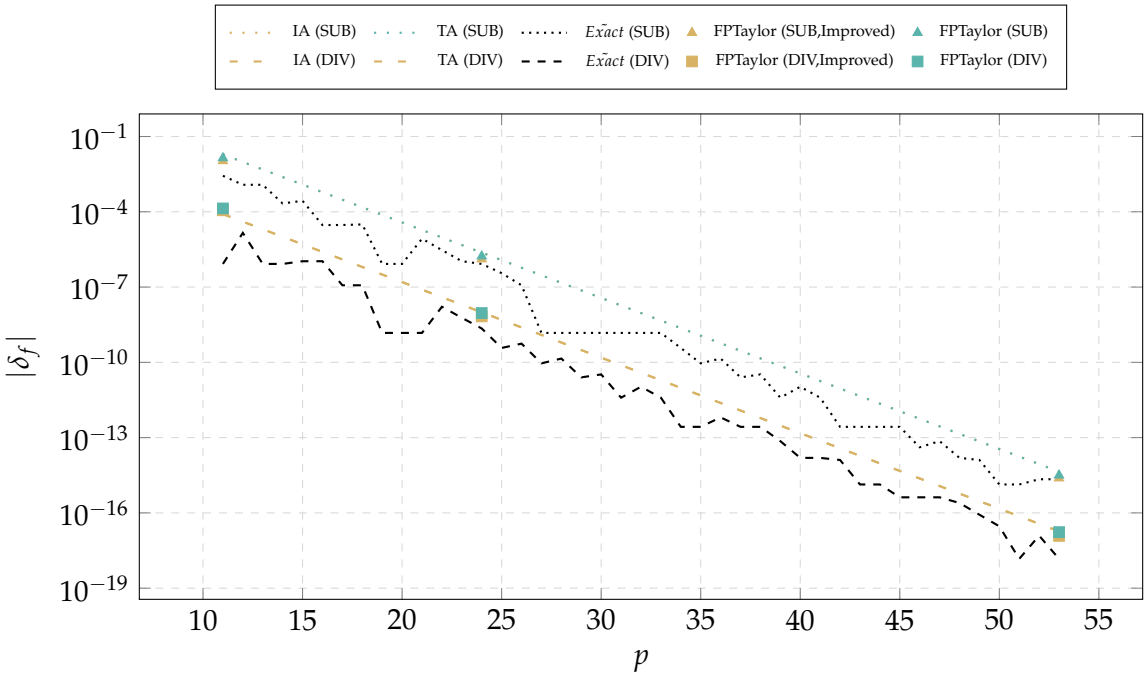


Figure 6.4.: Visualization of the error bounds for the *Stabilizing sqrt-Subtraction* benchmark from Table 6.4. The upper lines correspond to the unstable expression, noticeably distanced to its stable variant.

The bounds obtained by all the tools, but especially the improved FPTaylor version, approximate the exact error closely for most of the possible precision bits. Interestingly, the accuracy obtained for the lower precision bits is worse for FPTaylor when analyzing the stabilized algorithms.

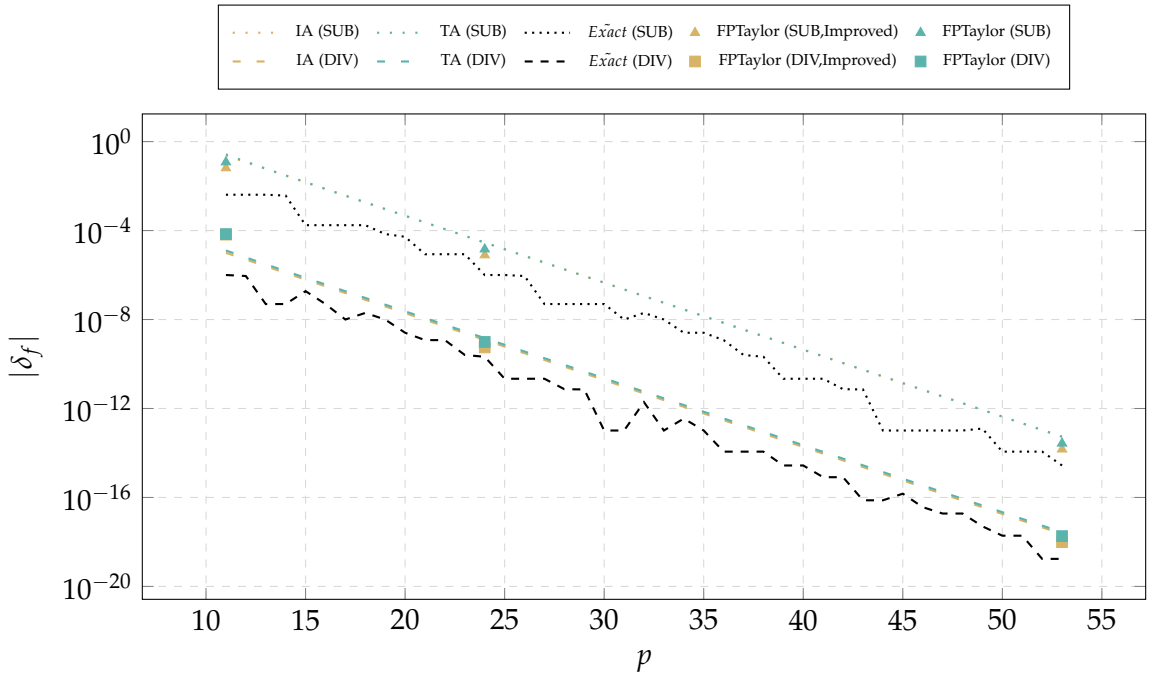


Figure 6.5.: The second stabilization benchmark from Table 6.4, *Stabilizing root-Finding*, also visualizes the accuracy gained from replacing the subtraction operation.

Solving ODEs

Finally, we want to compute bounds for the absolute roundoff-error of some very basic ODEs. The bounds are obtained for the final y_n at the end of the integration range. For the linear case, relatively close approximations of the round-off error are obtained, as can be seen in Figure 6.6. However, the non-linear case has prominent breakouts. This can be seen in Figure 6.7 at the precisions 18 and 23.

Name	Expression	Values
Linear RK-1	$f(y,t) := \frac{1}{3} \cdot y \cdot t$ $y_0 := \frac{1}{10}$	$h = \frac{1}{2}$ $n = 2$
Non-Linear RK-1	$f(y,t) := y^2 - t$ $y_0 := \frac{3}{9}$	$h = \frac{1}{2}$ $n = 2$

Table 6.5.: Basic examples for the numerical solving of an ODE using Runge-Kutta methods.

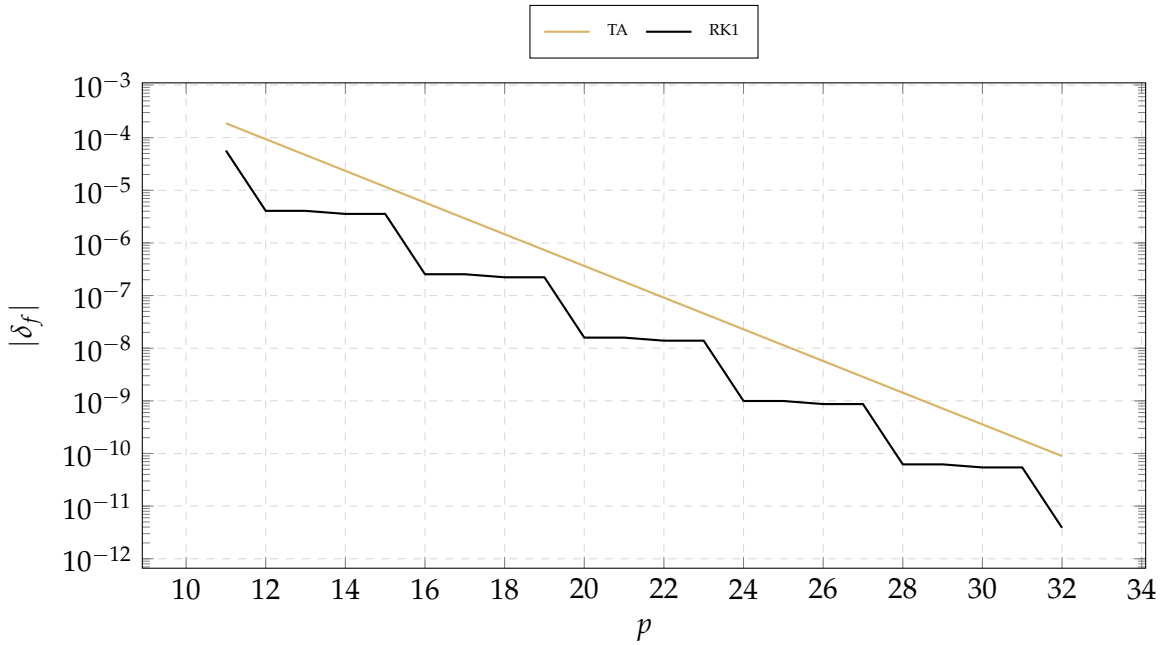


Figure 6.6.: Comparison between pseudo-exact and analyzed round-off error for a linear first order ODE being solved using the RK-1 method.

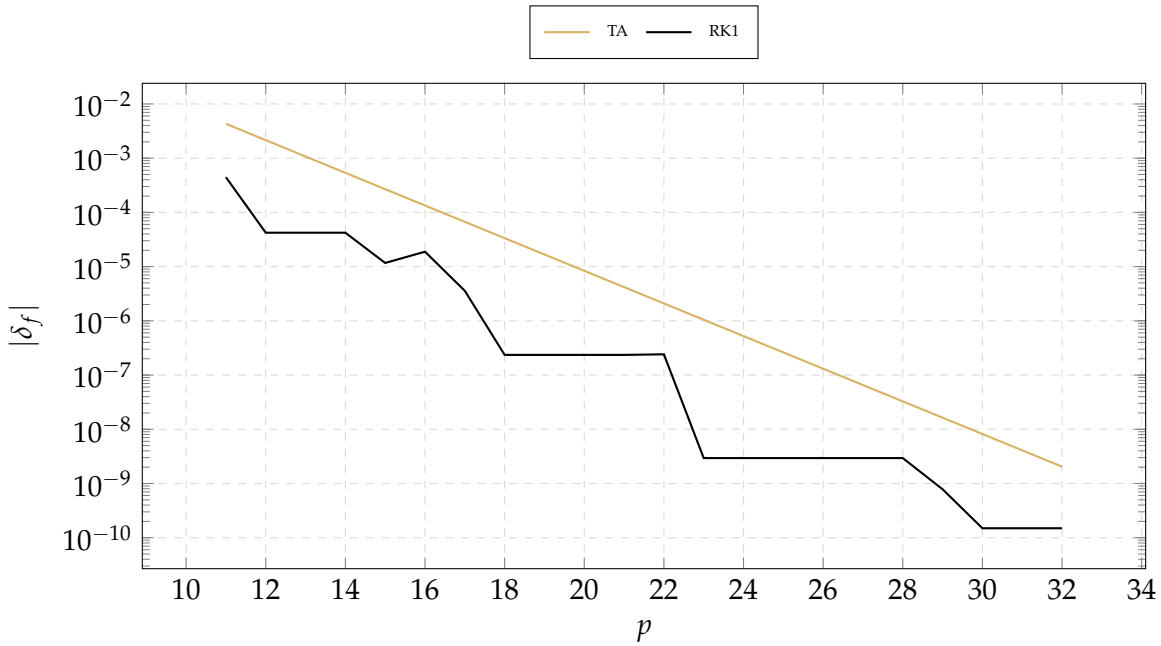


Figure 6.7.: Comparison between pseudo-exact and analyzed round-off error for a non-linear first order ODE being solved using the RK-1 method.

7. Conclusion

Within this thesis, we have provided the essential definitions for floating-point numbers and the errors introduced with their usage. We have proved the bounds for the relative rounding error, formalized it as $\bar{\epsilon}$ and used it as an essential concept within a very basic error model. Based on this model, we constructed two methods allowing us to obtain rigorous symbolic boundaries for the absolute error of calculating expressions using floating-point arithmetic. This focus contrasts currently available tools, as they limit themselves to obtaining constant bounds for a given precision. The first conceptualized method was the symbolic range analysis. It guarantees the inclusion of the exact absolute error by using the techniques of interval analysis [17] and extending them to support symbolic expressions. Importantly, rigorous approximations of the symbolic interval boundaries are required, as decidability could otherwise not be guaranteed when calculating the minimum and maximum. We then proceeded with extending the Taylor analysis, initially presented in [24], to support 2nd-order Taylor forms. A Taylor form is a rigorous approximation of a function using independent noise variables and is based on Taylor expansions with bound remainders. All the presented methods were then implemented using python in the project vodes allowing for an easy extensibility. From the empirical evaluations, we were able to deduce the relative tightness of the obtained bounds, ranging over a domain of possible precision, when compared with the currently most prominent and accurate analysis tools. While it was seldom possible to obtain better bounds in comparison, a major factor for this lies within the naive implementation and the basic error model. The remainders are bound overly pessimistic using interval analysis and the error model introduces unnecessary over-approximations. Furthermore, because we analyze a range of possible precisions the substitution of the noise variables with their possible value domain leads to further over-approximation. In regard to these limitations, the obtained results were very promising.

Future work should firstly focus on compiling all present error models within current literature ([24, 32],...) to one highly efficient and accurate model and extend it, if possible. All different analysis methods conceptualized and to be conceptualized can and should then be evaluated for their accuracy with this shared error model. One interesting, and seemingly the most promising, path to take for future analysis methods is the inclusion of further methods from approximation theory. Especially, as it provides the possibility of reformulating the definition of the absolute error [24] to reduce the computational complexity and increase accuracy. The implementation given within this thesis would greatly benefit in its accuracy from an advanced error model, improved methods of bounding the remainders, compaction of expression trees and sharing of noise variables for equivalent expressions and should in future provide proof certificates [24, 32]. It is then very likely, that its accuracy is going to surpass comparable analysis tools.

A. Additional Experimental Results

Table A.1 presents a detailed summary of the conducted comparative experiments.

Problem :	Point	Domain	Point	Domain	Point	Domain
Precision :	11		24		53	
IA	0.32427		$3.9573e-5$		$7.371e-14$	
TA	0.4868	6.6527	$5.9273e-5$	0.00080706	$1.1041e-13$	$1.5033e-12$
FPTaylor	0.32778	6.2794	$3.9445e-5$	0.00076294	$7.3471e-14$	$1.4211e-12$
FPTaylor (Imp.)	0.22885	4.4132	$2.7653e-5$	0.00053692	$5.1031e-14$	$1.0001e-12$
PRECiSA					$5.1747e-14$	
Exact	0.025		$6.1035e-6$		$5.9245e-15$	$1.0001e-12$

(a) Himmilbeau

Problem :	p_1	p_2	p_1	p_2	p_1	p_2
Precision :	11		24		53	
IA	0.019361	$8.089e-5$	$2.3627e-6$	$9.8666e-9$	$4.4009e-15$	$1.8378e-17$
TA	0.019387	$8.0975e-5$	$2.3627e-6$	$9.8667e-9$	$4.4009e-15$	$1.8378e-17$
FPTaylor	0.013867	0.00013663	$1.6911e-6$	$9.2127e-9$	$3.15e-15$	$1.716e-17$
FPTaylor (Imp.)	0.010588	0.00011433	$1.2923e-6$	$6.7635e-9$	$2.4066e-15$	$1.2108e-17$
PRECiSA						
Exact	0.0027152	$8.3595e-7$	$8.3595e-7$	$2.2371e-9$	$2.1942e-15$	$1.472e-18$

(b) Stabilizing sqrt-Subtraction p_1

Problem :	p_1	p_2	p_1	p_2	p_1	p_2
Precision :	11		24		53	
IA	0.23735	$9.9771e-6$	$2.8956e-5$	$1.2167e-9$	$5.3936e-14$	$2.2664e-18$
TA	0.26593	$1.2713e-5$	$2.8926e-5$	$1.4596e-9$	$5.3878e-14$	$2.7187e-18$
FPTaylor	0.11865	$6.9084e-5$	$1.4461e-5$	$9.8086e-10$	$2.6935e-14$	$1.827e-18$
FPTaylor (Imp.)	0.064265	$6.5622e-5$	$7.8403e-6$	$5.5639e-10$	$1.4604e-14$	$9.7735e-19$
PRECiSA						
Exact	0.0040807	$1.0033e-6$	$1.0033e-6$	$2.111e-10$	$2.739e-15$	$1.7308e-19$

(c) Stabilizing root-Finding p_1

Table A.1.: Round-off errors rounded to 5 significant digits for the different tools at shared evaluation points.

Bibliography

[1] M. Baboulin, A. Buttari, J. Dongarra, J. Kurzak, J. Langou, J. Langou, P. Luszczek, and S. Tomov, "Accelerating scientific computations with mixed precision algorithms," *Computer Physics Communications*, vol. 180, no. 12, pp. 2526–2533, Dec. 2009. DOI: 10.1016/j.cpc.2008.11.005.

[2] J. Sun, G. Peterson, and O. Storaasli, "High-performance mixed-precision linear solver for FPGAs," *IEEE Transactions on Computers*, vol. 57, no. 12, pp. 1614–1623, Dec. 2008. DOI: 10.1109/tc.2008.89.

[3] M. L. Gallo, A. Sebastian, R. Mathis, M. Manica, H. Giefers, T. Tuma, C. Bekas, A. Curioni, and E. Eleftheriou, "Mixed-precision in-memory computing," *Nature Electronics*, vol. 1, no. 4, pp. 246–253, Apr. 2018. DOI: 10.1038/s41928-018-0054-8.

[4] T. Carlyle, *On heroes, hero-worship, and the heroic in history*. Place of publication not identified: CreateSpace Independent Publishing Platform, 2011, ISBN: 9781461064619.

[5] Z. Fu, Z. Bai, and Z. Su, "Automated backward error analysis for numerical code," in *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, ACM, Oct. 2015. DOI: 10.1145/2814270.2814317.

[6] W.-F. Chiang, M. Baranowski, I. Briggs, A. Solovyev, G. Gopalakrishnan, and Z. Rakamarić, "Rigorous floating-point mixed-precision tuning," *ACM SIGPLAN Notices*, vol. 52, no. 1, pp. 300–315, May 2017. DOI: 10.1145/3093333.3009846.

[7] M. O. Lam, J. K. Hollingsworth, B. R. de Supinski, and M. P. Legendre, "Automatically adapting programs for mixed-precision floating-point computation," in *Proceedings of the 27th international ACM conference on International conference on supercomputing - ICS '13*, ACM Press, 2013. DOI: 10.1145/2464996.2465018.

[8] X. S. Li, J. W. Demmel, D. H. Bailey, G. Henry, Y. Hida, J. Iskandar, W. Kahan, S. Y. Kang, A. Kapur, M. C. Martin, B. J. Thompson, T. Tung, and D. J. Yoo, "Design, implementation and testing of extended and mixed precision BLAS," *ACM Transactions on Mathematical Software*, vol. 28, no. 2, pp. 152–205, Jun. 2002. DOI: 10.1145/567806.567808.

[9] M. D. Linderman, M. Ho, D. L. Dill, T. H. Meng, and G. P. Nolan, "Towards program optimization through automated analysis of numerical precision," in *Proceedings of the 8th annual IEEE/ ACM international symposium on Code generation and optimization - CGO '10*, ACM Press, 2010. DOI: 10.1145/1772954.1772987.

[10] C. Rubio-González, C. Nguyen, H. D. Nguyen, J. Demmel, W. Kahan, K. Sen, D. H. Bailey, C. Iancu, and D. Hough, "Precimonious," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ACM, Nov. 2013. DOI: 10.1145/2503210.2503296.

[11] *Solving Ordinary Differential Equations I*. Springer Berlin Heidelberg, 1993. DOI: 10.1007/978-3-540-78862-1.

[12] K. Gustafsson, "Control of error and convergence in ode solvers," PhD thesis, Department of Automatic Control, 1992.

[13] T. Huckle and S. Schneider, *Numerik für Informatiker*. Springer Berlin Heidelberg, 2002. DOI: 10.1007/978-3-662-09019-0.

[14] "Ieee standard for floating-point arithmetic," *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, 2019. DOI: 10.1109/IEEESTD.2019.8766229.

[15] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Computing Surveys*, vol. 23, no. 1, pp. 5–48, Mar. 1991. DOI: 10.1145/103162.103163.

[16] D. Monniaux, "The pitfalls of verifying floating-point computations," *ACM Transactions on Programming Languages and Systems*, vol. 30, no. 3, pp. 1–41, May 2008. DOI: 10.1145/1353445.1353446.

[17] R. E. Moore, R. B. Kearfott, and M. J. Cloud, *Introduction to Interval Analysis*. Society for Industrial and Applied Mathematics, Jan. 2009. DOI: 10.1137/1.9780898717716.

[18] A. Neumaier, *Introduction to Numerical Analysis*. Cambridge University Press, Sep. 2001. DOI: 10.1017/cbo9780511612916.

[19] L. H. de Figueiredo and J. Stolfi, "Affine arithmetic: Concepts and applications," *Numerical Algorithms*, vol. 37, no. 1-4, pp. 147–158, Dec. 2004. DOI: 10.1023/b:numa.0000049462.70970.b6.

[20] C. F. Fang, R. A. Rutenbar, M. Püschel, and T. Chen, "Toward efficient static analysis of finite-precision effects in DSP applications via affine arithmetic modeling," in *Proceedings of the 40th conference on Design automation - DAC '03*, ACM Press, 2003. DOI: 10.1145/775832.775960.

[21] F. Messine, "Extensions of affine arithmetic: Application to unconstrained global optimization," *J. UCS*, vol. 8, pp. 992–1015, Jan. 2002.

[22] C. Fang, T. Chen, and R. Rutenbar, "Floating-point error analysis based on affine arithmetic," in *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03).*, IEEE. DOI: 10.1109/icassp.2003.1202428.

[23] N. Revol, K. Makino, and M. Berz, "Taylor models and floating-point arithmetic: Proof that arithmetic operations are validated in COSY," *The Journal of Logic and Algebraic Programming*, vol. 64, no. 1, pp. 135–154, Jul. 2005. DOI: 10.1016/j.jlap.2004.07.008.

[24] A. Solovyev, M. S. Baranowski, I. Briggs, C. Jacobsen, Z. Rakamarić, and G. Gopalakrishnan, "Rigorous estimation of floating-point round-off errors with symbolic taylor expansions," *ACM Transactions on Programming Languages and Systems*, vol. 41, no. 1, pp. 1–39, Mar. 2019. DOI: 10.1145/3230733.

[25] D. Richardson, "Some undecidable problems involving elementary functions of a real variable," *Journal of Symbolic Logic*, vol. 33, no. 4, pp. 514–520, Jan. 1969. DOI: 10.2307/2271358.

[26] N. Bose and A. Modarressi, "General procedure for multivariable polynomial positivity test with control applications," *IEEE Transactions on Automatic Control*, vol. 21, no. 5, pp. 696–701, Oct. 1976. DOI: 10.1109/tac.1976.1101356.

[27] T. M. Apostol, *Calculus, Multi-Variable Calculus and Linear Algebra with Applications*. John Wiley and Sons, Jun. 1969, 698 pp., ISBN: 0471000078. [Online]. Available: https://www.ebook.de/de/product/3634937/t_m_apostol_calculus_multi_variable_calculus_and_linear_algebra_with_applications.html.

[28] G. VanRossum, *The Python language reference*. Hampton, NHRedwood City, Calif: Python Software FoundationSoHo Books, 2010, ISBN: 9781441412690.

[29] "Most used programming languages among developers worldwide, as of 2021," en, Aug. 2021, Retrieved September 09, 2021. [Online]. Available: https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/.

[30] "Pymbolic documentation," en, 2021, Retrieved September 09, 2021. [Online]. Available: https://documen.tician.de/pymbolic/index.html.

[31] "Sympy documentation," en, 2021, Retrieved September 09, 2021. [Online]. Available: https://www.sympy.org/en/index.html.

[32] M. Moscato, L. Titolo, A. Dutle, and C. A. Muñoz, "Automatic estimation of verified floating-point round-off errors via static analysis," in *Computer Safety, Reliability, and Security*, S. Tonetta, E. Schoitsch, and F. Bitsch, Eds., Cham: Springer International Publishing, 2017, pp. 213–229, ISBN: 978-3-319-66266-4.

[33] "Mpmath documentation," en, 2021, Retrieved September 10, 2021. [Online]. Available: https://mpmath.org/doc/1.2.0/.

[34] N. Damouche, M. Martel, P. Panchekha, J. Qiu, A. Sanchez-Stern, and Z. Tatlock, "Toward a standard benchmark format and suite for floating-point analysis," NSV'16, 2016.