DDVERK90: A User-Friendly Implementation of an Effective DDE Solver

by

Hossein Zivaripiran

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Abstract

DDVERK90: A User-Friendly Implementation of an Effective DDE Solver

Hossein Zivaripiran

Master of Science

Graduate Department of Computer Science

University of Toronto

2005

Delay differential equations(DDEs) provide powerful and rich mathematical models that have been proven to be useful in simulating some real life problems. The current publicly available DDE solvers are either very formidable to use (especially for new users) or designed to support only selected classes of problems.

In this thesis we have developed a new Fortran 90 DDE solver DDVERK90 that aims to combine the effectiveness of the Fortran 77 DDE solver DDVERK and the user-friendliness of the MATLAB ODE solvers. We have added the capability of embedded event location to facilitate many tasks that arise in applications. We have also introduced a new approach for classifying DDE problems that helps users to write driver programs to solve their problem very quickly. They do this by comparing their problem to other previously solved example problems in the same class and using the corresponding 'template' driver as a guide.

Acknowledgements

I would especially like to thank my supervisor, Professor Wayne Enright, for the guidance and encouragement he provided while I was working on this thesis. I would also like to thank Dr. Tom Fairgrieve for his careful reading of the thesis and for providing valuable comments.

Contents

Chapter 1

Introduction

1.1 Motivation

Interest in using delay differential equations (DDEs) to model real life problems has increased significantly in recent years. Although there are many DDE solvers that can effectively solve DDEs, the intrinsic difficulty of DDEs along with the weaknesses of early programming languages make the use of these solvers formidable, especially for new users. Furthermore special features such as event location has proved to be useful in the investigation of many models that are described by DDEs. This motivated us to use the capabilities of Fortran 90 to create a solver with a more user-friendly interface and the capability of embedded event location.

1.2 Background

1.2.1 Assumptions and Definitions

A retarded delay differential equation (RDE) is a system of differential equations defined by

$$\begin{aligned} y'(t) &= f(t, y(t), y(t-\sigma_1(t,y(t))), y(t-\sigma_2(t,y(t))), \\ &\qquad \cdots, y(t-\sigma_\nu(t,y(t))), \;\text{ for }\; t_0 \le t \le t_F, \qquad (1.1) \\ y(t) &= \phi(t), \;\text{ for }\; t \le t_0, \end{aligned}$$

where y, f, and ϕ are $\mathcal{N}$-vector functions and $\sigma_i, i = 1, 2, \cdots, \nu$ are scalar functions. A neutral delay differential equation (NDE) is a system of differential equations defined by

$$\begin{aligned} y'(t) &= f(t, y(t), y(t-\sigma_1(t,y(t))), y(t-\sigma_2(t,y(t))), \\ &\qquad \cdots, y(t-\sigma_\nu(t,y(t))), y'(t-\sigma_{\nu+1}(t,y(t))), \\ &\qquad \cdots, y'(t-\sigma_{\nu+\omega}(t,y(t))) \;\text{ for }\; t_0 \le t \le t_F, \qquad (1.2) \\ y(t) &= \phi(t), \;\; y'(t) = \phi'(t), \;\text{ for }\; t \le t_0, \end{aligned}$$

where y, f, and ϕ are $\mathcal{N}$-vector functions and $\sigma_i, i = 1, 2, \cdots, \nu+\omega$ are scalar functions. The term DDE refers to both an RDE and an NDE.

We call each of the functions $\sigma_i(t, y(t))$ a *delay*, each of the arguments $t - \sigma_i(t, y(t))$ a *delay argument*, a value of the solution delay term $y(t - \sigma_i(t, y(t)))$ the *(solution) delay value* and a value of the derivative delay term $y'(t - \sigma_i(t, y(t)))$ the *derivative delay value.* If a delay is a constant, it is called a *constant delay.* If it is a function of only time, then it is called a *time dependent delay.* If a delay is a function of the solution $y(t)$, it is called a *state dependent delay.* A delay argument that passes the current time, *i.e.* $t - \sigma_i(t, y(t)) > t$, is called an *advanced delay.* We call $\phi(t)$ the *history* function.

We define a *local solution* of the RDE (1.1) (associated with the $(n+1)^{st}$ step) as the solution of

$$\begin{aligned} y_n'(t) &= f(t, y_n(t), y_n(t-\sigma_1(t,y_n(t))), y_n(t-\sigma_2(t,y_n(t))), \\ &\qquad \cdots, y_n(t-\sigma_\nu(t,y_n(t))), \;\text{ for }\; t_n \le t \le t_F, \qquad (1.3) \\ y_n(t) &= z(t), \;\text{ for }\; t \le t_n, \qquad n = 0, 1, 2, \cdots \end{aligned}$$

where $z(t)$ is a known continuous approximation to $y(t)$ defined on $(t_0, t_n]$, and $z(t) = \phi(t)$

for $t \leq t_0$. A local solution of the NDE (1.2) is defined similarly as the solution of

$$
\begin{aligned}
y_n'(t) &= f(t, y_n(t), y_n(t - \sigma_1(t, y_n(t))), y_n(t - \sigma_2(t, y_n(t))), \\
&\quad \cdots, y_n(t - \sigma_\nu(t, y_n(t))), y_n'(t - \sigma_{\nu+1}(t, y_n(t))), \\
&\quad \cdots, y_n'(t - \sigma_{\nu+\omega}(t, y_n(t))) \;\text{ for }\; t_n \leq t \leq t_F, \qquad (1.4) \\
y_n(t) &= z(t), \;\; y_n'(t) = z'(t), \;\text{ for }\; t \leq t_n, \qquad n = 0, 1, 2, \cdots
\end{aligned}
$$

Let $z(t)$ be a continuous approximation to $y(t)$ on $[t_0, t_F]$ defined by $z(t) = z_i(t)$ for $t_i \leq t \leq t_{i+1}$ (for $i = 0, 1, \cdots, n$) and $z(t) = \phi(t)$, $z'(t) = \phi'(t)$ for $t \leq t_0$. The associated *defect* is a measure of the amount by which this continuous approximation fails to satisfy the differential equation. The defect for RDEs on the step $[t_n, t_{n+1}]$ is defined by

$$
\begin{aligned}
\delta_n(t) &= z_n'(t) - f(t, z_n(t), z(t - \sigma_1(t, z_n(t))), z(t - \sigma_2(t, z_n(t))), \\
&\quad \cdots, z(t - \sigma_\nu(t, z_n(t))). \qquad (1.5)
\end{aligned}
$$

The defect for NDEs on the step $[t_n, t_{n+1}]$ is defined by

$$
\begin{aligned}
\delta_n(t) &= z_n'(t) - f(t, z_n(t), z(t - \sigma_1(t, z_n(t))), z(t - \sigma_2(t, z_n(t))), \\
&\quad \cdots, z(t - \sigma_\nu(t, z_n(t))), z'(t - \sigma_{\nu+1}(t, z_n(t))), \\
&\quad \cdots, z'(t - \sigma_{\nu+\omega}(t, z_n(t))). \qquad (1.6)
\end{aligned}
$$

If $\tilde{y}_{n+1}$ is the approximate solution of 1.3 or 1.4 at t_{n+1} produced by a numerical method, then

$$
le_n = y_n(t_{n+1}) - \tilde{y}_{n+1} \qquad (1.7)
$$

is called the *local error* at t_{n+1}.

If the function f that defines the derivative in a differential equation depends on the values of the function prior to the current time but cannot be formulated in the form of 1.1 or 1.2, then we call the problem a *non-standard* DDE. Such problems are part of a wider class of problems sometimes referred to as *functional differential equations*.

1.2.2 Numerical Difficulties

There are two major complications that can cause numerical difficulties in conventional approaches for solving DDEs: First, discontinuities may occur in various derivatives of the solution. Second, a delay may vanish, *i.e.* $\sigma \to 0$. When a delay vanishes, we call it a *vanishing delay.*

The first difficulty is due to the presence of the delay terms. In general at the initial point, the right-hand derivative $y'(t_0)^+$, evaluated using f, does not equal the left-hand derivative $\phi'(t_0)^-$. Furthermore ϕ may have discontinuities. A discontinuity can therefore arise and propagate from both the initial time and the history function. In general, the order of a derivative discontinuity(when it is propagated) increases with t for RDEs, but this is not the case for NDEs. This non-smoothness of derivatives of the solution for NDEs causes a numerical difficulty especially when the delay vanishes in the integration interval.

The second complication is important because it may cause a solver to fail by forcing it to choose a sequence of very small steps.

1.3 A Review of Previous work

Almost all DDE solvers are written in one of the three programming languages : Fortran 77, MATLAB, or Fortran 90. There are translations to other programming languages, for example C, but they are basically line for line translations of the original versions and usually have the same interface and calling sequences.

DKLAG6 [2](global order 5), ARCHI [16](global order 4),and DDVERK [14](global order 6) are examples of DDE solvers written in Fortran 77. Although these solvers have proved to be very effective, all of them suffer from the deficiencies of Fortran 77, especially its lack of dynamic memory allocation. Therefore the casual or unaccustomed user usually has trouble remembering the purpose of each argument and might spend a lot of time writing even a simple driver. None of these solvers has an event handling facility.

DDE23 [20](global order 3) and DDESD [17](global order 4) are examples of DDE solvers implemented in MATLAB. They both have very user-friendly interfaces. They also have embedded event locators. DDE23 is restricted to RDEs with constant delays and DDESD is restricted to RDEs.

RADAR5 [11](global order 4), and DDE_SOLVER [21](global order 5) are examples of DDE solvers implemented in Fortran 90. RADAR5 is designed to solve stiff DDEs. Although it is written in Fortran 90, the user interface is much like Fortran 77 and is therefore not as user-friendly as it could be. RADAR5 does not have an embedded event locator. DDE_SOLVER combines the power of DKLAG6 with the user-friendly interface of DDE23. It is a good example of a modern interface for a DDE solver and enables the user to solve a standard problem by writing a very short driver program.

1.4 Contributions of the Thesis

The main contribution of this thesis is the implementation of a user-friendly DDE solver. The new solver DDVERK90 is essentially an improved version of DDVERK. In particular, by exploiting the modern features of Fortran 90 we were able to design an interface that is much easier to understand and use, and is consistent with the interface that has been adopted by Shampine et al. in developing the ODE library provided in MATLAB (see for example [18]).

We have embedded the ability of locating and handling of events in DDVERK90. Not very many DDE solvers provide for event location. We have done this in DDVERK90 in a way that allows the solver to locate multiple events that occur at the same time and perform a user-defined action in the case of the occurrence of an event.

Another important aspect of this thesis is that we have introduced a new approach for classifying DDE problems. This classification helps users to write driver programs to solve a new problem very quickly, by comparing it to similar previously solved example problems in the same class.

We have added the ability of solving some non-standard DDEs to DDVERK90, by providing a mechanism that enables the user to convert (or reduce) these problems to a form that is appropriate for the solver.

We have also introduced a new form for returning the solution as an structure that corresponds to the form adopted in the ODE methods of MATLAB. With this facility, it is easier to do further generic investigations of the solution after the return from any solver and these investigations can be performed in a different environment (for example in MATLAB).

1.5 An Outline of the Thesis

In Chapter 2, first we describe the basic structure of DDVERK90 for solving simple RDE problems, then we present a detailed description of the interface, optional arguments, and how the user solves more general problems. After that we discuss how the user can handle events and solve problems of neutral type. Finally we look at the mechanism of error control in DDVERK90. In Chapter 3, we introduce a new classification of DDE problems with the description and solution of selected examples in each class. In Chapter 4, we investigate the behaviour of DDVERK90 when applied to stiff problems. In Chapter 5, we present numerical results. In Chapter 6, we summarize the thesis and discuss future work.

Chapter 2

New Interface

2.1 Comparison of DDVERK90 with DDVERK from the Viewpoint of Users

In addition to implementing an interface that is as close to the interface that has been introduced for the MATLAB ODE library, one of our other goals was to make the interface as consistent as possible with that of DDVERK. We have used the capabilities of Fortran 90 to remove most unnecessary and distracting arguments, so that the arguments that appear in the call to the solver DDVERK90 are fewer and more meaningful than those of DDVERK.

The form of the functions that define equations for DDVERK and DDVERK90 are the same. DDVERK90 has special facilities for handling constant delays and constant histories, so (unlike DDVERK) in these cases there is no need to define them as functions. If delays are not constant, the form of the delay parameter function of DDVERK90 is also the same as that for DDVERK. For neutral problems in DDVERK there are two different functions for computing the history function and the derivative of the history function, but DDVERK90 uses one single function that has an extra parameter to select between the history function and its derivative.

In DDVERK if users want to know the location of events or investigate a property of the solution that may be of interest, they must do it by writing a special driver, but in DDVERK90, with its embedded ability of event handling and interpolation, users do not need to be as aware of details of the solver progress and are able to focus only on a generic method-independent investigation.

Another important difference is in the error tolerance. DDVERK uses a single error tolerance value, but DDVERK90 supports both relative and absolute error tolerances which is consistent with that provided in the MATLAB ODE methods.

2.2 Simple Calls for Simple Problems (Constant Delays, Constant History)

We tried to make solving simple problems as easy as possible. All one has to do is define the mathematical problem. For example an RDE problem with constant history and constant delays is solved with the call

```
SOL = DDVERK90(NVAR,NLAGS,DDES,LAGS,HISTORY,TSPAN)
```

Here NVAR is an integer array of one entry and its only entry is NEQN, the dimension of the problem. NLAGS is also an integer array of one entry and its only entry is the NU, the number of delays in the problem. The reason that they are arrays instead of scalars is for compatibility with corresponding MATLAB methods that will be clarified later. DDES is the name of the subroutine for evaluating the right hand side of the equation (1.1). It has the form

```
SUBROUTINE DDES(T,Y,Z,DY)
```

The input arguments are the independent variable T, a vector Y of NEQN components approximating $y(T)$, and a dimension NEQN×NU array Z. Column j of this array is an approximation to $y(T - \sigma_j(T, y(T))$. The subroutine evaluates the right hand side of the equation (1.1) with these arguments and returns $y'(t)$ as the vector DY of NEQN components. Returning to the description of the parameters of DDVERK90, LAGS is a vector of NU entries. LAGS(i) is the ith constant value of the delay. HISTORY is a vector of NEQN components. HISTORY(i) is the the constant value for y_i for $t \leq t_0$.

The last argument is the input vector TSPAN. The value returned by DDVERK90 is the structure SOL. These parameters have the same meaning and name that they have in the MATLAB ODE solvers. TSPAN is used to inform the solver of the interval of integration and where approximate solutions are desired. TSPAN has at least two entries. The first entry is the initial point of the integration, t_0, and the last is the final point, t_F. If TSPAN has only two entries, approximate solutions are returned at all the mesh points selected by the solver itself. If TSPAN has entries $t_0 < t_1 < \cdots < t_F$, the solver returns approximate solutions at (only) these points. The number and placement of these output points has little effect on the cost of the integration. This is because the solver selects a mesh that allows it to compute an accurate solution efficiently and evaluates a continuous extension (a polynomial interpolant) to obtain approximate solutions at specified output points. The polynomial interpolant is represented in a generic (method-independent) way by the structure, SOL. Table 2.1 shows concisely the required input arguments to DDVERK90.

2.3 General Problems

In general the initial history will not be constant and should be defined by a function. In this case the function has the form

```
SUBROUTINE HISTORY(T,Y)
```

The input argument is a time T $\leq t_0$. The output is a vector Y of NEQN components that is the value of the history function $\phi(T)$.

Similarly, the delays may be time dependent or time and state dependent. Therefore they should be defined by a function. In DDVERK90 the function must have the form

NVAR	Number of DDEs and event functions (vector)
NLAGS	Number of delays(NU,OMEGA) (vector)
DDES	Subroutine to evaluate DDEs
BETA(LAGS)	Subroutine for delays (vector for constant lags)
HISTORY	Subroutine for history (vector for constant history)
TSPAN	Interval of integration and output points

Table 2.1: Required input arguments to DDVERK90.

```
SUBROUTINE BETA(T,Y,BVAL)
```

The input arguments are the independent variable T, a vector Y of NEQN components approximating $y(T)$. The output variable BVAL is a vector of NU components. After the return, the element i of BVAL should be the value of the delay argument $T-\sigma_i(T, y(T))$.

2.4 Optional Arguments

One of the difficulties that users have with DDVERK and other Fortran 77 DDE solvers is that they must supply a large number of arguments. This is partly due to the static storage nature of Fortran 77. For most users it is hard to remember the purpose of each argument and to set them properly. Furthermore, since Fortran 77 does not have an easy way for using global variables, mechanisms that DDE solvers provide for users to deal with parameters can be very complicated and error-prone and they can vary considerably for different methods.

We used the capabilities of Fortran 90 to overcome these difficulties. To reduce the number of arguments and make them more meaningful we exploited the optional argument capability of Fortran 90. Fortran 90 allows optional arguments to functions and subroutines that follow the required arguments in the call list. Though optional

OPTIONS	Options structure formed with DDVERK_SET
EVENT_FCN	Subroutine to evaluate event functions
CHANGE_FCN	Subroutine for action at an event
OUT_FCN	Subroutine for output

Table 2.2: Optional input arguments to DDVERK90.

arguments can be passed by position in the list, we assume that they will always be identified with keywords, another new possibility in Fortran 90. Using keywords, we can set just the options of interest and we can set them in any order. These optional arguments are shown in Table 2.2. (We have done this in a way that is similar to what Thompson and Shampine implemented DDE_SOLVER [21].)

We discuss how optional argument are specified using the parameter OPTIONS below and will later discuss other uses of these parameters.

2.4.1 OPTIONS

We have defined a derived type DDVERK_OPTS with fields corresponding to various options. The auxiliary function DDVERK_SET is used to set the options. All arguments of DDVERK_SET are optional, so if the user does not provide an options structure to the solver, it will use OPTIONS=DDVERK_SET() to form an options structure with all fields set to the default values. The user has to set only those options for which default values are inappropriate for the problem at hand.

A typical call of DDVERK_SET is one that sets relative and absolute error tolerances. DDVERK90 uses a scalar relative error tolerance and either a scalar or vector absolute tolerance. If the scalar relative error tolerance RE is not specified, it is given a default value of 10^{-3}. Absolute error tolerances are more complicated. The most common case is the default, which is to set all NEQN components of the absolute error tolerance vector to 10^{-6}. Corresponding to the scalar RE option there is a scalar AE option. In

RE	Relative error tolerance	10^{-3}
AE	Absolute error tolerance	10^{-6}
AE_VECTOR	Vector of absolute error tolerances	10^{-6} for all
ISTERMINAL	Specify terminal events	.FALSE.
DIRECTION	Distinguish how event functions cross axis	0
INTERPOLATION	Prepare to interpolate solution	.FALSE.
NEUTRAL	Solve DDE of neutral type	.FALSE.
HINIT	Initial step size	no default
HMAX	Maximum step size	no default
THIT_EXACTLY	Times to be hit as mesh points	no default
DEFECT_ESTIMATE_TYPE	Type of the defect estimate	1
SHOW_PROGRESS	Show current t after each 1000 steps	.FALSE.

Table 2.3: Options set with DDVERK_SET and their default values.

addition there is a vector option called AE_VECTOR. If no absolute error tolerance is specified, the default is used. If the scalar option AE is set, the value input is assigned to all NEQN components of an absolute error tolerance vector used by the solver. If the vector option AE_VECTOR is set, the vector (of NEQN components) that is input is used for the absolute error tolerance vector. If both options are set inadvertently, the more detailed vector option is given precedence. Table 2.3 shows all the options set with DDVERK_SET and their default values. Some of them will be explained in future sections.

2.5 Event Location

Along with the integration of the DDEs, we may be interested in locating where any one of a collection of event functions,

$$g_j(t, y(t), y'(t), y(t - \sigma_1(t, y(t))), \cdots, y(t - \sigma_\nu(t, y(t)))) \quad (2.1)$$

vanishes. This is a generic type of an event that can arise in many applications. We have implemented an approach for specifying and investigating events that is an extension of that implemented in MATLAB and also in DDE_SOLVER. The places where these event functions vanish are called *events*. Sometimes we just want the solver to report the location of an event and the solution there. Other times we want to terminate the integration or restart the integration after changing the problem. Not very many DDE solvers provide for event location. Event location is optional, so the solver must be told that it is to locate zeros of a collection of functions evaluated in a given procedure. In DDVERK90 the user provides the name of the subroutine (to evaluate the g_j's) as an optional input argument of the solver itself that is specified with the keyword EVENT_FCN. The function has the form

```
SUBROUTINE EF(T,Y,DY,Z,G).
```

The input arguments are the independent variable T, a vector Y of NEQN components approximating $y(T)$, a vector DY of NEQN components approximating $y'(T)$, and array Z that has dimension NEQN×NU. Column j of this array is an approximation to $y(T - \sigma_j(T, y(T))$. The subroutine evaluates (2.1) with these arguments and returns $g_j(T)$ as the element j of a vector G of size equal to the number of event functions(NEF). When using events the user must define the vector NVAR with two components and the second component should be set to NEF. Recall that the first component of NVAR is set to NEQN.

It can be useful (even essential) to distinguish how an event function behaves near the associated zeros. For example, this knowledge can be used to deal with problems in which events are defined to find maximums or minimums. We added this capability to DDVERK90, with a vector DIRECTION that is set as an option using DDVERK_SET. Component j of the vector DIRECTION is given the value -1 when event j is interesting only if the event function decreases through 0, the value +1 when it is interesting only if the function increases through 0, and the value 0 when it does not matter.

A very important distinction is made between terminal and non-terminal events. As the name suggests, a terminal event causes the solver to halt and return control to the user. In our solver a vector ISTERMINAL (of NEF components and type LOGICAL), one of the optional arguments of DDVERK_SET, is used for this purpose.

When an event occurs the user may wish to make changes to the problem formulation or any other parameters that together define the problem. This can be done in our solver, with a user defined function that should be passed to the solver in the CHANGE_FCN optional argument of DDVERK90. This function is of the form

```
SUBROUTINE CHANGE(HAVE_EVENT,T,Y,DY,HINIT,DIRECTION,
                                   ISTERMINAL,QUIT,IS_CHANGED)
```

The solver calls this subroutine at every event. In this call HAVE_EVENT is an vector of NEF components and type LOGICAL that identifies which event(s) occurred (there may have been more than one event), T is the location of the event(s), and Y and DY are the approximations to the solution and its first derivative, respectively, at the event. In CHANGE the user can inspect this information and decide what, if any, action is appropriate. Y and DY are also output variables that the user can reset. In effect he/she can restart the integration with new initial values. The output variable HINIT allows a user to specify a step size for the solver to try on return. The integer array DIRECTION and the logical array ISTERMINAL can be changed so that the event functions will be interpreted differently on return from CHANGE. The integration can be terminated by changing QUIT from its input value of .FALSE. to .TRUE.. If the user makes any major changes and wants the solver to pay special attention to them, then the LOGICAL variable IS_CHANGED should be set to .TRUE.. This causes the solver to include T as a mesh point and treat it as a potential discontinuity point.

2.6 Form of the Solution

The default is to return the approximate solution at the mesh points selected by DDVERK90. If the solution structure is called SOL, the number of mesh points NPTS is returned as SOL%NPTS, the mesh points $t_0 < t_1 < \cdots < t_F$ chosen by the solver are returned as entries of SOL%T, and the corresponding approximate solutions are returned in the NPTS $\times$ NEQN array SOL%Y. When TSPAN has more than two entries then SOL%T is the same as TSPAN.

A general purpose DDE solver must have an associated interpolation scheme that allows the solution to be approximated accurately between mesh points. DDVERK90 can return, in the solution structure, the information needed to evaluate a continuous extension which also serves as the interpolation scheme anywhere in the interval of integration. Since the capability requires a considerable amount of additional information in SOL, we have made this an option called INTERPOLATION. By default it has the value of .FALSE.. DDVERK_SET cans be used to set this option to .TRUE.. If SOL contains the information needed for interpolation, then a call to the auxiliary function DDVERK_VAL of the form

```
YINT = DDVERK_VAL(T,SOL)
```

evaluates approximations to $y(T)$ for any vector of points T that all lie in $[t_0, t_F]$. Similarly approximations to $y'(T)$ are also returned when an optional argument of DDVERK_VAL with keyword DERIVATIVES is set to .TRUE.. Sometimes only selected components are of interest. The optional argument with keyword COMPONENTS is a vector that tells DDVERK_VAL which components of the solution are desired. For example, the first and fourth components of the solution at the two arguments $t = 2.35, 0.72$ would be returned by

```
YINT = DDVERK_VAL((/  2.35D0, 0.72D0 /),SOL,COMPONENTS=(/ 1, 4 /)).
```

The output argument is a derived type called DDVERK_INT, so YINT in this example must be declared as

```
TYPE(DDVERK_INT) :: YINT.
```

Although the vector T is available in the calling program, it is convenient to return it as the field YINT%TVALS in the interpolation structure. Similarly, if a vector is provided for the keyword COMPONENTS, it is returned as the field YINT%COMPONENTS. If this option is not set, the field is given the default value of all the components, namely the vector with entries 1,...,NEQN. The approximate solution is returned in the field YINT%YT. More specifically, YINT%YT(I,J) is an approximation to component J of the solution at T(I). Similarly, if DERIVATIVES=.TRUE., an approximation to component J of the first derivative of the solution at T(I) is returned in YINT%DT(I,J).

If the standard output options are not exactly what the user wants, he/she can write their own output function. For instance, if NEQN is large, the user might find that returning of all components of the solution at all mesh points to be excessive. By means of an output function the user could output only the components of particular interest. An output function must have the form

```
SUBROUTINE SOL_OUT(T,Y,DY,IS_EVENT,HAVE_EVENT)
```

The solver is informed of this using a keyword for this optional argument: OUT_FCN = SOL_OUT. The solver will call SOL_OUT at every step and every event. In this call T is the value of the independent variable and Y and DY are the NEQN components of the solution and its first derivative, respectively, at T. If IS_EVENT is .FALSE., the solver has stepped to the new meshpoint T. If IS_EVENT is .TRUE., there is at least one event function for which the LOGICAL vector HAVE_EVENT is .TRUE., indicating which event function has vanished at T. The user can inspect this information and do whatever is desired before returning control to the solver. For example, the user might write a particular solution component to a file.

2.7 Neutral Problems

Solving NDEs with DDVERK90 is very similar to solving RDEs. The option NEUTRAL must be set to .TRUE.. NLAGS should be defined as a vector of two elements with NU and OMEGA being its first and second elements. The definition of Z in the DDES subroutine should be extended so that the array is of size NEQN×(NU+OMEGA). The first NU columns correspond to usual $y(t-\sigma_j(t,y(t))), j=1,\cdots,\nu$ and the remaining OMEGA columns correspond to $y'(t-\sigma_j(t,y(t))), j=\nu+1,\cdots,\omega$. The HISTORY function should have an extra argument and should have the form

```
SUBROUTINE HISTORY(T,Y,IS_DERIVATIVE).
```

The extra input IS_DERIVATIVE is a LOGICAL variable. If it is .FALSE. the routine should set Y to $\phi(T)$, otherwise Y should be set to $\phi'(T)$. If the history is constant and supplied as a vector, the solver realizes that the derivative of the history is zero so that it is not necessary to pass this value to the solver. The BVAL output of the subroutine BETA that was described in Section 2.3 is extended to a vector of size NU+OMEGA. The first NU elements correspond to the usual $t-\sigma_j(t,y(t)), j=1,\cdots,\nu$, delay arguments and the remaining OMEGA elements correspond to the $t-\sigma_j(t,y(t)), j=\nu+1,\cdots,\omega$, delay arguments. The form of the event function g_j in (2.1) is extended to the form

$$
\begin{aligned}
g_j(t,y(t),y'(t),y(t-\sigma_1(t,y(t))),\cdots,y(t-\sigma_\nu(t,y(t))),\\
y'(t-\sigma_{\nu+1}(t,y_n(t))),\cdots,y'_n(t-\sigma_{\nu+\omega}(t,y(t)))
\end{aligned}
\tag{2.2}
$$

and hence the definition of Z in the EF subroutine described in Section 2.5 should be extended so that the array is of size NEQN×(NU+OMEGA).

2.8 Defect Control, Absolute Error, Relative Error

There are two types of error control strategies that we have considered for the numerical solution of DDEs: controlling the local error and controlling the defect. In the local error control strategy the solver attempts to ensure that the local error (1.7) satisfies a local error per step inequality of the form,

$$|le_n| \leq \text{absolute tolerance} \times h_n, \tag{2.3}$$

or a mixed local error inequality of the form

$$|le_n| \leq |\tilde{y}_{n+1}| \times \text{relative tolerance} + \text{absolute tolerance} \times h_n. \tag{2.4}$$

Since it is not generally possible to compute the exact value of le_n, solvers use an estimate of this value. ARCHI [16] uses (2.3). DKLAG6 [2], MATLAB DDE23 [20], DDE_SOLVER [21] and RADAR5 [11] use (2.4). All of the solvers are also able to use separate absolute tolerances for different components of the solution.

In the original defect control strategy for DDEs, evaluated and analysed in [6], a solver tries to control the defect in the form

$$\max_{t_n \leq t \leq t_{n+1}} |\delta_n(t)| \leq \text{absolute tolerance}, \tag{2.5}$$

or

$$\max_{t_n \leq t \leq t_{n+1}} \left| \frac{\delta_n(t)}{\tilde{y}'(t)} \right| \leq \text{relative tolerance}. \tag{2.6}$$

Since it is impossible (or at least very expensive) to compute the exact value of $\max_{t_n \leq t \leq t_{n+1}} |\delta_n(t)|$ or $\max_{t_n \leq t \leq t_{n+1}} \left| \frac{\delta_n(t)}{\tilde{y}'(t)} \right|$, solvers often use an estimate of these values by sampling. This strategy has proven to be effective for controlling the step size and global error [14]. To use a measure that supports both relative and absolute error control, it is possible to modify this original strategy. For example Shampine [17] controls $h_n \times (\max_{t_n \leq t \leq t_{n+1}} |\delta_n(t)|)$ at each step. He used this as a measure of local error and employed a strategy similar to 2.4. We adapted DDVERK defect control strategy in a similar way. In DDVERK90 the defect that is used inside the solver subroutines is based on a sample (or samples) of $|\delta_n(t)|$. But when the solver comes to the final decision of accepting or rejecting a step it uses,

$$\begin{aligned}\text{a sample (or all samples) of } |\delta_n(t)| \;\; &\leq \;\; |\tilde{y}_n'(t)| \times \text{relative tolerance} \\ &\quad +\text{absolute tolerance,} \end{aligned} \tag{2.7}$$

After testing strategy (2.7) on several problems that were solved before by DDVERK we deduced that this strategy does not have much affect on the efficiency of the solver, but it helps to give the user a more generic method-independent error control (without having to know anything about h_n).

The sampling options available in DDVERK90 are same as DDVERK and are selected by setting optional argument DEFECT_ESTIMATE_TYPE to values 1,2 and 3 corresponding to the one point, two point and asymptotically valid sampling strategies of DDVERK (see [14] for more details). The default value is 1.

Chapter 3

Different Types of Problems and Their Solutions

In this chapter, we describe various classes of problems that users can solve using DDVERK90. We present the description of some problems in each class and show the plots of numerical solution produced by DDVERK90. This classification helps users to write driver programs to solve a new problem very quickly, by comparing it to similar previously solved example problems in the same class. Although it is possible to solve almost all of these examples by existing solvers, the driver program for some of them would be very complicated and the users could spend a lot of time writing them. The detailed numerical results are given in Chapter 5.

3.1 Standard RDE Problems

3.1.1 Example 1

Our first example is Example 5 of Wille & Baker [24], which involves a scalar equation that exhibits chaotic behavior. It is also studied as Example 4.4.2 in reference [21] and Example 2 in reference [20]. The problem has a constant delay and a constant history,

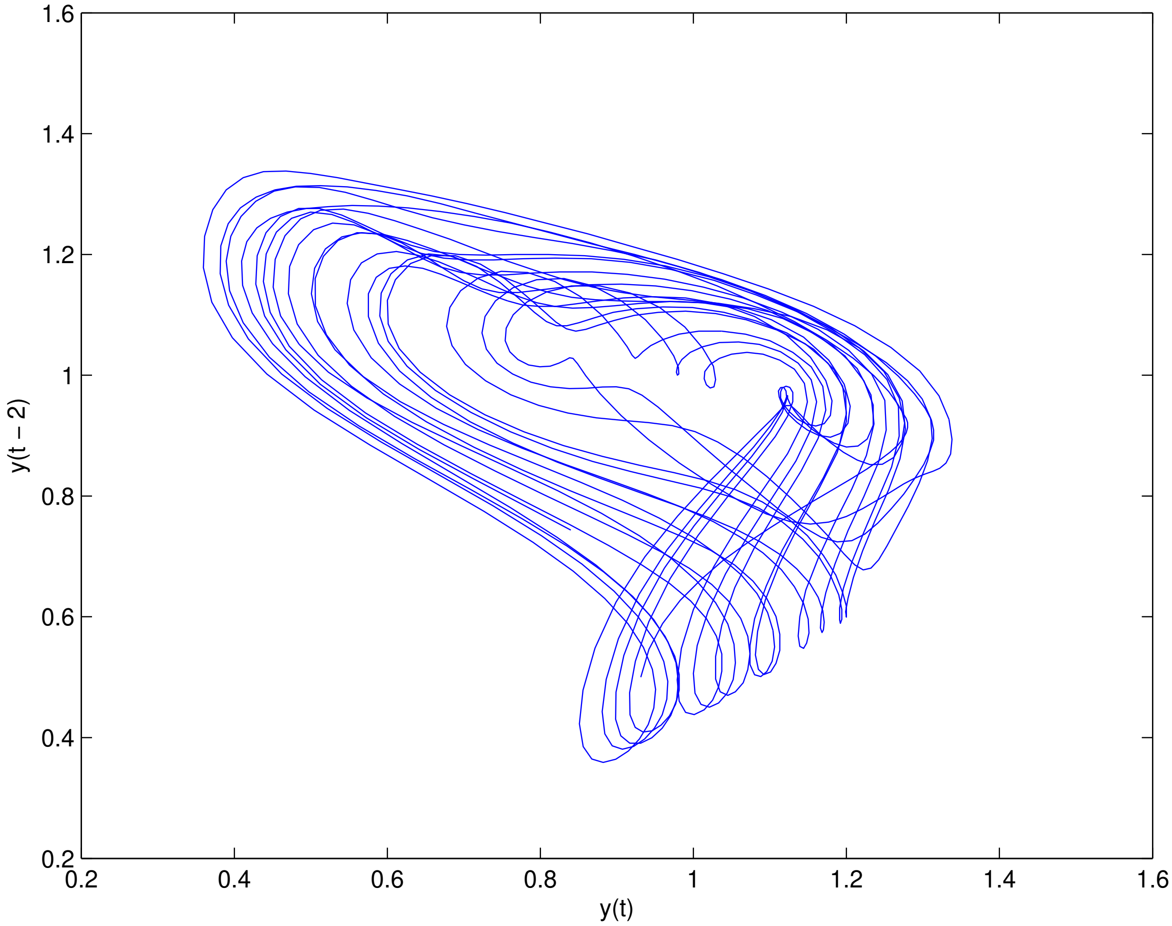

Figure 3.1: Phase plot of numerical solution to Example 1, computed using DDVERK90 with RE=AE=10^{-6}.

and is defined by

$$y' = \frac{2y(t-2)}{1+y(t-2)^{9.65}} - y,$$

for t in $[0, 100]$. The history function is

$$y = 0.5 \text{ for } t \leq 0.$$

After running a DDVERK90-based program and writing the approximate solution to an output file, we use a MATLAB program to plot the phase plane $y(t-2)$ versus $y(t)$, which is a common way of visualizing solutions in nonlinear dynamics. The exact solution of this problem is unknown. See Figure (3.1) for an accurate approximate solution.

3.1.2 Example 2

This example is called Example 3 in reference [5]. The problem has a time dependent delay and a time dependent history with an asymptotically vanishing delay (as t increases), and is defined by

$$y' = (1 + e^{-t})y(t - e^{-t})\exp(e^{-t+e^{-t}}),$$

for t in $[1, 30]$. The history function is

$$y = e^{t-e^{-t}} \text{ for } t \leq 1.$$

The exact solution to this problem (see Figure (3.2)) is,

$$y = e^{t-e^{-t}}.$$

3.1.3 Example 3

This example is called Example 1 in reference [2]. The problem has a state dependent delay and a constant history, and is defined by

$$y' = \frac{y(t)y(\ln(y(t)))}{t},$$

for t in $[1, 10]$. The history function is

$$y = 1 \text{ for } t \leq 1.$$

The exact solution to this problem (see Figure (3.3)) is

$$y(t) = \begin{cases} t & \text{if } 1 \leq t \leq e, \\ \exp(t/e) & \text{if } e \leq t \leq e^2, \\ \left(\frac{e}{3-\ln(t)}\right)^e & \text{if } e^2 \leq t \leq e_3, \\ \text{not known} & \text{if } e_3 < t \ , \end{cases}$$

where $e_3 = \exp(3 - \exp(1 - e))$.

Derivative jump discontinuities occur at $t = e$, $t = e^2$ and e_3.

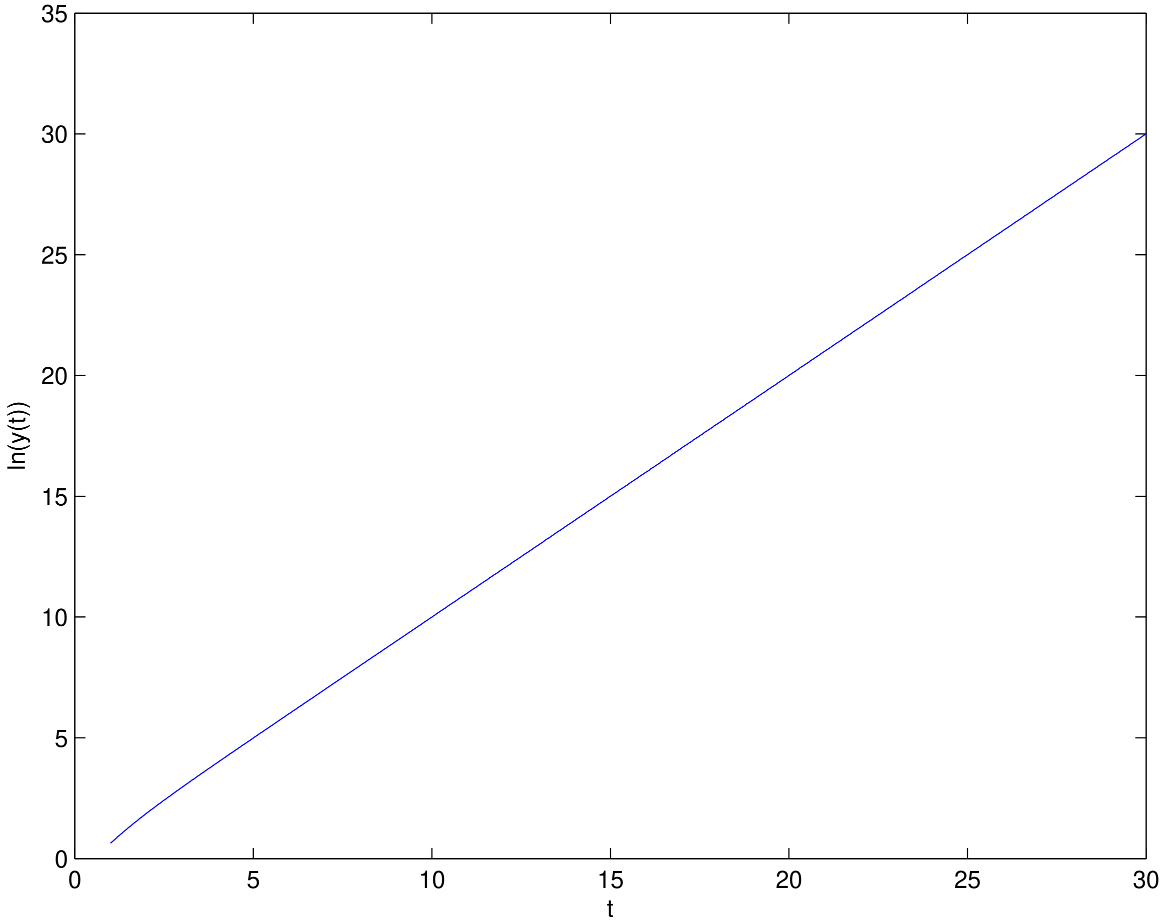

Figure 3.2: Plot of numerical solution to Example 2, computed using DDVERK90 with RE=AE=10^{-6}.

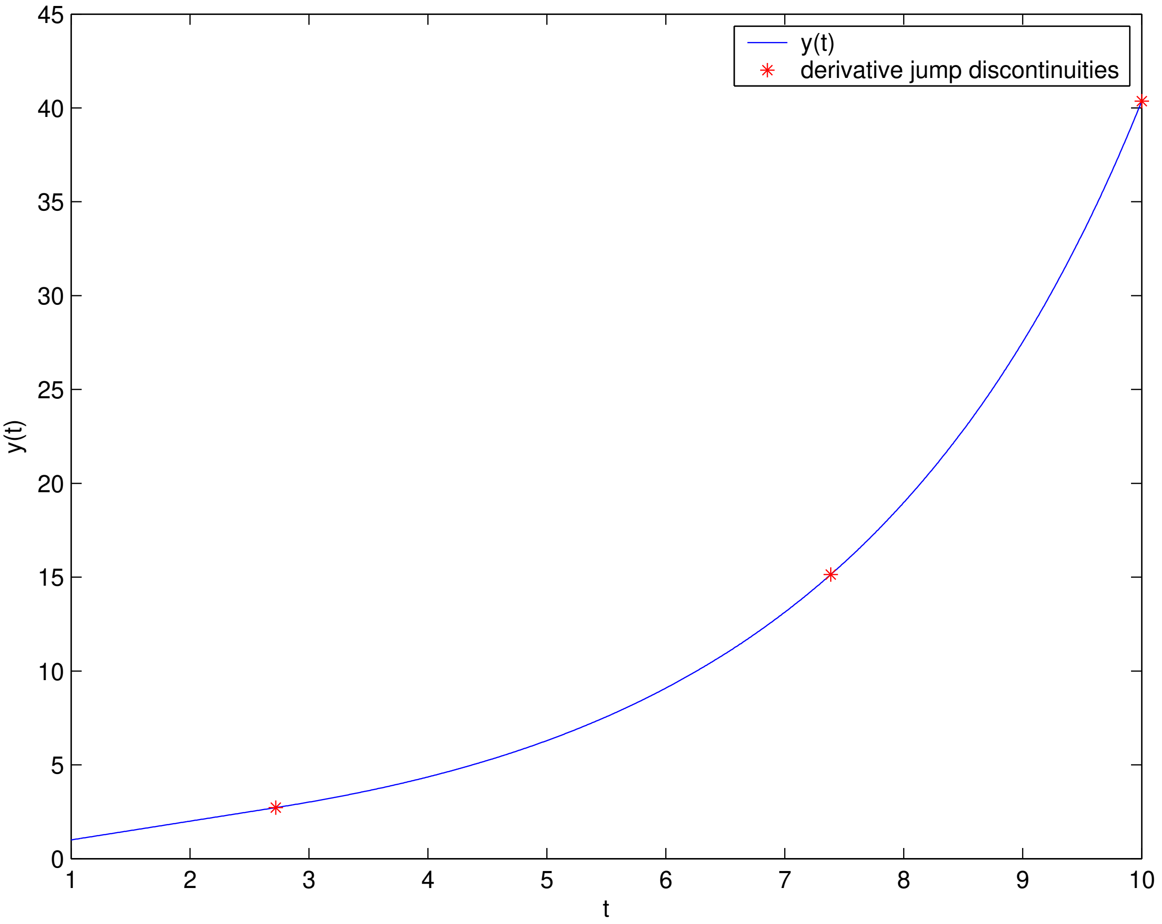

Figure 3.3: Plot of numerical solution to Example 3, computed using DDVERK90 with RE=AE=10^{-5}.

3.1.4 Example 4

This example is the SEIR epidemic model of Genik & van den Driessche [9]. This is a transformed version of an integro-differential equation system. The problem has constant delays and a constant history, and is defined by

$$\begin{aligned}
S' &= A - dS(t) - \lambda \tfrac{S(t)I(t)}{N(t)} + \gamma I(t-\tau)e^{-d\tau}, \\
E' &= \lambda \tfrac{S(t)I(t)}{N(t)} - \lambda \tfrac{S(t-\omega)I(t-\omega)}{N(t-\omega)} e^{-d\omega} - dE(t), \\
I' &= \lambda \tfrac{S(t-\omega)I(t-\omega)}{N(t-\omega)} e^{-d\omega} - (\gamma + \epsilon + d)I(t), \\
R' &= \gamma I(t) - \gamma I(t-\tau)e^{-d\tau} - dR(t),
\end{aligned}$$

where

$$N(t) = S(t) + E(t) + I(t) + R(t),$$

and $A = 0.33$, $d = 0.006$, $\lambda = 0.308$, $\gamma = 0.04$, $\epsilon = 0.06$, $\tau = 42$, $\omega = 0.15$, for t in $[0, 350]$. The history function is,

$$\begin{aligned}
S &= 15, \\
E &= 0, \\
I &= 2, \\
R &= 3.
\end{aligned}$$

for $t \leq 0$.

The exact solution of this problem is unknown. See Figure (3.4) for an accurate approximate solution.

3.2 Neutral Problems

3.2.1 Example 5

This example is called Example 8 in reference [14]. The problem is a state dependent neutral problem with vanishing delays, and is defined by

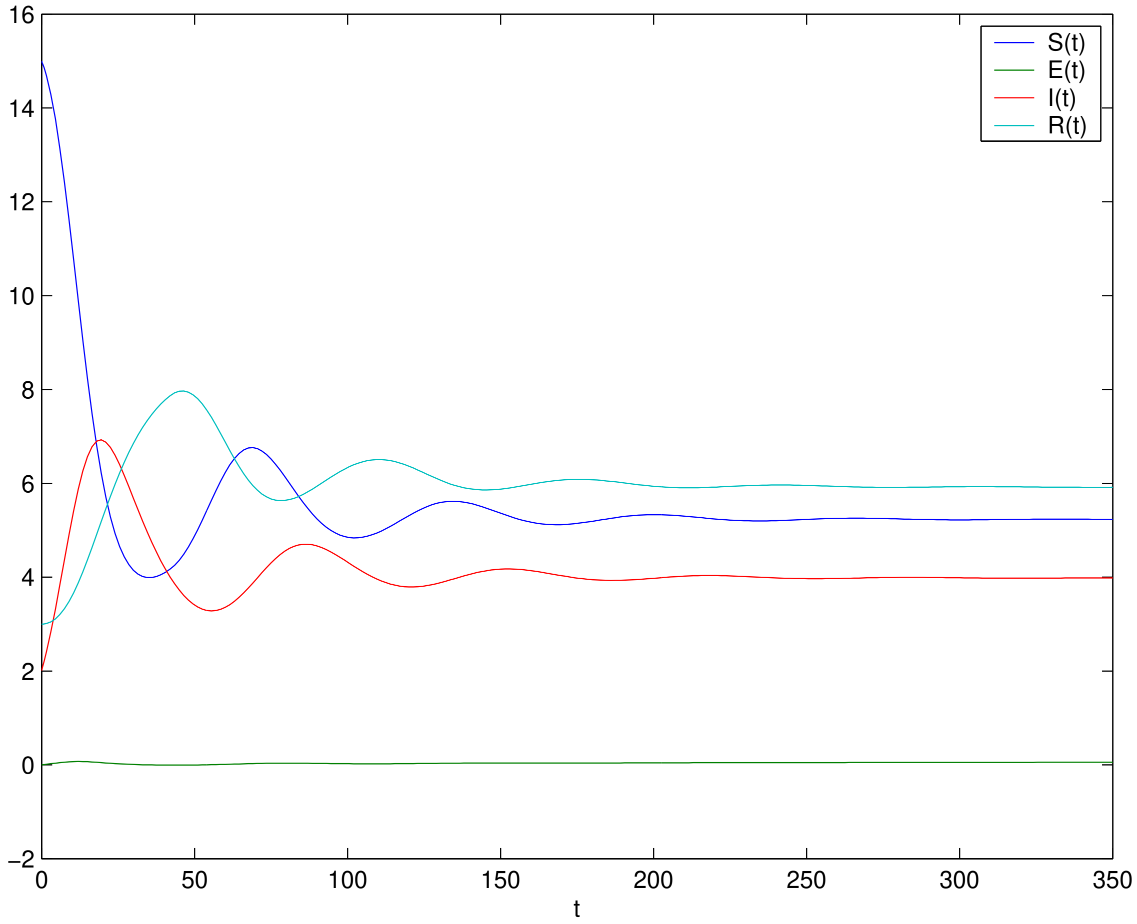

Figure 3.4: Plot of numerical solution to Example 4, computed using DDVERK90 with RE=10^{-5}.

$$\begin{aligned} y'(t) &= \cos(t)(1 + y(ty^2(t))) + L_3 y(t) y'(ty^2(t)) \\ &\quad + (1 - L_3)\sin(t)\cos(t\sin^2(t)) - \sin(t + t\sin^2(t)), \end{aligned}$$

for t in $[0, \pi]$ with $L_3 = 0.1, 0.3, 0.5^\dagger$. The initial condition is

$$\phi(0) = 0, \ \phi'(0) = 1.$$

The exact solution to this problem is $y(t) = \sin(t)$ (see Figure (3.5)).

There is no history or discontinuities associated with this problem but there are vanishing delays at $t = 0$, $t = \pi/2$, and $t = \pi$.

The vanishing delay at $t = \pi/2$ is challenging since it must be detected in the interior of the integration interval.

3.3 Problems with Associated Events

3.3.1 Example 6

This example is a Kermack-McKendrick model of an infectious disease with periodic outbreak and is called Problem 17.14 in reference [12]. It is also studied in reference [3] and is called Example 1 in reference [19] and Example 4 in reference [20]. The problem is defined by

$$\begin{aligned} y_1' &= -y_1(x)y_2(x-1) + y_2(x-10), \\ y_2' &= y_1(x)y_2(x-1) - y_2(x), \\ y_3' &= y_2(x) - y_2(x-10), \end{aligned}$$

for t in $[0, 55]$. The history function is

$$y_1 = 5,$$

$$y_2 = 0.1,$$

$$y_3 = 1,$$

$^\dagger L_3$ is a Lipschitz constant with respect to y' at $\pi/2$.

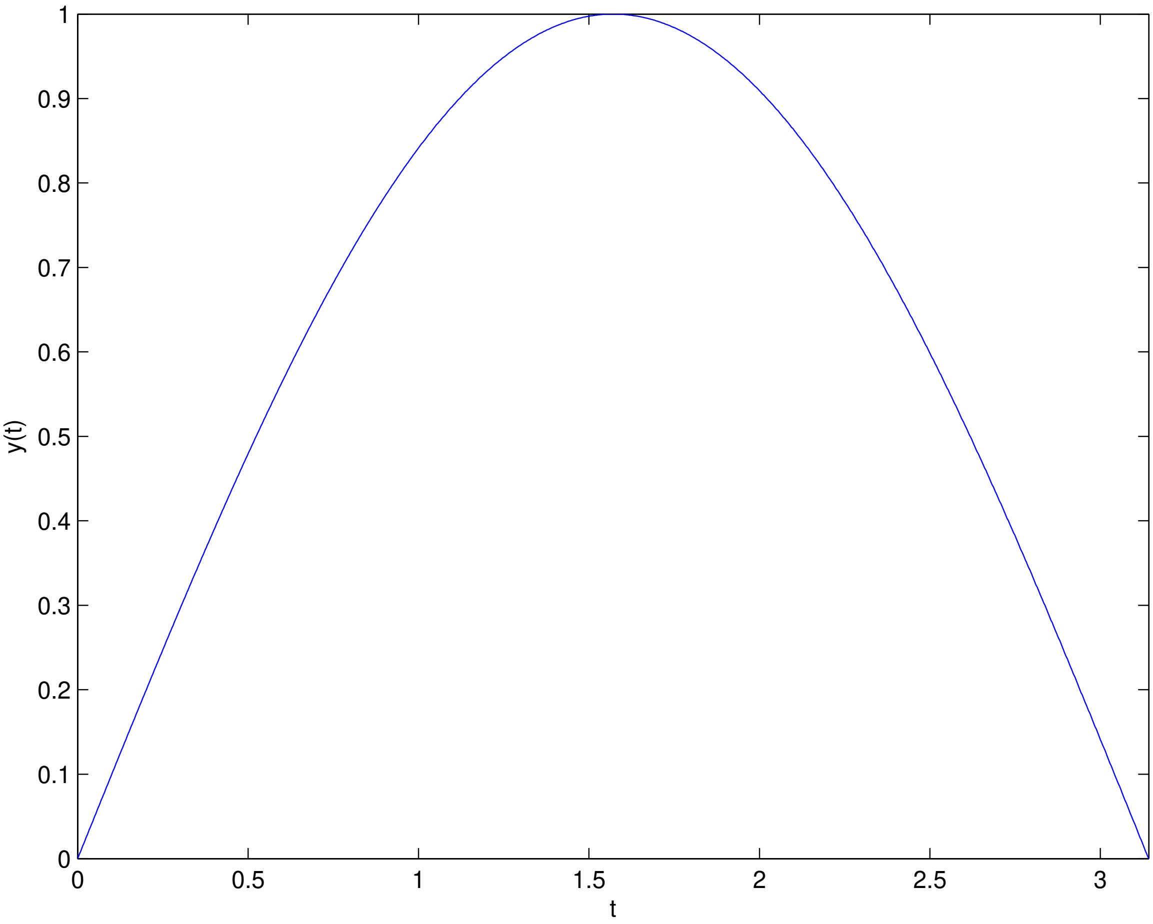

Figure 3.5: Plot of numerical solution to Example 5, computed using DDVERK90 with RE=AE=10^{-10}.

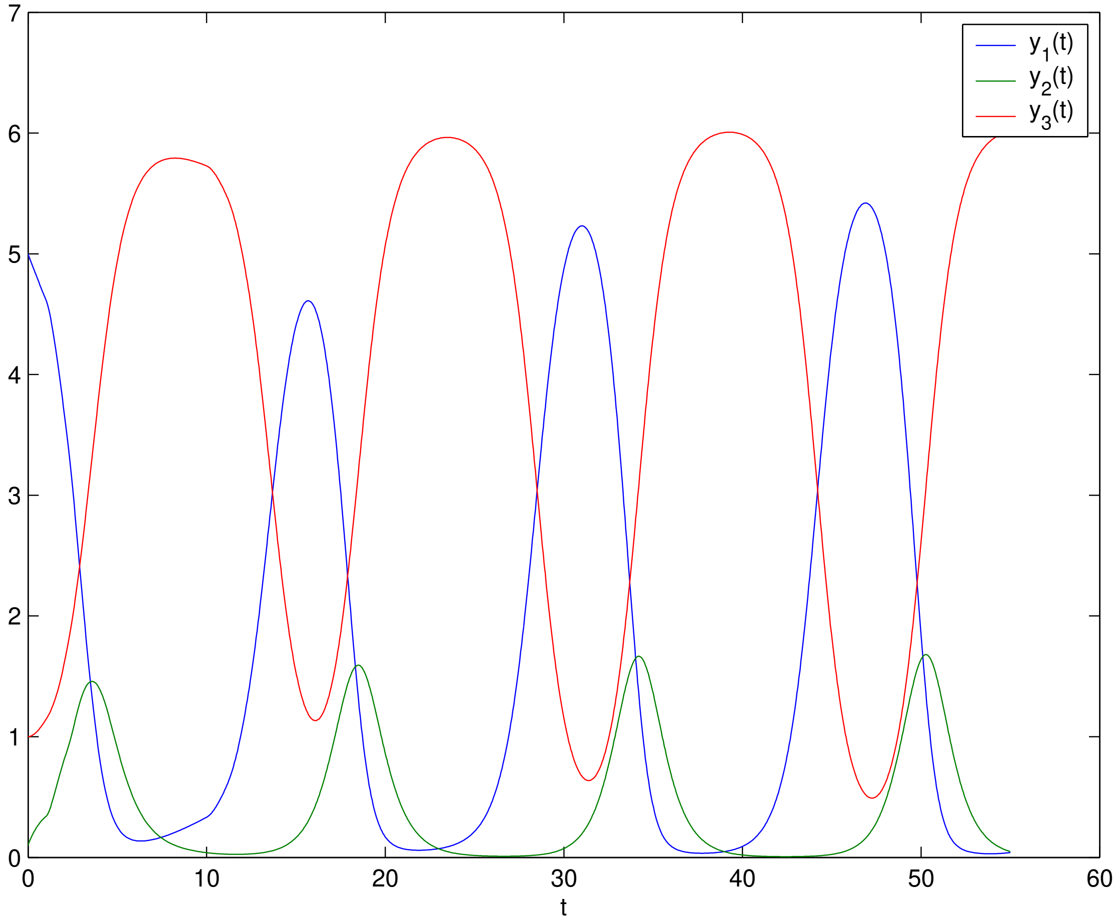

Figure 3.6: Plot of numerical solution to Example 6, computed using DDVERK90 with RE=AE=10^{-6}.

for $t \leq 0$.

The exact solution of this problem is unknown. See Figure (3.6) for an accurate approximate solution.

For this problem, one may be interested in investigating the local extrema of some of the populations. We can do this by defining events to correspond to points where the first derivative of the specific component is zero. We are also able to distinguish a local maximum from a local minimum by setting the DIRECTION argument appropriately.

3.4 Problems that Change after Events (Real Events, User-Made Events)

3.4.1 Example 7

This example is a two-wheeled suitcase model of Suherman, et al and is called Example 8 in reference [20] and Example 3 in reference [19]. It illustrates the use of events and a CHANGE routine to solve a sequence of problems corresponding to the rocking of the suitcase. The problem is defined by

$$y_1' = y_2,$$
$$y_2' = \sin(y_1) - \mathrm{sign}(y_1)\gamma\cos(y_1) - \beta y_1(t-\tau) + A\sin(\Omega t + \eta),$$

where $\gamma = 2.48$, $\beta = 1$, $\tau = 0.1$, $A = 0.75$, $\Omega = 1.37$, $\eta = \arcsin(\gamma/A)$,
for t in $[0, 12]$. The history function is

$$y_1 = 0,$$
$$y_2 = 0,$$

for $t \leq 0$.
The exact solution of this problem is unknown. See Figure (3.7) for an accurate approximate solution.

Due to the $\mathrm{sign}(y_1)$ term, the problem definition changes each time y_1 crosses 0. This is important because the solution is discontinuous at these points. We can detect this event and then change the problem formulation by changing a variable that reflects the status of y_1.

3.4.2 Example 8

This example is the Marchuk immunology model and is called Example 17.19 in reference [12] and Example 2 in reference [19] and Exercise 7 in reference [20]. The problem is

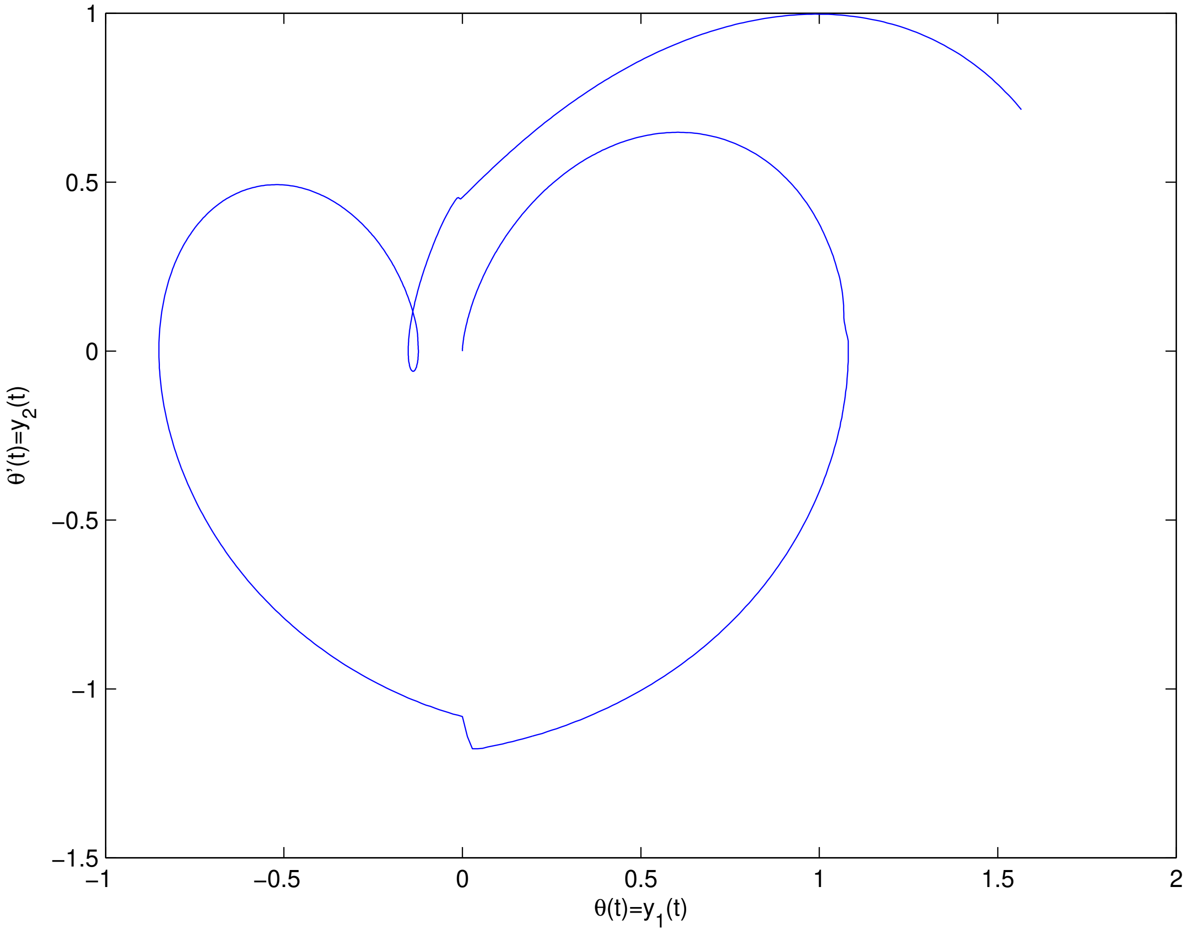

Figure 3.7: Phase plot of numerical solution to Example 7, computed using DDVERK90 with RE=10^{-8}.

defined by

$$
\begin{aligned}
V' &= (h_1 - h_2 F)V, \\
C' &= \xi(m) h_3 F(t-\tau)V(t-\tau) - h_5(C-1), \\
F' &= h_4(C-F) - h_8 FV, \\
m' &= h_6 V - h_7 m,
\end{aligned}
$$

where

$$
\xi(m) = \begin{cases} 1 & \text{if } m \le 0.1, \\ (1-m)\frac{10}{9} & \text{if } 0.1 \le t \le 1, \end{cases}
$$

and $\tau = 0.5$, $h_1 = 2$, $h_2 = 0.8$, $h_3 = 10^4$, $h_4 = 0.17$, $h_5 = 0.5$, $h_7 = 0.12$, $h_8 = 8$, and for different values of h_6 ($h_6 = 10$ or $h_6 = 300$).

It is solved for t in $[0, 60]$. The history function is

$$
\begin{aligned}
V &= \max(0, 10^{-6} + t), \\
C &= 1, \\
F &= 1, \\
m &= 0,
\end{aligned}
$$

for $t \le 0$.

Due to the $\xi(m)$ term, the problem definition changes each time m crosses 0.1. We can detect this event and then change the problem formulation by changing a variable that reflects the value of m. This example also has special history for $V(t)$, and hence can be classified under 3.5 as well.

The exact solution of this problem is unknown. See Figure (3.8) for an accurate approximate solution.

3.4.3 Example 9

This example simulates an HIV long-term partnership model and is called Example 3 in reference [2]. It illustrates the reduction of a Volterra-Integro-Differential equation to a system of DDEs. It further illustrates the use of events to switch between a system of

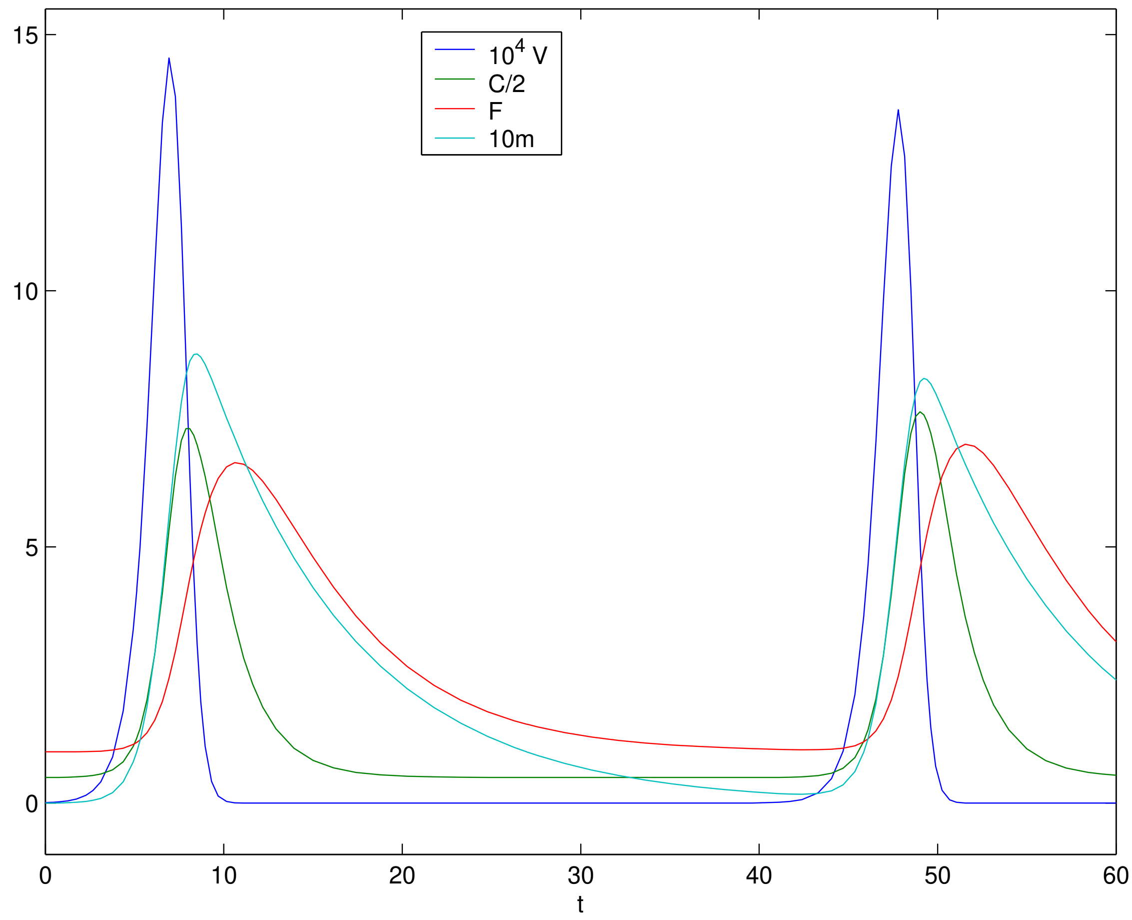

Figure 3.8: Plot of numerical solution to Example 8, computed using DDVERK90 with RE=10^{-5}, AE=10^{-8}.

ODEs and a system of DDEs. The problem is defined by

for $t \leq D$:

$$
\begin{aligned}
x'(t) &= -\lambda(t)x(t) + Gy(t), \\
y'(t) &= -x'(t), \\
\lambda'(t) &= \tfrac{c}{n}\left\{g'(t)I_4(t) + g(t)I_4'(t) + g'(t)I_3(t) + g(t)I_3'(t)\right\}, \\
I_1'(t) &= f_1(t), \\
I_2'(t) &= f_2(t), \\
I_3'(t) &= f_3(t), \\
I_4'(t) &= f_2(t)I_1(t),
\end{aligned}
$$

while for $t \geq D$:

$$
\begin{aligned}
x'(t) &= -\lambda(t)x(t) + Gy(t), \\
y'(t) &= -x'(t), \\
\lambda'(t) &= \tfrac{c}{n}\left\{g'(t)I_4(t) + g(t)I_4'(t) + g'(t)I_3(t) + g(t)I_3'(t)\right\}, \\
I_1'(t) &= f_1(t) - f_1(t-D), \\
I_2'(t) &= f_2(t) - f_2(t-D), \\
I_3'(t) &= f_3(t) - f_3(t-D), \\
I_4'(t) &= f_2(t)I_1(t) - f_1(t-D)I_1(t),
\end{aligned}
$$

where

$$
g(t) = e^{-Gt}\ ,\ f_1(t) = 1\ ,\ f_2(t) = e^{Gt}x(t)\lambda(t)\ ,\ f_3(t) = e^{Gt}y(t),
$$

and

$$
c = 0.5\ ,\ n = 100\ ,\ G = 1\ , \text{and } D = 5
$$

for t in $[0, 4D]$. The initial conditions are,

$$x(0) = 0.8n,$$
$$y(0) = 0.2n,$$
$$\lambda(0) = 0,$$
$$I_1(0) = 0,$$
$$I_2(0) = 0,$$
$$I_3(0) = 0,$$
$$I_4(0) = 0.$$

In our tests we use a dimension 6 equivalent formulation since $I_1(t)$ can be computed analytically. This problem is solved by defining an event when t crosses D and then changing the problem formulation by changing a variable that reflects whether $t \geq D$. The exact solution of this problem is unknown. See Figure (3.9) for an accurate approximate solution.

3.5 Problems that have Special Histories

3.5.1 Example 10

This example is a model of hematopoiesis and is called Example C3 in reference [7]. The problem is defined by

$$y_1'(t) = \hat{s}_0 y_2(t - T_1) - \gamma y_1(t) - Q,$$
$$y_2'(t) = f(y_1(t)) - k y_2(t),$$
$$y_3'(t) = 1 - \frac{Q e^{\gamma y_3(t)}}{\hat{s}_0 y_2(t - T_1 - y_3(t))},$$

for t in $[0, 300]$. The history function is

$$\phi_1(0) = 3.325,$$
$$\phi_2(t) = \begin{cases} 10 & \text{for} \quad -T_1 \leq t \leq 0, \\ 9.5 & \text{for} \quad t < -T_1, \end{cases}$$
$$\phi_3(0) = 120,$$

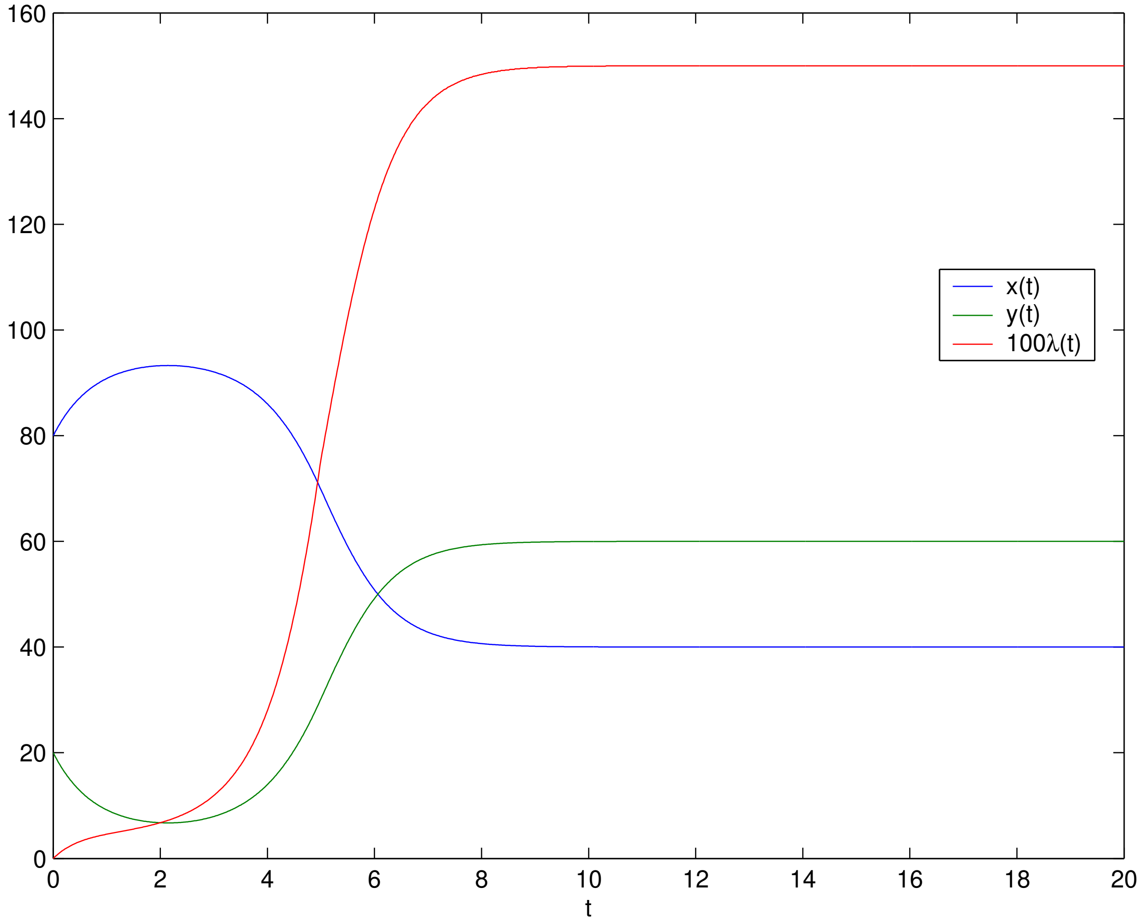

Figure 3.9: Plot of numerical solution to Example 9, computed using DDVERK90 with RE=10^{-10}, AE=10^{-6}.

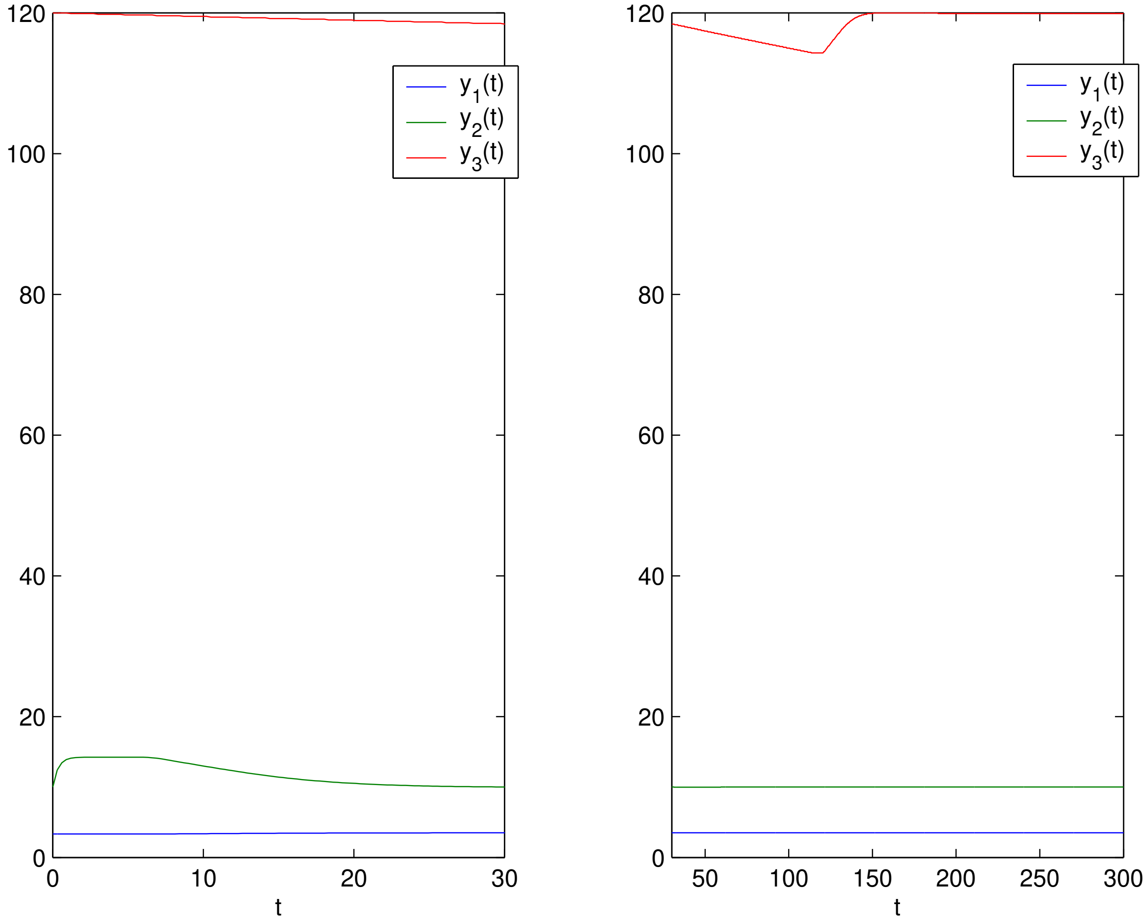

Figure 3.10: Plot of numerical solution to Example 10, computed using DDVERK90 with RE=AE=10^{-6}.

where $f(y) = \frac{a}{1+Ky^r}$, $\hat{s}_0 = 0.0031$, $T_1 = 6$, $\gamma = 0.001$, $Q = 0.0275$, $k = 2.8$, $a = 6570$, $K = 0.0382$, $r = 6.96$.

The history has a discontinuity at $-T_1$. Since DDVERK90 uses automatic detection of discontinuities, users do not need to do anything special to deal with this. The exact solution of this problem is unknown. See Figure (3.10) for an accurate approximate solution.

3.6 Non-Standard Problems

3.6.1 Example 11

This example shows how to solve a state dependent Volterra integral equation by embedding a quadrature component in an *extended* DDE. The problem is studied in references [21] and [16], and is defined by

$$y'(t) = \frac{\int_{ty(t)}^{t^2 y(t)} s^3 y(s) y'(s) ds \ - 1}{t^3},$$

for t in $[1, 2]$. The history function is

$$y(t) = 1 \ \text{ for } \ t = 1.$$

The exact solution (see Figure (3.11)) is

$$y = \frac{1}{t}.$$

A conventional method can solve this problem by introducing an extra component of the ODE to correspond to the integral that appears in the problem definition and then viewing this extended system to be an equivalent system of DDEs.

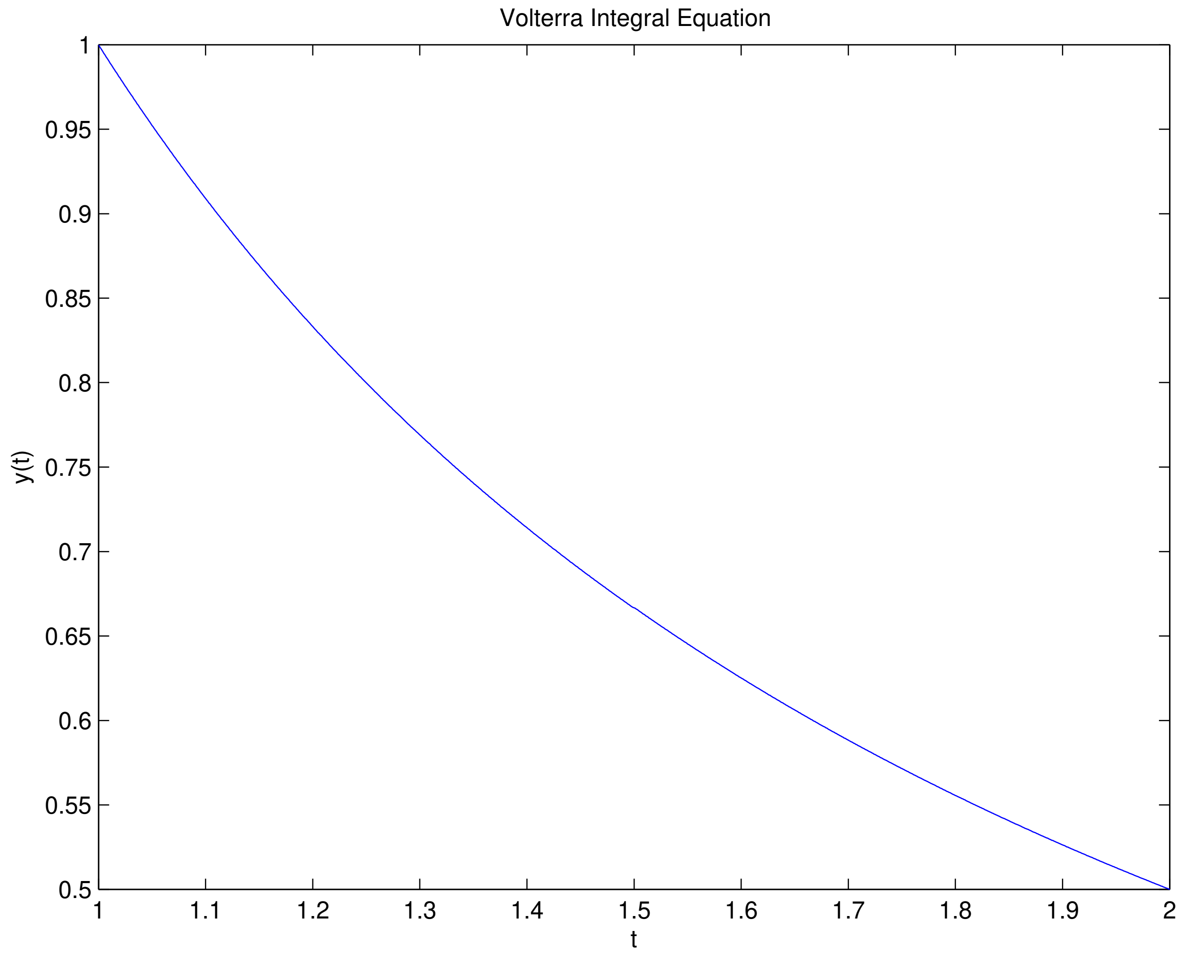

Figure 3.11: Plot of numerical solution to Example 11, computed using DDVERK90 with RE=AE=10^{-6}.

3.6.2 Example 12

This example is an epidemic model that is studied in references [15] and [22]. The problem is defined by

$$\tau'(t) = \begin{cases} \frac{\rho(t)I(t)}{\rho(\tau(t))I(\tau(t))} & \text{for} \quad t_0 < t, \\ 0 & \text{for} \quad t \le t_0, \end{cases}$$

$$S'(t) = -r(t)I(t)S(t),$$

$$y_3'(t) = \rho(t)I_0(t).$$

where t_0 is the point where $y_3(t) = m$, and

$$I(t) = \begin{cases} I_0(t) & \text{for} \quad -\sigma \le t \le t_0, \\ I_0(t) + S_0 - S(\tau(t)) & \text{for} \quad t_0 \le t \le t_0 + \sigma, \\ S(\tau(t-\sigma)) - S(\sigma(t)) & \text{for} \quad t_0 + \sigma \le t, \end{cases}$$

$$m = 0.1, \ \sigma = 1, \ S_0 = 10,$$

$$I_0(t) = \begin{cases} 0.4(1+t) & \text{for} \quad -1 \le t \le 0, \\ 0.4(1-t) & \text{for} \quad 0 \le t \le 1, \\ 0 & \text{otherwise}, \end{cases}$$

$$\rho(t) = 1, \ r(t) = r_0, \qquad \text{for example 12a;}$$

$$\rho(t) = e^{-t}, \ r(t) = r_0(1+\sin(5t)), \quad \text{for example 12b.}$$

For both cases the history function is

$$\tau(t) = 0,$$

$$S(t) = S_0,$$

$$y_3(t) = 0,$$

for $t \le 0$.

It is solved for t in $[0, 8]$.

Each version of the problem (12a and 12b) is solved for the four cases $r_0 = 0.2, 0.3, 0.4, 0.5$.

Due to the special definitions of $\tau'(t)$ and $I(t)$, this problem is also a problem that changes after events. Here the events are $y_3(t) = m$ and $t = t_0 + \sigma$.

To evaluate the derivatives, approximations are required for the quantities $S(\tau(t-\sigma))$, $S(\tau(t))$, $S(\tau(\tau(t)-\sigma))$, and $S(\tau(\tau(t)))$. The Z array supplied to subroutine DDES by DDVERK90 and the interpolation subroutine DDVERK_USER are used to generate these approximations when they are needed.

The exact solution of this problem is unknown. See Figures (3.12), (3.13), (3.14), (3.15) for an accurate approximate solution.

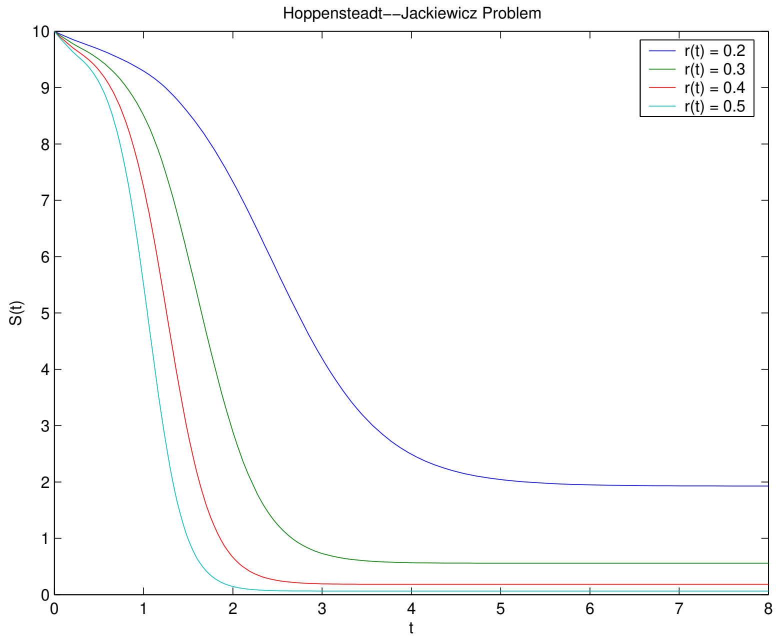

Figure 3.12: Plot of numerical solution to S(t) of Example 12a, computed using DDVERK90 with RE=AE=10^{-8}.

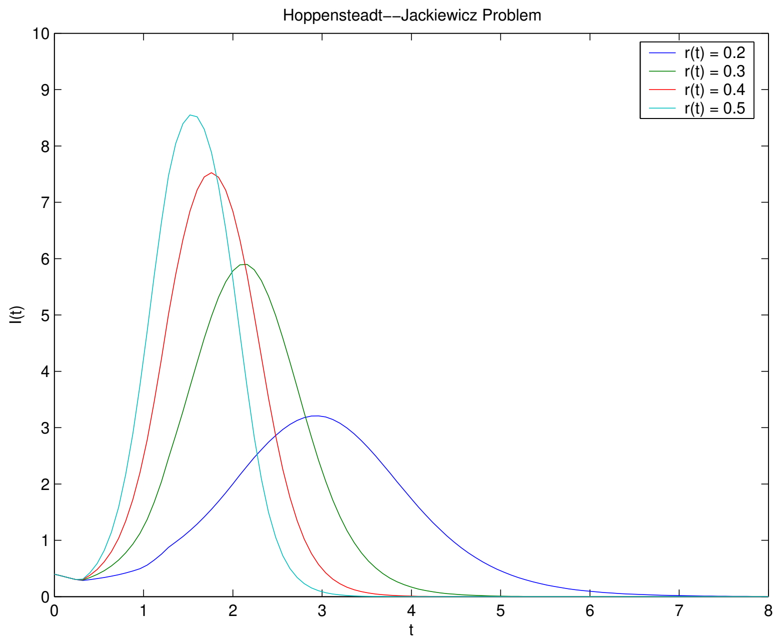

Figure 3.13: Plot of numerical solution to I(t) of Example 12a, computed using DDVERK90 with RE=AE=10^{-8}.

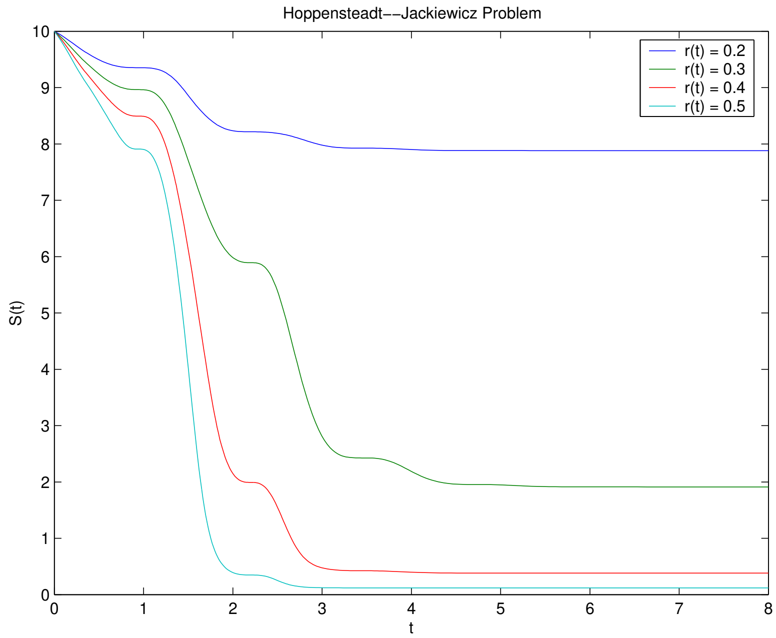

Figure 3.14: Plot of numerical solution to S(t) of Example 12b, computed using DDVERK90 with RE=AE=10^{-8}.

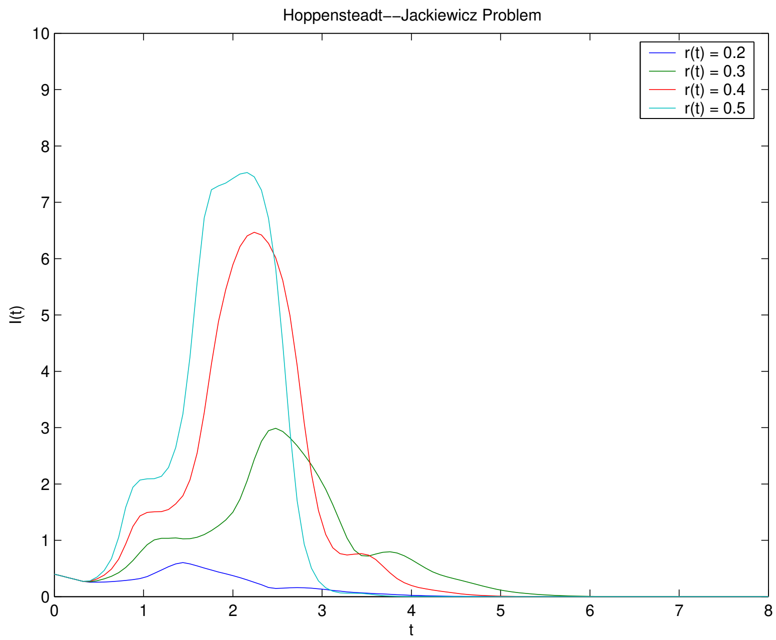

Figure 3.15: Plot of numerical solution to I(t) of Example 12b, computed using DDVERK90 with RE=AE=10^{-8}.

Chapter 4

More Examples and Some Difficulties

DDVERK90 is an extension and improved version of DDVERK, which is based on an underlying explicit Runge-Kutta formula. Therefore it is not suitable for stiff problems. It is accepted practice that for solving stiff DDEs, DDE methods based on formulas that are suitable for stiff IVPs should be used (for example RADAR5 [11]). It may nevertheless be useful to know how a nonstiff DDE solver such as DDVERK90 performs on stiff problems. This is because it is sometimes hard to identify whether a problem is stiff without a comprehensive investigation. In this chapter we try to show the behaviour of DDVERK90 when it tries to solve a stiff problem by presenting four examples.

4.1 Stiff Problems That Can Be Solved

4.1.1 Example 13

This example is studied in reference [8] and is called example (d) in reference [10]. The problem is defined by

$$y_1'(t) = kM1\,Ay_2(t) - kM2y_1(t)y_2(t-\tau) + kM3\,By_1(t) - 2kM4y_1(t)^2,$$
$$y_2'(t) = -kM1\,Ay_2(t) - kM2y_1(t)y_2(t-\tau) + fr\,kM3y_1(t),$$

where,

$$kM1 = 1.34,$$
$$kM2 = 1.6 \cdot 10^9,$$
$$kM3 = 8.0 \cdot 10^3,$$
$$kM4 = 4.0 \cdot 10^7,$$
$$kM5 = 1.0,$$
$$fr = 1.0,$$
$$A = 6.0 \cdot 10^{-2},$$
$$B = 6.0 \cdot 10^{-2},$$
$$\tau = 0.15,$$

for t in $[0, 100.5]$. The history functions are $y_1(t) = 10^{-10}$ and $y_2(t) = 10^{-5}$ for $t \leq 0$.

With error tolerances set to RE=10^{-9} and AE=10^{-18} it takes 118,136 steps to solve the problem with DDVERK90. The plot of the solution (see Figure (4.1)) shows the source of the stiffness of the problem. As the graph shows, both solution components (y_1 and y_2) have transient regions which arise periodically. This forces the solver to require very small step sizes to ensure numerical stability. RADAR5 solves this problem in 9072 steps.

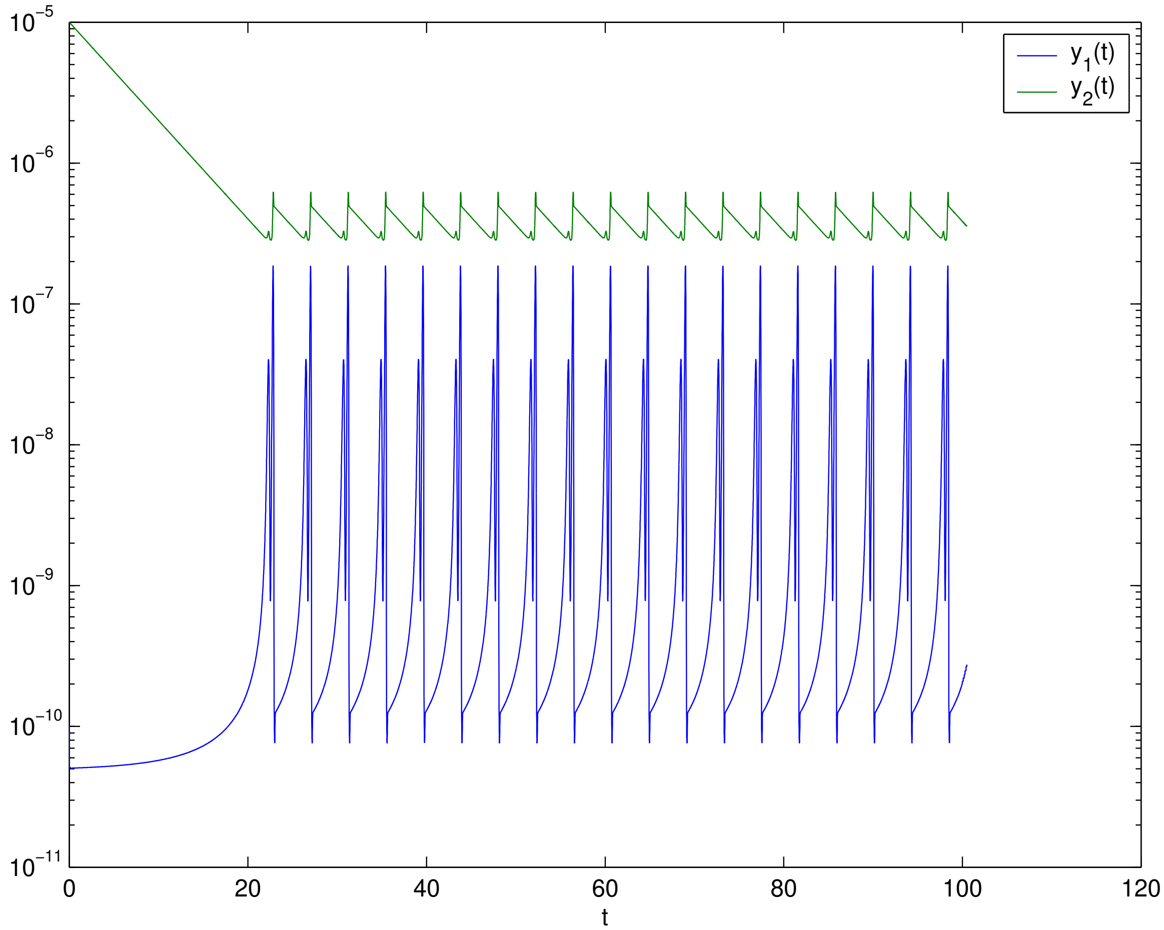

Figure 4.1: Plot of numerical solution to Example 13, computed using DDVERK90.

4.1.2 Example 14

This example is a threshold model for antibody production that first studied in reference [23] and is used as an example in reference [11]. The problem is defined by,

$$
\begin{aligned}
y_1'(t) &= -ry_1(t)y_2(t) - sy_1(t)y_4(t),\\
y_2'(t) &= -ry_1(t)y_2(t) + \alpha r y_1(y_5(t))y_2(y_5(t))H(t-t_0),\\
y_3'(t) &= ry_1(t)y_2(t),\\
y_4'(t) &= -sy_1(t)y_4(t) - \gamma y_4(t) + \beta r y_1(y_6(t))y_2(y_6(t))H(t-t_1),\\
y_5'(t) &= H(t-t_0)f_1(y_1(t),y_2(t),y_3(t))/f_1(y_1(y_5(t)),y_2(y_5(t)),y_3(y_5(t))),\\
y_6'(t) &= H(t-t_1)f_2(y_2(t),y_3(t))/f_2(y_2(y_6(t)),y_3(y_6(t))),
\end{aligned}
$$

where $\alpha = 1.8$, $\beta = 20$, $\gamma = 0.002$, $r = 5 \cdot 10^4$, $s = 10^5$, $t_0 = 35$, $t_1 = 197$. $H(x)$ is the Heavyside function ($H(x) = 0$ if $x < 0$ and $H(x) = 1$ if $x \geq 0$); $f_1(x, y, w) = xy + w$ and $f_2(y, w) = 10^{-12} + y + w$,
for t in $[0, 300]$. The history functions are $y_1(t) = 5 \cdot 10^{-6}$, $y_2(t) = 10^{-15}$, and $y_3(t) = y_4(t) = y_5(t) = y_6(t) = 0$ for $t \leq 0$.

This problem has several features that cause difficulties for the solver: First the delay becomes very small and vanishes asymptotically. Ans Second, the solution components y_2 and y_4, y_6 exhibit transient behaviour near the values $t = 35$ and $t = 197$, respectively.

With RE$< 10^{-4}$ DDVERK90 fails to solve the problem. The solver cannot proceed after a few steps, because it cannot satisfy the error tolerance.
With RE=10^{-4} and AE_VECTOR=$[10^{-14}, 10^{-14}, 10^{-14}, 10^{-14}, 10^{-7}, 10^{-7}]$, DDVERK90 takes 324 steps to solve the problem.

After comparing the solution (see Figures (4.2), (4.3) for DDVERK90 results) with the solution of a stiff DDE solver(for example RADAR5) we saw that the two solutions have the same behaviour, even though our solution is not reliable if the user needs an accurate solution. In this problem the equations change after events. RADAR5 solves this problem in 465 steps.

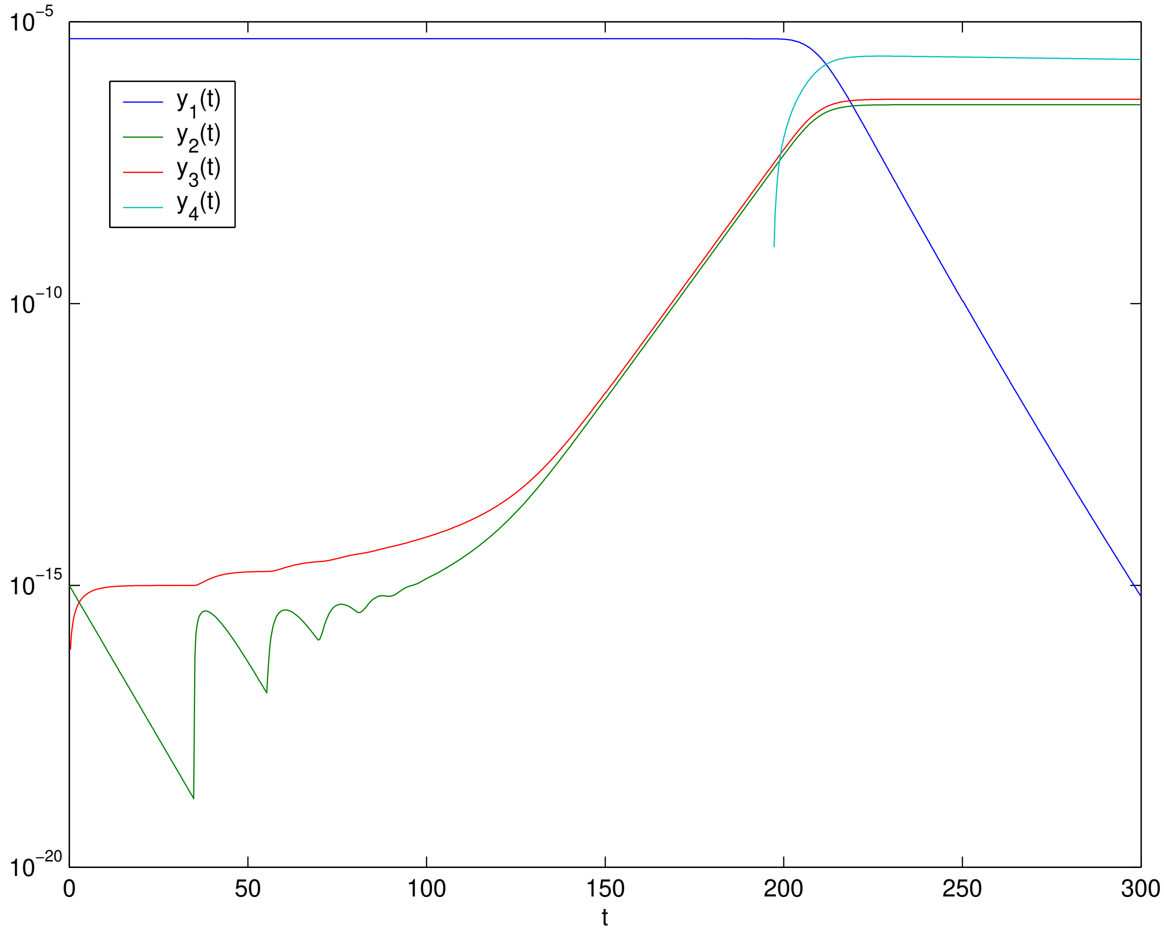

Figure 4.2: Plot of numerical solution to $y_1(t), y_2(t), y_3(t), y_4(t)$ of Example 14, computed using DDVERK90.

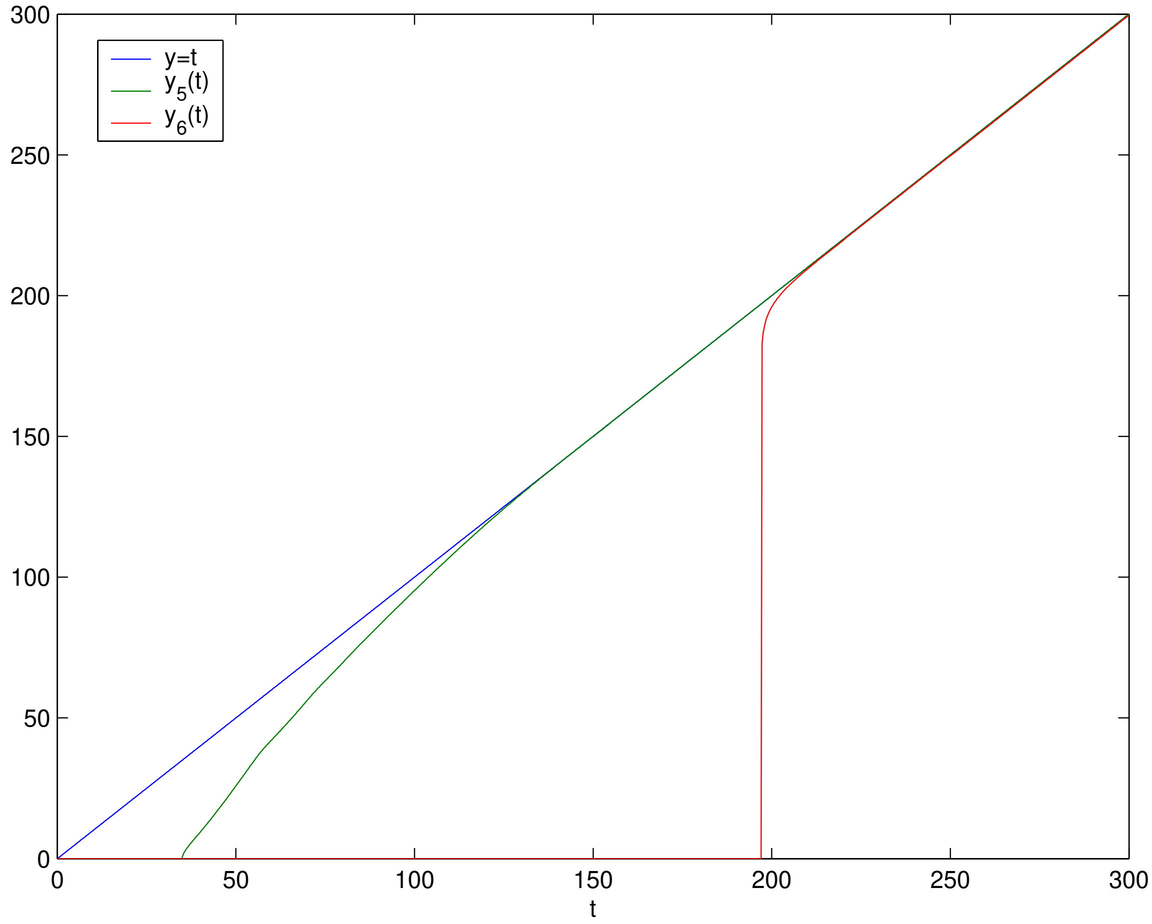

Figure 4.3: Plot of numerical solution to $y_5(t), y_6(t)$ of Example 14, computed using DDVERK90.

4.2 Very Stiff Problems

4.2.1 Example 15

This example is called Example 7.11 in reference [1] and example (c) in reference [10]. It corresponds to a mathematical model of antiviral immune response quantitatively describing (with the parameters of Table 4.1) the dynamics of hepatitis B virus infection over a 130 days interval. The problem is defined by

$$\begin{aligned}
y_1' &= \alpha_1 y_2 + \alpha_2\alpha_3 y_2 y_7 - \alpha_4 y_1 y_{10} - \alpha_5 y_1 - \alpha_6 y_1(\alpha_7 - y_2 - y_3),\\
y_2' &= \alpha_8 y_1(\alpha_7 - y_2 - y_3) - \alpha_3 y_2 y_7 - \alpha_9 y_2,\\
y_3' &= \alpha_3 y_2 y_7 + \alpha_9 y_2 - \alpha_{10} y_3, \quad \xi(y_3) = 1 - y_3/\alpha_7,\\
y_4' &= \alpha_{11}\alpha_{12} y_1 - \alpha_{13} y_4,\\
y_5' &= \alpha_{14}\left[\xi(y_3)\alpha_{15} y_4(t-\tau_1) y_5(t-\tau_1) - y_4 y_5\right] - \alpha_{16} y_4 y_5 y_7 + \alpha_{17}(\alpha_{18} - y_5),\\
y_6' &= \alpha_{19}\left[\xi(y_3)\alpha_{20} y_4(t-\tau_2) y_6(t-\tau_2) - y_4 y_6\right] - \alpha_{21} y_4 y_6 y_8 + \alpha_{22}(\alpha_{23} - y_6),\\
y_7' &= \alpha_{24}\left[\xi(y_3)\alpha_{25} y_4(t-\tau_3) y_5(t-\tau_3) y_7(t-\tau_3) - y_4 y_5 y_7\right]\\
&\quad -\alpha_{26} y_2 y_7 + \alpha_{27}(\alpha_{28} - y_7),\\
y_8' &= \alpha_{29}\left[\xi(y_3)\alpha_{30} y_4(t-\tau_4) y_6(t-\tau_4) y_8(t-\tau_4) - y_4 y_6 y_8\right] + \alpha_{31}(\alpha_{32} - y_8),\\
y_9' &= \alpha_{33}\xi(y_3)\alpha_{34} y_4(t-\tau_5) y_6(t-\tau_5) y_8(t-\tau_5) + \alpha_{35}(\alpha_{36} - y_9),\\
y_{10}' &= \alpha_{37} y_9 - \alpha_{38} y_{10} y_1 - \alpha_{39} y_{10},
\end{aligned}$$

with history $y_1(t) = 2.9 \cdot 10^{-16}$, $y_2(t) = 0$, $y_3(t) = 0$, $y_4(t) = 0$, $y_5(t) = \alpha_{18}$, $y_6(t) = \alpha_{23}$, $y_7(t) = \alpha_{28}$, $y_8(t) = \alpha_{32}$, $y_9(t) = \alpha_{36}$, $y_{10}(t) = \frac{\alpha_{37}\alpha_{36}}{\alpha_{39}}$, for $t \le 0$. The exact solution of this problem is unknown.

A challenging feature of the solution of this problem is the considerable variation in solution magnitude over the 130 days interval. The stiffness of the problem increases sharply when the time passes from 110 to 120 days.

If we run DDVERK90 with RE=10^{-11} and AE=10^{-31} we see that the code runs from $t = 0$ to $t = 97$ very quickly, then goes from $t = 97$ to $t = 114$ very slowly. After $t = 114$, the stepsize is so small that we need about 100,000 steps for every unit of time. This

α_1	83	α_2	5	α_3	$6.6 \cdot 10^{14}$	α_4	$3 \cdot 10^{11}$	α_5	0.4
α_6	$2.5 \cdot 10^{7}$	α_7	$0.5 \cdot 10^{-12}$	α_8	$2.3 \cdot 10^{9}$	α_9	0.052	α_{10}	0.15
α_{11}	$9.4 \cdot 10^{9}$	α_{12}	10^{-15}	α_{13}	1.2	α_{14}	$2.7 \cdot 10^{16}$	α_{15}	2
α_{16}	$5.3 \cdot 10^{27}$	α_{17}	1.0	α_{18}	10^{-18}	α_{19}	$2.7 \cdot 10^{16}$	α_{20}	2
α_{21}	$8 \cdot 10^{28}$	α_{22}	1.0	α_{23}	10^{-19}	α_{24}	$5.3 \cdot 10^{33}$	α_{25}	16
α_{26}	$1.6 \cdot 10^{14}$	α_{27}	0.4	α_{28}	10^{-18}	α_{29}	$8 \cdot 10^{32}$	α_{30}	16
α_{31}	0.1	α_{32}	10^{-18}	α_{33}	$1.7 \cdot 10^{30}$	α_{34}	3	α_{35}	0.4
α_{36}	$4.3 \cdot 10^{-22}$	α_{37}	$0.85 \cdot 10^{7}$	α_{38}	$8.6 \cdot 10^{11}$	α_{39}	0.043		
τ_1	0.6	τ_2	0.6	τ_3	2.0	τ_4	2.0	τ_5	3.0

Table 4.1: Parameters for problem 15 corresponding to acute hepatitis B virus infection

makes the problem unsolvable using a reasonable amount of time and memory.

One of the features of DDVERK90 is that users are able to monitor the progress of the solver (the simplest way is to monitor the progress of t by setting SHOW_PROGRESS optional argument to .TRUE.) and see the behaviour of the solution. If a problem behaves like this example then it is probably stiff and this suggests that the user try a stiff DDE solver to find the solution. We include here the solution plots for $t_F = 114.5$ (see Figure (4.4)). RADAR5 solves this problem in 2208 steps.

4.2.2 Example 16

This example is the Robertson problem [13, Section IV.10]. This problem is one of the most widely used test problems for stiff IVP methods. We consider here the following modified version of the problem:

$$\begin{aligned} y_1'(t) &= -0.04y_1(t) + 10^4 y_2(t-\tau)y_3(t), \\ y_2'(t) &= 0.04y_1(t) - 10^4 y_2(t-\tau)y_3(t) - 3 \cdot 10^7 y_2(t)^2, \\ y_3'(t) &= 3 \cdot 10^7 y_2(t)^2, \end{aligned}$$

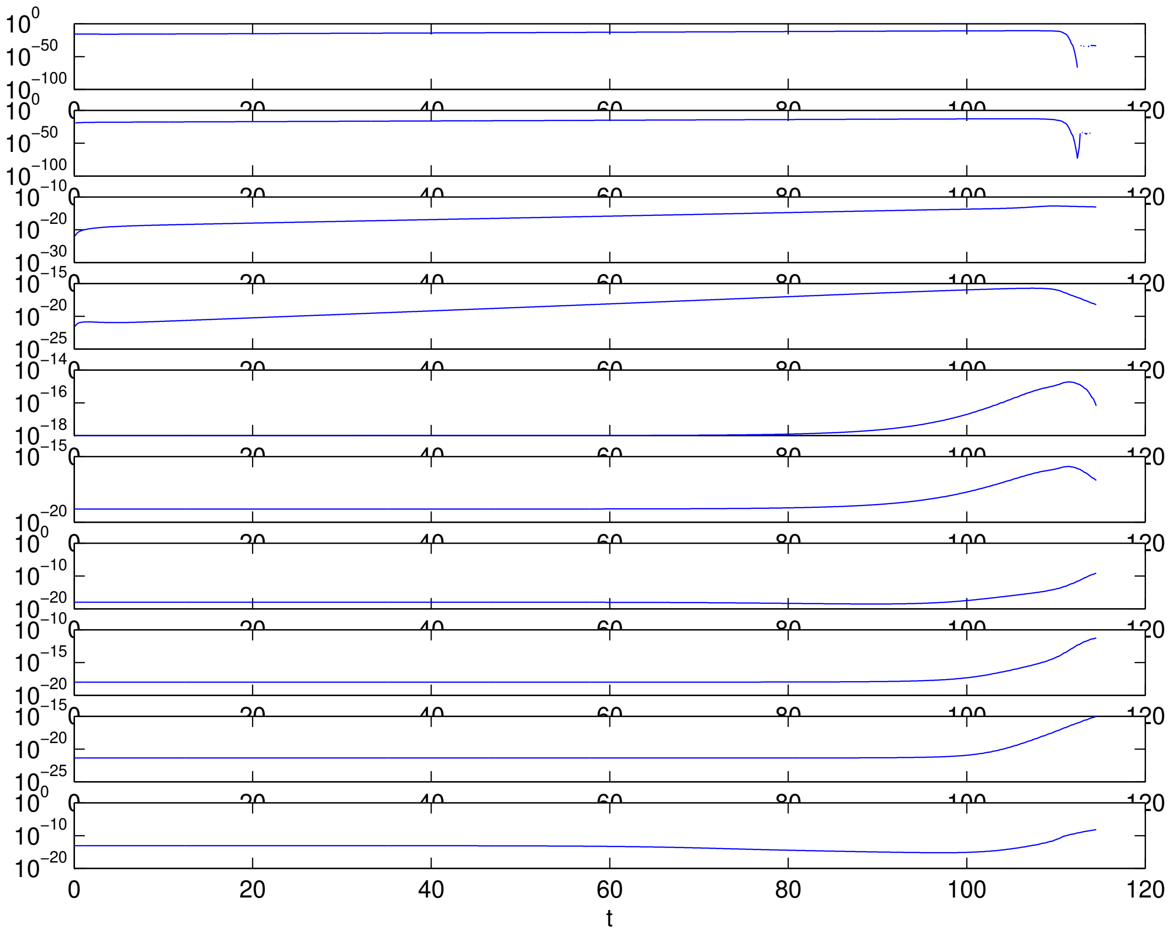

Figure 4.4: Semilogarithmic plot of numerical solution to $y_1(t), \cdots, y_{10}(t)$ (from top to bottom) of Example 15

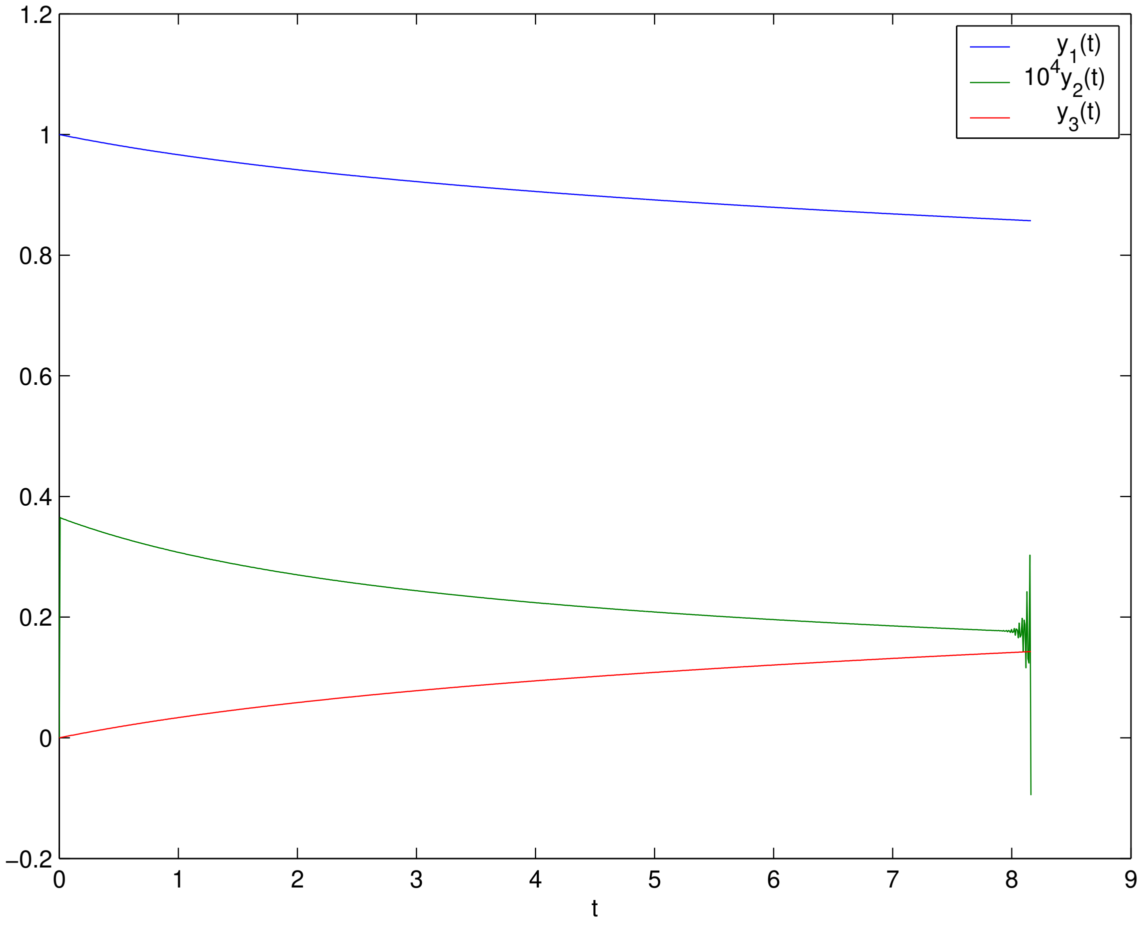

Figure 4.5: Plot of numerical solution to Example 16 with $\tau = 0.01$, computed using DDVERK90. The oscillation and explosion are the artifacts of stiffness.

for t in $[0, 10^5]$. With history $y_1(t) = 1$, $y_2(t) = 0$, $y_3(t) = 0$, for $t \leq 0$.

The exact solution of this problem is unknown. We set RE=10^{-9} and AE=10^{-14}.

With $\tau = 0.01$ and $\tau = 0.03$ the solver almost stops at $t = 8.16$ and $t = 9.24$, respectively. This appears to be due to the severe stepsize restriction imposed in DDVERK90 by numerical stability.

We include here the solution plots for $\tau = 0.01$, $t_F = 8.16$ and $\tau = 0.03$, $t_F = 9.24$ (see Figures (4.5), (4.6)). RADAR5 solves this problem in 702 steps.

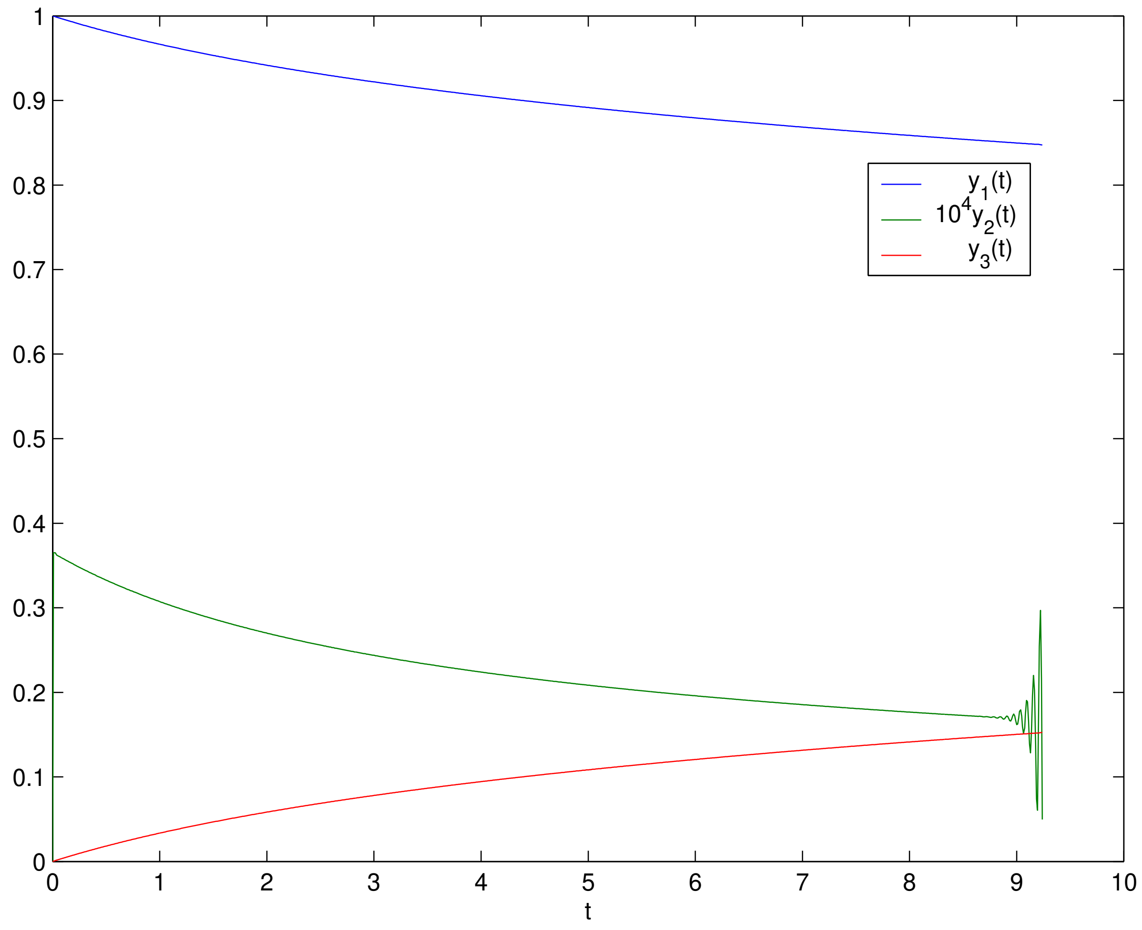

Figure 4.6: Plot of numerical solution to Example 16 with $\tau = 0.03$, computed using DDVERK90. The oscillation and explosion are the artifacts of stiffness.

Chapter 5

Numerical Results

In this chapter we present the detailed numerical results of DDVERK90 for the problems that are described in Chapter 3. We have also included the results from DDE_SOLVER for the comparison.

5.1 A description of Computations

We used DDVERK90 with the default one point sampling defect estimate and DDE_SOLVER with tracking discontinuities except for Example 12. We have used AE=RE=TOL with TOL=10^{-3},10^{-5},10^{-7} for both solvers. For a problem where the exact solution is unknown we have used the numerical solution computed with AE=RE=10^{-9} as the exact solution in global error calculations. The error overrun is defined by

$$\frac{||\text{Error in Computed Solution}||}{||\text{Exact Solution}|| \times \text{RE} + \text{AE}}$$

and the expected value of the error overrun is the condition number of the problem (see reference [4] for more details). The norm that is used is the infinity norm.

5.2 Numerical Results

The statistics we report in tables are as follows :

FCN: The number of derivative evaluations.

STEPS: The number of successful time steps.

GE_END: The global error overrun at the endpoint t_F.

GE_MESH: The maximum of the global error overrun at mesh points.

GE_CONT: The maximum of the global error overrun across the range of integration, computed using 1000 equally spaced points.

PROB	TOL	FCN	STEPS	GE_END	GE_MESH	GE_CONT
1	10^{-3}	2237	147	$8.7021e+01$	$1.6982e+02$	$1.6982e+02$
	10^{-5}	4331	304	$1.9470e+03$	$2.7003e+03$	$2.7003e+03$
	10^{-7}	8885	640	$8.5296e+02$	$1.2316e+03$	$1.2316e+03$
2	10^{-3}	1253	22	$8.5358e-02$	$8.5358e-02$	$8.5358e-02$
	10^{-5}	1336	24	$6.6731e+00$	$6.6731e+00$	$6.6731e+00$
	10^{-7}	2731	47	$1.1220e+01$	$1.1220e+01$	$1.1220e+01$
3	10^{-3}	151	10	$9.0137e-02$	$9.0137e-02$	$9.4694e-02$
	10^{-5}	217	14	$1.9170e-01$	$2.2374e-01$	$3.0492e-01$
	10^{-7}	326	21	$1.1146e+00$	$1.1146e+00$	$1.1146e+00$
4	10^{-3}	5209	237	$2.3109e-03$	$4.9000e-02$	$4.9000e-02$
	10^{-5}	5309	240	$6.8651e-03$	$4.3023e-02$	$4.3507e-02$
	10^{-7}	5641	247	$1.6968e-02$	$3.0524e-01$	$3.0524e-01$
5	10^{-3}	105	5	$2.8125e-01$	$2.8125e-01$	$2.8125e-01$
	10^{-5}	253	9	$4.8225e-03$	$2.9300e-02$	$4.1786e-02$
	10^{-7}	308	15	$1.0027e-01$	$2.4917e-01$	$2.5027e-01$
6	10^{-3}	1022	55	$2.3043e-01$	$3.8237e+00$	$4.0793e+00$
	10^{-5}	1607	99	$2.7379e+00$	$5.0906e+01$	$5.0932e+01$
	10^{-7}	2519	194	$3.6339e+00$	$6.4556e+01$	$6.4616e+01$

Table 5.1: Summary Statistics for DDVERK90 (Problems 1 to 6)

PROB	TOL	FCN	STEPS	GE_END	GE_MESH	GE_CONT
	10^{-3}	505	16	$9.9031e-01$	$2.3315e+00$	$4.4905e+01$
7	10^{-5}	790	30	$4.6285e-01$	$2.1985e+02$	$3.5443e+03$
	10^{-7}	1424	58	$2.3008e-01$	$3.2682e+03$	$5.1837e+04$
	10^{-3}	1462	74	$5.0307e+02$	$3.1554e+03$	$3.1607e+03$
8	10^{-5}	1923	101	$3.2223e+03$	$2.3349e+04$	$2.4190e+04$
	10^{-7}	2567	168	$5.0869e+03$	$4.5083e+04$	$4.5319e+04$
	10^{-3}	420	38	$3.4723e-02$	$3.9068e-01$	$4.1287e-01$
9	10^{-5}	794	73	$3.4780e-02$	$4.2059e-01$	$4.2762e-01$
	10^{-7}	1692	149	$4.4000e-02$	$4.2819e-01$	$4.3467e-01$
	10^{-3}	2568	219	$1.2759e-03$	$2.4334e-02$	$2.4334e-02$
10	10^{-5}	2989	254	$2.2988e-02$	$3.3531e-02$	$3.3531e-02$
	10^{-7}	4030	343	$8.2691e-03$	$3.8427e-02$	$3.8427e-02$
	10^{-3}	253	12	$1.8627e+00$	$1.8627e+00$	$1.8627e+00$
11	10^{-5}	299	14	$3.4241e+00$	$3.4241e+00$	$3.4241e+00$
	10^{-7}	437	20	$2.0367e+00$	$2.0367e+00$	$2.0367e+00$
	10^{-3}	525	22	$2.1566e-01$	$2.3438e-01$	$1.3474e+00$
12	10^{-5}	750	44	$1.7685e-02$	$4.3645e-01$	$2.0475e+00$
	10^{-7}	1161	78	$6.6851e-02$	$8.3924e-01$	$9.9328e-01$

Table 5.2: Summary Statistics for DDVERK90 (Problems 7 to 12)

PROB	TOL	FCN	STEPS	GE_END	GE_MESH	GE_CONT
	10^{-3}	2214	174	$2.4604e+02$	$2.6014e+02$	$2.6905e+02$
1	10^{-5}	4365	367	$2.0829e+02$	$3.0197e+02$	$3.0197e+02$
	10^{-7}	9144	796	$1.8431e+02$	$2.6615e+02$	$2.6615e+02$
	10^{-3}	1215	46	$5.6367e-01$	$5.6367e-01$	$6.0363e-01$
2	10^{-5}	1899	68	$4.2568e-01$	$4.2568e-01$	$4.4718e-01$
	10^{-7}	3690	118	$1.5908e-01$	$1.6226e-01$	$1.8592e-01$
	10^{-3}	189	19	$2.1370e-03$	$2.1370e-03$	$2.1990e-03$
3	10^{-5}	297	23	$7.3329e-02$	$7.3329e-02$	$1.2491e-01$
	10^{-7}	558	43	$1.2088e-01$	$1.2088e-01$	$1.7330e-01$
	10^{-3}	3051	140	$3.1211e-02$	$3.9570e-02$	$8.4170e-02$
4	10^{-5}	4041	175	$3.1626e-02$	$4.3086e-02$	$1.0654e-01$
	10^{-7}	6210	271	$1.0238e-02$	$3.1123e-02$	$9.2676e-02$
	10^{-3}	153	10	$2.5984e-04$	$5.8956e-04$	$7.0918e-04$
5	10^{-5}	189	11	$2.3493e-02$	$3.8028e-02$	$5.4940e-02$
	10^{-7}	405	17	$3.6536e-03$	$5.1999e-02$	$1.1898e-01$
	10^{-3}	1575	121	$1.1072e-02$	$2.1434e-01$	$2.1434e-01$
6	10^{-5}	1890	172	$2.2801e-02$	$3.9704e-01$	$4.0210e-01$
	10^{-7}	2943	308	$3.1036e-02$	$6.5388e-01$	$6.6070e-01$

Table 5.3: Summary Statistics for DDE_SOLVER (Problems 1 to 6)

PROB	TOL	FCN	STEPS	GE_END	GE_MESH	GE_CONT
	10^{-3}	819	42	$5.7175e-02$	$1.0769e-01$	$1.0769e-01$
7	10^{-5}	1494	64	$5.3207e-01$	$2.2847e+00$	$2.2847e+00$
	10^{-7}	1845	98	$1.1128e+00$	$3.4620e+00$	$3.5531e+00$
	10^{-3}	-	-	-	-	-
8	10^{-5}	3771	299	$8.1477e+02$	$7.5433e+03$	$7.5537e+03$
	10^{-7}	4545	404	$3.0163e+02$	$2.6741e+03$	$2.6776e+03$
	10^{-3}	387	43	$9.4164e-03$	$4.4483e-02$	$5.4647e-02$
9	10^{-5}	765	84	$1.2840e-02$	$8.7952e-02$	$8.8917e-02$
	10^{-7}	1521	169	$1.2000e-02$	$1.3200e-01$	$1.3582e-01$
	10^{-3}	10467	968	$3.9280e-03$	$3.7049e-02$	$4.1426e-02$
10	10^{-5}	11106	1013	$5.5091e-03$	$4.1209e-02$	$4.6787e-02$
	10^{-7}	14247	1360	$4.1345e-02$	$5.2034e-02$	$5.2034e-02$
	10^{-3}	180	10	$8.5227e-04$	$8.5227e-04$	$8.5227e-04$
11	10^{-5}	180	10	$8.4980e-02$	$8.4980e-02$	$8.4980e-02$
	10^{-7}	198	11	$6.9867e+00$	$6.9867e+00$	$6.9867e+00$
	10^{-3}	567	29	$4.8361e-03$	$1.2273e-01$	$2.0365e+00$
12	10^{-5}	972	54	$6.9282e-03$	$3.3708e-01$	$6.8098e-01$
	10^{-7}	1512	104	$7.9006e-02$	$2.1262e-01$	$4.9217e+00$

Table 5.4: Summary Statistics for DDE_SOLVER (Problems 7 to 12)

5.3 Discussion

From the FCN values, we observe that DDVERK90 needs fewer derivative evaluations than DDE_SOLVER for most of the problems. For Problem 10, DDVERK90 needs less than one-third the number of derivative evaluations than the number required by DDE_SOLVER. The STEPS values show that the number of time steps for DDVERK90 is less than that of the DDE_SOLVER.

GE_END, GE_MESH and GE_CONT are comparable for all problems and for all tolerances. DDE_SOLVER fails to solve Problem 8 with TOL=10^{-3}.

The high error overruns for Problem 1, 7 and 8 are due to the poor mathematical conditioning of these problems.

Chapter 6

Conclusion

6.1 Summary

We introduced a new solver DDVERK90 that has a new hierarchical interface that can be used to solve a wide range of problems from those that have a simple structure constant delay, and constant history to those that have events and a formulation that changes after each event. As we showed it is able to deal with NDE problems with some minor modifications. As we also showed the new interface is more intuitive and less likely to be misinterpreted.

With the classification that we have made and the sample problems that we have solved, it is easy to match a new problem to one of the classes and use the driver that is available for the similar problem as a template.

We have carried out extensive numerical experiments over various types of DDEs and compared the performance with DDE_SOLVER in terms of cost and reliability. The results show that DDVERK90 is generally less expensive and more reliable.

We also studied the behaviour of our solver when the problem is stiff and showed that, in these cases, one is able to detect stiffness.

6.2 Future Work

Although we have tried to test DDVERK90 over a wide range of sample problems, testing the reliability and robustness for other problems and identifying the situations in which DDVERK90 may not be appropriate is one of the areas for future investigations.

In DDVERK90, as we have mentioned, there is a mechanism that helps users to detect stiffness in the problem. Developing an automatic stiffness detection is also a subject for future research.

Appendix A

Sample Driver to Solve Example 6

```
MODULE define_DDEs
  IMPLICIT NONE
  INTEGER, PARAMETER :: NEQN=3,NU=2,NEF=1
CONTAINS

  SUBROUTINE DDES(T,Y,Z,DY)
    DOUBLE PRECISION :: T
    DOUBLE PRECISION, DIMENSION(NEQN) :: Y,DY
    DOUBLE PRECISION, DIMENSION(NEQN,NU) :: Z
    INTENT(IN) :: T,Y,Z
    INTENT(OUT) :: DY
    DY(1) = -Y(1) * Z(2,1) + Z(2,2)
    DY(2) =  Y(1) * Z(2,1) - Y(2)
    DY(3) =  Y(2) - Z(2,2)
    RETURN
  END SUBROUTINE DDES
```

```
  SUBROUTINE EF(T,Y,DY,Z,G)
     DOUBLE PRECISION :: T
     DOUBLE PRECISION, DIMENSION(NEQN) :: Y,DY
     DOUBLE PRECISION, DIMENSION(NEQN,NU) :: Z
     DOUBLE PRECISION, DIMENSION(NEF) :: G
     INTENT(IN) :: T,Y,DY,Z
     INTENT(OUT) :: G
     ! Locate extrema as points where the derivative
     ! vanishes.  The DIRECTION option is used to
     ! distinguish maxima and minima.
     G(1) = DY(2)
     RETURN
  END SUBROUTINE EF
END MODULE define_DDEs
!******************************************************************
PROGRAM Example6
! The DDE is defined in the module define_DDEs.  The problem
! is solved here with DDVERK90 and its output written to a
! file.  The auxilary function Example6.M imports the data into
! Matlab and plots it.
  USE define_DDEs
  USE DDVERK90_M
  IMPLICIT NONE
  ! The quantities
  !   NEQN  = number of equations
  !   NU    = number of delays
  !   NEF   = number of event functions
```

```
! are defined in the module define_DDEs as PARAMETERs so
! they can be used for dimensioning arrays here. They are
! passed to the solver in the arrays NVAR and NLAGS.
INTEGER, DIMENSION(2) :: NVAR = (/NEQN,NEF/)
INTEGER, DIMENSION(1) :: NLAGS = (/NU/)
TYPE(DDVERK_SOL) :: SOL
TYPE(DDVERK_INT) :: YINT
TYPE(DDVERK_OPTS) :: OPTS
DOUBLE PRECISION :: T0=0.0D0,TFINAL=55.0D0
! Prepare output points
INTEGER, PARAMETER :: NOUT=1000
DOUBLE PRECISION, DIMENSION(NOUT) :: TINT
! Local variables:
INTEGER :: I
DOUBLE PRECISION, DIMENSION(NU) :: LAGS=(/ 1.0D0 ,10.0D0/)
DOUBLE PRECISION, DIMENSION(NEQN) :: HISTORY= (/ 5.0D0 ,0.1D0,1.0D0/)

OPTS = DDVERK_SET(RE=1D-6,AE=1D-6,INTERPOLATION=.TRUE.,DIRECTION=(/-1 /))
SOL = DDVERK90(NVAR,NLAGS,DDES,LAGS,HISTORY, &
       (/ T0,TFINAL /),OPTIONS=OPTS,EVENT_FCN=EF)
! Was the solver successful?
IF (SOL%FLAG == 0) THEN
   ! Form values for smooth plot:
   TINT = (/ (T0+(I-1)*((TFINAL-T0)/(NOUT-1)), I=1,NOUT) /)
   YINT = DDVERK_VAL(TINT,SOL,DERIVATIVES=.FALSE.)
   ! Write the solution to a file for subsequent plotting
   ! in Matlab.
```

```
    OPEN(UNIT=6, FILE='Example6.dat')
    DO I = 1,NOUT
      WRITE(UNIT=6,FMT='(4D12.4)') TINT(I),YINT%YT(I,1),&
                                   YINT%YT(I,2),YINT%YT(I,3)
    END DO
    ! Write the extrema to a file for plotting.
    OPEN(UNIT=7,FILE='Example6extr.dat')
    DO I = 1,SOL%NE
       WRITE(UNIT=7,FMT='(I10,2D12.4)') SOL%IE(I), &
                          SOL%TE(I),SOL%YE(I,2)
    END DO
    PRINT *,' Normal return from DDVERK90 with results'
    PRINT *," written to the file 'Example6.dat' and the"
    PRINT *," extrema written to 'Example6extr.dat'."
    PRINT *,' '
    PRINT *,' These results can be accessed in Matlab'
    PRINT *,' and plotted by'
    PRINT *,' '
    PRINT *," >> [t,y,te,ye,ie] = Example6;"
    PRINT *,' '
    CALL DDVERK_PRINT_STATS(SOL)
  ELSE
    PRINT *,' Abnormal return from DDVERK90 with FLAG = ',&
            SOL%FLAG
  ENDIF
  STOP
END PROGRAM Example6
```

Bibliography

[1] G.A. Bocharov, G.I. Marchuk, and A.A. Romanyukha. Numerical solution by LMMs of stiff delay differential systems modelling an immune response. *Numer. Math.*, 73:131–148, 1996.

[2] S. C. Corwin, D. Sarafyan, and S. Thompson. DKLAG6: a code based on continuous imbedded sixth-order runge-kutta methods for the solution of state-dependent functional differential equations. *Appl. Numer. Math.*, 24:319–330, 1997.

[3] W. H. Enright. Software for delay differential equations: Accurate approximate solutions are not enough. Manuscript for Voltera 2004.

[4] W. H. Enright and H. Hayashi. Convergence analysis of the solution of retarded and neutral delay differential equations by continuous numerical methods. *SIAM J. Numer. Anal.*, 35:572–585, 1998.

[5] Wayne H. Enright and Min Hu. Interpolating Runge-Kutta methods for vanishing delay differential equations. Technical Report 292/94, Department of Computer Science, University of Toronto, 1994.

[6] W.H. Enright and H. Hayashi. A delay differential equation solver based on a continuous Runge-Kutta method with defect control. *Numerical Algorithms*, 16:349–364, 1997.

[7] W.H. Enright and H. Hayashi. The evaluation of numerical software for delay differential equations. In R.F. Boisvert, editor, *The Quality of Numerical Software: Assessment and Enhancement*, pages 179–192. Chapman & Hall, London, 1997.

[8] I. Epstein and Y. Luo. Differential delay equations in chemical kinetics. Nonlinear models: The cross-shaped phase diagram and the Oregonator. *J. Chemical Physics*, 95:244–254, 1991.

[9] Lynne Genik and P. van den Driessche. An epidemic model with Recruitment-Death demographics and discrete delays. In Shigui Ruan, Gail S. K. Wolkowicz, and Jianhong Wu, editors, *Differential equations with applications to biology*, number 21 in Fields Institute Communications, pages 237–249. American Mathematical Society, Providence, RI, 1999.

[10] N. Guglielmi and E. Hairer. Users' guide for the code RADAR5. Technical report, University of Geneva, Geneva, Switzerland, October 2000.

[11] N. Guglielmi and E. Hairer. Implementing Radau IIA methods for stiff delay differential equations. *Computing*, 67:1–12, 2001.

[12] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I. Nonstiff problems.* Springer Series in Computational Mathematics. Springer-Verlag, New York, second edition, 1993.

[13] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II. Stiff and differential-algebraic problems.* Springer Series in Computational Mathematics. Springer-Verlag, New York, second edition, 1996.

[14] H. Hayashi. *Numerical solution of retarded and neutral delay differential equations using continuous Runge-Kutta methods.* PhD thesis, Department of Computer Science, University of Toronto, Toronto, Canada, 1996.

[15] Z. Jackiewicz and F.C. Hoppensteadt. Numerical solution of a problem in the theory of epidemics. Manuscript for Voltera 2004.

[16] C.A.H. Paul. A user-guide to Archi: An explicit Runge-Kutta code for solving delay and neutral differential equations. Technical Report 283, Department of Mathematics, University of Manchester, Manchester, England, April 1997.

[17] L.F. Shampine. Solving ODEs and DDEs with residual control. Manuscript, available at http://faculty.smu.edu/lshampin/residuals.pdf, 2003.

[18] L.F. Shampine, I. Gladwell, and S. Thompson. *Solving ODEs with MATLAB*. Cambridge University Press, Cambridge, 2003.

[19] L.F. Shampine and S. Thompson. Solving DDEs in MATLAB. Manuscript, available at http://www.cs.runet.edu/∼thompson/webddes/ddepap.html, 2000.

[20] L.F. Shampine and S. Thompson. Solving delay differential equations with dde23. Manuscript, available at http://www.runet.edu/∼thompson/webddes/index.html, 2000.

[21] S. Thompson and L.F. Shampine. A Friendly Fortran DDE Solver. Manuscript for Voltera 2004 , available at http://www.radford.edu/∼thompson/ffddes/.

[22] S. Thompson and L.F. Shampine. A Note on Zdzislaw's Epidemic Model Problem. Available at http://www.radford.edu/∼thompson/ffddes/, 2004.

[23] P. Waltman. An threshold model of antigen-stimulated antibody production. In G.I. Bell, A.S. Perelson, and G.H. Pimbley, editors, *Theoretical Immunology*, number 8 in Immunology, pages 437–453. Marcel Dekker, New York, 1978.

[24] D.R. Willé and C.T.H. Baker. DELSOL - a numerical code for the solution of systems of delay-differential equations. *Appl. Numer. Math.*, 9:223–234, 1992.