Efficient Simulation, Accurate Sensitivity Analysis and Reliable Parameter Estimation for Delay Differential Equations

by

Hossein ZivariPiran

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

Abstract

Efficient Simulation, Accurate Sensitivity Analysis and Reliable Parameter Estimation for Delay Differential Equations

Hossein ZivariPiran

Doctor of Philosophy

Graduate Department of Computer Science

University of Toronto

2009

Delay differential equations (DDEs) are a class of differential equations that have received considerable recent attention and been shown to model many real life problems, traditionally formulated as systems of ordinary differential equations (ODEs), more naturally and more accurately. Ideally a DDE modeling package should provide facilities for approximating the solution, performing a sensitivity analysis and estimating unknown parameters. In this thesis we propose new techniques for efficient simulation, accurate sensitivity analysis and reliable parameter estimation of DDEs.

We propose a new framework for designing a delay differential equation (DDE) solver which works with any supplied initial value problem (IVP) solver that is based on a general linear method (GLM) and can provide dense output. This is done by treating a general DDE as a special example of a discontinuous IVP. We identify a precise process for the numerical techniques used when solving the implicit equations that arise on a time step, such as when the underlying IVP solver is implicit or the delay vanishes.

We introduce an equation governing the dynamics of sensitivities for the most general system of parametric DDEs. Then, having a similar view as the simulation (DDEs as discontinuous ODEs), we introduce a formula for finding the size of jumps that appear at discontinuity points when the sensitivity equations are integrated. This leads to an algorithm which can compute sensitivities for various kind of parameters very accurately.

We also develop an algorithm for reliable parameter identification of DDEs. We propose a method for adding extra constraints to the optimization problem, changing a possibly non-smooth optimization to a smooth problem. These constraints are effectively handled using information from the simulator and the sensitivity analyzer.

Finally, we discuss the structure of our evolving modeling package DDEM. We present a process that has been used for incorporating existing codes to reduce the implementation time. We discuss the object-oriented paradigm as a way of having a manageable design with reusable and customizable components. The package is programmed in C++ and provides a user-friendly calling sequences. The numerical results are very encouraging and show the effectiveness of the techniques.

Acknowledgements

I wish to express my gratitude to my supervisor, Professor Wayne Enright, for his continued encouragement and invaluable suggestions during this work.

I would also like to thank the members of my thesis committee: Professors Christina Christara, Tom Fairgrieve and Ken Jackson for reading my manuscripts and for providing insightful comments and suggestions for improvement.

Special thanks to Professor Larry Shampine for accepting to be the external examiner and Professor Rudi Mathon for joining my final oral committee.

I am grateful to the University of Toronto, and to the Department of Computer Science, for providing me the opportunity to pursue graduate studies and for their generous financial support over the years.

Finally, I want to thank my family. The encouragement and support from my beloved wife, Maryam, and my joyful daughter, Ayda, is a powerful source of inspiration and energy.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Differential equations are one of the most frequently used tools for mathematical modeling in engineering and life sciences. Numerical studies usually involve simulating with various parameter values, assessing the sensitivity to changes in parameters and estimating relevant parameters from data. The increasing processing power of computers has encouraged scientists to use more complex and more detailed mathematical models to study real life problems. Delay differential equations (DDEs) are a class of differential equations that have received considerable recent attention and been proven to model many real life problems, traditionally formulated as systems of ordinary differential equations (ODEs), more naturally and more accurately. Nevertheless, required modeling components involve more complex numerical procedures for DDEs compared to ODEs, making a robust and reliable implementation more than a direct transformation of the corresponding components for standard ODEs.

Several DDE solvers have been implemented during the past twenty years, based on different traditional ODE techniques such as those based on Runge-Kutta formulas or on linear multi-step formulas. The implementations of these solvers are usually based on

adapting an existing initial value problem (IVP) solver and is usually completely redone for every new IVP solver and hence results in a long gap between the release of an efficient IVP formula and the corresponding DDE solver. This motivated us to look for a unified structure for a DDE solver independent of the underlying IVP solver.

The particular issues accompanying the numerical computation of sensitivities and parameter identification from data, which mostly originate from the discontinuous nature of DDEs, have been studied in detail ([5], [6]). Nonetheless, nothing has been suggested to overcome those difficulties for a general system of DDEs. This observation motivated us to develop methods for accurate sensitivity analysis and reliable parameter estimation in the presence of propagated discontinuities.

The techniques for sensitivity analysis and parameter estimation can be exploited by interested users along with a DDE solver for developing practical codes. However, there are many benefits in having these two basic functionalities accompanying the simulator in an integrated design called a *modeling package*. First, sensitivity analysis and parameter estimation have become as essential as simulation for studying a phenomenon using a mathematical model. Second, for complex models like DDEs, sensitivity analysis and parameter estimation become much more complicated. As a result, implementing practical codes to carry out these tasks is tedious and highly time consuming for non-specialists. Finally, sensitivity analysis and parameter estimation are highly dependent on the simulator and hence an integrated design could lead to efficient inner communication and prevent possible redundancies.

1.2 A Review of Mathematical Modeling

In this section we briefly describe some of the important concepts and terminologies of mathematical modeling. These concepts will be referenced throughout the thesis and are essential for future discussions.

1.2.1 The Modeling Process

Figure 1.1 shows schematically the general process that is followed in developing a model to study a physical/biological phenomenon. The process starts with a scientist looking into the phenomenon and recognizing the governing physical laws or empirical rules. The result is a mathematical model with some unknown parameters. Sensitivity analysis of the model is done to determine the effect of different parameters on the behavior of the model and then the model equations are accordingly adjusted so that the qualitative behavior is compatible with the reality. The last task is to estimate the model parameters by using observations from the real world. The result is a practical model that can be used to make predictions.

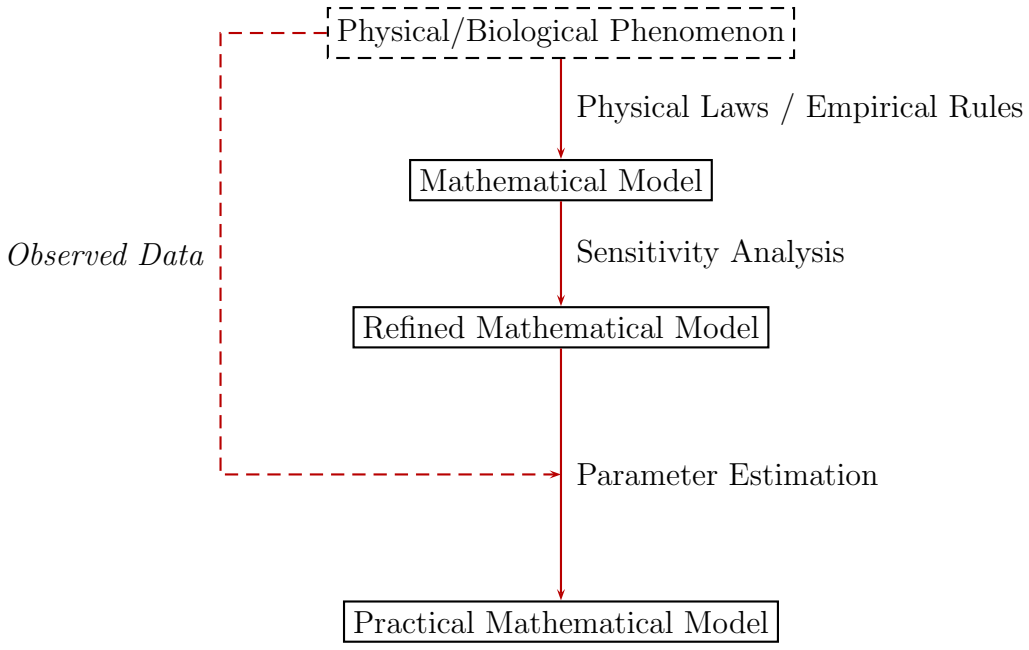

Figure 1.1: Schematic representation of the major steps that are performed to develop a practical mathematical model.

1.2.2 Parameterized Models

Usually a mathematical model used for the simulation of a real phenomenon contains some parameters. These parameters are used for different purposes. They can make the model applicable in similar situations where the only difference is the value of some constants. Furthermore, when some constants are just approximately known, it can be very helpful to analyze the effect of these uncertainties, which can be done by representing those constants as parameters. Finally, parameters can be used to represent unknown quantities.

The parameters affecting a state variable or a mathematical function are usually distinguished by being grouped separately in the list of arguments. Table 1.1 lists the notation commonly used to represent state variables for different types of differential equations and their corresponding parameterized variant.

Differential Equation	Traditional Form	Parameterized Form
IVP ODE/DAE/DDE	$y(t)$	$y(t;\mathbf{p})$
BVP ODE	$y(x)$	$y(x;\mathbf{p})$
BVP PDE	$u(\mathbf{x})$	$u(\mathbf{x};\mathbf{p})$
IVP PDE	$u(\mathbf{x},t)$	$u(\mathbf{x},t;\mathbf{p})$

Table 1.1: State variables and their corresponding parameterized version for some classes of differential equations. Bold symbols represent vectors.

1.2.3 Simulation

The term simulation refers to computing a numerical approximation for fixed values of parameters and some initial/boundary conditions for a phenomenon described using a mathematical model. Developing efficient simulation techniques is traditionally considered as the core of numerical studies and many researchers have been working on

developing faster and more reliable methods. One of the important issues that is always involved in this effort is the generality versus efficiency tradeoff. In other words, one wants to develop methods applicable for a broader range of problems. However, a specialized method, tailored with respect to the properties of a certain problem, may outperform and hence be more promising.

1.2.4 Sensitivity Analysis

The (first order) solution sensitivity with respect to the model parameter p_i is defined as the vector

$$s_i(t;\mathbf{p}) = \{\frac{\partial}{\partial p_i}\} y(t;\mathbf{p}), \quad (i = 1, \ldots, \mathcal{L}), \tag{1.1}$$

and the second order solution sensitivity with respect to the model parameters p_i and p_j is defined as the vector

$$r_{ij}(t;\mathbf{p}) = \{\frac{\partial}{\partial p_j}\} s_i(t;\mathbf{p}) = \{\frac{\partial^2}{\partial p_j \partial p_i}\} y(t;\mathbf{p}), \quad (i, j = 1, \ldots, \mathcal{L}). \tag{1.2}$$

They can be considered for all continuous time models including those defined by systems of differential equations.

Sensitivities of higher orders can be defined in a similar way but are rarely used in practice.

1.2.5 Parameter Estimation

Given a set of data $\{Y(\gamma_i) \approx y(\gamma_i;\mathbf{p})\}$ corresponding to a parameterized model (Table 1.1) the parameter estimation problem is that of estimating the best choice for vector $\mathbf{p}$. This is generally achieved by minimizing an objective function involving the data $\{Y(\gamma_i)\}$ and the corresponding values of the parameterized solution $\{y(\gamma_i;\mathbf{p})\}$.

A common objective function is based on the squared two-norm,

$$W(\mathbf{p}) = \sum_i \left[Y(\gamma_i) - y(\gamma_i;\mathbf{p})\right]^2. \tag{1.3}$$

because it is generally continuous (and hopefully smooth) with respect to variations in $\mathbf{p}$.

1.3 Definitions and Notations

- An Initial Value Problem (IVP) for Ordinary Differential Equations (ODEs) is a system of differential equations defined by

$$\begin{aligned} y'(t) &= f(t, y(t)), \quad \text{for} \quad t_0 \le t \le t_F, \\ y(t_0) &= y_0, \end{aligned} \tag{1.4}$$

  where y and f are $\mathcal{M}$-vector functions.

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

$$\begin{aligned} y'(t) &= f(t, y(t), y(t-\sigma_1), \ldots, y(t-\sigma_\nu)), \quad \text{for} \quad t_0 \le t \le t_F, \\ y(t) &= \phi(t), \quad \text{for} \quad t \le t_0, \end{aligned} \tag{1.5}$$

  where y, f, and ϕ are $\mathcal{M}$-vector functions and $\sigma_i = \sigma_i(t, y(t)) \ge 0, i = 1, 2, \ldots, \nu$ are scalar functions and in general time and state dependent, but for some problems they are only time dependent or even constants.

- A neutral delay differential equation (NDDE) is a system of differential equations defined by

$$\begin{aligned} y'(t) = f(t, y(t), y(t-\sigma_1), \ldots, y(t-\sigma_\nu), \\ y'(t-\sigma_{\nu+1}), \ldots, y'(t-\sigma_{\nu+\omega})), \quad \text{for} \quad t_0 \le t \le t_F, \\ y(t) = \phi(t), \quad y'(t) = \phi'(t), \quad \text{for} \quad t \le t_0. \end{aligned} \tag{1.6}$$

  The term DDE refers to both an RDDE and an NDDE.

  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*. (Note that delays are always considered to take positive values and advanced delays are only numerical artifacts.) We call $\phi(t)$ the *history* function.

- A parameterized IVP is defined by

$$\begin{aligned} y'(t;\mathbf{p}) &= f(t,y(t;\mathbf{p});\mathbf{p}), \quad \text{for } t_0 \le t \le t_F, \\ y(t_0) &= y_0(\mathbf{p}), \end{aligned} \tag{1.7}$$

  where $\mathbf{p}$ is an $\mathcal{L}$-vector of parameters.

1.4 Some Areas of Application of DDEs

Table 1.2 lists some areas where systems of DDEs are used for modeling some real life phenomena. (Interested readers can find more details in reference [1] and references therein.)

1.5 A Review of Previous Work

1.5.1 Existing DDE Simulators

Neves [47] proposed a general approach for the conversion of an ODE solver to a corresponding DDE solver. Based on that, he implemented the code DMRODE [46]. Corwin et al. [19] developed DKLAG6 following Neves's approach. More recently, Thompson and Shampine [59] released a newer version (of DKLAG6) called DDE_SOLVER. Butcher [12] modified the code STRIDE [14], originally designed for ODEs, to make it usable for

Area	Example
Ecology	predator-prey
Epidemiology	spread of infections [28]
Immunology	immune response models [25]
HIV infection [44]	
Physiology	human respiration system [18], nervous system [24]
Neural Networks	
Cell Kinetics [35]	
Chemical Kinetics	The Oregonator [23]
Physics	Ring Cavity Lasers [61], two-body problem of electrodynamics

Table 1.2: Some areas of applications of DDEs.

DDEs. Bocharov et al. [9] developed the code DIFSUB-DDE based on Gear's DIFSUB [27]. RADAR5 code developed by Guglielmi and Hairer [30] is a modification of the code RADAU5 [33]. Willé and Baker's DELSOL [62], Paul's ARCHI [50], Hayashi's DDVERK [34], and Zivari's DDVERK90 [63] are other DDE solvers developed using modifications of existing effective ODE solvers.

1.5.2 Current Sensitivity Analysis Techniques for DDEs

Baker and Rihan [6] have studied the sensitivity analysis problem for the subclass of parameterized DDEs defined by,

$$\begin{aligned} y'(t;\mathbf{p}) &= f(t, y(t;\mathbf{p}), y(t-\sigma(t;\mathbf{p}));\mathbf{p}), \quad \text{for } t \geq t_0(\mathbf{p}), \\ y(t;\mathbf{p}) &= \phi(t;\mathbf{p}), \quad \text{for } t \leq t_0(\mathbf{p}), \end{aligned} \tag{1.8}$$

where $\mathbf{p}$ is an $\mathcal{L}$-vector of parameters.

Following the method of *internal differentiation* (§3.3.2), by applying $\frac{\partial}{\partial \mathbf{p}}$ to (1.8) they

obtain, after some manipulations, the governing equations for the sensitivity coefficients,

$$S'(t) = J(t)S(t) + J_\sigma(t)S(t-\sigma) + B(t), \tag{1.9}$$

and

$$\begin{aligned} R'(t) =& A(t)S(t) + (I_{\mathcal{L}} \otimes J(t))R(t) + A_\sigma(t)S(t-\sigma) + \\ & (I_{\mathcal{L}} \otimes J_\sigma(t))R(t-\sigma) + K(t), \end{aligned} \tag{1.10}$$

where $S(t)$, $J(t)$, $B(t)$, $R(t)$, $A(t)$, and $K(t)$ are defined similar to ODEs (see §3.3.2 for details). The Jacobian matrices $J_\sigma(t)$ and A_σ are correspondingly defined by taking partial derivative of f w.r.t. the delayed term $y(t-\sigma(t;\mathbf{p}))$.

To find the sensitivity coefficient matrices, S and R, they solve the delay differential systems (1.9) and (1.10) simultaneously with the system (1.8), with respectively, associated initial functions

$$S(t,\mathbf{p}) = \frac{\partial}{\partial \mathbf{p}}\phi(t,\mathbf{p}), \text{ and } R(t,\mathbf{p}) = \frac{\partial^2}{\partial \mathbf{p}\partial \mathbf{p}^{\mathrm{T}}}\phi(t,\mathbf{p}). \tag{1.11}$$

In the case that a constant delay, σ, is a component, p_j, of $\mathbf{p}$ or the delay is state dependent or the original problem is an NDDE, the resulting systems (1.9, 1.10) become slightly different and in general could be a system of NDDEs. They do not provide an automatic sensitivity computation code and suggest users first derive the sensitivity equations and then solve them using an existing DDE simulator.

Baker and Paul [5] have studied the continuity properties of sensitivity coefficients for DDEs. They also classify situations where discontinuities occur.

Rihan [54] used adjoint equations and direct methods to estimate the sensitivity functions for a class of systems modeled by DDEs of the form (in his notation),

$$\begin{aligned} y'(t) &= f(t, y(t), y(t-\tau), u(t), u(t-\sigma), \mathbf{p}), \quad \text{for } 0 \le t \le T, \\ y(t) &= \Psi(t,\mathbf{p}), \quad \text{for } t \in [-\tau, 0), \\ y(0) &= y_0, \\ u(t) &= \Phi(t,\mathbf{p}), \quad \text{for } t \in [-\sigma, 0), \\ u(0) &= u_0, \end{aligned} \tag{1.12}$$

where $u(t)$ is a control variable that minimizes an objective function, and τ and σ are positive constant lags. He considered the sensitivity coefficients for the constant parameters as well as the functional derivative sensitivity coefficients for $\Psi(t)$, $\Phi(t)$, and $u(t)$.

1.5.3 Existing Parameter Estimation Methods for DDEs

Baker and Paul [5] have studied difficulties that arise when adapting the initial value approach (§4.2.1) for the parameter estimation of ODEs, to the parameter estimation problem for DDEs. The code ARCHI developed by Paul [52] can be used to estimate parameters of DDEs. In addition to the usual parameters, ARCHI is able to consider the constant delays and the starting point as unknown parameters. There is also the capability of handling some types of constraints on unknown parameters.

Horbert et al. [38] have extended the idea of multiple shooting (§4.2.1) to estimate parameters of DDEs. They have shown that their method can attain a high convergence rate in the case of noisy data.

Murphy [45] introduced a full discretization approach where he uses linear splines to describe solution and delay functions. The problem then is treated as a very large minimization problem.

Although initial function (history) identification can be considered as a special case of the general parameter estimation problem, more efficient methods can be found by applying specialized algorithms to this case. Baker and Parmuzin in a series of reports introduced some algorithms using this idea. In [2] they studied the problem for a system of linear DDEs with time dependent coefficients, of the form

$$\frac{dy(t)}{dt} - A(t)y(t) - B(t)y(t-\sigma) = f(t), \qquad \text{for } t \in [0,T], \tag{1.13}$$

subject to

$$y(t) = \phi(t), \qquad \text{for } t \in [-\sigma, 0], \tag{1.14}$$

where σ is a prescribed positive constant. The solution of (1.13)-(1.14) is dependent on

the function ϕ in (1.14), we can therefore write

$$y(t) = y(\phi; t), \qquad \text{for } t \in [-\sigma, T]. \tag{1.15}$$

They show that the optimal initial function $\phi^\star$ satisfies a coupled set of delay equations, involving "adjoint equations", and they give an iterative technique for obtaining successive approximations ϕ_n to $\phi^\star$.

In [3] they analyze a discrete analogue of the problem and establish a practical algorithm. They show the robustness of their method through several examples.

In [4] they apply the same ideas to a non-linear problem of the form

$$\frac{dy(t)}{dt} = f(t, y(t), y(t-\sigma)), \qquad \text{for } t \in [0, T], \tag{1.16}$$

subject to

$$y(t) = \phi(t), \qquad \text{for } t \in [-\sigma, 0]. \tag{1.17}$$

A "Pseudo-Newton" method is introduced to solve the optimization problem. A major difficulty with this method is its slow convergence.

1.6 Contributions of the Thesis

The contributions of this thesis fall in four different areas: simulation of DDEs, sensitivity analysis of DDEs, parameter estimation for DDEs, and developing effective software for DDE modeling.

We propose a new unified and effective structure for a DDE solver which is independent of the underlying IVP formula. We discuss how a system of DDEs can be considered as a discontinuous system of IVPs. Using this point of view, we are able to identify more precise techniques for dealing with issues arising in the simulation of DDEs, such as locating discontinuities and correct handling of possible irregularities. We discuss both explicit solvers and implicit solvers in detail. We also propose a general iterative method

for handling vanishing delays, while respecting the independence from the underlying IVP method.

We derive an equation governing the dynamics of sensitivities for the most general system of parametric NDDEs. Then, having a similar view as the simulation (DDEs as discontinuous IVPs), we derive a formula for finding the size of jumps that appear at discontinuity points when the sensitivity equations are integrated. The formula leads to an algorithm which can compute sensitivities for various kind of parameters very accurately and efficiently.

We develop an algorithm for reliable parameter identification of DDEs. We propose a method for adding extra constraints to the optimization problem, changing a possibly non-smooth optimization to a smooth problem. These constraints are effectively handled using information from the simulator and the sensitivity analyzer.

Finally, we present a design for a DDE modeling package including basic required functionalities. We exploit some programming techniques and develop a package that follows basic software development standards, such as user friendliness, thread safety, reusability and modularity.

1.7 An Outline of the Thesis

In Chapter 2, we propose a structure for a DDE solver which is independent of the underlying IVP solver. We first review the techniques used in the numerical solution of discontinuous IVPs and show that a general DDE can be treated as a special example of a discontinuous IVP. We then develop the details of a DDE solver based on this view and also the required interface with an IVP solver. In Chapter 3, we first, briefly, describe the adjoint method for sensitivity analysis, and then we study the details of the forward sensitivity analysis approach adapted for delay differential equations. The governing equations for sensitivities are derived and combined with an adapted jump equation and

an algorithm for accurate calculation of sensitivities is proposed. In Chapter 4, after reviewing some existing techniques for parameter estimation of problems involving systems of ODEs and DDEs, we discuss the necessity of considering possible discontinuities in the objective function in the DDEs case. We then introduce a new technique for handling the resulting discontinuous least-squares problem efficiently. In Chapter 5, we describe our designs for components of a DDE modeling package (simulation, sensitivity analysis, parameter estimation) and propose techniques to overcome some of the difficulties arising in the design of such a package. In Chapter 6, we present numerical results. In Chapter 7, we summarize the thesis and discuss future work.

Chapter 2

Efficient Simulation

2.1 Introduction

Several DDE solvers have been implemented during the past twenty years, based on the extension or modification of traditional ODE techniques such as those based on Runge-Kutta or linear multi-step formulas ([9], [19], [30], [34], [52], [59], [62]). The implementations of these solvers are usually based on adapting an existing initial value problem (IVP) solver. These DDE solvers use the provision for dense output, which is a key component of most modern IVP solvers, as the base and add some strategies for handling discontinuities and vanishing delays. During this process, special properties of the underlying IVP solvers are usually exploited to make the overall technique efficient.

There are some drawbacks associated with this approach for developing DDE solvers. First, it usually takes a long time for an IVP solver to be recognized and subsequently identified as a candidate for modification for use as a DDE solver. Therefore, it is difficult to have a timely investigation of the effectiveness of proposed new underlying formulas (used in IVP solvers), for DDEs. Second, when some or all added components of a DDE solver, such as step size selection strategy and discontinuity handling, rely on the underlying IVP formula, they need to be redeveloped and recoded for every new DDE

solver. This results in a lot of redundancy during the analysis and the coding, since many concepts involved in developing these components are common.

In this chapter we propose a structure for a DDE solver which is independent of the underlying IVP solver, with the only interactions between the two being through a common interface. We consider a general step-by-step IVP solver that finds continuous numerical approximations to problems of the general form (1.4). The resulting DDE solver that we develop can be applied to approximate the solution of a system of RDDEs (1.5), or a system of NDDEs (1.6).

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 RDDEs, but this is not the case for NDDEs.

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

In the remainder of this chapter we first review the techniques used in the numerical solution of discontinuous IVPs and show that a general DDE can be treated as a special example of a discontinuous IVP. In Section 2.5 we develop the details of a DDE solver based on this view and also the required interface with an IVP solver.

2.2 Discontinuous IVPs and Hybrid Systems

Hybrid systems are mathematical models that exhibit both discrete and continuous behavior over the time interval of interest. The continuous behavior of the model is usually described by one or more ODEs, DDEs, or differential-algebraic equations (DAEs). The discrete behavior, which occurs at particular points in time (events), includes phenomena such as nonsmooth forcing, switching of the vector field and jumps in the state. This is a new perspective, that can be compared with the traditional approach of formulating the model in terms of discontinuous vector fields. (For more information and discussion of hybrid systems arising in mathematical modeling see [8] or [29] and references therein.)

Consider a simple case of a hybrid system described using two sets of differential equations and a switching function,

$$y'(t) = \begin{cases} f_1(t, y(t)), & \text{for } g(t, y(t)) < 0, \\ f_2(t, y(t)), & \text{for } g(t, y(t)) \geq 0. \end{cases} \tag{2.1}$$

To simulate the system (2.1), one has to use an integration scheme along with a transition handler. The integration is usually done by a Runge-Kutta method or a linear multistep method. The transition handler is responsible for detecting events and locating switching points and changing the integration accordingly (transition). An important aspect of the transition handler is the correct detection of irregularities that may happen at a switching point, such as non-uniqueness or termination of the solution.

Suppose that the integration of the hybrid system (2.1) has reached t_n where we have $y_n \approx y(t_n)$. The local solution $z_n(t)$ over $[t_n, t_{n+1}]$ is defined by

$$\begin{aligned} z_n'(t) &= \begin{cases} f_1(t, z_n(t)), & \text{for } g(t, z_n(t)) < 0, \\ f_2(t, z_n(t)), & \text{for } g(t, z_n(t)) \geq 0, \end{cases} \\ z_n(t_n) &= y_n. \end{aligned} \tag{2.2}$$

2.3 Techniques for the Efficient Simulation of Discontinuous IVPs

Suppose that a solver is trying to compute a numerical approximation to the solution of (2.1) by computing a continuous approximation on each step. All standard solvers assume that the solution is continuous enough, over the entire step, for the underlying formulas to be applied with confidence. An accepted strategy is to detect and include the discontinuity points in the set of mesh points. For the successful application of this strategy, having an accurate discontinuity or switching point location is necessary.

For a state-dependent switching function, the location of discontinuities cannot be computed à priori because their unknown locations depend implicitly on the unknown solution.

When the solver wants to take a step from t_n to t_{n+1}, and a switching/discontinuity is suspected to occur in $[t_n, t_{n+1}]$, approximations based on sufficient differentiability of the solution become unreliable. Therefore, using the local continuous approximation to the solution in $[t_n, t_{n+1}]$ to locate the discontinuity may lead to an inaccurate approximation (see Figure 2.1).

A common treatment of this difficulty uses an iterative method which in turn computes the approximate solution y, and the zero crossing function of an associated *event function* g. Assuming that the iteration is convergent, the solution and the location of the discontinuity become more accurate on each iteration. The drawback of this method is the slow rate of convergence, which is linear in the best implementations.

Here we give details of a more efficient treatment introduced by Ellison [20]. The idea is to somehow reduce the effect of locating the zero crossing of g on the computation of y. This can be done by defining the functions $z_{[c]}$ and $g_{[c]}$ as follows. $z_{[c]}$ is defined as the solution of (2.2) when we eliminate the effect of the switching point λ, by using a smooth extension of the local solution $z_n(t)$ after λ (see Figure 2.1), that is, $z_{[c]}$ is the solution of

the local IVP,

$$\begin{aligned} z'_{[c]}(t) &= f_i(t, z_{[c]}(t)), \quad \text{for} \quad t_n \leq t \leq t_{n+1}, \\ z_{[c]}(t_n) &= y_n, \end{aligned} \tag{2.3}$$

where the index i is determined using the state of the system (controlled by g) at t_n and stays the same (either $i = 1$ or $i = 2$) for $t_n \leq t \leq t_{n+1}$.

Then, $g_{[c]}$ is defined as the switching (zero crossing) function computed using $z_{[c]}$,

$$g_{[c]}(t) = g(t, z_{[c]}(t)), \tag{2.4}$$

which is a function of only t, because $z_{[c]}(t)$ is defined as a usual IVP without any switching functions over $[t_n, t_{n+1}]$.

The governing systems of differential equations for $z_n(t)$ and $z_{[c]}(t)$ are the same before the switching point. Hence, we have,

$$z_n(t) = z_{[c]}(t), \quad \text{for} \quad t_n \leq t < \lambda. \tag{2.5}$$

Then, it is not hard to see that,

$$g_{[c]}(t) = g(t, z_n(t)), \quad \text{for} \quad t_n \leq t < \lambda. \tag{2.6}$$

Considering the limit case,

$$\lim_{t \nearrow \lambda} g_{[c]}(t) = \lim_{t \nearrow \lambda} g(t, z_n(t)), \tag{2.7}$$

or (using the continuity of $g_{[c]}$),

$$g_{[c]}(\lambda) = g(\lambda, \lim_{t \nearrow \lambda} z_n(t)), \tag{2.8}$$

which gives us,

$$g_{[c]}(\lambda) = 0, \tag{2.9}$$

because λ is a switching point for $z_n(t)$.

Equation (2.9) defines another function that crosses zero at λ. However, there is a big difference which makes Equation (2.9) very attractive. The difference is that $g_{[c]}$ is time dependent and also differentiable. This eliminates the possibility of computing a false solution (see Figure 2.1), and gives us a direct way of computing λ, by first computing $z_{[c]}$ and then applying a root finding algorithm to the associated $g_{[c]}$.

The differentiability of $g_{[c]}$, which results from the differentiability of $z_{[c]}$, enables us to apply efficient root finding methods such as Newton's method or its variations.

Standard numerical methods for IVPs compute an accurate approximation only for $z_{[c]}(t_{n+1})$. Therefore, an IVP method which provides an accurate continuous approximation is required for our root finding process. Those methods have been developed and are widely available. However, a numerical method used in practice only provides $\bar{z}_{[c]}(t)$, an accurate approximation to $z_{[c]}(t)$ over $[t_n, t_{n+1}]$. As a result, the function investigated by the root finder is actually

$$\bar{g}_{[c]}(t) = g(t, \bar{z}_{[c]}(t)). \tag{2.10}$$

If $g(t, y)$ is Lipschitz continuous and $\bar{z}_{[c]}(t) \approxeq z_{[c]}(t)$, then any discrepancy between the computed roots of $\bar{g}_{[c]}(t)$ and $g_{[c]}(t)$ will be within the accepted numerical error.

A similar idea has been used by Park and Barton [48] for handling transitions in hybrid systems of differential algebraic equations.

2.4 DDEs as Discontinuous IVPs

Consider a simple state-dependent retarded delay differential equation (RDDE) defined by

$$\begin{aligned} y'(t) &= f(t, y(t), y(\alpha(t, y(t)))), \quad \text{for } t \geq t_0, \\ y(t_0) &= y_0, \\ y(t) &= \phi(t), \quad \text{for } t < t_0, \end{aligned} \tag{2.11}$$

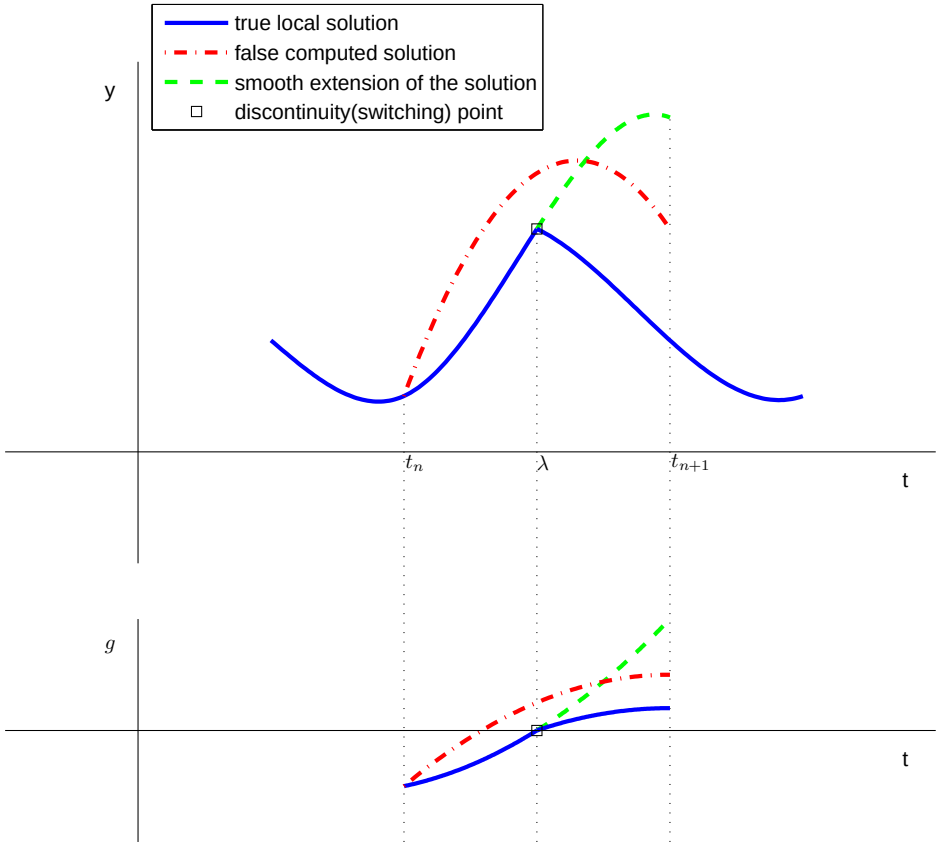

Figure 2.1: A typical situation for a state-dependent switching function. The true local solution refers to the exact solution of Equation (2.2); the false computed solution refers to the continuous approximate solution of Equation (2.2) (produced by a standard IVP method); and smooth extension refers to $z_{[c]}(t)$ (Equation (2.3)). Different approximations to the switching function g are computed using the corresponding solution approximations and are identified accordingly.

where $\alpha(t, y(t)) = t - \sigma(t, y(t))$, and $f(t, y, v)$ is sufficiently differentiable with respect to t, y and v.

With this assumption, the only discontinuities in the solution or its low order derivatives will be associated with the propagation of discontinuities introduced by the initial function or at the initial point.

Now assume that jumps in one of the derivatives of $y(t)$ with respect to t occur at the points

$$\cdots < \lambda_{-2} < \lambda_{-1} < \lambda_0 = t_0 < \lambda_1 < \lambda_2 < \cdots \tag{2.12}$$

where $\lambda_j,\ j < 0$, are discontinuities in the initial function. Then, artificial event functions

$$g_i(t, y(t)) = \alpha(t, y(t)) - \lambda_i, \quad i = \ldots, -2, -1, 0, 1, 2, \ldots \tag{2.13}$$

can be defined accordingly and used to write the equation characterizing the propagation of a discontinuity to $\lambda_r,\ r \geq 1$,

$$\lambda_r = \min\{\lambda > \lambda_{r-1} \colon \lambda \text{ is a root of odd multiplicity of } g_i(t, y(t)),\ \ i \leq r-1\}. \tag{2.14}$$

In other words, $\lambda_r,\ r \geq 1$, is the leftmost discontinuity of all propagated discontinuities arising from $\{\ldots, \lambda_{-1}, \lambda_0, \lambda_1, \ldots, \lambda_{r-1}\}$ and lying in $(\lambda_{r-1}, +\infty)$. The roots of $g_i(t, y(t))$ with even multiplicity do not cause discontinuities and they do not need to be identified, since the delay argument, $\alpha(t, y(t))$, *crosses* a previous discontinuity point only for roots which have odd multiplicity.

Note that for the special case involving a single increasing delay argument and a smooth history function, $\phi(t)$; each discontinuity is caused by propagation from the most recent previous discontinuity point, namely,

$$\alpha(\lambda_r, y(\lambda_r)) = \lambda_{r-1},\ \ r \geq 1. \tag{2.15}$$

Using the explicit identification of all sources of non-smoothness, it is not hard to see that the solution of the system (2.11) also satisfies the following system of discontinuous

IVPs,

$$\begin{aligned} y'(t) &= f_r(t, y(t)) = f(t, y(t), y_{[r]}(\alpha(t, y(t)))), \\ &\qquad \text{for } \lambda_r \le \alpha(t, y(t)) < \lambda_{r+1}, \\ y(t_0) &= y_0. \end{aligned} \tag{2.16}$$

where

$$y_{[r]}(\alpha) = \begin{cases} y(\alpha), & \text{for } \lambda_r \le \alpha < \lambda_{r+1}, \\ \text{smooth extension from } [\lambda_r, \lambda_{r+1}), & \text{for } \alpha < \lambda_r \text{ or } \alpha \ge \lambda_{r+1}, \end{cases} \tag{2.17}$$

The value of $y_{[r]}(\alpha)$ outside $[\lambda_r, \lambda_{r+1})$ is not required to be defined, as the right hand side of (2.16) switches if α goes outside this interval. Therefore, the smooth extension in (2.17) is only defined and used to facilitate the root-finding process during the numerical computations.

Now, using (2.13), Equation (2.16) can be rewritten in the standard form for discontinuous IVPs as

$$\begin{aligned} y'(t) &= f_r(t, y(t)), \\ &\qquad \text{for } g_r(t, y(t)) \ge 0 \text{ and } g_{r+1}(t, y(t)) < 0, \\ y(t_0) &= y_0. \end{aligned} \tag{2.18}$$

While (2.18) defines the switching functions, due to the (possible) presence of a C^0 discontinuity (i.e. discontinuity in the value), the transition still needs to be clarified when there are different choices for the value at a discontinuity point. In a hybrid system, those discontinuities can come from jumps in the state which are triggered by specially defined events. Traditionally, due to the correspondence of systems with real phenomena, all C^0 discontinuities are considered as transitions in state or control variables. Hence, the exact value of state or control variables at the point of discontinuity is not important, and can be considered to be evaluated using left or right segments. In this view, a discontinuity point is mainly considered as a border between two continuous segments. Therefore, if a

value of a variable needs to be evaluated at a discontinuity point λ, the segment used for the evaluation is picked with respect to the underlying transition. This means that if λ is approached from left(right) and we need the value before the (possible) transition, then the left(right) segment is used, and if the value after the (possible) transition is needed then the right(left) segment is used.

2.5 Interfacing with IVP Integrators

Assume that an approximate solution has been computed using a step-by-step IVP integrator over $[t_0, t_n]$ and now a step is to be taken from t_n to t_{n+1}. If $\theta_{[r]}(\alpha)$ is an associated accurate continuous approximation to $y_{[r]}(\alpha)$ (usually a piecewise polynomial), (2.18) can then be numerically integrated using the associated perturbed IVP,

$$f(t, y(t), y_{[r]}(\alpha(t, y(t)))) \approxeq f(t, y(t), \theta_{[r]}(\alpha(t, y(t)))). \tag{2.19}$$

2.5.1 Explicit IVP Integrators and Small/Vanishing Delays

Preserving explicit structure of the underlying IVP formula requires that the delay value, $y(\alpha)$, be independent of the values introduced in the current step $[t_n, t_{n+1}]$. The associated *explicitness condition*

$$\alpha(t, y(t)) \leq t_n, \quad \forall\, t \in [t_n, t_{n+1}],$$

can be monitored by introducing the local switching function,

$$g_e^n(t, y(t)) = \alpha(t, y(t)) - t_n,$$

(where 'e' denotes "explicit") and can be added to the system, (2.18),

$$\begin{aligned} y'(t) &= f_r(t, y(t)), \text{ for} \\ & g_r(t, y(t)) \geq 0, \quad g_{r+1}(t, y(t)) < 0, \\ & g_e^n(t, y(t)) \leq 0, \\ y(t_n) &= y_n. \end{aligned} \tag{2.20}$$

This formulation when combined with the approach for handling switching functions described in the previous section would impose the restriction that the step size $t_{n+1} - t_n$ be smaller than the minimum delay. In other words, if g_e^n is triggered, say at t_e, then the current step is partitioned at t_e, and the next step starts at $t_{n+1} = t_e$.

To avoid taking a series of extremely small steps in the case of a vanishing delay, we add the constraint,

$$\alpha(t, y(t)) < t - \epsilon,$$

where ϵ is a lower bound for small delays. The associated transition function is then,

$$g_v(t, y(t)) = \alpha(t, y(t)) - t + \epsilon, \tag{2.21}$$

and can be used to rewrite (2.20) as,

$$\begin{aligned} y'(t) &= f_r(t, y(t)), \text{ for} \\ &\quad g_r(t, y(t)) \geq 0, \quad g_{r+1}(t, y(t)) < 0, \\ &\quad g_e^n(t, y(t)) \leq 0, \\ &\quad g_v(t, y(t)) < 0, \\ y(t_n) &= y_n. \end{aligned} \tag{2.22}$$

If $g_v(t, y(t))$ is triggered, continuing with the explicit integration is not numerically feasible as it will result in an excessive number of small steps. There are two possible strategies for resolving this difficulty: taking a few special steps with the explicit integrator to pass the vanishing neighborhood or temporarily switching to an implicit integrator. Here we describe a possible approach for taking the special steps. After $g_v(t, y(t))$ is triggered, replace (2.20) with,

$$\begin{aligned} y'(t) &= f_r(t, y(t)) = f(t, y(t), \theta_{[r]}(\alpha(t, y(t)))), \text{ for} \\ &\quad g_r(t, y(t)) \geq 0, \quad g_{r+1}(t, y(t)) < 0, \\ &\quad g_v(t, y(t)) > 0, \\ y(t_n) &= y_n, \end{aligned} \tag{2.23}$$

where the component of $\theta_{[r]}$ in $[t_n, t_{n+1}]$, which is needed during the current step, has not been computed yet. Considering the fact that this component can be approximated using the computed y, we observe a potential loop. A common approach to terminate this loop is to treat it as a system of nonlinear equations. For the numerical solution of the resulting system of nonlinear equations, fixed point iterations or a modification of Newton-Raphson can be used. Here we give the details of fixed point iterations. (Note that this iteration is similar to the Picard iteration or the waveform relaxation iteration arising in the analysis of IVPs.)

1. Choose an initial guess for the interpolant $\mathcal{P}_n$ in $[t_n, t_{n+1}]$.
2. Compute the solution using the interpolant $\mathcal{P}_n$ as a part of the history.
3. Update the interpolant $\mathcal{P}_n$ using the last computed solution.
4. If the sequence of updated interpolants has converged, stop.
5. Continue with (2).

A good initial guess for the interpolant is usually obtained by extrapolation of the interpolant from the previous step. If there is not a previous step associated with the current step, or the previous step is not connected to the current step with sufficient continuity, then using extrapolation may not be possible or may give a poor result. In such cases an alternative is to treat the equations for the first iteration as specified below,

$$\begin{aligned} y'(t) = \tilde{f}(t, y(t)) = f(t, y(t), (1-\xi)y(t_n) + \xi y(t)), \text{ for} \\ g_r(t, y(t)) \geq 0, \quad g_{r+1}(t, y(t)) < 0, \\ g_v(t, y(t)) > 0, \\ y(t_n) = y_n, \end{aligned} \tag{2.24}$$

where

$$\xi = \frac{\alpha(t, y(t)) - t_n}{t - t_n},$$

and then, after we compute y, use it to define the initial guess for $\mathcal{P}_n$ in $[t_n, t_{n+1}]$ and switch back to our original equations (2.23) for further iterations.

The first order approximation $(1-\xi)y(t_n) + \xi y(t)$, usually leads to a more accurate starting approximation than the case when a constant approximation like $y(t_n)$ is used. This formula can be justified using backward error analysis in form of the associated defect (assuming the Lipschitz continuity of f) or, in other words, by observing that the residual after the first iteration will be at worse $\mathcal{O}(h_n^2)$.

2.5.2 Implicit IVP Integrators

Since implicit integrators are usually used when the system of ODEs is stiff, taking large steps with these integrators is not unusual. Therefore, we may encounter the case with an unknown interpolant for the current step arises on a large fraction of the attempted steps. Furthermore, in this situation a vanishing delay need not be treated as a special case. We can consider the interpolant on the current step to be represented by the implicitly defined stages introduced on this step, and try to find it in a similar way that we solve for the discrete solution y_n itself. Here we give the details for Runge-Kutta (RK) methods and linear multistep methods (LMMs). Since RK methods and LMMs are special cases of general linear methods (GLMs), we use the standard formulation [13] for GLMs. The details for RK methods or LMMs can be derived by standard translations from GLMs (see [13] for details).

The associated system of ODEs for $[t_n, t_{n+1}]$ is

$$\begin{aligned} y'(t) &= f_r(t, y(t)) = f(t, y(t), y_{[r]}(\alpha(t, y(t)))), \text{ for} \\ & g_r(t, y(t)) \geq 0, \quad g_{r+1}(t, y(t)) < 0, \\ y(t_n) &= y_n. \end{aligned} \tag{2.25}$$

After defining the equations for the unknown stage values $Y_j,\ j = 1, \ldots, s$, a modification of Newton-Raphson is usually used to solve for these unknown vectors. This nonlinear

iteration will involve the computations of

$$F_j = f_r(t_j, Y_j), \quad j = 1, 2, \ldots, s \tag{2.26}$$

and

$$\begin{aligned} &\frac{\partial F_j}{\partial Y_k}, \quad j = 1, 2, \ldots, s \\ &\qquad\qquad k = 1, 2, \ldots, s \end{aligned} \tag{2.27}$$

in an iterative scheme, attempting to converge to the solution of the nonlinear equations defining the unknown stage values. In the case of an unknown interpolant ($\alpha(t, y(t)) > t_n$ for some t in $[t_n, t_{n+1}]$), the theory of ODEs is not applicable directly. In the following we discuss a way to treat this case as a system of ODEs, even when an unknown interpolant is introduced.

Simultaneous Iterative Improvement

Assume that $\mathcal{P}_n$ is the local polynomial interpolant (associated with the current step from t_n to t_{n+1}), which in the most general case has structural dependencies on $Y_j\,, j = 1, 2, \ldots, s$ and $y_i^{[n]}\,, i = 1, 2, \ldots, q$ (input approximations), then

$$\theta_{[r]}(\alpha) = \begin{cases} \text{independent of } Y, & \text{for } \alpha < t_n, \\ \mathcal{P}_n[Y, y^{[n]}](\alpha), & \text{for } \alpha \geq t_n, \end{cases} \tag{2.28}$$

where $Y = \{Y_1, Y_2, Y_3, \ldots, Y_s\}$ and $y^{[n]} = \{y_1^{[n]}, y_2^{[n]}, \ldots, y_q^{[n]}\}$ are used for convenience. In the following $(\frac{\partial \mathcal{A}}{\partial q})$ is used to indicate partial differentiation of a parametric multivariate function $\mathcal{A}$ w.r.t. a parameter or variable q, and $(\frac{d\mathcal{A}}{dq})$ is used to indicate the total derivative of such a function.

Computing F_j : In either case of (2.28) the continuous approximation $\theta_{[r]}(\alpha)$ is computable at all required points, provided that all components of Y are available. In our iterative improvement scheme, these values are determined from the latest iteration or are as the initial guess for the first iteration.

Computing $\frac{\partial F_j}{\partial Y_k}$: Differentiating (2.26) and using (2.16) and (2.19),

$$\begin{aligned}\frac{\partial F_j}{\partial Y_k} = &\frac{\partial f}{\partial y}(t_j, Y_j, \theta_{[r]}(\alpha(t_j, Y_j))) \times \delta_{jk} + \\ &\frac{\partial f}{\partial v}(t_j, Y_j, \theta_{[r]}(\alpha(t_j, Y_j))) \times \frac{d\theta_{[r]}(\alpha(t_j, Y_j))}{dY_k},\end{aligned} \tag{2.29}$$

where δ_{jk} denotes the Kronecker symbol.

Using (2.28),

$$\frac{d\theta_{[r]}(\alpha(t_j, Y_j))}{dY_k} = \begin{cases} \theta'_{[r]}(\alpha(t_j, Y_j)) \times \frac{\partial \alpha}{\partial y}(t_j, Y_j) \times \delta_{jk}, & \text{for } \alpha(t_j, Y_j) < t_n, \\ \frac{d\mathcal{P}_n[Y, y^{[n]}](\alpha(t_j, Y_j))}{dY_k}, & \text{for } \alpha(t_j, Y_j) \geq t_n, \end{cases} \tag{2.30}$$

where $\theta'_{[r]}(\alpha) = \frac{d\theta_{[r]}(t)}{dt}(\alpha)$, and

$$\begin{aligned}\frac{d\mathcal{P}_n[Y, y^{[n]}](\alpha(t_j, Y_j))}{dY_k} &= \mathcal{P}'_n[Y, y^{[n]}](\alpha(t_j, Y_j)) \times \frac{\partial \alpha}{\partial y}(t_j, Y_j) \times \delta_{jk} + \\ &\quad \frac{\partial \mathcal{P}_n}{\partial Y_k}[Y, y^{[n]}](\alpha(t_j, Y_j)) \\ &= \theta'_{[r]}(\alpha(t_j, Y_j)) \times \frac{\partial \alpha}{\partial y}(t_j, Y_j) \times \delta_{jk} + \\ &\quad \frac{\partial \mathcal{P}_n}{\partial Y_k}[Y, y^{[n]}](\alpha(t_j, Y_j)),\end{aligned} \tag{2.31}$$

where $\mathcal{P}'_n[Y, y^{[n]}](\alpha) = \frac{d\mathcal{P}_n[Y, y^{[n]}](t)}{dt}(\alpha)$.

Combining (2.29), (2.30) and (2.31),

$$\begin{aligned}\frac{\partial F_j}{\partial Y_k} = &\left[\frac{\partial f}{\partial y}(t_j, Y_j, \theta_{[r]}(\alpha[j])) \right. \\ &\left. + \frac{\partial f}{\partial v}(t_j, Y_j, \theta_{[r]}(\alpha[j])) \times \theta'_{[r]}(\alpha[j]) \times \frac{\partial \alpha}{\partial y}(t_j, Y_j)\right] \times \delta_{jk} + \\ &\begin{cases} 0, & \text{for } \alpha[j] < t_n, \\ \frac{\partial f}{\partial v}(t_j, Y_j, \theta_{[r]}(\alpha[j])) \times \frac{\partial \mathcal{P}_n}{\partial Y_k}[Y, y^{[n]}](\alpha[j]), & \text{for } \alpha[j] \geq t_n, \end{cases}\end{aligned} \tag{2.32}$$

where $\alpha[j] = \alpha(t_j, Y_j)$.

The first component of $\frac{\partial F_j}{\partial Y_k}$ in (2.32) has the same structure as that arising in the corresponding IVP problem, due to the presence of δ_{jk}. For delays smaller than

the step size, adding the second component of (2.32) results in a different structure for the Jacobian. For an efficient implementation, usually an approximation of this Jacobian is used. Therefore, one might ignore the second component completely, or replace it with something with the same structure ($H\delta_{jk}$, for any, but usually a heuristically chosen, matrix H).

At present, we do not know of any mathematical or numerical evidence that shows the safety of this replacement (i.e., not causing divergence of iterations during the solution process). The second term represents the numerical complexity inherited from the presence of delays in the model. We are currently seeking possible problems (i.e., stiff DDEs) that necessarily need this second component for the convergence of the iterations. We are also looking for techniques to incorporate this term efficiently in the computation process, while respecting a generic interface (to be designed) between the DDE solver and IVP solvers.

2.5.3 Neutral Problems

For a system of NDDEs (1.6), including a term $y'(t-\sigma)$ or $y'(\alpha)$ as an argument of f, will lead to modifications to some of our expressions. Here, we discuss the important changes.

In the vanishing delay case for explicit formulas, if the vanishing delay appears in a derivative term, the same process can be applied, with $\mathcal{P}'_n$ playing a similar role for $y'(\alpha)$ as $\mathcal{P}_n$ did for $y(\alpha)$. However, if we have to use Equation (2.24), we can use the approximation

$$y'(\alpha(t, y(t))) \approx y'(t_n).$$

In this case, $y'(t_n)$ should be provided as an external value. We do not consider the case of an NDDE with accumulated discontinuities at a vanishing delay, because such problems can be mathematically ill-posed.

For delays appearing in derivatives, the second term in Equation (2.29) should be changed to

$$\frac{\partial f}{\partial w}(t_j, Y_j, \theta'_{[r]}(\alpha(t_j, Y_j))) \times \frac{d\theta'_{[r]}(\alpha(t_j, Y_j))}{dY_k},$$

where $f = f(t, y, w)$, and also all instances of $\theta_{[r]}$ and $\mathcal{P}_n$ should be replaced with $\theta'_{[r]}$ and $\mathcal{P}'_n$, respectively, in the subsequent Equations (2.30, 2.31, 2.32).

2.5.4 Extension to Multiple Delays

For a general system of DDEs with multiple delays (1.5), one must define corresponding switching functions for each delay, separately. The implementation of an effective DDE method becomes much more complex, since we have to monitor multiple switching functions and find the first one that is triggered on each step.

The equations for the implicit solver can then be derived by taking the sum over all delays α of the term,

$$\frac{\partial f}{\partial v_\alpha}(t_j, Y_j, \theta_{[r]}(\alpha(t_j, Y_j))) \times \frac{d\theta_{[r]}(\alpha(t_j, Y_j))}{dY_k},$$

appearing in (2.29).

Chapter 3

Accurate Sensitivity Analysis for DDEs

3.1 Introduction

Sensitivity analysis is concerned with the study of the relationship between infinitesimal changes in model parameters and changes in model outputs. Sensitivity information can be used to estimate which parameters are most influential in affecting the behavior of the simulation. Such information is crucial for experimental design, data assimilation, reduction of complex nonlinear models, and evaluating optimization gradients and Jacobians in the setting of dynamic optimization and parameter estimation.

Sensitivity analysis also plays a very important role in dynamical systems. For example, periodic orbits, the Lyapunov exponents, chaos indicators, and bifurcation analysis are fundamental components of a complete study of a dynamical system and their investigations require computation of the sensitivities with respect to the initial conditions of the problem (see [7] and references therein for more details).

There are two main approaches to sensitivity analysis: *Forward sensitivity analysis* and *adjoint sensitivity method*. The adjoint sensitivity method is advantageous where

the sensitivities of a few quantities with respect to a large number of parameters are needed. In the following we briefly describe the situation where the adjoint method is used, and then we study the details of the forward approach adapted for delay differential equations.

3.2 Adjoint Sensitivity Analysis

Suppose that $y(t)$ is a solution of a differential equation and we wish to evaluate the gradient $\frac{\partial H}{\partial p_i}$ of an objective function

$$H(\mathbf{p}) = \int_{t_0}^{t_f} h(t, y, \mathbf{p}) dt,$$

or alternatively the gradient $\frac{\partial h}{\partial p_i}$ of a function

$$h(t, y, \mathbf{p}),$$

at time t_f. If $\dim(h) < \dim(y)$ and we need $\frac{\partial}{\partial p}$ for many parameters, then the adjoint approach that does not explicitly compute the sensitivities of state variable y with respect to parameters is usually more efficient.

Adjoint sensitivity analysis involves integration of the original differential equations forward in time followed by the integration of the so-called adjoint equations backwards in time (see [16] for details).

3.3 A Review of Sensitivity Computation Techniques for ODEs

3.3.1 Finite Difference Approach

The simplest way of calculating sensitivity coefficients is to use the finite difference approximation,

$$\{\frac{\partial}{\partial p_i}\}y(t;\mathbf{p}) \approx \frac{y(t;\mathbf{p}+\mathbf{e}_i\Delta p_i) - y(t;\mathbf{p})}{\Delta p_i}. \tag{3.1}$$

This technique is very easy to implement because it requires no extra code beyond the original model solver, although it does require more applications of the underlying solver (one for each partial derivative approximation and one for $y(t;\mathbf{p})$), with the same value of a chosen tolerance Tol. However, when computations are done in finite precision, the presence of rounding errors prevents the use of a very small perturbation, Δp_i. Therefore, the approximation is only accurate to $\mathcal{O}(\sqrt{Tol})$ with the best choice for Δp_i [32].

Higher-order finite difference approximations, such as central difference formula, can also be used, but the problem of selecting a suitable perturbation does not have an easily implementable answer. Moreover, the obtainable accuracy in computing these sensitivity values will be much less than the tolerance of the numerical solution.

3.3.2 Internal Differentiation Approach

In this approach, the governing equations for the first order sensitivity coefficients are derived by differentiation of (1.7) with respect to the model parameter p_i and applying the chain rule and Clairaut's theorem, yielding

$$s_i' = \frac{\partial f}{\partial y}s_i + \frac{\partial f}{\partial p_i}, \quad s_i(t_0) = \frac{\partial y_0(\mathbf{p})}{\partial p_i}, \qquad (i = 1, \dots \mathcal{L}), \tag{3.2}$$

or in the matrix form

$$S'(t) = J(t)S(t) + B(t), \tag{3.3}$$

where $S(t)$ is the $\mathcal{M} \times \mathcal{L}$ sensitivity coefficient matrix ($S_{ij} \equiv \frac{\partial y_i}{\partial p_j}$), $J(t)$ is the $\mathcal{M} \times \mathcal{M}$ Jacobian matrix ($J_{ij} \equiv \frac{\partial f_i}{\partial y_j}$), $B(t)$ is an $\mathcal{M} \times \mathcal{L}$ matrix of partial derivatives ($B_{ij} \equiv \frac{\partial f_i}{\partial p_j}$).

The equations for the second order sensitivity coefficients are derived by further differentiation, applying $\frac{\partial}{\partial \mathbf{p}}$ to (3.3), the result in terms of *Kronecker* products is

$$R'(t) = A(t)S(t) + (I_{\mathcal{L}} \otimes J(t))R(t) + K(t), \tag{3.4}$$

where $R(t)$ is an $(\mathcal{LM}) \times \mathcal{L}$ sensitivity coefficient matrix of second order ($R_{ij} \equiv \frac{\partial s_i}{\partial p_j}$), $A(t)$ is an $(\mathcal{LM}) \times \mathcal{M}$ partial derivatives of the Jacobian matrix ($A_{ij} \equiv \frac{\partial J_i(t)}{\partial p_j}$, where, $J_i = \frac{\partial f_k}{\partial y_i}$) and $K(t)$ is an $(\mathcal{LM}) \times \mathcal{L}$ partial derivative matrix ($K_{ij} \equiv \frac{\partial \mathbf{b}_i(t)}{\partial p_j}$, where, $\mathbf{b}_i = \frac{\partial f_k}{\partial p_i}$).

To find the sensitivity coefficient matrices S and R, we need to solve the IVP (1.7) simultaneously with the system (3.3 and/or 3.4) using an appropriate differential equation solver. Leis and Kramer [42] have implemented ODESSA package based on the Fortran 77 initial value solver LSODE [53]. Serban and Hindmarsh [55] have extended the package CVODE [17], written in C, to develop the package CVODES. Both ODESSA and CVODES have an option for providing subroutines for the computation of partial derivatives ($\frac{\partial f}{\partial y}$ and $\frac{\partial f}{\partial p_i}$), otherwise these values are computed using finite difference-based approximations. Lee and Hovald [41] have extended the package PVODE [15], which is a parallel version of CVODE, to add the capability of sensitivity analysis. The new package SensPVODE uses the techniques of automatic differentiation (AD) to produce the codes needed for the computation of partial derivatives. The above mentioned methods compute only first order sensitivities. Barrio [7], using extended rules of AD, presented a new approach based on a Taylor method for computing the sensitivities of any order.

3.4 Sensitivity Computation for General DDEs

3.4.1 General Parameterized DDEs

Consider the general case of a system of state-dependent neutral delay differential equations (NDDEs),

$$\begin{aligned} y'(t;\mathbf{p}) &= f(t, y(t;\mathbf{p}), y(\alpha(t,y;\mathbf{p});\mathbf{p}) \\ &\qquad , y'(\alpha(t,y;\mathbf{p});\mathbf{p});\mathbf{p}) \ \text{ for } \ t \geq t_0(\mathbf{p}), \\ y(t;\mathbf{p}) &= \phi(t;\mathbf{p}), \ \text{ for } \ t < t_0(\mathbf{p}), \\ y'(t;\mathbf{p}) &= \phi'(t;\mathbf{p}), \ \text{ for } \ t < t_0(\mathbf{p}), \\ y(t_0) &= y_0(\mathbf{p}), \end{aligned} \tag{3.5}$$

where $\alpha(t,y;\mathbf{p})$ is a $\{\nu+\omega\}$-vector of delay arguments, y and f are $\mathcal{M}$-vector of functions and $\mathbf{p}$ is an $\mathcal{L}$-vector of parameters.

3.4.2 First Order Sensitivities

The governing equations for the first order sensitivity coefficients are derived by differentiation of (3.5) with respect to the model parameters and applying the chain rule and Clairaut's theorem, yielding (in the matrix form)

$$\begin{aligned} S'(t) = \frac{\partial f}{\partial y}S(t) &+ \sum_{k=1}^{\nu}\left[\frac{\partial f}{\partial y(\alpha_k)}\left(y'(\alpha_k)\left(\frac{\partial \alpha_k}{\partial y}S(t)+\frac{\partial \alpha_k}{\partial \mathbf{p}}\right)+S(\alpha_k)\right)\right] \\ &+ \sum_{k=\nu+1}^{\nu+\omega}\left[\frac{\partial f}{\partial y'(\alpha_k)}\left(y''(\alpha_k)\left(\frac{\partial \alpha_k}{\partial y}S(t)+\frac{\partial \alpha_k}{\partial \mathbf{p}}\right)+S'(\alpha_k)\right)\right] \\ &+ \frac{\partial f}{\partial \mathbf{p}}, \end{aligned} \tag{3.6}$$

where $S(t)$ is the $\mathcal{M}\times\mathcal{L}$ sensitivity coefficient matrix ($S_{ij} \equiv \frac{\partial y_i}{\partial p_j}$), $\frac{\partial f}{\partial y}$ is the $\mathcal{M}\times\mathcal{M}$ Jacobian matrix ($[\frac{\partial f}{\partial y}]_{ij} \equiv \frac{\partial f_i}{\partial y_j}$), $\frac{\partial f}{\partial y(\alpha_k)}$ is the $\mathcal{M}\times\mathcal{M}$ delayed Jacobian matrix ($[\frac{\partial f}{\partial y(\alpha_k)}]_{ij} \equiv \frac{\partial f_i}{\partial y_j(\alpha_k)}$), $\frac{\partial \alpha_k}{\partial y}$ is a $1\times\mathcal{M}$ row-vector of partial derivatives ($[\frac{\partial \alpha_k}{\partial y}]_{1j} \equiv \frac{\partial \alpha_k}{\partial y_j}$), $\frac{\partial \alpha_k}{\partial \mathbf{p}}$ is a $1\times\mathcal{L}$ row-vector of partial derivatives ($[\frac{\partial \alpha_k}{\partial \mathbf{p}}]_{1j} \equiv \frac{\partial \alpha_k}{\partial p_j}$) and $\frac{\partial f}{\partial \mathbf{p}}$ is an $\mathcal{M}\times\mathcal{L}$ matrix of partial derivatives ($[\frac{\partial f}{\partial \mathbf{p}}]_{ij} \equiv \frac{\partial f_i}{\partial p_j}$).

To find the sensitivity coefficient matrix S we need to solve the delay differential system (3.6) simultaneously with the system (3.5), with associated initial functions

$$\begin{aligned} S(t) &= \frac{\partial \phi(t;\mathbf{p})}{\partial \mathbf{p}}, \quad \text{for } t < t_0(\mathbf{p}), \\ S'(t) &= \frac{\partial \phi'(t;\mathbf{p})}{\partial \mathbf{p}}, \quad \text{for } t < t_0(\mathbf{p}), \\ S(t_0) &= \frac{\partial y_0(\mathbf{p})}{\partial \mathbf{p}}. \end{aligned}$$

3.4.3 Second Order Sensitivities

The equations for the second order sensitivity coefficients can be derived by further differentiation, applying $\frac{\partial}{\partial \mathbf{p}}$ to (3.6). The result is a system of linear DDEs very similar in structure to (3.6), but much more complicated as it contains double sums and many mixed partial derivatives and is usually described in terms of *Kronecker* products.

3.4.4 Handling C^0 Discontinuities in Sensitivities

The discontinuities that arise in simulation of DDEs can also propagate to the sensitivity coefficients and a similar treatment is required to perform reliable computations. However, when we integrate a usual system of DDEs (3.5), no discontinuity of zero order (i.e. a discontinuity in the solution values) can appear after the starting point. But for sensitivities we may have C^0 discontinuities, because the differential equations (3.6) are not valid at some points. These are the points where Clairaut's theorem is not applicable. At these points we may have C^0 jumps. It is the appearance of C^0 discontinuities that makes the task of computing sensitivities challenging. In this section we will describe the source of C^0 discontinuities and compute the size of the jumps at these points of discontinuity. The integration then can be restarted with new computed starting values for the sensitivity equations.

Barton's Formula for Hybrid ODE Systems

Tolsma and Barton [60] have considered extensions to the classical sensitivity theory that define the parametric sensitivity of discontinuous systems. Consider the general case where a transition is triggered by a zero crossing of an event function $g(t, y, y'; \mathbf{p})$ at the point λ, and let $y(\lambda^-)$, $y(\lambda^+)$ be the values of the state variables before and after the event. If the state transition is continuous

$$y(\lambda^+) = y(\lambda^-), \tag{3.7}$$

then differentiating both sides of this equation with respect to the parameter p_l and some rearrangement yields,

$$\frac{\partial y}{\partial p_l}(\lambda^+) = \frac{\partial y}{\partial p_l}(\lambda^-) + \left(y'(\lambda^-) - y'(\lambda^+)\right)\frac{d\lambda}{dp_l}, \tag{3.8}$$

where $\frac{d\lambda}{dp_l}$ represents the sensitivity of the event time with respect to the parameter p_l. To compute the value of $\frac{d\lambda}{dp_l}$, we can differentiate $g(t, y, y'; \mathbf{p}) = 0$ w.r.t. p_l and rearrange terms to obtain,

$$\frac{\partial g}{\partial y'}\left(\frac{\partial}{\partial t}\left(\frac{\partial y}{\partial p_l}\right) + y''\frac{d\lambda}{dp_l}\right) + \frac{\partial g}{\partial y}\left(\frac{\partial y}{\partial p_l} + y'\frac{d\lambda}{dp_l}\right) + \frac{\partial g}{\partial p_l} + \frac{\partial g}{\partial t}\frac{d\lambda}{dp_l} = 0, \tag{3.9}$$

which is a linear equation w.r.t. $\frac{d\lambda}{dp_l}$ (Tolsma and Barton [60]).

Adapting for jumps in DDEs

In Section 2.4 we showed that DDEs can be considered as a special subclass of discontinuous IVPs. Here we use those results to obtain equations for the jumps in the sensitivities.

Consider the case where a state transition arises in DDEs triggered by the propagation of a discontinuity of the solution. Since discontinuity points are determined by the value of parameters, we can define,

$$\Lambda(\mathbf{p}) \equiv \{\cdots < \lambda_{-2}(\mathbf{p}) < \lambda_{-1}(\mathbf{p}) < \lambda_0(\mathbf{p}) = t_0(\mathbf{p}) < \lambda_1(\mathbf{p}) < \lambda_2(\mathbf{p}) < \cdots\}, \tag{3.10}$$

which is the parameterized variant of (2.12). Considering the parameterized variant of event functions (Equation (2.13)), and letting i denote the index of the chosen minimum of Equation (2.14), $g_i(t, y; \mathbf{p})$ triggers a transition of the state variables at λ_r. Then, Equation (3.9) (using the fact that $\frac{\partial g_i}{\partial y'} = 0$), reduces to

$$\frac{\partial g_i}{\partial y}\frac{\partial y}{\partial p_l} + \frac{\partial g_i}{\partial p_l} + \left[\frac{\partial g_i}{\partial y}y' + \frac{\partial g_i}{\partial t}\right]\frac{d\lambda_r(\mathbf{p})}{dp_l} = 0. \tag{3.11}$$

For the partial derivatives we have the relations

$$\frac{\partial g_i}{\partial y} = \frac{\partial \alpha}{\partial y}, \tag{3.12}$$

$$\frac{\partial g_i}{\partial p_l} = \frac{\partial \alpha}{\partial p_l} - \frac{d\lambda_i(\mathbf{p})}{dp_l}, \tag{3.13}$$

$$\frac{\partial g_i}{\partial t} = \frac{\partial \alpha}{\partial t}. \tag{3.14}$$

Substituting in (3.11), we obtain

$$\frac{\partial \alpha}{\partial y}\frac{\partial y}{\partial p_l} + \frac{\partial \alpha}{\partial p_l} - \frac{\partial \lambda_i(\mathbf{p})}{\partial p_l} + \left[\frac{\partial \alpha}{\partial y}y' + \frac{\partial \alpha}{\partial t}\right]\frac{d\lambda_r(\mathbf{p})}{dp_l} = 0. \tag{3.15}$$

Assuming that $\frac{\partial \alpha}{\partial y}y' + \frac{\partial \alpha}{\partial t} \neq 0$, we are able to solve this linear equation to get

$$\frac{d\lambda_r(\mathbf{p})}{dp_l} = -\frac{\frac{\partial \alpha}{\partial y}\frac{\partial y}{\partial p_l} + \frac{\partial \alpha}{\partial p_l} - \frac{d\lambda_i(\mathbf{p})}{dp_l}}{\frac{\partial \alpha}{\partial y}y' + \frac{\partial \alpha}{\partial t}}, \tag{3.16}$$

and for the first discontinuity point ($\lambda_0(\mathbf{p}) = t_0(\mathbf{p})$) we have

$$\frac{d\lambda_0(\mathbf{p})}{dp_l} = \frac{\partial t_0(\mathbf{p})}{\partial p_l}, \tag{3.17}$$

and for the discontinuities in the history,

$$\frac{d\lambda_r(\mathbf{p})}{dp_l} = \frac{\partial \lambda_r(\mathbf{p})}{\partial p_l}, \quad r = \ldots, -2, -1, \tag{3.18}$$

are independently computable, since $\lambda_r(\mathbf{p})$ is given as an input function for $r = \ldots, -2, -1$.

Equation (3.8) for DDEs becomes

$$\frac{\partial y}{\partial p_l}(\lambda_r^+) = \frac{\partial y}{\partial p_l}(\lambda_r^-) + \left(y'(\lambda_r^-) - y'(\lambda_r^+)\right)\frac{d\lambda_r(\mathbf{p})}{dp_l}, \tag{3.19}$$

and the steps for integrating the first order sensitivity equations can be described as the following:

Algorithm 1: Computing First Order Sensitivities for DDEs

input : a general DDE (3.5); an approach for deriving and integrating the sensitivity equations with discontinuity tracking capability.

output: first order sensitivity coefficients.

1.1 Initialize ($\lambda_0 = t_0(\mathbf{p})$).

1.2 $r \leftarrow 1$.

1.3 Integrate the equations up to a C^1 discontinuity point (λ_r).

1.4 Update the state variables (sensitivities) using

$$\frac{\partial y}{\partial p_l}(\lambda_r^+) = \frac{\partial y}{\partial p_l}(\lambda_r^-) + \left(y'(\lambda_r^-) - y'(\lambda_r^+)\right)\frac{d\lambda_r(\mathbf{p})}{dp_l}, \qquad (l = 1, \dots \mathcal{L}).$$

1.5 $r \leftarrow r + 1$ and restart (step **1.3**).

Computing $y''(\alpha_k)$

The term $y''(\alpha_k)$ inside the Formula (3.6), needed for state-dependent and parameter-dependent NDDEs, can be computed using a similar method, namely differentiating (3.5) with respect to t and some rearrangements, yielding

$$\begin{aligned} x'(t) = \frac{\partial f}{\partial y} + \frac{\partial f}{\partial y}x(t) + \sum_{k=1}^{\nu}\frac{\partial f}{\partial y(\alpha_k)}x(\alpha_k)\left(\frac{\partial \alpha_k}{\partial y}x(t) + \frac{\partial \alpha_k}{\partial t}\right) \\ + \sum_{k=\nu+1}^{\nu+\omega}\frac{\partial f}{\partial y'(\alpha_k)}x'(\alpha_k)\left(\frac{\partial \alpha_k}{\partial y}x(t) + \frac{\partial \alpha_k}{\partial t}\right), \end{aligned} \tag{3.20}$$

where $x(t) = y'(t)$. The associated initial functions are

$$\begin{aligned} x(t) &= \phi'(t;\mathbf{p}), \quad \text{for } t < t_0(\mathbf{p}), \\ x'(t) &= \phi''(t;\mathbf{p}), \quad \text{for } t < t_0(\mathbf{p}), \\ x(t_0) &= y'(t_0). \end{aligned}$$

Equation (3.20), when required, is integrated simultaneously with other equations, providing the required values of $y''(\alpha_k) = x'(\alpha_k)$.

Handling Jumps in y''

Choosing $x(t) = y'(t)$ as a new state variable and integrating using the driven differential equations works well if $x(t)$ has no C^0 discontinuities (i.e. discontinuities in the value) after the starting time t_0. Otherwise, these jumps in the value cannot be captured by integrating using differential equations. In this case, which is inevitable when the original system is a system of NDDEs, these jumps should be treated as discrete events. Each time an event of this type is triggered, the initial values for the continued integration must be updated using the relation $x(\lambda^+) = y'(\lambda^+)$.

Chapter 4

Reliable Parameter Estimation for DDEs

4.1 Introduction

In many applications the system of differential equations used for modeling the underlying phenomenon involves some unknown parameters that appear in the equations (see §1.2.2). If we have observed the phenomenon and collected data, then we can try to determine the unknown parameters, by fitting the model equation to data. This parameter determination/estimation is an example of an "inverse" problem and may be resolved by minimizing a least-squares objective function (see §1.2.5). However, parameter estimation/fitting can be inherently complicated for many reasons. For instance the parameters may not be *identifiable*, which causes numerical difficulties in the optimization process. Furthermore, the efficiency of a numerical technique may depend strongly on the location and number of data points.

In this chapter after reviewing some existing techniques for parameter estimation of problems involving systems of ODEs and DDEs, we discuss the necessity of considering possible discontinuities in the objective function in the DDEs case. We then introduce a

new technique for handling the resulting discontinuous least-squares problem efficiently.

In the following, we limit our discussions to basic least-squares methods, without regularization.

4.2 A Review of Parameter Estimation Techniques

4.2.1 Parameter Estimation Techniques for ODEs

There are two main approaches for the parameter identification problem for ODEs. Here we give a brief description of each approach.

- **The initial value approach**: An optimization method is used to minimize $W(\mathbf{p})$ of Equation (1.3) with respect to $\mathbf{p}$ with the following steps,

  1. Choose an initial guess for the parameters.
  2. Solve model Equations (1.7).
  3. Check optimality conditions, (if satisfied, stop).
  4. Choose a better value for the parameters and continue with step (2).

  The model trajectory, $y(t;\mathbf{p})$, can be very sensitive to the parameters, especially in the case of nonlinear systems, and the objective function can exhibit a strong nonlinear dependence on the parameters $\mathbf{p}$, which can introduce several local minima, apart from the global one that corresponds to the true optimal parameters. Therefore, if the initial guess for the parameters is far from the correct one, the trial trajectory can soon lose contact with the measurements.

  The gradient or the Hessian of the objective function (if required by the optimization method) is computed using,

$$\left(\frac{\partial W(\mathbf{p})}{\partial p_l}\right) = -2\sum_i \left[Y(\gamma_i) - y(\gamma_i;\mathbf{p})\right]\left(\frac{\partial y(\gamma_i;\mathbf{p})}{\partial p_l}\right), \tag{4.1}$$

or

$$\left(\frac{\partial^2 W(\mathbf{p})}{\partial p_l \partial p_m}\right) = 2\sum_i \Bigg[\left(\frac{\partial y(\gamma_i;\mathbf{p})}{\partial p_l}\right)\left(\frac{\partial y(\gamma_i;\mathbf{p})}{\partial p_m}\right) - [Y(\gamma_i) - y(\gamma_i;\mathbf{p})]\left(\frac{\partial^2 y(\gamma_i;\mathbf{p})}{\partial p_l \partial p_m}\right)\Bigg], \tag{4.2}$$

and the sensitivity equations are usually used to provide the required values of $\frac{\partial y(\gamma_i;\mathbf{p})}{\partial p_l}$ or $\frac{\partial^2 y(\gamma_i;\mathbf{p})}{\partial p_l \partial p_m}$. Alternately, one can use divided differences or other methods of sensitivity computations.

- **Multiple shooting**: The technique of multiple shooting was developed in the context of parameter estimation in [10] (see also Krogh et al. [39]). The fitting interval is partitioned into many subintervals, each having its own initial values and the measurements are used to get starting guesses for them. The parameters of the ODE are held constant for all subintervals. This procedure leads to an initially discontinuous trajectory which is, however, close to the measurements. In an iterative process, the algorithm minimizes $W(\mathbf{p})$ on the one hand and enforces the continuity of the full trajectory on the other hand. The freedom of intermediate discontinuities allows the trajectory to stay close to the data. Divergences are avoided and the danger of local minima is reduced. For the reduction of $W(\mathbf{p})$, a generalized Gauss-Newton method is employed. Horbelt et al. [37] have used this approach for identifying physical properties of a CO_2 laser. They have shown that this method can be very effective, especially when the data is noisy.

4.2.2 Techniques for Parameter Estimation of DDEs

Most techniques for parameter identification of DDEs are derived directly from a corresponding ODE approach. However, there are methods proposed specially for DDEs. Here we discuss the adaptations of the methods of Section (4.2.1) and also one special method.

- **The adapted initial value approach** (Baker and Paul [5]): Discontinuities arising from the initial point $t_0(\mathbf{p})$ (and the initial function $\phi(t;\mathbf{p})$), may propagate into $W(\mathbf{p})$ via the solution values $\{y(\gamma_i;\mathbf{p})\}$. The first and second order partial derivatives of the objective function in the case of the parametric DDE (1.8) are

$$\left(\frac{\partial W(\mathbf{p})}{\partial p_l}\right)_{\pm} = -2\sum_i [Y(\gamma_i) - y(\gamma_i;\mathbf{p})] \left(\frac{\partial y(\gamma_i;\mathbf{p})}{\partial p_l}\right)_{\pm}, \tag{4.3}$$

and

$$\begin{aligned}\left(\frac{\partial^2 W(\mathbf{p})}{\partial p_l \partial p_m}\right)_{\pm\pm} = 2\sum_i \Bigg[&\left(\frac{\partial y(\gamma_i;\mathbf{p})}{\partial p_l}\right)_{\pm} \left(\frac{\partial y(\gamma_i;\mathbf{p})}{\partial p_m}\right)_{\pm} \\ &- [Y(\gamma_i) - y(\gamma_i;\mathbf{p})] \left(\frac{\partial^2 y(\gamma_i;\mathbf{p})}{\partial p_l \partial p_m}\right)_{\pm\pm}\Bigg],\end{aligned} \tag{4.4}$$

where the $\pm$ means right- and left-hand derivatives.

It is clear that a jump in a first or second partial derivative of $W(\mathbf{p})$ can occur if

$$s_l(\gamma_i;\mathbf{p}) := \frac{\partial y(t;\mathbf{p})}{\partial p_l},$$

or

$$r_{lm}(t;\mathbf{p}) := \frac{\partial^2 y(t;\mathbf{p})}{\partial p_l \partial p_m},$$

has a jump at $t = \gamma_i$ for some i. Investigating different scenarios for the propagation of discontinuities to the objective function suggests the general rule that if a discontinuity point $\lambda_r(\mathbf{p})$ coincides with one of the data points γ_i, and $\lambda_r(\mathbf{p})$ varies as some parameters vary, then $W(\mathbf{p})$ has a jump in its partial derivatives that correspond to the varying parameters. It should not be surprising then, that optimization codes that assume the smoothness of the objective function may fail, when applied to parameter estimation on problems involving DDEs.

- **Multiple shooting** (Horbert et al. [38]): The problem of the matching conditions necessary in this context is solved by using cubic splines to parameterize the initial curves and then formulating the continuity of the trajectory in terms of these spline variables. When the starting guesses for the dynamical parameters are far

from the true values, a two-phase procedure is used. During the first iterations of the optimization, the spline variables are held fixed because they are expected to be estimated well from the data. After the algorithm has converged for the first time, they are released and fitted together with the other variables until the final convergence is achieved. It is shown through several examples that this method can attain rapid convergence in the case of noisy data.

- **The full discretization approach** (Murphy [45]): Linear splines are used to describe solution and delay functions. As a result, the problem becomes a very large minimization problem. The number of mesh points is increased gradually to be able to satisfy the specified error tolerance. Although the method has the generality of dealing with any of several types of unknown parameters, it suffers from the heavy computations, slow convergence rate, and possibility of being trapped in a local minimum.

4.3 Handling the Non-smoothness in Parameter Estimation of DDEs

The non-smoothness issue, noticed by Baker and Paul [5] in adapting the initial value approach for DDEs, is also present in other aforementioned parameter estimation techniques. As a result, the theoretical assumptions needed for the justification of numerical results are not satisfied for those techniques. Furthermore, in some cases the efficiency of the numerical process is seriously affected by the non-smoothness. In this section we discuss this issue in detail and propose a new technique for dealing with propagated discontinuities in objective function for DDEs. Although the ideas are general and can be incorporated in all parameter estimation methods, we only give the details for the adapted initial value approach.

4.3.1 Algorithms for Nonlinear Least-Squares Problems

There are many algorithms to solve the unconstrained nonlinear least squares problem

$$\min_{\mathbf{p}} W(\mathbf{p}) = \sum_i \left[Y(\gamma_i) - y(\gamma_i; \mathbf{p})\right]^2, \tag{4.5}$$

or constrained nonlinear least squares problem

$$\begin{aligned} \min_{\mathbf{p}} W(\mathbf{p}) = \sum_i \left[Y(\gamma_i) - y(\gamma_i; \mathbf{p})\right]^2, \\ c_j(\mathbf{p}) = 0, \quad j \in \mathcal{E}, \\ c_j(\mathbf{p}) \geq 0, \quad j \in \mathcal{I}, \end{aligned} \tag{4.6}$$

among which the Levenberg-Marquardt algorithm for unconstrained problems and variations of sequential quadratic programming (SQP) algorithm for unconstrained and constrained problems have been shown to be very effective and efficient in practice.

In both cases the smoothness of functions involved in the problem (i.e., the objective function and constraints), is a necessary assumption (at least to second order, C^2). If some of the smoothness assumptions are violated then there is no guarantee that these algorithms will converge to a local optimum.

4.3.2 Avoiding the Non-smoothness

Figure (4.1-top) shows a typical situation for a data point γ_i. During the minimization process, when we need to evaluate $W(\mathbf{p})$ (or its partial derivatives) for the current value of $\mathbf{p}$, we need to have $y(t; \mathbf{p})$ (or its partial derivatives) at γ_i. If

$$\lambda_r(\mathbf{p}) \leq \gamma_i < \lambda_{r+1}(\mathbf{p}), \tag{4.7}$$

then the portion of $y(t; \mathbf{p})$ (or its partial derivatives) over the interval $[\lambda_r(\mathbf{p}), \lambda_{r+1}(\mathbf{p}))$ is involved.

To allow changes in $\mathbf{p}$ to be incorporated in a way that does not allow $W(\mathbf{p})$ to pass non-smooth regions, we need to add some constraints that prevent any changes in the

ordering of data points and discontinuity points. That is we would like to respect the ordering (4.7), for all data points, while the underlying least-squares method is searching for the best fit.

For these constraints to be useful in practice we need them to be differentiable. Furthermore, most implementations of SQP assume that the function and constraints can be evaluated outside the feasible set.

Differentiating the new constraints is possible as we have already shown that $\frac{d\lambda_r(\mathbf{p})}{dp_l}$ can be computed recursively (Equation (3.16)).

If we attempt to evaluate $W(\mathbf{p})$ outside the feasible region we violate the smoothness criteria. A possible remedy is to use the continuous extension of $y(t;\mathbf{p})$ and its partial derivatives for evaluations of $W(\mathbf{p})$ or its partial derivatives outside the feasible set, as shown in Figure (4.1-bottom). This may change the actual function $W(\mathbf{p})$ which we are minimizing, but changes are only in the infeasible set which we do not consider as valid solutions.

The whole process of optimization is then to start from a point $\mathbf{p}_0$ and conduct changes in $\mathbf{p}$ so that we safely pass through non-smooth regions.

4.3.3 Safe Guidance of Optimization

Let

$$ContinuityConstraints[\mathbf{p}_{\text{base}}](\mathbf{p}_{\text{trial}}) = \{\lambda_{r[\mathbf{p}_{\text{base}}]}(\mathbf{p}_{\text{trial}}) \leq \gamma_i \leq \lambda_{r[\mathbf{p}_{\text{base}}]+1}(\mathbf{p}_{\text{trial}})\},$$

be the set of all constraints needed to respect the initial ordering coming from $\mathbf{p}_{\text{base}}$ as starting point for the SQP optimizer. These constraints as well as their partial derivatives can be evaluated at any $\mathbf{p}_{\text{trial}}$ inside the optimizer.

The steps for finding a local optimum can be described as the following:

1. Start with $\mathbf{p}_c \leftarrow \mathbf{p}_0$ and $addedConstraints \leftarrow ContinuityConstraints[\mathbf{p}_0]$.

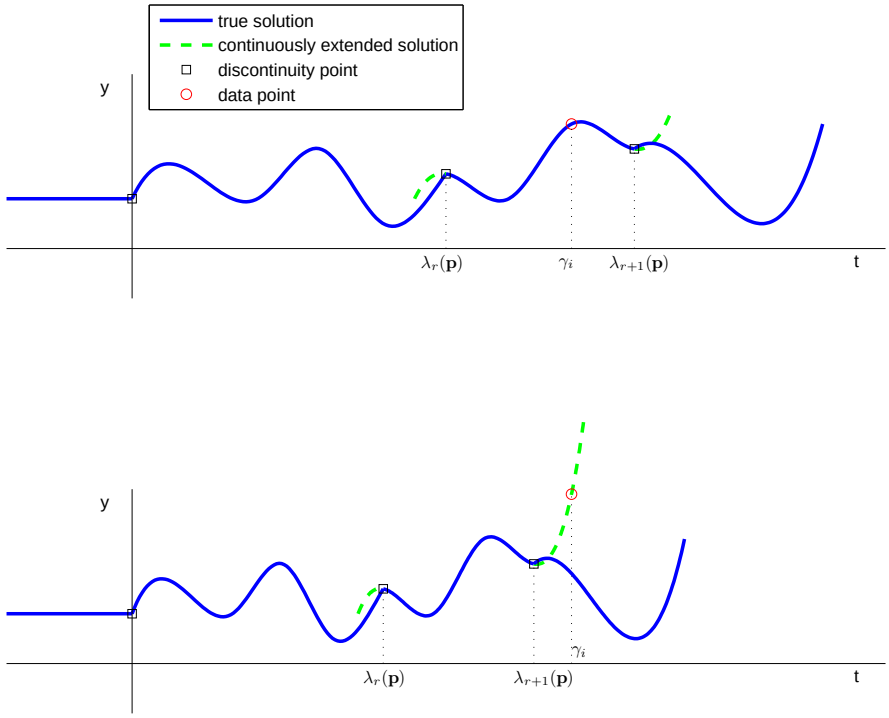

Figure 4.1: top : a typical situation for a data point. bottom : evaluating a data point outside of its initial interval.

2. Run the SQP optimizer with $\mathbf{p}_c$ and $addedConstraints$, to get $\mathbf{p}_{new}$. At this point we have a local minimum, but with some added artificial constraints.

3. If some of $addedConstraints$ are active, jump to (4), otherwise stop with $\mathbf{p}^\star = \mathbf{p}_{new}$.

4. The computed local minimum may be improved if we relax our artificially added constraints. In other words, the optimality of the fitted parameter is not guaranteed for the original problem. Different strategies can be proposed at this stage for taking steps that insure an actual local optimum.

   - **Combinatorial Investigation**: Set all possible alternations of active artificial constraints and feed the resulting problems, one by one, to the optimizer.

If none of these possibilities can improve the fitness of the parameter, then the artificial constraints have no effect on the fitting process and a local optimum is guaranteed. Otherwise, choose one of the alternatives with a better solution and continue the process using the output of that alternative, from step (3).

- **Follow The Descent Direction**: Change all active constraints to the corresponding reversed constraints and continue with the optimizer with $\mathbf{p}_c \leftarrow \mathbf{p}_{new}$ from step (2).

  This simulates the movement to the local optimum direction in the corresponding unconstraint problem. However, the total movement in this case is partitioned into two series of changes (before and after reversing the constraints), bordered by the constraints.

Convergence, Efficiency and Reliability

The combinatorial investigation approach described above is convergent and its output is reliable. However, it may be very inefficient when the number of active constraints/discontinuities is large, as there may be many alternatives and checking all of them may be impractical. The alternate strategy of following the descent direction is convergent and efficient, but we have to check for possible loops. A loop may happen, meaning the optimizer cannot move away from the border point defined by switching constraints. We should monitor for this situation and terminate the process. It can easily be proven that if the objective function is convex, the switching point is then a local minimum (in this case). Otherwise, due to the presence of unpredictable saddle points, a more sophisticated strategy is needed to handle the general case (see [11], [26] and the references therein for more details).

Chapter 5

Software Design

Designing a modeling package with different functionalities (simulation, sensitivity analysis, parameter estimation) is a challenging and difficult process. The time consuming task of implementing efficient algorithms for doing core computations, designing a user-friendly interface, balancing generality and efficiency, and manageability of the code are just some of the issues. In this chapter we describe our designs for the three required components of our package and show how we are able to overcome some of these difficulties for DDEs.

5.1 Finding Dependencies

Table 5.1 shows the basic functionalities/tasks provided by our modeling package. Each task requires some inputs and produces some outputs and also some byproducts. Understanding this information is crucial for the development of an efficient and integrated design. For instance, the inputs and byproducts of simulation are subsets of the required inputs and byproducts of sensitivity analysis. Therefore, data structures required for sensitivity analysis can be an extension of those required for simulation. Another important observation is that some inputs for parameter estimation are byproducts of the sensitivity analysis. This should be reflected in the interface design of the sensitivity

analyzer as it must be easily callable from the parameter estimator.

Task	Inputs	Outputs	Byproducts
Simulation	f, ϕ	y	$\{\lambda_r\}$
(NDDEs)	(ϕ')		(y')
(stiff)	$(\frac{\partial f}{\partial y}, \frac{\partial f}{\partial y(\alpha)})$		
Sensitivity Analysis	$f, \phi, \frac{\partial \phi}{\partial \mathbf{p}}$, $\frac{\partial f}{\partial y}, \frac{\partial f}{\partial y(\alpha)}, \frac{\partial f}{\partial \mathbf{p}}, \frac{\partial \alpha}{\partial y}, \frac{\partial \alpha}{\partial \mathbf{p}}$	$\frac{\partial y}{\partial \mathbf{p}}$	$y, y', \{\lambda_r\}, \{\frac{\partial \lambda_r}{\partial \mathbf{p}}\}$
(NDDEs)	$(\phi', \frac{\partial \phi'}{\partial \mathbf{p}}, \phi'', \frac{\partial f}{\partial t}, \frac{\partial \alpha}{\partial t})$		$(y'', (\frac{\partial y}{\partial \mathbf{p}})')$
(second order)	(all mixed $\frac{\partial^2 f}{\partial \ldots \partial \ldots}, \frac{\partial^2 \alpha}{\partial \ldots \partial \ldots}$ and $\frac{\partial f}{\partial t}$)	$(\frac{\partial^2 y}{\partial \mathbf{p}^2})$	$(y'', \{\frac{\partial^2 \lambda_r}{\partial \mathbf{p}^2}\})$
(second order & NDDEs)	(all mixed $\frac{\partial^2 \phi}{\partial \ldots \partial \ldots}, \frac{\partial^2 \phi'}{\partial \ldots \partial \ldots}$)		$(y''', (\frac{\partial y}{\partial \mathbf{p}})'')$
Parameter Estimation	$y(\gamma_i; \mathbf{p}), \frac{\partial y(\gamma_i; \mathbf{p})}{\partial \mathbf{p}}, \{\frac{\partial \lambda_r}{\partial \mathbf{p}}\}$	$\mathbf{p}^\star$	
(large residuals)	$(\frac{\partial^2 y(\gamma_i; \mathbf{p})}{\partial \mathbf{p}^2})$		

Table 5.1: Dependencies Between Different Components of a Modeling Package

5.2 Software Architecture

There are three basic components in the package, which correspond to the three basic tasks. A user should provide the required functions and data structures representing the problem. The simulator works as an independent component and can be used directly. The sensitivity analyzer component, when called by a user, automatically sets up a simulation problem and then calls the simulator. The parameter estimator uses the simulator and sensitivity analyzer for computing the objective function and its partial derivatives (see Figure 5.1). Now, we briefly discuss the structure of each component.

Simulator: The simulator (Figure 5.2) uses an IVP solver as a basic step integrator. This IVP solver provides the basic step of a typical IVP integration, along with a continuous approximation and also an error control strategy, on each step. The

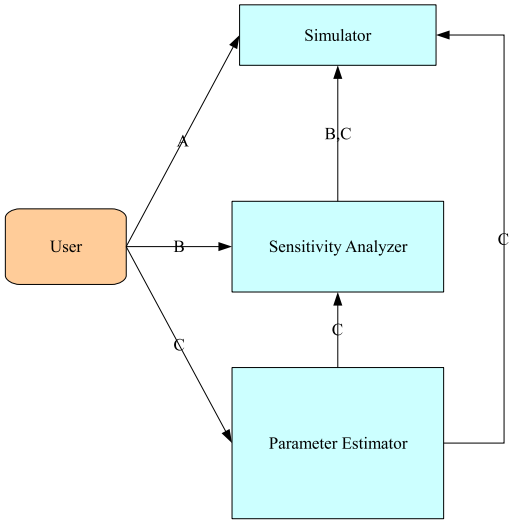

Figure 5.1: User Calls: Different letters (A,B,C) are used to specify the sequence of calls in different scenarios when using the package.

violation manager unit monitors for any possible violation of continuity assumptions forced by the IVP solver during each step. Then, based on reported violations, the discontinuity detector module locates the first discontinuity point. The suggested returned step from the IVP solver is partitioned at that discontinuity point using interpolation and the integration is then restarted.

Sensitivity Analyzer: The sensitivity analyzer (Figure 5.3) has a rather simple structure. The simulator is called to integrate the artificially created variational problem up to a discontinuity point. The jump handler, then calculates the sensitivity values right after the discontinuity point according to the jump formulas (Equation 3.19). The integration then is restarted with newly calculated values as the initial values.

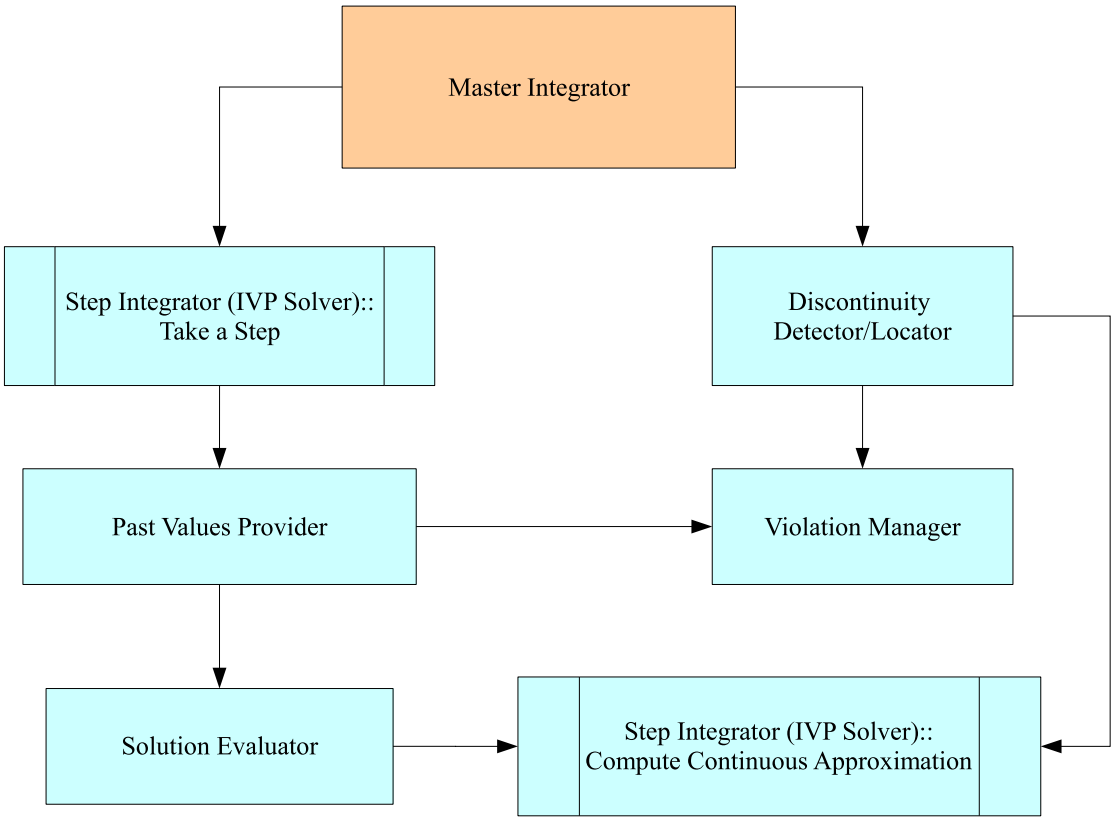

Figure 5.2: Simulator Structure. **A** → **B** indicates that module/component **A** calls or uses module/component **B**. Ordinary rectangles indicate main modules/components of the DDE solver, and double edged rectangles indicate externally provided modules/componets.

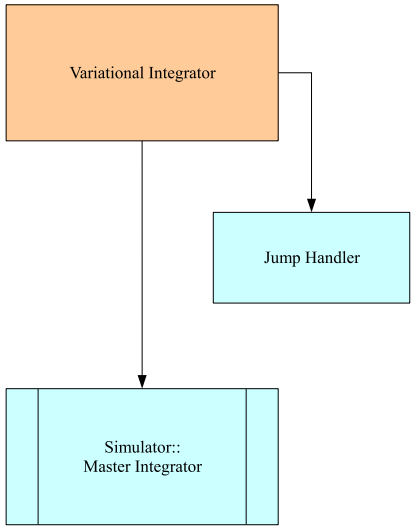

Figure 5.3: Sensitivity Analyzer Structure

Parameter Estimator: The master control is called with the initial parameter fitting problem and its associated data. Then, an artificial problem with added constraints (introduced to ensure smoothness on the iteration) is fed to a least-squares optimizer. When the optimizer module asks for the function or constraints for specific value of parameters, it is directed to a module which in turn provides the required values by calling the simulator or sensitivity analyzer.

5.3 User Interface

The package is implemented using the object-oriented paradigm in C++. The user interface is specially designed to be easy enough for nonspecialist users and controllable enough for advanced users. We have also been careful not to use any global variables and as a result the whole package is thread-safe, if it is to be used in parallel mode.

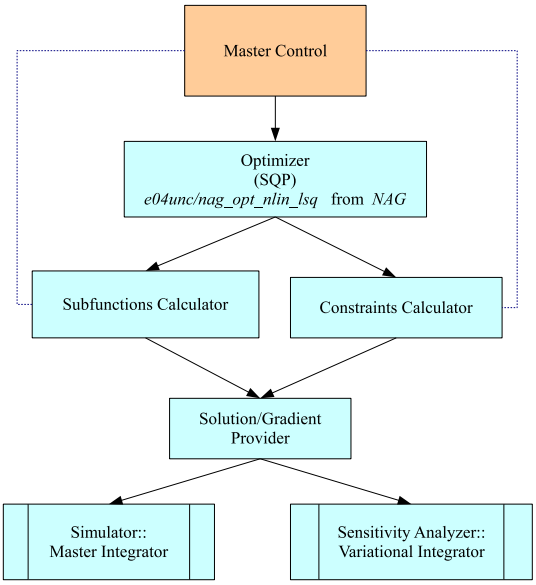

Figure 5.4: Parameter Estimator Structure

Another important aspect in the interface is that there is no assumption for *global modules*. Therefore, it is rather easy to incorporate it into other packages/programs and also easy to do experiments such as comparing different algorithms. As an example one can compare the results and statistics of the simulation with different basic IVP integrators (if available) in a single run. Another example is in the parameter fitting when a user wants to compare the effect of imposing various constraints on the result or performance of the optimizer. In the following we briefly discuss some of the most important aspects of the user interface for the different components.

Simulator: As usual for DDEs, users must provide the functions/constants defining the problem. The only important point is where the simulator is created ("`mySimulator1 = new ...`" in Figure 5.5). As expected, it takes an IVP solver as its underlying IVP integrator. In our sample code, the IVP solver is an object of class

`IVP2DDEImprovedCRK`, derived from an abstract class `ddemIVP2DDE`, which represents a general interface for explicit IVP solvers that can be used by our simulator. Interested users can implement a derived class based on an existing IVP solver and use it in their code. `IVP2DDEImprovedCRK` class, provided by us, is based on a particular IVP method developed by Enright and Yan [22].

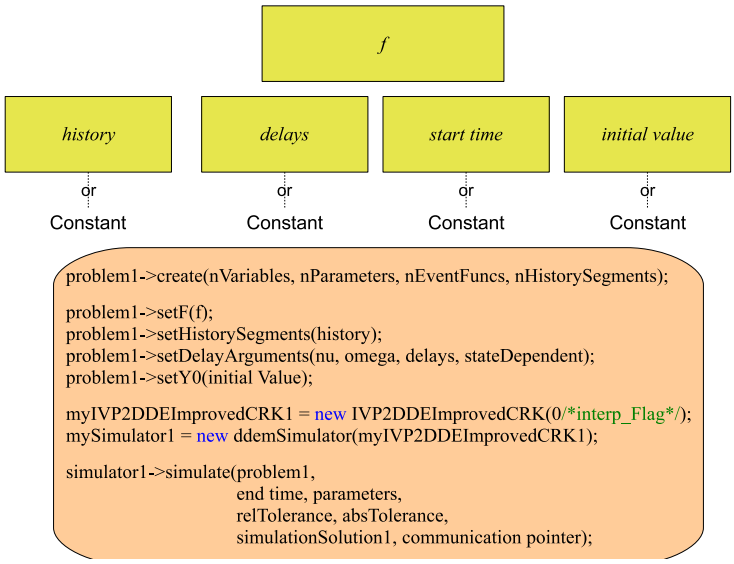

Figure 5.5: Simulator Interface

Sensitivity Analyzer: Most inputs are same as those of the simulator. The `needSensitivity` array determines the selected parameters for which sensitivities are to be computed. The DDE simulator is passed as an argument, which means it can work with different DDE simulators.

Parameter Estimator: In addition to the general input that defines the DDE problem, users need to provide the data and also possible nonlinear and linear constraints. The nonlinear constraints should be provided as a function and the linear constraints can be specified using a constant matrix of coefficients. However, some simple bounds on parameters can be set by calling special built-in subroutines, as these type of constraints appear very frequently in applications.

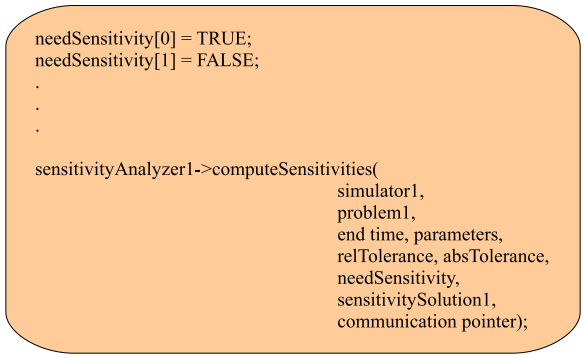

Figure 5.6: Sensitivity Analyzer Interface

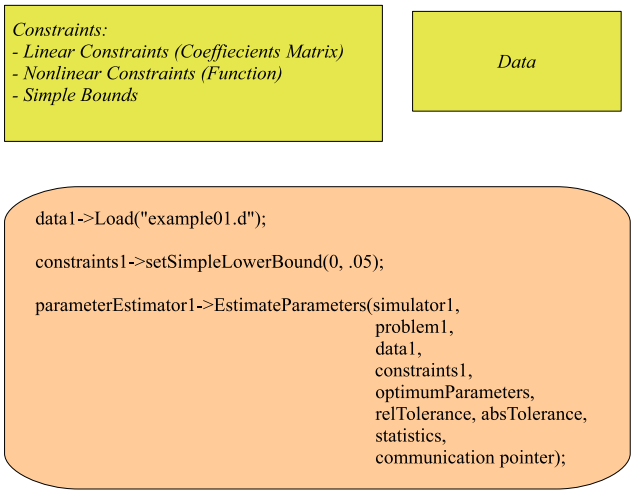

Figure 5.7: Parameter Estimator Interface

The *communication pointer*, present as the last argument of all three components, is used to eliminate the need for global variables. It is a head pointer to all common structures needed by a user inside the supplied subroutines. It will be passed back to the user routines when they are called by any of the three components of our package.

Chapter 6

Numerical Experiments

In this chapter we present some detailed numerical results of our experimental package DDEM.

6.1 Numerical Experiments with the Simulator

The package is currently able to use any explicit step-by-step IVP solver which provides an accurate local interpolant. For our experiments we use the IVP method CRK6X, which is an order 6 explicit continuous Runge-Kutta method with defect control (developed and discussed in [22]), choosing *relaxed defect control* as its defect control strategy. For neutral problems the derivative of the continuous approximation is required and should provide the same order of accuracy. Although this requirement is not met by most of the continuous extensions accompanying current IVP solvers, there is a "bootstrapping" process that can be exploited to develop that kind of continuous approximation for Runge-Kutta methods (see [22] for details). For vanishing delays, the systems of nonlinear equations is solved by fixed point iterations. All switching functions are actively monitored on each step for possible sign changes that indicate the presence of roots with odd multiplicity. If the problem is declared to have state-independent delays, using an optional input, then this information is actively used to accelerate the root-finding process. If the delays are

declared to be increasing functions then g_r (Equation 2.18) is not included in the set of monitored events. We have used a vectorized implementation of the well-known Illinois algorithm from [36], which finds the leftmost root of a set of arbitrary functions in an interval. The roots of odd multiplicity are recognized by the changes of sign.

6.1.1 Test Problems

The following problems are used to test the effectiveness of the proposed methods. "Test Problem 1" and "Test Problem 3" have state-dependent delays and were chosen to test the discontinuity tracking strategy of our method. "Test Problem 2" is an NDDE with many discontinuities and is used to test efficiency of the solver when dealing with persisting discontinuities and also to show its applicability for systems of DDEs. "Test Problem 4" and "Test Problem 5" have vanishing delays, during the integration and at the starting point, respectively. They were chosen to test the iterative scheme for handling vanishing delays. "Test Problem 6" is a 4-dimensional problem with two constant delays. For DDEM, we have considered this problem a vanishing delay problem by setting the lower bound constant in Equation (2.21) to be a value bigger than the smallest constant delay; otherwise forcing the explicitness condition will require at least $(t_F - t_0)/(\textit{the smallest delay}) = (350 - 0)/0.15 \approx 2333$ steps. All these problems are nonstiff.

Test Problem 1 [49]:

$$y' = y(y(t)),$$

for t in $[2, 5.5]$. The history function is

$$y = 0.5, \quad \text{for } t < 2,$$

and

$$y(2) = 1.$$

The C^0 discontinuity of the solution at $\xi_0 = 2$ introduces break points at $\xi_1 = 4$ (C^1) and $\xi_2 = 4 + 2\ln 2 \approx 5.386$ (C^2).

The exact solution to this problem is

$$y(t) = \begin{cases} t/2, & \text{for } \xi_0 \le t \le \xi_1, \\ 2\exp(t/2 - 2), & \text{for } \xi_1 \le t \le \xi_2, \\ 4 - 2\ln(1 + \xi_2 - t) & \text{for } \xi_2 \le t \le 5.5. \end{cases}$$

Test Problem 2. A neutral delay logistic Gause-type predator-prey system [40]:

$$\begin{aligned} y_1'(t) &= y_1(t)(1 - y_1(t-\tau) - \rho y_1'(t-\tau)) - \frac{y_2(t)y_1(t)^2}{y_1(t)^2 + 1}, \\ y_2'(t) &= y_2(t)\left(\frac{y_1(t)^2}{y_1(t)^2 + 1} - \alpha\right), \end{aligned}$$

where $\alpha = 1/10$, $\rho = 29/10$ and $\tau = 21/50$, for t in $[0, 30]$. The history functions are

$$\begin{aligned} \phi_1(t) &= \frac{33}{100} - \frac{1}{10}t, \\ \phi_2(t) &= \frac{111}{50} + \frac{1}{10}t, \end{aligned}$$

for $t \le 0$. The solution is C^1 discontinuous at the starting point which propagates as C^1 and C^2 discontinuities to $y_1(t)$ and $y_2(t)$, respectively, at $t = n\tau$ for $n \ge 1$. The exact solution of this problem is unknown.

Test Problem 3 [47]:

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

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

$$\phi(t) = 1, \quad \text{for } t \le 1.$$

The exact solution to this problem is

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

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

Derivative jump discontinuities occur at $t = 1$ (C^1), $t = e$ (C^2), $t = e^2$ (C^3) and $t = e_3$ (C^4).

Test Problem 4 [50]:

$$y'(t) = y(t - t^{-10}),$$

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

$$\phi(t) = t, \quad \text{for } \ t \le 1.$$

The exact solution of this problem is unknown.

This DDE has a vanishing (but non-singular) lag ($\lim_{t\to+\infty} t^{-10} = 0$; however, depending on the precision used, the vanishing behavior will first be recognized at some finite time $t^\star$ and persists for all $t > t^\star$).

Test Problem 5 [58]:

$$y'(t) = y(y(t)) + (3 + \mu)t^{(2+\mu)} - t^{(3+\mu)^2},$$

for t in [0, 1]. The initial value is

$$y(0) = 0.$$

The exact solution to this problem is

$$y(t) = t^{(3+\mu)}, \quad \text{for } \ 0 \le t \le 1.$$

This is an *initial value* DDE with no discontinuities.

We use $\mu = 0$ in our experiments. The exact solution is a low degree polynomial and any IVP method should have no trouble with this problem.

Test Problem 6. The SEIR epidemic model of Genik & van den Driessche [28]:

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

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 functions are

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

for $t \leq 0$.

The exact solution of this problem is unknown.

6.1.2 Results

Here we present the numerical results for the chosen test problems. We include results for a new version [31] of RADAR5 (Guglielmi and Hairer [30]), and DDE_SOLVER (Thompson and Shampine [59]). We have set all absolute tolerances and the relative tolerance to TOL with TOL=$10^{-3}, 10^{-6}$,10^{-9} for all solvers. The analytic solution or a very accurate approximation of it, obtained with a very small tolerance (TOL = 10^{-11}),

was used for computing the reported endpoint accuracy. The statistics we report in the tables are :

STEPS: The number of successful steps.

REJECTS: The number of rejected steps.

FCN: The total number of derivative evaluations.

ABS_ERR: The global absolute error at the end point of integration (maximum over all components for multidimensional problems).

REL_ERR: The global component-wise relative error at the end point of integration (maximum over all components for multidimensional problems).

Table 6.1: Summary Statistics for Problems 1 to 6 (TOL = 10^{-3}).

PROBLEM	SOLVER	STEPS	REJECTS	FCN	ABS_ERR	REL_ERR
	DDE_SOLVER	11	0	108	$7.8 \cdot 10^{-9}$	$1.8 \cdot 10^{-9}$
1	RADAR5	7	1	80	$6.7 \cdot 10^{-5}$	$1.5 \cdot 10^{-5}$
	DDEM	5	0	58	$9.5 \cdot 10^{-6}$	$2.2 \cdot 10^{-6}$
	DDE_SOLVER	255	226	5067	$4.4 \cdot 10^{-5}$	$1.3 \cdot 10^{-4}$
2	RADAR5	150	58	2002	$6.1 \cdot 10^{-4}$	$1.8 \cdot 10^{-3}$
	DDEM	73	0	875	$7.2 \cdot 10^{-5}$	$8.3 \cdot 10^{-5}$
	DDE_SOLVER	19	4	225	$1.2 \cdot 10^{-6}$	$3.1 \cdot 10^{-8}$
3	RADAR5	12	2	121	$8.8 \cdot 10^{-4}$	$2.1 \cdot 10^{-5}$
	DDEM	7	0	80	$2.1 \cdot 10^{-3}$	$5.4 \cdot 10^{-5}$
	DDE_SOLVER	15	2	405	$8.6 \cdot 10^{-1}$	$1.1 \cdot 10^{-4}$
4	RADAR5	25	1	215	5.4	$7.3 \cdot 10^{-4}$
	DDEM	35	2	436	3.0	$4.0 \cdot 10^{-4}$
	DDE_SOLVER	13	0	153	$1.1 \cdot 10^{-9}$	$1.1 \cdot 10^{-9}$
5	RADAR5	4	0	29	0.0	0.0
	DDEM	10	0	119	$3.2 \cdot 10^{-5}$	$3.2 \cdot 10^{-5}$
	DDE_SOLVER	140	7	3051	$3.4 \cdot 10^{-5}$	$5.9 \cdot 10^{-4}$
6	RADAR5	54	1	643	$1.3 \cdot 10^{-4}$	$1.2 \cdot 10^{-3}$
	DDEM	415	0	4813	$1.8 \cdot 10^{-8}$	$3.3 \cdot 10^{-7}$

Table 6.2: Summary Statistics for Problems 1 to 6 (TOL = 10^{-6}).

PROBLEM	SOLVER	STEPS	REJECTS	FCN	ABS_ERR	REL_ERR
	DDE_SOLVER	15	6	198	$1.0 \cdot 10^{-11}$	$2.5 \cdot 10^{-12}$
1	RADAR5	13	4	120	$3.1 \cdot 10^{-8}$	$7.4 \cdot 10^{-9}$
	DDEM	7	0	80	$1.4 \cdot 10^{-7}$	$3.4 \cdot 10^{-8}$
	DDE_SOLVER	907	947	16884	$1.4 \cdot 10^{-7}$	$4.4 \cdot 10^{-7}$
2	RADAR5	369	130	4592	$8.6 \cdot 10^{-8}$	$8.8 \cdot 10^{-8}$
	DDEM	135	23	1810	$6.5 \cdot 10^{-7}$	$7.4 \cdot 10^{-7}$
	DDE_SOLVER	31	12	405	$9.9 \cdot 10^{-8}$	$2.4 \cdot 10^{-9}$
3	RADAR5	28	1	225	$1.0 \cdot 10^{-5}$	$2.7 \cdot 10^{-7}$
	DDEM	18	2	223	$9.0 \cdot 10^{-6}$	$2.2 \cdot 10^{-7}$
	DDE_SOLVER	118	10	2673	$9.4 \cdot 10^{-3}$	$1.2 \cdot 10^{-6}$
4	RADAR5	73	1	608	$7.8 \cdot 10^{-3}$	$1.0 \cdot 10^{-6}$
	DDEM	64	4	792	$7.7 \cdot 10^{-4}$	$1.0 \cdot 10^{-7}$
	DDE_SOLVER	13	0	153	$1.1 \cdot 10^{-9}$	$1.1 \cdot 10^{-9}$
5	RADAR5	4	0	29	0.0	0.0
	DDEM	12	3	172	$2.0 \cdot 10^{-7}$	$2.0 \cdot 10^{-7}$
	DDE_SOLVER	211	12	4923	$6.8 \cdot 10^{-8}$	$5.8 \cdot 10^{-7}$
6	RADAR5	119	1	1413	$6.7 \cdot 10^{-7}$	$5.2 \cdot 10^{-6}$
	DDEM	417	0	4836	$1.6 \cdot 10^{-8}$	$2.9 \cdot 10^{-7}$

Table 6.3: Summary Statistics for Problems 1 to 6 (TOL = 10^{-9}).

PROBLEM	SOLVER	STEPS	REJECTS	FCN	ABS_ERR	REL_ERR
	DDE_SOLVER	21	11	297	$6.6 \cdot 10^{-12}$	$1.5 \cdot 10^{-12}$
1	RADAR5	24	5	207	$5.6 \cdot 10^{-9}$	$1.3 \cdot 10^{-9}$
	DDEM	12	3	168	$2.1 \cdot 10^{-9}$	$4.9 \cdot 10^{-10}$
	DDE_SOLVER	1718	1577	29655	$9.5 \cdot 10^{-11}$	$4.4 \cdot 10^{-11}$
2	RADAR5	918	123	10063	$3.8 \cdot 10^{-10}$	$1.1 \cdot 10^{-9}$
	DDEM	376	150	5858	$6.3 \cdot 10^{-10}$	$2.8 \cdot 10^{-10}$
	DDE_SOLVER	68	18	792	$1.4 \cdot 10^{-10}$	$3.6 \cdot 10^{-12}$
3	RADAR5	70	1	525	$1.0 \cdot 10^{-7}$	$2.6 \cdot 10^{-9}$
	DDEM	47	3	553	$1.5 \cdot 10^{-8}$	$3.7 \cdot 10^{-10}$
	DDE_SOLVER	789	18	15453	$3.5 \cdot 10^{-5}$	$4.5 \cdot 10^{-9}$
4	RADAR5	201	2	1672	$1.1 \cdot 10^{-5}$	$1.5 \cdot 10^{-9}$
	DDEM	144	6	1735	$4.9 \cdot 10^{-6}$	$6.7 \cdot 10^{-10}$
	DDE_SOLVER	16	6	243	$3.2 \cdot 10^{-11}$	$3.2 \cdot 10^{-11}$
5	RADAR5	4	0	29	0.0	0.0
	DDEM	23	6	325	$3.3 \cdot 10^{-10}$	$3.3 \cdot 10^{-10}$
	DDE_SOLVER	447	14	9360	$3.7 \cdot 10^{-11}$	$2.5 \cdot 10^{-11}$
6	RADAR5	281	10	3146	$3.5 \cdot 10^{-9}$	$6.4 \cdot 10^{-8}$
	DDEM	480	6	5627	$2.1 \cdot 10^{-9}$	$3.8 \cdot 10^{-8}$

6.1.3 Discussion and Observation

The numerical results clearly show that for these particular examples, the derived DDE solver is competitive with a state of the art special purpose DDE solver. These results are chosen from a more extensive investigation that we have performed on different problems using standard DDE test sets [51], and confirm our claim that the generality we have introduced does not cause a noticeable inefficiency compared to specially designed DDE solvers. We are currently working on the details of the interface for implicit solvers and hope to find a design that preserves this property.

6.2 Numerical Experiments with the Sensitivity Analyzer

6.2.1 Test Cases

The following cases are used to show the effectiveness of our method. "Test Case 1" and "Test Case 3" are interesting situations where the sensitivity of the solution is observed with respect to parameters controlling the status of the starting point. This situation cannot be handled using traditional approaches. "Test Case 2" is a two-dimensional model with several parameters, including the parameters defining the components of the history function. "Test Case 4" is chosen to study the sensitivities for a system with chaotic behavior.

Test Case 1 Sensitivity of the solution with respect to starting jump for "Test Problem 1",

$$\mathbf{p} = [y(2)].$$

Test Case 2 Sensitivity of the solution with respect to all parameters and histories for

"Test Problem 2",

$$\mathbf{p} = [\tau,\ \rho,\ \alpha,\ \mathrm{a},\ \mathrm{b},\ \mathrm{c},\ \mathrm{d}].$$

where

$$\phi_1(t) = \mathrm{a} + \mathrm{b}\, t,$$

$$\phi_2(t) = \mathrm{c} + \mathrm{d}\, t.$$

Test Case 3 Sensitivity of the solution with respect to the starting time for "Test Problem 3",

$$\mathbf{p} = [t_0].$$

Test Case 4 Sensitivity of the solution with respect to the delay, exponent and history for a scalar equation that exhibits chaotic behavior. It is an example of the well known Mackey-Glass delay differential equations which they proposed as a model for the production of white blood cells [43]. The problem has a constant delay and a constant history, and is defined by

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

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

$$\phi(t) = 0.5, \quad \text{for } t \leq 0.$$

The exact solution of this problem is unknown. See Figure (6.8) for an accurate approximate solution.

The parameters are,

$$\mathbf{p} = [\tau,\ n,\ \mathrm{A}],$$

where

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

and

$$\phi(t) = \mathrm{A}, \quad \text{for } t \leq 0.$$

6.2.2 Results and Discussion

Figures 6.1–6.9 and Table 6.4 present the numerical results for the chosen cases. Some interesting observations are made for each case in the respective captions. The results are also compared with the finite difference approach by showing the absolute error in the sensitivity for various parameter perturbations ($\Delta\mathbf{p}$). For this, we have used the results from our sensitivity analyzer code with a very tight tolerance (10^{-11}) as the exact values. The same tolerance (10^{-11}) was used for the simulations required in the finite difference approach. We also report the performance and accuracy of our code for "Test Case 1" for different tolerances.

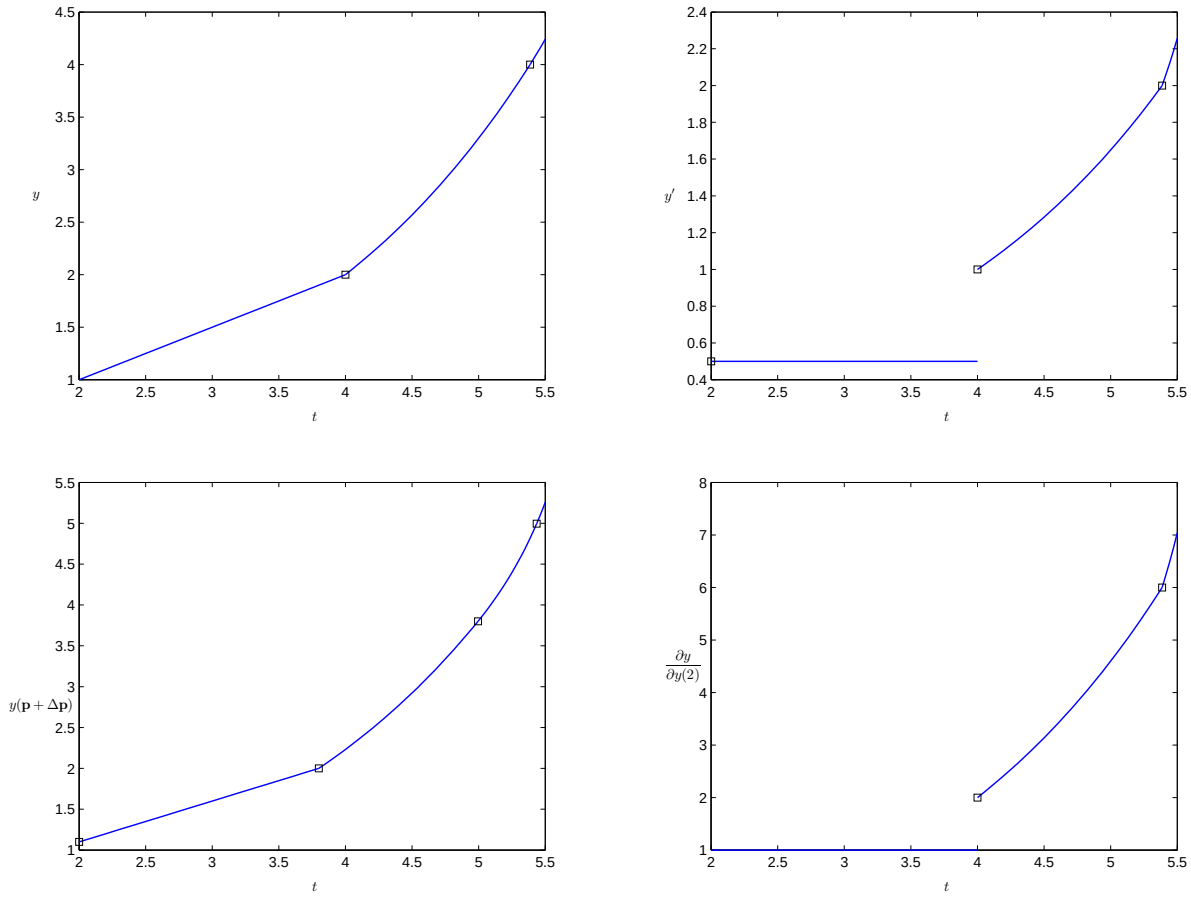

Figure 6.1: Plots of the numerical solution and the sensitivity for Test Case 1. Discontinuities of the solution at $t = 2, 4$ produces jumps in the sensitivity at those points. (Note that $y(\mathbf{p} + \Delta\mathbf{p})$ with $\Delta\mathbf{p} = 0.01$, has one more discontinuity point than y in the integration interval.)

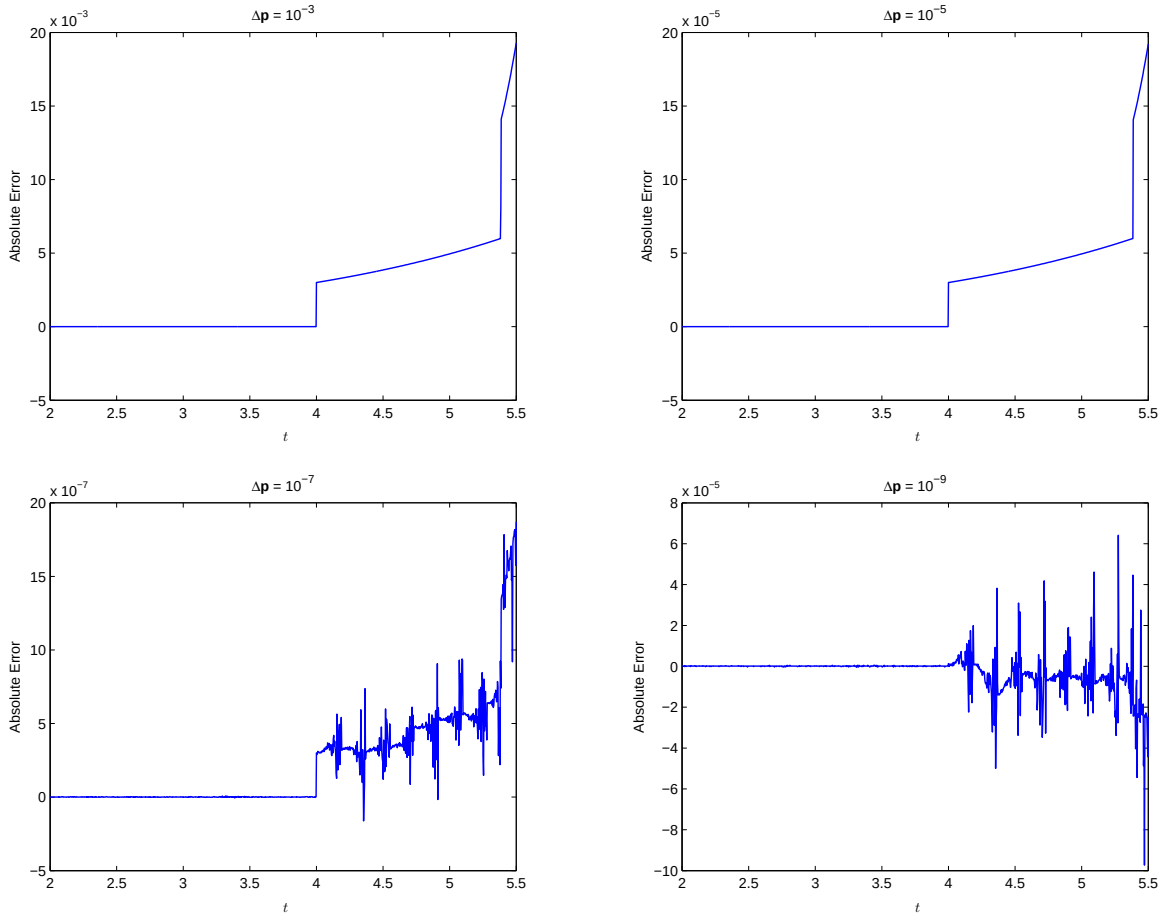

Figure 6.2: Plots of absolute errors of the sensitivity $\frac{\partial y}{\partial y(2)}$ computed using finite differences for Test Case 1. The limited accuracy of finite differences is clearly visible for $\mathbf{\Delta p} = 10^{-9}$.

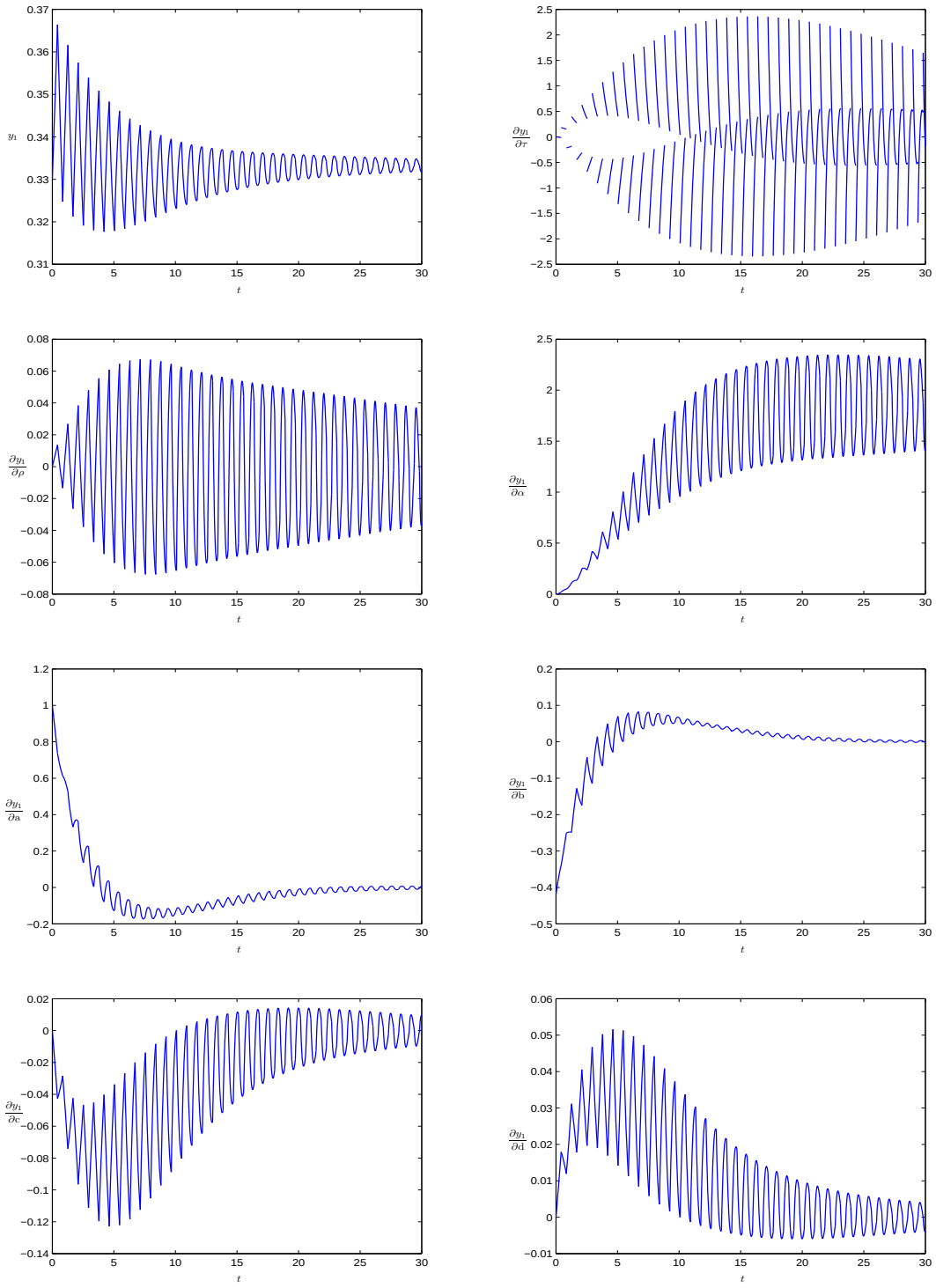

Figure 6.3: Plots of the numerical solution and the sensitivities for Test Case 2 (y_1). Discontinuities of the solution produce jumps in the sensitivity. Sensitivity coefficients for history related parameters, clearly show the transient effect of the history and the approximate time of this fading behavior.

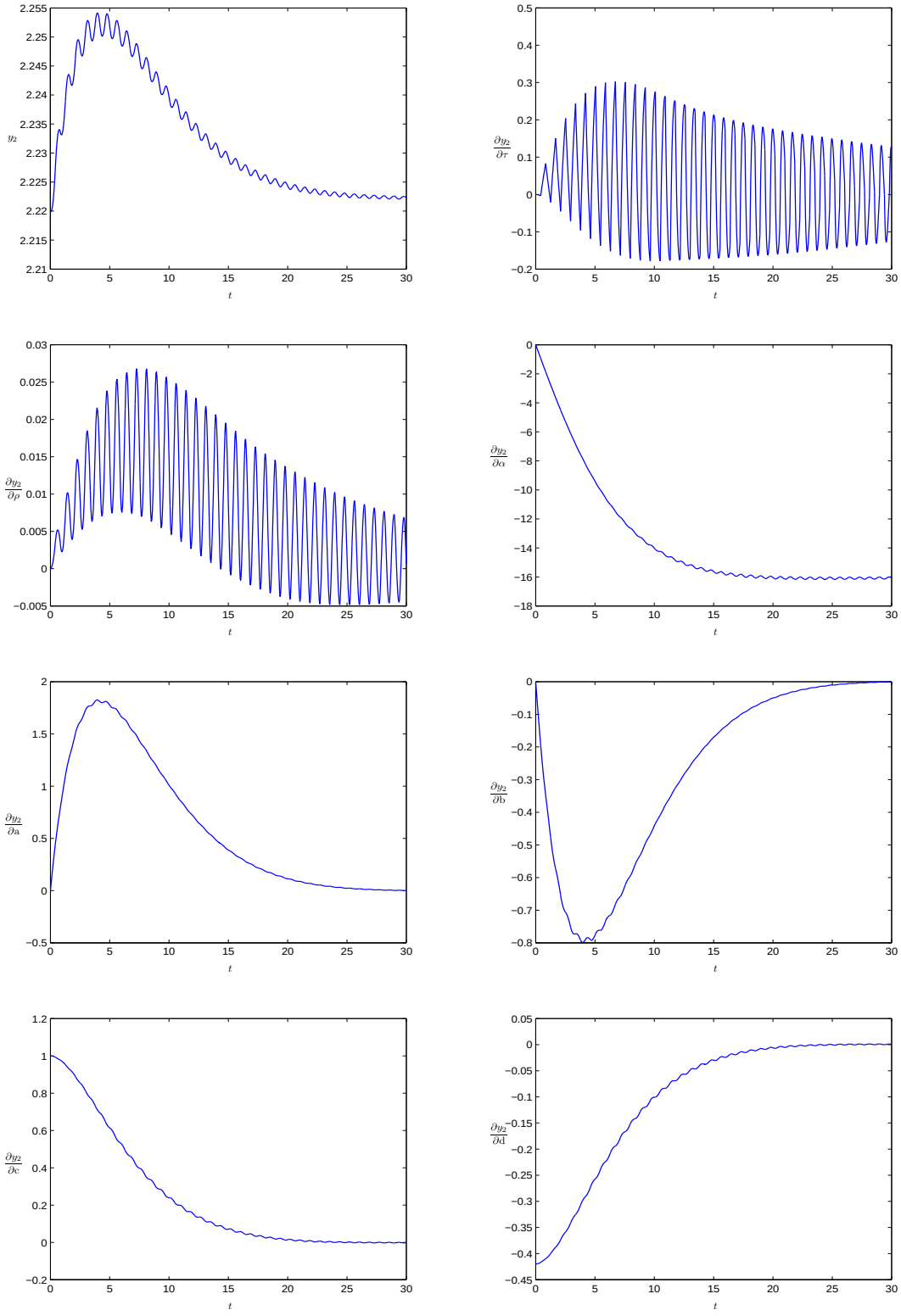

Figure 6.4: Plots of the numerical solution and the sensitivities for Test Case 2 (y_2). The sensitivity coefficient for the delay τ is smoother (persistent C^1 discontinuities), as well as the function itself (persistent C^2 discontinuities), compared to those of (y_1).

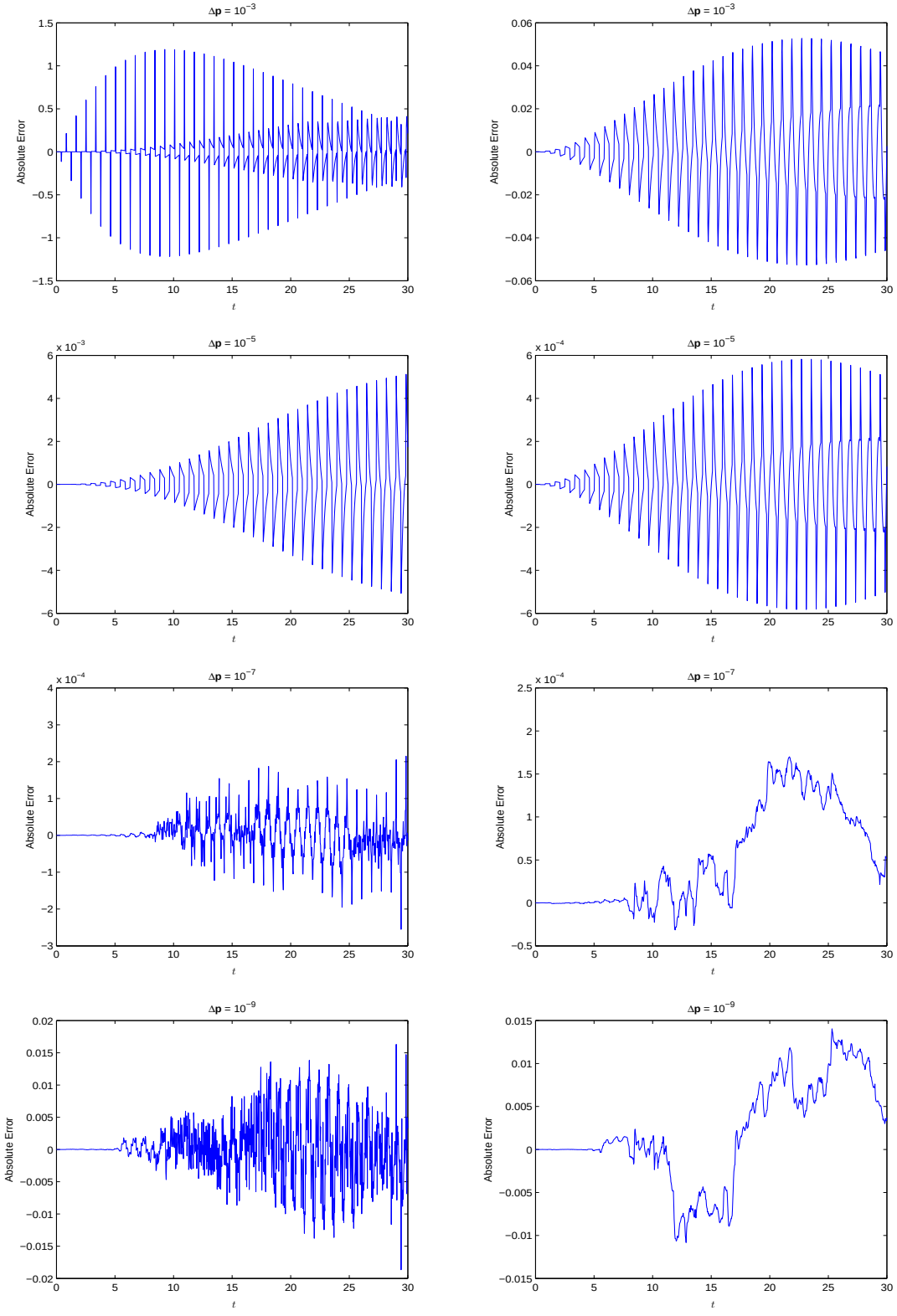

Figure 6.5: Plots of absolute errors of sensitivities $\frac{\partial y_1}{\partial \tau}$ (left column) and $\frac{\partial y_2}{\partial \tau}$ (right column) computed using finite differences for Test Case 2. The poor accuracy for $\Delta \mathbf{p} = 10^{-3}$ and the limited accuracy for $\Delta \mathbf{p} = 10^{-9}$ are clearly visible.

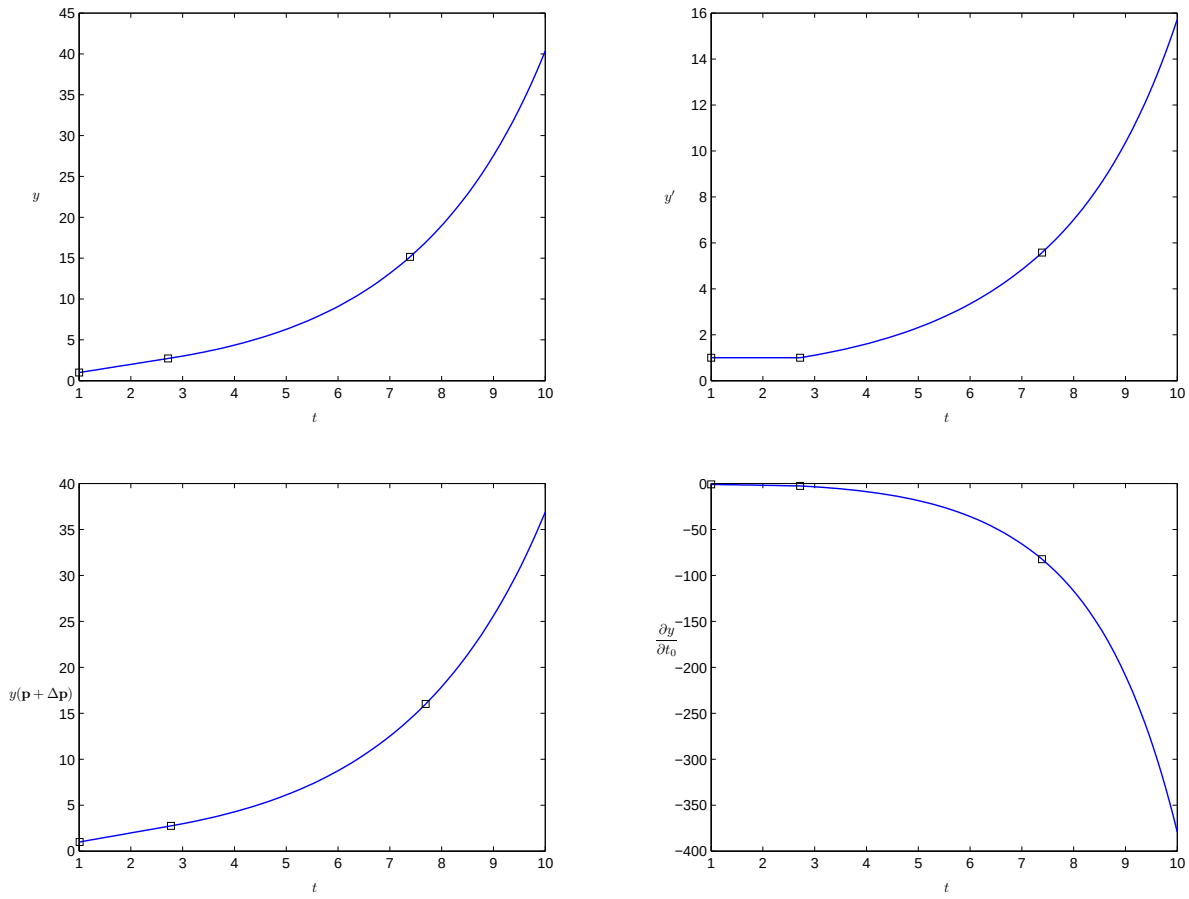

Figure 6.6: Plots of the numerical solution and the sensitivities for Test Case 3. The discontinuity of the solution at the starting point produces a jump in the sensitivity at that point (from 0 for $t < 1$, to -1 for $t = 1$). $y(\mathbf{p} + \Delta\mathbf{p})$ ($\Delta\mathbf{p} = 0.01$) shows a big reduction for a small change in the parameter as the sensitivity function predicts ($\frac{\partial y}{\partial t_0} \lll 0$).

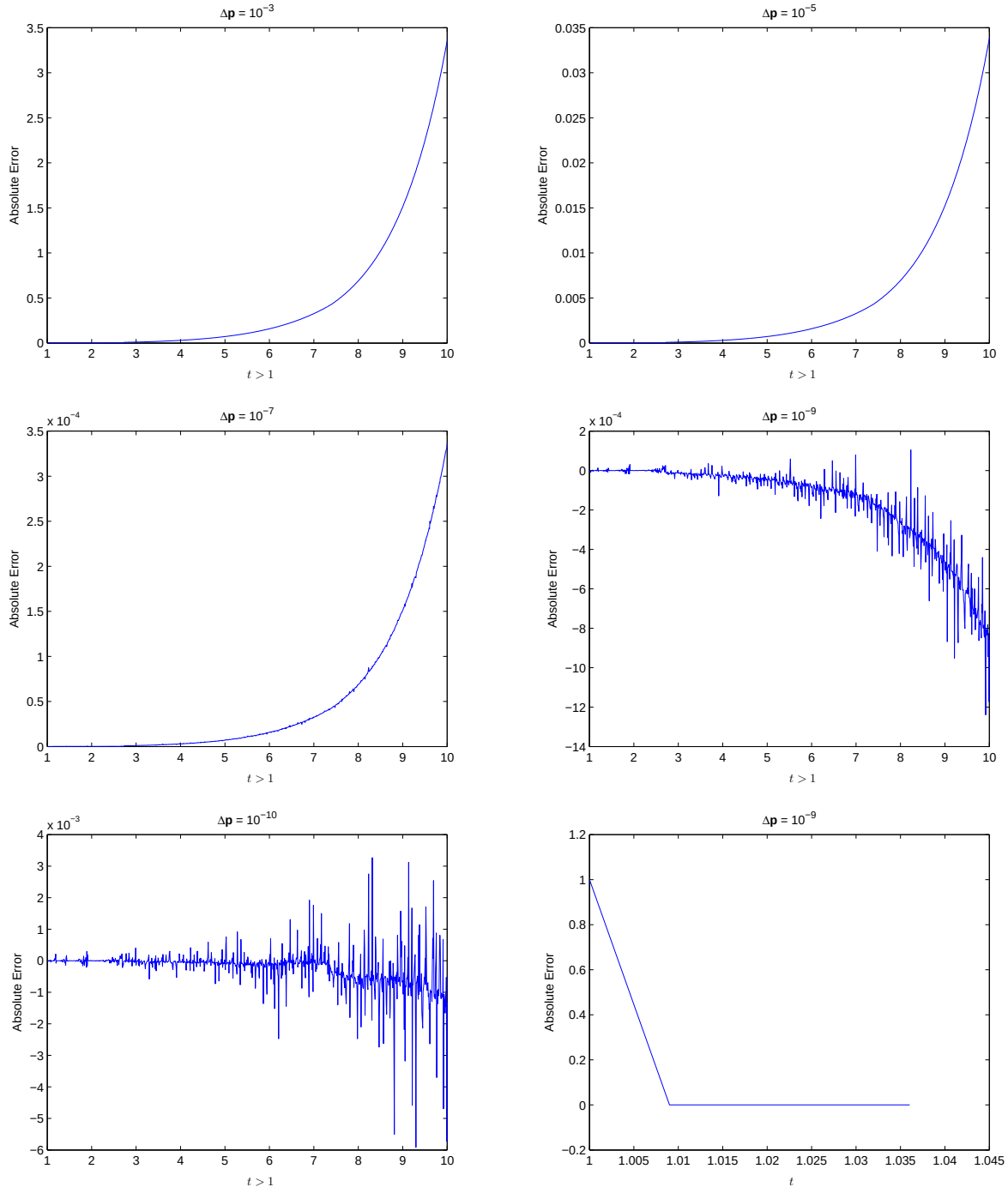

Figure 6.7: Plots of absolute errors of the sensitivity $\frac{\partial y}{\partial t_0}$ computed using finite differences for Test Case 3. The limited accuracy of finite differences is clearly visible for $\Delta\mathbf{p} = 10^{-10}$. Since the value of sensitivity computed by finite difference is very inaccurate at $t = t_0 = 1$ and the error is large, it is shown in a separate graph and excluded from the others.

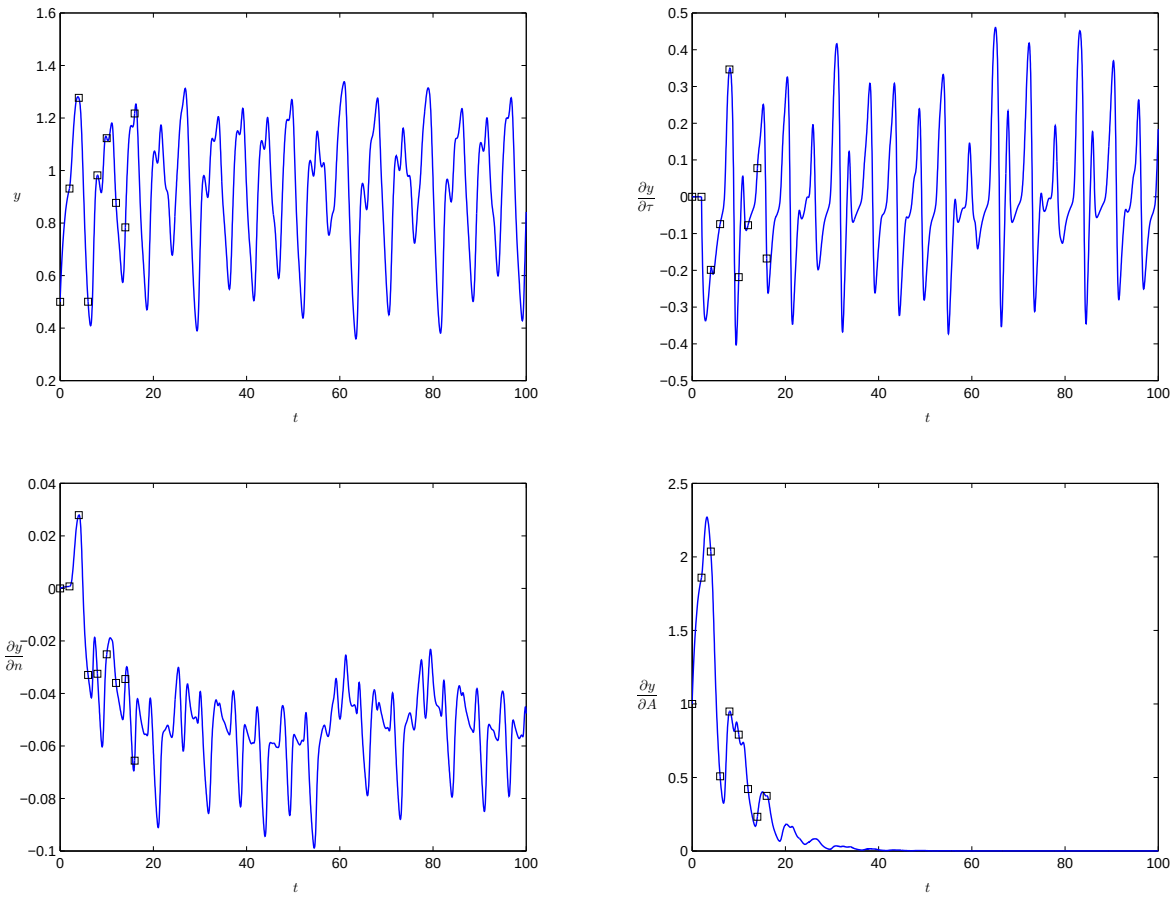

Figure 6.8: Plot of the numerical solution and the sensitivities for Test Case 4. The chaotic sensitivities $\frac{\partial y}{\partial \tau}$, $\frac{\partial y}{\partial n}$ and the non-chaotic sensitivity $\frac{\partial y}{\partial A}$ indicate that the chaos in y is a combined result of having a delay (τ) and an exponent (n) and is insensitive to the history.

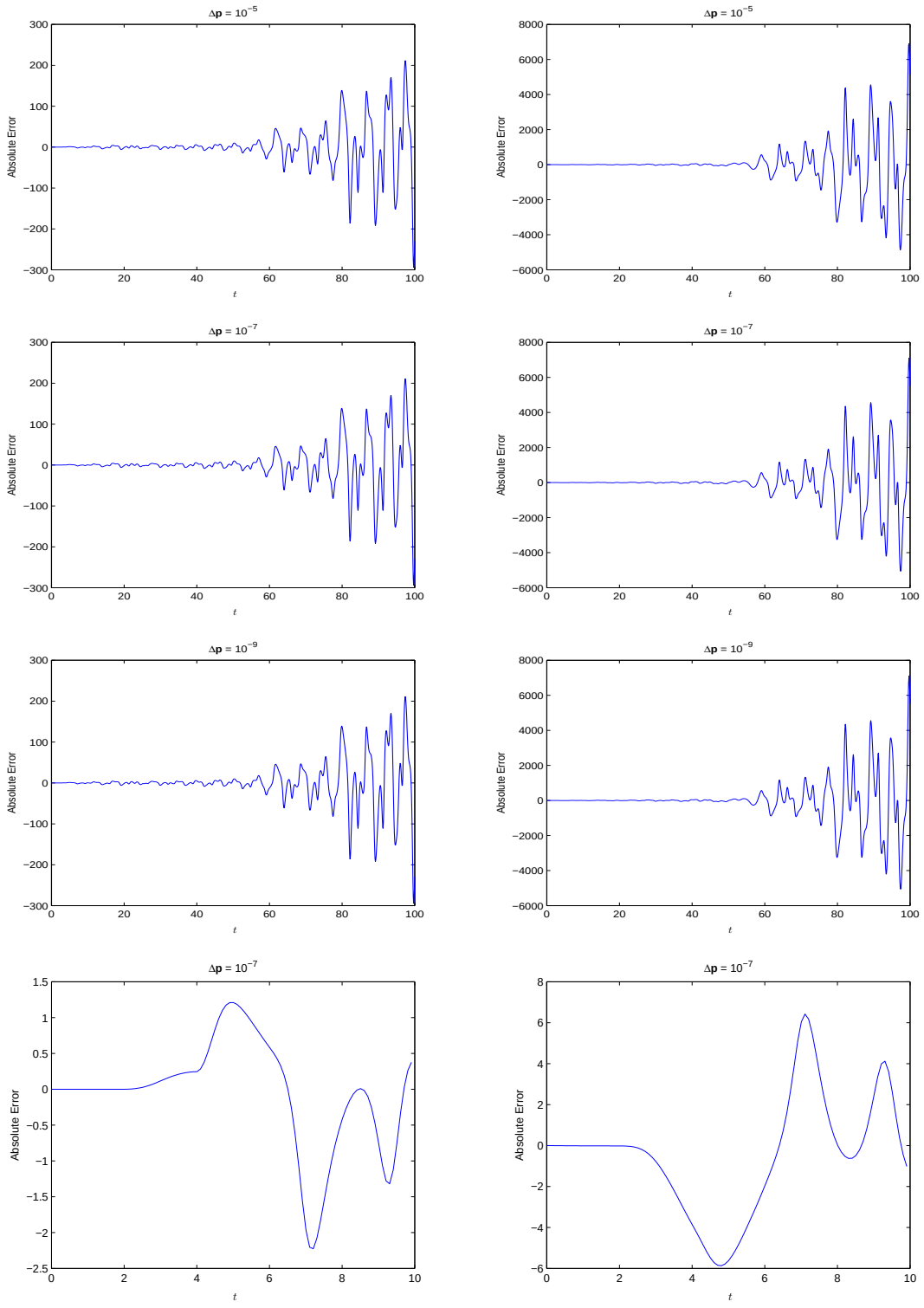

Figure 6.9: Plots of absolute errors of sensitivities $\frac{\partial y}{\partial \tau}$ (left column) and $\frac{\partial y}{\partial A}$ (right column) computed using finite differences for Test Case 4. The errors are extremely large; even for values near the starting point, similar large errors (more than 100%) can be seen if we look closely (bottom plots).

TOL	STEPS	REJECTS	FCN	ERROR
10^{-3}	6	0	69	$1.0 \cdot 10^{-6}$
10^{-5}	7	0	80	$3.1 \cdot 10^{-5}$
10^{-7}	10	1	124	$6.9 \cdot 10^{-8}$
10^{-9}	15	2	190	$5.2 \cdot 10^{-9}$
10^{-11}	24	3	300	-

Table 6.4: Summary Statistics of Sensitivity Analyzer for Test Case 1 with different tolerances (absolute tolerance = relative tolerance = TOL). The output with TOL = 10^{-11} is used as the exact value for error calculations. The reported error is the maximum absolute error in the sensitivities over the integration interval.

6.3 Numerical Experiments with the Parameter Estimator

6.3.1 Test Cases

The following cases are used to show the efficiency and reliability of our package. "Test Case 1" is a one-dimensional NDDE and is chosen as the simplest interesting situation. "Test Case 2" is a state-dependent RDDE where the parameter indirectly controls the location of discontinuities. "Test Case 3" is a three-dimensional model with two parameters, both causing discontinuities. "Test Case 4" is a two-dimensional NDDE with several parameters, where different components have different continuity properties. We start with up to 10% random perturbation in the original targeted parameter values. The simulator output with original parameter values is used as input Data (Y). For γ's we choose 10 equally spaced points, one of which is the first discontinuity point. (In the

following, $p^\star$ indicates the optimum parameter value.)

Test Case 1 Estimating the delay for a triangle wave defined by the following NDDE,

$$y'(t) = -y'(t-\tau),$$

where $\tau^\star = 0.1$, for t in $[0, 0.9]$. The history function is

$$\phi(t) = -t + \mathrm{c},$$

where $c = 8$, for $t \leq 0$.

The objective function is C^1 discontinuous for values of τ where a data point coincides with an integer multiple of τ.

Test Case 2 Estimating $y(2)$ for "Test Problem 1".

The locations of discontinuities depend on the parameter, but they are not easy to determine analytically.

Test Case 3 Estimating delays for a Kermack-McKendrick model of an infectious disease with periodic outbreak ([21], [32], [56], [57]). The problem is defined by

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

where $\tau^\star = 1$ and $\rho^\star = 10$, for t in $[0, 55]$. The history function is

$$\begin{aligned} y_1 &= 5, \\ y_2 &= 0.1, \\ y_3 &= 1, \end{aligned}$$

for $t \leq 0$.

The exact solution of this problem is unknown.

Both delays could cause C^2 discontinuities in the objective function whenever a data point is at $t = \tau$ or $t = \rho$.

Test Case 4 Estimating structure-related parameters τ, ρ, α for "Test Problem 2". A C^1 discontinuity is caused solely by the delay.

6.3.2 Results

We present the numerical results for the chosen cases for different variants of the parameter estimator as described below:

Very Simple: No additional constraints are added. The required Jacobians are calculated using divided differences.

SensJac: No additional constraints are added. The required Jacobians are calculated using the sensitivity analyzer.

SensJac AddedCons: Additional constraints are added. The required Jacobians are calculated using the sensitivity analyzer. Extrapolation is used for infeasible parameter values.

The statistics we report are averages for 10 runs and are as follows :

FCN: The total number of derivative (f) evaluations, including those required for approximating the Jacobians.

Iterations: The number of iterations during optimization.

Time: The total cpu time, in seconds, used by optimization.

OBJ: The value of the objective function W at the optimum point.

6.3.3 Discussions and Conclusions

The numerical results reported in Table 6.5 show a dramatic improvement in efficiency where sensitivities are used for computing required Jacobians during the optimization

Test Case	Estimator Choice	FCN	Iterations	Time	OBJ
1	Very Simple	15708	139.1	6.250	$2.9 \cdot 10^{-21}$
	SensJac	172	1.6	0.162	$1.1 \cdot 10^{-24}$
	SensJac AddedCons	108	1.0	0.132	$3.3 \cdot 10^{-25}$
2	Very Simple	3325	40.5	0.775	$1.5 \cdot 10^{-322}$
	SensJac	142	1.8	0.060	$6.2 \cdot 10^{-16}$
	SensJac AddedCons	79	1.0	0.033	$6.2 \cdot 10^{-16}$
3	Very Simple	783092	393.2	54.875	$7.4 \cdot 10^{-13}$
	SensJac	37344	13.8	2.262	$1.3 \cdot 10^{-9}$
	SensJac AddedCons	5293	2.1	0.314	$1.3 \cdot 10^{-9}$
4	Very Simple	1003787	696.3	93.520	$8.6 \cdot 10^{-6}$
	SensJac	60604	16.8	5.399	$8.6 \cdot 10^{-6}$
	SensJac AddedCons	11708	3.2	1.056	$8.6 \cdot 10^{-6}$

Table 6.5: Summary Statistics for Test Cases 1 to 4.

process. The parameter estimator which uses added constraints performs the best, even though the underlying optimization is of a larger dimension.

Chapter 7

Conclusions

7.1 Summary

We introduced a unified structure for a DDE solver and discussed the details for both explicit and implicit IVP formulas. We compared numerical results for an implementation based on a recently developed explicit IVP solver. By comparing these results with two state of the art special purpose DDE solvers, we showed the effectiveness of our design.

We developed the details of an algorithm for the accurate computation of sensitivities for general DDEs. We derived the governing equations and also an equation for calculating the size of jumps at discontinuity points. We compared our method with the traditional finite difference approach and showed that not only does our method produces more accurate results, but also it can handle a wider range of problems.

We proposed an algorithm for reliable parameter identification of DDEs, based on adding extra constraints to the non-smooth optimization problem and converting it to a smooth problem. We used our sensitivity analysis module for computing the required Jacobians and compared our method with two other possible methods. Numerical results clearly supported the superiority of our approach over the others.

Finally, we developed and justified an implementation for the three basic components

(simulation, sensitivity analysis, parameter estimation) of a DDE modeling package and showed that our proposed integrated design could be efficiently programmed and the resulting C++ code had most of the desirable properties demanded for a modern software package.

7.2 Future Work

Designing and implementing the simulator for an implicit IVP solver (such as RADAU5 [33]) and investigating and comparing the results with other DDE solvers is an area for future investigations.

Computing accurate second order sensitivities and using automatic differentiation to obtain the required partial derivatives for user provided functions are other areas that require further research.

For the parameter estimation problem we implemented our techniques for adaptation of a simple parameter fitting method (the initial value approach). Adaptations of more advanced parameter fitting methods with a similar strategy for preventing non-smoothness is also a subject for future study.

Bibliography

[1] C.T.H. Baker, G.A. Bocharov, and F.A. Rihan. A Report on the Use of Delay Differential Equations in Numerical Modelling in the Biosciences. Technical Report 343, Department of Mathematics, University of Manchester, Manchester, England, July 1999.

[2] C.T.H. Baker and E.I. Parmuzin. Identification of the initial function for delay differential equations: Part I: The continuous problem and an integral equation analysis. Technical Report 431, Department of Mathematics, University of Manchester, Manchester, England, Jan 2004.

[3] C.T.H. Baker and E.I. Parmuzin. Identification of the initial function for delay differential equations: Part II: A discrete analogue of the data assimilation problem and computational results. Technical Report 443, Department of Mathematics, University of Manchester, Manchester, England, March 2004.

[4] C.T.H. Baker and E.I. Parmuzin. Identification of the initial function for delay differential equations: Part III: Nonlinear DDEs and computational results. Technical Report 444, Department of Mathematics, University of Manchester, Manchester, England, March 2004.

[5] C.T.H. Baker and C.A.H. Paul. Pitfalls in Parameter Estimation for Delay Differential Equations. *SIAM J. Sci. Comput.*, 18(1):305–314, 1997.

[6] C.T.H. Baker and F.A. Rihan. Sensitivity Analysis of Parameters in Modelling with Delay-Differential Equations. Technical Report 349, Department of Mathematics, University of Manchester, Manchester, England, September 1999.

[7] R. Barrio. Sensitivity Analysis of ODEs/DAEs Using The Taylor Series Method. *SIAM J. Sci. Comput.*, 27(6):1929–1947, 2006.

[8] P.I. Barton and C.C. Pantelides. Modeling of combined discrete/continuous processes. *AIChE J.*, 40(6):966–979, 1994.

[9] 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.

[10] H.G. Bock. Recent Advances in Parameter Identification Techniques for ODE. In P. Deuflhard and E. Hairer, editors, *Numerical Treatment of Inverse Problems in Differential and Integral Equations*, pages 95–121. Birkhäuser, Boston, 1983.

[11] J.V. Burke, A.S. Lewis, and M.L. Overton. A Robust Gradient Sampling Algorithm for Nonsmooth, Nonconvex Optimization. *SIAM J. Optim.*, 15(3):751–779, 2005.

[12] J.C. Butcher. The adaptation of STRIDE to delay differential equations. *Appl. Numer. Math.*, 9:415–425, 1992.

[13] J.C. Butcher. *Numerical Methods for Ordinary Differential Equations*. J. Wiley, Chichester, second edition, 2008.

[14] J.C. Butcher, K. Burrage, and F.H. Chipman. STRIDE: stable Runge-Kutta integrator for differential equations. Technical Report Computational Mathematics Report No. 20, University of Auckland, Auckland, New Zealand, 1979.

[15] G.D. Byrne and A.C. Hindmarsh. PVODE, An ODE Solver for Parallel Computers. *Intl. J. High Perf. Comput. Apps.*, 13(4):254–365, 1999.

[16] Y. Cao, S. Li, L. Petzold, and R. Serban. Adjoint Sensitivity Analysis for Differential-Algebraic Equations: The Adjoint DAE System and Its Numerical Solution. *SIAM J. Sci. Comput.*, 24(3):1076–1089, 2003.

[17] S.D. Cohen and A.C. Hindmarsh. CVODE, a Stiff/Nonstiff ODE Solver in C. *Computers in Physics*, 10(2):138–143, 1996.

[18] K.L. Cooke and J. Turi. Stability, instability in delay equations modeling human respiration. *J. Math. Biol.*, 32:535–543, 1994.

[19] 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.

[20] D. Ellison. Efficient Automatic Integration of Ordinary Differential Equations with Discontinuities. *Math. Comput. Simut.*, 23(1):12–20, 1981.

[21] W.H. Enright. Software for Delay Differential Equations: Accurate Approximate Solutions are not Enough. Manuscript for Voltera 2004.

[22] W.H. Enright and L. Yan. The Quality/Cost Trade-off for a Class of ODE Solvers. *Numerical Algorithms*, DOI 10.1007/s11075-009-9288-x, 2009.

[23] 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.

[24] C.W. Eurichn, M.C. Mackey, and H. Schweglern. Recurrent Inhibitory Dynamics: The Role of State-Dependent Distributions of Conduction Delay Times. *J. Theor. Biol.*, 216:31–50, 2002.

[25] A.C. Fowler. Approximate Solution of a Model of Biological Immune Response Incorporating Delay. *J. Math. Biol.*, 13:23–45, 1981.

[26] A. Fuduli, M. Gaudioso, and G. Giallombardo. Minimizing Nonconvex Nonsmooth Functions via Cutting Planes and Proximity Control. *SIAM J. Optim.*, 14(3):743–756, 2004.

[27] C.W. Gear. Algorithm 407: DIFSUB for solution of ordinary differential equations [D2]. *Communications of the ACM*, 14(3):185–190, 1971.

[28] L. Genik and P. van den Driessche. An Epidemic Model with Recruitment-Death Demographics and Discrete Delays. In S. Ruan, G.S.K. Wolkowicz, and J. Wu, editors, *Differential equations with applications to biology*, number 21 in Fields Institute Communications, pages 237–249. co-publication of the AMS and Fields Institute, Providence, RI, 1999.

[29] R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel. *Hybrid Systems. Lecture Notes in Computer Science*, volume 736. Springer-Verlag, New York, 1993.

[30] N. Guglielmi and E. Hairer. Implementing Radau IIA Methods for Stiff Delay Differential Equations. *Computing*, 67:1–12, 2001.

[31] N. Guglielmi and E. Hairer. Computing breaking points in implicit delay differential equations. *Advances in Computational Mathematics*, 29(3):229–247, 2008.

[32] 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.

[33] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II. Stiff and differential-algebraic problems.* Springer Series in Computational Mathematics. Springer-Verlag, New York, 2nd edition, 1996.

[34] 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.

[35] U. an der Heiden. On the dynamics of nonlinear feedback loops in cell membrane. In S. Ruan, G.S.K. Wolkowicz, and J. Wu, editors, *Differential Equations with Applications to Biology*, number 21 in Fields Institute Communications, pages 143–158. co-publication of the AMS and Fields Institute, Providence, RI, 1999.

[36] K.L. Hiebert and L.F. Shampine. Implicitly Defined Output Points for Solutions of ODEs. Technical Report Sandia Report SAND80-0180, Dept. of Energy, Sandia Laboratories, Albuquerque, NM, 1980.

[37] W. Horbelt, J. Timmer, M.J. Bünner, R. Meucci, and M. Ciofini. Identifying physical properties of a CO2 laser by dynamical modeling of measured time series. *Physical Review E*, 64(016222), 2001.

[38] W. Horbelt, J. Timmer, and H.U. Voss. Parameter estimation in nonlinear delayed feedback systems from noisy data. *Physics Letters A*, 299:513–521, 2002.

[39] F.T. Krogh, J.P. Keener, and W.H. Enright. Reducing the Number of Variational Equations in the Implementation of Multiple Shooting. In U. Ascher and R.D. Russel, editors, *Numerical Boundary Value ODEs*, volume 5 of *Progress in Scientific Computing*, pages 121–135. Birkhäuser, Boston, 1985.

[40] Y. Kuang. On neutral delay logistic Gause-type predator-prey systems. *Dynamics and Stability of Systems*, 6:173–189, 1991.

[41] S.L. Lee and P. D. Hovland. Sensitivity Analysis Using Parallel ODE Solvers and Automatic Differentiation in C: SensPVODE and ADIC. Technical Report ANL/MCS-P818-0500, Argonne National Laboratory, May 2000.

[42] J.R. Leis and M.A. Kramer. The Simultaneous Solution and Sensitivity Analysis of Systems Described by Ordinary Differential Equations. *ACM Trans. Math. Soft.*, 14(1):45–60, 1988.

[43] M.C. Mackey and L. Glass. Oscillation and Chaos in Physiological Control Systems. *Science*, 197(4300):287–289, 1977.

[44] J.E. Mittler, B. Sulzer, A.U. Neumann, and A.S. Perelson. Influence of delayed viral production on viral dynamics in HIV-1 infected patients. *Math. Biosc.*, 152:143–163, 1998.

[45] K.A. Murphy. Estimation of Time- and State-Dependent Delays and Other Parameters in Functional Differential Equations. *SIAM Journal on Applied Mathematics*, 50(4):972–1000, 1990.

[46] K.W. Neves. Algorithm 497: Automatic Integration of Functional Differential Equations [D2]. *ACM Trans. Math. Soft.*, 1(4):369–371, 1975.

[47] K.W. Neves. Automatic Integration of Functional Differential Equations: An Approach. *ACM Trans. Math. Soft.*, 1(4):357–368, 1975.

[48] T. Park and P. Barton. State Event Location in Differential Algebraic Models. *ACM Transactions on Modeling and Computer Simulation*, 6(2):137–165, 1996.

[49] C.A.H. Paul. Developing a delay differential equation solver. *Appl. Numer. Math.*, 9:403–414, 1992.

[50] C.A.H. Paul. *Runge-Kutta Methods for Functional Differential Equations.* PhD thesis, Manchester Univ., England, 1992.

[51] C.A.H. Paul. A Test Set of Functional Differential Equations. Technical Report 243, Department of Mathematics, University of Manchester, Manchester, England, February 1994.

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

[53] K. Radhakrishnan and A.C. Hindmarsh. Description and Use of LSODE, the Livermore Solver for Ordinary Differential Equations. Technical Report UCRL-ID-113855, Lawrence Livermore National Laboratory, March 1994.

[54] F.A. Rihan. Sensitivity analysis for dynamic systems with time-lags. *Journal of Computational and Applied Mathematics*, 151:445–462, 2003.

[55] R. Serban and A.C. Hindmarsh. CVODES: An ODE Solver with Sensitivity Analysis Capabilities. Technical Report UCRL-JP-200039, Lawrence Livermore National Laboratory, September 2003.

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

[57] L.F. Shampine and S. Thompson. Solving Delay Differential Equations with dde23. Manuscript, available at http://www.runet.edu/∼thompson/webddes/index.html, 2000.

[58] L. Tavernini. The Approximate Solution of Volterra Differential Systems with State-Dependent Time Lags. *SIAM J. Numer. Anal.*, 15(5):1039–1052, 1978.

[59] S. Thompson and L.F. Shampine. A Friendly Fortran DDE Solver. *Appl. Numer. Math.*, 53(3):503–516, 2006.

[60] J.E. Tolsma and P. Barton. Hidden Discontinuities and Parametric Sensitivity Calculations. *SIAM J. Sci. Comput.*, 23(6):1861–1874, 2002.

[61] A.G. Vladimirov, D. Turaev, and G. Kozyreff. Delay differential equations for mode-locked semiconductor lasers. *Optics Letters*, 29(11):1221–1223, 2004.

[62] 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.

[63] H. Zivari. DDVERK90: A User-Friendly Implementation of an Effective DDE Solver. Master's thesis, Department of Computer Science, University of Toronto, Toronto, Canada, 2005.