





RICHARD HASENFELDER¹, LUTZ LEHMANN, MANUEL RADONS², TOM
STREUBEL³, CHRISTIAN STROHM⁴, ANDREAS GRIEWANK⁵


Computational aspects of the Generalized Trapezoidal Rule

¹  0000-0001-9699-5054

²  0000-0003-4272-2493

³  0000-0003-4917-7977

⁴  0000-0002-8876-7120

⁵  0000-0001-9839-1473

Zuse Institute Berlin
Takustr. 7
14195 Berlin
Germany

Telephone: +49 30-84185-0
Telefax: +49 30-84185-125

E-mail: bibliothek@zib.de
URL: <http://www.zib.de>

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

Computational aspects of the Generalized Trapezoidal Rule

Richard Hasenfelder¹, Lutz Lehmann¹, Manuel Radons², Tom Streubel^{3, 1}, Christian Strohm¹, and Andreas Griewank⁴

¹Humboldt University of Berlin, Germany

²Technical University of Berlin, Germany

³Zuse Institute Berlin, Germany

⁴School of Information Sciences Yachaytech, Ecuador

May 9, 2018

Abstract

In this article we analyze a generalized trapezoidal rule for initial value problems with piecewise smooth right hand side $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$. When applied to such a problem, the classical trapezoidal rule suffers from a loss of accuracy if the solution trajectory intersects a non-differentiability of F . In such a situation the investigated generalized trapezoidal rule achieves a higher convergence order than the classical method. While the asymptotic behavior of the generalized method was investigated in a previous work, in the present article we develop the algorithmic structure for efficient implementation strategies and estimate the actual computational cost of the latter. Moreover, energy preservation of the generalized trapezoidal rule is proved for Hamiltonian systems with piecewise linear right hand side.

Keywords Algorithmic Differentiation, Automatic Differentiation, Lipschitz Continuity, Piecewise Linearization, Nonsmooth, Trapezoidal Rule, Implementation, Computational Cost

MSC 2010 65L05, 65L06, 65L70, 65L99, 65P10

1 Introduction

Many realistic computer models are nondifferentiable in that the functional relation between input and output variables is not smooth. It was shown in [5, 8] that *algorithmic piecewise linearization*, which is a generalized form of *algorithmic differentiation* (AD), can be used to obtain piecewise linear models that approximate an underlying piecewise smooth function locally with second order error. In a number of recently developed algorithms that deal with

nonsmooth problems – e.g. Newton methods for piecewise smooth nonlinear systems [8] or solvers for ordinary differential equations (ODEs) with piecewise smooth right hand side [7] – these piecewise linear approximations take over the roles that local linear approximations have in the methods for the globally differentiable case.

In this article we focus on the analysis of implementational aspects of the generalized trapezoidal rule for initial value problems with piecewise smooth right hand side $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$ that was developed in [5] and analyzed in [7]. In [5] and [7] the following problem is considered: Let

$$\dot{x}(t) = F(x(t)), \quad x(0) = x_0, \quad (1)$$

be an initial value problem for an autonomous ODE, where $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is assumed to be locally Lipschitz continuous. It is well known that this system has a unique local solution up to some time $\bar{t} > 0$. For a time-step $h > 0$ the exact solution of (1) satisfies

$$\hat{x} = \check{x} + \int_0^h F(x(t))dt,$$

with $\hat{x} = x(h)$ and $\check{x} = x_0$. In general the integral cannot be evaluated exactly.

In the derivation of the classical trapezoidal rule a linear approximation of the right hand side is utilized. The integration of these approximations yields a third order local truncation error if F is smooth. If F is only Lipschitz continuous, the truncation error will drop to second order where the solution trajectory intersects a nondifferentiability.

The key idea in [7] to reestablish a third order truncation error everywhere is to approximate F by a *piecewise linear* function that reflects the structure of the nondifferentiabilities of F . Employing this approach allowed to construct a generalized trapezoidal rule with the following major benefits:

- Second order convergence is achieved in general and third order via Richardson extrapolation along solution trajectories with finitely many kink locations.
- A third order interpolating polynomial as continuous approximation of the trajectory is given.

Moreover, it was suggested that the method is energy preserving for Hamiltonian systems with piecewise linear right hand side. The latter is now formally proved in Section 3. However, the main result of this article concerns implementational aspects. To achieve a predefined approximation error, the generalized trapezoidal rule needs fewer steps than the classical method. But, if implemented naively, these steps are computationally more expensive than those of the classical method, which essentially equalizes the advantage of the generalized method's higher convergence order. In Section 4 we present efficient implementation strategies for the generalized trapezoidal rule, through which its overall computational effort can be pushed below that of the classical method. The theoretical results will be validated numerically in Section 5. More specifically, in addition to an ODE example we will consider the application to semi-explicit DAE system and a finite volume discretized PDE system. A brief introduction to the necessary terminology is given in Section 2.

2 Algorithmic Piecewise Differentiation

In this section we will present the tangent and secant linearization modes which were developed and analyzed in [5] and [7], respectively.

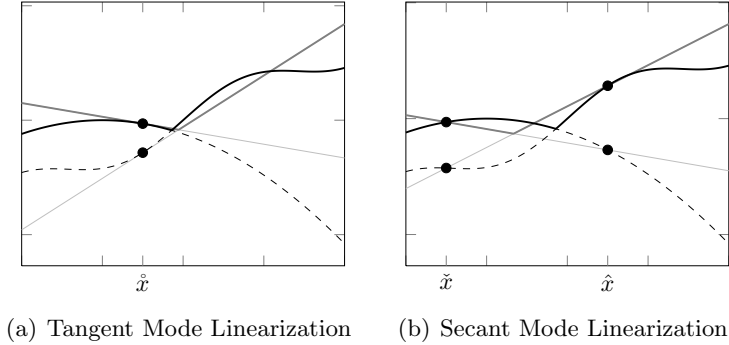


Figure 1: Piecewise linearization modes

2.1 Piecewise Linear Functions

Definition 2.1. A continuous function $F : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is called **piecewise linear** if there exists a finite number of affine **selection functions** $F_i : \mathbb{R}^n \rightarrow \mathbb{R}^m$ such that at any given $x \in \mathbb{R}^n$ there exists at least one index i with $F(x) = F_i(x)$.

Let the index set $I = \{1, \dots, k\}$ of the selection functions be given. According to [13, Prop. 2.2.2] we can find subsets $M_1, \dots, M_l \subset I$ such that a scalar valued piecewise linear function f can be represented as

$$f(x) = \max_{1 \leq i \leq l} \min_{j \in M_i} f_j(x).$$

This concept, which is called **max-min representation**, naturally carries over to vector valued functions F , where we can find such a decomposition for every component of the image. The special type of piecewise linear functions that will be utilized here will naturally have this representation.

Note that piecewise linear functions are globally Lipschitz continuous. For a further discussion of their properties we refer to [13].

2.2 Piecewise Linearization of Piecewise Composite Smooth Functions

Next we consider continuous, piecewise differentiable functions F that can be computed by a finite program called an *evaluation procedure*. An evaluation procedure is a composition of so-called elementary functions which make up the atomic constituents of more complex

functions. Basically, the selection of elementary functions for the library is arbitrary, as long as they comply with assumption (ED) (elementary differentiability, in [6]), meaning that they are at least once Lipschitz-continuously differentiable on their valid open domains. Common examples are:

$$\Phi := \{+, -, *, /, \sin, \cos, \tan, \cot, \exp, \log, \dots\}.$$

In our case, we will allow the evaluation procedure of $F : D \subseteq \mathbb{R}^n \rightarrow \mathbb{R}^m$ to contain, in addition to the usual smooth elementary functions, the absolute value $\text{abs}(x) = |x|$, i.e., our library is of the form

$$\Phi_{\text{abs}} := \Phi \cup \{\text{abs}\}.$$

Consequently, we can also handle the maximum and minimum of two values via the representation

$$\max(u, v) = (u + v + |u - v|)/2, \quad \min(u, v) = (u + v - |u - v|)/2.$$

We call the resulting functions *piecewise composite smooth* (PCS). These functions are locally Lipschitz continuous and almost everywhere differentiable in the classical sense. Furthermore, they are differentiable in the sense of Bouligand and Clarke, cf. [2]. The evaluation procedure of $y = F(x)$ can be interpreted as a directed, acyclic graph from $x = (v_{1-n}, \dots, v_0)$ to $y = (v_{l-m+1}, \dots, v_l)$, where the intermediate values $v_i, i = 1, \dots, l$ are computed by binary operations $v_i = v_j \circ v_k$ with $\circ \in \{+, -, *\}$ and $v_j, v_k \prec v_i$ or unary functions $v_i = \phi_i(v_j)$ with $v_j \prec v_i$, where $\phi_i \in \Phi_{\text{abs}}$. The relation \prec represents the data dependence in the graph of the evaluation procedure, which must be acyclic.

We now want to compute an incremental approximation $\Delta y = \Delta F(\overset{\circ}{x}; \Delta x)$ to $F(\overset{\circ}{x} + \Delta x) - F(\overset{\circ}{x}) = F(x) - F(\overset{\circ}{x})$ at a given $\overset{\circ}{x}$ and for a variable increment $\Delta x = x - \overset{\circ}{x}$. Assuming that all functions other than the absolute value are differentiable, we introduce the propagation rules

$$\begin{aligned} \Delta v_i &= \Delta v_j \pm \Delta v_k && \text{for } \overset{\circ}{v}_i = \overset{\circ}{v}_j \pm \overset{\circ}{v}_k, \\ \Delta v_i &= \overset{\circ}{v}_j * \Delta v_k + \Delta v_j * \overset{\circ}{v}_k && \text{for } \overset{\circ}{v}_i = \overset{\circ}{v}_j * \overset{\circ}{v}_k, \\ \Delta v_i &= \overset{\circ}{c}_{ij} \Delta v_j \quad \text{with } \overset{\circ}{c}_{ij} = \varphi'(\overset{\circ}{v}_j) && \text{for } \overset{\circ}{v}_i = \varphi_i(\overset{\circ}{v}_j) \neq \text{abs}(\cdot), \\ \Delta v_i &= \text{abs}(\overset{\circ}{v}_j + \Delta v_j) - \text{abs}(\overset{\circ}{v}_j) && \text{for } \overset{\circ}{v}_i = \text{abs}(\overset{\circ}{v}_j). \end{aligned} \tag{2}$$

Whenever F is globally differentiable or even more when it complies to assumption (ED) (i.e., there are no abs calls in the evaluating procedure) we get $\Delta y = F'(\overset{\circ}{x})\Delta x$, where $F'(\overset{\circ}{x}) = \frac{\partial}{\partial x} F(\overset{\circ}{x}) \in \mathbb{R}^{m \times n}$ is the Jacobian matrix.

Note that the propagation rules (2) rely on the so-called tangent approximation of F at a certain point $\overset{\circ}{x}$. However, there are applications of piecewise linearization (especially concerning ODE integration) where one wants to consider approximations of F based on secants. Given two points \check{x}, \hat{x} we compute $\overset{\circ}{x} = (\check{x} + \hat{x})/2$ and $\overset{\circ}{F} = (F(\check{x}) + F(\hat{x}))/2$. Now we consider the secant approximation of F :

$$F(x) \approx \overset{\circ}{F} + \Delta F(\check{x}, \hat{x}; x - \overset{\circ}{x}). \tag{3}$$

In order to utilize AD for the algorithmic computation of the secant approximation in (3) we observe that in (2) the intermediate values can be regarded as functions evaluated at the unique reference point \hat{x} , with $\hat{v}_i = v_i(\hat{x})$. Now consider as this reference point the midpoint $\hat{x} = (\tilde{x} + \hat{x})/2$ such that the intermediate values are

$$\hat{v}_i = (\check{v}_i + \hat{v}_i)/2, \text{ with } \check{v}_i = v_i(\tilde{x}), \hat{v}_i = v_i(\hat{x}).$$

Replacing \hat{v}_i in (2) with this expression based on \tilde{x} and \hat{x} , we observe that the first and second line are the same for the secant linearization. The third line has to be changed slightly, since the tangent slope \hat{c}_{ij} has to be replaced by the secant slope

$$\hat{c}_{ij} = \begin{cases} \varphi'_i(\hat{v}_j) & \text{if } \check{v}_j = \hat{v}_j \\ \frac{\hat{v}_i - \check{v}_i}{\hat{v}_j - \check{v}_j} & \text{otherwise} \end{cases} \quad (4)$$

The last rule is left unchanged except that now $\hat{v}_i = (\check{v}_i + \hat{v}_i)/2 = (|\check{v}_j| + |\hat{v}_j|)/2$. Note that, if $\tilde{x} = \hat{x}$, we obtain $\Delta F(\hat{x}, x - \hat{x}) = \Delta F(\tilde{x}, \hat{x}; x - \hat{x})$. A complete discussion on this implementation topic can be found in [8, Sec. 7]. Additionally, a division-free and thus numerically stable implementation is discussed in [8, Section 6]. Both piecewise linearization methods yield a second order local fit and are Lipschitz continuous with respect to variations of the development point, cf [8, Prop. 2.1].

And once again if F complies to assumption (ED) (i.e. it consists of operations from the library of smooth operations Φ only) the piecewise linear secant increment coincides with a matrix-vector product of a propagated secant matrix $\nabla_{\hat{x}} F$ and a direction, i.e. $\Delta F(\hat{x}, \tilde{x}; x - \hat{x}) = \nabla_{\hat{x}} F \cdot (x - \hat{x})$. Propagated secant matrix means that its entries are calculated with the table of rules (2) and using the secant slope formula (4) for \hat{c}_{ij} .

2.3 The Abs-Normal Form

Let $F : \mathbb{R}^n \rightarrow \mathbb{R}^m$ be an evaluation procedure consisting of operations Φ_{abs} . Then there are smooth vector valued evaluation procedures $G : \mathbb{R}^n \times \mathbb{R}^s \rightarrow \mathbb{R}^s$ and $\tilde{F} : \mathbb{R}^n \times \mathbb{R}^s \rightarrow \mathbb{R}^m$ of operations in Φ and hence complying (ED), such that F can be expressed as follows:

$$\begin{aligned} z &= G(x, |z|) \\ F(x) &= \tilde{F}(x, |z|), \end{aligned} \quad (5)$$

and such that the partial derivative matrix $\frac{\partial}{\partial |z|} G(x, |z|)$ is strictly lower triangular as a consequence of the acyclic graph representation of F . Applying an Taylor Expansion scheme on (5) up to a first order we obtain

$$\begin{bmatrix} z \\ y \end{bmatrix} = \begin{bmatrix} G(\hat{x}, |\hat{z}|) \\ \tilde{F}(\hat{x}, |\hat{z}|) \end{bmatrix} + \begin{bmatrix} \frac{\partial}{\partial x} G(\hat{x}, |\hat{z}|) & \frac{\partial}{\partial |z|} G(\hat{x}, |\hat{z}|) \\ \frac{\partial}{\partial x} \tilde{F}(\hat{x}, |\hat{z}|) & \frac{\partial}{\partial |z|} \tilde{F}(\hat{x}, |\hat{z}|) \end{bmatrix} \cdot \begin{bmatrix} x - \hat{x} \\ |z| - |\hat{z}| \end{bmatrix}. \quad (6)$$

The system (6) is a piecewise linear operator mapping $x \in \mathbb{R}^n$ to $y \in \mathbb{R}^m$ via intermediate vector switching variables $z \in \mathbb{R}^s$. Hence, it can be considered as a piecewise linearization of F in that $y = F(\hat{x}) + \Delta F(\hat{x}; x - \hat{x})$ holds.

With $Z \equiv \frac{\partial}{\partial x} G$, $L \equiv \frac{\partial}{\partial |z|} G$, $J \equiv \frac{\partial}{\partial x} F$, $Y \equiv \frac{\partial}{\partial |z|} F$ as well as vectors $c \equiv G(\hat{x}, |\hat{z}|) - L|\hat{z}|$ and $b \equiv \tilde{F}(\hat{x}, |\hat{z}|) - Y|\hat{z}|$, the system (6) is brought into the so called *abs-normal form* (ANF) as studied e.g. in [9]:

$$\begin{bmatrix} z \\ y \end{bmatrix} = \begin{bmatrix} c \\ b \end{bmatrix} + \begin{bmatrix} Z & L \\ J & Y \end{bmatrix} \cdot \begin{bmatrix} x - \hat{x} \\ |z| \end{bmatrix}. \quad (7)$$

The ANF is a block matrix and a block vector that allows for a compact and easy-to-evaluate representation of the polyhedral structure inherent to the piecewise linearization.

Similar to the tangent case an order 1 or linear expansion of the elementary differentiable functions G and \tilde{F} can be applied in the sense of secant linearization as defined by the table of rules (2) and the secant slope formula (4) for \hat{c}_{ij} . Thus again, by introducing vectors $c \in \mathbb{R}^s$, $b \in \mathbb{R}^m$ and matrices $Z \in \mathbb{R}^{s \times n}$, $L \in \mathbb{R}^{s \times s}$, $J \in \mathbb{R}^{m \times n}$, $Y \in \mathbb{R}^{m \times s}$:

$$c \equiv \hat{G} - L \cdot |\hat{G}| \in \mathbb{R}^s, \quad b \equiv \hat{F} - Y \cdot |\hat{F}| \in \mathbb{R}^m, \quad [Z \ L] = \nabla_{\hat{x}, |\hat{z}|} G, \quad [J \ Y] = \nabla_{\hat{x}, |\hat{z}|} F,$$

where $\hat{G} \equiv (G(\tilde{x}, |\tilde{z}|) + G(\hat{x}, |\hat{z}|))/2$, $\hat{F} \equiv (F(\tilde{x}, |\tilde{z}|) + F(\hat{x}, |\hat{z}|))/2$ and $\hat{x} = (\tilde{x} + \hat{x})/2$ we can define a secant piecewise linear operator in ANF (7).

3 Generalized Trapezoidal Rule

For the generalization of the trapezoidal rule, the linearization mode of choice is the secant mode. The construction largely follows the classical case (see, e.g. [1]) where the integrand $y(t) \equiv F(x(t))$ is replaced by a linear secant interpolant of $y(\tilde{t})$ and $y(\hat{t}) = y(\tilde{t} + h)$. Instead of choosing a linear approximation of the right-hand-side's (RHS) function we utilize a piecewise linear secant interpolation. By doing so, we arrive at the following defining equation, which was introduced by Griewank in [5]:

$$\hat{x} - \tilde{x} = hQ_F \equiv h \int_{-\frac{1}{2}}^{\frac{1}{2}} \left[\hat{F} + \Delta F(\tilde{x}, \hat{x}; t(\hat{x} - \tilde{x})) \right] dt, \quad (8)$$

where \tilde{x} is the current and \hat{x} the next point on the numerical trajectory. The restriction of ΔF to the segment $[\tilde{x}, \hat{x}] = \{\hat{x} + \tau(\tilde{x} - \hat{x}) \mid \tau \in [-1/2, 1/2]\}$ defines a subdivision $-1/2 = \tau_0 < \tau_1 < \dots < \tau_\mu = 1/2$ of the parameter interval so that the function $t \mapsto \Delta F(\tilde{x}, \hat{x}, t(\hat{x} - \tilde{x}))$ is linear on each of the sub-intervals $[\tau_{i-1}, \tau_i]$, $i = 1, \dots, \mu$. We call the $\mu - 1$ sub-division points $\tau_1, \dots, \tau_{\mu-1}$ *critical multipliers*. Each kink of the piecewise linear interpolation corresponds to one of the critical multipliers.

The piecewise linear quadrature on the right hand side of equation (8) is then a sum of areas of trapezoids:

$$Q_F \equiv \sum_{i=1}^{\mu} (\tau_i - \tau_{i-1}) \left[\frac{\hat{F} + \tilde{F}}{2} + \frac{\Delta F(\tilde{x}, \hat{x}; \tau_i(\hat{x} - \tilde{x})) + \Delta F(\tilde{x}, \hat{x}; \tau_{i-1}(\hat{x} - \tilde{x}))}{2} \right]. \quad (9)$$

This generalization has the property of preserving the third order local truncation error of the classical trapezoidal rule. By implementing Richardson extrapolation, one can even achieve a third order global convergence order compared to the second order global error of the classical method [7].

3.1 Energy Preservation

The following statement was conjectured but not proved in [7].

Proposition 3.1. *The generalized trapezoidal rule is energy preserving for Hamiltonian systems with piecewise linear right hand side. Generally for any vector field F that can be written as $F(x) = A \nabla V(x)$ with a skew-symmetric matrix A the integral curves will preserve the value of the function V . If again F is also piecewise linear, V is also preserved under the generalized trapezoidal rule.*

Proof. We know that for a piecewise linear right hand side F the generalized trapezoidal rule simplifies as follows

$$\hat{x} - \check{x} = h \int_0^1 \frac{F(\check{x}) + F(\hat{x})}{2} + \Delta F(\check{x}, \hat{x}; (\tau - \frac{1}{2})[\hat{x} - \check{x}]) d\tau = h \int_0^1 F((1 - \tau)\check{x} + \tau\hat{x}) d\tau,$$

since a piecewise linear function is its own secant piecewise linearization w.r.t. (3). Now we can proceed as in [12], set $\bar{\Delta}V(\check{x}, \hat{x}) = \int_0^1 \nabla V((1 - \tau)\check{x} + \tau\hat{x}) d\tau$ and observe that

$$V(\hat{x}) - V(\check{x}) = \bar{\Delta}V(\check{x}, \hat{x})^\top (\hat{x} - \check{x}) = \bar{\Delta}V(\check{x}, \hat{x})^\top h A \bar{\Delta}V(\check{x}, \hat{x}) = 0. \quad (10)$$

This completes the proof. \square

3.2 Implementation Strategies

To implement the implicit method $\hat{x} - \check{x} = hQ_F(\check{x}, \hat{x})$ there are two main strategies, whose merits strongly depend on the system type which is to be solved.

Fixed Point Approach

The direct implementation as the fixed-point iteration

$$\hat{x}_{m+1} = \check{x} + hQ_F(\check{x}, \hat{x}_m) \quad (11)$$

has the advantage that it can be realized without explicit computation of the ANF, since the only requirement for its realization is to calculate the critical multipliers. This is presented in Section 4.2. The upside of this ansatz is that the cost of up to $3(n + s)$ function evaluations for the calculation of the ANF is saved, which is especially advantageous for high-dimensional problems. The downside is the linear convergence rate of $O(h)$ with a possibly small convergence radius.

Newton Approach

The Newton-based approach transfers a large part of the right hand side to the left, that is, it solves a linearization in \hat{x}_{m+1} of

$$\hat{x}_{m+1} - h \frac{F(\tilde{x}) + F(\hat{x}_{m+1})}{2} = hQ_F(\tilde{x}, \hat{x}_m) - h \frac{F(\tilde{x}) + F(\hat{x}_m)}{2}. \quad (12)$$

The piecewise linearization has the form

$$\hat{x}_{m+1} - \frac{h}{2} \diamond_{\tilde{y}}^{\hat{y}} F(\hat{x}_{m+1}) = hQ_F(\tilde{x}, \hat{x}_m) - \frac{h}{2} \diamond_{\tilde{y}}^{\hat{y}} F(\hat{x}_m). \quad (13)$$

The most natural choices for the basis points are $(\tilde{y}, \hat{y}) = (\tilde{x}, \tilde{x})$ in tangent mode where the linearization is computed once per integration step, and $(\tilde{y}, \hat{y}) = (\tilde{x}, \hat{x}_m)$ in secant mode where one would have to renew the piecewise linearization in each iteration. In the tangent variant, the critical multipliers are computed using the cheaper ANF-free ELEMOP while in the secant case the ANF can also be used for this purpose, as will be explained in Section 4.1. Since the basis points are in close proximity and both piecewise linearization modes yield quadratic approximations, both Newton-type methods produce comparable results in terms of runtime and accuracy.

Comparison of Fixed Point and Newton Approach on Toy Example

In order to obtain an intuition for merits and limitations of both the fixed point and the Newton approach we compare them on a toy example, the ODE

$$\dot{x} = F(x) = \max(1, x).$$

As the right side is itself already piecewise linear, all piecewise linearizations are identical to it. The Lipschitz constant is $\beta_F = 1$, so that the first estimate for admissible step sizes based on the standard trapezoidal rule gives $h < \frac{2}{\beta_F} = 2$. Let $a \in [0, 1]$. Setting the initial point

$$x(0) = \tilde{x} = 1 - ah$$

to the left of the kink so that the step with size h traverses it, one finds that the initial error of $\hat{x}_0 = \tilde{x} + h$ is

$$\frac{1}{2}(1-a)^2 h^2 + O(h^3).$$

The contraction constant for the fixed point iteration corrector is about $q = \frac{h}{2}(1-a^2) \in O(h)$, providing convergence independently of $a \in [0, 1]$ for $h < 2$. The Newton corrector also has linear convergence with a contraction coefficient $q = -\frac{h}{2-h}a^2$ which will only converge independently of $a \in [0, 1]$ if $h < 1$, which is half the step size bound of the fixed point method.

The combination of one predictor and one corrector step has an error estimate of $\frac{h^3}{4}(1-a)^3(1+a)$ for the direct method and $-\frac{h^3}{4-2h}a^2(1-a)^2$ for the Newton method. The latter error

term is more balanced for variations of a with a maximum value of $\frac{h^3}{64(1-h/2)}$ while the error term of the fixed point method has a maximum for $a = 0$ of $\frac{h^3}{4}$. So, while both approaches, fixed point and Newton, display the same order of convergence, the Newton-type iteration has a significantly smaller error term. Throughout the experiments presented in Section 5 this will result in the observation that in those cases where it is computationally feasible to compute the ANF, the Newton-type iteration outperforms its fixed point counterpart, while the fixed point approach is superior when there are many switching variables and thus a large ANF.

Both variants can be extended for the solution of index-1 semi-explicit DAEs by including the algebraic part of the system in the piecewise linearization and its subsequent solution process.

4 Implementation Details

In the following we will describe the structure of the classic and the generalized trapezoidal rule. Pseudo-code is given for both of them in Algorithm 1 and 2. The generalized solver is also depicted in Figure 2.

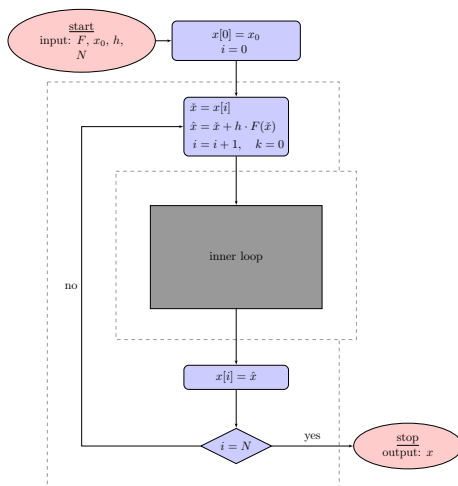


Figure 2: Structure of the Generalized Solver

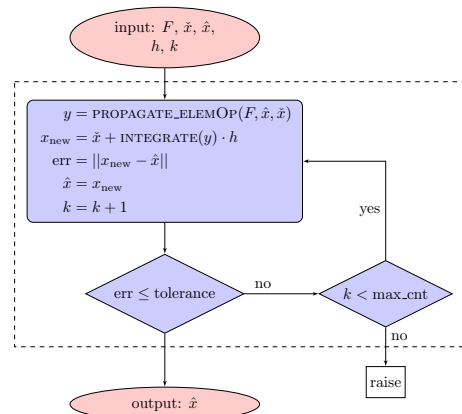


Figure 3: Inner loop with fixed-point solver

Firstly, one has to discretize the time interval. This can be done uniformly or adaptively. The discretization along with the RHS function F and the initial value(s) x_0 are the minimal necessary input for both solvers. It should be noted here that the classical rule can be applied to any black-box function, whereas the generalized trapezoidal rule can be applied to evaluation graphs of PCS functions only.

Now both algorithms have to produce a point on the numerical trajectory \hat{x} corresponding to each time step. We will call the iteration over all time steps the *outer loop*. To

Algorithm 1 Classical Solver

```
1:  $x[0] = x_0$ 
2: for  $i = 0$  to  $N - 1$  do
3:    $\tilde{x} = x[i]$ 
4:    $\hat{x} = \tilde{x} + hf(\tilde{x})$  Euler step
5:   while  $err > tolerance$  AND  $cnt++ < max\_cnt$  do
6:      $\hat{x}_{new} = \tilde{x} + h \frac{f(\tilde{x}) + f(\hat{x})}{2}$  Trapezoidal Rule
7:      $err = \|\hat{x}_{new} - \hat{x}\|$ 
8:      $\hat{x} = \hat{x}_{new}$ 
9:   end while
10:   $x[i + 1] = \hat{x}$ 
11: end for
```

Algorithm 2 ANF-free Solver

```
1:  $x[0] = x_0$ 
2: for  $i = 0$  to  $N - 1$  do
3:    $\tilde{x} = x[i]$ 
4:    $\hat{x} = \tilde{x} + hf(\tilde{x})$  Euler step
5:   while  $err > tolerance$  AND  $cnt++ < max\_cnt$  do
6:      $y = PROPAGATE\_ELEMOP(F, \hat{x}, \tilde{x})$ 
7:      $\hat{x}_{new} = \tilde{x} + h \cdot INTEGRATE(y)$  Gen. Trapezoidal Rule
8:      $err = \|\hat{x}_{new} - \hat{x}\|$ 
9:      $\hat{x} = \hat{x}_{new}$ 
10:  end while
11:   $x[i + 1] = \hat{x}$ 
12: end for
```

Algorithm 3 Secant ANF Solver

```
1:  $x[0] = x_0$ 
2: for  $i = 0$  to  $N - 1$  do
3:    $\tilde{x} = x[i]$ 
4:    $\hat{x} = \tilde{x} + hf(\tilde{x})$  Euler step
5:   while  $\text{err} > \text{tolerance}$  AND  $\text{cnt}++ < \text{max\_cnt}$  do
6:      $y = \text{PROPAGATE\_ELEMOP}(F, \hat{x}, \tilde{x})$ 
7:      $\text{integ} = \text{INTEGRATE}(y)$ 
8:      $\check{F} = F(\tilde{x})$ 
9:      $\hat{F} = F(\hat{x})$ 
10:    function RES_FUNC( $x$ )
11:      return  $x - \tilde{x} - \frac{h}{2}(F(x) - \check{F}) - (2 \cdot \text{integ} - (\check{F} + \hat{F}))$ 
12:    end function
13:    ANFinstance = CREATEANF(res_func,  $\tilde{x}, \hat{x}$ )
14:     $\hat{x}_{new} = \text{SOLVE}(\text{ANFinstance}, \hat{x})$ 
15:     $\text{err} = \|\hat{x}_{new} - \hat{x}\|$ 
16:     $\hat{x} = \hat{x}_{new}$ 
17:  end while
18:   $x[i + 1] = \hat{x}$ 
19: end for
```

Algorithm 4 Tangent ANF Solver

```
1:  $x[0] = x_0$ 
2: for  $i = 0$  to  $N - 1$  do
3:    $\tilde{x} = x[i]$ 
4:    $\hat{x} = \tilde{x} + hf(\tilde{x})$  Euler step
5:   while  $\text{err} > \text{tolerance}$  AND  $\text{cnt}++ < \text{max\_cnt}$  do
6:      $y = \text{PROPAGATE\_ELEMOP}(F, \hat{x}, \tilde{x})$ 
7:      $\text{integ} = \text{INTEGRATE}(y)$ 
8:      $\check{F} = F(\tilde{x})$ 
9:      $\hat{F} = F(\hat{x})$ 
10:    function RES_FUNC( $x$ )
11:      return  $x - \tilde{x} - \frac{h}{2}(F(x) - \check{F}) - (2 \cdot \text{integ} - (\check{F} + \hat{F}))$ 
12:    end function
13:    ANFinstance = CREATEANF(res_func,  $\hat{x}$ )
14:     $\hat{x}_{new} = \text{SOLVE}(\text{ANFinstance}, \hat{x})$ 
15:     $\text{err} = \|\hat{x}_{new} - \hat{x}\|$ 
16:     $\hat{x} = \hat{x}_{new}$ 
17:  end while
18:   $x[i + 1] = \hat{x}$ 
19: end for
```

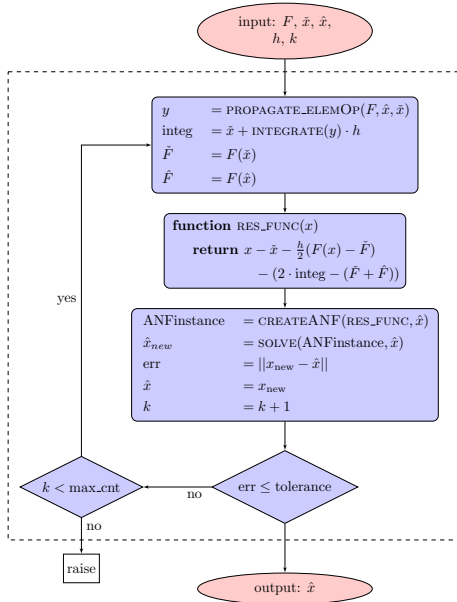


Figure 4: Inner loop with fixed-point solver

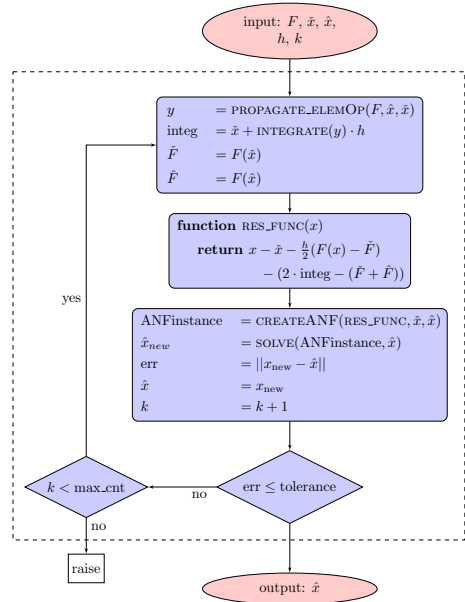


Figure 5: Inner loop with fixed-point solver

calculate \hat{x} , we have to solve an implicit problem and we do this by utilizing a fixed-point iteration scheme. An explicit Euler step predicts the next location of \hat{x} which is then corrected with one or several iterations of the generalized trapezoidal rule. We call this iteration the *inner loop*.

4.1 Calculation of Critical Multipliers and Integration using the ANF

If we have already created an ANF we can use it for both function evaluations of the piecewise linear secant model of F , as well as the calculation of critical multipliers.

As the core task for that purpose we determine, for a given line from x in direction δx , the largest segment $[-\frac{1}{2}, \tau]$ so that $t \mapsto \hat{F} + \Delta F(\check{x}, \hat{x}; x - \hat{x} + t \delta x)$ is linear on that segment.

On the segment $[-\frac{1}{2}, \tau]$ the internal functions z_i of the ANF are all linear. Hence, they can be expressed as

$$z_i(t) = \check{z}_i + t \delta z_i \quad \text{for } 0 \leq t \leq \tau.$$

The term δz is the directional derivative of $z(t)$ at $t = -1/2$ and can be derived by

$$\delta z_j = Z_j \delta x + L_j \sigma \delta z \quad \text{for } 1 \leq j \leq s,$$

where $\sigma_i = \mathbf{sign}(z_i)$ if $z_i \neq 0$ and else $\sigma_i = \mathbf{sign}(\delta z_i)$. During computation the second division-free formula from [8, Section 6] will be used. Linearity remains preserved as long as none of the functions $z_i + t \delta z_i$ change sign. Thus τ has to be chosen as the smallest positive element among the fractions $-\frac{\check{z}_i}{\delta z_i}$, $i = 1, \dots, s$.

To determine the critical multipliers, apply this procedure to $x = \tilde{x}$ and $\delta x = \hat{x} - \tilde{x}$ to get $\tau_1 = \tau_0 + \tau = \tau - 1/2$. Next increment x to $x + \tau_1 \delta x$ and repeat the procedure to determine $\tau_2 = \tau_1 + \tau$ by calculating a new δz and finding the smallest τ with the same formula. This is repeated until the upper boundary $\tau_{i-1} + \tau \geq 1/2$ is reached or surpassed in step i . Then set $\mu = i$ and $\tau_\mu = 1/2$.

If there are no nested absolute values, i.e. if no absolute value depends on another one, or $L = 0$, the list of fractions $-\frac{1}{2} - \frac{z_i}{\delta z_i}$ already contains all possible candidates for critical multipliers in the first step. It only remains to filter this list for values inside $(-1/2, 1/2)$.

In the worst case, where each z_i depends on $|z_{i-1}|$, that is, the absolute values are maximally nested, this leads to a computational cost that is exponential in the number of the absolute values occurring in the computational graph. Every $z_i(t)$ as evaluated on the line $x + t \delta x$ is a piecewise linear function that can consist of up to 2^{i-1} linear segments. Each segment can contain a root so that in constructing $|z_i(t)|$ the number of linear segments is doubled.

However, for practical problems this is almost never the case, since in most practical examples there are no nested absolute values and even if there are, one very rarely encounters more than one absolute value per \tilde{x} to \hat{x} interval. Thus the worst case computational cost will rarely occur, the average number of passes through the ANF to determine the critical multipliers is close to 1. Consequently, the computational cost of one iteration of the inner loop will be a small multiple of a function evaluation.

We can now calculate critical multipliers and integrate in parallel. We calculate the first critical multiplier, use it to get the first summand of (9), update the evaluation point and repeat until $i = \mu$. This procedure is shown in Algorithm 5.

4.2 ANF-free Calculation of Critical Multipliers and Integration

Consider the evaluation graph of a PCS system function of an ODE: $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$, as well as a reference point $\hat{x} \in \mathbb{R}^n$ and let $\mathcal{F} : \mathbb{R}^{n+s} \rightarrow \mathbb{R}^{s+n}$ be the smooth vertical concatenation of $G : \mathbb{R}^{n+s} \rightarrow \mathbb{R}^s$ and $\tilde{F} : \mathbb{R}^{n+s} \rightarrow \mathbb{R}^n$ from the nonlinear ANF representation (5) of F . From [6] follows that the costs of calculating a single column of the Jacobian-matrix by techniques of AD of \mathcal{F} is bounded by 2 up to 3 times the costs of a single function evaluation of the concatenation \mathcal{F} . Thus the costs of computing the ANF of the piecewise linearization of F is bounded by and often close to $3(n+s)$ times the costs of a single function evaluation of F . So, using the full ANF to compute a step of the generalized trapezoidal rule is quite costly, even if no kink occurs within one step from \tilde{x} to \hat{x} . Because then $\mu = 1$ and the ordered list of critical multipliers only consists of its bounds: $-1/2 = \tau_0 < \tau_1 = 1/2$. Consequently, equation (9) simplifies to the classic trapezoidal rule $Q_F = (\hat{F} + \check{F})/2$ because $\Delta F(\tilde{x}, \hat{x}; (\hat{x} - \tilde{x})/2) = -\Delta F(\tilde{x}, \hat{x}; -(\hat{x} - \tilde{x})/2) = (\hat{F} - \check{F})/2$. The latter follows from the secant condition $\hat{F} = \check{F} + \Delta F(\hat{x}, \tilde{x}; (\hat{x} - \tilde{x})/2)$, where $\check{F} = (\hat{F} + \tilde{F})/2$.

We will now outline an efficient, ANF-free implementation. To that end we define a class called ELEMOP which represents a piecewise linear function over the segment $[-1/2, 1/2]$ that is propagated through the elementary operations making up the function F via operator overloading, similar to other direct-forward methods of AD. It consists of an ordered array

Algorithm 5 integration using ANF

```
1: function INTEGRATE
2:    $\delta x = \hat{x} - \check{x}$ 
3:    $x = \check{x}$ 
4:    $z = \text{CALC\_Z}(x)$ 
5:    $y_{\text{old}} = \text{EVALANF}(x)$ 
6:   result = 0
7:    $\tau = -1/2$ 
8:   repeat integrate from kink to kink
9:      $\tau_{\text{new}} = 1$  marks the kink to be found
10:    for  $i = 0, s - 1$  do find the next kink
11:      if  $z_i \neq 0$  then
12:         $\sigma_i = \text{sign}(z_i)$ 
13:      else
14:         $\sigma_i = \text{sign}(\delta z_i)$ 
15:      end if
16:       $\delta z_i = \sum_{j=1}^n Z_{ij} \cdot \delta x_j + \sum_{j=1}^{i-1} L_{ij} \cdot \sigma_j \delta z_j$ 
17:      if  $-\frac{z_i}{\delta z_i} \in [0, \tau_{\text{new}}]$  then
18:         $\tau_{\text{new}} = -\frac{z_i}{\delta z_i}$ 
19:      end if
20:    end for
21:    if  $\tau + \tau_{\text{new}} > 0.5$  then check if kink is relevant
22:       $\tau_{\text{new}} = 0.5 - \tau$ 
23:       $\tau = 0.5$ 
24:    else
25:       $\tau = \tau + \tau_{\text{new}}$ 
26:    end if
27:     $x = x + \tau_{\text{new}} \cdot \delta x$ 
28:     $z = \text{CALC\_Z}(x)$ 
29:     $y = \text{EVALANF}(x)$ 
30:    result = result +  $\frac{\tau_{\text{new}}}{2}(y_{\text{old}} + y)$ 
31:     $\delta y_{\text{old}} = \delta y$ 
32:  until  $\tau \geq \frac{1}{2}$ 
33:  return  $h \cdot \text{result}$ 
34: end function
```

of length $\mu + 1$ containing all critical multipliers $-1/2 = \tau_0 < \tau_1 < \dots < \tau_\mu = 1/2$ as defined in section 3 as well as two arrays $\check{u} = (\check{u}_0, \dots, \check{u}_{\mu-1})$ and $\hat{u} = (\hat{u}_0, \dots, \hat{u}_{\mu-1})$ of length μ . The k^{th} entries \check{u}_k and \hat{u}_k represent values at $-1/2$ and $1/2$ of the active linear function on the k^{th} segment $[\tau_k, \tau_{k+1}]$, that is, the represented function is

$$u(t) = u_k(t) = \left(\frac{1}{2} - t\right) \check{u}_k + \left(\frac{1}{2} + t\right) \hat{u}_k \text{ for } t \in [\tau_k, \tau_{k+1}].$$

Rules for the forward propagation of ELEMOP objects w.r.t. to the secant piecewise linearization can be found in the appendix of this paper. Algorithm 6 is a function that allows to integrate ELEMOP objects over their compact domain $[-1/2, 1/2] \subseteq \mathbb{R}$ as it is called from Algorithm 2 or in Fig. 2.

To integrate a function given by this data structure, according to formula (9), we compute its trapezoidal sum as

$$Q_u = \sum_{i=0}^{\mu-1} \left(\frac{\hat{u}_i + \check{u}_i}{2} + \frac{\tau_{i+1} + \tau_i}{2} (\hat{u}_i - \check{u}_i) \right) (\tau_{i+1} - \tau_i). \quad (14)$$

Algorithm 6 integration/quadrature: $Q_u = \int_{-1/2}^{1/2} \dot{u} + \Delta u(\hat{x}, \check{x}; t \cdot \Delta x) dt$

Require: $\hat{u}, \check{u}, \tau_u$

Ensure: Q_u

- 1: $\mu_u = \text{LEN}(\tau_u)$
 - 2: $Q_u = 0$
 - 3: **for** $i_u = 0, 1, 2, \dots, \mu_u - 1$ **do**
 - 4: $\dot{u} = (\check{u}[i_u] + \hat{u}[i_u])/2$
 - 5: $\Delta u = \hat{u}[i_u] - \check{u}[i_u]$
 - 6: $Q_u += (\tau_u[i_u + 1] - \tau_u[i_u]) \cdot (\dot{u} + \Delta u \cdot (\tau[i_u + 1] + \tau[i_u])/2)$
 - 7: **end for**
-

4.3 Solving a Semi-Explicit Index-1 DAE

In addition to normal ODE systems we can use the generalized trapezoidal rule to solve DAE systems in semi-explicit form of differentiation index 1. These are systems of the form

$$x_1'(t) = F(x_1, x_2), \quad (15)$$

$$0 = G(x_1, x_2), \quad (16)$$

where $F: \mathbb{R}^{n+m+1} \rightarrow \mathbb{R}^n$ and $G: \mathbb{R}^{n+m+1} \rightarrow \mathbb{R}^m$ and the partial derivative $\partial_{x_2} G(x_1, x_2)$ is regular. In contrast to an explicit DAE where the differential and algebraic variables x_1 and x_2 are separated, this system cannot be integrated directly. Assuming that the state $(\check{x}_1, \check{x}_2)$ is consistent (within the error bounds) the next state (\hat{x}_1, \hat{x}_2) is computed as solution of

the system

$$\hat{x}_1 = \check{x}_1 + hQ_F((\check{x}_1, \check{x}_2), (\hat{x}_1, \hat{x}_2)), \quad (17)$$

$$0 = G(\hat{x}_1, \hat{x}_2), \quad (18)$$

To solve this with the Newton iteration in secant mode, in step m the piecewise linear secant approximations $\diamond F$ and $\diamond G$ are computed for the basis points $(\check{x}_1, \check{x}_2)$ and $(\hat{x}_{1,m}, \hat{x}_{2,m})$. Now integrate $\diamond F$ over the segment between the basis points and set $b = Q_F((\check{x}_1, \check{x}_2), (\hat{x}_1, \hat{x}_2)) - \frac{1}{2}\diamond F(\hat{x}_{1,m}, \hat{x}_{2,m})$. Finally the new points $\hat{x}_{1,m+1}$ and $\hat{x}_{2,m+1}$ are computed as the solution of the piecewise linear system

$$\hat{x}_{1,m+1} - \frac{h}{2}\diamond F(\hat{x}_{1,m+1}, \hat{x}_{2,m+1}) = \check{x}_1 + hb, \quad (19)$$

$$\diamond G(\hat{x}_{1,m+1}, \hat{x}_{2,m+1}) = 0. \quad (20)$$

The necessary changes to the solver are illustrated in pseudo-code in Algorithm 7.

Algorithm 7 Generalized Solver for semi-explicit DAE

```

1:  $x^1[0] = x_0^1$ 
2:  $x^2[0] = x_0^2$ 
3: for  $i = 0$  to  $N - 1$  do
4:    $\check{x}^1 = x^1[i]$ 
5:    $\check{x}^2 = x^2[i]$ 
6:    $(\hat{x}^1, \hat{x}^2) = \text{CALC\_EULER}(F, G, \check{x}^1, \check{x}^2)$  Euler step in  $F$ , solve  $G$  for  $\hat{x}^2$ 
7:   while  $\text{err} > \text{tolerance}$  AND  $\text{cnt}++ < \text{max\_cnt}$  do
8:      $y = \text{PROPAGATE\_ELEMOP}(F, (\hat{x}^1, \hat{x}^2), (\check{x}^1, \check{x}^2))$ 
9:      $\text{integ} = \text{INTEGRATE}(y)$ 
10:     $\check{F} = F(\check{x}^1, \check{x}^2)$  both are
11:     $\hat{F} = F(\hat{x}^1, \hat{x}^2)$  components of  $y$ 
12:    function  $\text{RES\_FUNC}(x^1, x^2)$ 
13:      return  $(x^1 - \check{x}^1 - \frac{h}{2}(F(x^1, x^2) - 2 \cdot \text{integ} - \hat{F}), G(x^1, x^2))$ 
14:    end function
15:     $\text{ANFinstance} = \text{CREATEANF}(\text{res\_func}, (\check{x}^1, \check{x}^2), (\hat{x}^1, \hat{x}^2))$ 
16:     $(\hat{x}_{new}^1, \hat{x}_{new}^2) = \text{SOLVE}(\text{ANFinstance}, (\hat{x}^1, \hat{x}^2))$ 
17:     $\text{err} = \|\hat{x}_{new}^1, \hat{x}_{new}^2 - \hat{x}^1, \hat{x}^2\|$ 
18:     $(\hat{x}^1, \hat{x}^2) = (\hat{x}_{new}^1, \hat{x}_{new}^2)$ 
19:  end while
20:   $x^1[i + 1] = \hat{x}^1$ 
21:   $x^2[i + 1] = \hat{x}^2$ 
22: end for

```

5 Numerical Results

In this section we will, for three concrete examples, compare the computational cost of the generalized methods to that of the classical trapezoidal rule. As a measure of computational cost we will use the number of function evaluations accumulated by each method. For the sake of simplicity, we use a constant step size for each of the methods. In order to obtain comparable results, extrapolation has to be applicable to each method. Thus we match the generalized methods against a classical trapezoidal rule with event detection. Note however, that some methods are not fully comparable since the ANF-free fixed point approach and the Newton-type iterations using the ANF have different fields of application. This can be seen in the numerical experiments performed below.

The ANF-free method will be compared to a classical trapezoidal rule for all examples. The first example has the feature that a wide range of step sizes leads to a solution when applying the Newton-type iterations, which is why the latter's costs were investigated for different values of h . In the cardiovascular example the range of feasible step sizes is much narrower, so that we omitted the test for different step sizes and instead performed an additional experiment with a classical trapezoidal rule utilizing a classical Newtons method with a full Jacobian in the inner loop. The shallow water example presents yet another situation due to the large number of events in the solution paths. Here the cost of evaluating the event function dominates the overall cost of the classical methods, which leads to a significant advantage of the ANF-free fixed point approach. The Newton-type approaches using the ANF turn out to be virtually infeasible due to the cost of computing the full ANF.

There are several parts of each of the algorithms that require function evaluations. The first one is the computation of the initial Euler guess which is identical for each method. Another part is the integration step. For the trapezoidal rule it always requires two function evaluations. The generalized rules use a single propagation of one ELEMOP object. The cost of such a propagation can be assessed as follows: The piecewise linearization has two development points \tilde{x} and \hat{x} . Every elementary function in the computational graph has to be evaluated once for each of these points. The total amount of machine instructions is thus virtually identical to that of two function evaluations. The methods using the ANF require function evaluations for the derivation of the ANF on top of that. The computational cost is similar to that of computing a Jacobian matrix using AD. We need a single forward evaluation per row of the ANF which can be bounded by roughly two function evaluations [6]. The number of rows in the ANF equals the dimension n of the RHS F plus the number of absolute values s . Thus the tangent ANF needs $2 \cdot (n + s)$ function evaluations while the secant ANF needs $2 \cdot 2 \cdot (n + s)$, because it has two development points.

For the event detection it is necessary to approximate the event location, i.e., the root of the event function. This can be accomplished by using any type of bisection or secant method. In our implementation, a Brent's method is employed. The evaluation cost for the event function is usually different from the RHS evaluation cost. To have a sharp estimate of the difference one would need to count evaluations of elementary functions for RHS

and event function and convert them into the corresponding number of memory accesses for a modern CPU. For the examples discussed below, the evaluation cost of the event function ranges from very cheap for Example 5.1 to comparable to an evaluation of the RHS function in Example 5.3. In neither of the examples event function evaluations are scale-tipping in the that a precise estimate of their cost would change the overall result. Thus they are not converted but documented in parallel to the RHS evaluations. Since the components of the event function have to be tracked separately, we chose to take an average over all components to give an approximation of the full event function evaluations. Note further, that, in addition to the evaluation of the event function, event handling needs additional integrations to evaluate at approximated event locations. This usually dominates the computational cost of event handling. For context on event detection, cf. [3, 14].

When solving a semi-explicit DAE system, the new point \hat{x} has to be derived as the root of a nonlinear equation system after each integration. This is accomplished in the usual way by employing a suitable type of root-finding algorithm. Below, the root algorithm of the SciPy package is used.

5.1 LC Circuit with Diode

As a first sample application we reprise a problem previously investigated in [7], which resembles problems arising in actual applications. We take a simple LC-circuit and replace the resistor with a diode, providing an element which causes a nondifferentiable impact on the equations describing the system. Figure 6 depicts the circuit.

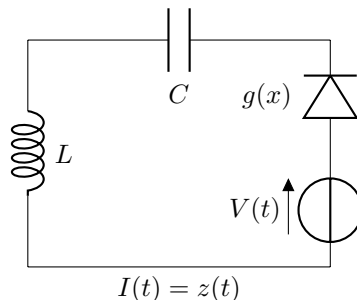


Figure 6: Circuit Diagram

It is modeled by the following system of ODEs, where x_1 represents the time, x_2 represents the charge (at the capacitor) and x_3 represents the electric current in the circuit:

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{pmatrix} = F(x) = \begin{pmatrix} 1 \\ x_3 \\ -(x_2 - C \cdot V(x_1) + g(Cx_3)) \frac{1}{LC} \end{pmatrix}. \quad (21)$$

Here $V(x_1) := \sin(\omega x_1)$ is the forcing current and $g(z)$ models the diode (for small currents)

in the piecewise linear form

$$g(z) = \frac{z + |z|}{2\alpha} + \frac{z - |z|}{2\beta} = \begin{cases} \frac{z}{\alpha} & \text{if } z \geq 0 \\ \frac{z}{\beta} & \text{if } z < 0 \end{cases}.$$

We consider a set of constants similar to those occurring in actual electric circuits:

$$L = 10^{-6}, \quad C = 10^{-13}, \quad \omega = 3 \cdot 10^9, \quad \alpha = 2, \quad \beta = 0.00001.$$

Moreover, we set the initial values to $x_1(0) = x_2(0) = x_3(0) = 0$. The result of the numerical

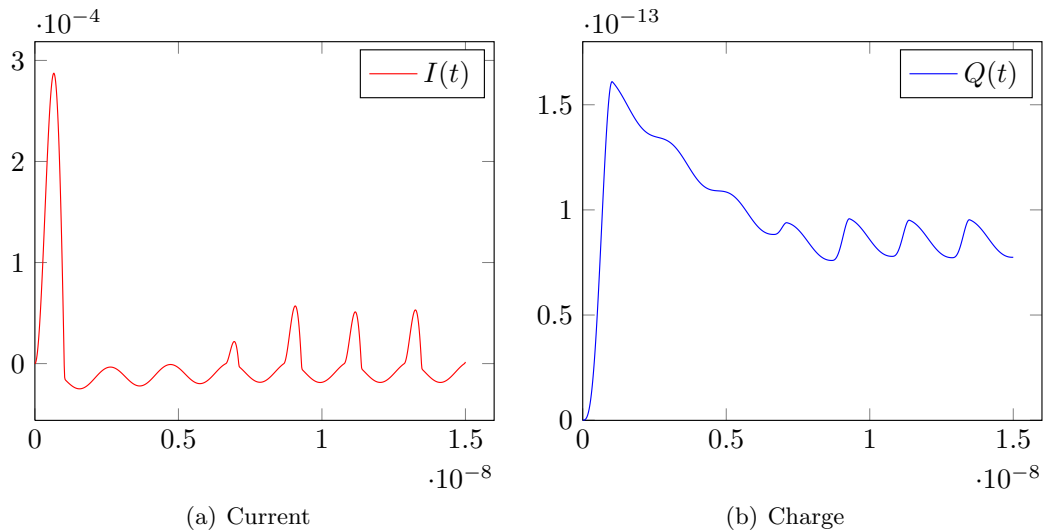


Figure 7: Solution of the ODE System

integration solution of (21) is depicted in Figure 7, (a) and (b). As one can see, the capacitor is initially charged over one cycle and discharged over a few more, before the solution adopts a periodic behavior. The solution trajectory changes its behavior every time the current changes its sign.

Comparison Results

For our comparison we chose the time interval $I = [t_0, T]$, where $t_0 = 0$, and $T = 2.5 \cdot 10^{-8}$, and an initial value $x_0 = (0, 0, 0)^\top$. We calculated 10,000 steps, resulting in a (uniform) step size of $h = 2.5 \cdot 10^{-12}$. Additionally, the two methods using the ANF were also computed with 500 steps. For this step size the ANF-free algorithm does not converge. Figure 8 depicts the 19 events occurring in the given time interval I .

Table 9 displays the number of function evaluations for each of the methods. We denote by EULER the number of function evaluations during the Euler guess, and by INTEG and EVENT the function evaluations due to integration and event detection, respectively. For

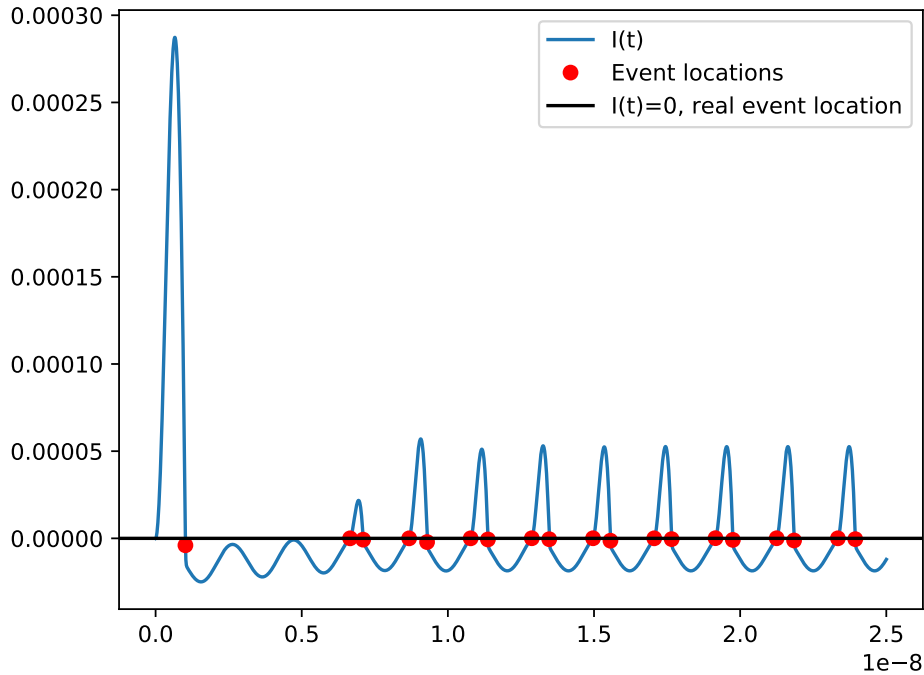


Figure 8: Numerical Solution with Event Locations

this example there is only a single absolute value and thus evaluating the event function is significantly cheaper than evaluating the RHS. The function evaluations for deriving the ANF are denoted by ANF.

In Table 9 it can be seen that the classical method is slightly more expensive than the ANF-free method, but – especially given the cheap event function – they are almost identical. This is due to the fact that this example is a stiff system and has an almost negligible amount of kinks and thus the impact of the nondifferentiabilities is minimal. If the same step size is employed, the Newton-type iterations using the ANF are 2-3 times as expensive both the ANF-free and the classical method. However, these methods already converge for much larger step sizes. For 500 steps only the Newton-type iterations succeed in producing a solution. At this setting they are several times cheaper than the classical and the ANF-free method. Unfortunately though, it is very hard to tell for which choice of step sizes ANF-free and ANF-utilizing methods produce a result of similar accuracy.

5.2 Human Cardiovascular System

The example in this section is based on the lumped-parameter model of the *human cardiovascular system* from [11]. Further informations complying with the model given here can

	Classical 10 ⁵ steps	ANF-free 10 ⁵ steps	Secant 10 ⁵ steps	Tangent 10 ⁵ steps	Secant 500 steps	Tangent 500 steps
EULER	10,020	10,000	10,000	10,000	500	500
INTEG	109,064	108,828	40,096	40,134	2,394	6,436
EVENT	20,063	0	0	0	0	0
ANF	0	0	320,768	160,536	19,152	25,744
TOTAL +EVENT	119,084 + 20,063	118,828	370,864	210,670	22,046	32,680

Figure 9: Function Evaluations of Classical and Generalized Trapezoidal Rule

be found in [4]. The latter is a system of differential-algebraic equations (DAE).

The system consists of 14 compartments. There are 2 for left resp. right atrium (la and ra), as well as for the left resp. right ventricle (lv and rv). These are connected to the systemic and pulmonary cycles. Each of these cycles is composed of 5 compartments where each compartment contains one of the following categories of blood vessels:

Arterial system: First section (sa1, pa1, resp.), second section (sa2, pa2, resp.), and third section (sa3, pa3, resp.).

Vein system: First section (sv2, pv2, resp.), and second section (sv1, pv1, resp.).

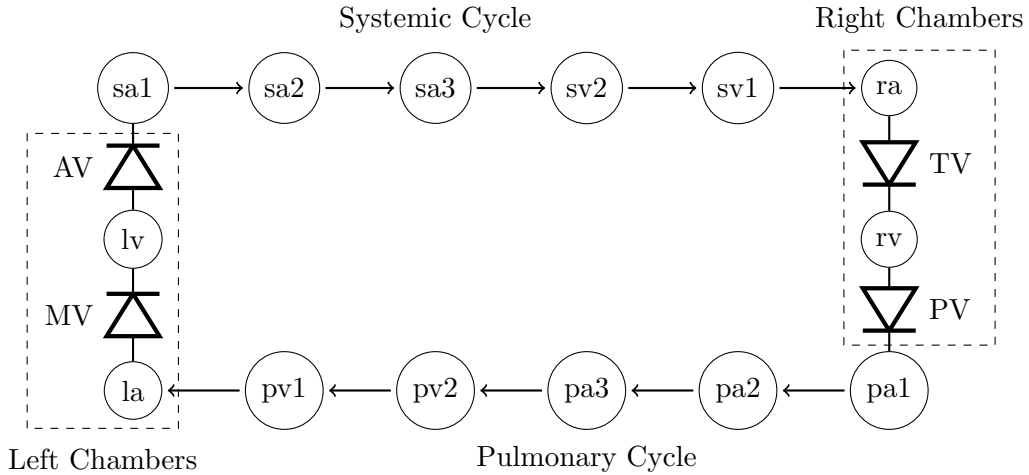


Figure 10: Schematic of the Compartments

The diodes in Figure 10 indicate the locations of the heart valves (MV - Mitral valve, AV - Aortic valve, TV - Tricuspid valve and PV - Pulmonary valve). These ensure a nearly uni-directional flow of blood with almost no friction. In forming the model equations we will deviate from the reference and employ a piecewise differentiable mechanism. The model equations for the respective compartments are thus:

$$\text{Large Vessels } k \in \left\{ \begin{array}{l} \text{sa1, sv1,} \\ \text{pa1, pv1} \end{array} \right\}$$

$$\text{Smaller Vessels } k \in \left\{ \begin{array}{l} \text{sa2, sa3, sv2,} \\ \text{pa2, pa3, pv2} \end{array} \right\}$$

$$L \cdot \dot{q}_k = p_k - p_{\text{succ}} - R_k \cdot q_k$$

$$R_k \cdot q_k = p_k - p_{\text{succ}}$$

$$\dot{v}_k = q_{\text{prec}} - q_k$$

$$\dot{v}_k = q_{\text{prec}} - q_k$$

$$C_k \cdot p_k = v_k - V_k$$

$$C_k \cdot p_k = v_k - V_k$$

The subscript in p_{succ} refers to the variable (pressure) of the next compartment, q_{prec} to the corresponding variable (flow) of the previous compartment in the sense of Figure 10. Further, L_k, R_k, C_k, V_k are compartment-specific (positive) constants. The heart chambers are modeled as:

$$\text{Atria } k \in \{\text{la, ra}\}$$

$$\text{Ventricles } k \in \{\text{lv, rv}\}$$

$$\dot{q}_k = \frac{\max(p_k - p_{\text{succ}}, (R_k - 60)q_k) - R_k q_k}{L_k}$$

$$\dot{q}_k = \frac{\max(p_k - p_{\text{succ}}, (R_k - 60)q_k) - R_k q_k}{L_k}$$

$$\dot{v}_k = q_{\text{prec}} - q_k$$

$$\dot{v}_k = q_{\text{prec}} - q_k$$

$$p_k = E_k \cdot (v_k - V_k)$$

$$p_k = E_k(t) \cdot (v_k - V_k)$$

$$E_k(t) = E_{\min, k}(1 - \phi(t)) + E_{\max, k}\phi(t)$$

$$\text{where } \phi(t) = \max(0, \alpha \sin(\bar{t}) - \beta \sin(2\bar{t})) \text{ and } \bar{t} = \pi \frac{t}{\kappa_0 + \kappa_1 t_h},$$

and $E_{\min, k}, E_{\max, k}$, as well as $\alpha, \beta, t_h, \kappa_i$ are positive constants.

Now let $\mathcal{C} = \{\text{la, lv, sa1, sa2, sa3, sv2, sv1, ra, rv, pa1, pa2, pa3, pv2, pv1}\}$ be the set of all 14 compartments and summarize the set of intrinsic variables into one vector $x = (\mathbf{v}, \mathbf{q}, \mathbf{p})$, where $\mathbf{v} = (v_k)_{k \in \mathcal{C}}$, $\mathbf{q} = (q_k)_{k \in \mathcal{C}}$ and $\mathbf{p} = (p_k)_{k \in \mathcal{C}}$.

Comparison Results

For our comparison we chose the time interval $I = [t_0, T]$, where $t_0 = 0$ and $T = 0.8$, which is the timespan corresponding to a single heartbeat. As initial values we used the ones given in Table 12.

For the secant ANF-based and tangent ANF-based iteration, as well as the one employing a classical Newtons method, $h = 10^{-2}$ was chosen.

For the classical and the ANF-free algorithm the considered step size was $h = 10^{-3}$. The reason for that is simply that the latter two methods do not converge for $h = 10^{-2}$. Figure 11 depicts the numerical solution. In Table 13 the accumulated function evaluations for each of the methods are documented. Again, INTEG documents the number of evaluations of F during the integration, ANF denotes the function evaluations for the ANF. For the classical method with Newton ANF represents the function evaluations for the Jacobian matrix. Note, that there are no Euler evaluations because it turns out that explicit Euler is

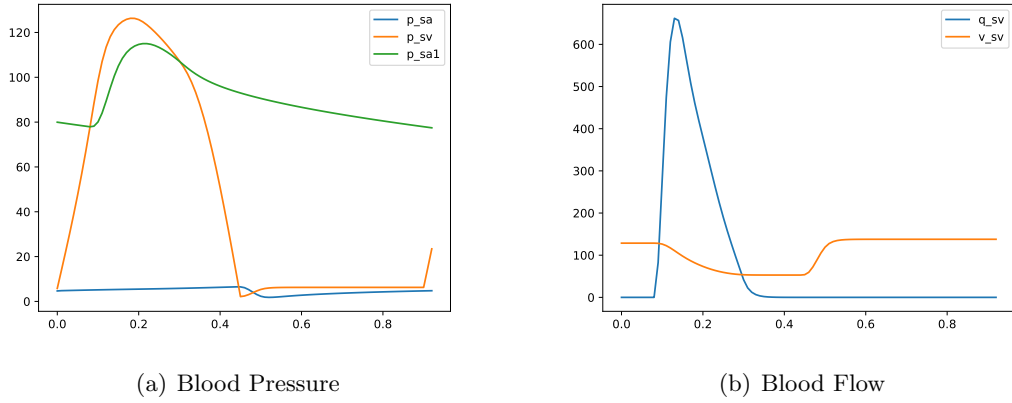


Figure 11: Solution of the Cardiovascular System

	Pressure	Flow	Volume
Arterial 0	4.67	0.00	93.28
Arterial 1	79.96	20.46	267.13
Arterial 2	78.28	61.08	498.38
Arterial 3	67.41	90.10	523.01
Vein 0	5.81	0.00	128.29
Vein 1	5.19	94.90	2321.30
Vein 2	7.31	95.14	692.78

(a) Systemic

	Pressure	Flow	Volume
Arterial 0	2.67	0.00	74.42
Arterial 1	13.31	32.07	79.58
Arterial 2	12.59	51.45	48.64
Arterial 3	9.86	165.24	70.53
Vein 0	3.10	0.00	137.50
Vein 1	5.52	61.86	102.58
Vein 2	3.67	73.53	102.05

(b) Pulmonary

Figure 12: Initial Values of the Cardiovascular System

	Classical	ANF-free	Secant	Tangent	Classic Newton
INTEG	31,350	30,664	688	570	0
ROOT	347,793	340,147	0	0	0
ANF	0	0	67,424	27,930	25,872
EVENT	1910	0	0	0	202
TOTAL	379,143	370,811	68,112	28,500	25,872
+EVENT	+ 1910				+ 202

Figure 13: Function Evaluations of Classical and Generalized Trapezoidal Rule

an unsuitable predictor for this particular problem. The question about a suitable predictor requires further investigation. In contrast to the diode example, we also have to count the function evaluations by the root-finding algorithm applied after the integration. They are denoted by ROOT.

The evaluations of the event function are tracked separately for each component. EVENT gives an average between these values. During the computation, 7 events occurred. For this example the evaluation of the event function is cheaper than the evaluation of the RHS.

As in Example 5.1, the step size for the classical and ANF-free method has to be chosen smaller. Additionally, the root-finding algorithm is expensive and dominates the overall cost. The event function is of very little impact in comparison. Thus, classical and ANF-free method are again similar in terms of computational cost. The algorithms using the ANF are much cheaper, roughly one order of magnitude for the considered step sizes. But as in Example 5.1, this estimate is not reliable. The tangent ANF method is roughly as expensive as the secant ANF based method. As an additional experiment we solve the system employing a classical trapezoidal rule with classical Newton. This method is asymptotically equivalent to the tangent ANF based method with a small constant offset in favor of the classical method. This is due to the fact that for the given example with few absolute values, i.e. n significantly bigger than s , the cost for the computation of a Jacobian matrix (via AD) is, up to a small constant factor, the same as the cost for the computation of an ANF (via generalized AD).

5.3 Shallow Water Equation

The final example is a 2-dimensional shallow water equation. This is a PDE with the following system function:

$$h_t + (hu)_x = 0, \quad (hu)_t + \left[\frac{(hu)^2}{h} + \frac{g}{2}h^2 \right]_x = 0. \quad (22)$$

The two system variables are the fluid depth h and the flow hu . The latter is given by the depth h and the fluid velocity $u := u(hu, h)$. We solve a finite volume discretization of system (22). For this we employ the well-balanced positivity-preserving central-upwind scheme

presented in [10] with the Runge-Kutta solver replaced by the generalized trapezoidal rule. The minmod flux limiter used in [10] is the system's main source of non-differentiability. Note however, that system (22) is simplified in comparison to the one in the latter reference in that we set the bottom elevation to zero. We intend to investigate in future publications whether our piecewise linearization methods can be used to obtain a piecewise linear approximation of the bottom topography that improves upon the formulation developed in [10].

Comparison Results

For this example we chose the time interval as $[t_0, T] = [0, 40]$ with a step size of $h = 0.5$. The space interval was chosen as $[a, b] = [0, 50]$, discretized into 20 cells. The initial values x_0 are chosen as a normal distribution centered at $(b - a)/2$ for h and zero for hu . Figure 14 shows the first component of the numerical solution.

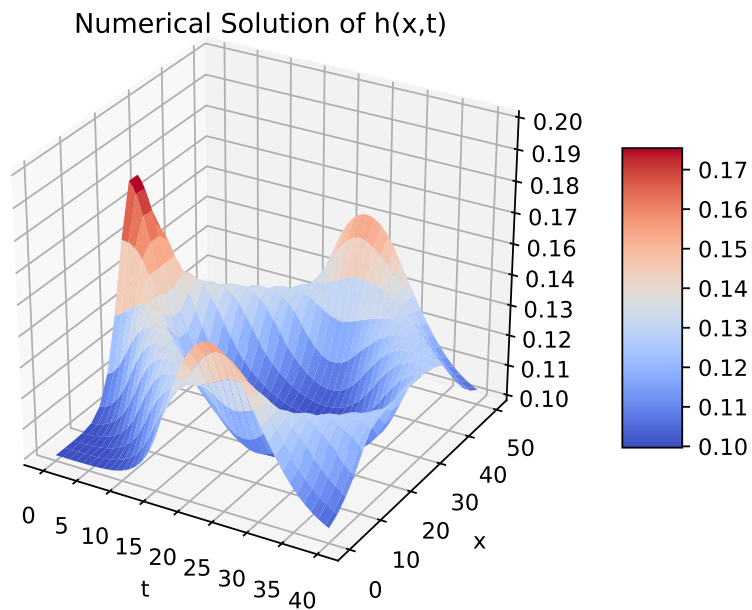


Figure 14: Numerical Solution for h

In Table 15 the number of function evaluations for each of the methods is presented. The same labels as in the previous examples are used. An event occurred in 40 of the time steps. As in Example 5.2, the different components of the event function were tracked separately. In total there were 162,671 event function component evaluations. Table 15

	Classical	ANF-free	Secant	Tangent
EULER	196	79	79	79
INTEG	862	366	336	316
EVENT	298	0	0	0
ANF	0	0	838,656	406,848
TOTAL	1,058	445	839,071	407,173
+EVENT	+ 298			

Figure 15: Function Evaluations of Classical and Generalized Trapezoidal Rule

gives an average over the 546 components. Note that the event function for this example is comparatively expensive. If the cost was converted into multiples of a RHS evaluation, the given average might actually underestimate this number.

The secant ANF-based method requires about twice as many function evaluations as the tangent ANF-based one. However, both methods should not be applied to this example since they are multiple orders of magnitude more expensive than the ANF-free approach. The computational cost is completely dominated by the calculation of the ANF. While there are only 56 variables, the ANF has a dimension of almost 1200 and is thus very expensive to derive. The ANF-free method is significantly cheaper than the classical method for this example. It is at least 2-3 times cheaper. How much exactly, depends on the cost of the event function evaluations. This result is in contrast to Example 5.1 and Example 5.2. The reason is that here about every other step contains a kink and thus the non-smooth elements are dominant while in the other examples the smooth parts of the system are dominant.

6 Conclusions

The merits of generalized and classical trapezoidal rule with event detection strongly depend on the system to be solved. An advantage of the generalized method is noticeable if many absolute values occur in the ODE resp. DAE function and becomes more pronounced the more they are nested. In these cases the event function has a complexity of the same magnitude as the ODE function and the location of the roots of the event function is costly. Testing for events incurs a constant cost per step. In the case that an event is found, multiple additional evaluations of ODE resp. DAE and event function become necessary. The more events are found along the solution path, the more this overhead is pronounced. On the other hand, in this situation the propagation of the piecewise linear segments is faster for the detection of the occurring kinks.

Acknowledgements

The work for the article has been partially conducted within the Research Campus MODAL funded by the German Federal Ministry of Education and Research (BMBF) (fund number

05M14ZAM).

Appendix

In the following we present pseudo-code for methods for the propagation of ELEMOP. The application of smooth elementary operations $v = \phi(u)$ or binary arithmetic operations $v = u \circ v$ translates into the application of a linear transformation to the piecewise linear approximation of the input, as $\Delta v = c_{v,u}\Delta u$ resp. $\Delta v = c_{v,u}\Delta u + c_{v,w}\Delta w$. Since the linear functions over the segments of the sub-division are computed as linear combinations of the data \tilde{u}, \hat{u} etc. in the ELEMOP instance, this linear transformation needs to be applied to all those values. For unary operations this is all, as depicted in Algorithm 8, while in the binary case the subdivision of the result v is a common refinement of the subdivisions of u and w , which requires care in the correct assignment of the relevant linear functions of u and w over the segments of the subdivision of v . See Algorithm 9 for the implementation of the general linear combination common to all binary arithmetic operations. The example code for the multiplication is given in Algorithm 10. Depending on the programming language and its data structures, allocation of arrays of sufficient length for refined sub-divisions can proceed implicitly with variable-length arrays or with a separate counting loop for fixed-length arrays.

Algorithm 8 unary propagation: $v = \varphi(u)$

Require: $\hat{u}, \tilde{u}, \tau_u$

Ensure: $\hat{v}, \tilde{v}, \tau_v$

```

1:  $\tau_v = \text{SET\_REFERENCE\_ON}(\tau_u)$ 
2:  $\mu_{uv} = \text{LEN}(\tau_v)$ 
3:  $\hat{v} = \text{array\_of\_zeros}(\text{len} = \mu_{uv} - 1)$ 
4:  $\tilde{\varphi} = \varphi(\tilde{u}[0])$ 
5:  $\hat{\varphi} = \varphi(\hat{u}[\mu_{uv} - 2])$ 
6: if  $\hat{u} \neq \tilde{u}$  then
7:    $\alpha = \frac{\hat{\varphi} - \tilde{\varphi}}{\hat{u} - \tilde{u}}$ 
8: else
9:    $\alpha = \frac{d}{du} \varphi(\hat{u})$ 
10: end if
11:  $\gamma = \frac{\hat{\varphi} + \tilde{\varphi}}{2} - \alpha \frac{\hat{u} + \tilde{u}}{2}$ 
12:  $\tilde{v} = \text{array\_of\_zeros}(\text{len} = \mu_v - 1)$ 
13: for  $i_{uv} = 0, 1, \dots$  until  $i_{uv} < \mu_v - 1$  do
14:    $\hat{v}[i_{uv}] = \alpha \hat{u}[i_{uv}] + \gamma$ 
15:    $\tilde{v}[i_{uv}] = \alpha \tilde{u}[i_{uv}] + \gamma$ 
16: end for

```

The only operation that, purposefully, deviates from that scheme is the absolute value as the sole non-differentiable operation. For $v = \text{abs}(u)$ the propagation rule $\Delta v = |\dot{u} +$

Algorithm 9 propagation of linear combination: $v = \alpha \cdot u + \beta \cdot w + \gamma$, for $\alpha, \beta, \gamma \in \mathbb{R}$

Require: $\hat{u}, \check{u}, \tau_u, \hat{w}, \check{w}, \tau_w, \alpha, \beta, \gamma$

Ensure: $\hat{v}, \check{v}, \tau_v$

```
1:  $i_u = 0, i_w = 0$ 
2:  $\tau_v = \text{MERGE\_ORDERED\_TAU\_LISTS}(\tau_u, \tau_w)$ 
3:  $\mu_v = \text{LEN}(\tau_v)$ 
4:  $\hat{v} = \text{array\_of\_zeros}(\text{len} = \mu_v - 1)$ 
5:  $\check{v} = \text{array\_of\_zeros}(\text{len} = \mu_v - 1)$ 
6: for  $i_v = 0, 1, \dots$  until  $i_v < \mu_v - 1$  do
7:    $\hat{v}[i_v] = \alpha \cdot \hat{u}[i_u] + \beta \cdot \hat{w}[i_w] + \gamma$ 
8:    $\check{v}[i_v] = \alpha \cdot \check{u}[i_u] + \beta \cdot \check{w}[i_w] + \gamma$ 
9:   if  $\tau_w[i_w + 1] < \tau_u[i_u + 1]$  then
10:      $i_w += 1$ 
11:   else if  $\tau_u[i_u + 1] < \tau_w[i_w + 1]$  then
12:      $i_u += 1$ 
13:   else
14:      $i_u += 1$ 
15:      $i_w += 1$ 
16:   end if
17: end for
```

Algorithm 10 propagation of multiplication: $v = u \cdot w$

Require: $\hat{u}, \check{u}, \tau_u, \hat{w}, \check{w}, \tau_w$

Ensure: $\hat{v}, \check{v}, \tau_v$

```
1:  $\mu_u = \text{LEN}(\tau_u)$ 
2:  $\mu_w = \text{LEN}(\tau_w)$ 
3:  $\alpha = \frac{\hat{w}[\mu_w - 2] + \check{w}[0]}{2}$ 
4:  $\beta = \frac{\hat{u}[\mu_u - 2] + \check{u}[0]}{2}$ 
5:  $\gamma = -\alpha \cdot \beta$ 
6:  $[\hat{v}, \check{v}, \tau_v] = \text{PROPAGATION\_OF\_LINEAR\_COMBINATION}(\hat{u}, \check{u}, \tau_u, \hat{w}, \check{w}, \tau_w, \alpha, \beta, \gamma)$ 
```

$\Delta u|_{-\hat{v}}$ which for the piecewise linear segment means that positive sub-segments get copied, completely negative sub-segments get the sign inverted and sub-segments with a sign change get split in two. The governing quantities in Algorithm 11 are thus the values $u_k(\tau_k)$ and $u_k(\tau_{k+1})$ at the left and right end of the sub-interval.

Algorithm 11 propagation of absolute value: $v = \text{abs}(u)$

Require: $\hat{u}, \check{u}, \tau_u$

Ensure: $\hat{v}, \check{v}, \tau_v$

```
1:  $\mu_u = \text{LEN}(\tau_u)$ 
2:  $\mu_v = \mu_u$ 
3: for  $i_u = 0, 1, 2, \dots, \mu_u - 1$  do
4:    $u_l = (0.5 - \tau_u[i_u]) \cdot \check{u}[i_u] + (0.5 + \tau_u[i_u]) \cdot \hat{u}[i_u]$ 
5:    $u_r = (0.5 - \tau_u[i_u + 1]) \cdot \check{u}[i_u] + (0.5 + \tau_u[i_u + 1]) \cdot \hat{u}[i_u]$ 
6:   if  $u_l \cdot u_r < 0$  then
7:      $\mu_v += 1$ 
8:   end if
9: end for
10:  $\hat{v} = \text{array\_of\_zeros}(\text{len} = \mu_v - 1)$ 
11:  $\check{v} = \text{array\_of\_zeros}(\text{len} = \mu_v - 1)$ 
12:  $\tau_v = \text{array\_of\_zeros}(\text{len} = \mu_v)$ 
13:  $i_v = 0$ 
14:  $\tau_v[0] = 0$ 
15: for  $i_u = 0, 1, 2, \dots, \mu_u - 1$  do
16:    $s = \text{sign}(\check{u}[i_u])$ 
17:    $\hat{v}[i_v] = s \cdot \hat{u}[i_u]$ 
18:    $\check{v}[i_v] = s \cdot \check{u}[i_u]$ 
19:   if  $\check{u}[i_u] \cdot \hat{u}[i_u] < 0$  then
20:      $\tau_{root} = -0.5 \cdot (\hat{u}[i_u] + \check{u}[i_u]) / (\hat{u}[i_u] - \check{u}[i_u])$ 
21:     if  $\tau_u[i_u] < \tau_{root} < \tau_u[i_u + 1]$  then
22:        $i_v += 1$ 
23:        $\tau_v[i_v] = \tau_{root}$ 
24:        $\hat{v}[i_v] = -s \cdot \hat{u}[i_u]$ 
25:        $\check{v}[i_v] = -s \cdot \check{u}[i_u]$ 
26:     end if
27:   end if
28:    $i_v += 1$ 
29:    $\tau_v[i_v] = \tau_u[i_u]$ 
30: end for
```

References

- [1] K. Atkinson, *An Introduction to Numerical Analysis (2nd ed.)*, John Wiley & Sons, New York, 1989.
- [2] F. Clarke, *Optimization and Nonsmooth Analysis*, Canadian Mathematical Society Series of Monographs and Advanced Texts, Wiley-Interscience, 1983.
- [3] W. Enright, K. Jackson, S. Nørsett, and P. Thomsen, *Effective solution of discontinuous ivps using a runge-kutta formula pair with interpolants*, Applied Mathematics and Computation 27 (1988), pp. 313–335.
- [4] L. Formaggia, A. Quarteroni, and A. Veneziani, *Cardiovascular Mathematics - Modeling and simulation of the circulatory system*, MS&A, Vol. 1, Springer-Verlag, Milan, 2009.
- [5] A. Griewank, *On stable piecewise linearization and generalized algorithmic differentiation*, Optimization Methods and Software 28 (2013), pp. 1139–1178, Available at <http://dx.doi.org/10.1080/10556788.2013.796683>.
- [6] A. Griewank and A. Walther, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, Other Titles in Applied Mathematics, Society for Industrial and Applied Mathematics (SIAM), 2008, Available at <http://epubs.siam.org/doi/book/10.1137/1.9780898717761>.
- [7] A. Griewank, R. Hasenfelder, M. Radons, and T. Streubel, *Integrating lipschitzian dynamical systems using piecewise algorithmic differentiation*, Submitted 2016 .
- [8] A. Griewank, T. Streubel, R. Hasenfelder, and M. Radons, *Piecewise linear secant approximation via algorithmic piecewise differentiation*, Submitted 2016 .
- [9] A. Griewank, J. Bernt, M. Radons, and T. Streubel, *Solving piecewise linear systems in abs-normal form*, Linear Algebra and its Applications 471 (2015), pp. 500 – 530, Available at <http://www.sciencedirect.com/science/article/pii/S0024379514008209>.
- [10] A. Kurganov and G. Petrova, *A second-order well-balanced positivity preserving central-upwind scheme for the saint-venant system*, Commun. Math. Sci. 5 (2007), pp. 133–160, Available at <https://projecteuclid.org:443/euclid.cms/1175797625>.
- [11] J. Ottesen, M. Olufsen, and J. Larsen, *Applied Mathematical Models in Human Physiology*, Monographs on Mathematical Modeling and Computation, Society for Industrial and Applied Mathematics, 2004, Available at <http://epubs.siam.org/doi/book/10.1137/1.9780898718287>.
- [12] D. Quispel G.R.W.; McLaren, *A new class of energy-preserving numerical integration methods*, Journal of Physics A: Mathematical and Theoretical 41 (2008).

- [13] S. Scholtes, *Introduction to Piecewise Differentiable Equations*, SpringerBriefs in optimization, Springer New York, 2012, Available at <http://link.springer.com/book/10.1007/978-1-4614-4340-7>.
- [14] L. Shampine and S. Thomson, *Event location for ordinary differential equations*, Computers & Mathematics with Applications 39 (2000), pp. 43–54.