European Society of Computational Methods in Sciences and Engineering (ESCMSE)

Journal of Numerical Analysis, Industrial and Applied Mathematics (JNAIAM)
vol. 3, no. 1-2, 2008, pp. 139-149
ISSN 1790–8140

Solving ODEs and DDEs with Impulses[1]

S.P. Corwin
Department of Mathematics & Statistics
Radford University
Radford, Virginia USA
scorwin@radford.edu

S. Thompson[2]
Department of Mathematics & Statistics
Radford University
Radford, Virginia USA
thompson@radford.edu

S.M. White
Genetics, Ecology & Evolution Research Group
Institute of Integrative and Comparative Biology
Faculty of Biological Sciences, L C Miall Building
University of Leeds, Leeds, LS2 9JT
S.M.White@leeds.ac.uk

Received 5 August, 2007; accepted in revised form 20 August, 2007

Abstract: This paper deals with the solution of systems of ordinary differential equations (ODEs) and systems of delay differential equations (DDEs) in which solution impulses are applied at specific times. Such systems include a wide range of biologically motivated examples. Event functions are used to locate the times at which impulses are applied. Systems of ODEs and especially DDEs with impulses are often difficult to solve accurately, but they can be solved quite efficiently using the event finders available in several capable solvers. The manner in which this may be accomplished is illustrated using the Matlab ODE and DDE solvers as well as the Fortran 90 ODE solver `vode_f90` and the Fortran 90 DDE solver `dde_solver`. Systems with both time-dependent and with state-dependent impulses are included. The use of a GUI for `vode_f90` is included to illustrate the need for such GUIs for popular solvers.

Keywords: Delay differential equations (DDEs); Event location; Ordinary differential equations (ODEs); Problem solving environment (PSE); State-dependent impulses; Time-dependent impulses

Mathematics Subject Classification: AMS-MOS Subject Classifications: 65.0, 65.3, 65.61, 65L06, 65Q05

[1]Published electronically March 31, 2008
[2]Corresponding author.

1 Introduction

Systems of ODEs of the form

$$y' = f(t, y), \qquad t_0 \le t \le t_f \tag{1}$$

with initial values

$$y(t_0) = y_0 \tag{2}$$

as well as systems of DDES of the form

$$y'(t) = f(t, y(t), y(\beta_1), \ldots, y(\beta_k)), \qquad t_0 \le t \le t_f \tag{3}$$

with initial history

$$y(t) = h(t), \qquad t \le t_0 \tag{4}$$

and delays $\beta_i = \beta_i(t, y(t)) \le t$ are sometimes subject to impulses. In the most common case, time-dependent impulses are used. Time-dependent impulses can be solved by hard coding the impulse, solving the problem on consecutive time intervals, applying the impulse to alter the solution at the end of each interval, and joining the output together to obtain the desired solution. For examples of studies involving time-dependent impulses, refer to [1], [4], [8], [13], and [14], and the references included in these studies. Refer to [5] and [9] for theoretical discussions of systems with impulses. More generally, impulses may be triggered by a state-dependent condition. This situation is more difficult because impulse times are not known in advance. We mention that time-dependent problems can sometimes be recast as related state-dependent problems. For example, [1] considers a time-dependent periodic pulse to reduce a population below a threshold. However, this problem could also be written as a state-dependent problem to calculate the impulse times in the manner described below for Example 3.

We will describe an event location based procedure which may be used to handle state-dependent impulses. Examples will be given to demonstrate that the algorithm is particularly suited to a problem solving environment (PSE) such as Matlab [6]. We will also describe how to use the procedure in Fortran 90 in a manner approaching the ease and user convenience of a PSE such as Matlab. Systems with time-dependent impulses can be solved by any code which contains provisions to return the solution at specified times and which allows the solution to be altered and the integration to be restarted at these times. However, in order to solve systems with state-dependent impulses, it is necessary that a code has provisions for locating the state-dependent impulses. If a code contains provisions for rootfinding or event location, it can be used to handle state-dependent impulses provided the relevant rootfinding is performed reliably and efficiently. Event location is accomplished by finding the times corresponding to a set of event functions

$$0 = g_i(t, y) \tag{5}$$

simultaneously with the integration of the system.

2 Using Event Functions to Apply Impulses

On the surface the numerical difficulties in solving problems with impulses look somewhat akin to those in solving ODEs and DDEs with discontinuities. Good ODE and DDE solvers are usually capable of handling derivative discontinuities without help from the user. However, since impulses must be applied at *specific* impulse times, it is not possible to simply ignore the impulse times and hope a solver will manage to solve the problem on its own. Discontinuities in the DDE history function caused by applying impulses must also be handled appropriately. In order to handle both systems of ODEs and DDEs with either time-dependent or state-dependent impulses, we will adopt an approach based on event location. Generally, both the efficiency and the reliability of a solver can be improved for problems with discontinuities by using event functions and integration switches that permit the solver to locate discontinuity times in a smooth fashion and allow appropriate changes in the problem to be made before continuing the integration.

A few comments are in order regarding the particular choices of solvers used in this paper. For the problems we consider a solver needs provisions for event location as well as the ability to continue an integration with a change of solution when events occur. The MATLAB ODE and DDE solvers, the Fortran 90 ODE solver `vode_f90` [3], and the Fortran 90 DDE solver `dde_solver` [15] contain effective event location provisions. We have used these solvers with very satisfactory results both for systems with time-dependent as well as systems with state-dependent impulses. Although we present ODE results for the Matlab solver `ode23`, we mention that similar results were easily obtained using `ode45`, and `ode15s` due to the common design of the solvers. The MATLAB DDE solvers `dde23` [12] and `ddesd` [10], and the Fortran 90 `dde_solver` and its f77 predecessors (not considered here) are the only readily available solvers we are aware of which contain event location provisions necessary for systems of DDEs with state-dependent impulses.

When an event time is located, the appropriate impulse can be applied − thus changing the solution. The integration can then be restarted at this time with the new solution. Although each of the MATLAB and Fortran 90 solvers has provisions for using event functions, the solvers use rather different approaches for applying impulses. The MATLAB solvers return control to the calling script when terminal events are encountered. The calling script then applies the impulse and the integration is restarted using the previous solution to obtain the solution history for the new integration. The function `predprey` given in Section 3.3 illustrates the manner in which this is accomplished using `dde23`. `dde_solver` allows the user to provide a `CHANGE` subroutine which is called at event times. The impulses are applied in this subroutine when it is called. The integration is then continued from the impulse time after the solution has been changed appropriately. Examples of the use of `CHANGE` functions may be found in [11]. `vode_f90` returns control to the calling program which can apply the impulse and restart the integration at the impulse time. We say more about the use of `vode_f90` in Section 4. Example programs for each of the problems and solvers considered in this paper are available from the website `http://www.radford.edu/~thompson/impulses/`.

For problems with time-dependent impulses occurring at known times $t = t_1 < t_2 < \cdots < t_n$, a single event function $g = t - T_e$ may be used. Initially, $T_e = t_1$. Each time an event time $T_e = t_i$ is located, T_e is changed to t_{i+1}. With this approach the solver is allowed to step across t_i smoothly so it is not forced to deal with a derivative discontinuity while locating T_e. The integration then proceeds from T_e using the new solution. The algorithms used in the solvers mentioned in this paper ensure that each T_e is located since they find the left-most zero of the event function. They also ensure that the time-dependent events are located *exactly* which is a necessity for problems with impulses.

For problems with state-dependent impulse times, an appropriate state dependent event function may be used. For example, if an impulse is to be applied when a solution component y_k is equal to a prescribed value Y, an event function $g_k(y_k) = y_k - Y$ is used. Event times are located as accurately as possible depending on the accuracy of the solution components. This capability is especially important for problems with impulses; it effectively precludes the use of a solver which requires the user to supply hard-to-determine rootfinding tolerances. Each of the solvers in this paper allows the direction of the zero crossing of an event function to be specified. This feature may be used to locate event times more efficiently and to avoid numerical difficulties that might otherwise arise if "false events" are located due to numerical inaccuracies in the impulse times T_e and corresponding solutions $y\,(T_e)$. Since each of the solvers allows a set of event functions to be used, the solvers may be used for problems with multiple impulse conditions.

3 Examples of Systems with Applied Impulses

3.1 Example 1: Time-Dependent DDE with Impulses

Time-dependent impulses arise naturally in many biological and physiological systems, including ones from delayed cellular neural networks with impulsive effects. We will illustrate the use of time-dependent impulses using the following example from [16]. Note that the delays for this problem are sometimes very small and in fact vanish periodically during the integration, making this a relatively difficult problem.

$$\begin{aligned}
y_1' &= -6y_1(t) + \sin(2t)f(y_1(t)) + \cos(3t)f(y_2(t)) \\
&\quad + \sin(3t)f\left(y_1\left(t - \frac{1+\cos(t)}{2}\right)\right) + \sin(t)f\left(y_2\left(t - \frac{1+\sin(t)}{2}\right)\right) \\
&\quad + 4\sin(t) \\
y_2' &= -7y_2(t) + \frac{\cos(t)}{3}f(y_1(t)) + \frac{\cos(2t)}{2}f(y_2(t)) \\
&\quad + \cos(t)f\left(y_1\left(t - \frac{1+\cos(t)}{2}\right)\right) + \cos(2t)f\left(y_2\left(t - \frac{1+\sin(t)}{2}\right)\right) \\
&\quad + 2\cos(t)
\end{aligned}$$

where

$$f(x) = \frac{|x+1| - |x-1|}{2}. \tag{6}$$

The initial history is given by $y_1(t) = -0.5$ and $y_2(t) = 0.5$. At each impulse time $t_k = 2k$ a time-dependent solution impulse is applied with $y_1(t_k)$ being replaced by $1.2y_1(t_k)$ and $y_2(t_k)$ being replaced by $1.3y_2(t_k)$. An event function $g(t) = t - T_e$ is used where $T_e = 2$ initially and $T_e = 2(k+1)$ once $T = 2k$ is located. Fig. 1 shows the phase plane for the solution computed with impulses using `dde_solver`. Similar results were obtained using `dde23` and `ddesd`.

3.2 Example 2: State-Dependent ODE with Impulses

In order to illustrate the solution procedure as simply as possible when state-dependent impulses are applied, we begin with the following simple predator-prey ODE system.

$$\begin{aligned}
y_1' &= \alpha y_1(t) + \beta y_1(t)y_2(t) \\
y_2' &= \gamma y_2(t) + \delta y_1(t)y_2(t)
\end{aligned}$$

where $\alpha = 0.25, \beta = -0.01, \gamma = -1.0, \delta = 0.01$. The initial history is given by $y_1(t) = 80$ and $y_2(t) = 30$. We modify this problem in the following manner. At each time T_e at which the size of the prey population $y_1(t)$ reaches a prescribed value Y we apply a solution impulse, changing $y_1(T_e)$ to $0.8y_1(T_e)$. A state-dependent event function $g(y_1) = y_1 - Y$ is used to accomplish this.

Four impulse times, $5.9, 8.5, 11.3$, and 14.8 are located for this value of Y. At each impulse time the prey population is reduced. After the last impulse time is located the correct periodic solution is obtained. Fig. 2 shows the phase plane for the solution computed using `dvode_f90` for $Y = 114$. Similar results were obtained using the MATLAB ODE solver `ode15s` (as well as the DDE solvers considered although, of course, one would not normally use a DDE solver to solve an ODE).

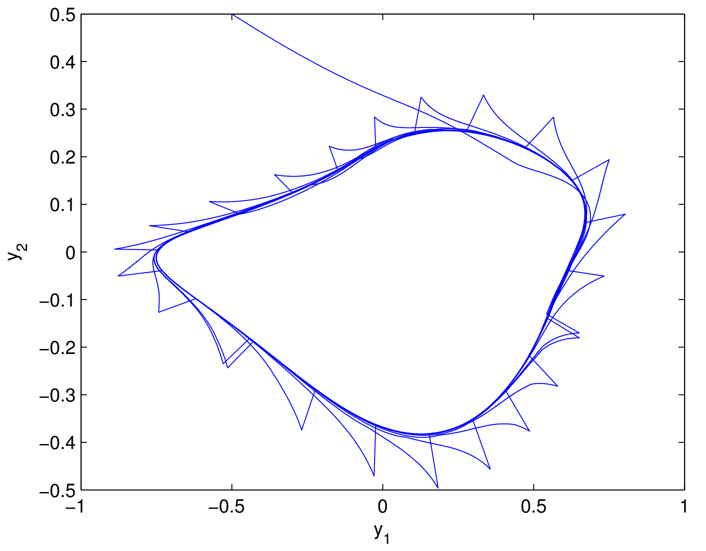


Figure 1: Neural Network DDE with Time-Dependent Impulses.

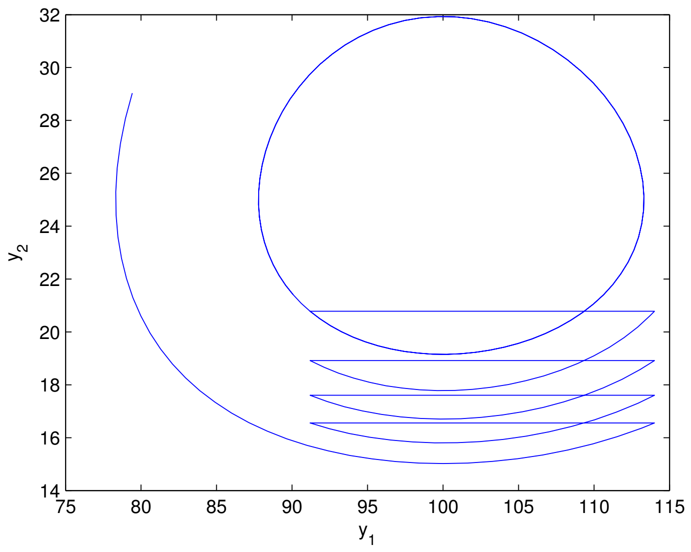


Figure 2: Predator-Prey ODE with State-Dependent Impulses.

3.3 Example 3: Impulse Harvesting Predator-Prey Models

This section considers more involved predator-prey systems with state-dependent impulses. The examples in question belong to a class of state-dependent impulse harvesting predator-prey models. Solving these problems demonstrates that the solvers considered are capable of successfully handling the numerical difficulties that arise when impulses are present. The motivation for the following model comes from fish conservation biology and harvesting where we are interested in the scenario in which a species of fish is harvested for human consumption. However, the species of interest is preyed upon by a predator species for which the prey species is its only food resource. Moreover, in the interest of conservation, we are interested in harvesting strategies that promote both stable fish harvest quotas and fish conservation of both predator and prey species.

Let $N(t)$ and $H(t)$ be the density of prey and predators respectively at time t. We assume that prey are harvested via a state-dependent impulse, that is, when prey density reaches a threshold level, Y, a proportion, $p \in (0, 1)$, of the prey population are harvested. The basic model is then given by the following system of coupled DDEs.

$$\frac{\mathrm{d}N}{\mathrm{d}t}(t) = rN(t)\left(1 - \frac{N(t-\tau_1)}{K}\right) - aN(t)H(t) \tag{7a}$$

$$\frac{\mathrm{d}H}{\mathrm{d}t}(t) = abN(t-\tau_2)H(t-\tau_2) - dH(t) \tag{7b}$$

for $N(t) \neq Y$, and

$$\Delta N(t) := N(t^+) - N(t^-) = -pN(t) \tag{7c}$$

$$\Delta H(t) := H(t^+) - H(t^-) = 0 \tag{7d}$$

for $N(t) = Y$.

Here we assume that in the absence of predation and harvesting the prey grow logistically with intrinsic growth rate r and carrying capacity K. However, the prey's regulatory effect depends on the population at an earlier time, $t - \tau_1$, which takes into account the development period, thus giving the classic delayed logistic equation. Prey are preyed upon at a per capita rate $aH(t)$ which is converted into predator biomass with conversion rate b. However, predator development time has period τ_2 which must be taken into account. We also assume that predators have a background mortality rate d.

This system has three equilibrium solutions with the one of interest (species coexistence) given by

$$(N_0, H_0) = \left(\frac{d}{ab}, \frac{r}{a}\left(1 - \frac{d}{abK}\right)\right). \tag{8}$$

The following code illustrates the manner in which `dde23` may be used to solve a problem of this type. (Note: For reasons of space, we have not included various plotting commands used in the program.) The program has several noteworthy features. It successively solves (7) between impulse times. Such times occur when $y_1(t) = Y$; they are located using the *Events* option. The impulse is then applied and the new solution is supplied as the initial solution for the next call using the *InitialY* option. This is crucial for problems with impulses since the previous history would be used to obtain the initial solution otherwise.

```
function predprey
   r = 1; K = 1; a = 2; b = 1; d = 1; impcount = 0;
   Y = 1.2 * (d/(a*b)); p = 0.3; tstar = 0; tfinal = 40;
   while true
      if (impcount ==0 )
         options = ddeset('Events',@events,'AbsTol',1e-9,...
                          'RelTol',1e-9);
         sol = dde23(@ddes, [0.0 0.0], ....
                    [0.2; 0.1], [tstar,tfinal], options);
      else
         % Specify the new solution at impulse times...
         options = ddeset(options, 'InitialY',...
                         [(1-p)*sol.y(1,end); sol.y(2,end)]);
         sol = dde23(@ddes, [0.0 0.0], ....
                    sol, [tstar,tfinal], options);
      end
      tstar = sol.x(end);
      impcount = impcount + 1;
      if (tstar >= tfinal)
         break;
      end
   end
   %===Nested functions==============================
   % Evaluate the DDES...
   function dydt = ddes(t,y,Z)
      dydt = [r*y(1)*(1-(Z(1,1)/K))-a*y(1)*y(2); ...
             a*b*Z(1,2)*Z(2,2)-d*y(2)];
   end
   % Evaluate the event function residuals...
   function [value,isterminal,direction] = events(t,y,Z)
      value = y(1)-Y;
      isterminal = 1;
      direction  = 1;
   end
   %=================================================
end
```

Using the above code, we demonstrate that `dde23` is successful in solving the simpler ODE case, $(\tau_1, \tau_2) = (0, 0)$, when the system is solved without impulses (Fig. 3 (a)) and with impulses (Fig. 3 (b)). Similar results were obtained using `ddesd` and `dde_solver`.

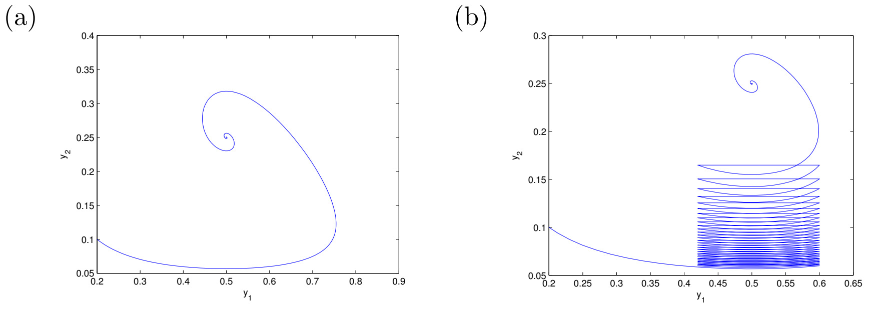


Figure 3: Predator-Prey Harvesting Model with State-Dependent Impulses ($\tau_1 = 0, \tau_2 = 0$).

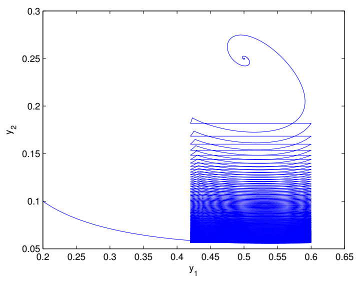
In the absence of harvesting the predator-prey ODE system oscillates into the coexistence equilibrium (8). However, when the system is subject to prey harvesting the long-term dynamics significantly changes depending on the harvesting threshold, Y. If $Y < N_0$, then $N(t) < N_0$ for all

t. It then follows that

$$\frac{dH}{dt} = abN(t)H(t) - dH(t) < (abN_0 - d)\, H(t) = 0 \tag{9}$$

since $N_0 = \dfrac{d}{ab}$. Because $dH/dt < 0$ the predators go extinct. Conversely, $\dfrac{dH}{dt} > 0$ if $Y > N_0$. In this case the predator population will always grow while the prey are subject to the harvesting (Fig. 3 (b)). At first there is insufficient prey to sustain the predators and so the prey increase and the predators decrease in density; but as the prey increase this allows the predators to increase until eventually the prey hit the threshold and are harvested. The solution now enters the harvesting strategy phase where the predators always increase (but may suffer a decrease after harvesting depending on the proportion of harvest, p). Eventually the predator population becomes so large that it stops the prey from reaching harvesting density, after which the system goes to equilibrium. Therefore, the ODE model predicts that either no sustainable harvesting strategy exists, or that sustained harvesting of prey is achievable but at the cost of causing predator extinction.

For the DDE case, `dde23` successfully solves the predator-prey model. For some delay parameter choices (*e.g.*, those given in the displayed `dde23` program) the solution eventually converges to the equilibrium solution of (7) (Fig. 4 (a)). Some other parameters choices (*e.g.*, $(\tau_1, \tau_2) = (4, 0.001)$) lead to solutions which are periodic for a very long time and do not converge to the equilibrium as the prey level zigzigs back and forth horizontally, indicating a successful harvesting strategy which conserves the predator population (Fig. 4 (b)).

(a)


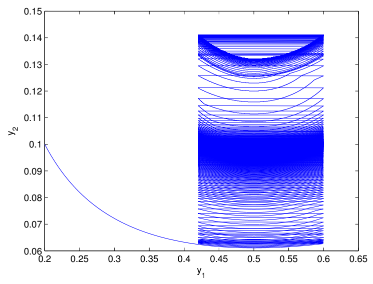
(b)


Figure 4: Predator-Prey Harvesting Model with State-Dependent Impulses. In (a) $\tau_1 = 0.2, \tau_2 = 0.2$ and in (b) $\tau_1 = 4, \tau_2 = 0.001$.

It is clear that the feedback between the time delays and harvesting can significantly influence the long-term dynamics of the model. For example, assume for the sake of simplicity that $\tau_2 = 0$. In this case, it is well-known for the delayed logistic equation that the system becomes unstable if τ_1 becomes sufficiently large [7]. However, when we couple this with the predator dynamics, the predator acts as a stabilizing feedback and actually manages to stabilize the system whereas the prey would oscillate otherwise. We demonstrate this in Fig. 5 where we vary the delays τ_1 and τ_2, showing the various periodic behavior of solutions (white indicates equilibria and black indicates long periodic behavior).

We have used `dde23`, `ddesd`, and `dde_solver` to solve Example 3 for a wide range of parameters and delays to determine that they correctly solve the problem. We note that solving the problem in the presence of impulses is a nontrivial task. In particular, numerous impulse times must be located in order to apply the relevant impulses. It is also necessary in some cases to integrate over very long intervals in order to obtain the correct limiting behavior of the solution. In addition, small step sizes are necessary at event times in order to handle the discontinuities specified by the

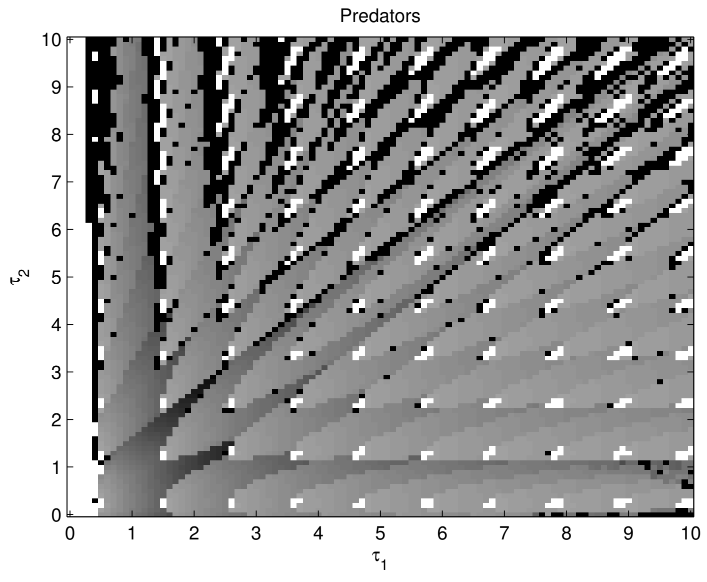


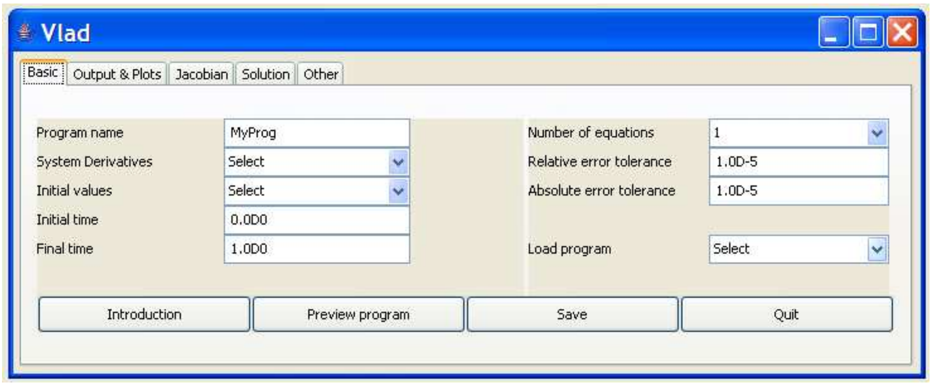
Figure 5: Behavior of Solutions for the Predator-Prey Harvesting Model.

InitialY option. Depending on the dynamics for particular choices of the problem parameters, very many (several thousand in some cases) closely spaced impulse times may occur. Although each of the solvers located the relevant impulse times, these systems pose worthy challenges for the solvers since an enormous amount of root finding is required.

4 A Step in the Right Direction: GUIs for Fortran Solvers

The solution procedure we have described is ideally suited to a PSE such as Matlab which contains solvers that have the ability to return control to the calling script when a terminal event occurs. Since the Fortran 90 DDE solver `dde_solver` was developed with the design of the Matlab program dde23 in mind, the necessary calculations are only slightly more involved than the corresponding calculations for the Matlab ODE and DDE solvers. In particular, `dde_solver` allows the user to supply a `CHANGE` function which is called whenever an event is located. However, a straightforward solution for a Fortran solver such as `vode_f90`, a Fortran 90 extension of the well-known `vode.f` Fortran 77 solver [2], becomes a bit awkward. After calling the solver, the solution must be processed manually to determine the type of return. If an impulse time is located, the solution must be altered and the integration restarted. In addition, the solution of complicated problems such as those described in this paper requires user provided subroutines to evaluate the derivatives, to evaluate the residuals of the event functions, and to process impulse times and apply solution impulses.

To facilitate the use of `vode_f90`, we developed an experimental Java Graphical User Interface (GUI), `Vlad`, which prompts the user for problem information and writes the necessary Fortran 90 subroutines and calling program. Fig. 6 depicts the top level `Vlad` menu screen. (All `vode_f90` options are available in appropriate submenus.) `Vlad` allows `vode_f90` to be used in a manner similar to that of the Matlab ODE solvers. It was designed in a fashion similar to ones previously developed for the Fortran 77 predecessors of `dde_solver`. Particularly noteworthy is the fact that the GUI allows the user to supply a `CHANGE` subroutine which resembles that used by `dde_solver`. Although use of `Vlad` requires a knowledge of Fortran 90, it is possible to easily organize the calculations and to simplify the usage of `vode_f90` considerably. This suggests that the development of similar GUIs for other available Fortran ODE and DDE solvers will increase the ease of use of the solvers.



Program name: MyProg
System Derivatives: Select
Initial values: Select
Initial time: 0.0D0
Final time: 1.0D0
Number of equations: 1
Relative error tolerance: 1.0D-5
Absolute error tolerance: 1.0D-5
Load program: Select

Figure 6: Vlad GUI for `vode_f90`.

5 Summary

This paper described the manner in which ODE and DDE solvers with root finding capabilities can be used to solve systems of ODEs and DDEs in which either time-dependent or state-dependent solution impulses must be applied at different times. The manner in which this can be done using event functions was illustrated using different solvers to solve several biologically motivated examples. Results were given for several rather different ODE and DDE solvers. The results show that systems with either time-dependent or state-dependent impulses can be solved satisfactorily provided the solver has provisions for effective event location. Experience with one of the solvers considered, `vode_f90`, illustrated that usage of popular ODE and DDE solvers can be simplified by the use of GUIs intended to perform many common tasks thereby relieving the user of the necessity to do so.

Acknowledgment

S.M. White is supported by NERC grant NE/B503276/1 awarded to Steve Sait (University of Leeds) and Pej Rohani (University of Georgia).

References

[1] Z. Agur, L. Cojocaru, G. Mazor, R.M. Anderson, and Y.L. Danon, Pulse Mass Measles Vaccination Across Age Cohorts, *P. Natl. Acad. Sci. USA*, **90**, 11698-11702 (1993).

[2] P.N. Brown, G.D. Byrne, and A.C. Hindmarsh, VODE: A Variable-coefficient ODE Solver, *SIAM J. Sci. Stat. Comput.*, **10**, 1038-1051 (1989).

[3] G.D. Byrne and S. Thompson, `vode_f90` ODE Solver Web Support Page, `http://www.radford.edu/~thompson/vodef90web/index.html`.

[4] M. Choisy, J.F. Guegan, P. Rohani, *Dynamics of Infectious Diseases and Pulse Vaccination: Teasing Apart the Embedded Resonance Effects*, *Phys D.*, **22**, 26-35 (2006).

[5] V. Lakshmikantham, D.D. Bainov, and P.S. Simeonov, *Theory of Impulsive Differential Equations*, World Scientic, Singapore, 1989.

[6] Matlab 7, The MathWorks, Inc., 3 Apple Hill Dr., Natick, MA 01760, 2006.

[7] R.M. May, Time-Delay versus Stability in Population Models with Two and Three Trophic Levels, *Ecology*, **54**, 315-325 (1973).

[8] A. d'Onofrio, A General Framework for Modeling Tumor-Immune System Competition and Immunotherapy: Mathematical Analysis and Biomedical Inferences, *Phys D.*, **208**, 220-235 (2005).

[9] A.M. Samoilenko and N.A. Perestyuk, *Impulsive Differential Equations*, World Scientific, Singapore, 1995.

[10] L.F. Shampine, Solving ODEs and DDEs with Residual Control, *Appl. Numer. Math.*, **52**, 113-127 (2005).

[11] L.F. Shampine and S. Thompson, `dde_solver` Web Support Page, `http://www.radford.edu/~thompson/ffddes/index.html`.

[12] L.F. Shampine and S. Thompson, Solving DDEs in Matlab, *Appl. Numer. Math.*, **37**, 441-458 (2001).

[13] X. Song, Z. Xiang, The Prey-Dependent Consumption Two-Prey One-Predator Models with Stage Structure for the Predator and Impulsive Effects, *J. Theor. Biol.*, **242**, 683-698 (2006).

[14] S. Tang and L. Chen, Density-Dependent Birth Rate, Birth Pulses and Their Population Dynamic Consequences, *J. Math. Biol.*, **44**, 185-199 (2002).

[15] S. Thompson and L.F. Shampine, A Friendly Fortran DDE Solver, *Appl. Numer. Math.*, **56**, 503-516 (2006).

[16] Y. Yongqing and J. Cao, Stability and Periodicity in Delayed Cellular Neural Networks with Impulsive Effects, *Nonlinear Anal. Real World Appl.*, **8**, 362-374 (2007).