Numer Algor (2015) 69:183–205
DOI 10.1007/s11075-014-9889-x

ORIGINAL PAPER

Rigorous integration of non-linear ordinary differential equations in chebyshev basis

Tomáš Dzetkulič

Received: 9 March 2012 / Accepted: 25 June 2014 / Published online: 5 July 2014

Abstract In this paper, we introduce a new approach to multiple step verified integration of non-linear ordinary differential equations. The approach is based on the technique of a Taylor model integration, however, a novel method is introduced to suppress the wrapping effect over several integration steps. This method is simpler and more robust compared to the known methods. It allows more general inputs, while it does not require rigorous matrix inversion. Moreover, our integration algorithm allows the use of various types of underlying function enclosures. We present rigorous arithmetic operations with function enclosures based on the truncated Chebyshev series. Computational experiments are used to show the wrapping effect suppression of our method and to compare integration algorithm that uses Chebyshev function enclosures with the existing algorithms that use function enclosures based on the truncated Taylor series (Taylor models).

Keywords Initial value problem · Rigorous integration · Taylor model · Chebyshev basis

1 Introduction

Non-rigorous numerical algorithms for the initial value problem of ordinary differential equations attempt to compute an approximate solution of the problem for a single initial state. These methods are reliable for most applications, but it is possible to find examples for which they compute very inaccurate results. Rigorous

T. Dzetkulič (✉)
Institute of Computer Science, Academy of Sciences of the Czech Republic, Pod Vodárenskou věží 2, 182 07, Prague 8, Czech Republic
e-mail: dzetkulic@cs.cas.cz

algorithms can compute a validated enclosure of the solution for a set of initial states and the enclosure is guaranteed to contain an exact solution for all initial states from the set [8]. Such enclosures are needed in some applications, for example in systems verification[13, 25], or in theorem provers [6, 27].

Current algorithms for the rigorous version of the initial value problem use either linear transformation of Cartesian product of intervals [16, 24], zonotope representation [15] or so-called Taylor models [21] for representing an enclosure of concrete solutions at each time step. In this paper, we extend the work on Taylor model based integration. We replace function enclosure based on truncated Taylor series (Taylor models) with function enclosure based on truncated Chebyshev series [4]. Moreover, we introduce a new method for suppression of the wrapping effect that is simpler than the known methods of shrink wrapping [3] and preconditioning [19], since it does not require rigorous matrix inversion.

We present rigorous operations with multi-variable function enclosures in the form of coefficients of the truncated Chebyshev series and the remainder term stored as an interval. In existing work [2, 5], only operations with functions in one variable are described. In [2], the function approximation is stored in the form of function values in the Chebyshev nodes. Authors use Fourier transform to compute coefficients of Chebyshev polynomials that they require in some operations, however, the methods are not rigorous (i.e., no enclosure of the exact function value is guaranteed). On the other hand the authors in [5] use rigorous methods, but not all operations required for solving the initial value problem are provided.

The authors of the original Taylor model method also investigated the truncated Chebyshev series, however, in [18] they list two fundamental limitations to the use of the truncated Chebyshev series:

- The Chebyshev series may lead to an increase in the magnitude of the coefficients, since low order approximations of high-order monomials like x^n usually involve larger low order terms cancelling each other.
- The truncated Chebyshev series of a product $f_1 \times f_2$, in general, cannot be obtained from the truncated Chebyshev series of the factors f_1 and f_2, because the result depends on the unknown high-order terms. As a consequence, the methods cannot be used to compute the truncated Chebyshev series of the multiplication, and the approximation will be sub-optimal.

In the corollary of Theorem 2, we argue that the first limitation of the use of the truncated Chebyshev series does not affect the algorithm described in this paper. Concerning the second limitation, in Section 5.3 we present an algorithm that computes the multiplication of two function enclosures in truncated Chebyshev series representation. The coefficients of Chebyshev polynomials computed in the algorithm are, in the general case, different from the coefficients in the optimal enclosure of $f_1 \times f_2$. Still, the computed result is a valid enclosure of $f_1 \times f_2$ and in Section 2.2 we show, that even though sub-optimal, the computed enclosure truncation error is lower compared to Taylor enclosure multiplication.

For the purpose of multiple step integration, we introduce a new method for handling the wrapping effect. The method is in its principles similar to the affine arithmetic approach [9] of handling uncertainties. We use a set of new variables to

represent the unknown uncertainty in each computation step. The computation of the solution enclosure in the following step handles these variables symbolically. The main difference to the affine arithmetic approach is that, after introducing affine combination of variables, we don't limit ourselves to affine operations and we treat new variables the same way as all statespace variables. New variables can appear in high order non-linear terms, thus this error handling mechanism has a potential to enclose non-convex error transformations and the wrapping effect does not affect the solution over multi-step integration. The proposed method is simpler and more robust compared to the existing methods.

Our implementation [11] supports function enclosures based on both Taylor and Chebyshev truncated series and allows their comparison. To our knowledge, this is the only publicly available rigorous implementation of the multi-variable function enclosure in the truncated Chebyshev series.

The content of the paper is as follows: In Section 2 we define our formalism for the rigorous function enclosure and the set of operations that are required in the verified integrator, in Section 3 we describe a single step of the integration method, in Section 4 we present a new method for suppression of the wrapping effect and we compare it to the existing methods, in Section 5 we construct function enclosure operations required in the integrator with both Taylor and Chebyshev function enclosures, in Section 6 we discuss our implementation, in Section 7 we evaluate the method using computation experiments, and in Section 8 we conclude the paper.

2 Function enclosure

Ordinary differential equations, initial values and the most of intermediate results are stored in the form of function enclosures in the rest of this paper. In this section we present the structure and meaning of the function enclosure used. We also list essential rigorous operations with the enclosures based on both the truncated Taylor series (Taylor function enclosures) and the truncated Chebyshev series (Chebyshev function enclosures). We also discuss how to construct function enclosures for a given function and we compare Taylor and Chebyshev function enclosures.

2.1 Polynomial basis

Definition 1 A *polynomial basis* is a set of linearly independent univariate polynomials that, in a linear combination, can represent every univariate polynomial in a polynomial vector space.

The *power polynomial basis* in variable x is the set of polynomials $P_i(x) = x^i$ for all $i \in \mathbb{N}_0$.

The *Chebyshev polynomial basis* in variable x is the set of polynomials $T_i(x)$ where $T_0(x) = 1$, $T_1(x) = x$ and $T_i(x) = 2x\ T_{i-1}(x) - T_{i-2}(x)$ for $i \in \{2..\infty\}$.

For a given polynomial basis, a univariate polynomial can be represented as a linear combination of some polynomials from that basis. For example polynomial

$x^3 + 3x^2 - 2$ can be represented as $P_3(x) + 3P_2(x) - 2P_0(x)$ in power basis or $0.25T_3(x) + 1.5T_2(x) + 0.75T_1(x) - 0.5T_0(x)$ in the Chebyshev basis.

The domain for all variables in the rest of the paper is the interval $[-1, 1]$. If a variable has a different domain, we do a substitution of the original variable to a shifted and scaled variable that fits the $[-1, 1]$ domain. The reason for this is that polynomials in both power and Chebyshev basis are bound on this interval as stated by the following theorem:

Theorem 1 *For all* $i \in \mathbb{N}_0$ *and* $x \in [-1, 1]$*:* $-1 \leq P_i(x) \leq 1$ *and* $-1 \leq T_i(x) \leq 1$.

The inequality for the power basis is trivial. In the Chebyshev basis, the bound for the value of $T_i(x)$ follows from the alternative equivalent definition of the Chebyshev basis: $T_i(x) = \cos(i \arccos(x))$.

2.2 Comparison Of taylor and chebyshev truncated series

In this section, we compare Taylor and Chebyshev truncated series. We show that the magnitude of coefficients, the truncation error and the multiplication error are lower or equal in the truncated Chebyshev series compared to the truncated Taylor series. In [18], the authors say that the Chebyshev series may lead to an increase in the magnitude of coefficients, since low order approximations of high-order monomials like x^n usually involve larger low order terms cancelling each other. This is true, when the approximation is expressed in the Taylor basis, but when the entire computation is evaluated in the Chebyshev basis, the magnitude of the coefficients does not increase as shown in the following Lemma. Because of this, such limitation does not affect algorithms in this paper.

Lemma 1 *For all n, the Chebyshev series of the function* $f(x) = x^n$ *is a Chebyshev polynomial of degree n, all the coefficients in the Chebyshev polynomial are positive and their sum is equal to 1.*

Proof We will prove the lemma by induction. For $n = 0$ the proposition is true. Let us assume it holds for all $n \leq k$. For k, we have that $x^k = \sum_{i=0}^{k} a_i T_i(x)$ for some $\{a\}_i$ positive with the sum of 1. For $n = k + 1$ we will use identity $T_i(x)T_j(x) = (T_{i+j}(x) + T_{|i-j|}(x))/2$: $x^{k+1} = T_1(x) \sum_{i=0}^{k} a_i T_i(x) = \sum_{i=0}^{k} a_i T_1(x) T_i(x) = \sum_{i=0}^{k} a_i T_{i+1}(x)/2 + \sum_{i=0}^{k} a_i T_{|i-1|}(x)/2$, thus for all i: $a_i/2$ appears twice in the series coefficients of function x^{k+1}. Since all a_i are positive, there is no cancellation and all coefficients are again positive. The sum of new coefficients is again equal to 1. □

Given a function series $f(x) = \sum_{i=0}^{\infty} a_i b_i(x)$ in either of the two bases, the sum $\sum_{i=n+1}^{\infty} |a_i|$, if it exists, is particulary important, because this sum gives an upper bound for the truncation error we make due to Theorem 1. In the following Theorem, we are given a function and we construct its Chebyshev polynomial from its Taylor polynomial. Since both polynomials are unique, this transformation allows us to compare truncation error in both bases.

Theorem 2 *For a function $f(x)$ with Taylor series $f(x) = \sum_{i=0}^{\infty} a_i x^i$: if the series coefficients $\{a\}_i$ converge absolutely (i.e., $\sum_{i=0}^{\infty} |a_i| < \infty$), the Chebyshev series coefficients $\{t\}_i$ converge absolutely as well and for all k: $\sum_{i=k}^{\infty} |t_i| \leq \sum_{i=k}^{\infty} |a_i|$.*

Proof We have that $f(x) = \sum_{i=0}^{\infty} a_i x^i$ and for each term $f_i(x) = a_i x^i$ we can construct Chebyshev series with coefficients t_j^i using Lemma 1 such that $\sum_{j=0}^{i} |t_j^i| = |a_i|$, for all $j > i$: $t_j^i = 0$ and $f_i(x) = \sum_{j=0}^{i} t_j^i T_j(x)$. We can now construct Chebyshev series for $f(x)$: $t_i = \sum_{j=i}^{\infty} t_i^j$. This sum exists since $|t_i^j| < |a_j|$ due to Lemma 1 and series $\{a\}_i$ converge absolutely. Thus for all k: $\sum_{i=k}^{\infty} |t_i| = \sum_{i=k}^{\infty} |\sum_{j=i}^{\infty} t_i^j| \leq \sum_{i=k}^{\infty} \sum_{j=i}^{\infty} |t_i^j| \leq \sum_{j=k}^{\infty} \sum_{i=k}^{\infty} |t_i^j| \leq \sum_{j=k}^{\infty} |a_j|$. □

Corollary 1 *If we apply Theorem 2 for $k = 0$, we see that for all functions with absolute convergent Taylor series, the Chebyshev series does not lead to an increase in the magnitude of coefficients.*

We have proved that the increase in the magnitude of coefficients in the Chebyshev basis does not affect functions with absolutely convergent power series. The truncation error is also lower in the Chebyshev basis. In some cases low order coefficients may be lower in the Taylor basis. Then, according to the above Theorem, higher order coefficients that contribute to the truncation error are higher. Decreasing the coefficient magnitude while increasing the truncation error negativelly affects precision thus we may conclude that the behavior in the Chebyshev basis is superior to the behavior in the Taylor basis.

Another fundamental limitation of the use of Chebyshev approximations according to the authors of [18] is that the truncated Chebyshev series of a product $f_1 \times f_2$, in general, cannot be obtained from the truncated Chebyshev series of the factors f_1 and f_2. The reason is, that the result depends on the unknown high-order terms. As a consequence, the authors say that the methods cannot be used to compute the truncated Chebyshev series of the multiplication, and the approximation will be sub-optimal.

While the argument that the optimal truncated Chebyshev series of the $f_1 \times f_2$ cannot be obtained is correct, this does not have such a negative consequences. If the truncated Chebyshev series of functions f_1 and f_2 are constructed, the exact product of those approximations can be obtained. If the input to the multiplication are two better approximations of f_1 and f_2, we show that the result is again a better approximation of $f_1 \times f_2$.

In the truncated series multiplication of functions $f_1(x) = \sum_{i=0}^{n} p_i b_i(x)$ with uncertainty e_1 and $f_2(x) = \sum_{i=0}^{n} q_i b_i(x)$ with uncertainty e_2, the error is generated from three sources:

- rounding
- $e_1 \times |max_x f_2(x)| + e_2 \times |max_x f_1(x)| + e_1 e_2$
- truncation of $(\sum_{i=0}^{n} p_i b_i(x)) \times (\sum_{i=0}^{n} q_i b_i(x))$

In Theorem 2, we proved that the magnitude of coefficients is smaller in the Chebyshev series. Because of this, we can expect also smaller rounding errors in the

Chebyshev series multiplication. In the same theorem, we argued, that the truncation error is lower in the truncated Chebyshev series. Thus the errors e_1 and e_2 are both lower in the Chebyshev basis and the second multiplication error is lower as well. Finally, the third error source for the power series is $\sum_{i=n+1}^{2n}\sum_{j=i-n}^{n} p_j q_{i-j} P_i(x)$, while for the Chebyshev series it is $\sum_{i=n+1}^{2n}\sum_{j=i-n}^{n} p_j q_{i-j} T_i(x)/2$. Because of the factor $1/2$ and the lower expected coefficients magnitude in the truncated Chebyshev series this error is lower as well. We conclude that, even though the computed Chebyshev approximation is sub-optimal, it is better than the optimal power series multiplication result. We demonstrate this on an example.

Example Assume we want to multiply order three approximations of functions $f_1(x) = \sin(x)$ and $f_2(x) = x^2$ ot the domain $x \in [-1, 1]$. The Taylor polynomial for $f_1(x)$ is $x + x^3/6$. The approximation error of $f_1(x)$ is highest in $x = \pm 1$ and it is $\sin(1) - 5/6 \cong 0.00814$. The multiplication result in the Taylor basis is $x^3 + x^5/6$ but since we work in order three, the final approximation is x^3. The approximation error is again highest in $x = \pm 1$ and it is $1 - \sin(1) \cong 0.15853$. The Chebyshev polynomial for $f_1(x)$ is approximately $0.88010T_1(x) - 0.03913T_3(x)$ and for $f_2(x)$ it is $T_0(x)/2 + T_2(x)/2$. The approximation error of $f_1(x)$ is 0.0005 in $x = 0.307721$. The truncated multiplication result in the Chebyshev basis is $0.65029T_1(x) + 0.20046T_3(x)$. The error of the approximation is 0.00973496 in $x = 0.307879$. In this example, both approximation errors increased after the multiplication by a similar factors. The Chebyshev approximation that we have computed is not the truncated Chebyshev series of $f_1(x) \times f_2(x)$, thus it is sub-optimal, but it is still better than the optimal Taylor approximation. □

2.3 Function enclosure and enclosure operations

In this section, we describe our function enclosure formalism, that we use for storing the function representation in the algorithm.

In our computations, we can use only the subset of the real numbers, that are representable on computer hardware. We denote the set of such numbers by Ω and the subset of representable numbers with the magnitude that cannot underflow or overflow in floating-point arithmetic operations by $\Omega^* \subset \Omega$. The detailed discussion of floating-point arithmetic can be found in Appendix 8. Now we define our rigorous function enclosure of a multivariate function.

The intuition behind the following function enclosure definition is that we store coefficients of the truncated Taylor or Chebyshev series of the represented function together with a single floating-point number used for the aggregate error from all sources (computation, rounding, truncation, etc.).

Definition 2 A *rigorous function enclosure* in polynomial basis $\{b\}_0^\infty$ is a pair $(F, e) \in (\mathbb{N}_0^m \to \Omega^*) \times \Omega$ of the truncated series coefficients F and an error bound e. The *center value* of such an enclosure is $val(F, x_1, \ldots, x_m) := \sum_{\mathbf{i}\in\mathbb{N}_0^m} F(\mathbf{i}) \prod_{j=1}^{m} b_{i_j}(x_j)$. The rigorous function enclosure (F, e) represents all

functions $g(x_1, \ldots, x_m)$ such that $\forall x_1, \ldots, x_m \in [-1, 1]^m : g(x_1, \ldots, x_m) \in [val(F, x_1, \ldots, x_m) - e, val(F, x_1, \ldots, x_m) + e]$. We denote this representation relation by $g \in (F, e)$. We denote the set of all rigorous function enclosures with m variables by Ψ_m. We say that the degree of enclosure is n, if for all $\mathbf{i} = (i_1, \ldots, i_m)$ such that $\sum_{j=1}^{m} i_j > n : F(\mathbf{i}) = 0$.

Using the rigorous function enclosure, we can directly represent any polynomial with coefficients in Ω^*, given that the degree of the enclosure is greater or equal to the degree of the polynomial represented. For example the polynomial $2b_1(x_1)b_0(x_2) + 3b_2(x_1)b_5(x_2)$ can be represented as $(F, e) = (\{a_{1,0} = 2, a_{2,5} = 3\}, 0)$ (we omit the zero polynomial coefficients in F). Moreover, using the error term, we can represent polynomials with coefficients that are not in Ω^*, polynomials with greater degree than the degree of the enclosure, and functions with absolutely convergent series. For example, the function $g(x) = 0.45x^2 + 0.04x^{100} + sin(x)$ can be represented using a Taylor enclosure of degree two using the following observations:

- $0.5 \in \Omega^*$ and for $x \in [-1, 1] : |0.45x^2 - 0.5x^2| \leq 0.05$
- for $x \in [-1, 1] : |0.04x^{100}| \leq 0.04$
- for $x \in [-1, 1] : |sin(x) - x| \leq 1 - sin(1) < 0.16$

thus we replace coefficient $0.45 \notin \Omega^*$ with $0.5 \in \Omega^*$, we include high-order term $0.04x^{100}$ in the error term and we replace $sin(x)$ with its approximation x. We add up all the errors and store it in the error term. $g(x)$ can now be represented as $(F, e) = (\{a_1 = 1, a_2 = 0.5)\}, 0.25)$, that is $P_1(x) + 0.5P_2(x) = x + 0.5x^2$ with the error 0.25.

In the verified integrator, the following operations with rigorous function enclosures with m variables are required:

- $unary\_minus : \Psi_m \to \Psi_m$, such that if $g \in (F, e)$ then $-g \in unary\_minus((F, e))$.
- $add : \Psi_m \times \Psi_m \to \Psi_m$, such that if $g_1 \in (F_1, e_1)$ and $g_2 \in (F_2, e_2)$ then $g_1 + g_2 \in add((F_1, e_1), (F_2, e_2))$.
- $mul : \Psi_m \times \Psi_m \to \Psi_m$, such that if $g_1 \in (F_1, e_1)$ and $g_2 \in (F_2, e_2)$ then $g_1 \times g_2 \in mul((F_1, e_1), (F_2, e_2))$.
- for all k: $substitute : \Psi_m \times \Psi_k^m \to \Psi_k$, such that if $g \in (F, e)$ and $s_1 \in (S_1, q_1), \ldots, s_m \in (S_m, q_m)$, then $g(s_1, \ldots, s_m) \in substitute((F, e), (S_1, q_1), \ldots, (S_m, q_m))$.
- $integrate : \Psi_m \to \Psi_m$, such that if $g(x_1, \ldots, x_m) \in (F, e)$ then $h(y, x_2, \ldots, x_m) = \int_0^y g(x_1, \ldots, x_m)dx_1 \in integrate((F, e))$.

Note that in the *substitution* operation, we substitute function enclosures (S_i, q_i) in the enclosure (F, e) whose domain is $[-1, 1]^m$. Thus the substitution is valid only if the range of all (S_i, q_i) is a subset of $[-1, 1]$. When using substitution, we have to prove that this inclusion holds.

The details of the construction of the operations above can be found in Section 5.

3 Verified integration - single step

The problem we are solving is the *initial value problem* for a given system of ordinary differential equations and *a set of* initial values. Given the input:

- system of n differential equations $\dot{\mathbf{x}} = f(\mathbf{x})$
- initial values over m free variables $\{\mathbf{x} \mid \exists \mathbf{a} \in [-1, 1]^m : g(\mathbf{a}) = \mathbf{x}\}$
- bound on time t_{max}

we want to compute the function $h(t, \mathbf{a})$, that is the solution to the system of differential equations $dh(t, \mathbf{a})/dt = f(h(t, \mathbf{a}))$ and $h(0, \mathbf{a}) = g(\mathbf{a})$.

Both input functions f and g are given in the form of rigorous function enclosures. The result of the computation is an enclosure of function h that is valid for time $t \in [-t_{max}, t_{max}]$ [1].

To obtain the enclosure for the solution h of the initial value problem, we use a method based on the iterated use of the Picard operator [21]. We set $h_0(t, \mathbf{a}) := 0$ and we compute a sequence of enclosures for the recurrence $h_{i+1}(t, \mathbf{a}) := g(\mathbf{a}) + \int_0^{t_{max}} f(h_i(t, \mathbf{a}))dt$ using the enclosure operations from Section 2. In case of convergence, the series converges to $h(t, \mathbf{a})$. Since the error term in the enclosure of $g(\mathbf{a})$ is constant and the error term in the enclosure of $\int_0^{t_{max}} f(h_i(t, \mathbf{a}))dt$ increases with the time bound t_{max}, we can control convergence by changing the time bound t_{max} to a value small enough to give us a convergent series.

If we denote by (H_i, e_i) the enclosure we obtain in the i-th step, we stop the iteration of the Picard operator when a fixpoint $(H_{i+1}, e_{i+1}) \subseteq (H_i, e_i)$ is reached. (H_{i+1}, e_{i+1}) is the enclosure of the problem solution $h(t, \mathbf{a})$.

4 Verified integration - wrapping effect supression

As explained in Section 3, there is a limit time bound t_{max} for which the iteration of Picard operator converges. If we want to integrate a differential equation over a larger time-span, we have to do several steps of integration. If we denote the solution in the i-th step by $h^i(t, \mathbf{a})$, then the most straightforward method to proceed with the next step is to set the initial condition in the step $i + 1$ to $g^{i+1}(\mathbf{a}) = h^i(t_{max}, \mathbf{a})$. In this section we explain, why such a simple approach is not suitable for the integration over several time steps. We introduce a new method that is more suitable for multi-step integration. We also discuss the difference between our method and the existing [3, 17, 19].

In case the initial state in i-th step $g^i(\mathbf{a})$ contains uncertainty, i.e., its enclosure has a non-zero error term, the error term in the enclosure of the single step integration result $h^i(t, \mathbf{a})$ is always bigger, since $h^i(0, \mathbf{a}) = g^i(\mathbf{a})$, thus the computed error term in the enclosure of h^i cannot be smaller compared to error term in the input enclosure of g^i. The uncertainty in the enclosure of h^i now applies for the entire time interval.

[1] In our algorithm, the function h is expanded at time 0, thus the computed result is valid also backwards in time.

Even if the differential equation was contracting, the same error term in h^i enclosure applies in both times $t = 0$ and $t = t_{max}$ thus the error does not contract as it should. Moreover, while applying the Picard operator iterativelly, we add g^i with its uncertainty in each iteration, but the information that we add the same uncertainty in each iteration is lost, and thus the resulting error is higher. Finally, the wrapping effect [22] causes exponential growth of the error term between the time steps. To solve these problems, we introduce the following method:

For integration step i and variable index j, let $\left(G^i_j, e^i_j\right)$ be the enclosure of $g^i_j(\mathbf{a})$. We introduce a new free variable p^i_j, we add term $e^i_j p^i_j$ to the enclosure coefficients and we clear the error term. The new enclosure $(Q^i_j, 0)$ now represents the same set as $\left(G^i_j, e^i_j\right)$, since any point in interval $\left[val(G^i_j) - e^i_j, val(G^i_j) + e^i_j\right]$ is the value of the function $val(Q^i_j) = val(G^i_j) + e^i_j p^i_j$ for some $p^i_j \in [-1, 1]$. The new variables $\mathbf{p}^i$ now carry the information about the unknown uncertainty in the i-th step and the dependency problem is solved. This is similar to the idea of uncertainty handling by affine arithmetic [9], because what we do is we introduce affine representation of the error to the system and then compute with it symbolically. The main difference is that once introduced into the system, new variables undergo full arithmetic as the regular statespace variables, thus we are able to enclose non-linear non-convex error transformations. Moreover, in the case of a contracting differential equation, substituting t_{max} into $h^i(t, \mathbf{a}, \mathbf{p}^i)$ handles variables $\mathbf{p}^i$ symbolically and the error contracts the same way as the initial set in the contracting differential equation.

The obvious problem is, that the initial set in the i-th step $(Q^i_j, 0)$ now contains more free variables than the input $\left(G^i_j, e^i_j\right)$. To avoid increasing the variable count in each integration step, we use the new set of variables $\mathbf{p}^i$ not only to store the error e^i but also to keep the information about the dependency of the input on the additional variables $\mathbf{p}^{i-1}$ from the previous step. This way the total number of variables is constant $n + m$ during the computation, where n is the problem dimension and m is the number of free variables $\mathbf{a}$. Given the initial condition $g^i_j(\mathbf{a}, \mathbf{p}^{i-1}) = h^{i-1}_j(t_{max}, \mathbf{a}, \mathbf{p}^{i-1}) \in \left(G^i_j, e^i_j\right)$, we split it into $g^{i+}_j(\mathbf{a}) \in (G^{i+}_j, 0)$ and $g^{i*}_j(\mathbf{a}, \mathbf{p}^{i-1}) \in \left(G^{i*}_j, e^i_j\right)$, where G^{i+}_j contains all coefficients of monomials independent on $\mathbf{p}^{i-1}$. Let us denote the bound on $\left(G^{i*}_j, e^i_j\right)$ by q_j. Now we introduce new variables p^i_j with coefficients q_j, i.e., the range of monomial $q_j p^i_j$ covers the unknown uncertainty $[g^{i*}_j(\mathbf{a}, \mathbf{p}^{i-1}) - e_j, g^{i*}_j(\mathbf{a}, \mathbf{p}^{i-1}) + e_j]$. We also store $\left(G^{i*}_j, e^i_j\right)$ and after the integration we construct a wrapping-effect free result through back-substitution of variables $\mathbf{p}$.

Example To illustrate the approach, let us consider a discrete system, where $\pm e$ denotes uncertainty of size e:

$$
\begin{aligned}
x_{n+1} &= x_n - y_n \\
y_{n+1} &= x_n + y_n \\
g_0(a) &= (3 + a \pm 0.1, \pm 0.1), \text{i.e.} (x_0, y_0) \in [1.9, 4.1] \times [-0.1, 0.1]
\end{aligned}
$$

where in addition to the initial state error, computation of each step involves a new error of magnitude 0.1. In case of a naive method, without the wrapping effect handling mechanism, the result is the following sequence of states:

$$\begin{aligned} g_1(a) &= ((3+a) \pm 0.3, (3+a) \pm 0.3) \\ g_2(a) &= (\pm 0.7, (6+2a) \pm 0.7) \end{aligned}$$

In our method, we replace $g_0(a)$ with equivalent $g_0(a, p_1^0, p_2^0)$ and we compute g_1:

$$g_0\left(a, p_1^0, p_2^0\right) = \left((3+a) + 0.1p_1^0, 0.1p_2^0\right)$$

$$g_1\left(a, p_1^0, p_2^0\right) = \left((3+a) + 0.1\left(p_1^0 - p_2^0\right) \pm 0.1, (3+a) + 0.1\left(p_1^0 + p_2^0\right) \pm 0.1\right)$$

Now we split g_1 into $g_1^+(a) = ((3+a), (3+a))$ and the remaining part $g_1^*\left(a, p_1^0, p_2^0\right)$. We find the bound on the latter part and construct new initial state. We also store the relation betweeen $\left(p_1^1, p_2^1\right)$ and $\left(p_1^0, p_2^0\right)$:

$$\begin{aligned} g_1\left(a, p_1^1, p_2^1\right) &= \left((3+a) + 0.3p_1^1, (3+a) + 0.3p_2^1\right) \\ \left(p_1^1, p_2^1\right) &= \left(\left(p_1^0 - p_2^0\right)/3 \pm 1/3, \left(p_1^0 + p_2^0\right)/3 \pm 1/3\right) \end{aligned}$$

Computing the following step leads to

$$g_2(a, p_1^1, p_2^1) = (0.3\left(p_1^1 - p_2^1\right) \pm 0.1, (6+2a) + 0.3\left(p_1^1 + p_2^1\right) \pm 0.1)$$

Using the back-substitution, we can estimate the error in g_2:

$$0.3\left(p_1^1 - p_2^1\right) \pm 0.1 = 0.3(\left(p_1^0 - p_1^0 - 2p_2^0\right)/3 \pm 2/3) \pm 0.1$$

Note that in the previous step, there is a cancellation in the symbolic computation of error and p_1^0 is removed from the expression. Substitution of ± 0.1 for p_2^0 in the last expression leads to the optimal error bound 0.5 compared to 0.7 in the naive method. □

The overall algorithm for the multiple step verified integration follows:

input:
  differential equations (F_j, e_j)
  initial condition $\left(G_j^0, e_j^0\right)$
  time bound t_{max}, number of steps N
for i **from** 0 **to** $N-1$ **do**
  // we start by introducing variables p_j^i
  for each j: split G_j^i into G_j^{i+} that does not contain variables $\mathbf{p}^{i-1}$
    and the remaining coefficients G_j^{i*}
  estimate the bound q_j on $\left(G_j^{i*}, e_j^i\right)$ and add term $q_j p_j^i$ to G_j^{i+}
  $(H_j, e_j) := (\emptyset, 0)$
  // Picard iteration cycle
  do
    for each j: $(H_j', e_j') := (H_j, e_j)$
      $(H_j, e_j) :=$
      $add((G_j^{i+}, 0), integrate(substitute((F_j, e_j), (H_1', e_1'), \ldots, (H_n', e_n'))))$

while $\exists j : (H_j, e_j) \not\subseteq (H'_j, e'_j)$
// Substitute t_{max} for time
for each j: $\left(G_j^{i+1}, e_j^{i+1}\right) := substitute((H_j, e_j), (\{a_\mathbf{0} = t_{max}\}, 0), \text{ID}, \ldots, \text{ID})$
where ID is a identity function
for each j: $\left(R_j^N, e_j^N\right) := \left(G_j^N, e_j^N\right)$
// back-substitution to compute result free of **p** variables
for i **from** $N-1$ **down to** 0 **do**
for each j: $\left(R_j^i, e_j^i\right) :=$
$substitute\left(\left(R_j^{i+1}, e_j^{i+1}\right), \text{ID}, \ldots, \text{ID}, \left(G_1^{i*}, e_1^i\right), \ldots, \left(G_n^{i*}, e_n^i\right)\right)$
i.e., we substitute $\left(G_k^{i*}, e_k^i\right)$ for variables p_k^i and keep variables **a** unchanged
return $\left(R_j^0, e_j^0\right)$

In addition to this algorithm, the error bound can be estimated not only in the end of the computation, but it can be computed more often based on a heuristic to minimize $q_j p_j^i$ introduced to the system.

There are several other known methods of dealing with the error growth in the multi-step verified integration. The shrink wrapping approach [3] is based on the idea of enclosing the remainder error into the range of the polynomial part of the function and dropping the error term. In the preconditioning approach [19], the idea is to write the initial values $g(\mathbf{a})$ in the form of composition of two functions, where the outer error-free function can be viewed as a specific coordinate system in which the motion is studied. In Lohner's QR approach [17] the error is represented in a moving orthogonal coordinate system that matches the rotation of the system. In Kühn's approach [15] the error is enclosed in the high order zonotope. In all of these methods, given a non-linear $g(\mathbf{a})$, authors use only linear part of it. In shrink wrapping, the linear part is then used to enclose the remainder error, while in preconditioning, the linear part is used as the basis of the outer function in the composition. In QR approach, the linear part is used to find suitable coordinate system to express the solution and in zonotope approach high order zonotopes are used. In all above methods, non-linear information is ignored and non-linear non-convex error transformations are enclosed by linear means. Such an approach works well in case the linear part dominates the non-linear part, however, in case the linear part is small or ill-conditioned, the result of applying such a computation based on linear terms to non-linear part is unpredictable. In addition to this, even if input set and differential equations are linear, the intermediate results may become non-linear. Another disadvantage of the shrink wrapping approach is that altering the polynomial part of the input weakens the corespondence between the input and output. Output $r(\mathbf{a})$ still represents the set $g(\mathbf{a})$ integrated over time t_{max} but for a given $\mathbf{a}_0$, the point $g(\mathbf{a}_0)$ integrated over time t_{max} is not $r(\mathbf{a}_0)$.

In their software COSY Infinity Makino and Berz also use additional variables to supress wrapping effect in the, yet unpublished, PSum algorithm. This algorithm again uses only linear combinations of additional variables. The method proposed in this paper has several differences from the PSum algorthm that allow us to enclose non-linear error transformations.

The advantage of the known methods is that they, except in the PSum algorithm, do not increase the number of free variables. Having less free variables reduces the computation cost of the single step integration and adding new variables causes exponential growth of computation time in the general case. In our case, the new variables represent the error that is usually small and we introduce only the affine combination of such variables into the enclosure of the initial set. The increase in the coefficient count of the result is thus not exponential and in our experiments the number of terms in the enclosure of the solution depending on variables $\mathbf{p}$ was approximately the same as the number of terms that do not depend on $\mathbf{p}$.

5 Polynomial operations

In this section, we show how to perform the polynomial operations listed in Section 2.

5.1 Basis independent polynomial operations

The operations $unary\_minus$ and add as presented in this paper are independent of the polynomial basis, so the same operation applies for the Taylor and Chebyshev enclosures.

For the $unary\_minus$ operation, it is enough to apply floating-point $unary\_minus$ on each individual coefficient in the enclosure. That is $unary\_minus((F, e)) = (unary\_minus(F), e)$.

Theorem 3 *For all functions* $g \in (F, e)$*:* $-g \in (unary\_minus(F), e)$.

Proof $g \in [val(F) - e, val(F) + e]$ implies that $-g \in [-val(F) - e, -val(F) + e]$. Since from definition of val, we have $-val(F) = val(-F)$, thus $-g \in [val(-F) - e, val(-F) + e]$ and $-g \in (unary\_minus(F), e)$. □

For the $add((F_1, e_1), (F_2, e_2))$ polynomial operation, we apply floating-point add on each pair of corresponding coefficients in the input enclosures. For all $\mathbf{i}$, we compute floating-point numbers $(a_\mathbf{i}, e_\mathbf{i}) = add(F_1(\mathbf{i}), F_2(\mathbf{i}))$. We compute e_3 as the upward rounded sum of all errors $|e_\mathbf{i}|$. For all $\mathbf{i}$, we set $F(\mathbf{i}) := a_\mathbf{i}$. Now we construct the result of the addition: $add((F_1, e_1), (F_2, e_2)) = (F, e_1 \overline{+} e_2 \overline{+} e_3)$.

Theorem 4 *For all functions* $g_1 \in (F_1, e_1)$ *and* $g_2 \in (F_2, e_2)$*:*
$g_1 + g_2 \in add((F_1, e_1), (F_2, e_2))$.

Proof $g_1 + g_2 \in [val(F_1) + val(F_2) - e_1 - e_2, val(F_1) + val(F_2) + e_1 + e_2]$. Let us examine $val(F_1) + val(F_2)$. It is the sum of: $(F_1(\mathbf{i}) + F_2(\mathbf{i})) \prod_{j=1}^{m} b_{i_j}(x_j) = (a_\mathbf{i} + e_\mathbf{i}) \prod_{j=1}^{m} b_{i_j}(x_j)$. From the Theorem 1, we can bound the absolute value of $e_\mathbf{i} \prod_{j=1}^{m} b_{i_j}(x_j)$ to be $|e_\mathbf{i}|$ and the sum of all of such terms never exceed e_3 in magnitude. The sum of the rest of the terms is exactly $val(F)$, thus we have $g_1 + g_2 \in (F, e_1 \overline{+} e_2 \overline{+} e_3)$. □

The result of the above operation may not be a valid function enclosure, since some of the computed $a_{\mathbf{i}}$ may not lie in Ω^* (however, $a_{\mathbf{i}} \in \Omega$, since we assume that both input enclosures are a valid function enclosures). We handle this by the following additional operation *normalize*: In case there is an overflowing coefficient $a_{\mathbf{i}}$, we report an overflow error and we stop the computation. For each underflowing $a_{\mathbf{i}}$, we set that coefficient in F to zero and we bound the maximum absolute value of the term corresponding to that coefficient using Therorem 1 to be $|a_{\mathbf{i}}|$. We add this additional error to the error term using upward rounded addition.

The *normalize* operation has to be used not only after the *add* operation, but after all remaining operations as well, since all of those perform some floating-point computation.

Now we formulate a lemma that allows us to use the *add* operation to simplify the rest of the function enclosure operations.

Lemma 2 *For function g and function enclosure (F, e) such that $g \in (F, e)$ and for arbitrary index $\mathbf{i}$, we can construct new coefficients G and their complement G^- such that $G(\mathbf{i}) := 0$ and for all $\mathbf{j}, \mathbf{j} \neq \mathbf{i}$: $G(\mathbf{j}) := F(\mathbf{j})$. G^- is the remaining $\mathbf{i}$-th coefficient: $G^- := F - G$. For the new coefficients, the following identity holds: $g - val(G^-) \in (G, e)$.*

Proof Since $g \in (F, e)$, $g \in [val(F) - e, val(F) + e]$ thus $g - val(G^-) \in [val(F) - val(G^-) - e, val(F) - val(G^-) + e]$. Since $F - G^- = G$, we have $g - val(G^-) \in [val(G) - e, val(G) + e]$. □

In a non-trivial case when $F(\mathbf{i})$ is non-zero, the enclosure of the function $val(G^-)$ is $(G^-, 0)$ and it has only one non-zero coefficient. The enclosure (G, e) of the function $g - val(G^-)$ has one less non-zero coefficient compared to (F, e), thus both splitted enclosures are simpler compared to the original enclosure. When multiplying, integrating or substituting into (F, e), we can now multiply, integrate or substitute into those simpler enclosures and then add the results, because of the following identities:

- $g \times f = (g - val(G^-)) \times f + val(G^-) \times f$
- $g(f) = (g - val(G^-))(f) + val(G^-)(f)$
- $\int g = \int (g - val(G^-)) + \int val(G^-)$

We can iterate the use of Lemma 2 to split enclosures with more than one coefficient until we arrive in enclosures with at most one non-zero coefficient. Because of this, it is enough to show the rest of the operations on such simple inputs.

5.2 Taylor enclosure operations

In this section we explain how to perform the *mul*, *substitute* and *integrate* function enclosure operations with Taylor enclosures.

For the $mul((F_1, e_1), (F_2, e_2))$, let us assume F_1 is non-zero only in a single index $\mathbf{i}$ and F_2 is non-zero only in index $\mathbf{j}$. We compute $(a, b) = mul(F_1(\mathbf{i}), F_2(\mathbf{j}))$.

The result of the multiplication $mul((F_1, e_1), (F_2, e_2))$ is then $(F(\mathbf{i}+\mathbf{j}) = a, b\overline{+}e_1\overline{\times}|F_2(\mathbf{j})|\overline{+}e_2\overline{\times}|F_1(\mathbf{i})|)$.

For the *substitute*$((F, e), (S_1, q_1), \ldots, (S_m, q_m))$ operation, we again show, how to do this operation when F has only one non-zero coefficient. Let us assume F is non-zero only in a single index $\mathbf{j}$. We compute enclosure for $F(\mathbf{j}) \times (S_1, q_1)^{\mathbf{j}_1} \times \cdots \times (S_m, q_m)^{\mathbf{j}_m}$ using the *mul* operation and finally we add e to the error using upward rounded addition.

For the *integrate*$((F, e))$ operation, we again assume that F has only one non-zero coefficient at index $\mathbf{j}$. We compute floating-point numbers $(a, b) = div(F(\mathbf{j}), \mathbf{j}_1 + 1)$ and new index $\mathbf{i} = \mathbf{j} + (1, 0, \ldots, 0)$. The result of the integration is then the enclosure $(F(\mathbf{i}) = a, b\overline{+}e)$.

5.3 Chebyshev enclosure operations

In this section we present new method to perform the *mul*, *substitute* and *integrate* function enclosure operations with multivariate Chebyshev enclosures.

For the $mul((F_1, e_1), (F_2, e_2))$, we again assume F_1 is non-zero in a single index $\mathbf{i}$ and F_2 is non-zero only in index $\mathbf{j}$. We compute $(a, b) = mul(F_1(\mathbf{i}), F_2(\mathbf{j}))$. To construct the result of the multiplication, we need the following identity: for all p, q, x: $T_p(x)T_q(x) = (T_{p+q}(x) + T_{|p-q|}(x))/2$. In case $p = 0$ or $q = 0$ this identity simplifies to $T_p(x)T_q(x) = T_{p+q}(x)$. We count the number m of indices k such that $\mathbf{i}_k = 0$ or $\mathbf{j}_k = 0$ and we compute $a' = a/2^m$. We construct the result coefficients F such that for all indices $\mathbf{r}$, where for all k: $\mathbf{r}_k = \mathbf{i}_k + \mathbf{j}_k$ or $\mathbf{r}_k = |\mathbf{i}_k - \mathbf{j}_k|$ we set $F(\mathbf{r}) = a'$. The result of the multiplication $mul((F_1, e_1), (F_2, e_2))$ is then $(F, b\overline{+}e_1\overline{\times}|F_2(\mathbf{j})|\overline{+}e_2\overline{\times}|F_1(\mathbf{i})|)$.

Multiplication of two n-variable function enclosures that have one non-zero coefficient each may result in up to 2^n non-zero coefficient result. This is a big increase compared to single non-zero coefficient in the same case for Taylor enclosures, however, it happens only in a special case when the multiplied coefficients belong to a terms that depend on most of the variables. In practice, such coefficients are small and the division with the divisor 2^m decreases their magnitude even more. In Section 6 we describe a recursive algorithm for multiplication that is efficient for the function enclosures that appear in practice.

For the *substitute*$((F, e), (S_1, q_1), \ldots, (S_m, q_m))$ operation we use the same approach as we did with the power basis, but we use Clenshaw Algorithm 7 to compute enclosure of $F(\mathbf{j}) \times T_{\mathbf{j}_1}((S_1, q_1)) \times \cdots \times T_{\mathbf{j}_m}((S_m, q_m))$.

For *integrate*$((F, e))$ operation, we need the following identities:

- $\int_0^y T_0(x)dx = T_1(y)$
- $\int_0^y T_1(x)dx = (T_0(y) + T_2(y))/4$
- for even $i > 1$: $\int_0^y T_i(x)dx = (-T_{i-1}(y)/(i-1) + T_{i+1}(y)/(i+1))/2$
- for $i > 1$ and $i \equiv 1 \mod 4$:
  $\int_0^y T_i(x)dx = ((1/(i-1)+1/(i+1))T_0(y) - T_{i-1}(y)/(i-1) + T_{i+1}(y)/(i+1))/2$
- for $i > 1$ and $i \equiv 3 \mod 4$:
  $\int_0^y T_i(x)dx = ((-1/(i-1) - 1/(i+1))T_0(y) - T_{i-1}(y)/(i-1) + T_{i+1}(y)/(i+1))/2$

We assume the only non-zero coefficient in F is $F(\mathbf{i})$ and $\mathbf{i}_1 \equiv 1 \mod 4$. The rest of the cases are analogous to this one. We compute floating point numbers $(a, b) = div(F(\mathbf{i}), \mathbf{i}_1 - 1)$, $(c, d) = div(F(\mathbf{i}), \mathbf{i}_1 + 1)$ and $(p, q) = add(a, c)$. To construct the result coefficients G of the integration, we set $G(\mathbf{i} - (1, 0, \ldots, 0)) = a/2$, $G(\mathbf{i} + (1, 0, \ldots, 0)) = c/2$ and $G(\mathbf{i} - (\mathbf{i}_1, 0, \ldots, 0)) = p/2$. The result of the integration is than $(G, e \overline{+} b \overline{+} d \overline{+} q/2)$.

6 Implementation

For enclosures appearing in practice, most of the high degree terms are usually small in magnitude. Because of this characteristics, it is essential for performance reasons to store and compute with enclosures in sparse form. In our experiments, the addition and integration operations are easy to perform since in those operations result can be constructed quickly based on Lemma 2. The multiplication operation is the most time-consuming operation and it is often used as a sub-operation in the substitution operation. Because of this, we explain how to perform the multiplication operation effectively.

In our implementation, we fix n, the degree of all stored enclosures. If an operation creates a term with order higher than n or a term that is much smaller in magnitude than the error term, we include such terms in the error using the same approach as we use for hiding underflowing terms in *normalize* operation in Section 5.1.

For the performance reasons, it is important to detect such a small and high-order terms early in the computation, so that the individual coefficients are estimated and never computed. To do this, in Taylor enclosure multiplication we group terms in the input based on their degree. For each pair of the groups from the input, we compute the multiplication only if the sum of the group degrees is less or equal to n. In case the sum of degrees is greater than n, the bound on the group is computed using upward rounded addition and then the upward rounded multiplication of the bounds is added to the error. This way individual coefficients from groups that would only contribute to error term are never multiplied.

Let m be the number of variables. In the Chebyshev enclosure multiplication, using a naive multiplication algorithm leads to a huge increase in the computation time, because each pair of the input coefficients contributes to up to 2^m coefficients in the result. We propose the following recursive algorithm for the multiplication to avoid this problem: When multiplying Chebyshev polynomials $p(x_1, \ldots, x_m)$ and $q(x_1, \ldots, x_m)$, we first isolate the first variable: $p(x_1, \ldots, x_m) = \sum_i a_i(x_2, \ldots, x_m) T_i(x_1)$, $q(x_1, \ldots, x_m) = \sum_i b_i(x_2, \ldots, x_m) T_i(x_1)$. Now, for each pair (i, j) we recursively multiply polynomials in one less variable: $c_{(i,j)}(x_2, \ldots, x_m) = a_i(x_2, \ldots, x_m) b_j(x_2, \ldots, x_m)$. We can then construct the result of the multiplication: $p \times q = \sum_{(i,j)} (T_{i+j}(x_1) + T_{|i-j|}(x_1)) c_{(i,j)}(x_2, \ldots, x_m)/2$. In this way, a lot of cancellation occurs in each level of the recursion since the total number of coefficients $c_{(i,j)}$ is much higher than the maximum degree n and the worst time case complexity of the multiplication is asymptotically the same as for power basis multiplication.

7 Computational experiments

To compare our simple implementation with the existing mature Taylor model software COSY Infinity, we use Volterra equation example from [20]:

$$\begin{aligned} \dot{x} &= 2x(1-y) \\ \dot{y} &= y(x-1) \\ x_0 &\in [0.95, 1.05];\ y_0 \in [2.95, 3.05];\ t_{max} = 5.488138468035 \end{aligned}$$

and Roessler system example from [19]:

$$\begin{aligned} \dot{x} &= -(y+z) \\ \dot{y} &= x + 0.2y \\ \dot{z} &= 0.2 + z(x-5.7) \\ x_0 &\in [-0.2, 0.2];\ y_0 \in [-8.58095, -8.18095];\ z_0 \in [-0.1704098, 0.2295902] \\ t_{max} &= 6 \end{aligned}$$

In the computation, we use small time step sizes, such that the precision is not limited by the Picard operator, but the representation order becomes the limit. This way we can compare the ability to enclose non-convex shapes by the Chebyshev function enclosure and the Taylor function enclosure. The width of the error of the enclosure in time t_{max} for both examples is presented in Table 1.

Our implementation using Taylor enclosure integration is worse than COSY, probably because of the worse Picard operator implementation. Replacing the Taylor enclosures with the Chebyshev enclosures and keeping the rest of the algorithm unchanged allows us to get a several orders of magnitude smaller error term in the problem solution. The difference in the error magnitude is higher with the higher order of the enclosure and with high-order enclosures, we compute better results compared to COSY.

We took additional benchmarks from the VERICOMP online database [1]. In the Table 2 we present enclosure width and computation time of our tool on VERICOMP benchmarks 1–6, and benchmark 28. We compare our tool to publicly available tools

Table 1 Experimental results. COSY Infinity results from [19, 20]

	Our tool		COSY
Problem (enclosure degree)	Taylor enclosure	Chebyshev enclosure	
Volterra (10)	1.1E-6	5.7E-9	
Volterra (12)	3.4E-8	5.2E-11	3E-9
Volterra (14)	1.1E-9	9.8E-13	
Roessler (12)	1.8E-6	1.4E-8	1.3E-9
Roessler (14)	1.2E-7	2.7E-10	2.9E-10
Roessler (16)	9E-9	5.7E-12	7.3E-11
Roessler (18)	6.6E-10	5.3E-13	3.9E-11

Table 2 VERICOMP computation experiments [1]

		VNODE_LP		RIOT		Our tool	
Benchmark #	Variable count	Width	Time	Width	Time	Width	Time
1	2	4.67079	0.01s	10.1	2s	4.67078	0.08s
2	3	0.232544	0.01s	0.235	0.7s	0.232544	0.03s
3	1	0.89	0.01s	0.44	40s	0.38	0.1s
4	2	0.073	0.02s	0.067569	38s	0.067561	0.3s
5	51	0.21527	2s	–	–	0.21527	12s
6	30	2.95E-5	3s	–	–	2.54E-5	93s
28	2	–	–	–	–	1.018	4.5s

VNODE_LP [24], RIOT [12] and ValenciaIVP [26], however we omit results of the tool ValenciaIVP in the table, bacause it does not provide tightest results on any of the displayed benchmarks. For each tool and benchmark the result depends on the solver settings. We present the best result based on the enclosure width. In case multiple settings provide narrow results, we display the one with the lowest computational time.

The benchmarks in the VERICOMP database include uncertainties in the ODE or in the initial state such that the optimal result is not a single point but rather a set of points. Where the optimal width of this set is known, our results in Table 2 match the optimal width in all displayed digits. From Table 2, we can see that tool VNODE_LP is always the fastest one but it cannot provide tightest results for non-linear benchmarks 3,4,6, and 28. The tool RIOT sometimes outperforms VNODE_LP in terms of precision, but it is much slower and sometimes struggles with simple linear benchmarks as benchmark 1. Our tool using Chebyshev function enclosures can solve some benchmarks unsolved by other tools, while providing tight results.

Finally, to evaluate our wrapping effect suppression method, we use two-state pincushion and stretch benchmark from [3]. Considering a two state system instead of an ODE allows us to perform hundreds of thousands of steps and evaluating wrapping effect suppression method on a greater scale.

From the results in Table 3 we can conclude that both tools can handle the wrapping effect in this benchmark and there is not an exponential growth of the error. For the initial state $(1, 1) + [-0.05, 0.05]^2$ both tools fail to compute the enclosure

Table 3 Error width in the two-state pincushion and stretch benchmark [3] after 100000 steps

Polynomial	COSY		Our tool	
order	$[-0.05, 0.05]^2$	$(1, 1) + [-0.05, 0.05]^2$	$[-0.05, 0.05]^2$	$(1, 1) + [-0.05, 0.05]^2$
5	2×10^{-2}	–	3.8×10^{-3}	–
10	1×10^{-7}	2×10^{-4}	1.3×10^{-8}	7.2×10^{-4}
20	1×10^{-9}	5×10^{-8}	2.7×10^{-12}	1.1×10^{-9}

with order 5 polynomials. COSY outperforms our tool with order 10 polynomials, but with higher orders our tool provides tighter enclosures. This type of behavior can be caused by various causes. One possible cause is the different implementation of enclosure division and square root. Another possibility is the different heuristic strategy in the removing of the underflowing polynomial terms. Tighter enclosures in the order 20 can be attributed to the use of the Chebyshev polynomial enclosures.

8 Conclusion

In this paper, we introduced two improvements to the Taylor model-based verified integration. Replacing the Taylor enclosures with the Chebyshev enclosures allows the method to compute enclosure of the solution of the initial value problem with higher precision while using low order approximations. This is a very important property when the number of variables grows big, because the number of polynomial coefficients increases drastically in such a case with each additional order. Being able to approximate function better with less coefficients is a crucial property of the future rigorous integrators. The method is also able to suppress the wrapping effect over multiple integration steps due to new error handling approach. Computational experiments confirm an advantage of the use of the Chebyshev enclosures over the Taylor enclosures as well as the suppression of the wrapping effect. In the future work, we plan to improve our step size control and use better single step algorithm. We plan to use the method in the hybrid system safety verification Algorithm [25].

Acknowledgements This work was supported by Czech Science Foundation grant 201/09/H057, Ministry of Education, Youth and Sports project number OC10048 and Institute of Computer Science institutional research plan AV0Z100300504. The author would like to thank Stefan Ratschan for a valuable discussion and helpful advice.

Appendix A: Self-validated floating point arithmetic

In this section we describe how we use the arithmetic operations to get rigorous bounds on all computed values and we compare the operations presented here to other rigorous frameworks. The approach is similar to [28], where authors use multiple floating point numbers to achieve high precision. We limit ourselves to single floating point number representation, but still in Theorem 5 we proove, that this method provides better results than the interval arithmetic approach [23].

In our algorithms we use the floating point numbers for representing the numerical quantities. All floating point numbers are assumed to be in IEEE *double precision format* [14]. We denote the set of all representable numbers by $\Omega \subset \mathbb{R}$. Since only the fixed number of binary digits are being stored, the arithmetic operations like addition and multiplication involve rounding of the result. In the following, we distinguish between the exact arithmetic operations ($+$, $-$, $\times$, $/$ etc.) and the rounded floating point operations ($\oplus$, $\ominus$, $\otimes$, $\oslash$ etc.). Rounded values are assumed to be computed in the default *round to nearest, ties to even* rounding mode.

The arithmetic operation on hardware may *overflow* or *underflow*, that is, it produces result greater or smaller in magnitude than the numbers in Ω. Since this may cause some unwanted behavior, we pick a big constant M (in our case 10^{100}), such that the arithmetic operation on the set $\Omega^* = \{x \mid x \in \Omega \text{ and } x \in [-M, M] \text{ and } (x = 0 \text{ or } x \leq -1/M \text{ or } x \geq 1/M)\}$ never overflows or underflows. In general, when an overflowing result is produced during the computation, we cannot continue with computation and we report an overflow error. However, when the underflowing result is produced, such value is treated as zero and rigorous bound for this result is treated as the roundoff error.

In the following, we require the existence of the following operations:

- $unary\_minus : \Omega \to \Omega$ such that $unary\_minus(x) = (-1) \times x$
- upward-rounded addition $\overline{+}$: $\Omega^* \times \Omega^* \to \Omega$ such that $x \overline{+} y = a$ implies that a is the smallest element in Ω such that $a \geq x + y$
- upward-rounded multiplication $\overline{\times}$: $\Omega^* \times \Omega^* \to \Omega$ such that $x \overline{\times} y = a$ implies that a is the smallest element in Ω such that $a \geq x \times y$
- $add : \Omega^* \times \Omega^* \to \Omega \times \Omega$ such that $add(x, y) = (a, b)$ implies that $x \oplus y = a$ and $x + y = a + b$
- $mul : \Omega^* \times \Omega^* \to \Omega \times \Omega$ such that $mul(x, y) = (a, b)$ implies that $x \otimes y = a$ and $x \times y = a + b$
- $div : \Omega^* \times (\Omega^* - \{0\}) \to \Omega \times \Omega$ such that $div(x, y) = (a, b)$ implies that $x \oslash y = a$ and $|x/y - x \oslash y| \leq b$

The operations $unary\_minus$, $\overline{+}$ and $\overline{\times}$ are guaranteed to exist due to the IEEE Standard [14]. The operations add and mul can be computed using the method introduced in [10], based on the observation that for $a, b \in \Omega^* : (a+b)-(a \oplus b) \in \Omega$ and $(a \times b)-(a \otimes b) \in \Omega$. The operation $div(x, y)$ is a pair of $x \oslash y$ and an error, that we compute using correctly rounded operations on the expression $|x-((x \oslash y) \times y)|/|y|$.

When using above addition and multiplication operations, each operation computes the result as the sum of two floating point values. For operation $op(x, y) = (a, b)$, we say that a is the result of the operation and $e = |b|$ is the error of the operation. We than say that the exact result of the operation lies in the interval $[a-e, a+e]$. In the subsequent operations we use only the value of a in the computation and we use the error separately to get the rigorous bounds on the computed values. The details on how the rigorous bound is estimated varies with each operation and we explain it individually for each rigorous operation in the rest of the paper.

Note that taking the absolute value of b, we discard the sign of the error and we store only the magnitude of the error. It is possible to construct an alternative approach that does not discard the sign of the error, but instead uses two error terms e_{lo} and e_{hi} and represents the quantity as $[a+e_{lo}, a+e_{hi}]$. With such an approach, we can further decrease the size of the error for the cost of higher storage requirements. We have decided for the simpler approach, because having only one error term simplifies our computation. Moreover, in vast majority of our computation, we deal with coefficients of the symbolic polynomials with range $[-1, 1]$. Interval multiplication of the error and interval $[-1, 1]$ makes the sign of error irrelevant.

To illustrate the approach, let us assume we want to compute rigorous bounds for $(x + y) \times z$, where $x, y, z \in \Omega^*$. First, we compute $add(x, y) = (a, b)$ and $e_1 = |b|$.

For simplicity, let us assume that $a \in \Omega^*$, otherwise we would have to handle the overflow or the underflow in the computation. Now we compute $mul(a, z) = (c, d)$ and $e_2 = |d|$. To estimate the total error e such that $(x + y) \times z \in [c - e, c + e]$, we observe that $(x + y) \times z = (a + b) \times z = c + d + b \times z$. We bound the error term $d + b \times z$ using the upward-rounded arithmetic: $e = |d|\overline{+}|b|\overline{\times}|z| = e_2\overline{+}e_1\overline{\times}|z|$.

Now we show, that addition and multiplication operations presented in this section behave superior compared to the similar operations in interval arithmetic. For an operation with exact result x, interval arithmetic computes $\underline{x}, \overline{x} \in \Omega$ such that $x \in [\underline{x}, \overline{x}]$, and $\underline{x}, \overline{x}$ are the closest lower and upper estimates of x in Ω.

Theorem 5 *For above operations add and mul, the computed interval* $[a - e, a + e]$ *has always smaller or equal width to the interval* $[\underline{x}, \overline{x}]$ *computed in interval arithmetic.*

Proof Let x be the exact result of the given arithmetic operation. Since a is the *round to nearest, ties to even* rounded x and $\underline{x}, \overline{x}$ are two closest values to x in Ω, a has to be one of them. Without loss of generality, let $a = \underline{x}$. Since e is the exact error, we have $x = a + e$. Because a is the nearest floating point number to x, $x - \underline{x} \leq \overline{x} - x$, thus $a + e - a \leq \overline{x} - a - e$ and $2e \leq \overline{x} - a = \overline{x} - \underline{x}$. □

As the consequence of the Theorem 5 we use a single floating point value for each numerical quantity and single positive floating point value for an error term throughout this paper. When representing polynomials we decided to use single floating point number for each polynomial coefficient and single error term for the entire polynomial. Comparing this approach to one, where there are interval polynomial coefficients and polynomial operations are evaluated in interval arithmetic, our technique generates less over-approximation due to Theorem 5 and requires less memory to store polynomial coefficients.

Appendix B: Vericomp example number 28

In this section we discuss benchmark number 28 from Section 7 that was not solved by any of the competitive tools. The benchmark equations are:

$$\begin{aligned} x_0' &= x_0^2 x_1^2 - x_0^4 - x_0 x_1^2 + x_0^3 - x_1^4 \\ x_1' &= -x_1^3 + x_0^2 x_1 \\ x_0 &\in [0.2, 1.0];\ x_1 \in [0.6, 1.0] \end{aligned}$$

The reason why this benchmark is hard to solve is the divergent behavior for negative values of variable x_0. The terms $-x_0^4 + x_0^3$ in the differential equation cause fast explosion of the solution states for flows that reach negative values of x_0. This is clearly visible also on the vector field for this example in Fig. 1. The actual solution does not reach this divergent flow, however it gets close to the states that do. As a result, it is crucial to keep overapproximation as small as possible in order to solve this benchmark. Using affine enclosures or zonotope enclosures of the solution

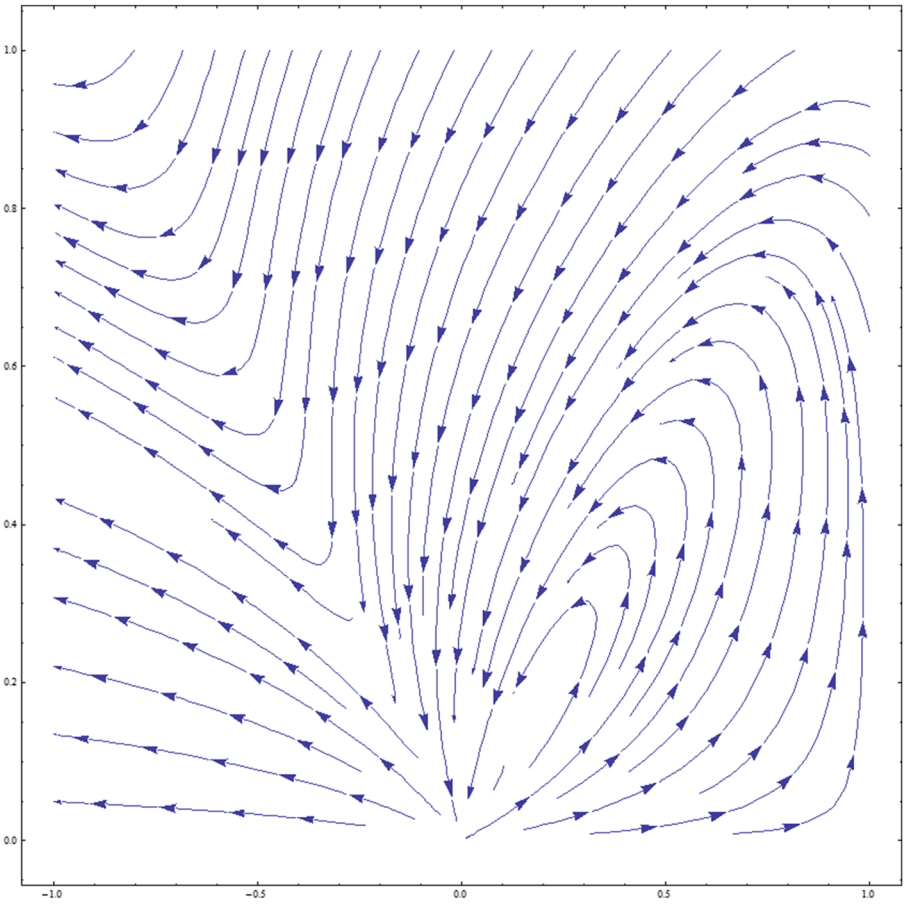

Fig. 1 Vector field of VERICOMP benchmark number 28

set always involve some systematic overapproximation and in systems like this, it leads to failure to solve the benchmark. This benchmark thus shows, that our effort to enclose non-linear solution sets with small overapproximation is important in modern rigorous IVP solvers.

The original VERICOMP benchmark number 28 time bound is $T = 1$, but we present our solution for times up to $T = 10$, while competitive solvers were unable

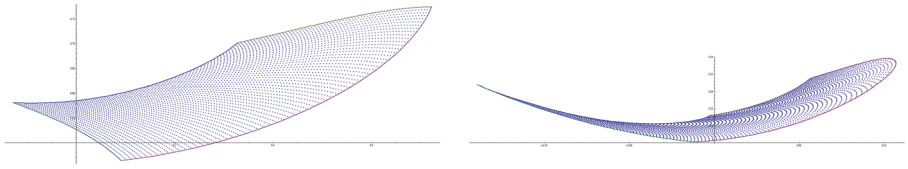

Fig. 2 Solution polynomial and sampled numerical solutions for time $T = 1$ and $T = 10$

to solve the basic version with any setting. In Fig. 2, we show solution polynomial for the benchmark 28 obtained with our tool and the numerical solution for the grid of 64x64 points covering the initial box. The numerical solution was obtained with the function `NDSolve` from the application `Mathematica`. Only the polynomial part of the solution is displayed. The error bounds for the three presented polynomial enclosures in times $T = 1$ and $T = 10$ were 0.00012 and 0.0061 respectively.

References

1. Auer, E., Rauh, A.: Vericomp: a system to compare and assess verified ivp solvers. Computing **94**, 163–172 (2012). doi:10.1007/s00607-011-0178-4
2. Battles, Z., Trefethen, L.N.: An extension of MATLAB to continuous functions and operators. SIAM J. Sci. Comput. **25**, 1743–1770 (2004)
3. Berz, M., Makino, K.: Suppression of the wrapping effect by Taylor model-based verified integrators: long-term stabilization by shrink wrapping. Int. J. Differ. Equ. Appl. **10**(4), 385–403 (2005)
4. Boyd, J.P.: Chebyshev and Fourier Spectral Methods, p. 688. Courier Dover Publications, New York (2001)
5. Brisebarre, N., Joldeş, M.: Chebyshev interpolation polynomial-based tools for rigorous computing. In: Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation, ISSAC '10, pp. 147–154. ACM, New York (2010)
6. Camacho, C., De Figueiredo, L.H.: The dynamics of the jouanolou foliation on the complex projective 2-space. Ergodic Theory Dyn. Syst. **5**(21), 757–766 (2001)
7. Clenshaw, C.W.: A note on the summation of Chebyshev series. MTAC **9**, 118–120 (1955)
8. Corliss, G.F.: Guaranteed error bounds for ordinary differential equations. In: Theory of Numerics in Ordinary and Partial Differential Equations, pp. 1–75. Oxford University Press (1994)
9. De Figueiredo, L.H., Stolfi, J.: Affine arithmetic: concepts and applications. Numer. Algorithm. **37**(1–4), 147–158 (2004)
10. Dekker, T.: A floating-point technique for extending the available precision. Numer. Math. **18**, 224–242 (1971/72)
11. Dzetkulič, T.: ODEIntegrator. http://odeintegrator.sourceforge.net, 2012. Software package
12. Eble, I.: ber Taylor-Modelle. PhD thesis, Institut fr Angewandte und Numerische Mathematik (2007)
13. Henzinger, T.A., Horowitz, B., Majumdar, R., Wong-Toi, H.: Beyond HyTech: hybrid systems analysis using interval numerical methods. In: Lynch, N., Krogh, B. (eds.) Proceedings of the HSCC'00, vol. 1790 of LNCS, pp. 130–144, Springer (2000)
14. IEEE Standards Board: IEEE standard for binary floating-point arithmetic. Technical report, The Institute of Electrical and Electronics Engineers. Technical Report IEEE Std 754–1985 (1985)
15. Kühn, W.: Rigorously computed orbits of dynamical systems without the wrapping effect. Computing **61**(1), 47–67 (1998)
16. Lohner, R.J.: Enclosing the solutions of ordinary initial and boundary value problems. In: Computer Arithmetic: Scientific Computation and Programming Languages, pp. 255–286. Teubner, Stuttgart (1987)
17. Lohner, R.J.: Computations of guaranteed enclosures for the solutions of ordinary initial and boundary value problems. In: Cash, J., Gladwell, I. (eds.) Computational Ordinary Differential Equationas, pp. 425–435. Clarendon Press, Oxford (1992)
18. Makino, K., Berz, M.: Taylor models and other validated functional inclusion methods. Int. J. Pur. Appl. Math. **4**(4), 379–456 (2003)
19. Makino, K., Berz, M.: Suppression of the wrapping effect by Taylor model-based verified integrators: long-term stabilization by preconditioning. Int. J. Differ. Equ. Appl. **10**(4), 353–384 (2005)
20. Makino, K., Berz, M.: Suppression of the wrapping effect by Taylor model-based verified integrators: the single step. Int. J. Pur. Appl. Math. **36**(2), 175–197 (2006)
21. Makino, K., Berz, M.: Rigorous integration of flows and ODEs using Taylor models. Symb. Numer. Comput., 79–84 (2009)
22. Moore, R.E.: Interval analysis. Prentice hall, Englewood cliffs (1966)

23. Moore, R.E., Kearfott, R.B., Cloud, M.J.: Introduction to interval analysis. SIAM (2009)
24. Nedialkov, N.S., Jackson, K.R., Corliss, G.F.: Validated solutions of initial value problems for ordinary differential equations. Appl. Math. Comput. **105**, 21–68 (1999)
25. Ratschan, S., She, Z.: Safety verification of hybrid systems by constraint propagation-based abstraction refinement. ACM Trans. Embed. Comput. Syst. **6**(1) (2007)
26. Rauh, A., Hofer, E.P., Valencia-ivp, E.Auer.: A comparison with other initial value problem solvers. In: Proceedings of the 12th GAMM - IMACS International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics, SCAN '06, pp. 36–. IEEE Computer Society, Washington (2006)
27. Tucker, W.: A rigorous ODE solver and Smale's 14th problem. Found. Comput. Math. **2**(1), 53–117 (2002)
28. Wittig, A., Berz, M.: Rigorous high precision interval arithmetic in COSY INFINITY. In: Proceedings of the Fields Institute (2009)