Preconditioning of Taylor models, implementation and test cases

Florian Bünger

Institute for Reliable Computing, Hamburg University of Technology
Am Schwarzenberg-Campus 3, D-21073 Hamburg, Germany

florian.buenger@tuhh.de

Received January 14, 2020; Revised April 3, 2020; Published January 1, 2021

Abstract: Makino and Berz introduced the Taylor model approach for validated integration of initial value problems (IVPs) for ordinary differential equations (ODEs). Especially, they invented preconditioning of Taylor models for stabilizing the integration and proposed the following different types: parallelepiped preconditioning (with and without blunting), QR preconditioning, and curvilinear preconditioning. We review these types of preconditioning and show how they are implemented in INTLAB’s verified ODE solver verifyode by stating explicit MATLAB code. Finally, we test our implementation with several examples.

Key Words: ordinary differential equations, initial value problems, Taylor models, QR preconditioning, curvilinear preconditioning, parallelepiped preconditioning, blunting

1. Introduction

Preconditioning is a general concept in large parts of numerical mathematics. It is also widely used in the area of reliable (verified) computing in which numerical results are calculated along with rigorous error bounds that cover all numerical as well as all rounding errors of a computer, meaning that a mathematically exact result is proved to be contained within these rigorous bounds.

For verified integration of initial value problems (IVPs) for ordinary differential equations (ODEs) preconditioning goes back to Lohner’s fundamental research [20] in which he particularly introduced his famous parallelepiped and QR methods to reduce the well-known wrapping effect from which naive verified integration methods for IVPs necessarily suffer. A detailed quantitative analysis of such anti-wrapping schemes including Lohner’s QR method was given by Nedialkov and Jackson in [22].

Makino and Berz invented and used so-called Taylor models for verified integration of IVPs and implemented their Taylor model approach in their software package COSY INFINITY [1]. In analogy to Lohner’s method Makino and Berz [7] invented preconditioning of Taylor models and proposed the following three types: parallelepiped preconditioning (with and without blunting), QR preconditioning, and curvilinear preconditioning. In addition, they also mentioned a fourth, very simple type of preconditioning named identity preconditioning.

Recently, we implemented the Taylor model approach for solving IVPs in INTLAB [25]. In the former release V11, parallelepiped and QR preconditioning were provided. The current, new release V12 additionally contains a variant of curvilinear preconditioning and simple identity preconditioning. Also blunted parallelepiped preconditioning has changed compared to V11.
In this technically oriented report all those types of preconditioning are reviewed and their concrete MATLAB/INTLAB implementation in our verified ODE solver verifyode is explained in detail. The focus is completely on preconditioning, the overall Taylor model approach for solving IVPs is not repeated here again. For that we refer to [4–7, 9, 13, 21, 23] and the references therein.

The paper is organized as follows. In Section 2 the definition and notation of Taylor models is shortly introduced. In Section 3 the implementation of preconditioning in INTLAB is described. Section 4 contains computational results for several IVPs solved with INTLAB’s verified ODE solver verifyode using different types of preconditioning.

2. Taylor models
Let \( n, k, m \in \mathbb{N} := \{1, 2, \ldots \} \) be fixed. As usual, the set of closed real intervals is denoted by \( \mathbb{R} \); more generally \( \mathbb{R}^n \) stands for the set of real interval vectors of length \( n \). An interval vector \( I \in \mathbb{R}^n \) is usually denoted in inf-sup notation \( I = [\mathfrak{a}, \mathfrak{b}] \) where \( \mathfrak{a}, \mathfrak{b} \in \mathbb{R}^n \) satisfy \( I = [\mathfrak{a}_1, \mathfrak{b}_1] \times \cdots \times [\mathfrak{a}_n, \mathfrak{b}_n] \).

We will consider vectors \( p(x) = (p_1(x), \ldots, p_n(x)) \) of length \( n \) where each component \( q(x) = p_i(x) \) is a real multivariate polynomial in \( k \) unknowns \( x_1, \ldots, x_k \) of degree less than or equal to \( m \). That is

\[
q(x) = \sum_{\alpha, |\alpha| \leq m} q_\alpha x^{\alpha}
\]

where \( \alpha = (\alpha_1, \ldots, \alpha_k) \in \mathbb{N}^k \) is a multiindex, \( |\alpha| := \sum_{i=1}^k \alpha_i \), \( x^{\alpha} := \prod_{i=1}^k x_i^{\alpha_i} \), and the \( q_\alpha \in \mathbb{R} \) are real polynomial coefficients. A domain \( D = \times_{c\in D} [u_c, v_c] \in \mathbb{R}^k \) and a center or centering point \( c \in D \) are fixed and

\[
p(D - c) = \{ p(x - c) = (p_1(x - c), \ldots, p_n(x - c)) \mid x \in D \} \subseteq \mathbb{R}^n
\]

is called the image of \( p \) on \( D \) centered at \( c \). Such multidimensional, multivariate polynomial images are used for validated enclosures in the context of Taylor models. Their great advantage is that they can enclose non-convex sets with curved boundaries without much overestimation. This is not possible in interval arithmetic or affine arithmetic, where the former uses interval vectors, i.e., axis parallel boxes, and the latter convex polytopes for enclosing sets. To cover ubiquitous floating-point rounding and degree truncation errors, a so-called remainder or error interval vector \( E \in \mathbb{R}^n \) is added enlarging the image \( p(D - c) \) to

\[
p(D - c) + E = \{ y + e \mid y \in p(D - c), e \in E \}
\]

which, for better distinction, will be called the range of the data \( p, D, c, E \). The aim will always be to keep the size/volume of the remainder \( E \) small compared to that of the image \( p(D - c) \) so that rendering by \( E \) does not change its shape essentially. Now, an \( m \)-th order Taylor model vector of length \( n \) with \( k \)-dimensional domain given by the data \( p, D, c, E \) is defined as

\[
p + E := \{ f \in C(D, \mathbb{R}^n) \mid \forall x \in D : f(x) \in p(x - c) + E \}
\]

where \( C(D, \mathbb{R}^n) \) denotes the set of all continuous functions \( f : D \to \mathbb{R}^n \). For ease of presentation we will just speak of Taylor models \( p + E \) if \( m, n, k, D, c, E \) are clear from the context. By (1) a Taylor model \( p + E \) is a set of continuous functions with domain \( D \) and values in \( \mathbb{R}^n \) which at each point \( x \in D \) deviate from \( p(x - c) \) by some \( e \in E \).

For \( n = 1 \), arithmetic operations \(+, -, \times, \div\), standard functions like exp, log, sin, cos, etc., and also integration with respect to one of the \( k \) variables can be defined on the set of all Taylor models with common, fixed \( D, c, m \). For details on these basic operations we refer to [9, 13, 21]. For \( n > 1 \), vector operations like \(+, -\) immediately carry over from the one-dimensional case.

As a simple example, a Taylor model \( y \) with polynomial part

\[
p(x_1, x_2) = 2 + x_1 - x_2 + 3x_1^2x_2^3
\]

of degree \( d = 5 \), with domain \( D := [-1, 1]^2 \), centering point \( c := (0, 0) \in \mathbb{R}^2 \), and error interval \( E := [-10^{-6}, 10^{-6}] \) can be created as follows in INTLAB:
Then, the Taylor model will be displayed in MATLAB’s command window as follows:

\[
\begin{array}{cccccccc}
\text{dim} & \text{order} & \text{type} & \text{iv_mid} & \text{iv_rad} & \text{im_inf} & \text{im_sup} \\
2 & 7 & 0 & 0 & 1.0001\times10^{-6} & -3 & 7 \\
\end{array}
\]

Here \( \text{dim} = 2 \) is the dimension of the domain \( D \), that is the number of unknowns \( x_1, x_2 \). \( \text{order} = 7 \) is the quite arbitrarily chosen upper degree bound \( m \geq d = 5 \), and \( \text{iv_mid} = 0, \text{iv_rad} = 1.0001\times10^{-6} \) is an interval enclosure of the remainder \( E \) in midpoint radius notation. A quite large enclosure \( \text{im_inf}, \text{im_sup} = [-3, 7] \) of the polynomial image \( p(D - c) = p(D) = [-3, 5] \) is computed automatically. The true range \( R := [-3, 5] + E \) of \( y \) is very much overestimated by \( [-3, 7] + E \). In practice, Taylor models with thin ranges of diameter much less than 1 are typical. Only for ease of presentation we chose the stated data with wide range.

Taylor models of \( \text{type} = 0 \) are the default, Taylor models of \( \text{type} = 1 \) are used for solving ODEs. As we will see in the next section, the latter have special domains and center points of the form \( D = [-1, 1] \times \cdots \times [-1, 1] \times [t_1, t_2], c = (0, \ldots, 0, t_1) \) where \( [t_1, t_2] \) is the time domain, i.e., the domain of the variable \( t \) with respect to which the integration is performed from \( t_1 \) to \( t_2 \).

Technically, the components of a Taylor model object in INTLAB read as follows:

\[
\text{struct} \begin{array}{l}
\text{ys} = \text{struct}(y) \\
\text{ys.dim} = \text{ys.dim} \quad \% \text{number of unknowns} \\
\text{ys.center} = \text{ys.center} \quad \% \text{center point c} \\
\text{ys.domain} = [\text{ys.domain.inf}, \text{ys.domain.sup}] \quad \% \text{domain D} \\
\text{ys.order} = \text{ys.order} \quad \% \text{degree bound m} \\
\text{ys.monomial} = \text{ys.monomial} \quad \% \text{polynomial exponents M} \\
\text{ys.coefficient} = \text{ys.coefficient} \quad \% \text{polynomial coefficients p} \\
\text{ys.interval} = \text{ys.interval} \quad \% \text{error interval E} \\
\text{ys.type} = \text{ys.type} \quad \% \text{Taylor model type} \\
\text{ys.image} = \text{ys.image} \quad \% \text{interval enclosure of the image p(D-c)}
\end{array}
\]

This code produces the following output which clarifies the internal structure.
3. Preconditioning

Let an interval initial value problem for an $n$-dimensional IVP be given:

$$
y' = f(t,y) \in \mathbb{R}^n, \quad t \in [t_0, t_f]
$$

$$
y(t_0) \in Y_0 = [\mathbf{y}, \mathbf{y}] \in \mathbb{R}^n.
$$

The problem is solved stepwise on subintervals $[t_i, t_{i+1}]$ where the time grid $t_0 < t_1 < \ldots < t_N = t_f$ is chosen by some automatic step size control. Without further ado we state that the Taylor model approach with preconditioning for validated integration of (2) and (3), see [7, 9, 23], computes for each $[t_i, t_{i+1}]$ three Taylor model vectors

$$
p^{(i)}_l(x), \quad p^{(i)}_r(x), \quad \hat{p}^{(i)}(x,t) + \hat{E}^{(i)}(x,t)
$$

called left, right, and integrated left Taylor model, respectively. They have length $n$, domain $D := [-1,1]^n \times [t_i, t_{i+1}]$, and centering point $c := (0, \ldots, 0, t_i) \in \mathbb{R}^{n+1}$ where $x = (x_1, \ldots, x_n)$. Thus, the $n$ space-like variables $x_1, \ldots, x_n$ have domain $[-1,1]$ and centering point zero, and the time variable $t$ has domain $[t_i, t_{i+1}]$ with centering point $t_i$ which is the left boundary point of the time domain. The left and right Taylor models are time-independent, wherefore the time variable $t$ is skipped in the notation of the polynomial parts $p^{(i)}_l(x)$ and $p^{(i)}_r(x)$. Abbreviating $B := [-1,1]^n$ the three Taylor models stated in (4) fulfill:

$$
E^{(i)}_l \approx 0 \in \mathbb{R}^n 
$$

$$
p^{(i)}_r(B) + E^{(i)}_r \subseteq B
$$

$$
y(t_i) \in [(p^{(i)}_l(x) + E^{(i)}_l) \circ (p^{(i)}_r(x) + E^{(i)}_r)](B)
$$

$$
\forall t \in [t_i, t_{i+1}] : \quad y(t) \in [(\hat{p}^{(i)}_l(x, t-t_i) + \hat{E}^{(i)}_l) \circ (p^{(i)}_r(x) + E^{(i)}_r)](B)
$$

where $y(t)$ is a solution of the IVP (2), (3) for any $y(t_0) \in Y_0$. Here Taylor model concatenation

$$
c(x) + K = (a(x) + I) \circ (b(x) + J) := a(b_1(x) + J_1, \ldots, b_n(x) + J_n) + I
$$

is simply defined by evaluating the right-hand side on the basis of Taylor model multiplication/exponentiation and summation. Note that inserting $b(x) + J$ into $a(x) + I$ only makes sense if the range of $b(x) + J$ is contained in the domain of $a(x) + I$. This is the reason why (6) must hold true. Property (7) means that the Taylor model range of the concatenation of the left and the right Taylor model is a validated enclosure of the ODE flow at the time grid point $t_i$. Similarly, (8) says that the range of the concatenation of the integrated left Taylor model and the right Taylor model yields validated enclosures at every fixed $t$ in the subinterval $[t_i, t_{i+1}]$.

The Taylor models in (4) are iteratively constructed as follows: For $i = 0$ define

$$
p^{(0)}_l(x) := y_0 + x \text{ diag}(r) \quad \text{and} \quad p^{(0)}_r(x) = \text{id}(x) = x
$$

where $y_0 = \frac{1}{2}(\mathbf{y} + \mathbf{y})$ and $r = \frac{1}{2}(\mathbf{y} - \mathbf{y})$ are the midpoint and radius vectors of the initial value interval vector $Y_0$, cf. (3). The left and right remainders $E^{(i)}_l, E^{(i)}_r$ are zero and therefore omitted. Then, $p^{(0)}_r(B) = B$, $p^{(0)}_l(x) \circ p^{(0)}_r(x) = p^{(0)}_l(x)$, and

$$
p^{(0)}_r(B) = \{y_0 + \text{ diag}(r)x \mid x \in B = [-1,1]^n\}
$$

$$
= \{(y_{0,1} + r_1x_1, \ldots, y_{0,n} + r_nx_n) \mid x_1, \ldots, x_n \in [-1,1]\} = Y_0.
$$

Thus, (5), (6), and (7) are fulfilled for $i = 0$. Next, suppose by induction that for $i \in \{0, \ldots, N - 2\}$,

$$
a(x) + I := p^{(i)}_l(x) + E^{(i)}_l \quad \text{and} \quad b(x) + J := p^{(i)}_r(x) + E^{(i)}_r
$$

are already constructed such that (5), (6), and (7) are fulfilled for $i$. The integrated time-dependent left Taylor model

\[\text{5}\]
Choosing which are chosen such that (6) holds true for \(i\) for some invertible function \(g\) without blowing up the remainders. It is not clear at all how a most favorable function \(g\) can be evaluated in Taylor model arithmetic without causing too much overestimation/wrapping, i.e., without blowing up the remainders. It is not clear at all how a most favorable function \(g(x)\) can be found. We emphasize that the evaluation order of the concatenation chain (18) is of crucial importance for an implementation. In order that preconditioning deserves its name, the left concatenation

\[
\hat{a}(x, t) + \tilde{I} := \hat{p}^{(i)}(x, t) + \tilde{E}^{(i)}
\]

is obtained from a validated integration of the time-independent left Taylor model \(a(x) + I\). For example, this can be done by a validated Picard iteration to which Schauder’s fixed point theorem is applied to prove (8). Although this is a crucial part in general, we skip the details here and just formally write

\[
\hat{a}(x, t) + \tilde{I} = \mathcal{F}(a(x) + I)
\]

for this validated integration process and notice that

\[
\forall t \in [t_i, t_{i+1}]: y(t) \in [(\hat{a}(x, t - t_i) + \tilde{I}) \circ (b(x) + J)](B)
\]

holds true for any solution \(y(t)\) of the IVP (2), (3). This is (8). One key point of preconditioning is to keep the remainder \(I = E^{(i)}_t\) small as required in (5) in order to diminish the well-known wrapping from which the integration operator \(\mathcal{F}\) may suffer. Now, abbreviate

\[
\hat{a}(x) + \tilde{I} := \hat{a}(x, t_{i+1} - t_i) + \tilde{I}
\]

so that (13) evaluated for \(t = t_{i+1}\) gives

\[
y(t_{i+1}) \in [(\hat{a}(x) + \tilde{I}) \circ (b(x) + J)](B).
\]

Choosing

\[
p^{(i+1)}(x) + E^{(i+1)}_t := \hat{a}(x) + \tilde{I}
\]

and leaving

\[
p^{(i+1)}_r(x) + E^{(i+1)}_r := b(x) + J = p^{(i)}_r(x) + E^{(i)}_r = ... = p^{(0)}_r(x) + E^{(0)}_r = x
\]

gives the so-called naive Taylor model method in which the right Taylor model stays the identity and can therefore be skipped. Using (15), these trivial settings fulfill (6) and (7) for \(i + 1\), however they do not make much sense for long term integration since the remainder \(\tilde{I} = E^{(i+1)}_t\) may not stay close to zero as claimed in (5). As stated before, this may possibly cause a strong wrapping effect during evaluation of the integration operator \(\mathcal{F}\), cf. (12). Thus, a different, nontrivial factorization of

\[
(\hat{a}(x) + \tilde{I}) \circ (b(x) + J)
\]

is desirable. A natural attempt is to write

\[
g(x) \circ [g^{-1}(x) \circ (\hat{a}(x) + \tilde{I}) \circ (b(x) + J)]
\]

for some invertible function \(g(x)\) and to define

\[
p^{(i+1)}_r(x) + E^{(i+1)}_r := g(Sx) \quad \text{where } E^{(i+1)}_t \approx 0 \in \mathbb{R}^n
\]

\[
p^{(i+1)}_r(x) + E^{(i+1)}_r := S^{-1}g^{-1}(x) \circ (\hat{a}(x) + \tilde{I}) \circ (b(x) + J).
\]

Here \(S = \text{diag}(s_1, \ldots, s_n)\) is a diagonal scaling matrix with positive real scaling factors \(s_1, \ldots, s_n\), which are chosen such that (6) holds true for \(i + 1\). The first problem is that \(g(x)\) as well as \(g^{-1}(x)\) must somehow be represented as vectors of multivariate polynomials which may become expensive if \(g(x)\) is complicated. The next even harder problem and the main point of preconditioning is to find \(g(x)\) such that the right factor in (17),

\[
g^{-1}(x) \circ (\hat{a}(x) + \tilde{I}) \circ (b(x) + J),
\]

can be evaluated in Taylor model arithmetic without causing too much overestimation/wrapping, i.e., without blowing up the remainders. It is not clear at all how a most favorable function \(g(x)\) can be found. We emphasize that the evaluation order of the concatenation chain (18) is of crucial importance for an implementation. In order that preconditioning deserves its name, the left concatenation
From the right-hand side it is clear that the preconditioned \( q \) suffers from strong overestimation. This also clarifies that an evaluation order interval arithmetic so that it is of crucial importance to choose \( q \) such that this action does not suffer from strong overestimation. This also clarifies that an evaluation order

\[
(q^{-1}(x) \circ (\hat{a}(x) + \hat{I})) \circ (b(x) + J) = (q(x) + E) \circ (b(x) + J) = q(b(x) + J) + E.
\]

From the right-hand side it is clear that the preconditioned \( q(x) \) acts especially on the remainder \( J \). Due to the nature of Taylor model arithmetic, the action on remainders mainly reduces to naive interval arithmetic. For example, taking the square effect is the repeated application of a 45 degree rotation leads to minimal remainder growth is still not obvious. Makino and Berz [7] decomposed into the linear part \( Ax \) with \( n \times n \)-matrix \( A \), the nonlinear part \( \zeta(x) \) containing all terms \( x^\alpha \) of degree \( |\alpha| \geq 2 \), and the constant part \( c \in \mathbb{R}^n \). They made the reasonable assumption that the linear part \( Ax \) has the leading effect; however, that might be wrong for specific ODEs. Under this assumption, the crucial action of \( \hat{a}(x) \) on the remainder \( J \) of the right Taylor model reduces to the evaluation of the matrix interval vector product \( AJ \) in interval arithmetic. A typical worst case scenario that one should always have in mind when talking about the wrapping effect is the repeated application of a 45 degree rotation \( A \) to a two-dimensional (error) interval vector \( J \) in interval arithmetic. For example, taking the square \( J = J_0 := [-0.1,0.1]^2 \). Fig. 1 shows the growth of the interval vectors \( J_{j+1} := AJ_j \) for \( j = 1,2,\ldots,6 \), when the evaluation of \( AJ \) is performed in interval arithmetic. Rotating six times transforms \( J_0 \) (yellow box) to \( J_6 \) (black box) which is obviously a large overestimation of the true result \( A^6J_0 = J_0 \).

![Diagram showing the wrapping effect.](image)

**Fig. 1.** Wrapping effect.

Coming back to preconditioning, the aim is to choose a well-conditioned matrix \( G \) such that the preconditioned matrix \( \hat{A} := G^{-1}A \) allows to evaluate \( \hat{A}J \) in interval arithmetic without much overestimation. Makino and Berz [7] proposed the following three choices for the preconditioner \( G \).
1. **parallelepiped preconditioning**: Here $G := A$ is chosen so that $\hat{A} := G^{-1}A = I_n$ is the identity matrix. Then $\hat{A}J = J$ does not cause any remainder growth. The drawback of this method is that verified inversion of $A$ introduces large overestimation if $A$ has a large condition number. To overcome this drawback Makino and Berz [4, 6] proposed a so-called blunted version of parallelepiped preconditioning where $G := \hat{A}$ is a modification of $A$ such that the angles between the column vectors of $A$ are slightly enlarged (blunted) so that the condition number of $\hat{A}$ becomes less than that of $A$ and verified inversion becomes more accurate. Such a regularization was already suggested by Lohner [20], p. 51, paragraph 2, but he did not implement it in AWA. Details on blunting are given in Subsection 3.1.

2. **QR preconditioning**: Here $G := Q$ is the orthogonal part $Q$ of a QR decomposition of the matrix $A := AP$ where $P$ is a permutation matrix which sorts the columns of $A$ by decreasing Euclidean length. This reordering was proposed by Lohner [20], pp. 49–51, to reduce the overestimation in the interval evaluation of $\hat{A}J = (G^{-1}A)J = (RP^T)J$. The benefit is shortly demonstrated by a simple two-dimensional example in Subsection 3.2. For a detailed, quantitative analysis of Lohner’s QR factorization method we refer to [22].

3. **curvilinear preconditioning**: Here $G := Q$ is the orthogonal part $Q$ of a QR decomposition of a matrix the columns of which built a so-called curvilinear basis of prescribed depth $k$ of the ODE flow with respect to a centered reference trajectory. Details are postponed to Subsection 3.3. For autonomous linear systems $y' = Ay$, QR and curvilinear preconditioning are, for $t \to \infty$, asymptotically similar, see [7], Propositions 12 and 16. Thus, both kinds of preconditioning should in general exhibit some similar behavior.

-1. **identity preconditioning**: Here always $G := I_n$ is the $n \times n$ identity matrix. This simple form of preconditioning is in general inferior to the other types above. Especially for long term integration it may not succeed at all. However, for short term integration it may give the same or at least similar results with better performance.

Having chosen the preconditioner $G$, the left and right Taylor models for the next integration step on $[t_{i+1}, t_{i+2}]$ as defined in (16), (17) read for $g(x) := Gx + c$ and $g^{-1}(x) = G^{-1}(x - c)$ as follows:

$$p^{(i+1)}(x) + E^{(i+1)} = S \left( \left[ G^{-1}(\tilde{a}(x) - c + \hat{I}) \right] \circ (b(x) + J) \right)$$

$$= S \left( \left[ (G^{-1}A)x + G^{-1}\zeta(x) + G^{-1}\tilde{I} \right] \circ (b(x) + J) \right)$$

$$p^{(i+1)}(x) + E^{(i+1)} := GS^{-1}x + c \quad (21)$$

The remainder $E^{(i+1)}$ in (22) covers rounding errors only so that (5) holds true for $i + 1$. By construction, also (7) is fulfilled for $i + 1$. As mentioned before, $(G^{-1}A)x$ is assumed to be the leading term in (21) in view of the action on the global remainder $J$. The higher order terms $G^{-1}\zeta(x)$ and the action of $G^{-1}$ on the supposedly small local remainder $\hat{I}$ are assumed to be negligible. This might, of course, be wrong for specific ODEs. Finally, we remark that it is numerically advantageous to choose the diagonal scaling matrix $S$ in (21) such that (6) holds true almost sharply, i.e., such that $B = [-1, 1]^n$ is a tight interval enclosure of the (possibly nonconvex) range $p^{(i+1)}(B) + E^{(i+1)}$ of the new right Taylor model. We want to point out clearly that the main performance bottleneck in a concrete implementation is the computation of the Taylor model concatenation in (21). Especially for higher dimensional systems this may lead to impracticality of the method.

Looking at (21) and (22) for simple identity preconditioning, i.e., $G := I_n$, shows that the only wrapping diminishing effect is the shift of the remainder $\hat{I}$ to the right Taylor model so that integration of the new left Taylor model $S^{-1}x + c$ will suffer less from overestimation caused by interval arithmetic. Note that therefore identity preconditioning is still superior to the naive Taylor model method.

Before describing our implementation of preconditioning, a connection between identity preconditioning and so-called error parametrization as proposed by Dzetkulić [12] shall be drawn. This error
parameterization introduces another \( n \) variables \( z_1, \ldots, z_n \) which, like \( x_1, \ldots, x_n \), vary in \([-1,1]\) with centering point zero. Then, the polynomial parts of all Taylor models have the form \( p(x,z,t) \).

Like in the naive Taylor model method the ODE flow is, at first glance, not divided into left and right Taylor models so that the right Taylor models \( p^{(i)}_r(x) + E^{(i)}_r \) disappear in (4) to (8), and the subscript \( \ell \) is dropped in the notation of the left Taylor models \( p^{(i)}_l(x) + E^{(i)}_l \) and \( \hat{p}^{(i)}_l(x,t) + \hat{E}^{(i)}_l \). Thus, according to (10), (11), they become

\[
a(x,z) + I := p^{(i)}(x,z) + E^{(i)} \quad \text{and} \quad \hat{a}(x,z,t) + \hat{I} := \hat{p}^{(i)}(x,z,t) + \hat{E}^{(i)}.
\]

Moreover, a so-called back-substitution Taylor model

\[
p^{(i)}_s(x,z) + I^{(i)}_s := b(x,z) + J
\]

is defined where the polynomial part \( p^{(i)}_s(x,z) = b(x,z) \) takes values in \( \mathbb{R}^n \) and \( I^{(i)}_s = J \in \mathbb{R}^n \). The construction of \( b(x,z) + J \) is done such that its interval enclosure is contained in, and almost equals the cube \( B = [-1,1]^n \). In the beginning of integration, for \( i = 0 \), \( a(x,z) + I = a(x) + I \) does not depend on \( z \), cf. (9), and

\[
p^{(0)}_s(x,z) + I^{(0)}_s := \text{id}(z) = z \quad \text{with} \quad I^{(0)}_s := 0 \in \mathbb{R}^n
\]
does not depend on \( x \). After one integration step the integrated, time-independent Taylor model

\[
\hat{a}(x,z) + \hat{I} = \hat{a}(x,z,t_{i+1} - t_i) + \hat{I},
\]

cf. (14), is additively split into two parts

\[
\hat{a}(x,z) + \hat{I} = \hat{a}^+(x) + \left( \hat{a}^*(x,z) + \hat{I} \right)
\]

where \( \hat{a}^+(x) \) contains the terms of \( \hat{a}(x,z) \) which are independent from \( z \), and

\[
\hat{a}^*(x,z) := \hat{a}(x,z) - \hat{a}^+(x)
\]
contains all terms depending on \( z \) as well as the remainder \( \hat{I} \). Note that \( \hat{a}^+(x) \) particularly contains the constant part \( c = \hat{a}(0,0) \), cf. (20). Now, like in (21) and (22), a diagonal scaling matrix \( S \) is introduced such that an interval enclosure of the range of

\[
p^{(i+1)}_s(x,z) + I^{(i+1)}_s := S \left( \left[ \hat{a}^*(x,z) + \hat{I} \right] \circ (b(x,z) + J) \right) = S \left( \hat{a}^*(x,b(x,z) + J) + \hat{I} \right)
\]
is contained in, and almost equals \( B = [-1,1]^n \). From (23) the naming “back-substitution” for the Taylor model \( b(x,z) + J \) becomes clear because it is substituted for \( z \) in \( \hat{a}^*(x,z) \). Then,

\[
p^{(i+1)}(x,z) + E^{(i+1)} := \hat{a}^+(x) + S^{-1}z
\]
represents the initial value set for the next integration step where, like in (22), the remainder \( E^{(i+1)} \) covers rounding errors only and is therefore almost zero. As a consequence, the integration (12) will suffer less from wrapping.

In the special case of point initial values, all polynomial parts are independent of \( x \), i.e.,

\[
\hat{a}^+(x) = \hat{a}^+(0) = c
\]
only contains the constant part. Thus, the error parametrization (23), (23) reduces to identity preconditioning (21), (22) where \( z \) takes the place of \( x \) (\( x \) can actually be dropped) and the back-substitution Taylor model \( b(x,z) + J = b(z) + J \) agrees with the right Taylor model.

Because of this accordance of error parametrization and identity preconditioning for point initial value problems we do not fully agree with the statements in [12], p. 193, which say that preconditioning, in contrast to error parametrization, takes only linear parts into account and ignores the nonlinear ones. This is simply not true as the right Taylor models contain these nonlinear parts.
which are stepwise resubstituted like it is done with the back-substitution Taylor models in the error parametrization method. The difference is that the latter has more variables at hand to organize the error shifting. One may even ask for a suitable way to incorporate non-identity preconditioners $G$ into error parameterization (23) to obtain better results. However, this question is not pursued here.

Now, we return to preconditioning. Listing 1 contains the concrete MATLAB/INTLAB implementation in our verified ODE solver verifyode. The code is commented and main parts will be related to the previous description now.

The first input parameter $a$ in line 1 is the Taylor model $\tilde{a}(x) + \tilde{I}$ defined in (14). The second input parameter and $b$ is the time-independent right Taylor model $b(x) + J$, cf. (10). The third input parameter base_tm represents the time-independent identity Taylor model $\text{id}(x) = x$. The fourth input parameter options contains global options that can be specified by the user, like the type of preconditioning that shall be used. The fifth input parameter $Q_{\text{CL}}$ is the precomputed preconditioning matrix $G$ in case of curvilinear preconditioning. Its derivation will be explained in Subsection 3.3. The return parameters $y_1$ and $y_r$ are the left and right Taylor models $p^{(i+1)}_t(x) + E^{(i+1)}_t$ and $p^{(i+1)}_r(x) + E^{(i+1)}_r$ for the next integration step on $[t_{i+1}, t_{i+2}]$, cf. (21), (22).

The variable $n$ defined in line 6 is the dimension of the ODE which equals the number of space-like variables $x_1, \ldots, x_n$. If parallelepiped or QR preconditioning is switched on, then lines 8, 9 provide the $n \times n$-matrix $A$ of the linear part of $\tilde{a}$, cf. (20).

If QR preconditioning is switched on, then lines 13–19 compute an approximately orthogonal floating-point matrix $Q$ of a $QR$ decomposition of the matrix $\hat{A}$ containing the columns of $A$ ordered by decreasing Euclidean length. The factor $Q$ is chosen such that the factor $R$ has nonnegative diagonal entries.

If parallelepiped preconditioning was chosen, then $Q := A$ is taken, cf. line 29. However, if the condition number of $A$ is larger than the heuristic bound $10^6$, then automatically a blunted version of $A$ is taken in order to decrease the condition number, cf. lines 24–27, even though blunting is not explicitly switched on. The code for blunting will be described in Subsection 3.1. If blunting is switched on permanently, then line 22 is executed. If finally the condition number of $Q$ still exceeds the large heuristic bound $10^{16}$, then $Q$ is replaced by the $n \times n$ identity matrix, cf. lines 32–35. This is particularly important for point initial values where in the very first integration step the matrix $A$ is always zero.

For curvilinear preconditioning $Q := Q_{\text{CL}}$ is taken, cf. line 37, as stated above, and in case of identity preconditioning $Q := \text{eye}(n)$ is chosen where $\text{eye}(n)$ is the $n \times n$ identity matrix, cf. line 40.

Line 44 computes a validated inverse $G^{-1} = Q_{\text{inv}}$ of the preconditioning matrix $G := Q$ by using INTLAB’s function $\text{inv}()$. Line 46 supplies the constant part $c$ of $\tilde{a}$, cf. (20), and line 47 yields

$$a_0 := (\tilde{a}(x) + \tilde{I}) - c = Ax + \tilde{c}(x) + \tilde{I}.$$  

Line 48 performs the Taylor model concatenation

$$\left[ G^{-1}(\tilde{a}(x) - c + \tilde{I}) \right] \circ (b(x) + J) = [Q_{\text{inv}}*a_0] \circ b =: y_r$$

stated in (21). Line 49 calculates an interval enclosure $I$ of the Taylor model range of the intermediate result $y_r$. The function $\text{iv\_plus}$ performs verified interval summation and is faster than INTLAB’s object oriented standard summation. Line 50 computes a midpoint radius enclosure

$$I \subseteq [m - s, m + s]$$

in order to derive the diagonal scaling matrix $S = \text{diag}(s_1, \ldots, s_n)$ needed in (21), (22). The final results for the new right and left Taylor models $y_r$ and $y_1$ are obtained in lines 53 and 54, respectively, which correspond to (21), (22). A slight enhancement is that midpoint shifts

$$y_r - m \quad \text{and} \quad s_* \text{base\_tm} + m$$

are introduced. Doing that the diagonal scaling by $S := \text{diag}(s)$ becomes more precise in view of fitting into $B = [-1, 1]^n$ compared to using $S' := \text{diag}(s + |m|)$. However, this is a minor technical issue which was skipped in (21), (22).
function [yl, yr] = precondition(a, b, base_tm, options, Q_CL)

PREC_QR = 1; % QR preconditioning
PREC_PE = 2; % parallelepiped preconditioning
PREC_CL = 3; % curvilinear preconditioning
PREC_ID = -1; % identity preconditioning

n = a(1).dim - 1;

if options.precondition == PREC_CL
    A = get_linear_terms(a);
    A = A(:, 1:n);
end

switch options.precondition
    case PREC_QR % QR preconditioning
        normsA = sqrt(sum(A.*A)); % sort columns of A by decreasing Euclidean norm
        [normsA, iA] = sort(normsA, 'descend'); % the column reordering is called As
        d = sign(diag(R)); % multiply columns of Q with +/-1
        d(d==0) = 1; % so that diag(R) >= 0
        Q = Q.*d';
    case PREC_PE % parallelepiped preconditioning
        if options.blunting
            Q = blunt(A, options); % perform blunting of A
        else
            cond_bound = 1e6; % heuristic upper bound for the condition number of A
            if cond(A) > cond_bound
                Q = blunt(A, options); % perform blunting of A
            else
                Q = A; % otherwise, take A as preconditioner
            end
        end
        cond_bound_max = 1e16; % heuristic upper bound for the condition number of Q; if it is exceeded, identity preconditioning is chosen for meaningful proceeding
        if cond(Q) > cond_bound_max
            Q = eye(n); % parameter Q_CL as preconditioner
        end
    case PREC_CL % curvilinear preconditioning
        Q = Q_CL;
    case PREC_ID % identity preconditioning
        Q = eye(n);
        Q_inv = Q;
end

if options.precondition == PREC_ID
    Q_inv = intval2iv(inv(intval(Q)));
end
[c, idx] = get_constant_term(a); % c is the constant term of a
a0 = subtract_constant_term(s, idx);
yr = (yr - m) / s;
yl = Qy(s.*base_tm + m) + c;

end % function precondition

Listing 1. Preconditioning of Taylor models.

3.1 Blunting for parallelepiped preconditioning

As stated before, Lohner [20], p. 51, paragraph 2, suggested a strategy of widening (blunting) the angles between the column vectors of an ill-conditioned matrix in order to reduce overestimation in the propagation of the global integration error. This idea was taken up and specified in [6, 7] where the name “blunting” was coined. Later on, Neher, Jackson, and Nedialkov [24] analyzed the effect of blunting in verified integration of ODEs in greater detail. In particular, they compared blunting and the QR method. According to their numerical experiments there is no clear winner within these two methods and the success of blunting may strongly depend on the chosen blunting factors. Here, we only state the implementation of blunting in INTLAB. For more background information on blunting we refer to the above references.
Let $A = QR$ be a QR decomposition of a matrix $A \in \mathbb{R}^{n \times n}$ with orthogonal matrix $Q$ and upper triangular $R$ such that all diagonal entries of $R$ are nonnegative, i.e., $R_{ii} \geq 0$. Furthermore, let $q \in \mathbb{R}^n_{>0}$ be a vector of real positive numbers, which are called blunting factors, and define $D = \text{diag}(q)$. Then the (unpermuted) $q$-blunting of $A$ is the matrix

$$\hat{A} := A + QD = Q(R + D) = Q(R + \text{diag}(q)).$$

(24)

Like for QR preconditioning, cf. Subsection 3.2, it is suggested in [6], p. 11, to sort the columns of $A$ by descending Euclidean length first. If $P$ is a corresponding permutation matrix, then a QR decomposition $AP = QR$ is taken and (24) transforms to the (permuted) $q$-blunting matrix of $A$:

$$\hat{A} := A + QDP^T = Q(R + D)P^T = Q(R + \text{diag}(q))P^T.$$  

(25)

Obviously, $\hat{A}$ is regular (cf. [6], Proposition 11) since $R_{ii} \geq 0$ and $q_i > 0$ imply

$$|\det(\hat{A})| = \prod_{i=1}^{n} (R_{ii} + q_i) > 0.$$

A crucial point is the choice of suitable blunting factors. There is not much said about this in the literature but heuristic attempts are proposed. For example, in [7], Section 4 “Linear Autonomous Examples”, constant blunting factors

$$q_1 = q_2 = \cdots = q_n := 10^{-3} \cdot \max_{1 \leq i \leq n} \|A_{[i,:]}\|_2$$

(26)

are suggested, where $\|A_{[i,:]}\|_2$ denotes the Euclidean norm of the $i$-th column vector of $A$. In [24], page 22 et seq.,

$$G_j = \varepsilon I_n, \quad \varepsilon > 0,$$

$$D_j = \text{diag}(\|R_j^*[1,:]\|_2^{-1}, \ldots, \|R_j^*[n,:]\|_2^{-1}) = \text{diag}(\|A_{[1,:]}\|_2^{-1}, \ldots, \|A_{[n,:]}\|_2^{-1}),$$

$$D = G_1 D_1^{-1} = \varepsilon \cdot \text{diag}(\|A_{[1,:]}\|_2, \ldots, \|A_{[n,:]}\|_2)$$

are chosen, i.e., the $i$-th blunting factor is the $\varepsilon$ multiple of the Euclidean norm of the $i$-th column of $A$. For the permuted version (25) this means

$$q_i := \varepsilon \|(AP)_{[i,:]}\|_2 = \varepsilon \|R_{[i,:]}\|_2.$$  

(27)

In [24] different choices of $\varepsilon$ were tested, and, for example, taking $\varepsilon := 10^{-3}$ in analogy to [7] was found to be not always appropriate. As noted before there is no advantageous general strategy known for choosing the blunting factors. In the former INTLAB version 11 we implemented (26) but in the current, new version 12 we switched to (27) where $\varepsilon$ can be chosen by the user via the option “blunting factor”. The default is $\varepsilon = 10^{-3}$.

The function blunt in Listing 2 has the matrix $A$ and user specified options as input parameters and returns the permuted $q$-blunting $B := \hat{A}$ of $A$ as defined by (25) and (27), provided that the condition number of $A$ exceeds some heuristically chosen bound. If this is not the case, i.e., if $A$ is quite well-conditioned, then $B := A$ is returned unchanged, see lines 2–7.

In lines 8, 9 the columns of $A$ are sorted by Euclidean length in descending order and are stored in the variable normsA. The corresponding permutation matrix $P$ for which the columns of $\text{as} := AP$ are in that order is implicitly given by the index set idx, see line 10. Then, lines 11–14 compute a nonverified QR decomposition of $\text{as} = QR$ such that $R$ has nonnegative entries so that $Q$ is the Gram-Schmidt orthonormalization of $\text{as}$. This part agrees with lines 13–19 of Listing 1.

---

1. This is blunting as stated in [6], Definition 10. A slightly different setting and notation is used in [24], page 22, Eqs. (14), (15), (16), (17). There, the matrices $TB_{j-1}$, $Q_j^*$, $R_j^*$, $D_j^*$, $V_j(D_jF_j)^{-1}$ correspond to $A$, $Q$, $R$, $D$, $A$, respectively, where $D_jF_j$ is a diagonal scaling matrix which is close to the identity matrix for small blunting factors $q$.

2. In [24] the initially arbitrary positive entries of the diagonal matrix $G_j$ are called blunting factors while in [6] these are the diagonal entries of $D = G_j D_j^*$. The difference is the scaling by the column norms of $A$ which are the diagonal entries of the diagonal matrix $D_j^{-1}$. 

---

This page contains text extracted from the document. The text is related to QR decomposition and blunting factors, discussing the process of blunting in matrices and how it is implemented in the INTLAB software. The text explains the mathematical formulation of the blunting process and provides a function in an algorithmic listing. The blunting factor $\varepsilon$ is discussed, along with its default value and the strategy for choosing it. The text also references other works for different settings and notations.
Line 16 sets the blunting factors $q$, see (27) and lines 17, 18 perform

$$B := [(AP + Q \text{ diag}(q))]^{P^{-1}} = A + Q \text{ diag}(q) \quad P^T = \hat{A}$$

which is (25). Finally we recall that blunting, like choosing any other preconditioner, does not require any verified computation.

```matlab
function B = blunt(A, options)
    % heuristic bound for the condition number
    bound = 1e2;
    % condition number of A
    condA = cond(A);
    if condA < bound
        % if A is well-conditioned, return B := A
        B = A;
    end
    % return
    normsA = sqrt(sum(A.*A));
    [normsA, ia] = sort(normsA, 'descend');
    % Euclidean column norms of A, % sort normsA in descending order
    As = A(:, ia);
    % As := A*P sorts columns of A.
    % As := Q*R is a nonverified QR decomposition
    [Q, R] = qr(As);
    d = sign(diag(R));
    % d = (sign(R_{11}), ..., sign(R_{nn}))
    % Selected d := 1 if R_{ii} := 0
    Q = Q.*d;
    % Q := Q*diag(d), R := diag(d)*R \Rightarrow R_{ii} >= 0
    e = options.blunting_factor;
    q = e*normsA;
    B = As + Q.*q;
    B(:, ia) = B;
end
```

Listing 2. Blunting of ill-conditioned matrices.

### 3.2 Permutated QR preconditioning

Consider $A := \begin{pmatrix} 1 & 7 \\ 2 & 10 \end{pmatrix}$ and $J := \begin{pmatrix} [-0.1,0.1] \\ [-0.1,0.1] \end{pmatrix}$. Then, an approximate QR decomposition of $A$ is given by

$$Q := \begin{pmatrix} -0.4472 & -0.8944 \\ -0.8944 & 0.4472 \end{pmatrix} \quad \text{and} \quad R := \begin{pmatrix} -2.2361 & -12.0748 \\ 0 & -1.7889 \end{pmatrix}.$$ 

The columns of $A$ have Euclidean lengths $\sqrt{1 + 2^2} = \sqrt{5} \approx 2.2361 = |R_{11}|$ and $\sqrt{7^2 + 10^2} = \sqrt{149} \approx 12.066$, respectively. Hence, the Euclidean norm of the second column is much larger than that of the first. Multiplying $A$ from the right with the permutation matrix $P := \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ gives the matrix

$$\hat{A} := AP = \begin{pmatrix} 7 & 10 \\ 1 & 2 \end{pmatrix}$$

the columns of which are sorted by Euclidean length in descending order. Then

$$\hat{Q} := \begin{pmatrix} -0.5735 & -0.8192 \\ -0.8192 & 0.5735 \end{pmatrix} \quad \text{and} \quad \hat{R} := \begin{pmatrix} -12.2066 & -2.2119 \\ 0 & 0.3277 \end{pmatrix}$$

form an approximate QR decomposition of $\hat{A}$. Taking $G := Q$ as preconditioner the evaluation of $(G^{-1}A)J$ in interval arithmetic gives

$$(G^{-1}A)J \approx RJ = \begin{pmatrix} -2.2361 & -12.0748 \\ 0 & -1.7889 \end{pmatrix} \begin{pmatrix} [-0.1,0.1] \\ [-0.1,0.1] \end{pmatrix} = \begin{pmatrix} [-1.4311,1.4311] \\ [-0.1789,0.1789] \end{pmatrix} := I.$$ 

On the other hand, taking $\hat{G} := \hat{Q}$, as preconditioner yields

$$(\hat{G}^{-1}A)J \approx \hat{R}P^TJ = \begin{pmatrix} -2.2119 & -12.2066 \\ 0.3277 & 0 \end{pmatrix} \begin{pmatrix} [-0.1,0.1] \\ [-0.1,0.1] \end{pmatrix} = \begin{pmatrix} [-1.4419,1.4419] \\ [-0.0328,0.0328] \end{pmatrix} := \hat{I}.$$ 

Thus, $\hat{I}$ encloses the exact result, the parallelepiped $(\hat{G}^{-1}A)J$, much tighter than $I$ encloses the parallelepiped $(G^{-1}A)J$. This is illustrated in Fig. 2. The left picture shows $I$ as red margined box.
enclosing the exact result, depicted as yellow parallelepiped, with much overestimation, whereas the right picture shows that \( \hat{I} \) is a tight enclosure.

This effect is clear since the QR decomposition is nothing but the Gram-Schmidt orthogonalization of the columns of \( A \) in their given order. Hence, starting the orthogonalization with the longest column \( A(:,1) \) of \( A \), transforms it to the first column of \( \hat{R} \) which is \( (\|A(:,1)\|_2,0)^T \). Thus, the longest edge of the parallelepiped spanned by the columns of \( A \) becomes an axis parallel edge of the parallelepiped spanned by the columns of \( \hat{R}\hat{P}^T \), see (28), so that enclosing the latter by an axis parallel interval box causes less overestimation. Finally, we remark that for higher dimensions \( n > 3 \) sorting the columns of \( A \) by Euclidean length in descending order is in general a good strategy for the same reasons. However, for special matrices \( A \) this may not be the optimal choice of a column reordering.

% permuted QR decomposition of A
\[ P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \]
\[ A_h = A \]
\[ \hat{Q}_h = \text{qr}(A_h) \]
\[ \hat{I}_h = (\hat{Q}_h^{-1} A) J \]
\[ \hat{K}_h = (\hat{Q}_h^{-1} A) \text{affari}(J) \]
figure(2), hold on
plotintval(Ih,'n')
plotaffari(Kh,'y')
axis([-2 2 -0.25 0.25])
title('permuted QR preconditioning', 'FontSize',14)

3.3 Curvilinear preconditioning

Curvilinear preconditioning was invented by Makino and Berz in [7], p. 111, Definition 8, as follows:\(^3\)

Let \( y^{(m)} = f(t, y, y', \ldots, y^{(m-1)}) \) be an \( m \)-th order ODE in \( n \) variables. Let \( y'_r(t) \) be a solution of the ODE and \( y'_r(t), \ldots, y^{(k)}_r(t) \) its first \( k \) time derivatives. Let \( e_1(t), \ldots, e_l(t) \) be \( l \) unit vectors not in the span of \( y'_r(t), \ldots, y^{(k)}_r(t) \) such that \( X = (y'_r(t), \ldots, y^{(k)}_r(t), e_1(t), \ldots, e_l(t)) \) have maximal rank.\(^5\) Then we call the Gram-Schmidt orthonormalization \( Q \) of \( X \) a curvilinear basis of depth \( k \), and we refer to its use for preconditioning as curvilinear preconditioning.

In contrast to parallelepiped and QR preconditioning, curvilinear preconditioning does not construct the preconditioner \( G \), see (21), (22), from the matrix \( A \) representing the linear part of the Taylor model

\[ ^3 \text{The notation is slightly adapted to ours.} \]
\[ ^4 \text{General background information on curvilinear coordinates can be found in [2, 3, 21] and will not be given here.} \]
\[ ^5 \text{In [8], slide 197, it is added that } e_1(t), \ldots, e_l(t) \text{ shall be sorted by distance from the span of } y'_r(t), \ldots, y^{(k)}_r(t). \]
\( \hat{a}(x) + I, \) see (20). Instead it uses the first \( k \) derivatives of a reference trajectory \( y_r(t) \) of the underlying ODE and extends them by some heuristically chosen unit vectors \( e_i(t) \) to a square matrix \( X \). Then, like for QR preconditioning, a nonverified QR factorization \( X = QR \) is computed such that the upper triangular matrix \( R \) has nonnegative diagonal entries and the nearly orthogonal factor \( Q \) is taken as the preconditioner \( Q \).

We want to address some difficulties that we encountered with definition (29) and that we could not fully resolve. Definition (29) is formulated for \( n \)-dimensional \( m \)-th order systems, so that the matrices \( X \) and \( Q \) in (29) are \( n \times n \) matrices. In [16–18], Hoefkens, Berz, and Makino used the antiderivative for Taylor models to solve \( m \)-th order ODEs more directly without transforming them by a standard order reduction to a first order system of dimension \( m \cdot n \). For example, see [18], Eq. (14)–(17), a second order IVP

\[
y'' = f(t, y, y'), \quad y(t_0) = y_0, \quad y'(t_0) = y'_0
\]

is rewritten as a first order IVP for the derivative \( \eta := y' \):

\[
\eta' = f(t, y_0 + \int_{t_0}^{t} \eta(\tau) \, d\tau, \eta) =: F(t, \eta), \quad \eta(t_0) = \eta_0 := y'_0.
\]

Since the antiderivative \( \int_{t_0}^{t} \eta(\tau) \, d\tau \) is directly available in Taylor model arithmetic, so is \( F(t, \eta) \). The \( n \)-dimensional first order system (31) may now be solved by using the preconditioning matrix \( Q \) and afterwards \( y(t) = y_0 + \int_{t_0}^{t} \eta(\tau) \, d\tau \) is obtained by one final application of the antiderivative to \( \eta \). However, the reference trajectory for \( \eta := y'_r \) and not \( y_r \) so that we would have expected to use a preconditioning matrix \( Q' \) derived from \( y'_r \) instead of \( y_r \). Moreover, the right-hand side \( F \) depends also on \( y_0 \) which is a time independent Taylor model in the Taylor model method, and we would have expected a preconditioning of \( y_0 \) also, in this case by \( Q \). To summarize, it is not completely clear to us, how definition (29) is intended to be implemented using only \( Q \).

It is also not clear to us if (29) fully complies with a standard order reduction to a first order system of dimension \( N := m \cdot n \). Nevertheless, we implemented such a first order version of curvilinear preconditioning in our solver verifyode knowing that for higher order ODEs \( (m > 1) \) this may deviate from curvilinear preconditioning as implemented in COSY and might be less effective.

The obvious attempt for a standard order reduction is to extend the \( n \)-dimensional reference curve \( y_r(t) \) to an \( N \)-dimensional reference curve \( \tilde{y}_r(t) \) of the corresponding first order system by setting

\[
\tilde{y}_r(t) := \left( y_r(t)^T, y'_r(t)^T, \ldots, y_{r(m-1)}'(t)^T \right)^T \in \mathbb{R}^N.
\]

Then, like in (29), an \( N \times N \)-matrix

\[
\tilde{X} := (\tilde{y}'_r(t), \ldots, \tilde{y}'_r^{(k)}(t), \tilde{e}_1(t), \ldots, \tilde{e}_L(t))
\]

is defined where \( \tilde{e}_i(t) \in \mathbb{R}^N, i = 1, \ldots, N-k \), are heuristically chosen unit vectors. In our implementation we use the standard unit vectors of \( \mathbb{R}^N \) and sort them by Euclidean distance from the span of \( \tilde{y}'_r(t), \ldots, \tilde{y}'_r^{(k)}(t) \) in descending order and take the first \( L := N-k \) of them as \( \tilde{e}_1(t), \ldots, \tilde{e}_L(t) \). It is not clear to us if this choice is always beneficial. However, different choices than that may lead to much worse remainder bounds as will be shown for the example in Section 4.4. Finally, a nonverified QR factorization \( \tilde{X} = \tilde{Q} \tilde{R} \) is performed and the almost orthogonal matrix \( G := \tilde{Q} \in \mathbb{R}^{N,N} \) is taken as preconditioner.

From now on we will work with such possibly order-reduced first order systems, i.e., \( m = 1 \) is assumed so that \( N = n, L = l = n-k, \tilde{y}_r = y_r, \tilde{X} = X \) and \( \tilde{e}_i(t) = e_i(t) \). In our implementation the depth \( k \) of the curvilinear coordinates can be prescribed via the option \texttt{CL_depth}, otherwise the default value \( k = \lceil n/2 \rceil \) is taken.

\footnote{The index “\( i \)” in \( y_r(t) \) stands for “reference” and not for “right” and should not be mixed up with right Taylor models.}

\footnote{If \( y_r(t) \) is a Frenet curve, i.e., \( y'_r(t), \ldots, y_{r(m-1)}'(t) \) are linearly independent for all \( t \), and if \( k = n-1 \), then the columns of \( Q \) simply build the Frenet frame of the curve at \( y_r(t) \) (in German “Frenet-n-Bein”) and for \( j \in \{1, \ldots, n-1\} \) the span of the first \( j \) columns of \( Q \) is the \( j \)-th osculating space (in German “\( j \)-ter Schmiegraum”) to the curve at that point.}
Another crucial point for curvilinear preconditioning is the choice of the reference trajectory $y_r(t)$ if it is not known a priori from theory. For the easy latter case we offer the option CL_reffun to pass a function handle of a user-implemented reference trajectory to verifyode. In the former case we proceed as follows. Let us denote the point initial value $c_0 := y_r(0)$ contained in the initial value set $C$ for the next integration step on $[t_{i+1}, t_{i+2}]$. Although the choice of the initial value $c_0$ for $y_r(t)$ is not explicitly stated in [7] (and, to our knowledge, not elsewhere in the literature), it is evident that $c_0$ should somehow be chosen close to the midpoint of $C$ in order to mimic a good average of the ODE flow. In that light $y_r(t)$ may also be called center trajectory. Recall from (15) that the initial value set $C$ is given by the Taylor model range

$$ C = \{(\hat{a}(x) + \hat{I}) \circ (b(x) + J)(B)\}. $$

Since $0 \in \mathbb{R}^n$ is the center of $B$ and also $0 \in J$ holds true almost always, it is reasonable to choose

$$ c_0 := \hat{a}(b(0)). \quad (32) $$

However, recall that (10) and (6) imply

$$ b(B) + J \subseteq B = [-1, 1]^n, $$

i.e., $B$ is an interval enclosure of the possibly nonconvex Taylor model range $b(B) + J$, and that the scaling matrix $S$ in (21) and (22) is chosen such that $B$ is a tight enclosure. Thus,

$$ C = \{(\hat{a}(x) + \hat{I}) \circ (b(x) + J)(B) \subseteq \hat{a}(B) + J, $$

wherefore it is also reasonable and computationally convenient to take

$$ c_0 := \hat{a}(0) = c, \quad \text{see (20) for the definition of } c, \quad (33) $$

as the initial value for $y_r(t)$ instead of $\hat{a}(b(0))$. We implemented and tested both, (32) and (33), and since our tests gave almost equal results, we decided for (33). The main task is to compute the derivatives $y_r^{(j)}(0)$, $j = 1, \ldots, k$, to construct the matrix

$$ X = (y_r'(0), \ldots, y_r^{(k)}(0), e_1(t), \ldots, e_l(t)). \quad (34) $$

For example, this can be done by automatic differentiation. In INTLAB, the Taylor toolbox or alternatively the awa toolbox can be used for this. However, we propose another, simpler way. Consider the function

$$ \hat{y}_r(t) := \hat{a}(0, t - t_i) \quad (35) $$

where $\hat{a}(x,t)$ is the polynomial part of the integrated left Taylor model on the time subinterval $[t_i, t_{i+1}]$, see (11). By (14), we have $\hat{a}(x) = \hat{a}(x, t_{i+1} - t_i)$, so that $\hat{y}_r(t_{i+1}) = \hat{a}(0) = c$, cf. (33). Therefore,

$$ \hat{y}_r^{(j)}(t_{i+1}) \approx y_r^{(j)}(0), \quad j = 1, \ldots, k, \quad (36) $$

are good approximates for the wanted derivatives. Obviously, this is advantageous because we already have $\hat{a}(x,t)$ at hand and building derivatives of the univariate polynomial $\hat{y}_r(t)$ is very easy. In our tests the approximates (36) were always very accurate. A relevant difference between this approach and using automatic differentiation, which we also implemented for comparison, was not recognizable.

Listing 3 shows our implementation of curvilinear preconditioning. The function curvilinear has the input arguments $y$, representing the integrated left Taylor model $\hat{a}(x, t) + \hat{I}$, see (11), (35), (36), $k$, fixing the depth of the curvilinear coordinates, and optionally reffun, which is a function handle of a user-implemented reference curve. The return parameter is the curvilinear preconditioner $G = Q$ as described above.

---

*For simplicity of notation we assume that the local reference trajectory $y_r(t)$ for the integration interval $[t_{i+1}, t_{i+2}]$ has the shifted domain $[0, t_{i+2} - t_{i+1}]$ so that simply $y_r(0)$ can be written instead of $y_r(t_i)$.**
The for-loop in lines 7–21 computes the derivatives $\hat{y}^{(j)}(t_{i+1})$, $j = 1, \ldots, k$, which are the first $k$ columns of the matrix $X$, see (34), (36). Alternatively, if the reference trajectory $y_r(t)$ is known a priori and is implemented by the user in `reffun`, then lines 23–29 compute $X$ by automatic differentiation of `reffun` using INTLAB’s Taylor objects.

If the depth $k$ is less then the dimension $n$, then lines 32–39 compute the standard unit vectors $e_1(t), \ldots, e_l(t)$ which are the trailing columns of the matrix $X$, see (34).

Lines 47–50 build the QR decomposition of $X$ such that the not explicitly adapted factor $R$ has nonnegative diagonal entries, and finally the corresponding approximately orthogonal factor $Q$ is returned as curvilinear preconditioner. Note again that all these computations are done in nonverified arithmetic.

```matlab
function Q = curvilinear(y,k,reffun)
    % # space-like variables x_1, ..., x_n
    y_ = y(1); % copy of the m-th component of y
    M = y_.monomial; % exponents of the polynomial part of y_
    c = y_.coefficient; % ... corresponding coefficients
    row = ~any(M(:,1:n),2); % index of x-independent exponents
    N = M(row,n+1); % ... corresponding time exponents
    c = c(row); % ... corresponding coefficients
    N = repmat(N,1,k); % N := [N,N,...,N] , k-fold repetition of N
    U = max(N-(0:k-1),0); % build derivates $\hat{p}^{(j)}(t)$ of order $j = 1, \ldots, k$
    U = cumprod(U,2); % by elementary calculus
    V = max(N-(1:k),0); %
    W = (c.*U).*(h.^(V)); % final evaluation at $t = t_{i+1}$
    X(m,:) = sum(W); %X(m,j) := $\hat{p}^{(j)}(t_{i+1})$
end
else % user-specified reference curve
    t_ = taylordin(t,k); % "taylor" object of Taylor order k for starting time t
    x = reffun(t_); % compute scaled derivatives of the reference trajectory
    x_t = struct(x); % convert x of type "taylor" to a simple structure
    x_t = x_t.t(2:k+1,:); % since x.t(1) is the function value (0-th derivative),
    X = x_t.*factorial(1:k); % compute the real, unscaled derivatives
end if k<n
I = eye(n); % nxn-identity matrix
Q = qr(X); % nxn-economy size QR decomposition of X
S = I-Q.*Q; % QQ' is the projection matrix to the subspace spanned
% by the columns of X, and S is the projection matrix to
% its orthogonal complement
normsS = sqrt(sum(S.*S)); % the Euclidean norms of the columns of S are
X = [X,I(:,1:is 1:n-k)]; % sorted in descending order and the first n-k
% columns of I according to this order are
% appended to X; these are the n-k standard unit
% vectors having the largest distance to the subspace
% spanned by the columns of X, among all possible
% choices of n-k standard basis vectors this specific
% choice aims to yield a reasonably well-conditioned
% extended matrix X
end
[Q,R] = qr(X); % QR decomposition of X
d = sign(diag(R)); % d := (sign(R_{11}),..., sign(R_{nn}))
d(d>0) = 1; %
Q = Q*d'; % the diagonal entries of R:=diag(d)*R are nonnegative
end % function curvilinear
```

Listing 3. Curvilinear preconditioning.
4. Examples
In this section we test our implementation of preconditioning with the following examples:

1. a quadratic model problem
2. the Higgins-Selkov oscillator
3. the Roessler attractor
4. an equation for a muon cooling ring
5. a Kepler problem for asteroid motion

The quadratic model problem is stated in [23]. The examples for the Roessler attractor and the muon cooling ring come from [7], Sections 3 and 7. The Kepler problem for asteroid motion is considered in [13, 16], and [19].

For each test case an executable MATLAB program is stated that produces our numerical results including all pictures. We believe that in this way everything becomes fully retraceable and reproducible.

All our computations were done with MATLAB 2018b under Linux openSUSE Leap 15.0 on a desktop PC with 15.6 GiB RAM and 8 x Intel Xeon E5-1620 0 @ 3.60 GHz CPU.

4.1 A quadratic model problem
In [23], Section 4.1, p. 243, the simple quadratic second order problem \( u'' = u^2 \) was considered. Rewritten as a two-dimensional first order system with \( y_1 := u \) and \( y_2 := u' \) this reads:

\[
\begin{align*}
y_1' &= y_2 \\
y_2' &= y_1^2
\end{align*}
\] (37)

Interval initial values are taken as

\[
y_0 \in (1 \pm 0.05, -1 \pm 0.05)
\] (38)

and integration is done for the time domain\(^9\)

\[
t \in [t_0, t_f] := [0, 6].
\] (39)

Listing 4 contains the implementation of (37) in a form that can be used by our solver `{verifyode}` as well as by MATLAB’s nonverified solvers like `{ode45}`.

```
function dy = quadratic_problem(t, y, i)
    if nargin == 2 || isempty(i)
        dy = [y(2); sqrt(y(1))];
    else
        switch i
            case 1
                dy = y(2);
            case 2
                dy = sqrt(y(1));
        end
    end
end  % function quadratic_problem
```

Listing 4. Quadratic model problem.

In [23], COSY was executed with Taylor model order 18 and QR preconditioning. In Listing 5 we use our solver `{verifyode}` with the same Taylor model order. Four different runs are executed with the naive Taylor model method, identity preconditioning, QR preconditioning, and curvilinear preconditioning, respectively. Parallelepiped preconditioning (with or without blunting) is not suited for this problem and breaks down prematurely at \( t \approx 3.13 \).

\(^9\)AWA [20] aborts integration at \( t = 3.75 \); that is also reported in [23].
The program computes verified inner and outer approximations at $t_f = 6$. To be precise, the verified inner and outer approximations are computed with respect to the enlarged initial value set $\hat{Y}_0 := [1; -1] + \text{infsup}(-5.5)/100$.

see line 1 of Listing 5. This is a tight floating-point enclosure of the true initial value set $Y_0 := [0.95, 1.05] \times [-1.05, -0.95]$ stated in (38). Then, clearly, a computed verified outer approximation at $t = t_f$ w.r.t. $\hat{Y}_0$ is also one w.r.t. $Y_0$. In contrast this not to be true for the inner approximation. However, we use the inner approximation only for judging the accuracy of the outer one. Computing a verified inner approximation with respect to $Y_0$ would have been possible also by using an inner floating-point approximation $\hat{Y}_0$ of $Y_0$, but this is not needed here for our purpose.

Listing 5 also generates a phase plane portrait of the ODE flow, see Fig. 3, which corresponds to and agrees with the right picture presented in [23], Fig. 6.1, p. 258.

```matlab
% create several option structures, each for a different run of verifyode
options = {verifyodeset( 'order' ,18, 'precondition ' ,0, 'h0' ,0.1, 'h_min ' ,0.01,...
            'loc_err_tol',1e-11,'sparsity_tol' ,1e-20);
            verifyodeset( 'order' ,18, 'precondition ' ,−1, 'h0' ,0.1, 'h_min ' ,0.01,...
            'loc_err_tol',1e-11,'sparsity_tol' ,1e-20);
            verifyodeset( 'order' ,18, 'precondition ' ,1 , 'h0' ,0.1, 'h_min ' ,0.01,...
            'loc_err_tol',1e-11,'sparsity_tol' ,1e-20);
            verifyodeset( 'order' ,18, 'precondition ' ,3 , 'h0' ,0.1, 'h_min ' ,0.01,...
            'loc_err_tol',1e-11,'sparsity_tol' ,1e-20)};

Y0 = [1;−1]+infsup(−5.5)/100; % interval initial values [0.95,1.05]×[−1.05,−0.95]
t0 = 0; tf = 6; % definition of integration interval [0,tf]

% create a phase portrait for identity precondition.
t = linspace(t0,tf,1e3); % fine time grid for plotting the corner trajectories
Y0 = [y0.inf y0.sup]; % Y0 contains the lower left and upper right corners of y0
Z = intval(zeros(2,2)); % initialize verified inner approximation

for i=1:2
    yf = intval(zeros(2,2)); % initialize result array for enclosures at tf
    for i=1:length(options)
        if options(i).precondition==−1 % create a phase portrait for identity precondition.
            t = linspace(t0,tf,16); % plot enclosures at all points in t
            figure(2), plot(mid(y(:,1)),mid(y(:,2),'-k')); % nonverified) plot of both
            figure(1), plot(mid(y(:,1)),mid(y(:,2),'-k')); % corner trajectories
        end
        % compute verified enclosure of the ODE flow, see Fig. 3, which
        % corresponds to and agrees with the right picture presented in [23], Fig. 6.1, p. 258.
    end
end

Listing 5. Preconditioning for the quadratic model problem.
Listing 5 produces the following output:

1. Naive Taylor model method
   Elapsed time is 13.289039 seconds.
   Number of integration steps: 288
   \[ t = 6 \]
   \[
   [y_1] = [-2.326517406532010e-001, \ 1.030206554551854e+000] \quad \text{d([y_1])} = 1.26e+00
   \]
   \[
   [y_2] = [ 3.497661822437370e-001, \ 1.122437540159332e+000] \quad \text{d([y_2])} = 7.73e-01
   \]

2. Identity preconditioning
   Elapsed time is 2.246480 seconds.
   Number of integration steps: 31
   \[ t = 6 \]
   \[
   [y_1] = [-2.326433087206398e-001, \ 1.030204256781837e+000] \quad \text{d([y_1])} = 1.26e+00
   \]
   \[
   [y_2] = [ 3.49777469826499e-001, \ 1.122431220603586e+000] \quad \text{d([y_2])} = 7.73e-01
   \]

3. QR preconditioning
   Elapsed time is 2.692056 seconds.
   Number of integration steps: 32
   \[ t = 6 \]
   \[
   [y_1] = [-2.32661307210758e-001, \ 1.030222247610039e+000] \quad \text{d([y_1])} = 1.26e+00
   \]
   \[
   [y_2] = [ 3.497621000683073e-001, \ 1.122446571618336e+000] \quad \text{d([y_2])} = 7.73e-01
   \]

4. Curvilinear preconditioning
   Elapsed time is 2.579662 seconds.
   Number of integration steps: 31
   \[ t = 6 \]
   \[
   [y_1] = [-2.326482841515020e-001, \ 1.030209231572328e+000] \quad \text{d([y_1])} = 1.26e+00
   \]
   \[
   [y_2] = [ 3.497728692424601e-001, \ 1.122435819424201e+000] \quad \text{d([y_2])} = 7.73e-01
   \]

Verified inner approximation
\[
\text{intval } z =
\begin{bmatrix}
-2.326328688858327e-001, & 1.030195971724399e+000 \\
3.497955234876077e-001, & 1.122415798738162e+000
\end{bmatrix}
\]
\[
\text{ans } =
\begin{bmatrix}
\text{logical} \\
1
\end{bmatrix}
\]

Relative width difference between inner and outer approximations:

\[
\begin{array}{ccc}
2.3324e-05 & 6.6112e-05 & \text{<=== naive method} \\
1.4828e-05 & 4.3325e-05 & \text{<=== identity preconditioning} \\
4.3325e-05 & 8.3082e-05 & \text{<=== QR preconditioning} \\
2.2707e-05 & 5.6231e-05 & \text{<=== curvilinear preconditioning}
\end{array}
\]

Fig. 3. Quadratic model problem: phase portrait for data obtained by identity preconditioning.
The validated enclosure intervals for the components \( y_1 \) and \( y_2 \) at the end of integration at \( t_f = 6 \) are denoted by \([y_1]\) and \([y_2]\), respectively, and their diameters \( d(y_1) \) and \( d(y_2) \) are printed behind. It turns out that all four methods yield enclosures of similar quality with diameters

\[
d(y_1) \approx 1.26 \quad \text{and} \quad d(y_2) \approx 0.77.
\] (40)

Even the integration with the naive method succeeds. However, it needs 288 integration steps and a runtime of 13 seconds. In contrast, the preconditioning methods take at most 32 integration steps and finish integration after less than 3 seconds. A closer look at the displayed intervals \([y_1]\) and \([y_2]\) yields that identity preconditioning gives the tightest enclosure, namely

\[
y(t_f = 6) \in ([-0.2326433087206398, 1.030204256781837], [0.349774698926499, 1.12243120603586]).
\]

This is followed by curvilinear preconditioning, the naive method, and QR preconditioning. Upon reversion, this means that the special advantages of QR, curvilinear, and parallelepiped precondition stated in Section 3 cannot be observed for this simple example. As stated above, parallelepiped precondition even has counterproductive effects leading to early breakdown. Thus, there seems to be no main rule for choosing the best type of preconditioning in advance.

Next, we want to rate the quality of the computed results. The diameters at end of integration, see (40), seem to be quite large compared to the diameter 0.1 of the initial values (38). However, this is not due to severe overestimation of the method but stems from diverging trajectories of the ODE flow. To prove that, the program computes in lines 34-47 a verified inner approximation \( z \) by executing \texttt{verifyode} another two times with point initial values

\[
y_0^{(1)} = (0.95, -1.05) \quad \text{and} \quad y_0^{(2)} = (1.05, -0.95).
\] (41)

These are the lower left and upper right corners of the initial value box \( y_0 \), see (38). The obtained inner approximation reads

\[
z = ([-0.232632868858327, 1.03019597124399], [0.3497955234876077, 1.122415798738162]).
\]

In line 49 it is checked that the inner approximation \( z \) is contained in all four outer ones. Finally, line 52 prints the relative deviation between the diameters of the inner and outer approximations. This deviation is, for all four methods, less than \( 10^{-4} \), i.e., less than 0.01%. This shows that the computed outer enclosures are tight, and this is also visualized in Fig. 3. For each time point

\[
t_k := 0.4k, \quad k = 0, \ldots, 15,
\]

from the integration domain \([t_0, t_f] = [0, 6]\), see (39), the image of the polynomial part of the corresponding Taylor model is printed in green. Neglecting remainders this represents the enclosure of the ODE flow at that time point computed by the Taylor model method. The blue solid line is the center trajectory, i.e., the solution for the point initial value \((1, -1)\) which is the midpoint of the initial value box \( y_0 \). The red dots are the values of the center trajectory at the time points \( t_k \). Finally, the black dashed lines are the corner trajectories having initial values (41). The plot shows that they almost exactly hit the endpoints of the elongated green shapes. We want to state clearly that the plots described so far use nonverified floating point arithmetic and shall only give a good impression of the ODE flow. Contrary to that, the black boxes in the right picture of Fig. 3 are overall verified enclosures of the ODE flow at the time points \( t_k \). They tightly enclose the green shapes, demonstrating once again the sharpness of the Taylor model results. These boxes also show the advantage of the Taylor model approach. It allows to enclose sets with curved boundaries (green shapes) without much overestimation.

\(^{10}\)For plotting these data the function \texttt{verifyode\_phase\_portrait} is used. It is not contained in INTLAB version 11, but shall be included in the next version 12.
4.2 The Higgins-Selkov oscillator

The Higgins-Selkov oscillator, see [11], p. 16, Example II.1,
\[ y_1' = v_0 - k_1 y_1 y_2^2 \]  
\[ y_2' = k_1 y_1 y_2^2 - k_2 y_2 \]  
with uncertain (interval) parameters
\[ v_0 = 1 \pm 2 \cdot 10^{-4} \]  
\[ k_1 = 1 \pm 2 \cdot 10^{-4} \]  
\[ k_2 = 1.00001 \pm 2 \cdot 10^{-4} \]  
and interval initial values
\[ y_0 \in (2 \pm 0.01, 1 \pm 0.01) \]  
shall be integrated on the time domain
\[ t \in [t_0, t_f] := [0, 10]. \]

Listing 6 contains the MATLAB/INTLAB implementation of the parametric ODE function (42), (43).

```matlab
function dy = higgen_selkov(t, y, i)
persistent v0 k1 k2
if isfloat(y) \% this branch is for MATLAB's nonverified ODE solvers like ode45
    v0 = 1;
    k1 = 1;
    k2 = 1.00001;
else \% this branch is executed by verifyode
    if isempty(v0) || isnumeric(v0)
        1 = infsup(-2,2)/1e4; \% parameter uncertainties of order +/- 2e-4
        v0 = 1+1i; \% verified enclosure of 1 +/- 2e-4
        k1 = 1+1i; \% ... same as before
        k2 = intval(’1.00001’) +1i; \% verified enclosure of 1.00001 +/- 2e-4
    end
end

c = k1.*y(1).*sqr(y(2));
if nargin==2 || isempty(i)
    dy = y;
    dy(1) = v0-c;
    dy(2) = c-k2.*y(2);
else
    switch i
        case 1
            dy = v0-c;
        case 2
            dy = c-k2.*y(2);
    end
end \% function higgen_selkov
```

Listing 6. Higgins-Selkov oscillator.

In line 2, the parameters \( v_0, k_1, \) and \( k_2, \) see (43), are defined as persistent so that they are initialized only once. This is done for performance reasons because interval initializations like in lines 9–12 are a bit expensive. In contrast, simple floating-point assignments like in lines 4-6, which are only implemented for the case that the function shall also be used for one of MATLAB’s nonverified ODE solvers like ode45, is very fast and variables need not to be declared as persistent. Another general performance issue, which shall clearly be addressed here, is the precomputation of common intermediate results. In this example the intermediate result \( c = k_1 y_1 y_2^2 \) is precomputed in line 15 and then used in lines 18, 19 for both equations (42): \( y_1' = v_0 - c \) and \( y_2' = c - k_2 y_2. \) The major disadvantage of Taylor models is their expensive arithmetic wherefore such simple code optimization should always be taken into account. This will pay off significantly, especially if Taylor models of high order are used.
In this example we will use Taylor models of order 6, minimum step size $h_{\text{min}} := 0.02$, and apply various kinds of preconditioning as well as the naive Taylor model method. The latter already breaks down at $t \approx 0.75$. Also identity preconditioning breaks down prematurely at $t \approx 4.5$. In contrast, QR, curvilinear, and parallelepiped preconditioning manage integration up to $t_f = 10$ without problems. Listing 7 calls verifyode five times with:

1. naive Taylor model method ($t_f := 0.75$)
2. identity preconditioning ($t_f := 4.5$)
3. QR preconditioning ($t_f := 10$)
4. curvilinear preconditioning with curvilinear depth $k := 2$, cf. (34), ($t_f := 10$)
5. parallelepiped preconditioning ($t_f := 10$)

After each run the validated enclosures for both components $y_1$ and $y_2$ at $t = t_f$ are computed and displayed along with their diameters. Also the total number of integration steps and the runtime are shown. Furthermore, four plots are produced. The first two, see Fig. 5, show the diameters of the enclosures for $y_1$ (left picture) and $y_2$ (right picture) for all five runs in logarithmic scale. The smaller the diameters the better suited is the method for this problem. The early breakdown of the naive method and identity preconditioning is evident by their exponential diameter growth. QR precondition and curvilinear preconditioning perform similarly, as expected, but give results that are worse by roughly one decimal order than those obtained by parallelepiped preconditioning. The latter read at $t_f := 0.75$:

- $y_1 \in [0.9237735798476144, 0.954628418449192]$, diameter = 0.0309
- $y_2 \in [0.8130172855842315, 0.8261504579731044]$, diameter = 0.0132

Since the initial values (44) have diameter 0.02 these enclosures seem reasonably tight.

The final two plots, see Fig. 6, are validated phase portraits depicting $y_1$ against $y_2$. We use the function plottaylormodel for that. The left picture shows the result for QR preconditioning and the right for parallelepiped preconditioning. The validated enclosures are plotted in black. Since small rectangles are used for that, the boundaries of these enclosures look spiky and not smooth.

The centered red trajectory is computed by MATLAB’s nonverified ODE solver ode45 executed with high accuracy where the midpoints of the intervals stated in (43) and (44) are taken. Obviously, parallelepiped preconditioning gives a much more precise phase portrait than QR preconditioning. Zooming in on the plot of the right picture of Fig. 6 at end of integration, which is done in Fig. 4, shows that the black enclosure pipe does not overlap.

---

11 In [11], p. 16, Example II.1, it is reported that a previous version of Flow* [10], which does not use symbolic remainder computations, breaks down at almost the same time $t \approx 4.44$.
12 MATLAB’s nonverified ODE solvers cannot deal with uncertainties in the input parameters.
13 Figure 2 in [11], p. 17, is of similar quality.
Preconditioning for the Higgins-Selkov oscillator example.

```matlab
% midpoint of initial values
y0_mid = [2;1];
% interval initial values ([1.99, 2.01], [0.99, 1.01])
t0 = 0; tf = 10; % definition of integration interval [t0, tf]

% execute verifyode with the naive Taylor model method
disp('1. Naive Taylor model method')
tf_NA = 0.75; % this is close to breakdown time
options = verifyodeset('order',6, 'precondition',0, 'blunting',0, 'h0',0.05,...
                    'h_min',0.02, 'loc_err_tol',1e-11, 'sparsity_tol',1e-16);
tic , [T_NA,Y_NA] = verifyode(@higgens_selkov, [t0, tf_NA], y0', options); toc
% display enclosure at tf
verifyode_disp(T_NA,Y_NA,[], y0,tf_NA,[],'longe'); % display enclosure at tf

% execute verifyode with identity preconditioning
disp('2. Identity preconditioning')
tf_ID = 4.5; % this is close to breakdown time
options = verifyodeset('order',6, 'precondition',-1, 'blunting',0, 'h0',0.05,...
                    'h_min',0.02, 'loc_err_tol',1e-11, 'sparsity_tol',1e-16);
tic , [T_ID,Y_ID,Yr_ID] = verifyode(@higgens_selkov, [t0, tf_ID], y0', options); toc
% display enclosure at tf
verifyode_disp(T_ID,Y_ID,Yr_ID,y0,tf_ID,[],'longe'); % display enclosure at tf

% execute verifyode with QR preconditioning
disp('3. QR preconditioning')
options = verifyodeset('order',6, 'precondition',2, 'blunting',0, 'h0',0.05,...
                    'h_min',0.02, 'loc_err_tol',1e-11, 'sparsity_tol',1e-16);
tic , [T_QR,Y_QR,Yr_QR] = verifyode(@higgens_selkov, [t0, tf_ID], y0', options); toc
% display enclosure at tf
verifyode_disp(T_QR,Y_QR,Yr_QR,y0,tf,[],'longe'); % display enclosure at tf

% execute verifyode with curvilinear preconditioning of depth k = 2
disp('4. Curvilinear preconditioning')
options = verifyodeset('order',6, 'precondition',3, 'CL_depth',2, 'blunting',0,...
                    'h0',0.05, 'h_min',0.02, 'loc_err_tol',1e-11, 'sparsity_tol',1e-16);
tic , [T_CL,Y_CL,Yr_CL] = verifyode(@higgens_selkov, [t0, tf_ID], y0', options); toc
% display enclosure at tf
verifyode_disp(T_CL,Y_CL,Yr_CL,y0,tf,[],'longe'); % display enclosure at tf

% execute verifyode with parallelepiped preconditioning
disp('5. Parallelepiped preconditioning')
options = verifyodeset('order',6, 'precondition',4, 'blunting',0, 'h0',0.05,...
                    'h_min',0.02, 'loc_err_tol',1e-11, 'sparsity_tol',1e-16);
tic , [T_PE,Y_PE,Yr_PE] = verifyode(@higgens_selkov, [t0, tf_ID], y0', options); toc
% display enclosure at tf
verifyode_disp(T_PE,Y_PE,Yr_PE,y0,tf,[],'longe'); % display enclosure at tf

% plot the diameters of the enclosures for y1 and y2 at the time grid points
for i=1 : 2
    figure(i)
xlabel('Time t', 'FontSize',14), ylabel('Diameters', 'FontSize',14)
semilogx(tNA, dNA(:,i), t_ID, d_ID(:,i), t_QR, d_QR(:,i), t_CL, d_CL(:,i),... 
t_PE, d_PE(:,i), 'LineWidth',2); 
legend('naive method', 'identity preconditioning', 'QR preconditioning', ...
       'curvilinear preconditioning', 'parallelepiped preconditioning')
end

% phase portrait of y1 against y2
figure(3), hold on, xlabel('y1', 'FontSize',14), ylabel('y2', 'FontSize',14)
n = length(y0); parts = 4; color = 'b'; fillcolor = 1; mode = 1;
plottaylormodel(tie(Y_QR,Yr_QR),[]), n+1,parts,color,fillcolor,mode); % QR cond.
options = odeset('RelTol',1e-14, 'AbsTol',[1e-14 1e-14]) ; % high accuracy for ode45
[t_ode45, y_ode45] = ode45(@higgens_selkov, [t0, tf], y0_mid, options); % call ode45
plot(y_ode45(:,1), y_ode45(:,2), 'r', 'LineWidth',1) % plot ode45 result in red
plottaylormodel(tie(Y_PE,Yr_PE),[]), n+1,parts,color,fillcolor,mode); % PE cond.
plot(y_ode45(:,1), y_ode45(:,2), 'r', 'LineWidth',1)
```

Listing 7. Preconditioning for the Higgins-Selkov oscillator example.
Listing 7 produces the following output and pictures:

1. Naive Taylor model method
   Elapsed time is 1.459524 seconds.
   Number of integration steps: 36
   \[ t = 7.500000000000000e-01 \]
   \[ y_1 = [5.450417696806213e-001, 6.848163629086781e+000] \]
   \[ d(y_1) = 1.40e-01 \]
   \[ y_2 = [1.920259817349872e+000, 2.071651047361289e+000] \]
   \[ d(y_2) = 1.52e-001 \]

2. Identity preconditioning
   Elapsed time is 12.392202 seconds.
   Number of integration steps: 224
   \[ t = 4.500000000000000e+00 \]
   \[ y_1 = [-6.537190577239006e-002, 2.51301533194301e+000] \]
   \[ d(y_1) = 2.58e+00 \]
   \[ y_2 = [-5.080968818093688e-001, 2.101587485805946e+000] \]
   \[ d(y_2) = 2.61e-01 \]

3. QR preconditioning
   Elapsed time is 29.818754 seconds.
   Number of integration steps: 499
   \[ t = 10 \]
   \[ y_1 = [8.715611024859512e-001, 1.011310727881227e+000] \]
   \[ d(y_1) = 1.40e-01 \]
   \[ y_2 = [7.836093026061106e-001, 8.533672120501039e-001] \]
   \[ d(y_2) = 6.98e-02 \]

4. Curvilinear preconditioning
   Elapsed time is 29.790794 seconds.
   Number of integration steps: 499
   \[ t = 10 \]
   \[ y_1 = [8.527767447551086e+000, 1.02989178522627e+000] \]
   \[ d(y_1) = 1.78e-01 \]
   \[ y_2 = [7.698836042278396e+000, 8.67128577108564e+000] \]
   \[ d(y_2) = 9.73e-02 \]

5. Parallelepiped preconditioning
   Elapsed time is 29.093021 seconds.
   Number of integration steps: 499
   \[ t = 10 \]
   \[ y_1 = [9.237735798476144e-001, 9.54628418449192e-001] \]
   \[ d(y_1) = 3.09e-02 \]
   \[ y_2 = [8.13017285582315e-001, 8.261504579731044e-001] \]
   \[ d(y_2) = 1.32e-02 \]

Fig. 5. Higgins-Selkov oscillator: diameters of \( y_1 \) (left) and \( y_2 \) (right).

Fig. 6. Higgins-Selkov oscillator: phase portraits, QR precond. (left), parallelepiped precond. (right).
4.3 The Roessler attractor

The Roessler equation

\[
\begin{align*}
    y'_1 &= -(y_2 + y_3) \\
    y'_2 &= y_1 + ay_2 \\
    y'_3 &= b + y_3(y_1 - c)
\end{align*}
\]  

shows chaotic behavior for the classical values \( a := 0.2, b := 0.2, \) and \( c := 5.7. \) In [7], Section 3, interval initial value conditions

\[ y(0) \in (0, -8.38095, 0.0295902) + [-r, r]^3, \quad r \in \{0.1, 0.2\} \]  

with two distinct choices of the radius \( r \) and the integration period

\[ [t_0, t_f] = [0, 6] \]  

were considered. An implementation of the Roessler equation (45) is given in Listing 8.

---

Listing 8. Roessler equation.

Listing 9 investigates the IVP (45), (46) for the large radius \( r := 0.2. \) The following six different settings of preconditioning are compared:

1. identity preconditioning
2. QR preconditioning
3. curvilinear preconditioning with default depth \( k = \lceil n/2 \rceil = 2 \)
4. parallelepiped preconditioning without blunting
5. parallelepiped preconditioning with default blunting factor \( \varepsilon = 0.001, \) see (27)
6. parallelepiped preconditioning with blunting factor \( \varepsilon = 0.1 \)

Taylor models of order 10 are taken and, like in [7], p. 118, a minimum step size \( h_{\text{min}} := 10^{-4} \) is fixed. The time point at which integration undergoes this step size is considered as breakdown time of the
Table I. Breakdown times of preconditioning methods.

| nr. | preconditioning method | breakdown time |
|-----|------------------------|---------------|
| 1.  | ID                     | 13.28         |
| 2.  | QR                     | 13.5          |
| 3.  | CL                     | 8.18          |
| 4.  | PE                     | 3.88          |
| 5.  | PEBL, $\varepsilon = 0.001$ | 7.06         |
| 6.  | PEBL, $\varepsilon = 0.1$ | 13.47        |

respective preconditioning method. These breakdown times have been precomputed approximately and are listed in Table I.

The winner is QR preconditioning followed by blunted parallelepiped preconditioning with blunting factor $\varepsilon = 0.1$ and identity preconditioning. Less effective are curvilinear preconditioning and blunted parallelepiped preconditioning with default blunting factor $\varepsilon = 0.001$. The early breakdown of parallelepiped precondition without blunting at $t \approx 3.88$ shows that this method is not well-suited for this IVP. All other methods easily manage to integrate up to $t_f = 6$.

By lines 2 and 3 of Listing 9 the breakdown times of Table I are stored in the array $T_f$. Then, the for-loop in lines 21–26 performs the verified integration up to $t_f = 6$ for all preconditioning methods except for parallelepiped preconditioning without blunting which cannot integrate that far. For example, QR preconditioning computes for the first component $y_1$ the verified enclosure

$$[0.3669054373395957, 1.544967237863772]$$

with diameter $1.1781 \approx 1.8$.

In lines 27–43 verified inner approximations are computed on the basis of the trajectories originating at the eight corner points of the initial value cube stated in (46) for $r = 0.2$. To be precise, we actually take the eight corner points of a verified enclosure $y_0$ of the exact initial value cube calculated in line 1. Thus, we compute verified inner approximations for the IVP having the enlarged initial value set $y_0$. In lines 44–46 the inner approximation at $t_f = 6$ is displayed along with its diameters. For example, the inner approximation for the first component $y_1$ reads

$$[0.3669404495193128, 1.536690846478785]$$

with diameter $1.1698$.

Afterwards, in lines 47, 48 the relative difference between the diameters of inner and outer approximations are computed and displayed. These data confirm the ranking mentioned above. For example, for QR preconditioning the diameters of inner and outer approximations for the first component $y_1$ deviate by approximately $1 - 1.1698/1.1781 \approx 0.7\%$ (best result) while blunted parallelepiped preconditioning with default blunting factor $\varepsilon = 0.001$ gives a deviation of $6.8\%$ (worst result).

Line 49 checks whether the inner approximation at $t_f = 6$ is contained in all outer ones. Of course, this must always hold true.

Next, the for-loop in lines 50–53 computes verified enclosures for all six considered preconditioning methods until their respective breakdown times. The evolution of the diameters of these enclosures and the diameters of the inner approximation are plotted by the for-loop in lines 55–65 for each of the three components $y_1, y_2,$ and $y_3$. This creates the first upper three pictures of Fig. 7. The pictures confirm once more that QR preconditioning (green line) gives best results (tightest enclosures), which are close to the inner approximation (black line). This is followed by blunted parallelepiped preconditioning with blunting factor $\varepsilon = 0.1$ (blue solid line) and identity preconditioning (red line).

Finally, lines 66–72 plot phase portraits of $y_i(t)$ against $y_j(t)$ for $t \in [0, 6], (i, j) \in \{(1, 2), (1, 3), (2, 3)\}$, see the last three pictures of Fig. 7. The phase portraits are analogues of the right picture of Fig. 3. The green dotted areas represent the projections of the respective Taylor model images to the $y_i, y_j$-plane and the black bordered rectangles are verified enclosures of their full ranges. The last picture, for which $i = 2$ and $j = 3$, corresponds to the second picture of Fig. 4 on page 117 of [7].
Preconditioning for the Roessler attractor example.

Listing 9.
Listing 9 produces the following screen output:

1. Identity preconditioning
   Elapsed time is 65.331210 seconds.
   Number of integration steps: 355
   \[ t = 6 \]
   \[
   \begin{align*}
   [y_1] &= [3.66553791511334e-001, 1.545017295654223e+000] \\
   d([y_1]) &= 1.18e+00 \\
   [y_2] &= [-9.480786849468717e+000, -7.579551527843899e+000] \\
   d([y_2]) &= 1.91e+00 \\
   [y_3] &= [3.067301477202024e-002, 3.728775759819790e-002] \\
   d([y_3]) &= 6.62e-03
   \end{align*}
   \]
   [81x752]

2. QR preconditioning
   Elapsed time is 90.611447 seconds.
   Number of integration steps: 355
   \[ t = 6 \]
   \[
   \begin{align*}
   [y_1] &= [3.669054373395957e-001, 1.544967237863772e+000] \\
   d([y_1]) &= 1.18e+00 \\
   [y_2] &= [-9.480727694632522e+000, -7.579610682665610e+000] \\
   d([y_2]) &= 1.91e+00 \\
   [y_3] &= [3.067321771945429e-002, 3.7287546992883e-002] \\
   d([y_3]) &= 6.62e-03
   \end{align*}
   \]
   [81x752]

3. Curvilinear preconditioning
   Elapsed time is 116.902652 seconds.
   Number of integration steps: 447
   \[ t = 6 \]
   \[
   \begin{align*}
   [y_1] &= [3.659190264147444e-001, 1.545953650673068e+000] \\
   d([y_1]) &= 1.19e+00 \\
   [y_2] &= [-9.481284011032413e+000, -7.5798659725804e+000] \\
   d([y_2]) &= 1.91e+00 \\
   [y_3] &= [3.065278640510429e-002, 3.730798659725804e-002] \\
   d([y_3]) &= 6.66e-03
   \end{align*}
   \]
   [81x752]

4. Blunted parallelepiped preconditioning with default blunting factor 0.001
   Elapsed time is 92.978000 seconds.
   Number of integration steps: 336
   \[ t = 6 \]
   \[
   \begin{align*}
   [y_1] &= [3.284410811137879e-001, 1.583431450968446e+000] \\
   d([y_1]) &= 1.26e+00 \\
   [y_2] &= [-9.581981163160708e+000, -7.478357241610144e+000] \\
   d([y_2]) &= 2.11e+00 \\
   [y_3] &= [3.041930623004050e-002, 3.754218050120008e-002] \\
   d([y_3]) &= 7.13e-03
   \end{align*}
   \]
   [81x752]

5. Blunted parallelepiped preconditioning with blunting factor 0.1
   Elapsed time is 86.396443 seconds.
   Number of integration steps: 321
   \[ t = 6 \]
   \[
   \begin{align*}
   [y_1] &= [3.668483645672234e-001, 1.545024310572516e+000] \\
   d([y_1]) &= 1.18e+00 \\
   [y_2] &= [-9.480778720867648e+000, -7.579550674465019e+000] \\
   d([y_2]) &= 1.91e+00 \\
   [y_3] &= [3.067282135509453e-002, 3.728795101718164e-002] \\
   d([y_3]) &= 6.62e-03
   \end{align*}
   \]
   [81x752]

Verified inner approximation at \( t = 6 \):
\[
\begin{align*}
[y_1] &= [3.669404495193128e-001, 1.536690846478785e+000] \\
[y_2] &= [-9.4803160663195e+000, -7.579645192062283e+000] \\
[y_3] &= [3.067364683001686e-002, 3.728730762815603e-002]
\end{align*}
\]

\begin{equation}
\text{diameters} = \begin{bmatrix}
1.1698e+00 \\
1.9011e+00 \\
6.6139e-03
\end{bmatrix}
\end{equation}

Relative width difference between inner and outer approximations:
\[
\begin{align*}
7.1396e-03 & \quad 9.8543e-05 & 1.3607e-04 & \quad \text{identity preconditioning} \\
7.0552e-03 & \quad 3.6318e-05 & 7.4707e-05 & \quad \text{QR preconditioning} \\
8.7152e-03 & \quad 6.2121e-04 & 6.2143e-03 & \quad \text{curvilinear preconditioning} \\
6.7921e-02 & \quad 9.6299e-02 & 7.4707e-02 & \quad \text{blunted parallelepiped prec., } \varepsilon = 0.001 \\
7.1514e-03 & \quad 9.9444e-05 & 1.9454e-04 & \quad \text{blunted parallelepiped prec., } \varepsilon = 0.1
\end{align*}
\]

\begin{equation}
\text{ans} = \begin{bmatrix}
\text{logical} \\
1
\end{bmatrix}
\end{equation}

Without further ado we also state the relative width differences between inner and outer approximations at \( t = 12 \) for QR preconditioning (tightest enclosure), blunted parallelepiped preconditioning with blunting factor \( \varepsilon = 0.1 \), and identity preconditioning (widest enclosure).

Relative width difference between inner and outer approximations at \( t = 12 \):
\[
\begin{align*}
5.4710e-01 & \quad 4.0518e-01 & 6.5271e-01 & \quad \text{identity preconditioning} \\
1.2010e+01 & \quad 7.4549e-02 & 2.1366e-01 & \quad \text{QR preconditioning} \\
2.5396e-01 & \quad 2.7682e-01 & 4.7999e-01 & \quad \text{blunted parallelepiped prec., } \varepsilon = 0.1
\end{align*}
\]
Listing 9 plots the following pictures:

Fig. 7. Roessler attractor: diameter evolution up to breakdown times and phase portraits up to $t_f = 6$. 
4.4 A muon cooling ring

In [7], Section 7, the following example for a muon cooling ring was given:

\[
\begin{align*}
y_1' &= y_3 \\
y_2' &= y_4 \\
y_3' &= y_4 - \frac{\alpha}{y_3 + y_4} \cdot y_3 + \frac{\alpha}{y_3 + y_4} \cdot y_2 \\
y_4' &= y_3 - \frac{\alpha}{y_3 + y_4} \cdot y_4 - \frac{\alpha}{y_3 + y_4} \cdot y_1
\end{align*}
\]  

(48)

Interval initial values with three distinct radii \( r \) are considered

\[
y(0) \in (0, 1, 1, 0) + [-r, r]^4 =: Y_0, \quad r \in \{10^{-2}, 10^{-4}, 10^{-6}\},
\]

(49)

and it is reported that the large radius \( r = 10^{-2} \) represents practical needs.\(^{14}\) The integration period

\[
[t_0, t_f] = [0, 20\pi]
\]

corresponds to 10 revolutions. The factor \( \alpha \) in (48) is a so-called cooling factor which takes values in \([0, 1]\), where \( \alpha = 1 \) means fastest cooling. The value of \( \alpha \) for which the computations were done in [7] is not explicitly stated, also the used Taylor model order is not specified. In [14], Section 4.2, the same example appears with fixed radius \( r := 10^{-4} \) and \( \alpha := 0.1 \). In [15], Section 4, a slightly different, enhanced equation for the muon cooling ring is considered where also \( \alpha := 0.1 \) is taken. Thus we will also choose \( \alpha := 0.1 \) for our computations. The implementation of the ODE function (48) is stated in Listing 10.

```matlab
function dy = muon_cooling(t, y, i)
persistent alpha
if isfloat(y) % this branch is for MATLAB's nonverified ODE solvers like ode45
    alpha = 0.1;
else % this branch is executed by verifyode
    if isempty(alpha) || isnumeric(alpha)
        alpha = intval('0.1'); % verified enclosure of 0.1
    end
end
if nargin == 2 || isempty(i) || i > 3
    z1 = alpha*(sqrt(y(3))+sqrt(y(4)))^0.5;
    z2 = alpha*(sqrt(y(1))+sqrt(y(2)))^0.5;
end
if nargin == 2 || isempty(i)
    dy = y;
dy(1) = y(3);
dy(2) = y(4);
dy(3) = y(4)-z1*y(3)+z2*y(2);
dy(4) = -y(3)-z1*y(4)-z2*y(1);
else
    switch i
        case 1
            dy = y(3);
        case 2
            dy = y(4);
        case 3
            dy = y(4)-z1*y(3)+z2*y(2);
        case 4
            dy = -y(3)-z1*y(4)-z2*y(1);
    end
end
end
```

Listing 10. Muon cooling ring.

Listing 11 takes \( r := 10^{-2} \) and executes `verifyode` three times, with QR, curvilinear, and parallelepiped preconditioning. The computations are done with Taylor model order 10. For curvilinear

\(^{14}\)AWA [20] succeeds for \( r = 10^{-4} \) but not for \( r = 10^{-2} \).
preconditioning the default depth \( k := \lceil n/2 \rceil = 2 \) is used, cf. (34), where \( n := 4 \) is the dimension of the ODE (48). A reference curve \( y_r(t) \) describing a simple circular motion is known a priori:

\[
y_r(t) := (\cos(t - \pi/2), -\sin(t - \pi/2), -\cos(t - \pi/2) - \sin(t - \pi/2))
\]

It solves (48) for the initial value \( y_r(0) = (0, 1, 1, 0) \) which is the midpoint of \( Y_0 \), see (49).

```matlab
% check whether all ode45 results are contained in the verifyode enclosures at t = tf
a = A(:, :); y0 = a.*y0.inf+(1-a).y0.sup; % build a convex combination of the corners of y0
for j = 1:size(A,2)
a = A(:, j); y0 = a.*y0.inf+(1-a).y0.sup; % build a convex combination of the corners of y0
y0_midi = [0;1;1;0]; % midpoint of initial value interval vector
y0 = y0_midi + infsup(-1,1)/100; % verified enclosure of y0_midi + [-0.01 0.01]
t0 = 0; tf = 20*pi; % tf := 20*pi is rounded to a floating-point number
% execute verifyode with Q R preconditioning
disp('1. Q R preconditioning')
options = verifyodeset('order',10,'precondition',1,'blunting',0,'h0',0.1,'hmin',0.001,'loc_err_tol',1e-11,'sparsity_tol',1e-16);
tic
[TQR,YQR,YrQR] = verifyode(@muon_cl,Y,y0,t0,tf,options);
time_QR = toc
% execute verifyode with curvilinear preconditioning
disp('2. Curvilinear preconditioning (default depth of curvilinear coordinates)')
reffun = @(t)[cos(t*pi/2);-sin(t*pi/2);-sin(t*pi/2);-cos(t*pi/2)]; % reference curve
options = verifyodeset('order',10,'precondition',3,'CL_reffun',reffun,'h0',0.1,'hmin',0.001,'loc_err_tol',1e-11,'sparsity_tol',1e-16);
tic
[TCYL,YCL,YrCL] = verifyode(@muon_cl,Y,y0,t0,tf,options);
time_CL = toc
% execute verifyode with parallelepiped preconditioning
disp('3. Parallelepiped preconditioning')
options = verifyodeset('order',10,'precondition',2,'blunting',0,'h0',0.1,'hmin',0.001,'loc_err_tol',1e-11,'sparsity_tol',1e-16);
tic
[TPE,YPE,YrPE] = verifyode(@muon_cl,Y,y0,t0,tf,options);
time_PE = toc
% for comparison execute MATLAB's nonverified ODE solver ode45 with high accuracy
options_ode45 = odeset('RelTol',1e-14,'AbsTol',[1e-14 1e-14 1e-14 1e-14 1e-14 1e-14]);
y_ode45end = infsup(y0,options_ode45(end,:)); % build convex hull of results
disp('Enclosure of several ode45 runs for distinct initial values in y0:')
format shorte
disp(y_ode45end)
```

Listing 11. Preconditioning for the muon cooling ring example.

For comparison, code lines 33–41 use MATLAB’s nonverified ODE solver ode45 to compute results for 100 random initial values \( y_0 \in Y_0 \), cf. (49), and also for all corner points of \( Y_0 \). The convex hull
of all ode45 results is stored in the variable \texttt{y ode45 end} which may serve as a nonverified inner approximation of the true result. For calculating a validated inner approximation it would have been possible to use \texttt{verifyode} instead of \texttt{ode45} with point initial values. However, since \texttt{ode45} is much faster than \texttt{verifyode} and gives sharp results for this IVP, we content with a nonverified check. Line 46 stores the diameters of \texttt{y ode45 end} in the variable \texttt{diameters ode45} which is used in line 48 to compute the relative deviation in width between the nonverified inner approximation computed by \texttt{ode45} and all verified outer approximations computed by \texttt{verifyode}. Finally, it is checked in lines 51-53 whether the inner approximation is contained in the outer ones. In our tests this was always the case, but we note clearly again that this enclosure property is just a nonverified, reasonable check because \texttt{ode45} - although executed with high accuracy - may calculate wrong results that are not contained in the validated outer enclosures.

Using the function \texttt{tie}, lines 55–57 concatenate the integrated left Taylor models \texttt{Y(:,1)} for the first component \(y_1\) and the right Taylor models \texttt{Yr_}, cf. (8), and store the results in \texttt{y_}, for all three preconditioning methods QR, CL, PE, respectively.\(^\text{15}\) Then, the diameters \(d_\) of the remainder intervals \(z\) of \(y_\) are computed. Lines 58-61 plot these remainder diameters in logarithmic scale. Executing Listing 11 produces the following result:

1. QR preconditioning
runtime\_QR = 9.0551e+03
Number of integration steps: 355
\[
t = 6.2838530719587e+01
\]
\[
[y_1] = [-1.187780927677712e-002, 1.181262497117186e-002] \quad d([y_1]) = 2.37e-02
\]
\[
[y_2] = [9.89318978084536e-001, 1.010001962412228e+000] \quad d([y_2]) = 2.02e-02
\]
\[
[y_3] = [9.89766010374851e-001, 1.010000896415407e+000] \quad d([y_3]) = 2.03e-02
\]
\[
[y_4] = [-1.187895107677784e-002, 1.196180630108117e+000] \quad d([y_4]) = 2.39e-02
\]

2. Curvilinear preconditioning (with default depth of curvilinear coordinates)
runtime\_CL = 7.0576e+03
Number of integration steps: 315
\[
t = 6.2838530719587e+01
\]
\[
[y_1] = [-1.187776523811264e-002, 1.181258093273356e-002] \quad d([y_1]) = 2.37e-02
\]
\[
[y_2] = [9.89318802885699e-001, 1.010001959183887e+000] \quad d([y_2]) = 2.02e-02
\]
\[
[y_3] = [9.89766374723428e-001, 1.010000892771928e+000] \quad d([y_3]) = 2.03e-02
\]
\[
[y_4] = [-1.18789476538215e-002, 1.196179999999924e+000] \quad d([y_4]) = 2.39e-02
\]

3. Parallelepiped preconditioning
runtime\_PE = 7.1057e+03
Number of integration steps: 297
\[
t = 6.2838530719587e+01
\]
\[
[y_1] = [-1.1877789473706e-002, 1.181257358833832e-002] \quad d([y_1]) = 2.37e-02
\]
\[
[y_2] = [9.8931887882602e-001, 1.010001958406558e+000] \quad d([y_2]) = 2.02e-02
\]
\[
[y_3] = [9.8976645144167e-001, 1.010000892004769e+000] \quad d([y_3]) = 2.03e-02
\]
\[
[y_4] = [-1.18789735935885e-002, 1.19617525840233e+000] \quad d([y_4]) = 2.39e-02
\]

Enclosure of several ode45 runs for distinct initial values in \(y_0\):
\[
[ -1.187775701689736e-002, 1.181257271166257e+000]
[ 9.89318835292090e-001, 1.00999999999965e+000]
[ 9.89776648530768e-001, 1.00999999999999e+000]
[ -1.18189068797946e-002, 1.196178178307387e-002]
\]
\[
diameters ode45 =
\]
\[
2.3691e-02
2.0169e-02
2.0224e-02
2.3843e-02
\]

Relative width difference between inner and outer approximations:
\[
\begin{align*}
4.4119e-06 & < 4.5266e-04 & 2.8575e-04 & \ll \text{QR preconditioning} \\
6.9406e-07 & < 4.4616e-04 & 2.8187e-04 & \ll \text{curvilinear preconditioning} \\
7.4082e-08 & < 4.4091e-04 & 2.8214e-04 & \ll \text{parallelepiped preconditioning}
\end{align*}
\]
\[
\text{ans} =
\]
\[
\begin{array}{ll}
\text{logical} & 1
\end{array}
\]

\(^{15}\)Concatenating all computed left and right Taylor models in hindsight, is time consuming and not a common task in practice.
For example, QR preconditioning needs 355 integration steps and a runtime of 9055 seconds, that is 2.5 hours. At end of integration, at \( t = t_f \), the validated enclosure

\[ y_1(t_f = 20\pi) \in [-0.01187780927677712, 0.01181262497117186] \]

for the first component \( y_1 \) is obtained which has diameter \( 2.37 \cdot 10^{-2} \). Curvilinear preconditioning needs only 315 integration steps, has a runtime of 7058 seconds, that is 1.96 hours, and the enclosures are a bit tighter than those of QR preconditioning. Finally, parallelepiped preconditioning needs only 297 integration steps, lasts 1.99 hours, and gives slightly tighter enclosures than curvilinear preconditioning. Thus, it produces the best results.\(^{16}\)

The nonverified inner approximation computed by \texttt{ode45} has diameters that are less than one per mille smaller than those of the verified outer enclosures computed by \texttt{verifyode}. This suggests reasonable tightness of the latter. Finally, we mention that simple identity preconditioning cannot cope with the quite long integration period. Around \( t = 23 \) step sizes are already small of order \( 10^{-3} \) and polynomials became more and more dense so that the computation factually stops.

The left picture in Fig. 8 corresponds to the dotted line in Fig. 14 of [15] (which has the legend entry \( DX = 1e-2 \)). This dotted line in Fig. 14 of [15] computed by COSY with curvilinear preconditioning gives at end of integration at \( t_f = 20\pi \) a remainder bound of order \( 10^{-9} \) while our result for curvilinear preconditioning yields a remainder bound of order \( 10^{-8} \) which is therefore by roughly one order worse than the COSY result. Nevertheless, our result obtained with parallelepiped preconditioning also achieves a remainder bound of order \( 10^{-9} \). Finally, QR preconditioning ends up with the largest remainder bounds of order \( 10^{-7} \). However, recall that the diameter of the enclosure of \( y_1 \) at \( t = t_f = 20\pi \) is around \( 2.37 \cdot 10^{-2} \) and, according to the nonverified inner approximation, this ought to be sharp. Thus, the remainder bounds arising in all three methods of preconditioning are negligible.

At this point we want to come back to our implementation of curvilinear preconditioning as described in Subsection 3.3. The reference curve \( y_r(t) \), see (50), fulfills \( y''_r(t) = -y'_r(t) \) so that \( y'_r(t), y''_r(t), y'''_r(t) = -y'(t) \in \mathbb{R}^4 \) are linearly dependent for all \( t \). This limits the depth of curvilinear coordinates to \( k = 2 \) which coincides with the default depth \( \lceil n/2 \rceil \) used in Listing 11. Thus, for suitable vectors \( e_1(t) \) and \( e_2(t) \), the matrix \( X = \tilde{X} \) defined in (3.3), where \( y_r(t) \) is viewed as a column vector, has the form

\[
X = \begin{pmatrix}
-\sin(t) & -\cos(t) \\
-\cos(t) & \sin(t) \\
-\cos(t) & \sin(t) \\
\sin(t) & \cos(t)
\end{pmatrix}
\begin{pmatrix}
e_1(t) \\
e_2(t)
\end{pmatrix}
\in \mathbb{R}^{4,4}.
\] (51)

\(^{16}\)Using additional blunting for parallelepiped preconditioning does not further improve the results.
It is easy to see that all standard unit vectors have the same Euclidean distance to the plane spanned by \( y_1'(t), y_2'(t) \) so that according to our approach \( e_1(t) \) and \( e_2(t) \) could have been chosen constant, for example \( e_1(t) \equiv (1, 0, 0, 0) \) and \( e_2(t) \equiv (0, 1, 0, 0) \).

We already mentioned in Subsection 3.3 that choosing \( e_1(t), e_2(t) \) in a different way, not as standard unit vectors, may give significantly larger remainders. For example, if we set \( Y := (y_1'(t), y_2'(t)) \in \mathbb{R}^{3,2} \) and use MATLAB’s command \( E = \text{null}(Y') \) to compute an orthonormal basis \( E = (e_1(t), e_2(t)) \) of the orthogonal complement of the span of \( y_1'(t), y_2'(t) \) and take \( X := (Y, E) \), then large remainder bounds of order \( 10^{-3} \) instead of order \( 10^{-8} \) are obtained for the first component \( y_1 \) as shown in the right picture of Fig. 8. The enclosure for \( y \) at \( t_f = 20\pi \) becomes significantly wider for all four components:

\[
\begin{align*}
[y_{1,1}] &= [-1.279331764659630\text{e-002}, 1.272813329184739\text{e-002}] \quad d([y_{1,1}]) = 2.56\text{e-02} \\
[y_{1,2}] &= [9.889349451442530\text{e-001}, 1.010916526589714\text{e+000}] \quad d([y_{1,2}]) = 2.20\text{e-02} \\
[y_{1,3}] &= [9.888763935674657\text{e-001}, 1.01091401138\text{e+000}] \quad d([y_{1,3}]) = 2.21\text{e-02} \\
[y_{1,4}] &= [-1.280319868022257\text{e-002}, 1.287705386044198\text{e-002}] \quad d([y_{1,4}]) = 2.57\text{e-02}
\end{align*}
\]

The enclosure for the first component \( y_1 \) now has a diameter \( 2.57 \cdot 10^{-2} \) instead of \( 2.37 \cdot 10^{-2} \) computed beforehand. This shows that the choice of \( e_1(t), e_2(t) \) is not arbitrary and we point out again that it is not clear to us how this is done best.

### 4.5 A Kepler problem for asteroid motion

In [16], p. 144 et seq., and [19] the motion of the asteroid 1997 XF11 is modeled by the Kepler problem

\[
y'' = -\gamma \frac{y}{(y_1^2 + y_2^2 + y_3^2)^{3/2}}, \quad \gamma := 0.9986,
\]

where \( y_1, y_2, y_3 \) are the \((x, y, z)\)-coordinates of the asteroid in the solar system with the sun in its center.\(^{17}\) Rewritten as a six-dimensional first order ODE, (52) reads

\[
y'_i = y_{i+3}, \quad y'_{i+3} = -\gamma \frac{y_i}{d}, \quad i = 1, 2, 3, \quad \text{where} \quad d := (y_1^2 + y_2^2 + y_3^2)^{3/2}.
\]

The following interval initial values are taken:

\[
\begin{align*}
y_1(0) &\in -1.77269098191512 \pm 0.5 \cdot 10^{-7} \\
y_2(0) &\in 0.1487214852342955 \pm 0.5 \cdot 10^{-7} \\
y_3(0) &\in -0.07928350462244194 \pm 0.5 \cdot 10^{-7} \\
y_4(0) &\in 0.2372031916516237 \pm 0.5 \cdot 10^{-6} \\
y_5(0) &\in 0.612524538758628 \pm 0.5 \cdot 10^{-6} \\
y_6(0) &\in 0.04583217572165624 \pm 0.5 \cdot 10^{-6}
\end{align*}
\]

Long term integration is done for 23 years. In scaled time units one year corresponds to \( 2\pi \). Thus, the integration period is

\[
t \in [t_0, t_f] := [0, 46\pi].
\]

In [9] we already considered this example and stated verified enclosures at \( t = t_f \) computed with blunted parallelepiped preconditioning without going into details. Here, like in the previous examples, we write down explicit, executable MATLAB/INTLAB code to make the example recomputable. Moreover, also QR preconditioning will be applied although this gives significantly worse results than blunted parallelepiped preconditioning. Parallelepiped preconditioning without blunting as well as curvilinear and identity preconditioning do not succeed integration for 23 years. Listing 12 implementations (53).

\(^{17}\)The problem was reconsidered in [13], p. 161 et seq.

35
function dy = kepler(t,y,i)
    persistent g.*
    if isempty(g.*)
        g.* = iv2intval(iv_rdivide(-9996.1,E4)); % verified inclusion of -0.9986
    end
    if nargin==2 || isempty(i) || i >= 4
        g = -0.9986; % nonverified branch for MATLAB's
        g = g.*;
    end
    d = sqrt(y(1)) + sqrt(y(2)) + sqrt(y(3));
    z = (d.'-1.5).*g; % note that 1.5 is a floating-point number
    if nargin==2 || isempty(i)
        dy = [y(4:6);y(1:3).*z];
    else
        dy = y(i-3).*z;
    end
end % function kepler

Listing 12. Kepler problem.

Listing 13 executes `verifyode` two times, with QR and blunted parallelepiped preconditioning. The latter uses the default blunting factor $\varepsilon = 0.001$, see (27).

Taylor model order 10 is taken, and, like in [9], a more sensitive so-called relative $\varepsilon$-inflation constant $\varepsilon_{rel} := 0.01$ (1 percent) is used instead of the default value 0.1 (10 percent). This is done by setting the option `eps_rel`, see lines 8 and 15.\(^{18}\)

```matlab
% execute verifyode with QR preconditioning
    disp('1. QR preconditioning')
    options = verifyodeset('order',10,'precondition','1','h0','0.1','h_min',0.001,...
        'loc_err_tol',1e-11,'sparsity_tol',1e-16,'eps_rel',0.01):
    tic,
    [T,QR,Y_QR,Yr_QR] = verifyode(@kepler,[t0,tf],y0,options); runtime_QR = toc
    disp(['Number of integration steps: ',num2str(length(T))]);
    ['.Y_QR_end'] = verifyode(T,QR,Y_Q_r,tf,[],'longe');
    disp('execute verifyode with blunted parallelepiped preconditioning
    disp('2. Blunted parallelepiped preconditioning with default blunting factor 0.001')
    options = verifyodeset('order',10,'precondition','2','blunting','1','h0','0.1....
        'h_min',0.001,'loc_err_tol','1e-11','sparsity_tol','1e-16','eps_rel',0.01):
    tic,
    [T_PEBL,Y_PEBL,Yr_PEBL] = verifyode([t0,tf],y0,options);
    runtime_PEBL = toc
    disp(['Number of integration steps: ',num2str(length(T))]);
    ['.Y_PEBL_end'] = verifyode(T,PB,Y_PEBL,Yr_PEBL,tf,[],'longe');
    % for comparison execute MATLAB's nonverified ODE solver ode45 with high accuracy
    options_ode45 = odeselect('RelTol',1e-14,'AbsTol',1e-14,'1 1 1 1');
    y_ode45_end = intval([]); A = [(dec2bin(0:63)-'0')',(1-1e-12)*rand(6,100)];
    for j = 1:size(A,2)
        a = A(:,j); y0 = a*y0.inf+(1-a).*y0.sup;
        [t_ode45,y_ode45] = ode45(@kepler,[t0,tf],y0,options_ode45);
        y_ode45_end = hull(y_ode45_end,y_ode45(end,:));
    end
    disp('Enclosure of several ode45 runs for distinct initial values in y0:')
    format longe, format insup
    disp(''), disp(y_ode45_end), format shorte
    disp(''), diameters_ode45 = diam(y_ode45_end)
    disp(''), disp('Relative width difference between inner and outer approximations:')
    disp(''), disp(1-diam(y_ode45_end)) ./ diam(1-y_ode45_end(end)) , format short
    % check whether all ode45 results are contained in the verifyode enclosures at t = tf
    all(in(y_ode45_end,y_QR_end)) && all(in(y_ode45_end,y_PEBL_end))
```
The code of Listing 13 is similar to that of Listing 11 for the muon cooling ring. Like there, a nonverified inner approximation is computed by MATLAB’s nonverified ODE solver ode45, see lines 20 to 31, and the relative differences between the diameters of the verified outer enclosures and the nonverified inner approximation are displayed, see lines 32, 33. Finally, line 35 checks whether the nonverified inner approximation is contained in the verified outer ones, which was always the case in our tests. Running Listing 13 gives the following results:

1. QR preconditioning
   runtime_QR = 4.2044e+03
   Number of integration steps: 1214
   t = 1.445132620651306e+02
   [y_1] = [ -9.21378656741531e-001, -8.861376608426567e-001 ]
   d([y_1]) = 3.53e-02
   [y_2] = [ -7.60683583748121e-001, -7.39076509634272e-001 ]
   d([y_2]) = 2.11e-02
   [y_3] = [ 7.309014090578357e-003, 9.967275593574382e-003 ]
   d([y_3]) = 2.66e-03
   [y_4] = [ 9.10643090131485e-001, 9.349927675055015e-001 ]
   d([y_4]) = 2.44e-02
   [y_5] = [ -4.06034724384662e-001, -3.878691286919924e-001 ]
   d([y_5]) = 1.82e-02
   [y_6] = [ 6.001870538191992e-002, 6.051063616154172e-002 ]
   d([y_6]) = 4.92e-04

2. Blunted parallelepiped preconditioning with default blunting factor 0.001
   runtime_PEBL = 1.4201e+04
   Number of integration steps: 1204
   t = 1.445132620651306e+02
   [y_1] = [ -9.037465020114251e-001, -9.031668499179916e-001 ]
   d([y_1]) = 5.80e-04
   [y_2] = [ -7.4958682536163e-001, -7.4958682536163e-001 ]
   d([y_2]) = 2.48e-04
   [y_3] = [ 8.639600651413365e-003, 8.676151550212157e-003 ]
   d([y_3]) = 3.66e-05
   [y_4] = [ 9.22845452175189e-001, 9.21950766272715e-001 ]
   d([y_4]) = 3.50e-04
   [y_5] = [ -3.97003123125e-001, -3.96720663715123e-001 ]
   d([y_5]) = 2.91e-04
   [y_6] = [ 6.02682416365615e-002, 6.02682416365615e-002 ]
   d([y_6]) = 4.27e-06

Enclosure of several ode45 runs for distinct initial values in y0:
   [-9.03723572749966e-001, -9.03189769637247e-001]
   [-7.49585560607393e-001, -7.49585560607393e-001]
   [8.64106651411336e-003, 8.676151550212157e-003]
   [9.22845452175189e-001, 9.21950766272715e-001]
   [-3.97003123125e-001, -3.96720663715123e-001]
   [6.02682416365615e-002, 6.02682416365615e-002]

Diameters ode45 =
   5.3381e-04
   2.9301e-04
   3.6326e-04
   3.2204e-04
   2.6870e-04
   3.6922e-06

Relative width difference between inner and outer approximations:
   9.8486e-01 9.8910e-01 9.8735e-01 9.8677e-01 9.8521e-01 9.9250e-01 <- QR preconditioning
   7.9098e-02 7.5507e-02 7.9873e-02 7.8443e-02 7.5384e-02 1.3504e-01 <- blunted parallelepiped prec.

ans =
   logical
   1

The diameters of the enclosures obtained with blunted parallelepiped preconditioning are by two orders smaller than those obtained with QR preconditioning. This shows that the latter gives rather poor results. The former diameters deviate from those of the nonverified inner approximation computed with MATLAB’s ode45 by roughly ten percent which seems acceptable for this long integration period.

Table II compares the results of COSY\(^19\) stated in [13], p. 163, Table 5.6, and those of verifyode executed with blunted parallelepiped preconditioning. The latter are slightly tighter than the former but the diameters, stated in the columns \(d([y])\), are of the same order.

Runtimes cannot be compared directly since different computers are used. Moreover, COSY is compiled FORTRAN code while verifyode is interpreted MATLAB code so that runtime comparison is not really fair anyway. However, COSY’s runtime of 4.53 hours stated in [13] and the runtime of 3.95 hours needed by verifyode may still give some impression.

\(^{19}\)Newer releases of COSY may perform differently and may give different enclosures.
Table II. Enclosures for the Kepler problem at $t_f = 46\pi$.

| $y_1$ | COSY (QR-BL) | verifyode (PEBL) |
|-------|---------------|------------------|
|       | $d([y])$     | $d([y])$        |
| $y_1$ | $-9.035^{+656}_{-513} e^{-1}$ | $9.87 e^{-4}$   |
| $y_2$ | $-7.495^{+944}_{-525} e^{-1}$ | $4.32 e^{-4}$   |
| $y_3$ | $8.675^{+153}_{-669} e^{-3}$  | $6.70 e^{-5}$   |
| $y_4$ | $9.231^{+148}_{-273} e^{-1}$  | $5.91 e^{-4}$   |
| $y_5$ | $-3.967_{-768}^{+055} e^{-1}$ | $4.96 e^{-4}$   |
| $y_6$ | $6.027_{-0.357}^{+035} e^{-2}$| $6.78 e^{-6}$   |

| time [s] | $16316 \pm 4.53 h$ | $14201 \pm 3.95 h$ |

Table III. Ranking of preconditioning methods.

| Example                                      | ID | QR | PE | PEBL | CL |
|----------------------------------------------|----|----|----|------|----|
| 1. quadratic model problem                   | 1  | 3  | -  | -    | 2  |
| 2. Higgins-Selkov oscillator                 | -  | 2  | 1  | n.a. | 3  |
| 3. Roessler attractor, $t_f := 12$           | 3  | 1  | -  | -    | 2  |
| 4. muon cooling ring                         | -  | 3  | 1  | n.a. | 2  |
| 5. Kepler problem                            | -  | 2  | -  | 1    | -  |

5. Conclusion

We reviewed preconditioning of Taylor models as proposed by Makino and Berz in [7], namely:

1. parallelepiped preconditioning (PE)
2. blunted parallelepiped preconditioning (PEBL)
3. QR preconditioning (QR)
4. curvilinear preconditioning (CL)
5. identity preconditioning (ID)

All of them are linear methods meaning that preconditioning is in essence done by multiplication with a suitable matrix $G$, see (21), (22). The listed methods only differ in the choice of $G$.

Along with explaining main principles and difficulties of preconditioning we stated explicit MATLAB code that implements preconditioning in our verified ODE solver verifyode. The code is presented in the form of the current, new version 12 of INTLAB [25].

In order to compare the different kinds of preconditioning and to test our implementation we carried out various numerical simulations for several examples of initial value problems. For each example we stated a complete executable MATLAB program that produces our numerical results including all presented pictures. We believe that in this way everything becomes fully retrievable and reproducible.

Table III contains for each example a rough ranking of the suitability of the respective preconditioning methods according to our numerical results. An entry “1” means best result, a dash “-” indicates the failure of the preconditioning method, and “n.a.” means that the method was not applied.

Although far from giving a complete picture, Table III may already suggest that in general there is no overall method for choosing the best preconditioning type somehow automatically in advance. This choice is part of the user’s duty and may come down to a “Try-and-Error” approach.
Note that even simple identity preconditioning may sometimes give good or at least sufficient results. However, QR preconditioning seems to be a stable method working for all examples. In contrast, parallelepiped preconditioning may fail if badly conditioned linear parts of Taylor models occur, even if additional blunting is used. Nevertheless, if this is not the case, parallelepiped preconditioning with or without blunting may outperform the other methods by several orders as can be seen for the difficult example of long term integration of the Kepler problem.

Acknowledgments
I thank S.M. Rump for reading the manuscript and for many helpful discussions and suggestions. Finally, my sincere thanks go to the unknown referee for many valuable remarks and suggestions.

References
[1] Berz, M., et al., The COSY INFINITY web page, www.bt.pa.msu.edu/index_cosy.htm
[2] M. Berz and K. Makino, “Preservation of canonical structure in non-planar curvilinear coordinates,” International Journal of Applied Mathematics, vol. 3, no. 4, pp. 401–419, 2000.
[3] M. Berz and K. Makino, “Perturbative equations of motion and differential operators in non-planar curvilinear coordinates,” International Journal of Applied Mathematics, vol. 3, no. 4, pp. 421–440, 2000.
[4] M. Berz and K. Makino, Suppression of the wrapping effect by Taylor model-based validated integrators, MSU report MSUHEP-40910, Michigan State University, 2003. available from https://bt.pa.msu.edu/cgi-bin/display.pl?name=VIRC03
[5] M. Berz and K. Makino, “Suppression of the wrapping effect by Taylor model-based verified integrators: the single step,” International Journal of Pure and Applied Mathematics, vol. 36, no. 2, pp. 175–197, 2006.
[6] M. Berz and K. Makino, “Suppression of the wrapping effect by Taylor model-based verified integrators: long-term stabilization by shrink wrapping,” International Journal of Differential Equations and Applications, vol. 10, no. 4, pp. 385–403, 2005.
[7] M. Berz and K. Makino, “Suppression of the wrapping effect by Taylor model-based verified integrators: long-term stabilization by preconditioning,” International Journal of Differential Equations and Applications, vol. 10, no. 4, pp. 105–136, 2005.
[8] M. Berz and K. Makino, Taylor model methods and computer assisted proofs, Lecture Notes, Course Presented at Department of Mathematics, University of Barcelona, June 2008, MSU report MSUHEP-080609, Michigan State University, 2008. available from https://bt.pa.msu.edu/cgi-bin/display.pl?name=CAP08
[9] F. Bünger, “A Taylor model toolbox for solving ODEs implemented in MATLAB/INTLAB,” Journal of Computational and Applied Mathematics, vol. 368, article 112511, 2020.
[10] X. Chen, Reachability analysis of non-linear hybrid systems using Taylor models, Dissertation at RWTH Aachen University, 2015. Software: Flow*, https://flowstar.org/downloads/
[11] X. Chen and S. Sankaranarayanan, “Decomposed reachability analysis for nonlinear systems” in 2016 IEEE Real-Time Systems Symposium (RTSS), pp. 13–24, IEEE Computer Society Conference Publishing Services, 2016. available from https://ieeexplore.ieee.org/document/7809839
[12] T. Dzetkulić, “Rigorous integration of non-linear ordinary differential equations in Chebyshev basis,” Numer. Algorithms, vol. 69, pp. 183–205, 2015. Software: ODEintegrator, https://sourceforge.net/projects/odeintegrator/
[13] I. Eble, Über Taylor-Modelle, Dissertation at Karlsruhe Institute of Technology, 2007. Software: RIOT, www.math.kit.edu/ianm1/~ingo.eble/de
[14] J. Grote, M. Berz, and K. Makino, “High-order representation of Poincaré maps,” Lecture Notes on Computational Science and Engineering, vol. 50, pp. 59–66, 2005.
[15] J. Grote, M. Berz, and K. Makino, “High-order representation of Poincaré maps,” Nuclear Instruments and Methods in Physics Research Section A, vol. 558, no. 1, pp. 106–111, 2006.
[16] J. Hoefkens, *Rigorous numerical analysis with high-order Taylor models*, Dissertation at Michigan State University, 2001.

[17] J. Hoefkens, M. Berz, and K. Makino, “Verified high-order integration of DAEs and higher-order ODEs” in *Scientific Computing, Validated Numerics and Interval Methods*, ed. W. Kraemer, J.W.v. Gudenberg, Kluwer Academic Publishers, Boston, Dordrecht, London, 2001.

[18] J. Hoefkens, M. Berz, and K. Makino, “Computing validated solutions of implicit differential equations,” *Advances in Computational Mathematics*, vol. 19, pp. 231–253, 2003.

[19] J. Hoefkens, M. Berz, and K. Makino, “Controlling the wrapping effect in the solution of ODEs for asteroids,” *Reliab. Comput.*, vol. 8, pp. 21–41, 2003.

[20] R. Lohner, *Einschließung der Lösung gewöhnlicher Anfangs- und Randwertaufgaben und Anwendungen*, Dissertation at Karlsruhe Institute of Technology, 1988. Software: AWA, http://www2.math.uni-wuppertal.de/org/WRST/xsc-frame/xsc/pxsc_software.html#awa

[21] K. Makino, *Rigorous analysis of nonlinear motion in particle accelerators*, Dissertation at Michigan State University, 1998.

[22] N.S. Nedialkov and K.R. Jackson, “A new perspective on the wrapping effect in interval methods for initial value problems for ordinary differential equations,” in *Perspectives on Enclosure Methods*, ed. U. Kulisch, R. Lohner, A. Facius, pp. 219–263, Springer, Vienna, 2001.

[23] M. Neher, K.R. Jackson, and N.S. Nedialkov, “On Taylor model based integration of ODEs,” *SIAM Journal on Numerical Analysis*, vol. 45, no. 1, pp. 236–262, 2007.

[24] M. Neher, K.R. Jackson, and N.S. Nedialkov, “On the blunting method in verified integration of ODEs,” *Reliab. Comput.*, vol. 23, pp. 15–34, 2016.

[25] S.M. Rump, “INTLAB - INTe rval LABoratory” in *Developments in Reliable Computing*, ed. Tibor Csendes, pp. 77–104, Kluwer Academic Publishers, Dordrecht, 1999. Software: INTLAB, http://www.ti3.tu-harburg.de/intlab/