Available online at www.sciencedirect.com
SCIENCE DIRECT
ELSEVIER
Mathematical and Computer Modelling 40 (2004) 1335–1350
MATHEMATICAL AND COMPUTER MODELLING
www.elsevier.com/locate/mcm

Pulse Detection Software for Initial Value ODEs

A. M. Hynick and P. Keast
Department of Mathematics and Statistics, Dalhousie University
Halifax, Nova Scotia, B3H 3J5, Canada
keast@mathstat.dal.ca

P. H. Muir
Department of Mathematics and Computing Science, Saint Mary's University
Halifax, Nova Scotia, B3H 3C3, Canada
muir@stmarys.ca

Abstract—In many physical models ordinary differential equations (ODEs) arise with the general form, $\mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}) + \mathbf{g}(t)$, in which abrupt but large changes of limited duration, known as pulses, occur in $\mathbf{g}(t)$. These pulses may begin at times which are not known beforehand and may have unknown durations. If the duration is sufficiently short, standard differential equation solvers may miss the pulse completely, stepping over it, especially if, prior to the pulse, the solution is well behaved. In this paper, we discuss software which employs standard initial value ODE software and a process of detect sampling to attempt to detect, and handle efficiently, any pulses which arise. A key advantage of this software and the algorithms for pulse detection and handling described in this paper is that they do not involve modification of the initial value ODE solver. The performance of the new software will be investigated by applying it to several test problems exhibiting pulses. The results show that pulses can be detected and efficiently handled by the new software and that significant computational savings are achieved. © 2005 Elsevier Ltd. All rights reserved.

Keywords—Pulse detection, Initial value ordinary differential equations, Defect sampling, Efficiency, Performance.

1. INTRODUCTION

We consider systems of initial value ordinary differential equations (IVODEs) of the form

$$\mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}(t)) + \mathbf{g}(t), \qquad \mathbf{y}(t_0) = \mathbf{y}_0, \tag{1}$$

where $\mathbf{y} : \mathbb{R} \to \mathbb{R}^n$, $\mathbf{f} : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$, $\mathbf{g} : \mathbb{R} \to \mathbb{R}^n$, and $\mathbf{y}_0 \in \mathbb{R}^n$. It is assumed that $\mathbf{f}$ is continuous and that $\mathbf{g}$ is zero except during some relatively short time period, whose duration and position may be unknown, and where it acquires an instantaneous and relatively large value. Such a sudden change will be called a *pulse*. Depending on $\mathbf{f}$, the underlying system of IVODEs may be stiff or nonstiff.

The work of these authors was partially supported by grants from the Natural Sciences and Engineering Research Council of Canada.
The authors would like to acknowledge the many helpful suggestions provided by the referees.

0895-7177/05/$ - see front matter © 2005 Elsevier Ltd. All rights reserved. Typeset by $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX
doi:10.1016/j.mcm.2005.01.023

To see the difficulty presented by systems such as (1), it is necessary to consider how most standard IVODE solvers behave. Such a solver begins at t_0 and computes solution approximations at a set of points t_i, $i = 1, 2, 3, \dots$, where, usually, $t_i < t_{i+1}$. These points are selected by the software on the basis of a step selection algorithm which attempts to take steps that are as large as possible while keeping some estimate of the local error in each step within some user provided tolerance. The solution approximations will of course involve calculations based on evaluations of the right-hand side of the ODE system. If, prior to a pulse being encountered, the solution of the IVODE is well behaved, showing no rapid changes, the software will increase the stepsize to improve efficiency while still satisfying the error requirements. The danger then, however, is that after a successful step *the code may step completely over the pulse*, and continue on the assumption that all is well, *even though a major feature of the solution has been missed.* An example of this kind of behavior, when the well-known IVODE code LSODE [1,2] is used to attempt to solve a problem with a pulse, is given in Figure 6; LSODE computes a solution which shows no indication of a pulse, whereas the correct solution exhibits an obvious reaction to the pulse, as in Figure 5.

Two of the most critical quantities associated with a pulse are where it begins and how long it lasts; we will refer to these as the *start* and *duration* of the pulse. When both of these are known, we shall see that it is easy to force an IVODE solver to detect the pulse and integrate through it in an efficient fashion. We shall refer to this as Case (a). However, we are more interested in three other cases:

(b) the start of the pulse is known but its duration is unknown,
(c) the duration of the pulse is known but its start is unknown, and
(d) neither the duration nor the start is known.

We wish to emphasize that in this paper, we assume that the duration of the pulse is very much smaller than the natural stepsizes employed by the ODE solver and that, if there are multiple pulses, they are also relatively well separated, with respect to the usual stepsizes taken by the IVODE solver. *This means we are assuming that the pulses are missed by the IVODE solver when it is allowed to employ its standard step selection algorithm.* Thus, the pulse definition we are using is very much dependent on the assumption that the IVODE solver is able to take large stepsizes; this assumption is most relevant for IVODE solvers that employ high order or stiff methods where considerable effort has been invested to allow the solver to take large steps.

When the start of the pulse is unknown, simply detecting that a pulse is present becomes the central issue. The general approach we will use to detect a pulse is based on sampling the defect. In this approach, one assumes that in addition to providing a discrete numerical solution at the output points, t_i, the IVODE solver also provides a continuous solution approximation (i.e., interpolant) over each step. The defect is defined to be the amount by which this continuous solution approximation fails to satisfy the IVODE. That is, if $\mathbf{u}(t)$ is the continuous solution approximation at some point t, then the defect at t is given by

$$\mathbf{r}(t) = \mathbf{u}'(t) - [\mathbf{f}(t, \mathbf{u}(t)) + \mathbf{g}(t)]. \tag{2}$$

Note that this assumes that the IVODE solver also provides the derivative of $\mathbf{u}(t)$. A measure of the quality of $\mathbf{u}(t)$ is obtained by sampling the defect, i.e., computing $\mathbf{r}(t)$, at a set of points within the current step.

When the underlying method upon which the IVODE solver is based is a multistep method (see, e.g., [3]), the solution approximation at t_i is based on a computation in which the only evaluation of the right-hand side of (1) within the current step, $(t_{i-1}, t_i]$, occurs at t_i. Thus if the pulse begins and ends strictly within this interval the solver has no way of noticing it. In the software described in this paper, we will augment the calculation done by the IVODE solver with a defect sampling process, which involves evaluating the defect and thus the right-hand side of (1) at several points within $(t_{i-1}, t_i]$. When, the duration of the pulse is known, we can guarantee that the pulse will be detected. Even when the duration is unknown, we can substantially improve the likelihood that the pulse will be detected. Once a pulse has been detected, it is important to the efficiency of the IVODE solver that the pulse be handled in an appropriate way; i.e., since the beginning and end of the pulse look like discontinuities to the IVODE solver, see, e.g., [4], it is important that our algorithms control the integration step sequence so that the IVODE code steps into and out of the pulse efficiently.

In this paper, we have assumed that the pulse term is not state-dependent, i.e., $\mathbf{g}(t)$ does not depend on $\mathbf{y}(t)$. The state-dependent case is significantly more complicated since one may need to have a reasonably accurate approximation to the solution at the potentially unknown time at which the pulse is triggered. The approach to be described in this paper allows the pulse to be efficiently detected even when the IVODE solver gives a poor approximation to the solution in the region of the pulse, provided the pulse depends only on time. Further investigation of the state-dependent pulse case will be the subject of future work.

We note that in some applications it will be possible for the user to determine the circumstances which will trigger a pulse. If this is possible, most IVODE solvers will allow the user to handle the pulse using an *event location* option. Such an option allows the user to instruct the code to return whenever a prespecified condition arises. Upon return in such a case, the user can then sharply reduce the stepsize and force the solver to correctly detect the pulse. We wish to emphasize that in this paper, we are assuming the more general case in which the user does not have sufficient *a priori* knowledge to predict the conditions which will cause a pulse to occur.

An important goal of our work is to develop software which can detect and efficiently treat pulses while employing an unmodified IVODE solver. The key idea is that the algorithms and software presented in this paper are to be essentially independent of the IVODE solver used to integrate the ODE system, provided the solver has several commonly available facilities (to be identified later in this paper). This goal addresses a significant software issue: while it might be possible to obtain a more sophisticated algorithm by focusing on a modification of LSODE, this would significantly limit the applicability of our work to other IVODE solvers. In this paper, we consider coupling our software with an IVODE solver based on a family of multistep methods, but with only minor modifications, our software could be employed, for example, with a one-step Runge-Kutta solver, provided that the solver can provide continuous solution and first derivative approximations over each step, based on, for example, a continuous extension of the Runge-Kutta formula; see, e.g., [5].

This paper is organized as follows. In Section 2, we briefly review some related work. Section 3 considers the four different pulse cases mentioned above and describes a detailed algorithm for the efficient treatment of pulses within the context of these four cases. Section 4 presents and discusses our numerical results and Section 5 provides our summary, conclusions, and suggestions for future work.

2. RELATED WORK

The most closely related work is in the area of handling of discontinuities in IVODE software. However, a fundamental difference is that in our context the primary problem is the actual detection of the pulse whereas in the treatment of discontinuities, detecting the presence of a discontinuity is only a secondary issue; the primary issues involve being efficient in precisely locating and stepping across the discontinuity. A standard IVODE code will often locate and step across a discontinuity with a large number of failed steps in a very inefficient way; see, e.g., [4], Figure 1. In that paper, Gear and Osterby discuss modifications to the stepsize and order selection algorithms of a variable order multistep code in order to allow it to efficiently locate and step over discontinuities in $\mathbf{f}(t, \mathbf{y}(t))$ or its derivatives. Based on an estimate of the order of the discontinuity (i.e., the lowest derivative of $\mathbf{f}(t, \mathbf{y}(t))$ that exhibits a discontinuity) and its magnitude, a sufficiently small step size in the region prior to the discontinuity is determined in order to allow the code to step over the discontinuity with a local error that will be within the user defined tolerance. The order of the discontinuity also dictates the order of the method used by the code at it steps over the discontinuity. Related work is discussed by Enright *et al.* [6] where the authors discuss modifications to an IVODE code based on Runge-Kutta methods to allow it to efficiently locate and step over discontinuities in $\mathbf{f}(t, \mathbf{y}(t))$ or its derivatives. Within a step where a discontinuity is suspected, the main idea is to use several defect samplings, based on evaluations of the local high-order interpolant from the *previous* step, to accurately locate the discontinuity, and then to step over the discontinuity using an evaluation of the interpolant from the previous step. (The interpolant from the previous accepted step must be employed because no interpolant is available for the current step which has failed due to the presence of the discontinuity.)

While the discontinuity handling problem is of course related to the pulse detection and handling problem (in fact, for pulses of long duration, the pulse problem is equivalent to a pair of discontinuities), for pulses of short duration (which is a fundamental assumption of this paper), the primary issue, as mentioned above, is the actual detection of the pulse. When the start of the pulse is unknown a great percentage of the computational effort is associated with determining if a pulse is present within the current step taken by the IVODE code. Once this is determined, only a relatively minor cost is associated with accurately determining the start of the pulse (and the end of the pulse if the duration is also unknown).

3. PULSE DETECTION AND TREATMENT

Our pulse detection software employs a reliable IVODE solver which is used to solve the differential equations in a standard way except for interaction with the pulse detection algorithm at various times. Almost any IVODE program could be used, provided that it has certain features which the detection program requires. These features are as follows.

(i) The IVODE software should have the ability to restart at any time t with no information given about the solution or previous computation except the value of y at t. We call such a restart a *cold start*.

(ii) It should be possible to request a return from the IVODE solver after every accepted step, with the possibility of continuing, using all the available information currently known about the solution. This will be called a *continuation*.

(iii) After an accepted step, an interpolant should be available to enable the user to efficiently compute approximations to the solution and its first derivative at arbitrary points within the current step.

(iv) It should be possible to specify a value of t, say t_{crit} beyond which the solver will not integrate. Without this feature, the code might take a last step beyond the stopping value, and then interpolate to obtain the solution at this stopping value.

As mentioned earlier, the interpolant will be used to compute a defect, as in (2). Our defect sampling approach is based on the following observation. *Suppose that we have an interpolant that is based only on evaluations of the right-hand side of the ODE system which are outside the pulse. Then, a large defect will be obtained for any sampling that takes place within the pulse.*

We will call a defect *large* if, for some j,

$$\frac{|r_j(t)|}{\max(1, |f_j(t, \mathbf{u}(t)) + g_j(t)|)} > \frac{1}{2}, \tag{3}$$

where r_j, f_j, and g_j are the j^{th} components of $\mathbf{r}(t)$, $\mathbf{f}(t, \mathbf{u}(t))$, and $\mathbf{g}(t)$, respectively, and t is the sample point. This is somewhat conservative but it is consistent with the fact that we are using LSODE, a code which does not use local extrapolation—see, e.g., [5]. If we were to use an IVODE solver that did employ local extrapolation, since the relative/absolute local error is controlled by tol, we could expect the relative/absolute defect to be at most some reasonably small multiple of tol—typical results, [7], in such a case put the defect within a factor of 2 of tol—and we could therefore replace the right-hand side of (3) with, for example, $10 \times$ tol, and have a tighter indicator of a large defect.

The software we have developed can handle a sequence of nonoverlapping pulses occurring in different components of the right-hand side of (1); however, for the purposes of clarity the following discussion will assume the presence of a single pulse.

There are three parameters defining a pulse.

(i) t_c: the time at which the pulse begins.

(ii) δ: the width or duration of the pulse interval, assumed to be small relative to the total time interval and the stepsizes employed by the IVODE solver. We require, however, $\delta > t_c \cdot \epsilon_{\text{mach}}$, where ϵ_{mach} is machine epsilon, thus ensuring that at least one machine number lies in the interval during which the pulse is active. For practical applications, we expect that the pulse width will actually be somewhat larger than this.

(iii) $g_j(t)$: for $j \in \{1, \dots, n\}$, the change in the j^{th} component of the right-hand side value contributed by the pulse, assumed to become large at t_c. That is, for some j, $g_j(t)$ is assumed to be 0 for $t < t_c$ and for $t > t_c + \delta$, and is assumed to be large in magnitude for $t_c \le t \le t_c + \delta$.

We identify in more detail the four different cases distinguished by how much is assumed to be known about the pulse.

(a) The pulse begins at a known time and has a known duration, i.e., t_c and δ are both known.

(b) The start time of the pulse is known, but the duration is unknown, i.e., t_c is known but δ is not.

(c) The duration of the pulse is known, but it is not known when the pulse begins, i.e., δ is known but t_c is not.

(d) Neither the start time nor the duration of the pulse is known, i.e., t_c and δ are both unknown.

We now provide an overview of our algorithm for handling these four cases.

STEP 1. Cases (a) and (b). The IVODE solver is asked to integrate from t_0 to the known t_c value where the pulse begins.

STEP 1. Cases (c) and (d). As the integration proceeds, after every successful step control is returned from the IVODE software to the pulse detection program. Let the time at the end of the current accepted step be t_{cur} and let the current stepsize be h_{cur}. Then, we wish to sample at a number of points in the interval between the previously accepted time, $t_{\text{prev}} = t_{\text{cur}} - h_{\text{cur}}$ and t_{cur}. If δ is known (Case (c)), the number of sample points clearly should depend on it; we choose

$$n_s = \text{number of sample point} = \frac{h_{\text{cur}}}{\delta} s,$$

where s is an integer whose value may be set by the user, with a default value of $s = 2$. This guarantees at least one sample point within the pulse. If δ is unknown (Case (d)), the user can specify a number of sample points, n_s (or the default of 20 points may be used). The sample points are chosen to be uniformly distributed across the current step.

As the defect is evaluated at the sample points, the detection algorithm looks for any sample point for which a large defect is obtained. If all calculated defects are small, the integration proceeds with a continuation at t_{cur}. If the defect is found to be large at one or more sample points, then we find the smallest value of t, say $t_{\max}$, with a large defect and the largest t, say $t_{\min}$, less than $t_{\max}$, that has a small defect—see Figure 1. We can then begin a bisection process to accurately locate t_c using $[t_{\min}, t_{\max}]$ as our interval.

The bisection algorithm we employ uses a relative tolerance of ϵ_{mach} to obtain a very accurate estimate for the beginning and end of the pulse. From the work of [4] and [6], it is clear that it is possible to save a few function evaluations by truncating this search earlier. As long as

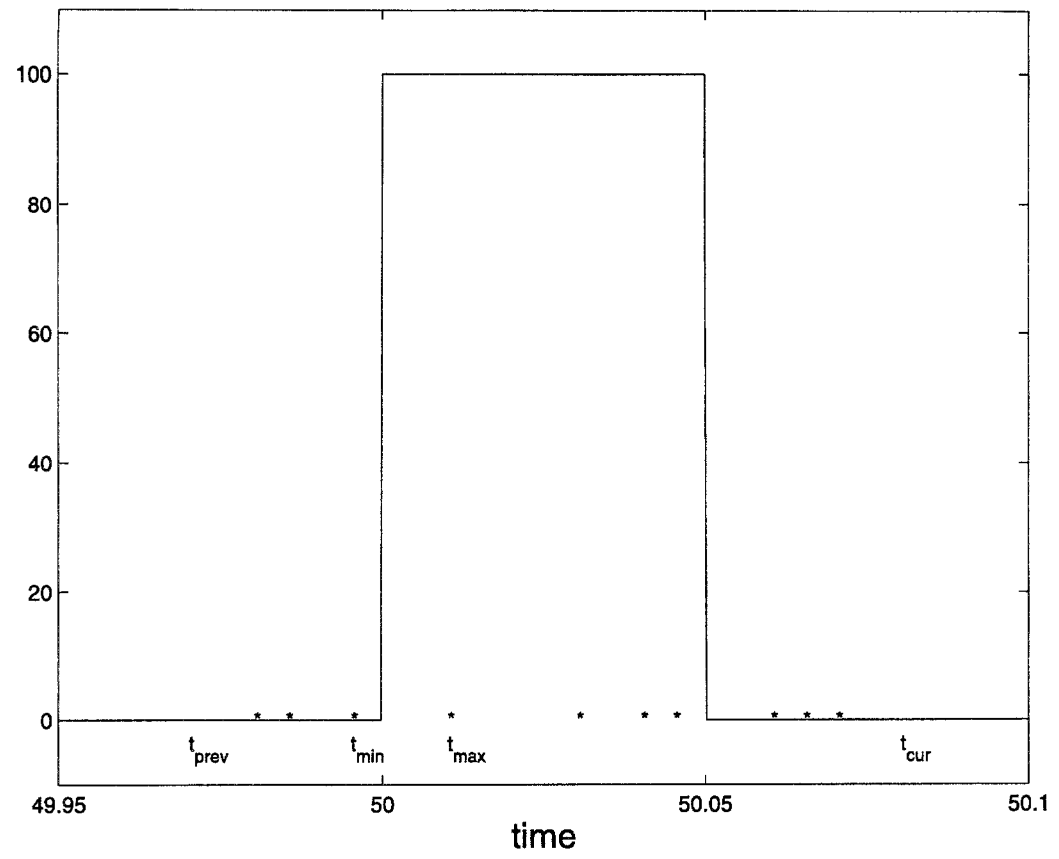

Figure 1. Plot of time vs. defect for a step containing a pulse, with defect sample points labelled by $*$. Smallest sample point inside the pulse is $t_{\max}$. Largest sample point to the left of the pulse is $t_{\min}$.

the distance to the beginning of the pulse is less than h, where h times the absolute value of the height of the pulse is less than the local error tolerance (in the appropriate relative/absolute sense), the code can step into the pulse while incurring a local error that is within the tolerance. A similar situation arises at the other end of the pulse. However, given the large number of function evaluations this algorithm already employs, we do not bother to introduce this extra complexity in our algorithm since the savings would be quite negligible. We note that by choosing the bisection tolerance in this way we avoid having to ask the user to provide such a tolerance; this is helpful since most users would not know how to choose it appropriately.

We complete this part of the algorithm for Cases (c) and (d) by evaluating the current interpolant at t_c to get a corresponding solution approximation.

STEP 2. Cases (a) and (c). We determine the end of the pulse from the known δ value.

STEP 2. Cases (b) and (d). We again use defect sampling within a bisection algorithm to accurately locate the end of the pulse. For Case (b), we recall that the IVODE solver has already integrated to t_c. We save the solution value at t_c and then ask the IVODE code to take one more step with the stepsize prescribed by its step selection algorithm for the next step; since we assume that the pulse width is much smaller than a normal step, this step will carry it over the pulse. We can then employ defect sampling as described earlier. Assuming a sufficient number of sample points, this will allow us to determine some sample points with large defects, as in the previous section. Once this has been done, we can now treat Cases (b) and (d) in the same way.

In order to locate the end of the pulse, we look for the largest sample point with a large defect, let us call it $t_{\max}$, and the smallest sample point, greater than $t_{\max}$, that has a small defect, say $t_{\min}$—see Figure 2. Then, $[t_{\max}, t_{\min}]$ is the appropriate interval for a bisection process that uses further defect sampling to accurately locate the end of the pulse.

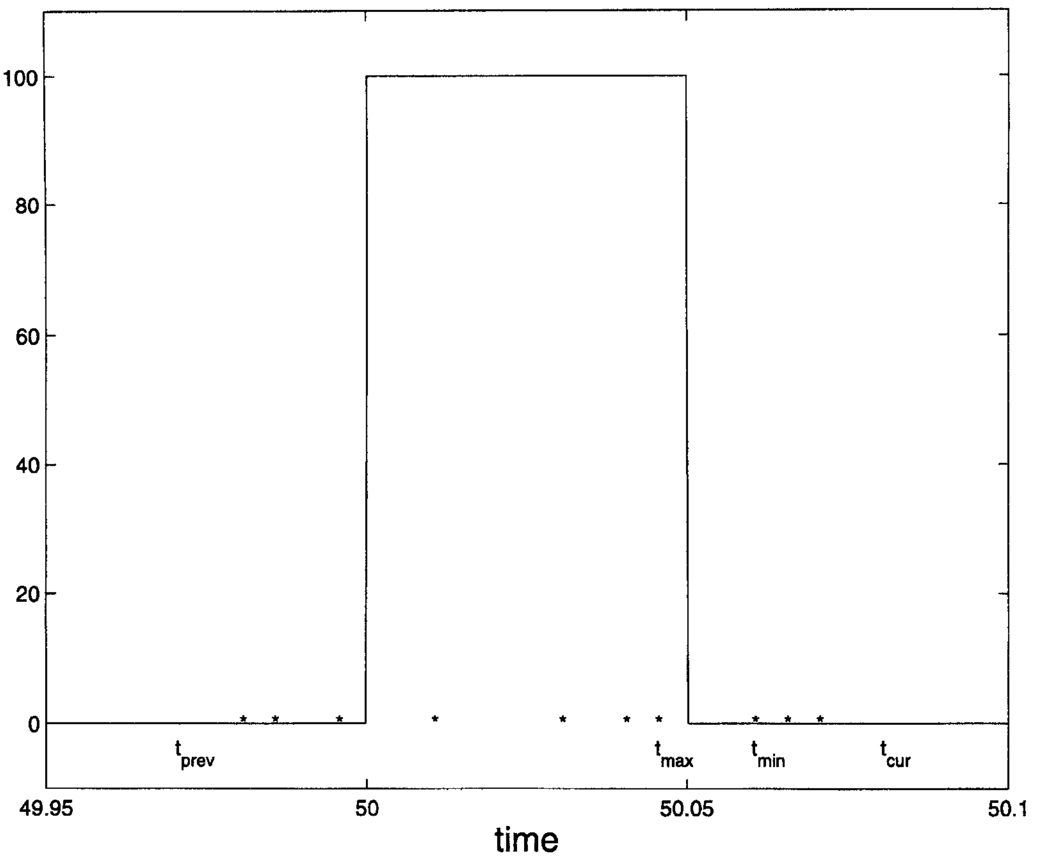

Figure 2. Plot of time vs. defect for a step containing a pulse, with defect sample points labelled by $*$. Largest sample point inside the pulse is $t_{\max}$. Smallest sample point to the right of the pulse is $t_{\min}$.

STEP 3. In all four cases, we now have complete information: both t_c and δ are known. Also, in all four cases, the IVODE solver is now at t_c with a corresponding solution approximation to provide initial conditions for the next step. Thus, the remainder of the algorithm is the same for all four cases. We restart the IVODE solver at the beginning of the pulse with a cold start, asking the code to integrate to the end of the pulse and then return control to the pulse handling software. After control is returned, the pulse detection software restarts the IVODE solver with a cold start, just beyond the end of the pulse.

In the implementation of this algorithm, when integrating to the beginning of the pulse, rather than taking $t_{\text{crit}} = t_c$, we actually take t_{crit} to be the largest machine number less than t_c. This prevents the IVODE solver from sensing a discontinuity at t_c and repeatedly incurring failed steps and halving the step size unnecessarily. Once inside the pulse, we integrate to $t_{\text{crit}} = t_c + \delta$. After exiting from the pulse, we restart at the machine number immediately after $t_c + \delta$.

4. NUMERICAL EXPERIMENTS

The above algorithms have been implemented in the code PDODE (pulse detection in ordinary differential equations). The code, together with sample driving programs, may be obtained at `http://www.mscs.dal.ca/~keast/research/pulse`. The IVODE solver employed within PDODE is LSODE [1,2]. This solver satisfies all the requirements of an IVODE solver necessary for our pulse detection code.

To demonstrate the efficacy of PDODE, we look at several test problems in which pulses occur. Although we do include this case in PDODE for convenience, we do not consider Case (a) in this section; it is straightforward to get any standard IVODE solver to handle this case efficiently: integrate to $t_{\text{crit}} = t_c$; perform a cold start at t_c with a return at $t_{\text{crit}} = t_c + \delta$; perform a cold

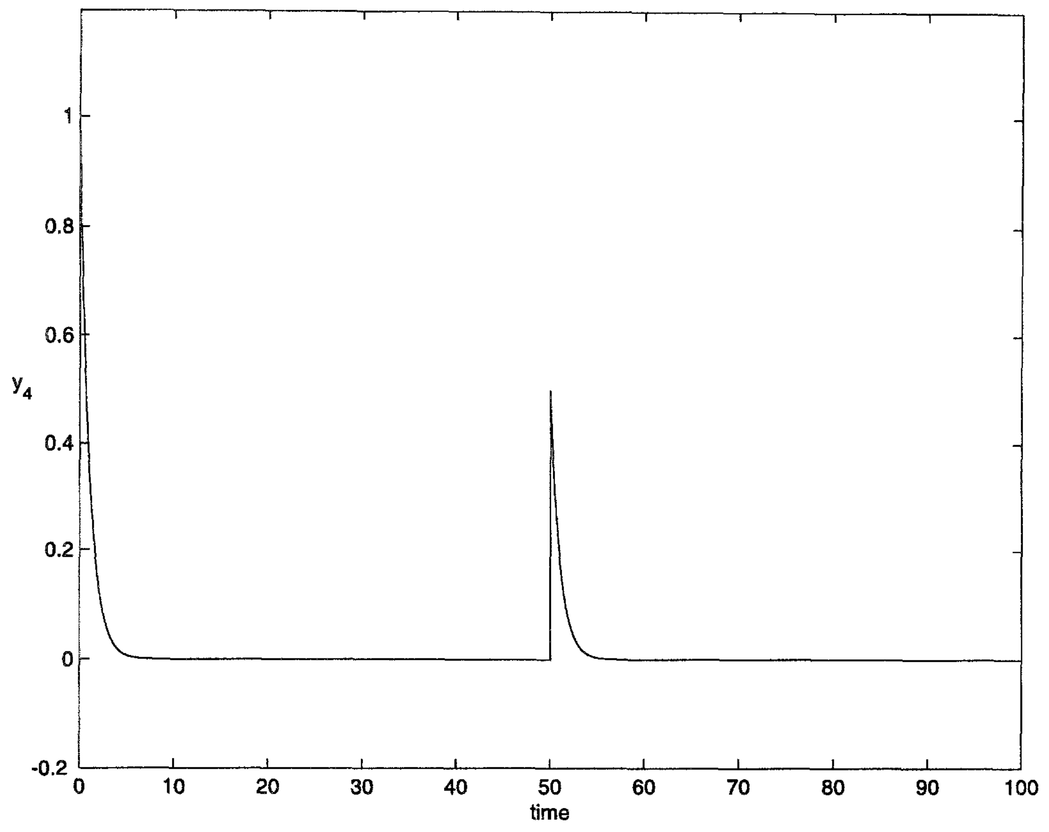

Figure 3. Correct $y_4(t)$ for Problem 1.

start at $t_c + \delta$, integrating to t_{final}. We therefore use the test problems to allow us to compare the performance of a standard IVODE solver (LSODE) with our pulse detection software, PDODE, for Cases (b), (c), and (d). Recall that in Case (b) we know t_c but not δ, in Case (c) we know δ but not t_c, and in Case (d) we know neither t_c nor δ.

We will see that for each problem both codes can obtain the correct solution. PDODE does this automatically but some intervention is required with LSODE so that it will not miss the pulse: in Case (b), we run LSODE in its usual stepsize mode with $t_{\text{crit}} = t_c$ and then perform a cold start at t_c; in Case (c), we run LSODE with the maximum stepsize parameter set slightly less than the known pulse width; in Case (d) LSODE must be run with a small maximum stepsize.

Two machine independent measures of the execution costs of ODE solvers are the required number of evaluations of the derivative, i.e., the right-hand side function, $\mathbf{F}(t, \mathbf{y}(t)) \equiv \mathbf{f}(t, \mathbf{y}(t)) + \mathbf{g}(t)$, which we will refer to as the number of function calls, and the required number of Jacobian evaluations, i.e., $\frac{\partial \mathbf{F}(t,\mathbf{y}(t))}{\partial \mathbf{y}(t)}$, which we will refer to as the number of Jacobian calls. The Jacobian evaluations involve $O(n^2)$ elements and more significantly lead to matrix computation costs that are $O(n^3)$. In the following experiments, we compare LSODE and PDODE with respect to these measures.

Several other test problems were also investigated; results similar to those reported here were obtained.

4.1. Problem 1

The first problem is SB2 from [8], to which we have added a pulse term in the fourth ODE. The equations are

$$\begin{aligned} y_1' &= -10y_1 + 3y_2, & y_2' &= -3y_1 - 10y_2, & y_3' &= -4y_3, \\ y_4' &= -y_4 + P, & y_5' &= -0.5y_5, & y_6' &= -0.1y_6, \end{aligned} \tag{4}$$

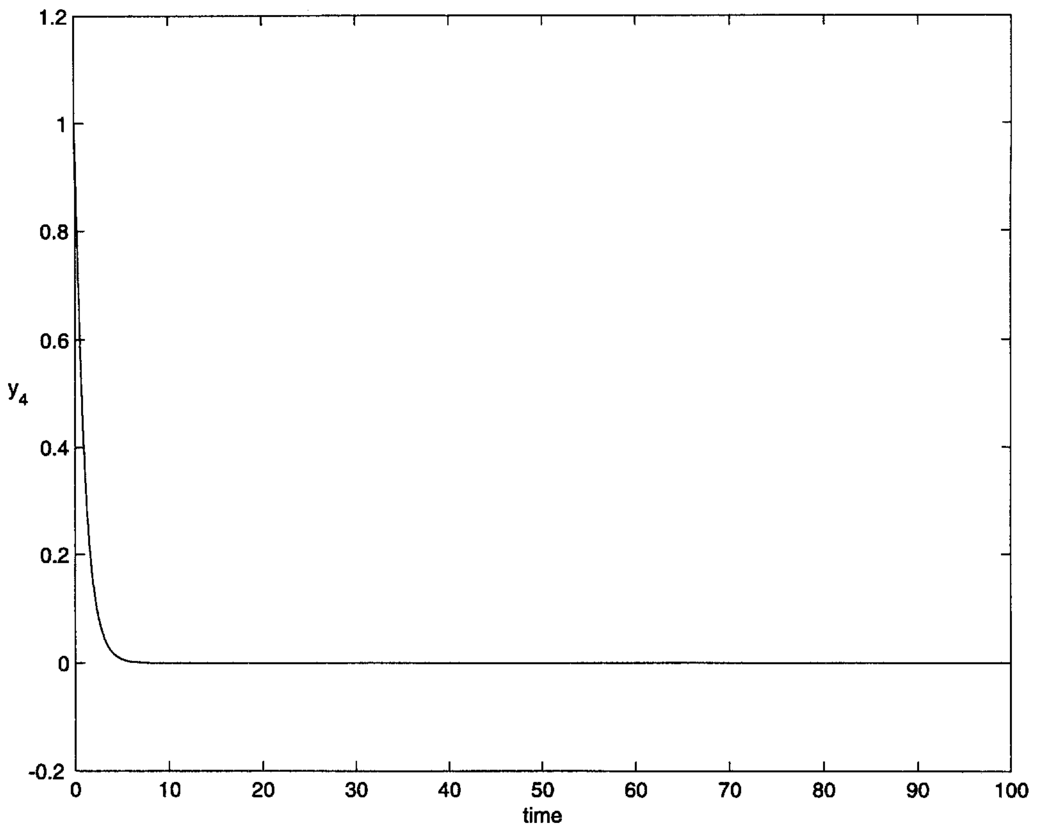

Figure 4. $y_4(t)$ from LSODE, with default $h_{\max}$, for Problem 1; pulse missed.

where P is zero except in the range $50 \le t \le 50.005$ where $P = 100$. Thus, $t_c = 50$ and $\delta = 0.005$. The initial conditions are $y_i(0) = 1$, $i = 1, \dots, 6$. This set of equations is stiff, which implies that LSODE must run in stiff mode [2]. We have $t_{\text{final}} = 100$ and relative and absolute tolerances of 10^{-10}. The correct $y_4(t)$ is shown in Figure 3. We begin our comparison by considering what happens when the above problem is given to LSODE, *with no restriction placed on the stepsize.* Since the solution components decay rapidly to zero and LSODE employs a method suitable for stiff systems, the stepsize very quickly increases from about 10^{-11} at $t = 0$ to about 1 at $t = 50$, and consequently, *LSODE misses the pulse completely.* In Figure 4 the graph of the solution given by LSODE in this case is shown.

The results for LSODE and PDODE under Cases (b), (c), and (d) are given in Table 1. PDODE finds the correct solution shown in Figure 3 and, with some intervention (as specified earlier), LSODE can also find it. From Table 1, we see that in Case (b) the codes use about the same number of function evaluations but PDODE uses about two-thirds as many Jacobian evaluations as LSODE. For Case (c) PDODE shows some improvement over LSODE in terms of evaluations of the right-hand sides, with about 20% fewer function evaluations. More significantly PDODE uses only about 5% of the number of Jacobian evaluations as LSODE does. For Case (d) LSODE takes almost ten times as many function evaluations and more than a hundred times the number of Jacobian evaluations as PDODE. (Since the pulse width is not known it is difficult to decide on an appropriate value of $h_{\max}$ for LSODE. If we were to experiment with $h_{\max} = 10^{-r}$, $r = 1, 2, 3$, LSODE would not discover the pulse until $r = 3$. Then, LSODE would use about 100,000 function evaluations and about 5000 Jacobian evaluations.)

4.2. Problem 2

One of many possible irregularities in the heart is a phenomenon that involves a wave of electrical activity reentering previously excited tissue, exciting it again. If this occurs repeatedly,

Table 1. Function calls and Jacobian calls for Problem 1.

Method	Functions Calls	Jacobian Calls
Pulse missed		
LSODE, default $h_{\max}$	638	39
Case (b). Pulse detected		
LSODE, default $h_{\max}$	983	99
PDODE, $s = 2$	921	68
Case (c). Pulse detected		
LSODE, $h_{\max} = 0.004 < \delta$	25369	1370
PDODE, $s = 2$	20887	67
Case (d). Pulse detected		
LSODE, $h_{\max} = 0.0005$	200296	10109
PDODE, $n_s = 40$	22597	67

it can lead to conditions in the heart conducive to cardiac arrhythmias. The work of Clements, Clements and Horáček [9] employs the Luo-Rudy model [10] to investigate the phenomenon of re-excitation of the heart. The Luo-Rudy model, for the action potential of a single cardiac cell, gives rise to a system of eight coupled nonlinear ODEs, one of which includes a pulse term. This system contains a number of constants and auxiliary functions which are defined in [9]; the ODE system, with initial conditions, has the form,

$$\begin{aligned}
y_1' &= \frac{\left(I_{\text{app}}(t) - \sum\limits_{k=1}^{6} I_{\text{ion},k}(\mathbf{y})\right)}{C_m}, & y_1(t_0) &= -84.0 \equiv y_1^0,\\
y_2' &= \frac{(m_{\text{inf}}(y_1) - y_2)}{\tau_m(y_1)}, & y_2(t_0) &= m_{\text{inf}}\left(y_1^0\right),\\
y_3' &= \frac{(h_{\text{inf}}(y_1) - y_3)}{\tau_h(y_1)}, & y_3(t_0) &= h_{\text{inf}}\left(y_1^0\right),\\
y_4' &= \frac{(j_{\text{inf}}(y_1) - y_4)}{\tau_j(y_1)}, & y_4(t_0) &= j_{\text{inf}}\left(y_1^0\right),\\
y_5' &= \frac{(d_{\text{inf}}(y_1) - y_5)}{\tau_d(y_1)}, & y_5(t_0) &= d_{\text{inf}}\left(y_1^0\right),\\
y_6' &= \frac{(f_{\text{inf}}(y_1) - y_6)}{\tau_f(y_1)}, & y_6(t_0) &= f_{\text{inf}}\left(y_1^0\right),\\
y_7' &= \frac{(X_{\text{inf}}(y_1) - y_7)}{\tau_X(y_1)}, & y_7(t_0) &= X_{\text{inf}}\left(y_1^0\right),\\
y_8' &= -0.0001 I_{\text{ion},2}(\mathbf{y}) + 0.07(0.0001 - y_8), & y_8(t_0) &= 0.0002,
\end{aligned} \tag{5}$$

where $\mathbf{y} = [y_1, y_2, \dots, y_8]^{\mathrm{T}}$, C_m, the membrane capacitance per unit area, is a constant, and the time-dependent pulse function, the applied current $I_{\text{app}}(t)$, given by ([9] considers several pulse cases; we provide an example here)

$$\begin{aligned}
I_{\text{app}} &= 0.0\,\frac{\mu\text{A}}{\text{cm}^2}, & 0.0\,\text{ms} &\le t \le 100\,\text{ms},\\
I_{\text{app}} &= 55\,\frac{\mu\text{A}}{\text{cm}^2}, & 100\,\text{ms} &\le t \le 100.05\,\text{ms},\\
I_{\text{app}} &= 0.0\,\frac{\mu\text{A}}{\text{cm}^2}, & 100.05\,\text{ms} &< t,
\end{aligned} \tag{6}$$

contributes a pulse to the first ODE. We therefore have $t_c = 100$ and $\delta = 0.05$. The ionic current components, $I_{\text{ion},k}(\mathbf{y})$, and the remaining functions in (5) are given in [9]. Since this system

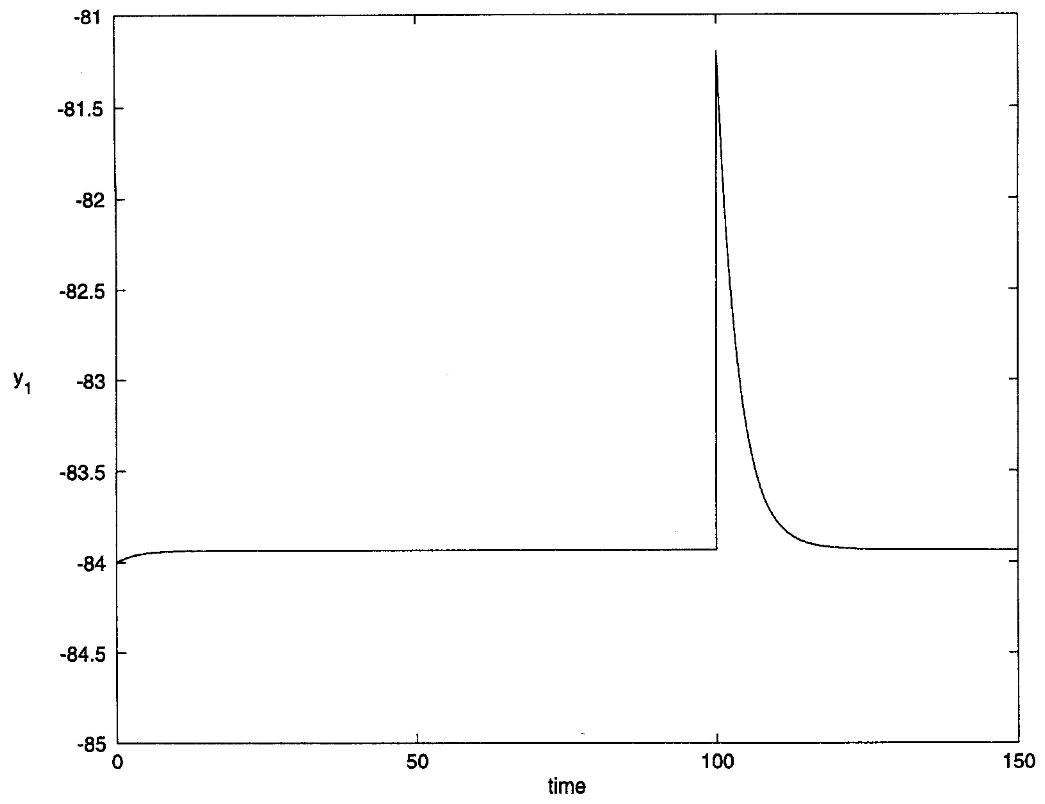

Figure 5. Correct $y_1(t)$ for Problem 3.

is stiff, LSODE must be run in stiff mode. The relative and absolute tolerances are 10^{-8} and $t_{\text{final}} = 150$. The correct y_1 is given in Figure 5.

When this problem is presented to LSODE with no restriction on the stepsize, it misses the pulse and obtains the approximation for $y_1(t)$ shown in Figure 6. When PDODE and LSODE are applied to this problem, PDODE is able to detect the pulse automatically while LSODE can be made to detect the pulse with some intervention. Both codes return a solution as in Figure 5 with performance results as given in Table 2. These results show that for Case (b) PDODE uses slightly fewer function evaluations and Jacobian evaluations than LSODE, for Case (c) LSODE uses somewhat more function evaluations than PDODE but above five times as many Jacobian evaluations, and for Case (d) LSODE uses about 25 times as many function evaluations and about 100 times as many Jacobian evaluations as PDODE.

Table 2. Function calls and Jacobian calls for Problem 3.

Method	Function Calls	Jacobian Calls
Pulse missed		
LSODE, default $h_{\max}$	201	14
Case (b). Pulse detected		
LSODE, default $h_{\max}$	837	59
PDODE, $s = 2$	752	49
Case (c). Pulse detected		
LSODE, $h_{\max} = 0.04 < \delta$	5991	253
PDODE, $s = 2$	4733	48
Case (d). Pulse detected		
LSODE, $h_{\max} = 0.005$	42431	1537
PDODE, $n_s = 20$	1628	15

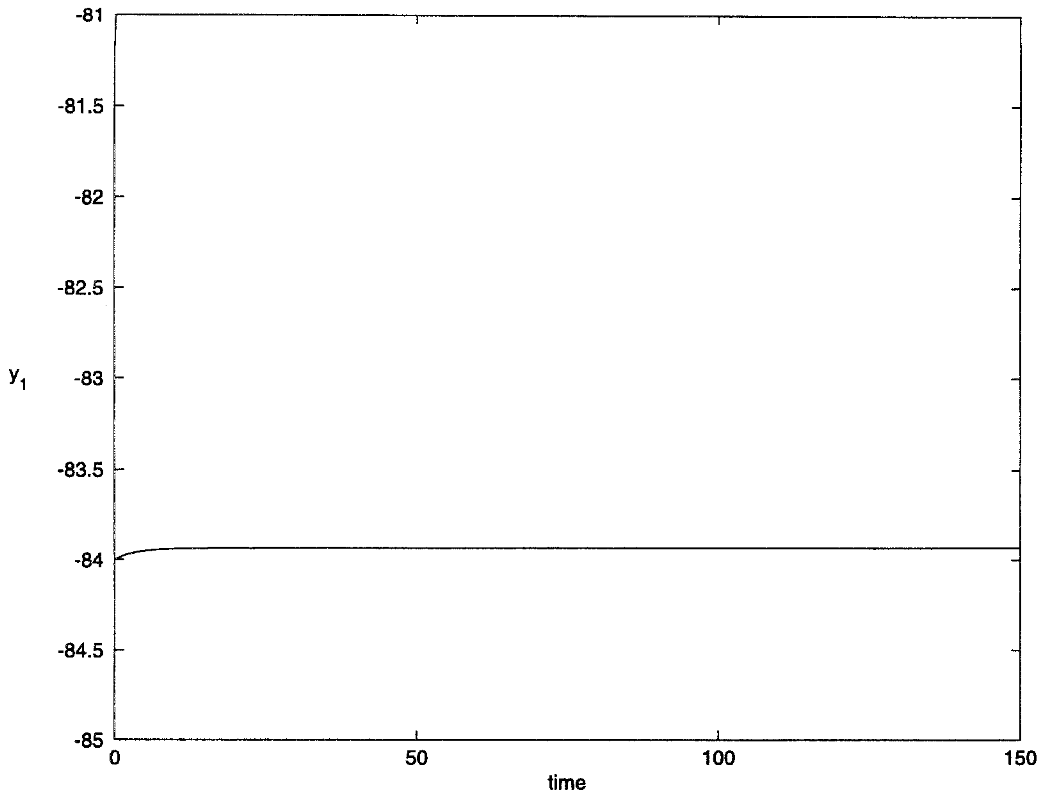

Figure 6. $y_1(t)$ from LSODE, with default $h_{\max}$, for Problem 3; pulse missed.

4.3. Problem 3

The text [5] describes a model from pharmacokinetics in which the blood level of lithium carbonate, a drug used to treat manic-depressive disorder, is studied. The model considers a dosage of one tablet every 12 hours, beginning at $t = 0$, with the assumption that the lithium carbonate is uniformly released over half an hour. This leads to equations in which pulses arise periodically.

The equations are

$$\begin{aligned} y_1' &= -5.6y_1 + P, & y_1(0.0) &= 0.0, \\ y_2' &= 5.6y_1 - 0.7y_2, & y_2(0,0) &= 0.0, \end{aligned} \tag{7}$$

where the pulse term, P, is zero except when $m/2 \le t \le m/2 + \delta$, for $m = 0, 1, 2, 3, \dots$, in which case $P = 48$; the pulse width, δ, is $1/48 = 0.0208333\dots$. We therefore have a sequence of tc values, $tc = m/2$, $m = 0, 1, 2, 3, \dots$. There is a pulse at the beginning of each day and halfway through each day, with a duration of 1/48 of a day, i.e., half an hour. The requested relative and absolute tolerances are 10^{-6} and $t_{\text{final}} = 6.0$. The blood level of lithium carbonate is modelled by $y_2(t)$. The correct $y_2(t)$ is shown in Figure 7.

Essentially any IVODE solver will detect the initial pulse since the integration begins exactly at the start of the first pulse and in fact due to the relatively long duration of these pulses the solver will likely even detect some of the subsequent pulses. However, as the integration proceeds and the natural stepsize of the IVODE solver increases, many of the pulses are missed. When we apply LSODE to the above problem with no restriction on the stepsize we obtain the $y_2(t)$ given in Figure 8. Comparing Figure 7 with Figure 8, we see that LSODE misses many of the pulses, obtaining a significantly different approximate solution.

In this paper, we assume that the pulses which arise are sufficiently narrow and far apart that the IVODE solver will miss them. However, in this pharmacokinetics problem, the pulses are of somewhat long duration, relative to the distance between them, and arise fairly frequently,

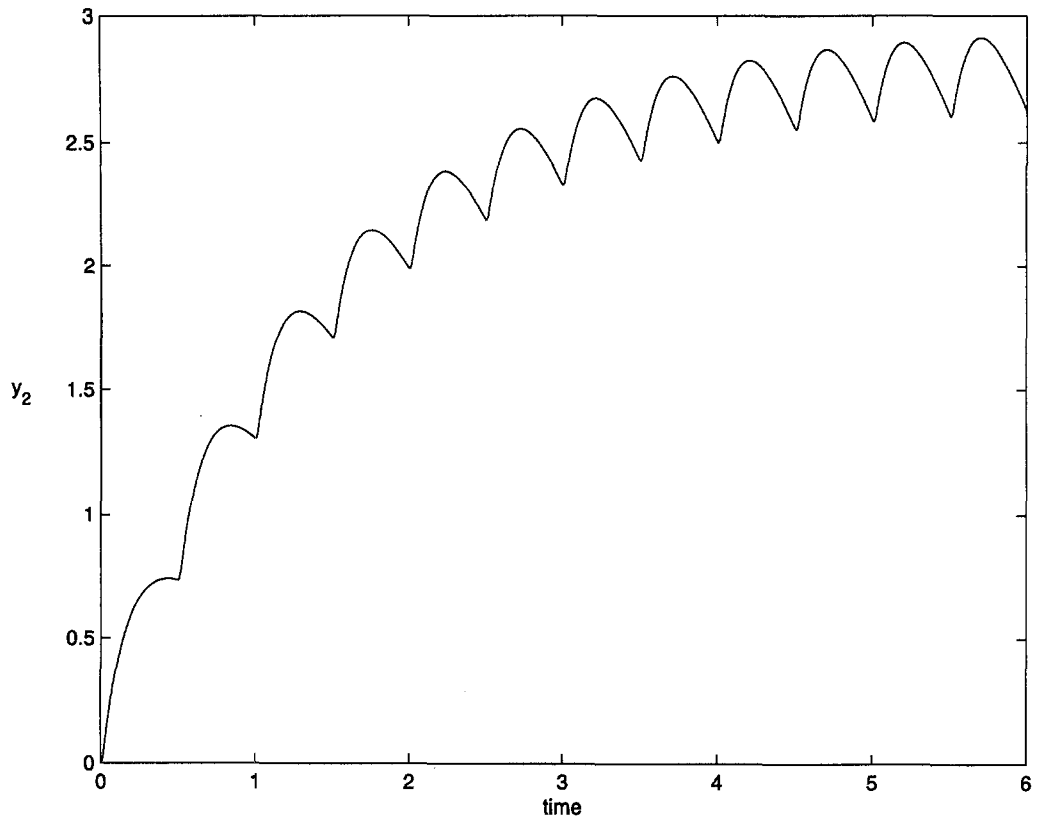

Figure 7. Correct $y_2(t)$ for Problem 2—original form.

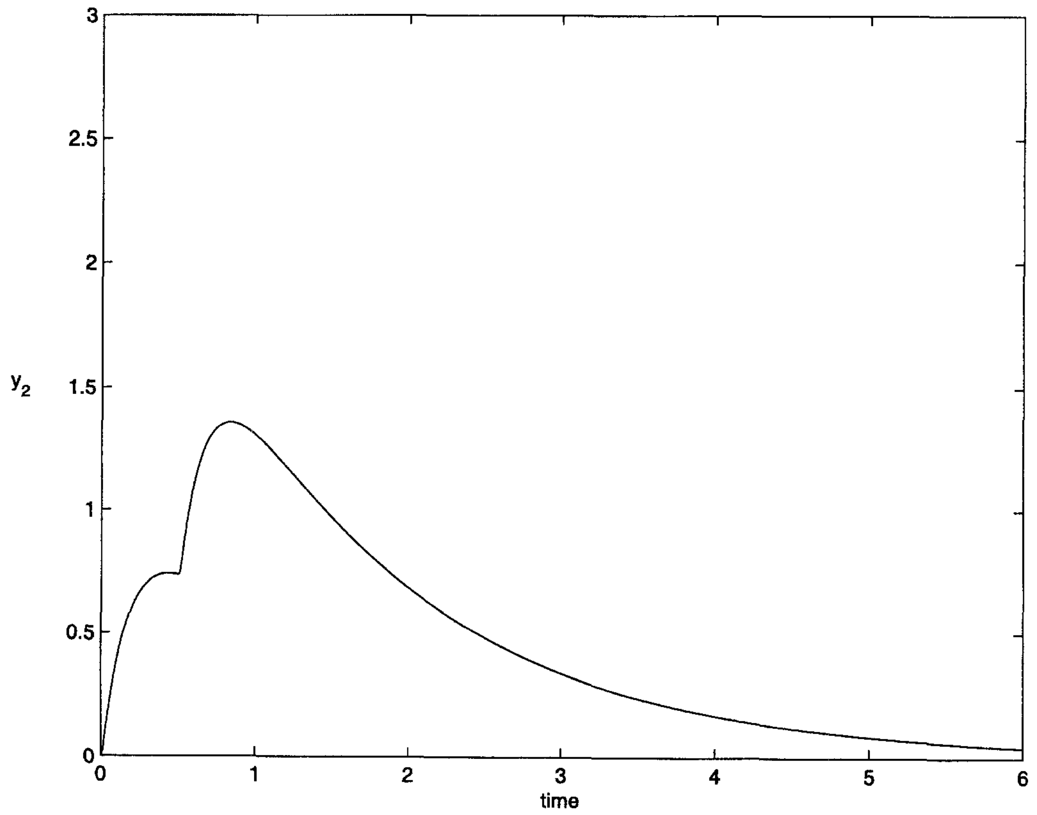

Figure 8. $y_2(t)$ from LSODE, with default $h_{\max}$, for Problem 2. Original form; many pulses missed.

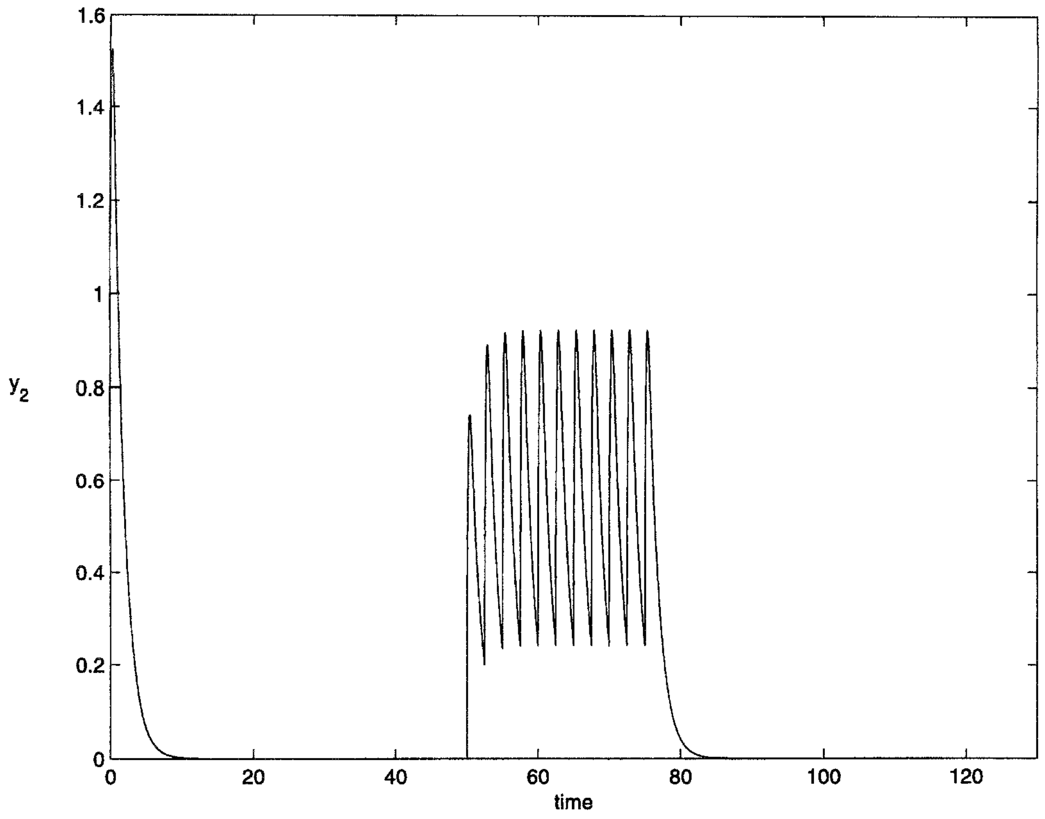

Figure 9. Correct $y_2(t)$ for Problem 2. Modified form.

relative to the length of the region of integration and the stepsizes selected by LSODE. We see that LSODE is able to find several of the pulses, without intervention, using its natural step selection sequence. Furthermore, when LSODE is employed within the PDODE software, where there is of course substantial intervention with its step selection process, even more of the pulses are "accidently" detected by LSODE. We note that whenever LSODE does detect a pulse and then readjusts its stepsize to successfully step through the pulse, the approximate solution it obtains will of course accurately reflect the presence of the pulse and the corresponding defect samples will not be large. In such cases PDODE will therefore not detect the pulse and will be unable to intervene to improve the efficiency of the computation.

We are therefore motivated to consider a modified version of this problem which is more consistent with the problem class we are studying in this paper, i.e., a problem for which LSODE, even with the stepsize intervention performed introduced by PDODE, does not accidentally hit any of the pulses. This version of the problem has the same ODE system, (7), but has a longer time interval, $t_{\text{final}} = 130.0$, and less frequent pulses. There are eleven pulses which start at times, $50.0, 52.5, 55.0, 57.5, \dots, 75.0$. We use the same pulse duration, $\delta = 1/48$, and the same pulse height, $P = 48.0$. The initial conditions are $y_1(0) = 1.0$, $y_2(0) = 1.0$. In this case the correct $y_2(t)$ is as shown in Figure 9. When we apply LSODE to the modified problem, with no intervention, all the pulses are missed and LSODE gives the result shown in Figure 10.

When PDODE is applied to solve the modified problem it is able to detect all the pulses. With some intervention, LSODE can be forced to detect all the pulses. Both codes return a solution as in Figure 9; the corresponding performance results are given in Table 3. These results show that for Case (b) LSODE uses more than 10 times the number of function evaluations and 20 times the number of Jacobian evaluations as PDODE, for Case (c) LSODE uses only about 60% of the number of function evaluations used by PDODE but about four times as many Jacobian evaluations as PDODE, and for Case (d) PDODE uses slightly fewer function evaluations and about 15% of the number of Jacobian evaluations as LSODE.

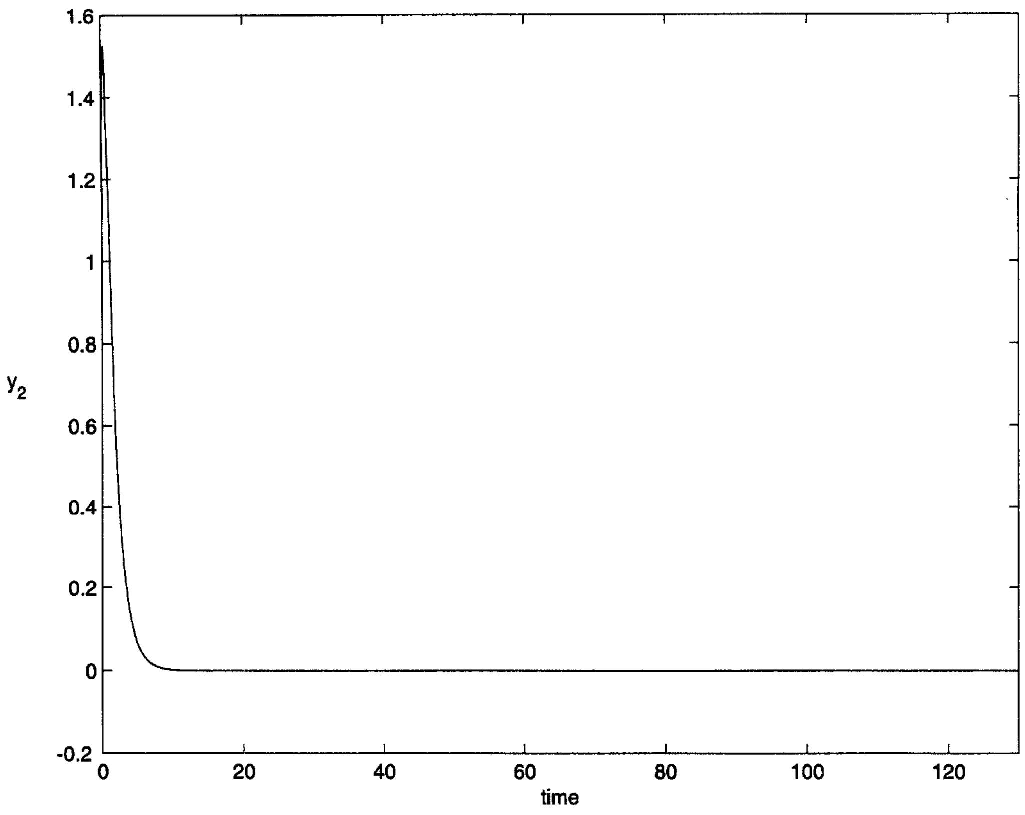

Figure 10. $y_2(t)$ from LSODE, with default $h_{\max}$, for Problem 2. Modified form; all pulses missed.

Table 3. Function calls and Jacobian calls for Problem 2. Modified form.

Method	Functions Calls	Jacobian Calls
Pulse missed		
LSODE, default $h_{\max}$	182	26
Case (b). Pulse detected		
LSODE, default $h_{\max}$	1927	380
PDODE, $s = 2$	156	19
Case (c). Pulse detected		
LSODE, $h_{\max} = 0.02 < \delta$	8361	935
PDODE, $s = 2$	14641	245
Case (d). Pulse detected		
LSODE, $h_{\max} = 0.005$	27669	1886
PDODE, $n_s = 20$	24427	246

5. SUMMARY, CONCLUSIONS, AND FUTURE WORK

In this paper, it has been shown through numerical experiments that standard IVODE software will normally miss a pulse arising in the right-hand side function of an ODE system. To address this difficulty, this paper describes the design and implementation of a high-level algorithm called PDODE that employs standard IVODE software (LSODE in this paper) to efficiently detect and treat such pulses. *A key advantage of the algorithm is that it is essentially independent of the underlying IVODE code*; thus, for example, LSODE could be replaced by a Runge-Kutta solver or by another multistep solver in a relatively straightforward fashion. *Furthermore, the algorithms described in this paper require no modifications to the LSODE package.*

PDODE has been shown to be reliable in finding and handling efficiently pulses for which either the start or the duration is not known, or when both are unknown. The most difficult case is when neither is known. In this case, LSODE has to have a severely restricted stepsize to ensure that it does not step over the pulse. PDODE typically uses fewer function evaluations than LSODE; more significantly, PDODE provides quite substantial savings in Jacobian evaluations, compared to LSODE. Since Jacobian evaluations are generally substantially more expensive than function evaluations (as discussed in the previous section) this advantage for PDODE translates into large savings in overall execution time. (It should be noted that LSODE makes no effort to save Jacobian evaluations for future use; an IVODE solver which was more conservative in its use of Jacobian evaluations would of course fare better than LSODE in a comparison with PDODE. However, it is clear that the severe stepsize restriction that must be placed upon any IVODE solver in order for it to detect a pulse will imply that the solver will be more expensive than PDODE.)

The experiments presented in this paper have been run at fairly sharp relative/absolute tolerances with the hope that this would improve the ability of LSODE to detect the pulse without any interference in its step selection. As we have seen, even the use of sharp tolerances does not help; experiments with coarse tolerances will of course reduce further the likelihood of LSODE finding a pulse.

Possible topics for future work include generalizing the algorithms to handle state-dependent pulses and to handle overlapping pulses in different ODE components, as well as investigating the performance of PDODE with IVODE solvers other than LSODE.

REFERENCES

1. A.C. Hindmarsh, ODEPACK, a systematized collection of ODE solvers, In *Scientific Computing*, (Edited by R.S. Steplman *et al.*), pp. 55–64, North-Holland, Amsterdam, (1983); available at `http://www.netlib.org`.
2. A.C. Hindmarsh, Source for module LSODE from package ODEPACK, In *NIST Guide to Available Math Software*, (1987).
3. C.W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, NJ, (1971).
4. C.W. Gear and O. Osterby, Solving ordinary differential equations with discontinuities, *ACM Trans. Math. Softw.* **10**, 23–44, (1984).
5. L.F. Shampine, *Numerical Solution of Ordinary Differential Equations*, Chapman & Hall, New York, (1994).
6. W.H. Enright, K.R. Jackson, S.P. Nørsett and P.G. Thomsen, Effective solution of discontinuous IVPs using a Runge-Kutta formula pair with interpolants, *Appl. Math. Comp.* **27**, 313–355, (1988).
7. W.H. Enright, Continuous numerical methods for ODEs with defect control, *J. Comp. Appl. Math.* **125**, 159–170, (2000).
8. W.H. Enright, T.E. Hull and B. Lindberg, Comparing numerical methods for stiff systems of ordinary differential equations, *BIT* **15**, 10–48, (1975).
9. C.J. Clements, J.C. Clements and B.M. Horáček, On the formation of scroll waves in an anisotropic ventricular myocardium, In *Differential Equations with Applications to Biology*, (Edited by S. Ruan *et al.*), pp. 97–107, Amer. Math. Soc., Providence, RI, (1999).
10. C.-H. Luo and Y. Rudy, A model of the ventricular cardiac action potential: Depolarization, repolarization, and their interaction, *Circ. Res.* **68**, 1501–1526, (1991).
11. American Heart Association, `http://www.americanheart.org`.
12. J.S. Levine and K.R. Miller, *Biology: Discovering Life*, D.C. Heath and Company, Lexington, MA, (1991).
13. R.D. Skeel, Equivalent forms of multistep formulas, *Math. Comp.* **33**, 1229–1250, (1979).