Implementation of Taylor Models in CORA 2018

Matthias Althoff¹, Dmitry Grebenyuk², and Niklas Kochdumper¹

¹ Technische Universität München, Department of Informatics, Munich, Germany
althoff@in.tum.de
niklas.kochdumper@tum.de

² University of Melbourne, School of Computing and Information Systems, Melbourne, Australia
dgrebenyuk@student.unimelb.edu.au

Abstract

Tool Presentation: Computing guaranteed bounds of function outputs when their input
variables are bounded by intervals is an essential technique for many formal methods. Due
to the importance of bounding function outputs, several techniques have been proposed
for this problem, such as interval arithmetic, affine arithmetic, and Taylor models. While
all methods provide guaranteed bounds, it is typically unknown to a formal verification
tool which approach is best suitable for a given problem. For this reason, we present an
implementation of the aforementioned techniques in our MATLAB tool CORA so that
advantages and disadvantages of different techniques can be quickly explored without hav-
ing to compile code. In this work we present the implementation of Taylor models and
affine arithmetic; our interval arithmetic implementation has already been published. We
evaluate the performance of our implementation using a set of benchmarks against Flow*
and INTLAB. To the best of our knowledge, we have also evaluated for the first time how
a combination of interval arithmetic and Taylor models performs: our results indicate that
this combination is faster and more accurate than only using Taylor models.

1 Introduction

Providing guaranteed bounds of multi-dimensional functions \( r = f(x) \) (\( f : \mathbb{R}^n \rightarrow \mathbb{R}^m \)) subject
to inputs \( x \in [\underline{x}, \overline{x}], \underline{x} \leq \overline{x}, \underline{x}, \overline{x} \in \mathbb{R}^n \) bounded by a multi-dimensional interval is essential in
many formal verification techniques. For instance, in [45], the error of an approximate solution
of ordinary differential equations is obtained from function bounds. In [12,14,35,43], bounds
on functions are used to compute reachable sets via Picard iteration or a truncated Lie series in
combination with computing guaranteed bounds of functions. The remainder of Taylor series
is computed in [2,23,24] to abstract arbitrary differential equations to polynomial differential
equations for reachability analysis. The Jacobian of linearized systems is bounded in [20]
to compute reachable sets. Other approaches for reachability analysis use interval constraint
propagation [27,44]. A combination of Taylor models with SAT solving is presented in [19].
Function values are also bounded for verified runtime validation [36, Sec. 3.2]. Further tools
requiring bounds of functions can be found in [4,13,47].
Due to the importance of being able to bound values of arbitrary functions \( r = f(x) \), many techniques have been developed for this problem. Among them are interval arithmetic [26], generalized interval arithmetic [22], affine arithmetic [17], and Taylor models [9, 18]. Since generalized interval arithmetic and affine arithmetic are conceptually identical [17, Sec. 5], we will only consider affine arithmetic from now on. The most general technique for inputs bounded by intervals are Taylor models since interval arithmetic is a zeroth-order Taylor model and affine arithmetic is a first-order Taylor model when the inputs are bounded by intervals. In general, interval arithmetic is the fastest technique, but it typically results in the largest over-approximation. On the other end are Taylor models, which typically require the most computation time, but typically provide the tightest solutions. However, there exist rather many cases in which interval arithmetic provides tighter results [41]. Affine arithmetic is in most cases a good compromise between interval arithmetic (efficiency) and Taylor models (accuracy).

Taylor models cannot be directly evaluated as it is done in interval arithmetic or affine arithmetic. For this reason, we present the implementation of different techniques to obtain the bounds of Taylor models in CORA [3, 5]. Not only is it unclear for a given problem whether interval arithmetic, affine arithmetic, or Taylor models provide a tighter bound—it is also unknown what Taylor model evaluation technique is best for a given problem [41, Sec. 4]. This motivated us to introduce the method zoo, which evaluates several techniques in parallel and then intersects all results. While this method is quite trivial, we obtained superior results and can show that a mix of methods is faster and more precise than using Taylor models with high accuracy, which are slower and less precise in our tested examples. Our evaluation is performed on examples taken from [25] to remove bias from choosing the examples considered. We also compare our results with the established tools Flow* [12] for Taylor models and INTLAB [46] for interval arithmetic. Please note that Flow* has been primarily designed for reachability analysis and not for bounding function values. Another work that has compared interval arithmetic, affine arithmetic, and Taylor models is [38], but this work investigated whether other useful representations exist theoretically (e.g., a combination of trigonometric functions), without comparing the methods experimentally. A further work compared Taylor models with centered and mean value forms [32], but this work has evaluated the performance on rather small input uncertainties, while reachability analysis typically evaluates larger input uncertainties.

This paper is organized as follows: Sec. 2 defines Taylor models and how to perform operations on them. Affine arithmetic is presented in Sec. 3. In Sec. 4 we detail our implementation in CORA. We evaluate the implemented methods in Sec. 5 and also compare the performance with Flow* and INTLAB. Sec. 6 provides some final conclusions.

2 Taylor Models

In this section we briefly recall Taylor models [9, 31, 32, 35]. An earlier development of Taylor models, which was not introduced under that name can be found in [18]. Taylor models are implemented in the tools Flow* [11, 12], COSY INFINITY [34], and Ariadne [7, 8]; the implementation of Taylor models in Ariadne is additionally validated in Coq [15]. Other tools also use concepts from Taylor models, such as VNODE-LP [37].

To define Taylor models, we first introduce an \( n \)-dimensional interval \( [x] := [\underline{x}, \overline{x}], \forall i : \underline{x}_i \leq \overline{x}_i, \underline{x}, \overline{x} \in \mathbb{R}^n \) and define Taylor polynomials:
**Definition 1** (Taylor polynomial (see Sec. 3 in [40])). Let us first introduce the multi-index set

\[ \mathcal{L}^q = \left\{ (l_1, l_2, \ldots, l_n) \mid l_i \in \mathbb{N}, \sum_{i=1}^{n} l_i \leq q \right\}. \]

We define \( P^q(x - x_0) \) as the \( q \)-th order Taylor polynomial of \( f(x) \) around \( x_0 \) \((x, x_0 \in \mathbb{R}^n)\):

\[
P^q(x - x_0) = \sum_{l \in \mathcal{L}^q} \frac{(x_1 - x_{0,1})^{l_1} \cdots (x_n - x_{0,n})^{l_n}}{l_1! \cdots l_n!} \left( \frac{\partial^{l_1 + \cdots + l_n} f(x)}{\partial x_1^{l_1} \cdots \partial x_n^{l_n}} \right) \bigg|_{x = x_0}. \tag{1}
\]

**Definition 2** (Taylor model (see Def. 1 in [32])). Let \( f : \mathbb{R}^n \to \mathbb{R}^m \) be a function that is \((q + 1)\) times continuously differentiable in an open set containing the \( n \)-dimensional interval \([x] \). Given \( P^q(x - x_0) \) as the \( q \)-th order Taylor polynomial of \( f(x) \) around \( x_0 \in [x] \), we choose an \( n \)-dimensional interval \([I]\) such that

\[
\forall x \in [x] : \quad f(x) \in P^q(x - x_0) + [I]. \tag{2}
\]

The pair \( T = (P^q(x - x_0), I) \) is called an \( q \)-th order Taylor model of \( f(x) \) around \( x_0 \).

From now on we use the shorthand notation \((P, I)\), and we omit \( q \), \( x \), and \( x_0 \) when it is self-evident. Further information can be found in [31, Sec. 2]. An illustration of a fourth-order Taylor model is shown in Fig. 1 for \( r = \cos(x) \) and the range \([x] = [-\pi/3, \pi/2] \).

![Fourth-order Taylor model for cos(x) and [x] = [-π/3, π/2].](image)
2.1 Scalar Operations

Since Taylor models are essentially polynomials with an interval uncertainty, one requires interval arithmetic to perform operations on Taylor models. We briefly recall interval arithmetic and refer to [5] for information on its implementation in CORA. Binary operations typically required for scalar intervals are addition, subtraction, multiplication, and division:

\[
[x] + [y] = [x + y, \overline{x} + \overline{y}],
\]

\[
[x] - [y] = [x - y, \overline{x} - \overline{y}],
\]

\[
[x] \cdot [y] = \min([xy, x\overline{y}, x\overline{y}, \overline{x}y]), \max([xy, x\overline{y}, \overline{x}y, \overline{x}y]).
\]

\[
\frac{x}{y} = \begin{cases} 
  \emptyset & \text{if } y = [0, 0], \\
  \frac{[1/x, 1/y]}{[1, \overline{y}]} & \text{if } 0 \notin [y], \\
  \frac{[1, \overline{y}]}{[1, \overline{y}]} & \text{if } (y = 0) \land (\overline{y} > 0), \\
  \frac{(-1/y)}{(-1/y)} & \text{if } (y < 0) \land (\overline{y} = 0), \\
  \frac{(-1/\overline{y})}{(-1/\overline{y})} & \text{if } (y < 0) \land (\overline{y} > 0). 
\end{cases}
\]

Similarly, bounds on standard functions, such as \(\sin([x])\) can be obtained [5]. Interval arithmetic treats each variable as an independent entity, causing over-approximative results. For instance, the intervals in the expression \([x] - [x]\) are treated independently as

\[
[x] - [x] = \{x_1 \in [x]\} - \{x_2 \in [x]\} = [x - \overline{x}, \overline{x} - \overline{x}] \neq 0.
\]

This phenomenon is often referred to as the dependency problem [26, Sec. 2.2.3]. Taylor models are particularly designed to address the dependency problem. Due to the use of polynomials, dependent variables are considered by so-called cancellation effects [41, Sec. 12]. Let us first introduce the polynomials \(P_1\) and \(P_2\) (both of order \(q\)), the remaining part \(P_r\) of the polynomial \(P_1 \cdot P_2\) including terms of orders \(0\) to \(q\), and the cut-off part \(P_c\) of the polynomial \(P_1 \cdot P_2\) including terms of orders \(q + 1\) to \(2q\). We also require the operator

\[
B(P^q(x - x_0)) = \min_{x \in [x]} P^q(x - x_0), \max_{x \in [x]} P^q(x - x_0)
\]

returning an over-approximation of the exact bounds. Let us further introduce \(c_f = f(x_0)\) and \(\overline{T} = T - c_f\). Addition, subtraction, multiplication, and division are computed for \(T_1 = (P_1, I_1)\) and \(T_2 = (P_2, I_2)\), and constants \(\alpha, \beta \in \mathbb{R}\) as [32, Sec. 2]:

\[
\alpha T_1 + \beta T_2 = (\alpha P_1 + \beta P_2, \alpha I_1 + \beta I_2),
\]

\[
\alpha T_1 - \beta T_2 = (\alpha P_1 - \beta P_2, \alpha I_1 - \beta I_2),
\]

\[
T_1 \cdot T_2 = (P_r, B(P_1) + B(P_1) \cdot I_2 + B(P_2) \cdot I_1 + I_1 \cdot I_2),
\]

\[
\frac{T_1}{T_2} = T_1 \cdot \left(\frac{1}{T_2}\right),
\]

\[
\frac{1}{T} = \frac{1}{c_f} \left[1 - \frac{T}{c_f} + \frac{T^2}{c_f^2} \ldots + (-1)^q \frac{T^q}{c_f^q} \right] + (-1)^{q+1} \frac{B(\overline{T})^{q+1}}{c_f^{q+2}} \left(1 + \left[0, 1\right] \cdot \frac{B(\overline{T})}{c_f^{q+2}} \right)^{q+2}.
\]

Please note that \(1/T\) as well as \([1]/[y]\) is only defined if the division by zero is excluded. We have improved the computation of \(1/T\) as presented in Appendix B. It is now obvious that the previous example \([x] - [x] = 0\) returns the exact result due to cancellations of polynomials.
Unary operations, such as $e^x$, $\sin(x)$, etc. are handled similarly. A list of unary operations implemented in CORA can be found in Appendix A. Obviously, interval arithmetic is identical to Taylor models of zeroth order. It remains to compute $B(\cdot)$, which is addressed subsequently.

### 2.2 Conversion of an Interval to a Taylor Model

Converting an interval to a Taylor model is trivial [39, Sec. 2.4]. A Taylor model $T$ of an interval $[a]$ is

$$T = P(x) + I = 0.5(\overline{x} + \underline{x}) + 0.5(\overline{x} - \underline{x})x + [0, 0], \text{ where } x \in [-1, 1].$$

(11)

Using the above conversion with $x \in [-1, 1]$ has the effect that all operations result in Taylor models whose variables are also bounded by $[-1, 1]$ (ranges of variables are not affected by operations on them). This does not only simplify the implementation since ranges do not have to be stored, but also provides tighter over-approximations (see Sec. 2.3.1). The positive effects of a so-called centered form or mean-value form have been well studied, but mostly for polynomials of order 1 [1, 28]. From now on, we refer to Taylor models with $[x] = [-1, 1]$, where $1$ is a vector of ones of proper dimension, as normalized Taylor models.

### 2.3 Over-approximation of Bounds

In contrast to interval arithmetic and affine arithmetic, Taylor models do not directly provide a range of possible values. The bounds of a Taylor model $T$ with a polynomial $P$ and interval $[I]$ can be over-approximated as

$$B(T) = B(P) + [I]$$

using (5). Several approaches to obtain $B(P)$ exist, out of which interval arithmetic, branch and bound, and the LDB/QFB algorithm are currently implemented in CORA.

#### 2.3.1 Interval Arithmetic

Obviously, one can obtain the bounds of a polynomial by interval arithmetic, which is demonstrated for a polynomial $P(x, y)$ with two arguments:

$$B(P(x, y)) = \left\{ P(x, y) = \sum_{i=0}^{n} \sum_{j=0}^{n} c_{i,j}x^iy^j \mid x \in [\underline{x}, \overline{x}], y \in [\underline{y}, \overline{y}] \right\} \subseteq \sum_{i=0}^{n} \sum_{j=0}^{n} c_{i,j}[\underline{x}, \overline{x}]^i[\underline{y}, \overline{y}]^j.$$  

(12)

This approach works particularly well for monotone polynomials. To obtain better results, we compute the bounds from interval arithmetic using the normalized form as demonstrated by the next example.

**Example 1** (Evaluation of original and normalized Taylor model). Let us consider the function $r = 0.1x^3 - 0.5x^2 + 1$ and the range $[x] = [0, 6]$. The normalized function for $[\tilde{x}] = [-1, 1]$ is $\tilde{r} = 2.7\tilde{x}^3 + 3.6\tilde{x}^2 - 0.9\tilde{x} - 0.8$. Evaluating the original function and the normalized function with interval arithmetic results in

$$[r] = 0.1[0, 6]^3 - 0.5[0, 6]^2 + 1 = [-17.0, 22.6],$$

$$[\tilde{r}] = 2.7[-1, 1]^3 + 3.6[-1, 1]^2 - 0.9[-1, 1] - 0.8 = [-4.4, 6.4],$$

showing that the normalized form clearly outperforms the original form in this example.
2.3.2 Branch and Bound

For most polynomials, interval arithmetic provides too conservative results. Finding guaranteed bounds is the goal of global optimization techniques \cite{21}. Techniques which can efficiently obtain a global minimum or maximum, such as convex optimization or geometric programming, cannot be applied to general polynomials. Therefore, we have implemented a branch and bound algorithm \cite{6,30} in CORA.

Our implementation of a branch and bound algorithm is shown in Alg. 1, which evaluates each coordinate of the output $r = f(x)$ individually. In line 2-5 we first compute the bound using interval arithmetic. We use the short form $P([x]) = \{ P(x) | x \in [x] \}$; in Alg. 1 it is assumed that $P : \mathbb{R}^n \rightarrow \mathbb{R}$ due to evaluating each coordinate individually. We refine the computation by splitting the input intervals in sub-intervals $[x^{(k)}]$ resulting in corresponding sub-intervals of outputs $[r^{(k)}]$. The index $\bar{\alpha}$ of the input intervals $[x^{(\bar{\alpha})}]$ resulting in the output intervals $[r^{(\bar{\alpha})}]$ with the largest upper bound, and the index $\underline{\alpha}$ resulting in the smallest lower bound are determined in line 8 and line 9, respectively. If $\bar{\alpha} = \underline{\alpha}$ (which is typically only the case when $[x]$ has not yet been split), we split the coordinate of $[x^{(k)}]$ with the largest radius (line 11-16). Otherwise, the intervals $[x^{(\bar{\alpha})}]$ and $[x^{(\underline{\alpha})}]$ are each split in their respective coordinate with the largest radius (line 18-29). If the lower and upper bounds of the overall range $[r] \leftarrow \bigcup_i [r^{(i)}]$ (line 30) do not change more than $2\varepsilon \cdot \text{rad}([r])$ (line 31), where $\text{rad}([x]) := 0.5(x - \bar{x})$, the final range $[r]$ is returned. If we set $\varepsilon \rightarrow \infty$, the branch and bound technique becomes identical to interval arithmetic.

In the lines 15, 22, and 28, we re-expand the original Taylor model $P$ (using the operator $\text{reexp}(\cdot)$) at the midpoints of the newly generated sub-intervals before the bounds are calculated. This re-expansion leads to tighter bounds, but can also be computationally expensive, especially for Taylor models with high polynomial order and a large number of variables. Due to this trade-off between computation time and precision, CORA offers the two algorithm variants standard (without re-expansion) and advanced (with re-expansion).

**Example 2** (Branch and bound on a non-monotonic function). We consider the non-monotonic function $r = 0.1x^3 - 0.5x^2 + 1$. When applying Alg. 1, the upper and lower bounds are reduced in each iteration as shown in Fig. 2.

Figure 2: Improvements of bounding a polynomial using branch and bound for the first three iterations of Alg. 1. The improvements of $[r]$ over the iterations are $[-3.50, 5.50]$, $[-2.15, 5.05]$, $[-1.25, 1.91]$ (values are rounded). We have used the setting standard.
In addition to the branch and bound algorithm described in the previous subsection, we have also implemented an algorithm for bounding values combining the Linear Dominated Bounder (LDB) and the Quadratic Fast Bounder (QFB) presented in [33]. Similar to branch and bound, this algorithm is based on the iterative splitting of the input interval into smaller sub-intervals to determine the real minimum and maximum of a polynomial up to a user-defined precision $\epsilon$. Please note that due to the interval remainder part of the Taylor model, the difference between the real minimum and maximum of the function $r = f(x)$ and the calculated bounds can still be much larger than $\epsilon$. Our algorithm iteratively eliminates all sub-intervals that are guaranteed...
Implementation of Taylor models in CORA 2018 Althoff, Grebenyuk, and Kochdumper

to not contain the minimum and maximum of the polynomial, until only one sub-interval is left for the minimum, and one for the maximum. A linear and a quadratic approximation of the Taylor model is used to determine the sub-intervals that are most likely to contain the minimum and the maximum, which significantly speeds up the search.

2.4 Matrix Operations

This section addresses how CORA handles vectors and matrices represented by Taylor models, where a vector is defined as a matrix with a single row or column. A matrix Taylor model is a matrix in which each element is represented by a Taylor model. The operations of transpose, addition, subtraction, multiplication, and division by a scalar are defined in a standard way. The aforementioned operations can be broken down to addition and multiplication of scalar Taylor models, which are realized as presented in Sec. 2.1. All unary operations are realized element-wise as

\[ f(X)_{ij} = f(X_{ij}) \]

so that we again resort to the implementation described in Sec. 2.1.

One exception is the computation of the inverse of a matrix Taylor model. Although in principle, one could obtain the inverse of a matrix symbolically, e.g.,

\[
\begin{bmatrix}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{bmatrix},
\begin{bmatrix}
a_{22} & -a_{12} \\
-a_{21} & a_{11}
\end{bmatrix},
\]

this approach is infeasible for larger matrices. For this reason we use the approach in [49, Sec. V.B], which is based on the Sherman-Morrison (SM) formula (see [42, Sec. 2.7.1])

\[
(A + B)^{-1} = A^{-1} - \frac{1}{1 + \text{tr}(A^{-1}B)}(A^{-1}BA^{-1}),
\]

(13)

where \(\text{tr}(\cdot)\) returns the trace of a matrix. The formula holds when \(B\) has unitary rank. By splitting a Taylor model in a constant part and a remaining part, we obtain

\[
T = (P_{x - x_0} + I) = c_f + (P_{x - x_0} + I) = c_f + T,
\]

where \(P_{x - x_0} = P_{x - x_0} - c_f\) and \(T = T - c_f\). When \(T\) has unitary rank, we can compute

\[
(c_f + T)^{-1} = c_f^{-1} - \frac{1}{1 + \text{tr}(c_f^{-1}T)}(c_f^{-1}TC_f^{-1})
\]

(14)

using only methods from Sec. 2.1. However, in most cases, \(T\) has full rank so that [49, Sec. V.B] splits the matrix \(T\) into a sum of rank-one matrices \(T = \sum_{i=1}^n T_i\), where \(T_i\) is a matrix whose \(i^{th}\) column coincides with \(T\) and all other entries are zero. One first applies (14) for \(T_1\) and \(c_f\). Next, \((c_f + T_1 + T_2)^{-1}\) is calculated by choosing \(A = c_f + T_1\) and \(B = T_2\) in (13); the inverse of \(A\) is available from the previous step. This is repeated until the final matrix \(T_n\) has been considered.

3 Affine Arithmetic

Affine arithmetic uses affine forms, i.e., first-order polynomials consisting of a vector \(x \in \mathbb{R}^n\) and noise symbols \(\epsilon_i \in [-1, 1]\) (see e.g., [17]):

\[
\hat{x} = x_0 + \epsilon_1 x_1 + \epsilon_2 x_2 + \ldots + \epsilon_p x_p.
\]
The possible values of $\hat{x}$ lie within a zonotope [29]. In contrast to Taylor models, it is possible that $p > n$ so that affine forms are not a special case of Taylor models. This is the same reason why polynomial zonotopes are different from Taylor models, since polynomial zonotopes are a generalization of zonotopes [2].

However, in this work, we only consider intervals as inputs and outputs so that Taylor models of first order can implement an affine arithmetic. Obviously, other methods for realizing an affine arithmetic exist, e.g., Chebyshev (minimax) approximations or the min-range approximation [48, Sec. 3].

4 Implementation in CORA

Taylor models are implemented in CORA by the class `taylm`. To make use of cancellation effects, we have to provide names for variables in order to recognize identical variables; this is different from implementations of interval arithmetic, where each variable is treated individually as demonstrated in (4). We have realized three primal ways to generate a matrix containing Taylor models.

**Method 1: Composition from scalar Taylor models.** The first possibility is to generate scalar Taylor models from intervals as shown subsequently.

```matlab
1 a1 = interval([-1, 2]); % generate a scalar interval [-1,2]
2 b1 = taylm(a1, 6); % generate a scalar Taylor model of order 6
3 a2 = interval([2, 3]); % generate a scalar interval [2,3]
4 b2 = taylm(a2, 6); % generate a scalar Taylor model of order 6
5 c = [b1; b2] % generate a row of Taylor models
```

When a scalar Taylor model is generated from a scalar interval, the name of the variable is deduced from the name of the interval. If one wishes to overwrite the name of a variable `a2` to `c`, one can use the command `taylm(a2, 6, {'c'})`.

**Method 2: Converting an interval matrix.** One can also first generate an interval matrix, i.e., a matrix containing intervals, and then convert the interval matrix into a Taylor model. The subsequent example generates the same Taylor model as in the previous example.

```matlab
1 a = interval([-1;2], [2;3]); % generate an interval vector [[-1,2]; [2,3]]
2 c = taylm(a, 6, {'a1'; 'a2'}) % generate a Taylor model (order 6) with variable names a1 and a2
```

Note that the cell for naming variables `{'a1'; 'a2'}` has to have the same dimensions as the interval matrix `a`. If no names are provided, default names are automatically generated.

**Method 3: Symbolic expressions.** We also provide the possibility to create a Taylor model from a symbolic expression.

```matlab
1 syms a1 a2; % instantiate symbolic variables
2 s = [2 + 1.5*a1; 2.75 + 0.25*a2]; % create symbolic function
3 c = taylm(s, interval([-2;-3],[0;1]), 6) % generate Taylor model
```
This method does not require naming variables since variable names are taken from the variable names of the symbolic expression. The interval of possible values has to be specified after the symbolic expression $s$; here: $[-2,0] [-3,1]^T$.

All examples generate a row vector $c$ as shown in Sec. 2.2. Since all variables are normalized to the range $[-1,1]$ according to (11), we obtain

$$c = \begin{bmatrix} 0.5 + 1.5 \cdot \tilde{a}_1 + [0,0] \\ 2.5 + 0.5 \cdot \tilde{a}_2 + [0,0] \end{bmatrix}.$$ 

The following workspace output of MATLAB demonstrates how the dependency problem is considered by keeping track of all encountered variables:

```matlab
>> c(1) + c(1)
ans =
  1.0 + 3.0*a1 + [0.00000,0.00000]
>> c(1) + c(2)
ans =
  3.0 + 1.5*a1 + 0.5*a2 + [0.00000,0.00000]
```

### 4.1 Implementation of Polynomials

Competitive runtimes of Taylor model implementations can only be realized by efficient addition and multiplication of polynomials, since these two base operations are repeatedly used as shown in Appendix A. To this end, we use a concept similar to the MONOMIAL Toolbox\(^1\). A monomial is defined as a product of powers of variables with nonnegative integer exponents, e.g., $x_1^2x_2^3x_3^4$. Let us define $d$ as the number of monomial terms and $n$ as the number of variables. We encode polynomials as tuples with three entries $P(x) = \{c, E, id\}$, where $c \in \mathbb{R}^d$ is a vector storing the coefficients for each monomial term of the polynomial, $E \in \mathbb{R}^{n \times d}$ is a matrix that stores the exponents of the monomials, and $id \in S^n$ is a list of strings $S$ that stores the identifiers for all variables that are part of the polynomial. Let us consider the following polynomial $p(x)$ that consists of $d = 4$ monomial terms and has $n = 3$ variables as a demonstrating example:

$$p(x) = 1.3 + 3.4x_1^2x_2 - 2.3x_1^3x_4 + 4.5x_1^3x_2^2x_4^3. \quad (15)$$

This polynomial is represented in CORA as:

$$P(x) = \{c, E, id\}, \quad c = \begin{bmatrix} 1.3 \\ 3.4 \\ -2.3 \\ 4.5 \end{bmatrix}, \quad E = \begin{bmatrix} 0 & 2 & 3 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 2 & 3 \end{bmatrix}, \quad id = \begin{bmatrix} x_1^n \\ x_2^n \end{bmatrix} \begin{bmatrix} x_4^n \end{bmatrix}. \quad (16)$$

Before two polynomials can be added or multiplied, we have to ensure that the identification vectors of both polynomials are identical. The first step is therefore to determine a common identification vector $id_{new}$ from the vectors $id_1$ and $id_2$ of the two polynomials. After this, the exponent matrices $E_1$ and $E_2$ have to be transformed to this new common representation,\(^1\)MONOMIAL is a MATLAB toolbox for multivariate polynomials. The toolbox is available at https://people.sc.fsu.edu/~jburkardt/m_src/monomial/monomial.html

\(^{1}\)MONOMIAL is a MATLAB toolbox for multivariate polynomials. The toolbox is available at https://people.sc.fsu.edu/~jburkardt/m_src/monomial/monomial.html
which can be easily realized by commutation of matrix rows and insertion of new all-zero rows. We demonstrate this concept using two polynomials $P_1(x)$ and $P_2(x)$:

$$P_1(x) = \{c_1, E_1, id_1\}, \quad \text{with } c_1 = \begin{bmatrix} 2.3 \\ 3.1 \\ -2.7 \end{bmatrix}, \quad E_1 = \begin{bmatrix} 0 & 1 & 4 \\ 1 & 2 & 2 \end{bmatrix}, \quad id_1 = \begin{bmatrix} ^n x_2^3 \\ ^n x_5^n \end{bmatrix}. \quad (17)$$

$$P_2(x) = \{c_2, E_2, id_2\}, \quad \text{with } c_2 = \begin{bmatrix} -1.1 \\ 5.2 \end{bmatrix}, \quad E_2 = \begin{bmatrix} 0 & 3 \end{bmatrix}, \quad id_2 = \begin{bmatrix} ^n x_3^n \end{bmatrix}.$$

The transformation of these two polynomials to a common representation yields

$$P_1(x) = \{c_1, E_1, id_1\}, \quad \text{with } c_1 = \begin{bmatrix} 2.3 \\ 3.1 \\ -2.7 \end{bmatrix}, \quad E_1 = \begin{bmatrix} 0 & 1 & 4 \\ 0 & 0 & 0 \\ 1 & 2 & 2 \end{bmatrix}, \quad id_1 = \begin{bmatrix} ^n x_2^n \\ ^n x_3^n \\ ^n x_5^n \end{bmatrix}. \quad (18)$$

$$P_2(x) = \{c_2, E_2, id_2\}, \quad \text{with } c_2 = \begin{bmatrix} -1.1 \\ 5.2 \end{bmatrix}, \quad E_2 = \begin{bmatrix} 0 & 0 \\ 0 & 3 \\ 0 & 0 \end{bmatrix}, \quad id_2 = \begin{bmatrix} ^n x_2^n \\ ^n x_3^n \\ ^n x_5^n \end{bmatrix}.$$

After the polynomials have been transformed to a common representation, addition is realized with a simple concatenation $\| \|$ of the coefficient vectors $c_{\text{new}} = c_1 \| c_2$ and the exponent matrices $E_{\text{new}} = E_1 \| E_2$. The representation of the resulting polynomial $P_{\text{new}}(x)$ possibly contains redundant information, because there could be multiple monomials with identical exponents. In order to obtain a compact representation, these monomials have to be combined to a single monomial by adding their respective coefficients. In order to find all monomials that have identical exponents, we first sort the columns of the exponent according to the graded lexicographic order [16, Chap. 2]. After that, we only have to compare neighboring columns in the sorted exponent matrix to determine all monomials with identical exponents. Note that the sorting of the exponent matrix can be costly, since the number of variables $n$ in the Taylor model can be large. To speed up the computations, we pre-sort the matrix $E_{\text{new}}$ according to a scalar hash value $h(e_{\text{new}}^{(i)})$, where $e_{\text{new}}^{(i)}$ is the i-th matrix column of $E_{\text{new}}$ and $h(\cdot)$ is a hash function. By using this pre-sorting, only columns with identical hash values have to be further sorted according to the entries in the matrix rows. Note that this concept implements a bucket sort algorithm [10], where each bucket contains elements with identical hash values. In order to get a low number of hash function collisions for matrix columns that are not identical, we choose the hash function $h(e_{\text{new}}^{(i)}) = [1, 1, \ldots, 1] e_{\text{new}}^{(i)}$ for our Taylor model implementation, which is identical to the sum of entries in the column $e_{\text{new}}^{(i)}$.

The multiplication of two polynomials can be implemented similar to addition, where all $d_{\text{new}} = d_1 \cdot d_2$ monomial terms of the resulting polynomial have to be generated by multiplication of the respective coefficients in combination with an addition of the corresponding exponent matrix columns. Afterwards, all monomials with identical exponents have to be combined in order to obtain a compact representation, as described above.
4.2 List of Functions of the Class \texttt{taylm}

In this subsection we list the functions realized in CORA. Since CORA is implemented in MATLAB, the function names are chosen such that they overload the built-in MATLAB functions. Tab. 1 lists the implemented mathematical functions and Tab. 2 shows auxiliary functions, which are required to e.g., specify Taylor models, arrange them in matrices, access values, perform set operations, and display results, among others. Examples of using some of the listed functions in MATLAB are provided in Appendix C.

<table>
<thead>
<tr>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acos</td>
<td>arccos(·) function as defined in (31)</td>
</tr>
<tr>
<td>asin</td>
<td>arcsin(·) function as defined in (30)</td>
</tr>
<tr>
<td>atan</td>
<td>arctan(·) function as defined in (32)</td>
</tr>
<tr>
<td>cos</td>
<td>cos(·) function as defined in (25)</td>
</tr>
<tr>
<td>cosh</td>
<td>cosh(·)-function as defined in (28)</td>
</tr>
<tr>
<td>det</td>
<td>determinant of a Taylor model matrix</td>
</tr>
<tr>
<td>exp</td>
<td>exponential function as defined in (21)</td>
</tr>
<tr>
<td>interval</td>
<td>various implementations of the bound operator $B(\cdot)$ as presented in Sec. 2.3</td>
</tr>
<tr>
<td>log</td>
<td>natural logarithm function as defined in (22)</td>
</tr>
<tr>
<td>minus</td>
<td>overloaded ‘-’ operator, see (7)</td>
</tr>
<tr>
<td>mpower</td>
<td>overloaded ‘^’ operator (power)</td>
</tr>
<tr>
<td>mrdivide</td>
<td>overloaded ‘/’ operator (division), see (9)</td>
</tr>
<tr>
<td>mtimes</td>
<td>overloaded ‘*’ operator (multiplication), see (8) for scalars and Sec. 2.4 for matrices</td>
</tr>
<tr>
<td>plus</td>
<td>overloaded ‘+’ operator (addition), see (6) for scalars and Sec. 2.4 for matrices</td>
</tr>
<tr>
<td>power</td>
<td>overloaded ‘.^’ operator (elementwise power)</td>
</tr>
<tr>
<td>rdivide</td>
<td>overloads the ‘./’ operator; provides elementwise division of two matrices</td>
</tr>
<tr>
<td>reexpand</td>
<td>re-expand the Taylor model at a new expansion point</td>
</tr>
<tr>
<td>sin</td>
<td>sin(·) function as defined in (24)</td>
</tr>
<tr>
<td>sinh</td>
<td>sinh(·) function as defined in (27)</td>
</tr>
<tr>
<td>sqrt</td>
<td>$\sqrt{\cdot}$ function as defined in (23)</td>
</tr>
<tr>
<td>tan</td>
<td>tan(·) function as defined in (26)</td>
</tr>
<tr>
<td>tanh</td>
<td>tanh(·) function as defined in (29)</td>
</tr>
<tr>
<td>times</td>
<td>overloaded ‘.*’ operator for elementwise multiplication of matrices</td>
</tr>
<tr>
<td>trace</td>
<td>trace of a Taylor model matrix</td>
</tr>
<tr>
<td>uminus</td>
<td>overloaded ‘-’ operator for a single operand</td>
</tr>
<tr>
<td>uplus</td>
<td>overloaded ‘+’ operator for a single operand</td>
</tr>
</tbody>
</table>

4.3 Additional Parameters for the Class \texttt{taylm}

CORA’s Taylor model implementation contains some additional parameters which can be modified by the user:

- **max_order**: Maximum polynomial degree of the monomials in the polynomial part of the Taylor model. Monomials with a degree larger than \texttt{max_order} are bounded with the bounding operator $B(\cdot)$ and added to the interval remainder. Further, $q = \texttt{max_order}$ is used for the implementation of the formulas listed in Appendix A.
### Table 2: Auxiliary methods of the class `taylm`.

<table>
<thead>
<tr>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>display</td>
<td>displays the values of a <code>taylm</code> object in the MATLAB workspace</td>
</tr>
<tr>
<td>getCoef</td>
<td>returns the array of polynomial coefficients of a <code>taylm</code> object</td>
</tr>
<tr>
<td>getRem</td>
<td>returns the interval part of a <code>taylm</code> object</td>
</tr>
<tr>
<td>getSyms</td>
<td>returns the polynomial part of a <code>taylm</code> object as a symbolic expression</td>
</tr>
<tr>
<td>optBnb</td>
<td>implementation of the branch and bound algorithm as presented in Sec. 2.3.2</td>
</tr>
<tr>
<td>optBnbAdv</td>
<td>implementation of the advanced branch and bound algorithm as presented in Sec. 2.3.2</td>
</tr>
<tr>
<td>optLinQuad</td>
<td>implementation of the algorithm based on LDB and QFB as presented in Sec. 2.3.3</td>
</tr>
<tr>
<td>horzcat</td>
<td>overloads the operator for horizontal concatenation, e.g., <code>a = [b, c, d]</code></td>
</tr>
<tr>
<td>set</td>
<td>set the additional class parameters (see Sec. 4.3)</td>
</tr>
<tr>
<td>setName</td>
<td>set the names of the variables in <code>taylm</code></td>
</tr>
<tr>
<td>subsasgn</td>
<td>overloads the operator that assigns elements of a <code>taylm</code> matrix <code>I</code>, e.g., <code>I(1,2) = value</code>, where the element of the first row and second column is set</td>
</tr>
<tr>
<td>subsref</td>
<td>overloads the operator that selects elements of a <code>taylm</code> matrix <code>I</code>, e.g., <code>value = I(1,2)</code>, where the element of the first row and second column is read</td>
</tr>
<tr>
<td>taylm</td>
<td>constructor of the <code>taylm</code> class</td>
</tr>
<tr>
<td>vertcat</td>
<td>overloads the operator for vertical concatenation, e.g., <code>a = [b; c; d]</code></td>
</tr>
</tbody>
</table>

- **tolerance**: Minimum absolute value of the monomial coefficients in the polynomial part of the Taylor model. Monomials with a coefficient whose absolute value is smaller than `tolerance` are bounded with the bounding operator $B(\cdot)$ and added to the interval remainder.

- **eps**: Termination tolerance $\epsilon$ for the branch and bound algorithm from Sec. 2.3.2 and for the algorithm based on the Linear Dominated Bounder and the Quadratic Fast Bounder from Sec. 2.3.3.

These parameters are stored as properties of the class `taylm`. In the functions `plus`, `minus`, and `times`, two Taylor models are combined to one resulting Taylor model object using the rules

\[
\begin{align*}
\text{max_order}\_\text{new} &= \max(\text{max_order}\_1, \text{max_order}\_2), \\
\text{tolerance}\_\text{new} &= \min(\text{tolerance}\_1, \text{tolerance}\_2), \\
\text{eps}\_\text{new} &= \min(\text{eps}\_1, \text{eps}\_2),
\end{align*}
\]

where the subscript `new` refers to the resulting object, and 1 and 2 to the initial objects.

### 4.4 Class `zoo`

As later shown in Sec. 5, it is often better to use several simple range bounding methods in parallel and intersect the results. For instance, it is advantageous for most range bounding problems to combine interval arithmetic with Taylor models of order 3 rather than only applying one accurate method, such as using Taylor models of order 6. On average this strategy is more accurate and faster to compute.

To facilitate mixing different techniques, we have created the class `zoo` in which one can specify the methods to be combined. All methods implemented for Taylor models in Tab. 1 and Tab. 2 are also available for the class `zoo`.

For the initialization of the `zoo` object, the different techniques that should be used for range bounding can be specified as a list of strings. Currently, the following techniques are available:
Implementation of Taylor models in CORA 2018
Althoff, Grebenyuk, and Kochdumper

- **interval**: Interval arithmetic.
- **affine(int)**: Affine arithmetic; the bounds $B(\cdot)$ are calculated with interval arithmetic.
- **affine(bnb)**: Affine arithmetic; the bounds $B(\cdot)$ are calculated with the branch and bound algorithm (see Sec. 2.3.2).
- **affine(bnbAdv)**: Affine arithmetic; the bounds $B(\cdot)$ are calculated with the advanced branch and bound algorithm (see Sec. 2.3.2).
- **taylm(int)**: Taylor models; the bounds $B(\cdot)$ are calculated with interval arithmetic.
- **taylm(bnb)**: Taylor models; the bounds $B(\cdot)$ are calculated with the branch and bound algorithm (see Sec. 2.3.2).
- **taylm(bnbAdv)**: Taylor models; the bounds $B(\cdot)$ are calculated with the advanced branch and bound algorithm (see Sec. 2.3.2).
- **taylm(linQuad)**: Taylor models; the bounds $B(\cdot)$ are calculated with the algorithm that is based on LDB and QFB (see Sec. 2.3.3).

In addition to the bounding techniques, the additional parameters from Sec. 4.3 as well as the names of the variables can be specified as optional input arguments. The bounds of the **zoo** object can be computed with the **interval** operator, which intersects the intervals obtained by all specified techniques.

```matlab
1 int = interval(1, 2); % generate a scalar interval [1,2]
2 methods = {'interval', 'taylm(bnb)', 'affine(int)'}; % range bounding techniques
3 max_order = 10; % max. polynomial order of the Taylor models
4 epsilon = 0.001; % termination tolerance for the branch and bound algorithm
5 tolerance = 1e-8; % minimum absolute value for monomial coefficients
6 z = zoo(int, methods, 'x', max_order, epsilon, tolerance); % generate the zoo-object
7 s_z = sin(z); % calculate the sine of the zoo-object
8 r = interval(s_z) % calculate the bounds
```

The above code produces the following output:

```
int = interval(1, 2); % generate a scalar interval [1,2]
methods = {'interval', 'taylm(bnb)', 'affine(int)'}; % range bounding techniques
max_order = 10; % max. polynomial order of the Taylor models
epsilon = 0.001; % termination tolerance for the branch and bound algorithm
tolerance = 1e-8; % minimum absolute value for monomial coefficients
z = zoo(int, methods, 'x', max_order, epsilon, tolerance); % generate the zoo-object
s_z = sin(z); % calculate the sine of the zoo-object
r = interval(s_z) % calculate the bounds
```

```
[0.84147, 1.00000]
```

### 5 Numerical Experiments

In this section, we compare two aspects: a) the performance of Taylor models implemented in CORA with other tools and b) the accuracy of Taylor model computations with interval arithmetic, affine arithmetic, and a combination of techniques. We first start with a few motivating examples which highlight advantages and disadvantages of the investigated techniques.
5.1 Motivating Examples

We first motivate the study of complementary techniques by presenting examples for which different techniques provide the best result.

Example 3 (Example in favor of interval arithmetic). Let us obtain the bound \( r = \sin([x]) \), \([x] = [0, \frac{\pi}{2}]\).

**Interval arithmetic:** Since \( \sin(x) \) is monotone on \([0, \frac{\pi}{2}]\), interval arithmetic returns (see [5, eq. (12)])

\[ r = [\sin(0), \sin\left(\frac{\pi}{2}\right)] = [0, 1], \]

which is the exact result.

**Affine arithmetic:** The affine form of \([x]\) is \([x] = \frac{\pi}{4} + \epsilon\frac{\pi}{4}\). To use (24), we first require \( x_0 = 0.5(\pi + x) = \frac{\pi}{4} \), \( c_f = x_0 = \frac{\pi}{4} \), and \( \tilde{T} = x - c_f \). Using (24) we obtain

\[
[r] = \sin(c_f) + \cos(c_f)\tilde{T} - \frac{1}{2}B(\tilde{T}^2)\sin(c_f + [0, 1]B(\tilde{T}))
= \sin(\pi/4) + \cos(\pi/4)(x - \pi/4)
- \frac{1}{2}B((x - \pi/4)^2)\sin(\pi/4 + [0, 1]B(x - \pi/4))
= \sin(\pi/4) - \cos(\pi/4)\pi/4 + \cos(\pi/4)x
- \frac{1}{2}([0, \pi/2] - \pi/4)^2\sin(\pi/4 + [0, 1]([0, \pi/2] - \pi/4))
= 0.1517 + 0.7071x - [0.00000, \frac{\pi^2}{32}]
= [-0.1567, 1.2624].
\]

**Taylor model:** We use a Taylor model of third order. Due to the length of the computation, we do not present it and only provide the output interval using interval arithmetic (see Sec. 2.3.1): \([r] = [-0.1234, 1.3354]\).

Example 4 (Example where affine arithmetic is better than a Taylor model). Typically, Taylor models provide better results than affine arithmetic, but for the example \( r = \sqrt{x^3 - x} \) with \([x] = [2, 3]\) affine arithmetic is better. However, interval arithmetic provides the best result.

**Interval arithmetic:** After inserting the intervals, interval arithmetic returns (see [5, eq. (5)])

\[ r = \sqrt{[2, 3]^3 - [2, 3]} = \sqrt{[5, 25]} = [\sqrt{5}, 5] = [2.2361, 5.0]. \]

**Affine arithmetic:** The affine form of \([x]\) is \([x] = 2.5 + \epsilon 0.5\). Using (23), we obtain

\[
[r] = \sqrt{(2.5 + \epsilon 0.5)^3 - (2.5 + \epsilon 0.5)}
= \sqrt{8.875\epsilon + 13.125 + [0, 2]}
= 1.2249\epsilon + 3.6228 + [-1.68727, 0.78250]
= [0.71071, 5.63021].
\]
Taylor model: We use a Taylor model of third order; the Taylor model of \([x]\) is \(T = 2.5 + 0.5x + [0, 0]\). When using (23) and interval arithmetic to bound the polynomial (see Sec. 2.3.1), the result is

\[
[r] = \sqrt{T^3 - T} \\
= \sqrt{0.125x^3 + 1.875x^2 + 8.875x + 13.125} \\
= -0.00023256x^3 + 0.51714x^2 + 1.875x + 13.125 = (-3.86179, 1.86704) \\
= [-1.46404, 6.76670].
\]

Example 5 (Example in favor of Taylor models). We use the same example as in example 2: \(r = 0.1x^3 - 0.5x^2 + 1\) and \([x] = [0, 6]\).

Interval arithmetic: After inserting the intervals, interval arithmetic returns (see [5, eq. (9)])

\[
[r] = 0.1[0, 6]^3 - 0.5[0, 6]^2 + 1 = 0.1[0, 216] - 0.5[0, 36] + 1 = [-17.0, 22.6].
\]

Affine arithmetic: The affine form of \([x]\) is \([x] = 3 + 3\). Since power is realized by multiple times applying multiplication, we only require (6), (7), and (8):

\[
[r] = 0.1(3 + 3)^3 - 0.5(3 + 3)^2 + 1 \\
= -0.9x - 0.8 + [-4.5, 10.8] \\
= [-6.2, 10.9].
\]

Taylor model: We use a Taylor model of third order. The Taylor model of \([x]\) is \(T = 3 + 3x + [0, 0]\). When using interval arithmetic to bound the polynomial (see Sec. 2.3.1), we obtain

\[
[r] = 0.1T^3 - 0.5T^2 + 1 \\
= 0.1(27x^3 + 81x^2 + 81x + 27) - 0.5(9x^2 + 18x + 9) + 1 \\
= 2.7x^3 + 3.6x^2 - 0.9x - 0.8 \\
= [-4.4, 6.4].
\]

Next, we compare our results with Flow* and INTLAB.

5.2 Comparison with Flow* and INTLAB

In this subsection, we compare our implementation of Taylor models with Flow* [12] and our implementation of affine arithmetic in Sec. 3 with INTLAB [46]. Flow* and INTLAB both consider rounding errors of floating-point numbers. Since CORA has been designed primarily for prototyping new methods for reachability analysis and for other set-based techniques, CORA is currently not considering floating-point rounding errors.

All computations are performed on an Intel Core i7-7820HQ processor running at 2.9GHz. In order to reduce any possible bias, we have chosen the benchmarks from another work on relative error bounds for floating-point arithmetic [25]. For our results, the relative precision of an interval \(x\) is obtained as

\[
\left[\begin{array}{c}
\delta_1 \\
\delta_2
\end{array}\right] = \left[\begin{array}{c}
x - x_{ref} \\
x_{ref} - x_{ref}
\end{array}\right] \cdot 100%,
\]

(20)
where \( x_{\text{ref}} \) is a reference value. This measure of the precision is more informative than the scalar overestimation factor used in [32, Eq. 5.1], because it evaluates the precision of the lower and the upper range bounds separately. Another possibility would be to focus on the remainder of the Taylor models; however, by comparing the obtained function bounds, we also evaluate how well the polynomial part can be bounded. We used the real bounds of the function for the reference interval \( x_{\text{ref}} \). This ensures that \( x_{\text{ref}} \subseteq x \), so that the relative precision \( \delta \geq 0 \). Large values for \( \delta \) therefore correspond to a large over-approximation, and a value of \( \delta = 0 \) means that the calculated bound is identical to the real bound. We determined the real bounds for all benchmark models with a precision of \( 10^{-6} \) using the verified global optimization approach from [33]. If the computation of the bounds took longer than 10 seconds we terminated it. These cases are marked with the symbol "−" in Tab. 3 and Tab. 4.

For our Taylor model and affine arithmetic implementation, we compare the results for bounding the polynomial parts with interval arithmetic (Interval subs.) as described in Sec. 2.3.1, branch and bound (BnB) as well as advanced branch and bound (BnB adv.) as described in Sec. 2.3.2 and Alg. 1 (with \( \epsilon = 0.001 \)), and our algorithm based on LDB and QFB (LDB/QFB) as described in Sec. 2.3.3 (with \( \epsilon = 0.001 \)).

The results from the comparison of our Taylor model implementation with Flow* in speed and accuracy are summarized in Tab. 3, Tab. 4, and Tab. 5. We also show the results obtained by using a zoo object with the range bounding techniques interval and taylm(int) (see Sec. 4.4). For each benchmark model, we compare the results computed with different maximal polynomial orders. Since Flow* is written in C++, the implementation is significantly faster for all functions compared to CORA. For most benchmark models the precision of Flow* is slightly worse than the precision of Taylor models with interval substitution in CORA. An exception to this are the last two benchmark models, for which the use of Flow* results in large over-approximations. Since these two benchmark models are the only ones that contain fractions, this is most probably caused by a large over-approximation from the Lagrange remainder of the inverse, as it is described in Appendix B. Flow* is not primarily designed to use Taylor models alone, but for computing reachable sets using Taylor models. We are grateful to Xin Chen for providing us with a standalone Taylor model library comprising basic mathematical operations.

For the different bounding techniques implemented in CORA the numerical results are as expected: The more sophisticated algorithms LDB/QFB, and branch and bound in general, have a higher precision than interval substitution; in return, however, they lead to significantly larger execution times.

In Tab. 6 and Tab. 7 we list the results from our comparison with INTLAB. In addition to the results for affine arithmetic, the results calculated with CORA’s interval arithmetic are displayed, too. Since INTLAB considers rounding errors, it is slower than CORA’s interval arithmetic implementation. The numerical results show further that the precision of INTLAB’s affine arithmetic is better than the one for affine arithmetic in CORA. A possible reason for this is that affine arithmetic in CORA is implemented with Taylor models that have a polynomial order of one, which seems to not fully exploit all advantages of the affine arithmetic approach.

Contrary to the Taylor model implementation, using the branch and bound algorithm for affine arithmetic does not improve the precision. The LDB/QFB algorithm cannot be applied for affine arithmetic, since it is based on a separation of the Taylor model monomials into linear and higher-order monomials. Since affine arithmetic objects are linear by definition, they do not contain any higher-order monomials, which makes the application of the algorithm impossible.
Table 3: Comparison of Taylor model techniques (part 1). The precision is provided in percent compared to the exact result as shown in (20). The input ranges are presented in Tab. 8 and the examples are taken out of [25].

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Max. order</th>
<th>CORA</th>
<th>BnB</th>
<th>LDB/QFB</th>
<th>Zoo (Int. subs.)</th>
<th>Flow*</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Interval subs.</td>
<td>BnB (adv.)</td>
<td>BnB subs. BnB</td>
<td>LDB/QFB</td>
<td>Zoo (Int. subs.)</td>
</tr>
<tr>
<td>sin</td>
<td>2</td>
<td>[140, 148]</td>
<td>[140, 148]</td>
<td>[86.7, 148]</td>
<td>[82.0, 148]</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>sin</td>
<td>5</td>
<td>[166, 146]</td>
<td>[41.5, 8.23]</td>
<td>[6.29, 3.55]</td>
<td>[5.92, 3.54]</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>sin</td>
<td>10</td>
<td>[162, 146]</td>
<td>[35.1, 3.87]</td>
<td>[0.40, 0.03]</td>
<td>[0.01, 0.01]</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>sin</td>
<td>2</td>
<td>[27.4, 0]</td>
<td>[27.4, 0]</td>
<td>[1.33, 0]</td>
<td>[1.00, 0]</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>bspline0</td>
<td>5</td>
<td>[0, 30.9]</td>
<td>[0, 30.9]</td>
<td>[0, 3.61]</td>
<td>[0, 2.86]</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>bspline0</td>
<td>10</td>
<td>[0, 30.9]</td>
<td>[0, 30.9]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>bspline1</td>
<td>2</td>
<td>[33.8, 0]</td>
<td>[33.8, 0]</td>
<td>[6.77, 0]</td>
<td>[5.08, 0]</td>
<td>[3.92, 0]</td>
</tr>
<tr>
<td>bspline1</td>
<td>5</td>
<td>[33.8, 0]</td>
<td>[33.8, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>bspline1</td>
<td>10</td>
<td>[33.8, 0]</td>
<td>[33.8, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[3.92, 0]</td>
</tr>
<tr>
<td>bspline2</td>
<td>2</td>
<td>[34.9, 0]</td>
<td>[34.9, 0]</td>
<td>[8.40, 0]</td>
<td>[6.41, 0]</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>bspline2</td>
<td>5</td>
<td>[34.9, 0]</td>
<td>[34.9, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>bspline2</td>
<td>10</td>
<td>[34.9, 0]</td>
<td>[34.9, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>doppler</td>
<td>2</td>
<td>[0.07, 1.58]</td>
<td>[0.07, 1.58]</td>
<td>[0.06, 1.41]</td>
<td>[0.07, 1.58]</td>
<td>[0.07, 0.50]</td>
</tr>
<tr>
<td>doppler</td>
<td>5</td>
<td>[0.07, 1.58]</td>
<td>[0.07, 1.58]</td>
<td>[0.06, 1.41]</td>
<td>[0.07, 1.58]</td>
<td>[0.07, 0.50]</td>
</tr>
<tr>
<td>doppler</td>
<td>10</td>
<td>[0.07, 1.58]</td>
<td>[0.07, 1.58]</td>
<td>[0.06, 1.41]</td>
<td>[0.07, 1.58]</td>
<td>[0.07, 0.50]</td>
</tr>
<tr>
<td>himmilbeau</td>
<td>2</td>
<td>[105, 90.2]</td>
<td>[105, 90.2]</td>
<td>[56.0, 89.5]</td>
<td>[0, 0]</td>
<td>[105, 12.7]</td>
</tr>
<tr>
<td>himmilbeau</td>
<td>5</td>
<td>[105, 90.2]</td>
<td>[29.7, 11.5]</td>
<td>[0.03, 0]</td>
<td>[0, 0]</td>
<td>[105, 12.7]</td>
</tr>
<tr>
<td>himmilbeau</td>
<td>10</td>
<td>[105, 90.2]</td>
<td>[29.7, 11.5]</td>
<td>[0.03, 0]</td>
<td>[0, 0]</td>
<td>[105, 12.7]</td>
</tr>
</tbody>
</table>

6 Conclusions

This paper has four main contributions. First, we present how we have implemented Taylor models and affine arithmetic in CORA. To our best knowledge, our implementation of Taylor models is the only one available for MATLAB.

Second, we have improved the existing technique for computing divisions with Taylor models by computing a tighter Lagrange remainder.

Third, we compare our implementation of Taylor models with Flow* and our implementation of affine arithmetic with INTLAB. While Flow* is faster due to its implementation in C++, the implementation in CORA is easier to use due to the prototyping capabilities of MATLAB. Our implementation of affine arithmetic in CORA is faster compared to INTLAB, which is also implemented using MATLAB. However, our implementation does not consider floating-point errors.
Table 4: Comparison of Taylor model techniques (part 2). The precision is provided in percent compared to the exact result as shown in (20). The input ranges are presented in Tab. 8 and the examples are taken out of [25].

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Max. order</th>
<th>Interval subs.</th>
<th>BnB</th>
<th>BnB (adv.)</th>
<th>LDB/QFB</th>
<th>Zoo (Int. subs.)</th>
<th>Flow*</th>
</tr>
</thead>
<tbody>
<tr>
<td>kepler0</td>
<td>2</td>
<td>[8.22, 15.9]</td>
<td>[8.22, 15.9]</td>
<td>[1.23, 1.79]</td>
<td>[0, 0]</td>
<td>[8.22, 15.9]</td>
<td>[8.22, 19.2]</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>[8.22, 15.9]</td>
<td>[8.22, 15.9]</td>
<td>[1.23, 1.79]</td>
<td>[0, 0]</td>
<td>[8.22, 15.9]</td>
<td>[8.22, 19.2]</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>[8.22, 15.9]</td>
<td>[8.22, 15.9]</td>
<td>[1.23, 1.79]</td>
<td>[0, 0]</td>
<td>[8.22, 15.9]</td>
<td>[8.22, 19.2]</td>
</tr>
<tr>
<td>kepler1</td>
<td>2</td>
<td>[11.8, 37.7]</td>
<td>[11.8, 37.7]</td>
<td>[7.36, 4.70]</td>
<td>-</td>
<td>[11.8, 37.7]</td>
<td>[12.6, 51.4]</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>[11.8, 37.7]</td>
<td>[2.14, 11.9]</td>
<td>[0, 0.02]</td>
<td>[0, 0]</td>
<td>[11.8, 37.7]</td>
<td>[12.6, 51.4]</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>[11.8, 37.7]</td>
<td>[2.14, 11.9]</td>
<td>[0, 0.02]</td>
<td>[0, 0]</td>
<td>[11.8, 37.7]</td>
<td>[12.6, 51.4]</td>
</tr>
<tr>
<td>kepler2</td>
<td>2</td>
<td>[31.7, 42.1]</td>
<td>[31.7, 42.1]</td>
<td>[4.86, 6.16]</td>
<td>-</td>
<td>[31.7, 42.1]</td>
<td>[32.0, 53.8]</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>[31.7, 42.1]</td>
<td>[15.7, 28.0]</td>
<td>[0, 0.04]</td>
<td>[0, 0]</td>
<td>[31.7, 42.1]</td>
<td>[32.0, 53.8]</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>[31.7, 42.1]</td>
<td>[15.7, 28.0]</td>
<td>[0, 0.04]</td>
<td>[0, 0]</td>
<td>[31.7, 42.1]</td>
<td>[32.0, 53.8]</td>
</tr>
<tr>
<td>rigidBody1</td>
<td>2</td>
<td>[0, 14.7]</td>
<td>[0, 14.7]</td>
<td>[0, 0.01]</td>
<td>[0, 0]</td>
<td>[0, 10.9]</td>
<td>[0, 14.7]</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>[0, 14.7]</td>
<td>[0, 14.7]</td>
<td>[0, 0.01]</td>
<td>[0, 0]</td>
<td>[0, 10.9]</td>
<td>[0, 14.7]</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>[0, 14.7]</td>
<td>[0, 14.7]</td>
<td>[0, 0.01]</td>
<td>[0, 0]</td>
<td>[0, 10.9]</td>
<td>[0, 14.7]</td>
</tr>
<tr>
<td>rigidBody2</td>
<td>2</td>
<td>[13.5, 3.57]</td>
<td>[13.5, 3.57]</td>
<td>[1.95, 2.12]</td>
<td>-</td>
<td>[13.5, 3.57]</td>
<td>[21.9, 3.57]</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>[12.6, 2.64]</td>
<td>[10.5, 1.32]</td>
<td>[0.05, 0]</td>
<td>[0, 0]</td>
<td>[12.5, 2.64]</td>
<td>[21.0, 2.64]</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>[12.5, 2.64]</td>
<td>[10.50, 1.32]</td>
<td>[0.05, 0]</td>
<td>[0, 0]</td>
<td>[12.5, 2.64]</td>
<td>[21.0, 2.64]</td>
</tr>
<tr>
<td>turbine1</td>
<td>2</td>
<td>[135, 148]</td>
<td>[123, 140]</td>
<td>[122, 61.3]</td>
<td>-</td>
<td>[135, 2.67]</td>
<td>[1645, 1703]</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>[20.4, 62.1]</td>
<td>[20.4, 62.1]</td>
<td>[18.4, 7.85]</td>
<td>-</td>
<td>[20.4, 2.67]</td>
<td>[6113, 14542]</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>[229, 49.5]</td>
<td>[2.25, 49.5]</td>
<td>[2.20, 2.98]</td>
<td>-</td>
<td>[2.29, 2.67]</td>
<td>[1.4, 1.4]·107</td>
</tr>
<tr>
<td>turbine2</td>
<td>2</td>
<td>[124, 152]</td>
<td>[116, 147]</td>
<td>[58.7, 128]</td>
<td>-</td>
<td>[2.72, 152]</td>
<td>[1200, 1203]</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>[60.0, 66.6]</td>
<td>[28.0, 32.2]</td>
<td>[5.77, 17.3]</td>
<td>-</td>
<td>[2.72, 66.6]</td>
<td>[3296, 4739]</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>[50.5, 53.4]</td>
<td>[22.3, 18.9]</td>
<td>[0.78, 2.14]</td>
<td>-</td>
<td>[2.72, 53.4]</td>
<td>[51543, 51546]</td>
</tr>
</tbody>
</table>

Fourth, we provide an evaluation of range bounding techniques. We are not aware of any other paper that has evaluated the combined use of several range bounding techniques. Our finding is that it is typically better to use several simple range bounding methods in parallel and intersect the results, rather than using a single, precise technique.

Acknowledgments

The authors gratefully acknowledge financial support by the European Commission project UnCoVerCPS under grant number 643921 and the German Research Foundation (DFG) under grant AL 1185/5-1. We are also grateful to Xin Chen for providing us with a standalone Taylor model library comprising basic mathematical operations.
Table 5: Execution time of Taylor models in [ms]. The precision is shown in Tab. 3 and Tab. 4.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Max. order</th>
<th>2</th>
<th>5</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>sin</td>
<td>2</td>
<td>5.121</td>
<td>10.711</td>
<td>20.686</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>9.160</td>
<td>29.834</td>
<td>54.855</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>20.190</td>
<td>45.990</td>
<td>83.278</td>
</tr>
<tr>
<td></td>
<td></td>
<td>662.538</td>
<td>761.600</td>
<td>1183.286</td>
</tr>
<tr>
<td>bspline0</td>
<td></td>
<td>662.538</td>
<td>5.444</td>
<td>21.189</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.123</td>
<td>0.760</td>
<td>2.038</td>
</tr>
<tr>
<td>bspline1</td>
<td></td>
<td>662.538</td>
<td>11.338</td>
<td>4.955</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.123</td>
<td>0.315</td>
<td>0.229</td>
</tr>
<tr>
<td>bspline2</td>
<td></td>
<td>662.538</td>
<td>21.189</td>
<td>2.038</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.123</td>
<td>0.315</td>
<td>0.229</td>
</tr>
<tr>
<td>bspline3</td>
<td></td>
<td>662.538</td>
<td>4.955</td>
<td>0.229</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.123</td>
<td>0.315</td>
<td>0.229</td>
</tr>
<tr>
<td>doppler</td>
<td></td>
<td>662.538</td>
<td>7.808</td>
<td>0.368</td>
</tr>
<tr>
<td></td>
<td></td>
<td>662.538</td>
<td>7.656</td>
<td>0.368</td>
</tr>
<tr>
<td>himmilbeau</td>
<td></td>
<td>662.538</td>
<td>14.149</td>
<td>2.254</td>
</tr>
<tr>
<td></td>
<td></td>
<td>662.538</td>
<td>7.668</td>
<td>0.368</td>
</tr>
<tr>
<td>kepler0</td>
<td></td>
<td>662.538</td>
<td>13.034</td>
<td>0.248</td>
</tr>
<tr>
<td></td>
<td></td>
<td>662.538</td>
<td>13.034</td>
<td>0.248</td>
</tr>
<tr>
<td>kepler1</td>
<td></td>
<td>662.538</td>
<td>13.034</td>
<td>0.248</td>
</tr>
<tr>
<td></td>
<td></td>
<td>662.538</td>
<td>13.034</td>
<td>0.248</td>
</tr>
<tr>
<td>kepler2</td>
<td></td>
<td>662.538</td>
<td>13.034</td>
<td>0.248</td>
</tr>
<tr>
<td></td>
<td></td>
<td>662.538</td>
<td>13.034</td>
<td>0.248</td>
</tr>
<tr>
<td>rigidBody1</td>
<td></td>
<td>662.538</td>
<td>13.034</td>
<td>0.248</td>
</tr>
<tr>
<td></td>
<td></td>
<td>662.538</td>
<td>13.034</td>
<td>0.248</td>
</tr>
<tr>
<td>rigidBody2</td>
<td></td>
<td>662.538</td>
<td>13.034</td>
<td>0.248</td>
</tr>
<tr>
<td></td>
<td></td>
<td>662.538</td>
<td>13.034</td>
<td>0.248</td>
</tr>
<tr>
<td>turbine1</td>
<td></td>
<td>662.538</td>
<td>13.034</td>
<td>0.248</td>
</tr>
<tr>
<td></td>
<td></td>
<td>662.538</td>
<td>13.034</td>
<td>0.248</td>
</tr>
<tr>
<td>turbine2</td>
<td></td>
<td>662.538</td>
<td>13.034</td>
<td>0.248</td>
</tr>
<tr>
<td></td>
<td></td>
<td>662.538</td>
<td>13.034</td>
<td>0.248</td>
</tr>
</tbody>
</table>
Table 6: Comparison of affine arithmetic. The precision is provided in percent compared to the exact result as shown in (20). The input ranges are presented in Tab. 8 and the examples are taken out of [25].

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>CORA</th>
<th>interval subs.</th>
<th>BnB</th>
<th>BnB (adv.)</th>
<th>Interval</th>
<th>IntLab</th>
</tr>
</thead>
<tbody>
<tr>
<td>sin</td>
<td>[170.89, 106.22]</td>
<td>[170.89, 106.22]</td>
<td>[170.89, 106.22]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td></td>
</tr>
<tr>
<td>bspline0</td>
<td>[21.76, 0]</td>
<td>[21.76, 0]</td>
<td>[30.02, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td></td>
</tr>
<tr>
<td>bspline1</td>
<td>[0, 23.83]</td>
<td>[0, 23.83]</td>
<td>[-0.00, 33.31]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td></td>
</tr>
<tr>
<td>bspline2</td>
<td>[25.12, 0]</td>
<td>[25.12, 0]</td>
<td>[35.73, 0]</td>
<td>[3.92, 3.92]</td>
<td>[3.92, 0]</td>
<td></td>
</tr>
<tr>
<td>bspline3</td>
<td>[24.69, 0]</td>
<td>[24.69, 0]</td>
<td>[35.94, 0]</td>
<td>[0, 0]</td>
<td>[0, 0]</td>
<td></td>
</tr>
<tr>
<td>doppler</td>
<td>[0.44, 1.95]</td>
<td>[0.44, 1.95]</td>
<td>[0.44, 1.95]</td>
<td>[1.31, 0.50]</td>
<td>[0.45, 0.50]</td>
<td></td>
</tr>
<tr>
<td>himmilbeau</td>
<td>[136.64, 92.14]</td>
<td>[136.64, 92.14]</td>
<td>[159.66, 92.40]</td>
<td>[110.10, 12.71]</td>
<td>[110.10, 12.71]</td>
<td></td>
</tr>
<tr>
<td>kepler0</td>
<td>[8.22, 15.89]</td>
<td>[8.22, 15.89]</td>
<td>[8.22, 16.72]</td>
<td>[21.12, 32.14]</td>
<td>[8.22, 19.22]</td>
<td></td>
</tr>
<tr>
<td>kepler1</td>
<td>[16.82, 42.66]</td>
<td>[16.82, 42.66]</td>
<td>[16.82, 46.10]</td>
<td>[34.96, 77.05]</td>
<td>[20.74, 59.55]</td>
<td></td>
</tr>
<tr>
<td>kepler2</td>
<td>[67.20, 77.68]</td>
<td>[67.20, 77.68]</td>
<td>[67.20, 80.83]</td>
<td>[139.40, 148.98]</td>
<td>[69.01, 90.83]</td>
<td></td>
</tr>
<tr>
<td>rigidBody1</td>
<td>[0, 14.71]</td>
<td>[0, 14.71]</td>
<td>[0, 14.71]</td>
<td>[0.72, 10.85]</td>
<td>[0, 10.85]</td>
<td></td>
</tr>
<tr>
<td>rigidBody2</td>
<td>[18.32, 8.53]</td>
<td>[18.32, 8.53]</td>
<td>[20.46, 8.53]</td>
<td>[15.65, 9.86]</td>
<td>[15.65, 8.53]</td>
<td></td>
</tr>
<tr>
<td>turbine1</td>
<td>[206.21, 184.71]</td>
<td>[206.21, 184.71]</td>
<td>[208.26, 191.47]</td>
<td>[240.73, 2.67]</td>
<td>[206.98, 2.67]</td>
<td></td>
</tr>
<tr>
<td>turbine2</td>
<td>[148.01, 202.92]</td>
<td>[148.01, 202.92]</td>
<td>[153.10, 205.02]</td>
<td>[2.72, 238.35]</td>
<td>[2.72, 203.87]</td>
<td></td>
</tr>
</tbody>
</table>

Table 7: Execution time of affine arithmetic in [ms]. The precision is shown in Tab. 6.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>CORA</th>
<th>Int. subs.</th>
<th>BnB</th>
<th>BnB (adv.)</th>
<th>Interval</th>
<th>IntLab</th>
</tr>
</thead>
<tbody>
<tr>
<td>sin</td>
<td>3.293</td>
<td>6.540</td>
<td>11.332</td>
<td>0.192</td>
<td>10.649</td>
<td></td>
</tr>
<tr>
<td>bspline0</td>
<td>3.921</td>
<td>7.084</td>
<td>14.532</td>
<td>0.111</td>
<td>6.245</td>
<td></td>
</tr>
<tr>
<td>bspline1</td>
<td>6.305</td>
<td>11.060</td>
<td>21.072</td>
<td>0.136</td>
<td>8.966</td>
<td></td>
</tr>
<tr>
<td>bspline2</td>
<td>7.322</td>
<td>11.543</td>
<td>22.552</td>
<td>0.132</td>
<td>10.235</td>
<td></td>
</tr>
<tr>
<td>bspline3</td>
<td>3.544</td>
<td>6.476</td>
<td>13.892</td>
<td>0.095</td>
<td>4.071</td>
<td></td>
</tr>
<tr>
<td>doppler</td>
<td>9.728</td>
<td>24.025</td>
<td>35.102</td>
<td>0.197</td>
<td>10.624</td>
<td></td>
</tr>
<tr>
<td>himmilbeau</td>
<td>12.429</td>
<td>23.759</td>
<td>43.541</td>
<td>0.146</td>
<td>13.347</td>
<td></td>
</tr>
<tr>
<td>kepler0</td>
<td>12.546</td>
<td>28.480</td>
<td>39.405</td>
<td>0.216</td>
<td>12.799</td>
<td></td>
</tr>
<tr>
<td>kepler1</td>
<td>19.186</td>
<td>42.304</td>
<td>63.121</td>
<td>0.237</td>
<td>21.082</td>
<td></td>
</tr>
<tr>
<td>kepler2</td>
<td>30.710</td>
<td>79.738</td>
<td>129.766</td>
<td>0.348</td>
<td>33.086</td>
<td></td>
</tr>
<tr>
<td>rigidBody1</td>
<td>4.703</td>
<td>8.768</td>
<td>13.309</td>
<td>0.106</td>
<td>5.735</td>
<td></td>
</tr>
<tr>
<td>rigidBody2</td>
<td>11.942</td>
<td>24.764</td>
<td>41.808</td>
<td>0.161</td>
<td>14.412</td>
<td></td>
</tr>
<tr>
<td>turbine1</td>
<td>30.123</td>
<td>48.471</td>
<td>76.614</td>
<td>0.300</td>
<td>19.165</td>
<td></td>
</tr>
<tr>
<td>turbine2</td>
<td>19.423</td>
<td>34.023</td>
<td>53.376</td>
<td>0.219</td>
<td>12.887</td>
<td></td>
</tr>
</tbody>
</table>
Table 8: Input uncertainty for all benchmark models downloadable from https://github.com/malyzajko/daisy/blob/master/testcases/. The placeholders for the uncertainties are defined as follows: $a = [-4.5, -0.3]$, $b = [0.4, 0.9]$, $c = [3.8, 7.8]$, $d = [8, 10]$, $e = [-10, 8]$, $f = [1, 2]$.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Uncertainty</th>
<th>File path</th>
</tr>
</thead>
<tbody>
<tr>
<td>sin</td>
<td>$sin(x)$, $x \in a$</td>
<td></td>
</tr>
<tr>
<td>bspline0</td>
<td>$u \in a$</td>
<td>mixed-precision/Bsplines0.scala</td>
</tr>
<tr>
<td>bspline1</td>
<td>$u \in a$</td>
<td>rosa/Bsplines.scala</td>
</tr>
<tr>
<td>bspline2</td>
<td>$u \in a$</td>
<td>rosa/Bsplines.scala</td>
</tr>
<tr>
<td>bspline3</td>
<td>$u \in a$</td>
<td>rosa/Bsplines.scala</td>
</tr>
<tr>
<td>doppler</td>
<td>$u \in a$, $v \in b$, $T \in c$</td>
<td>rosa/Doppler.scala</td>
</tr>
<tr>
<td>himmilbeau</td>
<td>$x_1 \in a$, $x_2 \in b$</td>
<td>real2float/Himmilbeau.scala</td>
</tr>
<tr>
<td>kepler0</td>
<td>$x_1 \in a$, $x_2 \in b$, $x_3 \in c$, $x_4 \in d$, $x_5 \in e$, $x_6 \in f$</td>
<td>real2float/Kepler.scala</td>
</tr>
<tr>
<td>kepler1</td>
<td>$x_1 \in a$, $x_2 \in b$, $x_3 \in c$, $x_4 \in d$</td>
<td>real2float/Kepler.scala</td>
</tr>
<tr>
<td>kepler2</td>
<td>$x_1 \in a$, $x_2 \in b$, $x_3 \in c$, $x_4 \in d$, $x_5 \in e$, $x_6 \in f$</td>
<td>real2float/Kepler.scala</td>
</tr>
<tr>
<td>rigidBody1</td>
<td>$x_1 \in a$, $x_2 \in b$, $x_3 \in c$</td>
<td>control/RigidBody.scala</td>
</tr>
<tr>
<td>rigidBody2</td>
<td>$x_1 \in a$, $x_2 \in b$, $x_3 \in c$</td>
<td>control/RigidBody.scala</td>
</tr>
<tr>
<td>turbine1</td>
<td>$v \in a$, $w \in b$, $r \in c$</td>
<td>rosa/Turbine.scala</td>
</tr>
<tr>
<td>turbine2</td>
<td>$v \in a$, $w \in b$, $r \in c$</td>
<td>rosa/Turbine.scala</td>
</tr>
</tbody>
</table>

References


A List of Smooth Functions

Let $T = P(x - x_0) + [I]$ be a Taylor model $\forall x \in [x]$ (see Def. 2), where $P(x - x_0)$ is a polynomial part up to the $q$-th order, $I$ is an interval, $e_I = P(0)$, $\tilde{P}(x - x_0) = P(x - x_0) - e_I$, $\tilde{T} = \tilde{P}(x - x_0) + [I]$, and $B(\cdot)$ returns over-approximative bounds for Taylor models.

Monotonic functions:
Implementation of Taylor models in CORA 2018
Althoff, Grebenyuk, and Kochdumper

- Exponential from [32, Eq (2.2)-(2.4)]:

\[
\exp(T) = \exp(c_f) \cdot \left[ 1 + \frac{T}{2} + \frac{T^2}{2!} + ... + \frac{T^q}{q!} \right] \\
+ \frac{\exp(c_f)}{(q+1)!} \left( B(\tilde{T}^{q+1}) \cdot \exp([0, 1] \cdot B(\tilde{T})) \right).
\] (21)

- Logarithm from [32, p. 386] is defined under the condition \(B(T) \subset (0, \infty)\):

\[
\log(T) = \log(c_f) + \frac{T}{c_f} - \frac{1}{2} \frac{T^2}{c_f^2} + \frac{(-1)^{q+1}}{q} \frac{T^q}{c_f^q} \\
+ (-1)^{q+2} \frac{1}{q+1} \frac{B(\tilde{T}^{q+1})}{c_f^{q+1}} \frac{1}{(1 + [0, 1] \cdot B(\tilde{T}))^{q+1}}.
\] (22)

- Square root from [32, p. 386] is defined under the condition \(B(T) \subset (0, \infty)\):

\[
\sqrt{T} = \sqrt{c_f} \left[ 1 + \frac{T}{2 c_f} - \frac{1}{2!} \frac{T^2}{c_f^2} + ... + (-1)^{q-1} \frac{(2q-3)!}{q! 2^q c_f^q} \right] \\
+ (-1)^q \sqrt{c_f} \left( \frac{(2q-1)!}{(q+1)! 2^{q+1}} \right) \frac{B(\tilde{T}^{q+1})}{c_f^{q+1}} \frac{1}{(1 + [0, 1] \cdot B(\tilde{T}))^{q+1/2}}.
\] (23)

- Inverse \(\frac{1}{T}\): see Sec. 2.1

Trigonometric functions:

- Sine from [32, p. 387]:

\[
\sin(T) = \sin(c_f) + \cos(c_f) \cdot \tilde{T} - \frac{1}{2!} \sin(c_f) \cdot \tilde{T}^2 - \frac{1}{3!} \cos(c_f) \cdot \tilde{T}^3 + ... \\
+ \frac{1}{(q+1)!} B(\tilde{T}^{q+1}) \cdot J,
\] (24)

where

\[
J = \begin{cases} 
-J_0 & \text{if } \mod(q, 4) = 1 \text{ or } 2, \\
J_0 & \text{else},
\end{cases}
\]

\[
J_0 = \begin{cases} 
\cos(c_f + [0, 1] \cdot B(\tilde{T})) & \text{if } q \text{ is even}, \\
\sin(c_f + [0, 1] \cdot B(\tilde{T})) & \text{else}.
\end{cases}
\]

- Cosine from [32, p. 387]:

\[
\cos(T) = \cos(c_f) - \sin(c_f) \cdot \tilde{T} - \frac{1}{2!} \cos(c_f) \cdot \tilde{T}^2 + \frac{1}{3!} \sin(c_f) \cdot \tilde{T}^3 + ... \\
+ \frac{1}{(q+1)!} B(\tilde{T}^{q+1}) \cdot J,
\] (25)
where

\[ J = \begin{cases} -J_0 & \text{if } \mod(q, 4) = 0 \text{ or } 1, \\ J_0 & \text{else}, \end{cases} \]

\[ J_0 = \begin{cases} \sin(cf + [0, 1] \cdot B(\hat{T})) & \text{if } q \text{ is even}, \\ \cos(cf + [0, 1] \cdot B(\hat{T})) & \text{else}. \end{cases} \]

- Tangent:

\[ \tan(T) = \sin(T) \cdot \left( \frac{1}{\cos(T)} \right). \] (26)

**Hyperbolic functions:**

- Hyperbolic sine from [32, p. 388]:

\[ \sinh(T) = \sinh(cf) + \cosh(cf) \cdot \hat{T} + \frac{1}{2!} \sinh(cf) \cdot \hat{T}^2 + \frac{1}{3!} \cosh(cf) \cdot \hat{T}^3 + \ldots \]

+ \frac{1}{(q + 1)!} B(\hat{T}^{q+1}) \cdot J_0, \]

where

\[ J_0 = \begin{cases} \cosh(cf + [0, 1] \cdot B(\hat{T})) & \text{if } q \text{ is even}, \\ \sinh(cf + [0, 1] \cdot B(\hat{T})) & \text{else}. \end{cases} \]

- Hyperbolic cosine from [32, p. 388 ]:

\[ \cosh(T) = \cosh(cf) + \sinh(cf) \cdot \hat{T} + \frac{1}{2!} \cosh(cf) \cdot \hat{T}^2 + \frac{1}{3!} \sinh(cf) \cdot \hat{T}^3 + \ldots \]

+ \frac{1}{(q + 1)!} B(\hat{T}^{q+1}) \cdot J_0, \]

where

\[ J_0 = \begin{cases} \sinh(cf + [0, 1] \cdot B(\hat{T})) & \text{if } q \text{ is even}, \\ \cosh(cf + [0, 1] \cdot B(\hat{T})) & \text{else}. \end{cases} \]

- Hyperbolic tangent:

\[ \tanh(T) = \frac{\sinh(T)}{\cosh(T)}. \] (29)

**Inverse trigonometric functions:**

- Arcsine from [32, p. 388] under the condition of \( B(T) \subset (-1, 1) \):

\[ \arcsin(T) = \arcsin(cf) + G + \frac{1}{3!} G^3 + \frac{3^2}{5!} G^5 + \frac{3^2 5^2}{7!} G^7 + \ldots \]

+ \frac{1}{(q + 1)!} B(G)^{q+1} \arcsin^{(q+1)}([0, 1] \cdot B(G)), \] (30)
where
\[ G = T \cdot \sqrt{1 - c_f^2} - c_f \cdot \sqrt{1 - T^2} \]
and \( \arcsin^{(1)}(0) = 1, \arcsin^{(2)}(0) = 0, \arcsin^{(3)}(0) = 1, \) and \( \arcsin^{(q)}(0) = (q - 2)^2 \arcsin^{(q-2)}(0). \)

- Arccosine from [32, p. 389]:
  \[ \arccos(T) = \frac{\pi}{2} - \arcsin(T). \quad (31) \]

- Arctangent from [32, p. 389]:
  \[
  \arctan(T) = \arctan(c_f) + G^{q+1} \left( \frac{1}{1 + \left[0, 1\right] \cdot B(\tilde{T})} \right) \\
  \cdot \sin \left( (q + 1)(\arctan([0, 1] \cdot B(G)) + \frac{\pi}{2}) \right),
  \]
  where
  \[ G = \frac{T - c_f}{1 + c_f \cdot T}. \]

\section*{B Tight Bounds for the Taylor Model Inverse Operation}

As shown in (10), the equation for the \( \downarrow \) operation for a Taylor model \( T = P(x - x_0) + I \) can be derived from the Taylor series of the function \( f(x) = \frac{1}{x} \). The Lagrange remainder term
\[
L_q(B(\tilde{T}), c_f) = (-1)^{q+1} \frac{B(\tilde{T})^{q+1}}{c_f^{q+2}} \frac{1}{\left(1 + [0, 1] \cdot \frac{B(\tilde{T})}{c_f}\right)^{q+2}}, \quad c_f = P(0), \quad \tilde{T} = T - c_f, \quad (33)
\]
thereby represents an over-approximation of the exact integral remainder
\[
R_q(x, c_f) = \int_{c_f}^{x} f^{(q+1)}(t) \frac{(x - t)^q}{q!} dt, \quad x \in B(T), \quad (34)
\]
where \( f^{(i)} \) represents the \( i \)-th derivative of the function \( f(\cdot) \). Evaluation of the Lagrange remainder (33) for a Taylor model can lead to very large over-approximations, as the following example demonstrates:
\[
T = 0.45 + 0.35x + [0, 0], \quad x \in [-1, 1], \\
B(T) = [0.1, 0.8], \quad c_f = 0.45, \quad B(\tilde{T}) = [-0.35, 0.35], \\
\frac{1}{B(T)} = [1.25, 10], \\
L_{q=6}([-0.35, 0.35], 0.45) = [-64339.29688, 64339.29688].
\]

171
One solution is to analytically solve the integral in (34) for the function \( f(x) = \frac{1}{x} \), which results in an equation for the exact remainder:

\[
R_q(x, c_f) = \frac{(q + 1) \cdot (-1)^{(q+1)}}{2} \int_{c_f}^x \frac{(x - t)^q}{t^{(q+2)}} dt, \\
= -\frac{a \cdot (x - t)^q}{q + 1} \bigg|_{c_f}^x - \frac{a \cdot q}{q + 1} \int_{c_f}^x \frac{(x - t)^{(q-1)}}{t^{(q+1)}} dt, \\
= \frac{a \cdot (x - c_f)^q}{c_f^{(q+1)}} - \frac{q}{q + 1} \cdot R_{q-1}(x, c_f),
\]

with

\[
R_0(x, c_f) = a \cdot \int_{c_f}^x \frac{1}{t^2} dt = -\frac{a}{t} \bigg|_{c_f}^x = -\frac{a}{c_f} - \frac{a}{x}.
\]

The exact remainder therefore represents a polynomial, and its bounds on the range of the Taylor model \( R_q(B(T), c_f) \) can be calculated with standard interval arithmetic. Due to the limitations of interval arithmetic, this can also lead to possibly large over-approximations, which are however usually smaller than the over-approximations resulting from evaluation of the Lagrange remainder (33). Tighter bounds for the exact remainder can be obtained if the input interval \( B(T) \) is split into several sub-intervals. The over-approximation error can be made arbitrarily small if the sub-intervals are chosen small enough. Since the problem is one-dimensional, this splitting-based technique does not suffer from the curse of dimensionality.

Because the evaluation of the Lagrange remainder (33) is faster than the evaluation of the exact remainder (36), and because the Lagrange remainder does not always result in large over-approximations, we implemented the following heuristic in CORA for the calculation of the remainder \( \text{rem} \):

\[
\text{rem} = \begin{cases} 
L_q(B(\tilde{T}), c_f) & \text{if } L_q(B(\tilde{T}), c_f) \subset \frac{1}{B(T)}, \\
R_q(B(T), c_f) & \text{otherwise}
\end{cases}
\]

where \( c_f = P(0) \) and \( \tilde{T} = T - c_f \).

C Examples

This section presents the results of several examples evaluated in CORA:

```
1 a1 = interval(-1, 2); % generate a scalar interval [-1,2]
2 a2 = interval(2, 3); % generate a scalar interval [2,3]
3 a3 = interval(-6, -4); % generate a scalar interval [-6,4]
4 a4 = interval(4, 6); % generate a scalar interval [4,6]
5 b1 = taylm(a1, 6); % Taylor model with order 6 and name a1
6 b2 = taylm(a2, 6); % Taylor model with order 6 and name a2
7 b3 = taylm(a3, 6); % Taylor model with order 6 and name a3
8 b4 = taylm(a4, 6); % Taylor model with order 6 and name a4
9
10 B1 = [b1; b2] % generate a row of Taylor models
11 B2 = [b3; b4] % generate a row of Taylor models
12
13 B1 + B2 % addition
```
The resulting workspace output is:

\[
\begin{align*}
B_1 &= 0.5 + 1.5 \cdot a_1 + [0.00000,0.00000] \\
    &+ 2.5 + 0.5 \cdot a_2 + [0.00000,0.00000] \\
B_2 &= -5.0 + a_3 + [0.00000,0.00000] \\
    &+ 5.0 + a_4 + [0.00000,0.00000] \\
B_1 + B_2 &= -4.5 + 1.5 \cdot a_1 + a_3 + [0.00000,0.00000] \\
    &+ 7.5 + 0.5 \cdot a_2 + a_4 + [0.00000,0.00000] \\
B_1' \cdot B_2 &= 10.0 - 7.5 \cdot a_1 + 2.5 \cdot a_2 + 0.5 \cdot a_3 + 2.5 \cdot a_4 + 1.5 \cdot a_1 \cdot a_3 + 0.5 \cdot a_2 \cdot a_4 + [0.00000,0.00000] \\
B_1 \cdot B_2 &= -2.5 - 7.5 \cdot a_1 + 0.5 \cdot a_3 + 1.5 \cdot a_1 \cdot a_3 + 0.5 \cdot a_2 \cdot a_4 + [0.00000,0.00000] \\
B_1 / 2 &= 0.25 + 0.75 \cdot a_1 + [0.00000,0.00000] \\
    &+ 1.25 + 0.25 \cdot a_2 + [0.00000,0.00000] \\
B_1 \cdot B_2 &= -0.1 - 0.3 \cdot a_1 - 0.02 \cdot a_3 - 0.06 \cdot a_1 \cdot a_3 - 0.004 \cdot a_3^2 - 0.012 \cdot a_1 \cdot a_3^2 \\
    &- 0.0008 \cdot a_3^3 - 0.0024 \cdot a_1 \cdot a_3^3 - 0.00016 \cdot a_3^4 - 0.00048 \cdot a_1 \cdot a_3^4 \\
    &- 0.000032 \cdot a_3^5 - 0.0000096 \cdot a_1 \cdot a_3^5 - 6.4 \cdot 10^{-6} \cdot a_3^6 + [-0.000005,0.000005] \\
    &+ 0.5 + 0.1 \cdot a_2 - 0.1 \cdot a_4 - 0.02 \cdot a_2 \cdot a_4 + 0.02 \cdot a_4^2 - 0.004 \cdot a_2 \cdot a_4^2 \\
    &- 0.004 \cdot a_4^3 - 0.0008 \cdot a_2^2 \cdot a_4^3 + 0.0008 \cdot a_4^4 + 0.00016 \cdot a_2 \cdot a_4^4 \\
    &- 0.00016 \cdot a_4^5 - 0.000032 \cdot a_2 \cdot a_4^5 + 0.000032 \cdot a_4^6 + [-0.000005,0.000005] \\
B_1 \cdot 3 &= 0.125 + 1.125 \cdot a_1 + 3.375 \cdot a_1^2 + 3.375 \cdot a_1^3 + [0.000000,0.000000] \\
    &+ 15.625 + 9.375 \cdot a_2 + 1.875 \cdot a_2^2 + 0.125 \cdot a_2^3 + [0.000000,0.000000] \\
\sin(B_1) &= 0.47943 + 1.3164 \cdot a_1 - 0.53935 \cdot a_1 \cdot a_3 - 0.49364 \cdot a_1^2 + 0.10113 \cdot a_1^3 + 0.055535 \cdot a_1^4 \\
    &+ 0.0075847 \cdot a_1^5 - 0.0000059 \cdot a_1^6 + [-0.0000339,0.0000339] \\
    &+ 0.59847 - 0.40057 \cdot a_2 - 0.074809 \cdot a_2 \cdot a_3 - 0.01669 \cdot a_2^2 + 0.0015585 \cdot a_2^3 \\
    &- 0.00020863 \cdot a_2^4 - 0.000012988 \cdot a_2^5 + [-0.000000,0.000000] \\
    &+ 0.59847 - 0.40057 \cdot a_2 - 0.074809 \cdot a_2 \cdot a_3 - 0.01669 \cdot a_2^2 + 0.0015585 \cdot a_2^3 \\
    &- 0.00020863 \cdot a_2^4 - 0.000012988 \cdot a_2^5 + [-0.000000,0.000000] \\
\sin(B_1(1,1)) \cdot B_1(2,1) \cdot 2 - B_1' \cdot B_2 &= -3.2706 + 8.8164 \cdot a_1 - 0.5 \cdot a_3 - 2.5 \cdot a_4 - 0.53935 \cdot a_1 \cdot a_3 - 0.5 \cdot a_2 \cdot a_4 \\
    &+ 1.5 \cdot a_1 \cdot a_3 - 0.5 \cdot a_2 \cdot a_4 - 0.49364 \cdot a_1^2 + 0.10113 \cdot a_1^3 \\
    &+ 0.055535 \cdot a_1^4 - 0.0075847 \cdot a_1^5 + [-0.0000339,0.0000339]
\end{align*}
\]