CHANGE OF INDEPENDENT VARIABLE FOR STATE EVENT DETECTION IN SYSTEM SIMULATION - EVALUATION WITH ARGESIM BENCHMARKS

Felix Breitenecker[(a)], Horst Ecker [(b)], Berhard Heinzl [(c)], Andreas Körner[(d)], Matthias Rößler[(e)], Niki Popper[(f)]

[(a) (d)] Vienna University of Technology, Institute for Analysis and Scientific Computing, Austria
[(c) (d)] Institute for Computer Aided Automation, Vienna University of Technology
[(b)] Vienna University of Technology, Institute for Mechanics and Mechatronics, Austria
[(e)]dwh Simulation Services,Vienna, Austria

[(a)] felix.breitenecker@tuwien.ac.at, [(b)] horst.ecker@tuwien.ac.at, [(c)]bernhard.heinzl@tuwien.ac.at, [(d)]andreas.koerner@tuwien.ac.at, [(e)] matthias.roessler@tuwien.ac.at, [(f)] niki.popper@drahtwarenhandlung.at

1. ABSTRACT

In system simulation state event descriptions and state event handling are used to model and to simulation 'discontinuous' changes of the system under investigation. State events may cause simple 'jumps' of parameters, or complex changes of models, as in case of loss or increase of degrees of freedom. The classical method of state event handling consists of an interpolative and /or iterative method to locate the event with ongoing time – based on the zero of the state event function. This contribution discusses first event classification and time-based event location algorithms, and second, it discusses an alternative method for state event location by change of the independent variable. This method, first suggested by Henon in 1982, makes use of the principles of Poincare mappings. This alternative method is analyzed and sketched with models from the *ARGESIM Benchmarks on Modelling Approaches and Simulation Techniques*.

Keywords: system simulation, state event handling, structural dynamic systems, state event finding, benchmarks

2. STATE EVENT DESCRIPTION IN SYSTEM SIMULATION

Simulation systems for system simulation generate usually an implicit state space description of type

$$\dot{\vec{x}}(t) = \dot{f}(t, \vec{x}(t), \vec{z}(t), \vec{u}(t), \vec{p}), \vec{x}(t_0) = x_0 \qquad (1)$$

$$\vec{g}(\vec{x}(t), \vec{z}(t), \vec{u}(t), \vec{p}) = \vec{0} \qquad (2)$$

For subsequent numerical analysis, an appropriate DAE solver with certain accuracy is to be used.

In models from application it is often necessary to model discontinuities in the model description, because a certain system phenomenon can only be described 'approximatively' by a discontinuous change in the model. These discontinuous changes are called *events*; if the time instant of the change is known in advance, the event is called a *time event*.

If the event depends on a certain values or thresholds for state variables or functions of state variables, the time instant of occurrence is not known in advance, and the event is called a *state event*.

For rough numerical analysis these discontinuous changes may be mimicked *by if – then – else* constructs in the right hand side of the system equations. The DAE solver takes these changes into account at the next solver step after the event. For accurate numerical simulation and in case of more complex discontinuous changes it is necessary to synchronise the event with the DAE solver. For time events, this synchronisation is easy. For state events, a relatively complex algorithm – state event handling – must be performed.

A state event is defined

- by an *event function*

$$h(\vec{x}(t), \vec{z}(t), \vec{u}(t), \vec{p}) \qquad (3)$$

  whose zero determines the time instant $\hat{t}$ of the occurrence of the event,

- and by an *event action*

$$E(\vec{x}(\hat{t}), \vec{z}(\hat{t}), \vec{u}(\hat{t}), \vec{p}) \qquad (4)$$

  which performs the discontinuous change.

An event function $h(\cdots)$ can cause the associated event action $E(\cdots)$ several times.

$$h^A(\vec{x}(t), \vec{u}(t), \vec{p}) \overset{!\pm}{=} 0 \overset{\hat{t}^A}{\Rightarrow} E^A(\vec{x}(\hat{t}^A), \vec{u}(\hat{t}^A), \vec{p}) \qquad (5)$$

The event description (5) is to be read as follows: the event 'A' occurs, if the event function (3) associated with the event 'A', $h^A(\vec{x}(t), \vec{z}(t), \vec{u}(t), \vec{p})$ has a zero crossing, whose zero $\hat{t}^A$ is to be determined; dependent on the crossing direction, the associated event action $E^A(\vec{x}(\hat{t}^A), \vec{u}(\hat{t}^A), \vec{p})$ has to be performed: only in case of crossing from positive in negative direction, only in case of crossing from negative in positive direction, or in both cases, indicated by the crossing operator:

$$\binom{!-}{=}, \binom{!+}{=}, \binom{!\pm}{=}$$

The associated event action $E^A(\vec{x}(\hat{t}), \vec{u}(\hat{t}), \vec{p})$ must handle the discontinuous change in the model description, which ranges from simple to very complex. It makes sense to classify events with respect to their complexity of action of change:

- state event – output change – SE-O
- state event – parameter change – SE-P
- state event – input change – SE-I
- state change – state value change – SE-S
- state event – derivative vector change - SE-D
- state event – model change - SE-M

3. CLASSICAL STATE EVENT HANDLING

The primary tasks of event handling are the synchronisation of the state event with the ODE/DAE solver, and the 'execution' of the event – i.e. the discontinuous change of parameters, inputs, and states, and the choice of new derivatives or new models. The classical state event algorithm requires the following steps:

- *Detection* of the event
- *Localisation* of event and solver stopping
- Event *Action*
- *Restart* of ODE solver

Event Detection. Observing the algebraic sign of the event function during the time advance of the ODE/DAE solver in each integration interval $[t_k, t_{k+1}]$ allows detection of the event:

$$\operatorname{sign} h\big(\vec{x}(t_k)\big) \neq \operatorname{sign} h\big(\vec{x}(t_{k+1})\big).$$

On inequality, the event occurred at a time instant $\hat{t}, \hat{t} \in [t_k, t_{k+1}]$. On occurrence, depending on the defined crossing direction, the state event must be handled, or not: in case of crossings into negative direction ('-' in (14)) if $\operatorname{sign} h\big(\vec{x}(t_k)\big) = +1$, in case of crossings into positive direction ('+' in (5)) if $\operatorname{sign} h\big(\vec{x}(t_k)\big) = -1$, and in case of crossings in both directions ('±' in (5)) in any case.

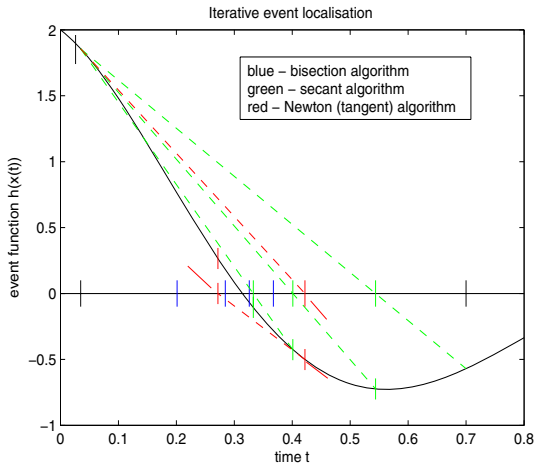

Figure 1: Algorithms for Iterative State Event Location (zero of function $h\big(\vec{x}(t)\big)$ - black line): Bisection – blue; Secant Method – green; Tangent Method or Newton Algorithm resp. – red.

Classical Event Location. After detection, the event time $\hat{t}$, $\hat{t} \in [t_k, t_{k+1}]$ must be determined with sufficient accuracy, followed by a 'closing' solver step from t_k to $\hat{t}$. Two related algorithms are used,

Interpolative algorithms $\hat{t} = \Psi_I(t_k, t_{k+1})$

Iterative algorithms $\hat{t}_m = \Psi_I(\hat{t}_m; t_k, t_{k+1})$

A very classical method is bisection (iterative); common method is the secant formula (interpolative or iterative), and advanced method is the tangent formula (interpolative) or the Newton algorithm resp. (iterative.). The iterative event location algorithms stop if the length of the *event window* is less equal a given minimal length:

$$\Delta\hat{t}_m = |\hat{t}_{m-1} - \hat{t}_m| \leq \varepsilon_{event} \rightarrow \hat{t} = \hat{t}_m$$

Figure 1 shows the first steps of the forementioned methods which are given in detail in the following.

Bisection method. The iterative bisection method is the slowest method, but the most 'stable':

- The interval $[t_k, t_{k+1}]$ is divided into two halves
  $$[t_k, t_{B,1}], [t_{B,1}, t_{k+1}], t_{B,1} = \frac{1}{2}(t_{k+1} - t_k),$$
  checking of the algebraic sign determines, in which interval the event happens,
  e.g. $\hat{t} \in [t_k, t_{B,1}]$
- Again interval $[t_k, t_{B,1}]$ is divided into two halfs
  $$[t_k, t_{B,2}], [t_{B,2}, t_{B,1}], t_{B,2} = \frac{1}{2}(t_{B,1} - t_k)$$
  and the algebraic sign determines the next 'event interval', and so on with further n-2 times dividing of the event interval, until event time $\hat{t}$ is sufficiently approximated due to (23) and finally set to $\hat{t} = t_{B,n}$.
- As last step, integration of (2) and (3) on the 'remaining' interval $[t_k, t_{B,n} = \hat{t}]$ updates the states until event time.

Another iterative method is the *secant formula*, which determines the zeros of the secant line connecting the event function at the boundaries of the event interval:

- With interval $[t_k, t_{k+1}]$ and values of the event function at the interval borders given by $h_k = h\big(x(t_k), u(t_k)\big), h_{k+1} = h(x(t_{k+1}, u(t_{k+1})$ the first approximation for the event time $\hat{t}$ is the zero of the secant which connects (t_k, h_k) and (t_{k+1}, h_{k+1}):
  $$t_{S,1} = t_k - \frac{t_{k+1} - t_k}{h_{k+1} - h_k} h_k$$
- The time instant $t_{S,1}$ divides the first event interval, and the algebraic sign function determines, where the event happens, e.g. $\hat{t} \in [t_k, t_{S,1}]$;
  with event function values $h_k, h_{S,1}$ at the interval borders of $[t_k, t_{S,1}]$, the next approximation for $\hat{t}$ is the zero of the connecting secant line:

$$t_{S,2} = t_k - \frac{t_{S,1} - t_k}{h_{S,1} - h_k} h_k$$

- This procedure is further iterated *m*-2 times, until event time $\hat{t}$ is sufficiently approximated due to (23) and finally set to $\hat{t} = t_{S,m}$.
- As last step, integration of (2) and (3) on the 'remaining' interval $[t_k, t_{S,m} = \hat{t}]$ updates the states until event time.

A very fast iterative method is the *Newton algorithm*, or simply the *tangent method*:

- With interval $[t_k, t_{k+1}]$ and value and derivative value of the event function at the left border given by $h_k = h\big(x(t_k), u(t_k)\big), \dot{h}_k = \frac{d}{dt} h\big(x(t_k), u(t_k)\big)$, the first approximation for event time $\hat{t}$ is the zero of the tangent in (t_k, h_k):

$$t_{N,1} = t_k - \frac{1}{\dot{h}_k} h_k$$

- This procedure is further iterated *r*-2 times, until event the time $\hat{t}$ is sufficiently approximated due to (23) and finally set to $\hat{t} = t_{N,r}$.
- As last step, integration of (2) and (3) on the 'remaining' interval $[t_k, t_{N,r} = \hat{t}]$ updates the states until event time.

In principle, this tangent method (Newton algorithm) approximates the event function by a Taylor series expansion of 1st order. Accuracy can be increased by using a Taylor series approximation of 2nd order, requiring the second derivative of the event function, which is available in special cases (event function is threshold of distance, whereby distance and velocity are states). Figure 2 compares the discussed Taylor approximations for the event function; the tangent method is an interesting link to the Henon method (section 4).

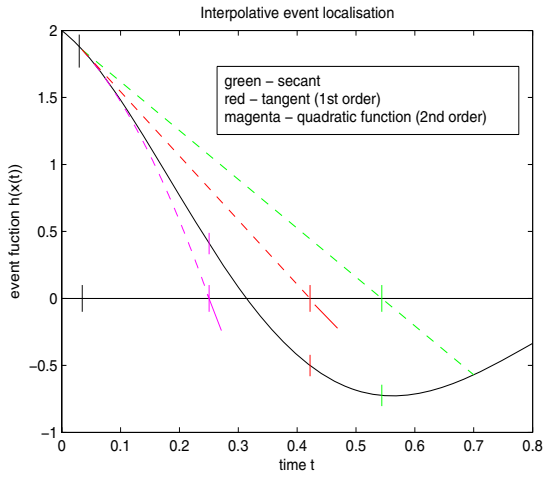

Figure 2: Comparison of interpolative algorithms for state event location (zero of function $h\big(\vec{x}(t)\big)$ - black line): secant method – green; tangent method – red, quadratic interpolation - magenta.

Some aspects with respect to event location must be taken into account:

- Iterative methods can give more accurate results. But in case of different event functions with nearby roots, the iterative methods may cause a deadlock, may let events vanish, etc.
- Iterative methods generally cause backstepping in time, which causes problems with not synchronised time events occurring in the iteration interval, and with stepsize control of the ODE solver.
- A general, partly essential and partly philosophical problem comes up with methods which make use of the 'virtual' value $h_{k+1} = h(\vec{x}(t_{k+1}))$ of the event function at the time step after the event, which is definitely wrong from viewpoint of physics (in case of bouncing ball example the algorithms accepts, that the ball is flying *in* the ground !). This physical error may give totally wrong results for $h_{k+1} = h(\vec{x}(t_{k+1}))$, or it may be only a strange fact.
- ODE solvers' stepsize control and event location algorithm are partly 'competitive' strategies. If the DAE/ODE solver takes a too long stepsize, an event may vanish, in case the step bridges two zero crossings. In principle, the event time instant $\hat{t}$ is to be seen as so-called *barrier time* for the solver, which limits the stepsize.
- The DAE/ODE solver themselves are approximations of a specific order – if for instance an explicit Euler solver has calculated $\vec{x}(t_{k+1})$ and the event function (3) is a linear function of the states, then the tangent method and secant method for determining the event time coincide with the ODE solver step, so that the event time determined by the cited algorithms is exact, and– further iteration makes no sense. In general, the desired accuracy for determining the event time must be coordinated with the solver's order and solver's accuracy.
- Generally, the event function is not really known as formula; as the function depends on the states, which are only known at grid points, also the event function is only known at gridpoints (before and after the event), so that the secant interpolation is the first choice for event location because of genuine accurateness; iteration or more complex methods refine the accuracy of the ODE solvers in behind.

Many advanced DAE/ODE solvers offer the steps *detection* and *localisation* with *solver stopping* as generic feature – so called *root finding* feature. Additionally to system equations (1) and (2) also the event function (3) or (5) resp. with indication of crossing direction can be provided, resulting in a stopping of the ODE/DAE solver at localized event time.

Sometimes tuning parameters for the event location can be chosen, but only in rare cases the location method can be chosen. Simulation systems make either use of a DAE/ODE solver with root finding, or they incorporate the event algorithm into a general event mechanism which controls the ODE solver stopping.

Event Action. Due to the state event classification sketched before, the event actions in (4) consist of discontinuous change of parameter values, of input values, of output values, or of state values, or the actions, switch to new derivative functions (new derivative vector) or to a new model description (with changing dimension). A general view is to interpret a state event as the end of validity of a certain model, and the necessity to restart with a new model, which gets initial values from the previous model. This approach – the ***hybrid decomposition*** approach -decomposes the system into conditionally consecutive dynamic models, which are interrupted and controlled by discrete events.

The approach, strongly related to ***structural-dynamic systems*** is recommended for handling ***model change*** events SE-M and partly ***derivative vector*** events SE-D. The approach might be an overdo for handling state events of type SE-P, SE-I, SE-S and partly for events of type SE-D, because these events can be handled within one model by specific features like event-synchronised conditions or schedulable discrete segment.

While – If-Then-Else Constructs. The action of state events of type SE-P, SE-I and partly of type SE-D can be handled in ***if-then-else*** constructs or ***while*** constructs, which are 'synchronised' with the ODE/DAE solver, i.e. which cause implicitly a state event. Most simulation systems offer features for this 'integrated' approach which requires artificial constructs for structural dynamic changes and state jumps (frozen states, integrator hold, etc.

4. STATE EVENT DETECTION BY CHANGE OF INDEPENDENT VARIABLE

An unusual, but eventually efficient alternative method for state event location can be based on a method for calculation of Poincare maps, suggested by M. Henon. Henon interprets the state space (1) as (autonomous) dynamical system

$$\begin{aligned}\frac{dx_1}{dt} &= f_1(x_1, x_2, \dots, x_n)\\ \frac{dx_2}{dt} &= f_2(x_1, x_2, \dots, x_n)\\ &\dots\dots\dots\dots\\ \frac{dx_n}{dt} &= f_n(x_1, x_2, \dots, x_n)\end{aligned} \qquad (6)$$

which describes curves in the n-dimensional space. A Poincare map is roughly speaking the set of intersections of the curves with (n-1)-dimensional subspace

$$H(x_1, x_2, \dots, x_n) = 0 \qquad (7)$$

representing an autonomous event function of type (3):

$$h(\vec{x}(t), \vec{u}(t), \vec{p}) = H(x_1, x_2, \dots, x_n) = 0$$

For simple subspaces of type

$$h(\vec{x}(t), \vec{p}) = H(x_k) = x_k = a \qquad (8)$$

in system (6) now the independent variable t can be changed with the dependent variable x_k (e.g. $k = n$) giving a modified system with independent variable x_n and dependent variables $x_1, x_2, \dots, x_{n-1}, t$:

$$\begin{aligned}\frac{dx_1}{dx_n} &= \frac{f_1(x_1, x_2, \dots, x_n)}{f_n(x_1, x_2, \dots, x_n)}\\ \frac{dx_2}{dx_n} &= \frac{f_2(x_1, x_2, \dots, x_n)}{f_n(x_1, x_2, \dots, x_n)}\\ &\dots\dots\dots\dots\\ \frac{dx_{n-1}}{dx_n} &= \frac{f_{n-1}(x_1, x_2, \dots, x_n)}{f_n(x_1, x_2, \dots, x_n)}\\ \frac{dt}{dx_n} &= \frac{1}{f_n(x_1, x_2, \dots, x_n)}\end{aligned} \qquad (9)$$

System (9) is usually much more complex, and division by $f_n(x_1, x_2, \dots, x_n)$ might cause severe problems, as well as implicit function theorem must hold, but the systems allows numerical solution of the system with accurate intersection of the subspace $H(x_n) = x_n = a$.

Henon suggest now a clever modification:

- Solution of the original system (6) by an ODE solver until t_m just before the intersection with $H(x_n) = x_n = a$ stopping with $x_n(t_m) \neq 0$.
- Solution of the modified system (9) by the same ODE solver, from $x_n(t_m) = x_{n0}$ with one step of length $\pm x_{n0}$, meeting the intersection almost exactly.

The subspace $H(x_n) = x_n = a$ can be interpreted now as set of zeros of the event function $h(x_1, x_2, \dots, x_n)$ so that the above algorithm can be used as alternative method for state event location without any interpolation and iteration.

For complex subspaces $H(x_1, x_2, \dots, x_n) = 0$ Henon suggests reformulation of the subspace by a differential equation

$$x_{n+1}(t) = H(x_1, x_2, \dots, x_n)$$

$$\frac{dx_{n+1}}{dt} = \frac{d}{dt} H(x_1, x_2, \dots, x_n) =$$

$$\sum_1^n \frac{dH}{dx_i} \cdot f_i(x_1, x_2, \dots, x_n)$$

Henon's method allows now formulating a modified event location algorithm which meets the zero of the event function $h(x_1, x_2, \dots, x_n)$ 'exactly' within the accuracy of the ODE solver used.

Event Detection. Observing the algebraic sign of the event function during the time advance of the ODE/DAE solver in each integration interval $[t_k, t_{k+1}]$ allows detection of the event:

$$\text{sign}\, h\big(\vec{x}(t_k)\big) \neq \text{sign}\, h\big(\vec{x}(t_{k+1})\big).$$

On inequality, the event occurred at a time instant $\hat{t}, \hat{t} \in [t_k, t_{k+1}]$. On occurrence, depending on the defined crossing direction, the state event must be handled.

Event Location by Variable Change. After detection and stopping time integration at t_k with $\vec{x}(t_k)$, the event time $\hat{t}$, $\hat{t} \in [t_k, t_{k+1}]$ is determined by solving the modified extended state space (9) if $f_{n+1}(x_1, x_2, \dots, x_n)$ does not get zero on $(t_k - \varepsilon, \hat{t} + \varepsilon)$:

$$\frac{dx_1}{dx_{n+1}} = \frac{f_1(x_1, x_2, \dots, x_n)}{f_{n+1}(x_1, x_2, \dots, x_n)}$$

$$\frac{dx_2}{dx_{n+1}} = \frac{f_2(x_1, x_2, \dots, x_n)}{f_{n+1}(x_1, x_2, \dots, x_n)}$$

$$\dots\ \dots\ \dots\ \dots$$

$$\frac{dx_n}{dx_{n+1}} = \frac{f_n(x_1, x_2, \dots, x_n)}{f_{n+1}(x_1, x_2, \dots, x_n)}$$

$$\frac{dt}{dx_{n+1}} = \frac{1}{f_{n+1}(x_1, x_2, \dots, x_n)}$$

$$f_{n+1} = \sum_1^n \frac{dH}{dx_i} \cdot f_i(x_1, x_2, \dots, x_n) \qquad (10)$$

using the ODE solver on the interval $[x_{n+1}(t_k), 0]$ with

$$x_{n+1}(t_k) = H\big(\vec{x}(t_k)\big)$$

and stepsize $\pm x_{n+1}(t_k)$ or fractions of $\pm x_{n+1}(t_k)$

resulting in all state values and time instant of the zero of the event function:

$$\hat{t} = t(x_{n+1} = 0)$$

$$\vec{x}(\hat{t}) = \vec{x}(x_{n+1} = 0)$$

In case of simple event functions of type (8) the modified state space simplifies to

$$\frac{dx_1}{dx_n} = \frac{f_1(x_1, x_2, \dots, x_n)}{f_n(x_1, x_2, \dots, x_n)}$$

$$\frac{dx_2}{dx_n} = \frac{f_2(x_1, x_2, \dots, x_n)}{f_n(x_1, x_2, \dots, x_n)}$$

$$\dots\ \dots\ \dots\ \dots$$

$$\frac{dx_{n-1}}{dx_n} = \frac{f_{n-1}(x_1, x_2, \dots, x_n)}{f_n(x_1, x_2, \dots, x_n)}$$

$$\frac{dt}{dx_n} = \frac{1}{f_n(x_1, x_2, \dots, x_n)} \qquad (12)$$

to be solved on the interval $[x_n(t_k), 0]$ with stepsize $\pm x_n(t_k)$ or fractions of $\pm x_{n+1}(t_k)$ resulting in all state values and time instant of the zero of the event function:

$$\hat{t} = t(x_n = 0),$$

$$\vec{x}(\hat{t}) = \vec{x}(x_n = 0)$$

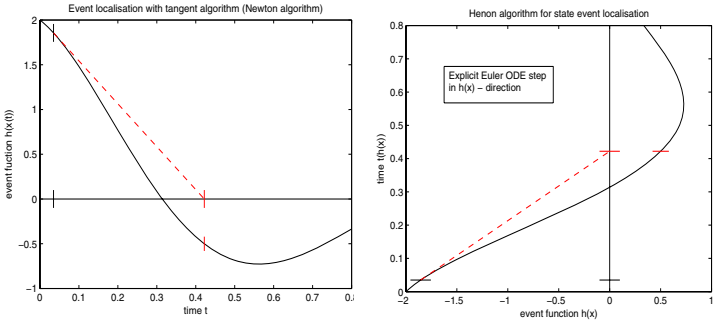

Figure 3: State Event Location with Tangent Method in Time Domain (at left), and State Event Location by Euler ODE Solver in State Domain

The suggested method requires that locally the implicit function theorem holds at least on $(t_k - \varepsilon, \hat{t} + \varepsilon)$, without singularities. Figure 3 shows that indeed uniqueness of inversion can be violated, e.g. as long the function is not monotone. The following evaluation discusses some advantages and disadvantages of this method, and suggests modifications.

- Henon suggest a combination of systems (6) and (9) with a general independent variable and with scaling factors (transformation factors) so that indeed he quotients in equation (12) are used. It might be appropriate to perform symbolical simplifications on these quotients.
- The original method makes one final solver step with the new independent variable – for improving accuracy, or in case of zeros which build up a series of limit points, it might be better to use more than one solver step with fractions of the remaining step, or to continue with the solver in the new independent variable.

Figure 3 shows an interesting link between classical state event location algorithms and Henon's state event location. The tangent interpolation method in the time domain (at left) makes use of the same tangent line as the Euler ODE solver in the state domain (at right):

$$x = \dot{h}_k t + h_k - t_k \dot{h}_k$$

$$t = \frac{1}{\dot{h}_k} x + t_k - \frac{1}{\dot{h}_k} h_k$$

While in first time domain formula the zero must be calculated, in the second state domain formula simply a zero must be inserted: $t = t_k - \frac{1}{\dot{h}_k} h_k$.

5. EVALUATION WITH ARGESIM BENCHMARKS

The journal SNE – *Simulation Notes Europe* is publishing benchmarks for modelling approaches and implementations of approaches in simulation systems. In these benchmarks, state event handling plays an important role.

At present Henon's method is tested with the models from ARGESIM Benchmark C6 *Constrained Pendulum* and ARGESIM Benchmark C20 *Hybrid and Structural Dynamic Systems*.

Benchmark C6 *Constrained Pendulum* relates to the physical pendulum described by angle $\varphi(t)$, given by state space

$$\frac{d\varphi_1}{dt} = \varphi_2$$

$$\frac{d\varphi_2}{dt} = -\frac{g}{l} sin\varphi_1 - \frac{d}{m}\varphi_2 \qquad (13)$$

Task is to model the discontinuous change of pendulum length and angular velocity, if the pendulum hits a pin which is locates at angle φ_p. The event function becomes

$$h(\varphi_1, \varphi_2) = \varphi_1 - \varphi_p$$

This event function is of the simple type (8), and for using the method of independent variable change for state event location, the modified system with independent variable φ_1 is

$$\frac{d\varphi_2}{d\varphi_1} = -\frac{g}{l}\frac{sin\varphi_1}{\varphi_2} - \frac{d}{m}$$

$$\frac{dt}{d\varphi_1} = \frac{1}{\varphi_2} \qquad (14)$$

Henon's method works well, if φ_2 does not get zero when hitting or leaving the pin. Unfortunately the angular velocity $\dot{\varphi} = \varphi_2$ gets zero, if the pendulum is on point of return – which may happen at pin position φ_p – here a modified event function could help (to be investigated further).

Benchmark C20 *Hybrid and Structural Dynamic Systems* test more or less complex state event changes in models from mechanical and electrical engineering: bouncing ball, rotating pendulum, and switching circuit.

The bouncing ball model with dynamic contact phase and states ball position , ball velocity, and ball deformation is modelled during flight by

$$\dot{x} = v, \qquad \dot{v} = -g$$

$$\dot{w} = -\frac{k}{d} \cdot w \qquad (15)$$

Flight ends, if the ball gets in contact with the ground, given by the event function

$$h(x, v, w) = x + w \qquad (16)$$

Although event function (16) is simple, it must be made a differential equation

$$\frac{dh}{dt}(x, v, w) = \dot{x} + \dot{w} = v - \frac{k}{d} \cdot w$$

The modified system with independent variable h for use of the Henon method is then given by

$$\frac{dx}{dh} = \frac{d \cdot v}{d \cdot v - k \cdot w}$$

$$\frac{dv}{dh} = -\frac{d \cdot g}{d \cdot v - k \cdot w}$$

$$\frac{dw}{dh} = -\frac{k \cdot w}{d \cdot v - k \cdot w}$$

Henon's method only would fail, if flight velocity and deformation change both get zero – which cannot happen or add to zero – which could happen in a very rare case.

Another test model from Benchmark C20, the rotating pendulum, is at present under investigation. Use of Henon's method can range from simple to complex, depending on the chosen coordinate system.

SUMMARY - OUTLOOK

Henon's method offers itself as charming alternative for state event handling. Perhaps the main advantage is the fact, that the ODE solver is not interrupted by a numerical root finding algorithm – whereby both algorithms influence each other. On the other hand, the problem of singularities in the modified state space must not be neglected.

First numerical tests with the cited benchmarks are satisfying and promising because of the 'simpler' algorithms. At present the method is tested within master theses using the ARGESIM Benchmarks and an extended rotor – stator model.

After discussion with experts from mechatronic it is intended to investigate the method within new research project in more detail. There, also possible links to F. Cellier's and E. Kofman's QSS method are to be considered: QSS - Quantized State Systems – integration works with a discretisation of state instead of time, so that state events are determined by a state increment known in advance.

REFERENCES

Henon, M., 1982. *On the Numerical Computation of Poincare Maps.* Physics 5D, 412-414.

Henon, M., 1964. *L'evolution Initiale d'un Amas Spherique.* Ann. Astrophys. 27 (1964), 83-91.

ARGESIM Benchmarks – Benchmark List. www.argesim.org, 2012.