PERGAMON Computers and Mathematics with Applications 43 (2002) 65–79

An International Journal
computers & mathematics
with applications

www.elsevier.com/locate/camwa

Efficient Integration over Discontinuities for Differential-Algebraic Systems

GUIYOU MAO*
Center of Computation, Jianghan Aviation Equipment Institute
No. 2 Xinyuan Road, Xiangfan, Hubei Province, 441003 P.R. China

L. R. PETZOLD
Department of Computer Science, University of California
Santa Barbara, CA 93106, U.S.A.

(Received January 2001; accepted February 2001)

Abstract—In recent years, hybrid (discrete/continuous) dynamic systems that exhibit coupled continuous and discrete behavior have attracted much attention. Many engineering problems can be formulated as ODE/DAEs which can be solved by numerical methods. In process design, dynamic optimization and simulation, however, many systems of interest experience significant discontinuities during transients. This paper describes some simple strategies for discontinuity detecting and handling in DAE embedded systems. The described algorithm supports flexible representation of state conditions in propositional logic. By making full use of the discontinuity functions, both efficiency of integration and effectiveness of discontinuity detection are achieved. Considerable care is taken to handle problems that can arise during mode transitions. We outline the implementation in a general-purpose solver DASPKE for differential-algebraic equations containing discontinuities, and present some numerical experiments which illustrate its effectiveness.

1. INTRODUCTION

Many dynamic systems may be simulated by ordinary differential equations (ODEs) written as

$$\begin{aligned} \frac{dy_i}{dt} &= f_i(y_1, y_2, \ldots, y_n, u, t), \qquad (i = 1, \ldots, n), \\ y_i(0) &= y_{i0} \end{aligned} \tag{1}$$

or more generally, by differential-algebraic equations

$$\mathbf{f}(\mathbf{y}_d, \dot{\mathbf{y}}_d, \mathbf{y}_a, \mathbf{u}, t, \mathbf{p}) = 0, \tag{2}$$

where $\mathbf{y}_d$ and $\mathbf{y}_a$ are usually referred to as differential and algebraic variables, respectively, $\mathbf{u}$ are known system inputs, and $\mathbf{p}$ are parameters. Such a system of DAEs can be most efficiently integrated using an error controlled variable step algorithm such as Runge-Kutta, Adams, Gear,

*The work of this author was partially supported by China Scholarship Council.

0898-1221/01/$ - see front matter Typeset by $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX
PII: S0898-1221(01)00272-3

DSL48S, DASPK, etc. In applied simulations, however, the system of equations frequently contains discontinuities in the form of switches, which are thrown when certain conditions are fulfilled. Equation set (1) then becomes

$$\frac{dy_i}{dt} = f_{ij}(y_1, y_2, \ldots, y_n, u, t), \qquad (i = 1, n;\ j = 1, m), \tag{3}$$

where the m states of the equation system are determined by a set of arbitrary algebraic discontinuity functions $g_k(t, y_1, y_2, \ldots, y_n)$, $(k = 1, \ldots, N_g)$.

A switch or change of state occurs at a critical point defined by one of the g_k passing through zero. Similarly, the DAE model (2) containing discontinuities can be formulated as a combined discrete/continuous simulation problem [1,2]. The time interval of interest $[t^{(0)}, t^{(f)}]$ is partitioned into N_s continuous subintervals $[t^{k-1}, t^k)$, $k = 1, \ldots, N_s$. The combined simulation problem can be defined as

$$\begin{aligned} \mathbf{f}^{(k)}\left(\mathbf{y}_d^{(k)}, \dot{\mathbf{y}}_d^{(k)}, \mathbf{y}_a^{(k)}, \mathbf{u}^{(k)}, t\right) &= 0, \\ \mathbf{u}^{(k)} &= \mathbf{u}^{(k)}(t), \end{aligned} \tag{4}$$

where $t \in [t^{k-1}, t^k)$, $(k = 1, \ldots, Ns)$, $\mathbf{f}(k) : R^{n^{(k)}} \times R^{n^{(k)}} \times R^{m^{(k)}} \times R^{l^{(k)}} \times R$, and $\mathbf{y}^{(k)} \in R^{n^{(k)}}$. Given the initial time $t^{(0)}$, the end of each subinterval is determined by the occurrence of a state event during the process of simulation.

These different possible functional forms are some times known as the *modes* of a hybrid system. Hybrid systems may be modeled by a variety of embedded differential or difference equations, and may exhibit discontinuous behavior at discrete points in time known as *events*, including nonsmooth forcing, switching of modes, and jumps in the state. A switch refers to discrete changes in the functional form of (4) as a consequence of events that occur instantaneously at a point in time. The time of occurrence of events may either be defined *a priori* (time event) or implicitly by the system state satisfying some conditions (state event). For example, a class of implicit switches can be expressed with the notation

$$\begin{aligned} f^{(1)}(t, \mathbf{u}, \mathbf{y}, \dot{\mathbf{y}}, \mathbf{p}) = 0, \qquad & \forall t \in \left[0, t^{(f)}\right] : g(t, \mathbf{y}) > 0, \\ f^{(2)}(t, \mathbf{u}, \mathbf{y}, \dot{\mathbf{y}}, \mathbf{p}) = 0, \qquad & \forall t \in \left[0, t^{(f)}\right] : g(t, \mathbf{y}) \leq 0, \end{aligned}$$

where $f^{(1)}$ is a subset of the model equation (2) that is inserted in the overall model when the (scalar) state condition $g(t, \mathbf{y}) > 0$, and $f^{(2)}$ is inserted otherwise. In this case, the state events defining switching times, i.e., $t^* = t \in (0, t^{(f)}) : g(t, \mathbf{y}) = 0$ are not known in advance because they are functions of the system state, and therefore, the timing and order of equation switches are also not known in advance.

The logical expression related to a state event is referred to as a state condition. The mode changes whenever a state condition is satisfied. In practice, the state conditions of a system may be determined by a simple relational expression $\alpha(t, \mathbf{y}) > \beta(t, \mathbf{y})$, such as $1 - y_1 > 0.5$; or by a complex logical proposition that contains a set of logical operators (e.g., AND, OR, NOT) and a set of relational expressions (with relational operators $<, >, \geq, \leq$). Taking the friction problem in [3] as an example, the motion of a sliding object of mass m subject to an applied force F_a and frictional resistance F_f is governed by

$$m\ddot{x} = F_a - F_f.$$

Friction, F_f, is a discontinuous function for static and dynamic conditions such that

(i) if $(\dot{x} = 0) \wedge (|F_a| \leq F_s)$, $F_f = F_a$,
(ii) if $((\dot{x} = 0) \wedge (F_a > F_s)) \vee (\dot{x} > 0)$, $F_f = F_1 + F_2 x$,
(iii) if $((\dot{x} = 0) \wedge (F_a < -F_s)) \vee (\dot{x} < 0)$, $F_f = -F_1 + F_2 x$.

State events pose particular problems for simulation. In each step in the DAE model, whenever a discontinuity function crosses zero the associated state conditions may switch their logical values. An event is defined as the *earliest* time at which one of the currently pending state conditions becomes true. So, in the process of simulation, the actual mode switching depends on which state condition is satisfied first. This in turn depends on parameters and /or initial conditions. Once the system is in one of these new modes, it may evolve in a completely different way from that if it had switched to another mode. Thus, it is very important to locate the state events in strict time order and implement the correct mode changes. This requires that events must not be missed by stepping over them completely.

Following a successful event detection, the system enters a new mode and the algorithm must continue to detect new state events. Most state event detection algorithms are based on the root-finding of discontinuity functions. One problem, i.e., the problem of false zero crossing of discontinuity functions, arises during mode transitions and must be handled properly. This involve the difficulties of discontinuity sticking and erroneous zero crossing. The first difficulty is related to the transition of state variables, which is concerned with how an initial condition for the new mode is determined in terms of the final state of the predecessor mode. We use a transition function

$$T_j^{(k)}\left(\dot{\mathbf{y}}^{(k)}, \mathbf{y}^{(k)}, \mathbf{u}^{(k)}, \dot{\mathbf{y}}^{(j)}, \mathbf{y}^{(j)}, \mathbf{u}^{(j)}, \mathbf{p}, t\right) = 0$$

to map the final values of the variables in the current mode (k) to the initial values in the next mode (j). After location of a state event, a consistent initialization calculation is required to restart the integration in the new subinterval [4,5]. This calculation is usually based on the assumption of continuity of the differential variables, i.e., given values for the differential variables, the new system of DAEs is solved to find consistent initial values for the algebraic variables and the time derivatives of the differential variables, which may be different from their original values if the event time t^* is not on the mesh points. Thus, the process of reinitialization may lead to false zero crossings, which Barton [6] termed as "discontinuity sticking", i.e., the same zero crossing is detected again in the first step following that event. The second type of false zero crossing happens when the discontinuity function that caused the mode switching deflects at state event time t^* [7]. These false zero crossings will greatly reduce the integration efficiency, and can even lead to completely wrong simulation results if they are not handled properly.

This paper will present some strategies for discontinuity detecting and handling in simulation with DAE models. These strategies have been implemented in DASPKE for the handling of mode switching as a new extension to the general-purpose DAE solver DASPK3.0 [8], which uses variable-order, variable-stepsize backward differention formula (BDF) methods and a choice of two linear system solution methods: direct (dense or banded) or Krylov (iterative). Finally, we present some simulation results for a set of example problems.

2. LITERATURE REVIEW

Most approaches to the discontinuity problem for DAE systems employ a discontinuity locking mechanism. The idea is to "lock" the function evaluator for the IVP solver so that the equations evaluated are fixed while an integration step is being taken, thus presenting a smooth vector field to the solver. During the process of event detection, the same mode is used regardless of the current values of the inputs and state conditions. This strategy greatly facilitates state event location in the simulation of hybrid discrete/continuous systems. To detect any event in a step, a set of discontinuity functions (or variables) $\mathbf{z}(t) = \mathbf{g}(t, \mathbf{u}, \mathbf{y})$ are constructed from the relational expressions in the pending state conditions. When one or more relational expressions change their values, the state conditions may change accordingly.

Implemented in conjunction with the gear integration algorithm, Carver [3] tackled the discontinuity problem for stiff equation sets (ODEs). The discontinuity function $z(t)$, instead of being treated algebraically, is regarded as an additional differential equation, i.e., $\dot{z} = \dot{g}(t, \mathbf{u}, \mathbf{y})$ and

appended to the ODE system. The functions ϕ of scaled derivatives are stored up to the current order q such that the prediction of z at time $t+h$ is given simply by

$$z_{t+h} = z_t + \sum_{i=1}^{q} \phi_i(z_i)$$

and at any other stepsize h_1 by

$$z_{t+h_1} = z_t + \sum_{i=1}^{q} \phi_i(z_i) \left(\frac{h_1}{h}\right)^i. \tag{5}$$

The strategy for discontinuity detection is based on the signs of $s_1 = z_t * z_{t+h}$ and $s_2 = \dot{z}_t * \dot{z}_{t+h}$.

(a) If $s_1 > 0$ and $s_2 > 0$, continue the integration.
(b) If $s_1 > 0$ and $s_2 < 0$, reduce the stepsize h and repeat to attain Condition (a) or (c).
(c) If $s_1 < 0$ and $s_2 > 0$, solve equation (5) for h_1.

This method can greatly reduce the number of function evaluations in the neighbourhood of the discontinuity, but needs to adjust (reduce) the stepsize frequently for a possible discontinuity.

As an extension to the general DAE solver DASSL, Petzold [4] developed DASRT which uses the Illinois algorithm [9] to find the roots of the functions $\mathbf{g}(t, \mathbf{y})$ along the trajectory of the solution. Software for the handling of mode transitions at the discontinuities is not given and must be provided by the user.

Rather than using discontinuity functions to handle state events, Gear [10] deals with the problem in a direct manner by examining the behavior of the local truncation error and estimates the order of the discontinuity, and hence, the stepsize which keeps the error under control while stepping over the discontinuity.

Pantelides [11] uses the state conditions directly. The state event is detected by comparing the logical values of the state conditions at t_k and t_{k+1}. If an event is detected, the time of its occurrence is located by a bisection algorithm with interpolation formulae $\mathbf{y}_d^p(t)$ and $\mathbf{y}_a^p(t)$. The algorithm supports implicit state conditions having complex logical structures involving multiple clauses linked with AND/OR/NOT operators, but cannot handle discontinuity sticking problems.

Barton [6] noticed possible changes in the values of algebraic variables and discontinuity variables after reinitialization in the new mode, which may lead to discontinuity sticking problems, and developed an algorithm consisting of two main phases:

(1) event detection, and
(2) consistent event location (event polishing).

In the first phase, an efficient hierarchical polynomial root-finding procedure based on interval arithmetic guarantees detection of the state event t^* even if multiple state condition transitions exist in an integration step. In the second phase, an extra equation: $g^*(\mathbf{y}_d, \dot{\mathbf{y}}_d, \mathbf{y}_a, \mathbf{u}, t_l^*) = \pm\epsilon_g$ is introduced to form a system of nonlinear equations

$$\begin{aligned} \mathbf{f}(\mathbf{y}_d, \dot{\mathbf{y}}_d, \mathbf{y}_a, \mathbf{u}, t_l^*) &= 0, \\ \mathbf{u} &= \mathbf{u}(t_l^*), \\ \mathbf{y} &= \mathbf{y}^p(t_l^*), \\ g^*(\mathbf{y}_d, \dot{\mathbf{y}}_d, \mathbf{y}_a, \mathbf{u}, t_l^*) &= \pm\epsilon_g, \end{aligned} \tag{6}$$

where ϵ_g, called the discontinuity tolerance, is a small positive value and is bounded by $\epsilon_g^{\min}$ and $\epsilon_g^{\max}$. This equation set is solved while imposing time tolerances δ and Δ ($\delta \leq \Delta$) to determine the consistent state event time t_l^*

$$t_l^* \in [t^*, t^* + \delta] \qquad \text{or} \qquad t_l^* \in [t^*, t^* + \Delta]$$

at which consistency between the differential and algebraic variables is retained. Thus, the discontinuity functions (or variables) do not change their signs after consistent reinitialization under the new mode and consequently avoids discontinuity sticking problems. Barton's method needs some extra polishing computations when an event is detected. This polishing process will encounter erroneous zero crossing problems when the discontinuity function that caused the mode switching deflects at t_l^* (Figure 1). Another special situation as shown in Figure 2 will result in the second phase failing to find a consistent event time t_l^* even after an event has been found at t^*.

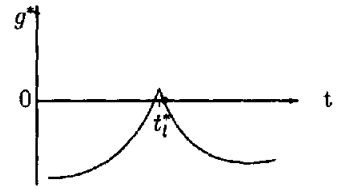

Figure 1. Erroneous zero-crossing.

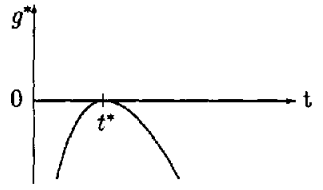

Figure 2. g^* tangent to t-axis.

Birta [7] approximates $z_i(t, \mathbf{y})$ using a cubic model to determine whether or not $z_i(t, \mathbf{y})$ has a cross-over within a step. The approach for dealing with the erroneous zero crossing problem is simply not to monitor that particular discontinuity function until it has, in fact, changed sign. If another discontinuity function has a zero crossing within the first integration step following t^*, the integration process is returned to t^* and the step-size is reduced by half. If the detection process still predicts a cross-over within the new interval then this process is repeated. In this way, the integration step-size used in proceeding from a point t^* at which a discontinuity occurs is reduced until the detection process is satisfied that no further zero-crossing of discontinuity functions takes place in this initial interval. This algorithm is very effective and robust, but there is no provision for the discontinuity sticking problems in DAE-based algorithms, and the first stepsize may need to be adjusted in the new mode.

3. DESCRIPTION OF THE PROCEDURE

The new algorithm consists of two phases:

(1) discontinuity detecting, and
(2) discontinuity handling.

3.1. Discontinuity Detecting

3.1.1. Root-finding for the discontinuity functions

The goal is to determine the specific time at which any of the discontinuity functions crosses zero. Suppose the discontinuity function is a continuous function of t and $\mathbf{y}$, for a given interval $[t_a, t_b]$, where $g(t_a, \mathbf{y}_a)$ and $g(t_b, \mathbf{y}_b)$ have opposite signs. According to the intermediate value theory of calculus, there must exist at least one root r in the interval $[t_a, t_b]$. The interval $[t_a, t_b]$ corresponds to any step advanced from t_n to t_{n+1}. For a set of arbitrary discontinuity functions $g_i(t, \mathbf{y})$ $(i = 1, \ldots, N_g)$, we use the Illinois algorithm [9] to find the earliest root t_c in a step. The algorithm systematically moves the endpoints of the interval closer and closer together until an interval of arbitrarily small width (δ_1) that brackets the root (zero point) is obtained (Figure 3).

3.1.2. Improvements

As described above, the rootfinding algorithm requires that the signs of $g(t, \mathbf{y})$ at the two ends of the interval are opposite. Consequently, only odd numbers of roots in the interval are found. To prevent missing the root detection of an even numer of roots of $g(t, \mathbf{y})$, the interval $[t_a, t_b]$ or,

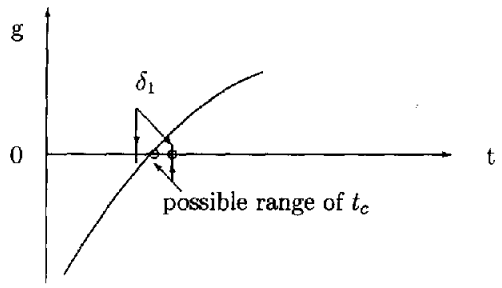

Figure 3. Time tolerance for root-finding algorithm.

in another words, the stepsize must be small enough to include only one root in it. Unfortunately, the integration and the evaluation of discontinuity functions are two separate processes. Although the integration process is very efficient, this may lead to problems for the rootfinding. That is, the stepsize may be relatively "large" compared with the sign changes of the discontinuity functions, so there exists the risk of missing the detection of the zero crossing within a step. To reduce this risk, the stepsize should be small enough to include at most one root. Although this can be achieved by directly limiting the stepsize h, the question remains how should a suitable stepsize be chosen? Another problem is that the efficiency of the integration can be greatly reduced by choosing the stepsize too conservatively so as not to miss any roots. We relate the discontinuity functions with the stepsize by appending the differential variables

$$\dot{y}_{n+i} = \frac{dg_i(t, \mathbf{y})}{dt}$$

or the algebraic variables

$$y_{n+i} = g_i(t, \mathbf{y}), \qquad (i = 1, \ldots, N_g)$$

to the original DAE system. Because all of the m discontinuity functions are included in the error control mechanism, the stepsizes are adaptively controlled by the discontinuity functions.

There are, in each step from t_a to t_b, three possibilities:

(1) if all of the discontinuity functions have sign changes, the earliest root is found using the Illinois algorithm;
(2) if none of the discontinuity functions have sign changes, we examine the signs of their derivatives at t_a and t_b to decide whether one more evaluation of $g_i(t, \mathbf{y})$ $(i = 1, \ldots, N_g)$ at the middle of the interval is needed;
(3) if some of the discontinuity functions have sign changes, the Illinois algorithm is used to find the first root t_c.

Then for those discontinuity functions having no sign changes, the signs of their derivatives $\dot{g}_i(t_a, \mathbf{y}_a)$ and $\dot{g}_i(t_b, \mathbf{y}_b)$ are checked. If they are different, we further search for a possible earlier root in the interval $[t_a, t_c]$. Thus, the following strategy applies.

In this way, for short-lived switching, only one more evaluation of the discontinuity function is taken. The event detection is improved without further reducing the stepsize. On the other hand, the N_g augmented discontinuity variables can be treated as quadrature variables [8], which

Situations		Actions
All $g_i(t_a, \mathbf{y}_a) \cdot g_i(t_b, \mathbf{y}_b) < 0$	NA	Solve for the first root t_c
All $g_i(t_a, \mathbf{y}_a) \cdot g_i(t_b, \mathbf{y}_b) > 0$	All $\dot{g}_i(t_a, \mathbf{y}_a) \cdot \dot{g}_i(t_b, \mathbf{y}_b) > 0$	Continue integration
	Any $\dot{g}_i(t_a, \mathbf{y}_a) \cdot \dot{g}_i(t_b, \mathbf{y}_b) < 0$	Divide the interval $[t_a, t_b]$ by 2, and check for root in the subintervals
Some $g_i(t_a, \mathbf{y}_a) \cdot g_i(t_b, \mathbf{y}_b) > 0$	but $\dot{g}_i(t_a, \mathbf{y}_a) \cdot \dot{g}_i(t_b, \mathbf{y}_b) < 0$	Solve for the first root t_c, and search for earlier root in $[t_a, t_c]$

can be handled very efficiently in DASPK3.0. Thus, both efficiency of integration and effectiveness of discontinuity detection are fulfilled. Although the combination of including the discontinuity functions in the stepsize selection and monitoring the signs of the derivative of the discontinuity function to decide whether an evaluation of g at the midpoint of the current interval may be necessary to avoid missing multiple roots does not absolutely guarantee that no roots will be missed, we have found that it is both reliable and efficient in practice.

3.2. Discontinuity Handling

3.2.1. Notation

For correct mode switching, we not only need to detect the zero crossings of the discontinuity functions, but also to determine in which direction they cross zero. Here we introduce some notation to identify the two directions of zero crossing. We define the logical values of the discontinuity function

$$\begin{aligned} \mathrm{logg}(i) &= +1, \quad \text{if } g_i(t,\mathbf{y}) \geq 0, \quad \text{and} \\ \mathrm{logg}(i) &= -1, \quad \text{if } g_i(t,\mathbf{y}) < 0. \end{aligned}$$

If the discontinuity functions have been added to the DAEs, the values of $g_i(t,\mathbf{y})$ $(i = 1,\ldots,N_g)$ are available by interpolation. But there exists the problem of discontinuity sticking [6]. Here we evaluate their values by using the interpolation values of the original state variables. The advantage is that there will be no discontinuity sticking problems if the original system is ODE, or if the discontinuity function $g_i(t,\mathbf{y})$ is related only to the differential variables in a DAE system.

3.2.2. Updating of the logical values of discontinuity functions

We must update the logical values of the discontinuity functions in the following cases:

(1) immediately after each initialization;
(2) at time $t^* + h_{\min}$, where t^* is the event time (or $t^{(0)}$) and $h_{\min}$ is the minimal stepsize determined by the machine ϵ (here $h_{\min} = 4.0 * UROUND * \max\{|t_n|, |t_{\text{out}}|\}$);
(3) any time a zero crossing among the discontinuity functions is detected.

For the first case, we use the values of the discontinuity variables (at t^*) to determine their logical values according to the above notation. For the second case, we evaluate $g_i(t,\mathbf{y})$ at $t^* + h_{\min}$ and use the same notation as in (1), but if $g_i(t,\mathbf{y})$ equals to zero, its logical value is updated by multiplying the original value (at t^*) by -1. For the third case, we locate the time t_c of the earliest zero-crossing of the discontinuity functions in a step, and update the corresponding logical value(s) (multiplying by -1) for those having a zero-crossing at or a little earlier than t_c (see Figure 3). However, it should also be taken as a simultaneous zero crossing if another discontinuity function has a zero crossing immediately after t_c. This is achieved by evaluation of the discontinuity functions at time $t_c+\delta_2$, where $\delta_2 > 0$ is an assigned parameter that defines the width of an equivalence band. If the sign of a discontinuity function at $t_c + \delta_2$ is different from its sign at t_c, its logical value is also updated by the current value multiplied by -1, and the zero crossing time t_c is replaced by $t_c + \delta_2$.

3.2.3. Checking of state conditions

Whenever a zero crossing at t_c among the discontinuity functions is found, we first update the corresponding logical values, then check the state conditions using these logical values to determine whether a mode switch is needed. If it is, the zero crossing time t_c is taken as the event time t^*, the DAEs are changed and the integration is reinitialized at t^*. But instead of detecting a zero crossing from t^*, the algorithm begins a new event detection from $t^* + h_{\min}$. This

means that we check pending state conditions for possible mode switching only after a minimal step has been taken in the new mode (but the initial stepsize is not necessarily limited to $h_{\min}$). This strategy assumes that a new mode must remain unchanged for a period of at least $h_{\min}$ and is also used to avoid the false zero crossings caused by discontinuity sticking and/or erroneous zero crossings (see Figure 4).

It may be possible that a zero crossing at t_c is found, but no state condition is satisfied. In this case, the algorithm continues to search for events from t_c.

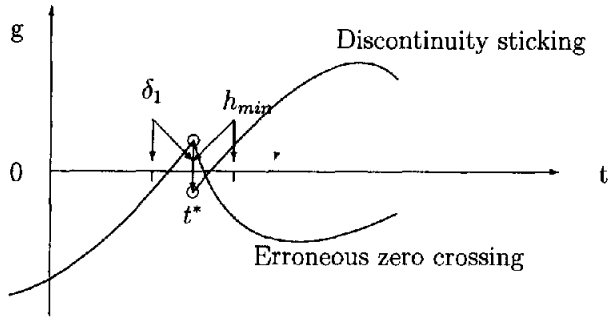

Figure 4. Avoiding false zero crossing.

3.3. Implementation

We have implemented the algorithm described above on the basis of DASPK3.0 to form a new package DASPKE with the capability of discontinuity handling for DAE systems. In each integration step, DASPKE searches for the first root(s) of the discontinuity functions. When they are found, the corresponding logical values of the discontinuity functions are updated (multiplied by -1) and the related state conditions are checked to see whether mode switching is needed, and if so, the DAEs change accordingly. INFO(1), INFO(11) are set to "0" and "1", respectively, to indicate the restart of the integration and reinitialization under the new mode. Note that the searching interval is not always from t_n to t_{n+1}. Depending on which way the user chooses to run the code, the end point is t_{n+1} or t_{out}, whichever comes first. Below is the pseudo-code of the algorithm.

ALGORITHM.

Eventhandle( )
Initialize Jstate, Logg, $t_n = t_0$
While($t_n < t_{\text{out}}$) Do
A step forward from t_n to $t_{n+1} = \min(t_n + h, t_{\text{out}})$
Rootfind (**G**, N_g, t_c, Root, Jroot, t_n, t_{n+1})
If (Root) Then
Loggupdate(Logg, Jroot)
Modcheck(Jstchange, Logg, Jstout, Jstin)
If (Jstchange) Then
$t^* := t_c$
Jstate = Jstout
Reinitialize at t^* and search for new event from $t^* + h_{\min}$
Else
Continue root-finding from t_c to t_{n+1}
Endif
Else
$t_n = t_{n+1}$
Endif

```
  Endwhile

Rootfind (G, N_g, t_c, Root, Jroot, t_n, t_{n+1})
Begin
  t_c := min {t_c^i : g_i(t_c^i - δ1) · g_i(t_c^i) ≤ 0, t_n < t_c^i ≤ t_{n+1}}
  For i:=1 To N_g
    If (g_i(t_c - δ_1) · g_i(t_c) ≤ 0) Then
      Jroot(i):=1
      Rroot:=1
    Else
      If (g_i(t_c + δ_2) · g_i(t_c) ≤ 0) Then
        Jroot(i)=:1
        t_c = t_c + δ_2
        Rroot:=1
      Endif
    Endif
  End
End

Loggupdate(Logg, N_g, Jroot)
Begin
  For k:=1 To N_g
    If (Jroot(k) = 1) Then
      Logg(k) = Logg(k) * (-1)
    Endif
  End
End

Modcheck(Jstchange, Logg, Jstout, Jstin)
Begin
   Return Jstout,Jstchange
End
```

4. NUMERICAL EXPERIMENTS AND DISCUSSION

Here we provide some simulation results for the new algorithm applied to various test problems. In all of the examples, the absolute and relative tolerances are 10^{-5}. The time tolerances for the root-finding δ_1 and the time equivalence band of a simultaneous zero crossing δ_2 are set to 10^{-9}. For comparison purposes, we also provide some results using Petzold's root-finder (DASRT) and Barton's algorithm (DSL48E).

EXAMPLE 1. Consider the differential equation

$$\begin{aligned} \dot{y}(t) &= y(t), \qquad \sin(20\pi t) \geq 0, \\ \dot{y}(t) &= 0, \qquad \sin(20\pi t) < 0, \end{aligned}$$

with

$$y(0) = 0.1, \qquad 0 \leq t \leq 3.5.$$

To compute the value of $y(3.5)$, we need to define the discontinuity function $g(t, y) = \sin(20\pi t)$. Its sign changes define the points where the differential equation changes.

Figure 5 shows the integration result of $y(t)$ obtained by using DASRT. Apparently, the algorithm used by DASRT skips over 16 events! This is because an even number of events are

included in some of the steps, which leads to the algorithm failing to detect any of them. Table 1 shows some of the integration steps (from $t = 0.3992$ to $t = 2.0009$) taken by DASRT. It reveals that the integration steps are inappropriate for the event detection of the discontinuity function $\sin(20\pi t)$. Figure 6 gives the output of $y(t)$ using the new algorithm with the discontinuity function as algebraic variable appended to the original ODE. The stepsizes are adjusted by the discontinuity function. Consequently, the capability of event detection is increased.

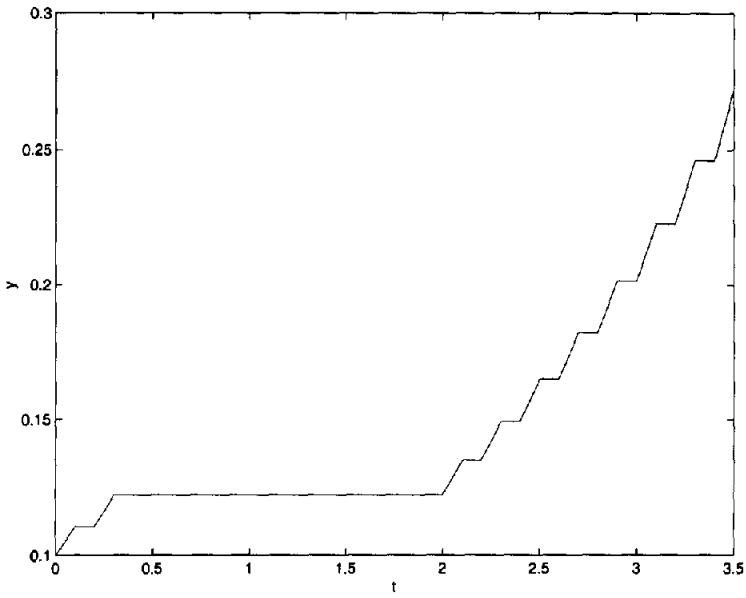

Figure 5. Output of y(t) by DASRT.

Table 1. Partial steps taken by DASRT for Example 1.

Time (t)	$\sin(20\pi t)$	y(t)
0.3992	−0.025113	0.122156
0.5016	−0.050244	0.122156
0.7064	−0.199710	0.122156
1.1160	−0.481754	0.122156
1.9352	−0.893841	0.122156
2.0000	0.0000000	0.122156
2.0009	0.0029452	0.122168

EXAMPLE 2. This is an example used by Birta [7]. The problem is given by

$$\begin{aligned} \dot{y}_1(t) &= \alpha_1 y_1(t), & y_1(0) &= 0.5, \\ \dot{y}_2(t) &= \alpha_2 y_2(t), & y_2(0) &= -0.5, \\ \dot{y}_3(t) &= y_1(t) + y_2(t), & y_3(0) &= 0. \end{aligned}$$

The initial values of parameters α_1 and α_2 are taken to be 2 and -1, respectively. The events of interest are specified as

(a) $y_1(t) = +1.0$ with $\dot{y}_1(t) \geq 0$, and
(b) $y_2(t) = -1.0$ with $\dot{y}_2(t) \leq 0$.

Upon the occurrence of either of the events, these parameters interchange their values. The output of $y_3(t)$ is most interesting.

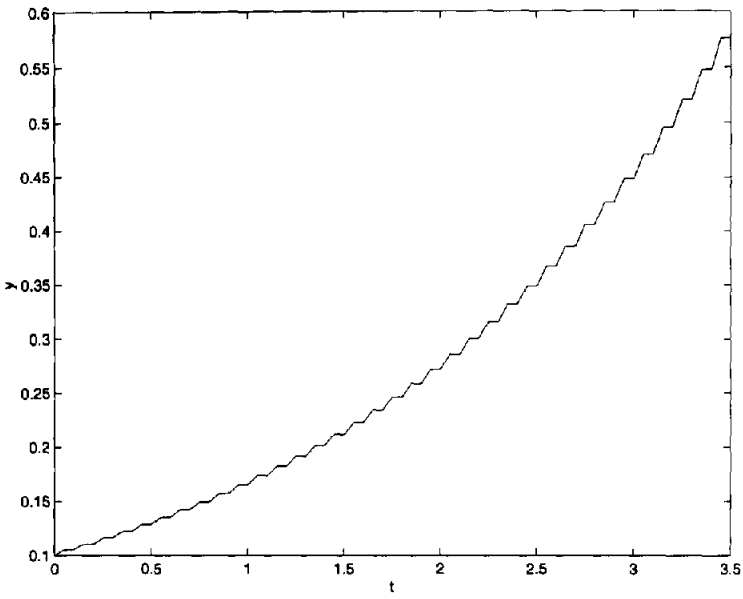

Figure 6. Output of y(t) by DASPKE.

This problem can be described by two discontinuity functions

$$g_1(t, \mathbf{y}) = 1 - y_1(t) \qquad \text{and} \qquad g_2(t, \mathbf{y}) = 1 + y_2(t),$$

which will remain nonnegative on the solution trajectories. This is a distinctive feature of this example and can be treated satisfactorily by the new discontinuity handling procedure. This problem, however, can not be expressed and solved with Barton's [6] software (DSL48E), because the mode of the system is not absolutely determined by the state conditions, but is determined by the state conditions at the event time and the mode before the event, i.e., there is no direct map between a state condition and a mode. The system may be in a different mode even if the state condition is the same (see Figure 1). In this example, the discontinuity functions deflect when an event is detected, so DSL48E cannot be used to solve this kind of problem. Figure 7 shows the integration results of $y_1(t)$, $y_2(t)$, and $y_3(t)$ using DASPKE (for $t = 0$ to 1.4), from which we can see the exceedingly small separation between event times when t increases in value.

EXAMPLE 3. Taking another problem used by Birta [7] as an example, the dynamics for this case are specified as

$$\begin{aligned} \dot{y}_1(t) &= \omega y_2(t), & y_1(0) &= 0, \\ \dot{y}_2(t) &= -\omega y_1(t), & y_2(0) &= 1, \\ \dot{y}_3(t) &= u^3(t), & y_3(0) &= 0, \end{aligned}$$

with $\omega = \pi$, and the time interval is $0 \le t \le 3$. The initial value of the input function $u(0)$ is 1. The single discontinuity function associated with the problem is

$$g_1(t, \mathbf{y}(t)) = y_1(t) - At, \qquad (A \text{ is a given constant}).$$

Upon the occurrence of each event, the value of u is replaced with $-u(t^*)y_1(t^*)$, where t^* is the time of event.

Table 2 gives some results using DASPKE and DSL48E. The event times (in the second column) for this problem can be obtained directly by solving $y_1(t) - At = 0$, i.e., $\sin(\pi t) - At = 0$. For most of the possible parameters A, both algorithms obtain similar results. But when the parameter A

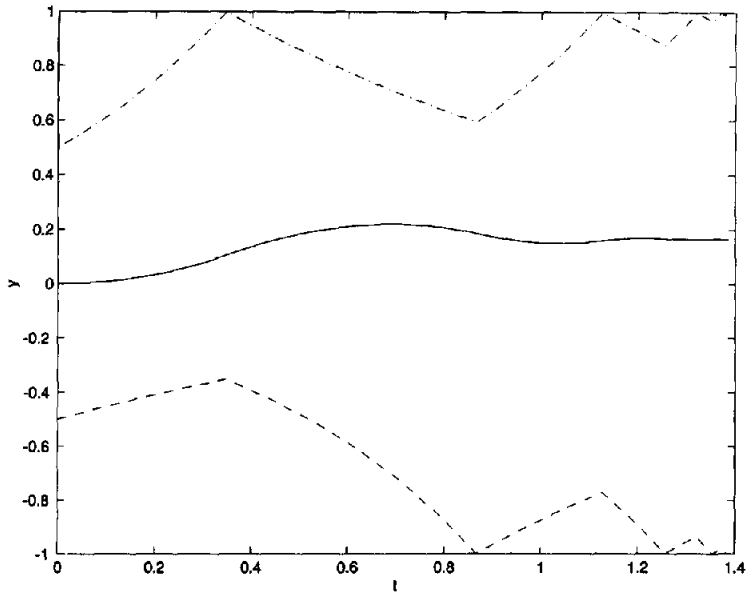

Figure 7. Solution of Example 2 by DASPKE. ($y_1(t)$: dot-dashed line, $y_2(t)$: dashed line, $y_3(t)$: solid line.)

is chosen as $A = 0.403$, the trajectory of $y(t) = 0.403 * t$ is nearly tangent to $y_1(t)$ at some point, which means that the second and third events are very close. Both algorithms can detect these events, but around the third event, a lot of steps are needed. This is because frequent events are detected and very small initial stepsizes are taken. Eventually, our algorithm gets the result of $y_3(3.0)$, but due to the short-lived state conditions and the failure to calculate a consistent event time t_l^*, DSL48E failed to compute the final value of $y_3(3.0)$.

Table 2. Results for Example 3.

A	Event Times (Direct)	Event Times (DASPKE)	Event Times (DSL48E)	$x_3(3.0)$ (DASPKE)	$x_3(3.0)$ (DSL48E)
0.35	0.898206 2.29733 2.62827	0.898223 2.297377 2.628262	0.898211 2.297334 2.628245	0.393366	0.393346
0.40	0.884843 2.41850 2.50000	0.884858 2.418639 2.499864	0.884849 2.418542 2.499899	0.243088	0.2431339
0.403	0.884047 2.446759 2.471331	0.884064 2.447192 2.470742	0.884055 2.441031 2.471031	0.242976	Failed
0.45	0.871693	0.871708	0.871696	0.7432421	0.743240

Figure 8 shows the integration results of $y_1(t)$ and $y_3(t)$ using DASPKE for $A = 0.35$. There are three discontinuities in the interval.

EXAMPLE 4. The fourth test problem is concerned with current flow in a diode circuit, and has been previously used by Carver [3] and Barton [6]. There are three state conditions, each with four relational expressions. This is a typical DAE system with propositional logic state conditions. For this problem, all of the three algorithms obtain correct results, but DASRT experiences six false zero crossings including two instances of discontinuity sticking and four erroneous zero crossings. The new algorithm successfully avoided all the false zero crossings and implemented correct mode switching. Compared with DASRT, DASPKE usually needed more

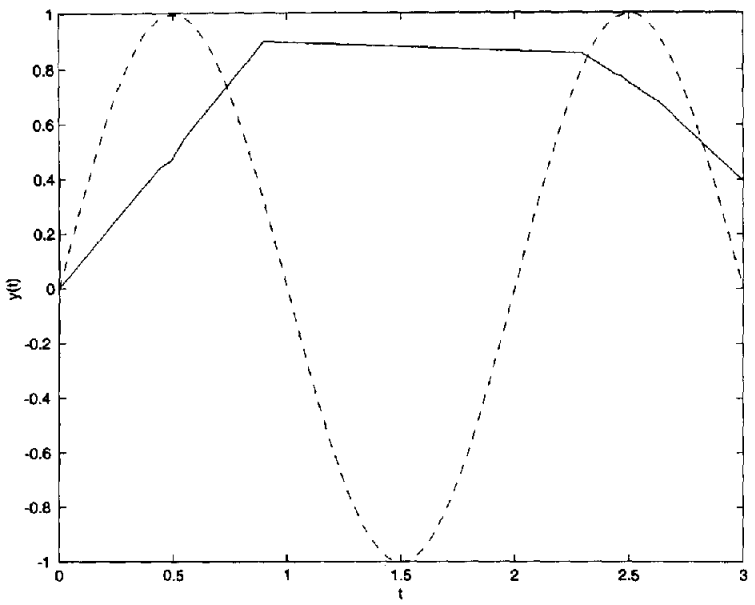

Figure 8. Solution of Example 3 by DASPKE. ($y_1(t)$: dashed line, $y_3(t)$: solid line.)

steps and residual evaluations, but fewer discontinuity function evaluations because most of the false zero crossings can be avoided during event detection. Table 3 shows the number of steps (NStep), residual evaluations (NRes) and discontinuity function evaluations (NGev) needed when using DASRT, DASPKE, respectively. Figures 9 and 10 show the solution profile produced by DASPKE (discontinuity functions as algebraic variables).

Table 3. Comparison of computational cost for Example 4.

	NStep	NRes	NGeV	NDis[1]	NError[2]
DASRT	477	2229	610	2	4
DASPKE[3]	486	2778	574	0	0
DASPKE[4]	471	3163	568	0	0
DASPKE[5]	465	2834	572	0	0

[1]NDis: Number of discontinuity sticking

[2]NError: Number of erroneous zero crossing

[3]: Discontinuity functions as quadrature variables

[4]: Discontinuity functions as differential variables

[5]: Discontinuity functions as algebraic variables

5. CONCLUSIONS

Efficient integration over discontinuities in a DAE model requires accurate detection and location of state events and proper handling of problems arising at mode transitions. While many ODE/DAE solvers such as DASSL and DASPK are reliable and efficient, they are not equipped for handling discontinuities. Some root-finding algorithms do a good job of detecting discontinuities, but in case of a short-lived switch such as Example 1, discontinuities can be missed unless the user is sufficiently alert to impose a stepsize limit. There is a tradeoff between efficiency of integration and effectiveness of detecting a discontinuity.

By appending the discontinuity functions (either as algebraic variables or differential variables) to the original DAE system, the stepsizes for the integration are adaptively adjusted in part by the discontinuity functions. Based on the derivative information of discontinuity variables at the two

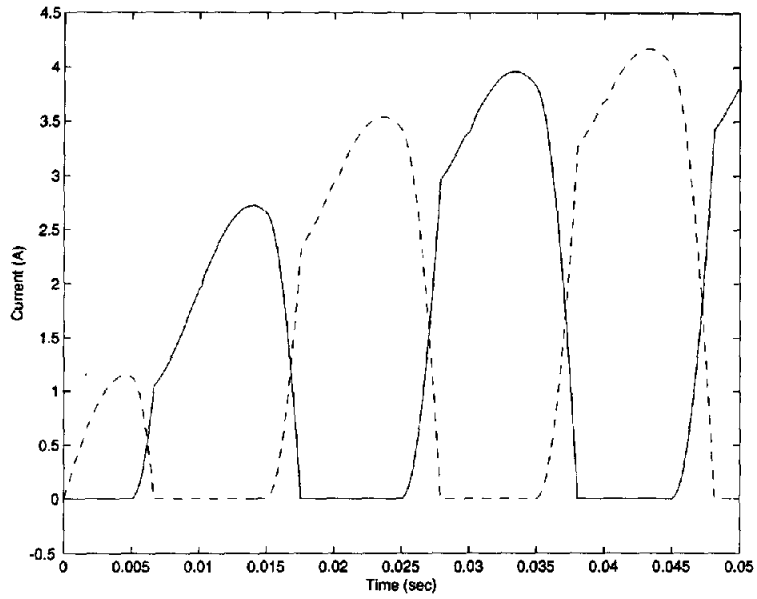

Figure 9. Output of $i_1(t)$ and $i_2(t)$ by DASPKE. ($i_1(t)$: dashed line, $i_2(t)$: solid line.)

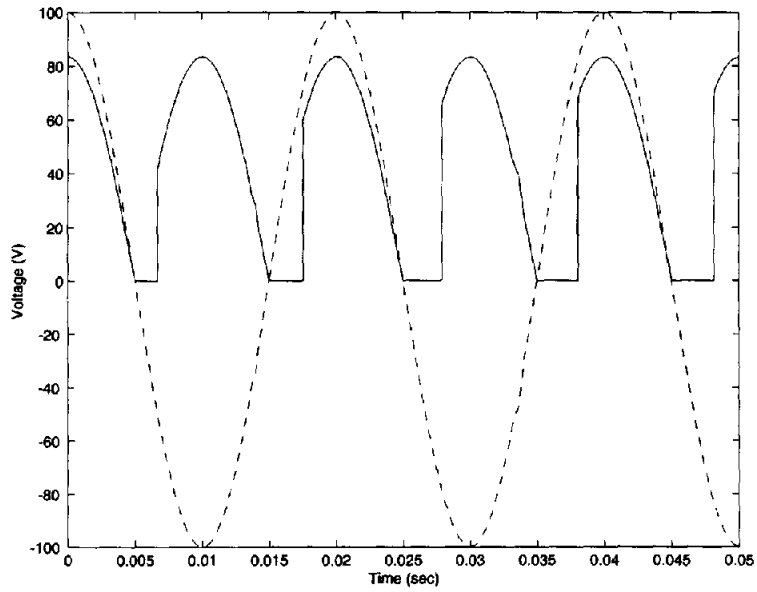

Figure 10. Output of $v_1(t)$ and $v_3(t)$ by DASPKE. ($v_1(t)$: dashed line, $v_3(t)$: solid line.)

ends of a step, a decision regarding whether one extra evaluation of the discontinuity functions in a step is made, which effectively solves the problem of missing the detection of an even number of zero crossings in a step without limiting the stepsizes of integration. The root-finding algorithm itself cannot guarantee that the first root is found when a discontinuity function has multiple zero crossings in a step, but with these strategies combined, the purpose can be achieved.

With special notation for the logical values of the discontinuity functions, a strategy for state condition checking and mode determination is proposed, which can handle short-lived ($< h_{\min}$) state condition changes and effectively avoid false zero crossings at mode transitions caused by discontinuity sticking and erroneous zero crossings without adjusting the initial stepsize in the new mode.

The new software supports flexible representation of state conditions using propositional logic. For general engineering problems where the number of discontinuity functions is small relative to the size of the state system, this algorithm can be very efficient. Simulations from various test problems show very good results for the computation of state variables in DAE systems containing discontinuities.

REFERENCES

1. P.I. Barton, R.J. Allgor, W.F. Feehery and S. Galan, Dynamic optimization in a discontinuous world, *Ind. Eng. Chem. Res.* **37**, 966–981, (1998).
2. S. Galan, W.F. Feehery and P.I. Barton, Parametric sensitivity functions for hybrid discrete/continuous systems, *Applied Numerical Mathematics*, 1–31, (1998).
3. M.B. Carver, Efficient integration over discontinuities in ordinary differential equation simulations, *Mathematics and Computers in Simulation* **20**, 190–196, (1978).
4. K.E. Brenan, S.L. Campbell and L.R. Petzold, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*, Second Edition, SIAM, (1995).
5. P.N. Brown, A.C. Hindmarsh and L.R. Petzold, Consistent initial condition calculation for differential-algebraic systems, *SIAM J. Sci. Comput.* **19** (5), 1495–1512, (1998).
6. T. Park and P.I. Barton, State event location in differential-algebraic models, *ACM Transactions on Modeling and Computer Simulation* **6** (2), 137–165, (1996).
7. L.G. Birta and T.I. Ören, A robust procedure for discontinuity handling in continuous system simulation, *Transactions of the Society for Computer Simulation* **2** (3), 189–205, (1985).
8. S. Li and L. Petzold, Software and algorithm for sensitivity analysis of large-scale differential-algebraic systems, *J. Comput and Appl. Math.*, (2000).
9. G. Dahlquist and A. Björk, *Numerical Methods*, (N. Anderson, translator), Prentice-Hall, Englewood Cliffs, NJ, (1974).
10. C.W. Gear, Solving ordinary differential equations with discontinuities, *ACM Transactions on Mathematical Software* **10** (1), 23–44, (March 1984).
11. C.C. Pantelides, Speedup-recent advances in process simulation, *Computers and Chem. Engng.* **12** (7), 745–755, (1988).
12. A.J. Preston and M. Berzins, Algorithms for the location of discontinuities in dynamic simulation problems, *Computers and Chemical Engineering* **15** (10), 701–713, (1991).
13. U.M. Ascher and L.R. Petzold, *Computer Methods for Ordinary Differential Equations and Differential Algebraic Equations*, SIAM Publications, Philadelphia, PA, (1998).
14. H. Mathews, *Numerical Methods for Computer Science, Engineering, and Mathematics*, Prentice-Hall, Englewood Cliffs, NJ, (1987).
15. K.E. Atkinson, *An Introduction to Numerical Analysis*, Second Edition, John Wiley & Sons, (1989).
16. S. Li, L. Petzold and W. Zhu, Sensitivity analysis of differential-algebraic equations: A comparison of methods on a special problem, *Applied Numerical Mathematics* **32**, 161–174, (2000).
17. V. Gopal and L.T. Biegler, A successive linear programming approach for initialization and reinitialization after discontinuities of differential-algebraic equations, *SIAM Journal on Scientific Computing* **20** (2), 447–467, (1998).