Discontinuities handled with events in Assimulo, a practical approach

Emil Fredriksson* Christian Andersson*,** Johan Åkesson*,***
*Modelon AB, Sweden
Lund University, Sweden
**Department of Numerical Analysis
***Department of Automatic Control

Abstract

Often integrating ordinary differential equations or differential algebraic equations (DAE) do not constitute the problem alone. A common complement is finding the root of an algebraic function (an event function) that depends on the states of the problem. This formulation of a model enables the possibility of including discontinuities, an important part of the Functional Mock-up Interface standard which allows hybrid models of differential algebraic equations. The problem of root-finding during integration is however difficult. Both in a theoretical aspect and as a software problem.

An implementation of software for root-finding is done in Assimulo, a Python/Cython wrapper for integrators. The implementation takes the Functional Mock-up Interface standard into consideration. The implementation is made usable for a wide variety of integration algorithms and is also verified and benchmarked with advanced industrial models, showing good indications of being robust and scaling well for large systems.

Keywords:FMI; JModelica.org; Assimulo; events; discontinuities; Illinois algorithm; safeguard

1 Introduction

Models based on differential equations may contain discontinuities. One simple example is the bouncing ball. Gravity acting on the ball is modelled with a differential equation while the bouncing on the floor will result in discontinuities in the velocity. A result from the velocity changing sign by impact. A reasonable way to model this would be to restart the integration of the differential equation with new initial values as the ball hits the floor. In this way, the discontinuity is modelled with what is called an event and the handling of that event (event handling).

Models with discontinuities are not only interesting theoretically but are also widely used in industry, something the Functional Mock-up Interface (FMI) standard [1] contributes to by making distribution and use of these models convenient. The explanation for why it is used by the industry can be found in [3], where the elements that give rise to discontinuities in models are listed. Some of them are:

- Friction
- Impact phenomena
- The degrees of freedom vary in time
- Time dependent input functions

Most advanced models in industry consist of many separate but interacting parts and, therefore, have at least one or some of the listed properties. With today's modelling tools and computational power, more and more advanced models become realistic to simulate. Which means new and increased demands on the integrators to support the solving of models with discontinuities robustly and with good scaling of the performance regarding the size of the models.

There are many difficulties with having discontinuities in differential equations. Missing a discontinuity or acting on the wrong discontinuity can be disastrous,

The authors gratefully acknowledge the support from the Lund Center for Control of Complex Engineering Systems (LCCC).

[1] See https://www.fmi-standard.org/.

leading to integrating the wrong equations or missing impacts. Furthermore the integration methods make assumptions on the smoothness of the solution and the incorrect handling of the discontinuities will most certainly violate these assumptions. The result will be an incorrect error estimate, leading to a significant decrease in the integration performance[6], or even leading to an incorrect solution.

To construct a state of the art event detection algorithm, undertaking these considerations of the need for correct event handling and performance, the demands will be that it should be robust and scale well, handling large systems originating from industry. It should also be clear what data is expected from the user, and in return the algorithm should guarantee correct event detection and event handling.

The contributions of this article is an implementation of an event algorithm with robust event handling and good performance. Using a safeguard and applying the domain formulation (used for event localization in the FMI specification, explained in Section 2.4) as opposed to the zero-crossing paradigm that uses a sign change to detect events. This algorithm converge and gains a robust performance and has an advantage for a special set of problems, this will be seen for the clutch example later. Given the additions to the event algorithm, a benchmark using advanced industry relevant models following the FMI standard was made to ensure that the performance is not compromised. In Assimulo[2] the algorithm can be utilized as a module that can be mounted onto solvers as needed. This result in an extension of Sundials and increases the number of solvers that can handle discontinuities and therefore the number of solvers that support the FMI standard.

Section 2 starts by giving background and motivation and moves on to highlighting the principles of event detection and event localization and the difficulties associated with it. In Section 3, the ideas for the event location algorithm are laid out. Leading up to the presentation of the algorithm in Section 4, where details of the implementation are discussed. Section 5 demonstrates and verifies the implementation on a number of test examples together with testing the performance. A summary and critical examination of the algorithm and the results are given in Section 6.

[2] See http://www.jmodelica.org/assimulo.

2 Background

Assimulo is a Cython/Python wrapper around various Ordinary Differential Equations (ODE) and Differential Algebraic Equations (DAE) solvers. An important aspect of Assimulo is to make it easy to access both state of the art solvers and more experimental solvers for both industry and the academic world. Assimulo is also the back-end simulation engine for JModelica.org.

On the other hand, for enabling the exchange of models in industry and the academic world there is the FMI standard, which is a standardized way of formulating models of ODEs. PyFMI can be used to wrap models that are instances of the FMI standard (FMUs) making them easy to simulate with Assimulo.

A powerful use of Assimulo and the FMI would be to give industry a larger variety of solvers from the academic world, while the academic world is given access to a large number of relevant models from industry, offering remedies for two weaknesses often present in the world of numerics.

The FMI allows advanced hybrid dynamic models by also allowing event functions. This standard therefore demands that the solver can handle discontinuities in the form of events. Currently, the only solvers in Assimulo that can do so are those of Sundials [12] and LSODAR and many solvers do not have the possibility of handling discontinuities on their own, leading to the need of a module in Assimulo that can handle the discontinuities for all solvers.

2.1 A motivating example

A motivational example for when the zero-crossing approach fail to detect the event correctly is:

```
model motivating_example
Real y;
Real x(start = 1.0);

equation
  y = noEvent(if 1-time > 0 then (1-time)^5
              else 0);
  der(x) = if y <= 0 then -x else x;
end motivating_example;
```

The model has a variable y that smoothly goes to zero at $t = 1$ which should result in an event there. With the usual approach for detecting events, using an FMU from JModelica.org, the event is found significantly later than it occurred because the event is not

DOI 10.3384/ECP14096827

localized. This is especially a problem when the integrator take large steps. It is clear that the robustness of the event detection is questionable in for this case.

With an FMU from Dymola the event is not detected at all.

2.2 The theory of event location

Integration methods are dependent on that the problem has a continuous solution with continuous derivatives to a certain order [3]. Moreover, mathematically, continuity is needed to guarantee a unique solution. For example for the problem of an explicit ODE

$$\dot{y}(t) = f(t, y(t)), \quad y(t_0) = y_0,$$

f should be continuous in t and Lipschitz continuous in y to guarantee a unique solution by the Picard-Lindelöf theorem. This is a big concern with multistep methods, which will incorrectly use information before and after discontinuities if the problem is integrated straight forwardly without explicit event handling.

To avoid integrating over a discontinuity, the event formulation can be used. One way of looking at a problem formulated with events is to imagine that it has two states, each state representing a different continuous right hand side. The discontinuity then becomes switching between these two states (an event). More formally, the point in time of switching is decided by the sign change of an event function, $g(t,y)$, and at this time the integration is re-initialized with updated continuous variables and discrete variables, where the different states are represented by discrete variables that only change at events. For example, the discontinuous problem:

$$\dot{y} = f(t,y) = |y|,$$

is rewritten with the event formulation as:

$$\dot{y} = f(t,y,s) = \begin{cases} y & \text{if } s = s_1 \\ -y & \text{if } s = s_2, \end{cases}$$

$$g(t,y,s) = y,$$

where g and f are continuous for a fixed value of the discrete state s (that represent a state). The downside being that this add a root-finding problem for the event function g on the interval of the latest time steps, t_n and t_{n+1}, in case an event is detected.

What is described in this section is usually called discontinuity handling. It can intuitively be divided into three steps:

- Event detection
- Event localization (in $[t_n, t_{n+1}]$)
- Event handling

The detection is often done by checking the sign of g after every time step. Locating the event is done with a root-finding algorithm and the event handling is mainly a modelling question that is done by the user.

The integrating of ODEs with discontinuities has received a lot of attention over the years. Many of the differences between the approaches is how g will be represented on $[t_n, t_{n+1}]$ and how the time of the event is localized.

Most of the methods for localizing the event require the ability to evaluate $g(t,y)$ on $[t_n, t_{n+1}]$. In doing so effectively, a continuous extension of $y(t)$ on $[t_n, t_{n+1}]$ is desired. Not using a continuous representation when solving problems with discontinuities result in larger global error and more evaluations of f, see [4]. Also, the dependence between the global error and the tolerance was smoother for problems with discontinuities when using an interpolation polynomial for root-finding.

Further theoretical results strengthening the use of interpolation polynomials for problems with discontinuities are that if the interpolation polynomial is of the same order as the integration method the entire method has this order [16].

Besides the representation of y with an interpolation polynomial, there is the idea that additional states could be introduced through new state equations of the form $\dot{y}_{n_y+1:n_y+n_g} = \dot{g}$, where n_y is the dimension of y and n_g is the dimension of g [2] [17]. This will force the integrator to take steps such that the dynamics of g is captured, if this is not the case it is more likely that an event will be missed due to two changes in sign of g are canceled out or that not the first event of many on $[t_n, t_{n+1}]$ is found.

2.2.1 The root-finding problem

The root-finding problem for localizing the event with the event function, g, have the properties that $g \in C^0$ (on a bounded interval $[a,b]$) and $g(a)g(b) < 0$. Through the intermediate value theorem, the existence of a zero in the interval $[a,b]$ is guaranteed. Due to the nature of solving the problem numerically, the zero can often not be found exactly. Therefore the problem is said to be solved if an interval $[a^*, b^*]$ is found, such that:

$$g(a^*)g(b^*) < 0 \text{ and } |a^* - b^*| < \delta.$$

This means that the zero of g is contained in a small interval of length δ. Also, note how the condition $g(a)g(b) < 0$ functions as an enclosing property; this will from here on be known as a regula falsi. The goal is to find an algorithm that always converge and doing so as quickly as possible for a wide range of functions. The goal can be considered delicate because of the large spectrum of functions that are allowed.

2.3 Integrators in Assimulo

Presented here are some of the ODE integrators wrapped by Assimulo that are suited to be used together with an event localization algorithm and therefore use to simulate FMUs. Their types, orders and interpolation are listed.

Explicit and Implicit Euler: Fixed step-size methods of order 1 with linear interpolation implemented.

RungeKutta34: Adaptive Runge-Kutta of order 4(3) with a Third-order Hermitian polynomial for interpolation.

Radau5ODE: Runge-Kutta method based on Radau IIA of order 5, with interpolation from its collocation solution. The interpolation polynomial is of order 3 [9] [10].

Dopri5: Runge-Kutta method, is of order 5(4) with an interpolation of order 4 [9] [10].

RodasODE: A Rosenbrock method of order 4(3). The order of the interpolation is not stated explicitly, but it is said to fulfil conditions such that the continuous solution is of the same order as the discrete points. Uses variable step-size [9] [10].

CVode: Uses BDF methods for stiff problems and Adams-Moulton methods for non-stiff problems. For both cases, the solver is of variable-order and has variable step-size. Contains an internal event localization algorithm [12].

2.4 FMI semantic and domain formulation

The condition that an event occurs when g changes sign (a change between $g < 0$ and $g > 0$, zero-crossing formulation) means that the zero needs to be treated as an exception. An option is that one should instead look for an alternation between the domains $g < 0$ and $g \geq 0$ [14]. This leads to a formulation similar to the event formulation in the FMI standard, where the enclosing and detection properties (regula falsi) change from $g(a)g(b) < 0$ to $(g(a) > 0) \bigoplus (g(b) > 0)$[3].

3 Event algorithm

3.1 Domain or zero-crossing formulation

The arguments for using the domain formulation, in addition to being consistent with the FMI standard, are that the zero is no longer a special case. [11].

The FMI formulation also has a major advantage when modeling systems that can take an unknown input. Let us suppose that $g(t,y) = u(t)$, where u is a signal that can become and stay at zero. A practical example would be if u is the power to a system. Imagine now that the system as a safety measure has a magnet-locking system or clutches that locks when the power disappears. An event is then expected when $u > 0$ goes to $u \leq 0$ or vice versa. This is an intuitive way to state the model and would, with the zero-crossing formulation, force the user to modify g or the inequality with a small ε to ensure zero-crossing. Moreover, the choice of ε is often not an easy task in this case because of scaling.

3.2 Event detection

One of the usual ways to detect an event is to check the regula falsi for $g(t_n, y_n)$,$g(t_{n+1}, y_{n+1})$ after having integrated from t_n to t_{n+1}. Other ways are, of course, possible - such as also checking g in the middle of $[t_n, t_{n+1}]$ - but these are considerably slower. This is the case for many of the more sophisticated methods for detecting events and they furthermore demand access to the partial derivatives of g. This is also the case for the methods of adding extra states. Demanding the user to supply these or compute them numerically, giving the user a solver that scales badly (computing the derivatives numerically would also result in extra evaluations of g) is not an option. It does not align with our demands of speed, it would also exclude models of the FMI standard.

Going with the simpler method of checking for a regula falsi, there is the possibility of having two events in $[t_n, t_{n+1}]$ for a component of g. This is a problem, as pointed out in Section 2.2. The practical solution used here lies in letting the user supply a maximum stepping length, $h_{\max}$, such that all events are separated by at least $h_{\max}$ in time.

[3] The logical symbol $\bigoplus$ is XOR and in code the condition would be $(g(a) > 0) != (g(b) > 0)$

DOI 10.3384/ECP14096827

3.3 Locating the event

A root-finding algorithm that is usually used in this context is the Illinois algorithm [5] [13] [12] [4]. Naturally, it gives fast convergence for most functions when doing event localization. There are of course other root-finding algorithms that would perform well for event localization, Illinois is however well tested. Furthermore, if g is multidimensional and the first root in time should be found for any of the components of g and this vectorization is best done for the Illinois algorithm.

The Illinois algorithm uses a linear interpolation that weights the function values to ensure convergence. The weight is applied so that the algorithm will not keep any bracket constant indefinitely. This gives major advantages when it comes to convergence compared to the false position algorithm. This is especially evident for all convex functions, a comparison can be seen in Figure 1.

An improvement can be done for when the function is zero for most of the interval as Illinois algorithm behaves badly in this case. This comes from the Illinois algorithm's inability to use its weights properly - zero times two is still zero. The intuitive remedy for this would be to use a bisection step if either $g(t_n)$ or $g(t_{n+1})$ are equal to zero. With this modification, the Illinois algorithm behaves better for this case.

3.3.1 Safeguard

To ensure that the algorithm converges a safeguard method is used [7]. It consist of three points that, if followed, guarantee that the root-finding algorithm always converge. These are:

- The new bracket is inside the current interval
- The new bracket is closest to the best bracket [5]
- The new bracket is not too close to an existing bracket

Point one is incorporated in the Illinois algorithm. Point two is not always valid during the iterations, due to the weights. Point two is however only meant to ensure the speed of convergence, something that the weights do very well.

[4] The origin of the method is unknown - some believe it to come from the staff at the computer centre at the University of Illinois computer department.

[5] The best bracket is the one with the lowest function value, as it is expected to lie closer to the root.

Point three is often done by choosing a small number δ and check if the new bracket is closer than δ to a previous bracket, if it is the new bracket is moved by δ towards the middle. It should be noted that Illinois algorithm dose not originally account for this. There is, however, no technical problem in implementing this (Sundials has this modification in its implementation).

4 Implementation

4.1 User interaction

The form of the event function g defined by the user should be a function returning an array of all the components of g. This is also how the event function is evaluated in the FMI standard, but results in that all components are evaluated, even if only one component has a change in sign when iterating with the Illinois algorithm. However, this gives the possibility of detecting roots that otherwise would have been missed due to being close to another root.

Also, the reporting of the found roots should be done with a separate array. The user cannot know for which component a root was detected in the case where the exact zero is not found due to scaling.

The tolerance for enclosing the event cannot be set and has a default value such that the error caused by it is small compared to that caused by the interpolation polynomial.

4.2 Algorithm details

The tolerance for which the domain change is locked into is somewhat weaker than machine precision. Specifically, the tolerance is set to:

$$\text{TOL} = max(|t_n|, |t_{n+1}|) \cdot 10^{-13},$$

being roughly a factor 100 times larger than machine precision, it still does not introduce an error that is noticeable compared to that of the interpolation polynomial.

The small step δ in the safeguard (see Section 3.1.1) is taken from Sundials as:

$$\delta = \frac{|a-b|}{2 \cdot \min(5, |a-b|/TOL)},$$

note that δ is a minimum of a tenth of the current length of the interval and a maximum of half the interval, going from a tenth to a half as the interval goes toward TOL.

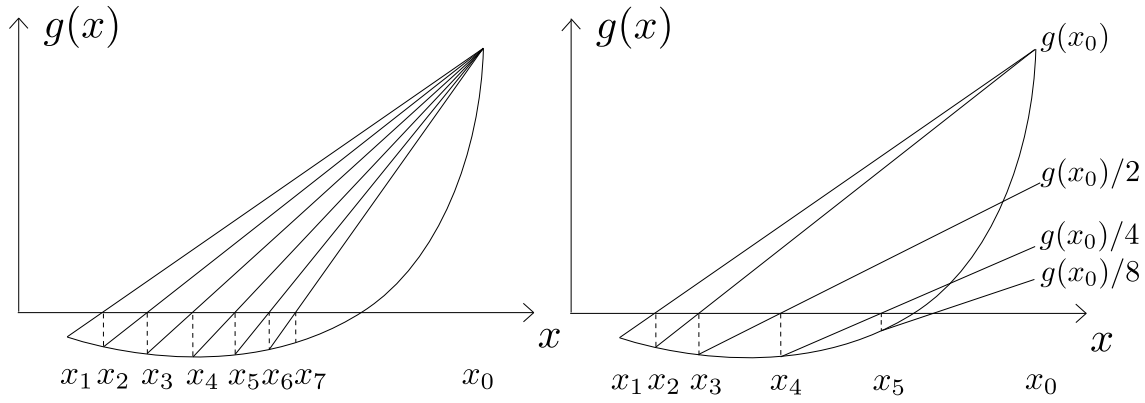

Figure 1: Comparison between the Illinois algorithm and the false position method, showing how the failing mode is escaped and fast convergence is achieved.

The domain formulation and the bisection step for the special case mentioned earlier are used by the algorithm. The algorithm is implemented in Assimulo using Python. It is called before the complete step function is called just after a successful step is taken.

5 Application example

For evaluating the implementation, benchmark models are selected to assert the correctness and scalability of the event location algorithm. A Furuta pendulum is chosen for having many event functions, a clutch model with inputs is chosen for causing events when the input reach zero for a finite time and a racing car is chosen for being a large advanced model with events.

The models are FMUs and PyFMI is used for simulating these models in Assimulo. Benchmarks are made for each model and each ODE solver that the event location is implemented for. The quantities included in the benchmark are: steps taken, function evaluations, Jacobian evaluations, event-function evaluations, number of events and simulation time in seconds.

The options for the solvers are tuned so that the solution at the final step is found with a relative error of roughly 10^{-6} compared to a reference solution, with 10^{-6} here roughly meaning that the relative error is in the interval $[10^{-6}, 2 \cdot 10^{-6}]$, preferably as close to the left bracket as possible. Formally, this means that the solution should satisfy the condition:

$$10^{-6} \leq \frac{||y(t_{\text{final}}) - y_{\text{ref}}(t_{\text{final}})||_2}{||y_{\text{ref}}(t_{\text{final}})||_2} \leq 2 \cdot 10^{-6}.$$

This error is foremost tuned by changing the relative- and absolute-tolerance. In case the integrator takes too large steps and events therefore are missed, a maximum step length is also used to tune the accuracy of the solution. For RungeKutta34, the optional initial step length is also used - RungeKutta34 would otherwise have a problem reducing the tolerance sufficiently, as it has no possibility to reject its first step. The other steps are affected by the tolerance through the variable step-size (even though the step cannot be rejected).

For finding the reference solution, CVode with its internal event location is used. The options for relative and absolute tolerance is set to 10^{-12}. At this tolerance, the relative error estimate does not change if the tolerance is increased or decreased by a factor 10, meaning that significantly more correct decimals are found for this solution compared to the solutions satisfying the condition on relative error.

An important reason for using CVode is that it is extensively tested and well established.

Following the numerical results there is a discussion of them. In the discussion, important differences between the solvers are pointed out, and extra emphasis is placed on the differences between CVode with internal event location and CVode with Assimulos event location, here after known as CVode(I) and CVode(A).

5.1 Furuta pendulum

The Furuta pendulum (see Figure 2), generated by Dymola, is an extremely non-linear model and is often investigated in the field of control theory [18]. This model of the problem generates events from introduced friction, resulting in a problem with 32 event functions. When simulated for 5 seconds, 21 events occur for this problem, making it a good test model for the.

DOI 10.3384/ECP14096827

Solver options	CVode(I)	CVode(A)	Dopri5	Rodas	Radau5	RungeKutta 34
Relative tolerance	$2 \cdot 10^{-8}$	$3.1 \cdot 10^{-8}$	$1.81 \cdot 10^{-6}$	$1.42 \cdot 10^{-6}$	$1.8 \cdot 10^{-7}$	$5 \cdot 10^{-7}$
Absolute tolerance	$2 \cdot 10^{-8}$	$3.1 \cdot 10^{-8}$	$1.81 \cdot 10^{-6}$	$1.42 \cdot 10^{-6}$	$1.8 \cdot 10^{-7}$	$5 \cdot 10^{-7}$
Initial step length	-	-	-	-	-	10^{-6}

Table 1: The solver options to obtain the desired accuracy for the Furuta pendulum. CVode(I) is CVode with its internal event location, while CVode(A) is Cvode with Assimulo's event location.

Run statistics	CVode(I)	CVode(A)	Dopri5	Rodas	Radau5	RungeKutta 34
Steps taken	1107	1050	113	329	231	589
f evaluations	1468	1430	776	2109	1680	2945
J evaluations	34	30	-	329	156	-
g evaluations	1332	1266	372	615	504	787
Execution time	2.760	2.888	0.6183	3.165	0.9516	6.157
Relative error	$1.05 \cdot 10^{-6}$	$1.08 \cdot 10^{-6}$	$1.13 \cdot 10^{-6}$	$1.03 \cdot 10^{-6}$	$1.18 \cdot 10^{-6}$	$1.13 \cdot 10^{-6}$

Table 2: Run-time statistics for the solvers used on the Furuta pendulum. CVode(I) is CVode with its internal event location, while CVode(A) is CVode with Assimulo's event location. All other solvers relies on Assimulos event handling algorithm.

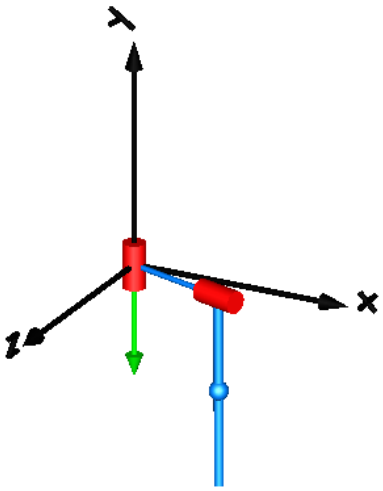

Figure 2: A Furuta pendulum. The red cylinders being joints with given frictions and the blue cylinders being bars with given weights.

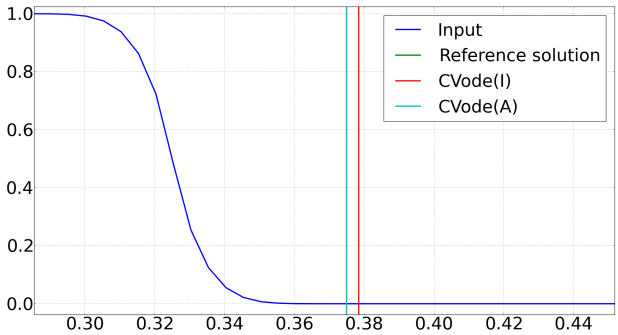

Figure 3: Plot of the input and where the first event is found for the different solvers. The event found by the reference solution and CVode(A) can not be distinguised in the figure.

The first thing to note is that CVode(A) achieves the same error as CVode(I) with higher tolerances, somewhat adding robustness to the solving process. Nonetheless, the execution time is longer despite using fewer function evaluations.

Dopri5 is the solver that performs best.

5.2 Clutches with input

This system is of practical industrial interest, where an input signal causes events when reaching zero for a finite time, very much as described in Section 3.1. The number of events that occur for the 5 seconds that the system is simulated varies, as some of them are handled internally by the FMI's event iteration. However, all the events are detected and handled in one way or another, as indicated by the small relative errors the problem is solved for. The FMU is generated by JModelica.org.

The events caused by the input signals are located with close to machine precision for CVode(A) and only with the current step length when detecting $g(t_{n+1}) = 0$ for CVode(I). Despite this, the same choices for tolerances give the same relative error. This is however still considered an

Solver options	CVode(I)	CVode(A)	Dopri5	Rodas	Radau5	RungeKutta 34
Relative tolerance	$6 \cdot 10^{-7}$	$6 \cdot 10^{-7}$	$1.7 \cdot 10^{-6}$	$7 \cdot 10^{-5}$	$4.39 \cdot 10^{-5}$	$5.033 \cdot 10^{-9}$
Absolute tolerance	$6 \cdot 10^{-7}$	$6 \cdot 10^{-7}$	$1.7 \cdot 10^{-6}$	$7 \cdot 10^{-5}$	$4.39 \cdot 10^{-5}$	$5.033 \cdot 10^{-9}$
Max step length	-	-	-	0.1	0.1	-
Initial step length	-	-	-	-	-	10^{-10}

Table 3: The solver options to obtain the desired accuracy for the Clutch system with the input signal. CVode(I) is CVode with its internal event location, while CVode(A) is CVode with Assimulo's event location.

Run statistics	CVode(I)	CVode(A)	Dopri5	Rodas	Radau5	RungeKutta 34
Steps taken	1594	1594	656	305	233	5312
f evaluations	2568	2552	4498	2370	1275	26545
J evaluations	71	71	-	305	93	-
g evaluations	2480	2733	1876	1739	1746	6223
Execution time	8.476	8.425	17.56	7.709	6.756	55.42
Relative error	$1.11 \cdot 10^{-6}$	$1.07 \cdot 10^{-6}$	$1.16 \cdot 10^{-6}$	$1.08 \cdot 10^{-6}$	$1.24 \cdot 10^{-6}$	$1.09 \cdot 10^{-6}$

Table 4: Run-time statistics for the solvers used on the Clutch system with the input signal. CVode(I) is CVode with its internal event location, while CVode(A) is CVode with Assimulo's event location. All other solvers relies on Assimulos event handling algorithm.

important result, finding the events with good accuracy is desired and is expected to pay off for other models. The first event from the input signal at 0.0375 is found to be 0.375000000000015, 0.378400333934374, 0.375000000000025 for the reference solution, the CVode(I) and the CVode(A) respectively, a significant improvement as can be seen in Figure 3. Radau5 performs best for this model.

5.3 Racing car

A large and advanced model from industry is the racing car. Here, the model not only contains the racing car but also a virtual driver. It consists of a regulator that tries to drive the car on an eight-shaped course. One reason for simulating this might, for example, be to investigate the dynamic response of the car.

Consisting of 47 states, 44 event functions and simulated for 30 seconds, this is the largest model used for testing. During simulation, the model caused 11 events. The model is developed and generated from Dymola originating from the Vehicle Dynamics Library, see Figure 4.

Here, CVode(A) performs better than CVode(I), in the sense that it needs less tolerance for achieving the same error. This is a good sign of robustness and scalability.

For the racing car model, evaluations of f are more

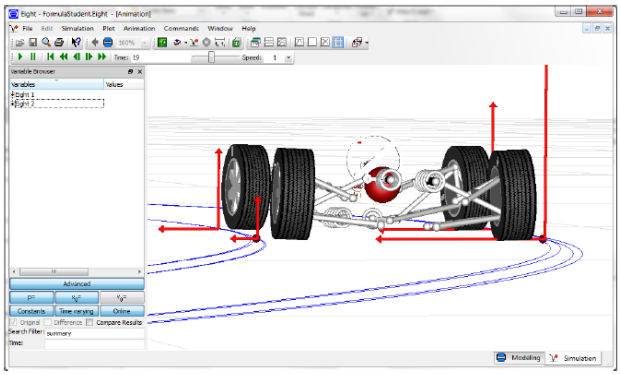

Figure 4: A picture of the car in Dymola.

costly compared to those in the Furuta model. This makes CVode the most effective integrator. Also note that Radau5 need the setting $h_{\max} = 0.1$ for its maximum step length to capture all the events.

5.4 Summary

The event localization of Assimulo, using the domain formulation, finds the events caused by the inputs in the clutch model significantly better than Sundials, using the zero-crossing formulation.

If the event localization of Assimulo is compared by looking at the results for CVode(A) and CVode(I), it is clear that Assimulos event algorithm do not perform significantly worse than that of Sundials. For all

DOI 10.3384/ECP14096827

Solver options	CVode(I)	CVode(A)	Dopri5	Rodas	Radau5	RungeKutta 34
Relative tolerance	$5.5 \cdot 10^{-6}$	10^{-5}	$3 \cdot 10^{-3}$	$8 \cdot 10^{-5}$	$3.5 \cdot 10^{-5}$	$7.3 \cdot 10^{-3}$
Absolute tolerance	$5.5 \cdot 10^{-6}$	10^{-5}	$3 \cdot 10^{-3}$	$8 \cdot 10^{-5}$	$3.5 \cdot 10^{-5}$	$7.3 \cdot 10^{-3}$
Max step length	-	-	-	-	0.1	-
Initial step length	-	-	-	-	-	10^{-9}

Table 5: The solver options to obtain the desired accuracy for the racing car. CVode(I) is CVode with its internal event location, while CVode(A) is CVode with Assimulo's event location.

Run statistics	CVode(I)	CVode(A)	Dopri5	Rodas	Radau5	RungeKutta 34
Steps taken	1248	1287	2163	333	358	2817
f evaluations	1720	1736	13152	2143	2330	14085
J evaluations	38	34	-	333	248	-
g evaluations	1485	1478	2409	539	638	3039
Execution time	14.45	14.59	23.76	21.44	17.78	27.49
Relative error	$1.31 \cdot 10^{-6}$	$1.54 \cdot 10^{-6}$	$1.39 \cdot 10^{-6}$	$1.57 \cdot 10^{-6}$	$1.17 \cdot 10^{-6}$	$1.31 \cdot 10^{-6}$

Table 6: Run-time statistics for the solvers used on the racing car model. CVode(I) is CVode with its internal event location, while CVode(A) is CVode with Assimulo's event location. All other solvers relies on Assimulos event handling algorithm.

models except the racing car CVode(A) uses more g evaluations than CVode(I). Sometimes, this results in better accuracy for the selected tolerances. This is the case for the Furuta pendulum and the racing car model. For the clutch model with an input signal, there is, on the other hand, nothing to be gained and all that is achieved is extra g evaluations.

The new solvers that now support event localization all performed reasonably. For this set of models CVode and also Dopri5 performs well, making it together with all the new solvers supporting event location a welcome addition to the toolbox of integrators supporting event location and therefore also FMUs.

For some of the models, it is noted that Runge-Kutta methods performs very well as indicated by the small number steps taken. This is made possible by the sparse occurrence of events and it is worth noting that none of the models here has events occurring with a high frequency. In [8], this was investigated using small balls bouncing around, receiving different frequencies for the events by changing the number of balls.

6 Summary and conclusions

It was found that an algorithm based on the Illinois algorithm works well. The improvements include applying the domain formulation, extra safeguarding and the special case when $g = 0$ resulting in a bisection step. It was implemented in Assimulo and was shown to locate certain types of events more accurately for an industry-relevant model with clutches than Sundials, without losing much in performance. Much of the performance lost is, of course, because Assimulo's event location is written in Python while Sundial's is written in C. Further improvements can be made reducing this difference by typing the variables using the possibilities of Cython in Assimulo, leading to the code executing more like C code, even though the differences in speed can probably never be fully remedied.

The implementation can be used together with many solvers as it is a separate subroutine, a module that is easily mounted onto solvers. Resulting in the possibility to choose among many solvers when simulating FMUs. Also, solvers with interpolation polynomials of lower order than the method itself still performed well, without any guarantees from the theory.

For the case of a problem containing a large number of event functions, the improvement is being able to choose Runge-Kutta methods, which are good at solving many of the models with sparse occurrence of events. In the case of the extreme opposite, many events occurred with high frequency, CVode with a $h_{\max}$ matching this frequency would probably perform best.

More work for the future might include the possibil-

ity of supplying the partial derivatives of g or of calculating these numerically. Uses might be enhanced event location, as with [17] or [1]. Further work in investigating event location for implicit DAEs is also needed, for example implementing consistent event localization [15] and test the gain in accuracy and the loss in computational speed on relevant industry problems.

References

[1] CARVER, M. Efficient integration over discontinuities in ordinary differential equation simulations. *Mathematics and Computers in Simulation 20* (1978), 190 – 196.

[2] CARVER, M., AND MACEWEN, S. Numerical analysis of a system described by implicity-defined ordinary differential equations containing numerous discontinuities. *Applied Mathematical Modelling 2* (1978), 280 – 286.

[3] EICH-SOELLNER, E., AND FÜHRER, C. *Numerical methods in multibody dynamics*, corr. repr.; 2. corr. repr., 2008 ed. Teubner, [Lund], 2002.

[4] ENRIGHT, W., JACKSON, K., NORSETT, S., AND THOMSEN, P. Interpolants for Runge-Kutta formulas. *ACM Transactions on Mathematical Software 12*, 3 (1986), 193–218.

[5] FORD, J. *Improved Algorithms of Illinois-type for the Numerical Solution of Nonlinear Equations*. University of Essex, Department of Computer Science, 1995.

[6] GEAR, C. W., AND OSTERBY, O. Solving ordinary differential equations with discontinuities. *ACM Transactions on Mathematical Software 10*, 1 (1984), 23.

[7] GILL, P. E., MURRAY, W., AND WRIGHT, M. H. *Practical optimization*. Academic Press, London, 1981.

[8] GRABNER, G., AND KECSKEMETHY, A. An integrated Runge-Kutta root finding method for reliable collision detection in multibody systems. *MULTIBODY SYSTEM DYNAMICS 14*, 3-4 (2005), 301 – 316.

[9] HAIRER, E., NORSETT, S. P., AND WANNER, G. Solving ordinary differential equations i: Nonstiff problems (e. hairer, s. p. norsett, and g. wanner). *SIAM Review 32*, 3 (1990), 485.

[10] HAIRER, E., AND WANNER, G. *Solving Ordinary Differential Equations II [Elektronisk resurs] : Stiff and Differential-Algebraic Problems / by Ernst Hairer, Gerhard Wanner.* Springer Series in Computational Mathematics: 14. Berlin, Heidelberg : Springer Berlin Heidelberg, 2010., 2010.

[11] HIEBERT, K., AND SHAMPINE, L. Implicitly defined output points for solutions of odes. *Sandia National Laboratory Report SAND80-0180* (1980).

[12] HINDMARSH, A., BROWN, P., GRANT, K., LEE, S., SERBAN, R., SHUMAKER, D., AND WOODWARD, C. Sundials: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software 31*, 3 (2005), 363–396.

[13] MOLER, C. Are we there yet? zero crossing and event handling for differential equations. *EE Times Simulink 2 Special Edition* (1997), 16–17.

[14] OLSSON, H. *Runge-Kutta solution of initial value problems : methods, algorithms and implementation / Hans Olsson.* Lund : Univ., 1998, 1998.

[15] PARK, T., AND BARTON, P. I. State event location in differential-algebraic models. *ACM Transactions on Modeling and Computer Simulation 6*, 2 (1996), 137–165.

[16] SHAMPINE, L., AND THOMPSON, S. Event location for ordinary differential equations. *Computers and Mathematics with Applications 39*, 5-6 (2000), 43–54.

[17] SHAMPINE, L. F., GLADWELL, I., AND BRANKIN, R. W. Reliable solution of special event location problems for odes. *ACM Transactions on Mathematical Software 17*, 1 (1991), 11.

[18] XU, Y., IWASE, M., AND FURUTA, K. Time optimal swing-up control of single pendulum. *Journal of Dynamic Systems, Measurement and Control, Transactions of the ASME 123*, 3 (2001), 518–527.

DOI 10.3384/ECP14096827