An Efficient Unified Approach for the Numerical Solution of Delay Differential Equations

Hossein ZivariPiran and Wayne Enright
Department of Computer Science, University of Toronto
Toronto, ON, M5S 3G4, Canada
{hzp, enright}@cs.toronto.edu

Abstract

In this paper 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 standard step-by-step approach, such as Runge-Kutta or linear multi-step methods, and can provide dense output. This is done by treating a general DDE as a special example of a discontinuous IVP. Using this interpretation we develop an efficient technique to solve the resulting discontinuous IVP. We also give a more clear 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.

The new modular design for the resulting simulator we introduce, helps to accelerate the utilization of advances in the different components of an effective numerical method. Such components include the underlying discrete formula, the interpolant for dense output, the strategy for handling discontinuities and the iteration scheme for solving any implicit equations that arise.

1 Introduction

Differential equations are one of the most frequently used tools for mathematical modeling in engineering and life sciences. 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. 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 ([2], [4], [9], [11], [17], [20], [21]). 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 more 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 stepsize 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 paper 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

$$\begin{aligned} y'(t) &= f(t,y(t)), \text{ for } t_0 \le t \le t_F \\ y(t_0) &= y_0, \end{aligned} \tag{1}$$

where y and f are vector-valued functions. The resulting DDE solver that we develop can be applied to approximate the solution of a system of retarded delay differential equations (RDDE),

$$\begin{aligned} y'(t) &= f(t,y(t),y(t-\sigma_1),\ldots,y(t-\sigma_\nu)), \text{ for } t_0 \le t \le t_F \\ y(t) &= \phi(t), \text{ for } t \le t_0 \end{aligned} \tag{2}$$

or a system of neutral delay differential equations (NDDE)

$$\begin{aligned} y'(t) = f(t,y(t),y(t-\sigma_1),\ldots,y(t-\sigma_\nu),& \\ y'(t-\sigma_{\nu+1}),\ldots,y'(t-\sigma_{\nu+\omega})),& \text{ for } t_0 \le t \le t_F \\ y(t) = \phi(t),\ y'(t) = \phi'(t),& \text{ for } t \le t_0 \end{aligned} \tag{3}$$

where ϕ, the *history*, is a vector-valued function and $\sigma_i = \sigma_i(t,y(t)) \ge 0, i = 1,2,\ldots,\nu+\omega$, are scalar functions (which can in general be time and state dependent).

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 paper 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 5 we develop the details of a DDE solver based on this view and also the required interface with an IVP solver. In Section 6 we report on some numerical experiments with a DDE solver developed using this approach.

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 [1] or [8] and references therein.)

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

$$y'(t) = \begin{cases} f_1(t,y(t)), & \text{for } g(t,y(t)) < 0 \\ f_2(t,y(t)), & \text{for } g(t,y(t)) \geq 0 \end{cases} \tag{4}$$

To simulate the system (4), 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 (4) 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

$$\begin{aligned} z_n'(t) &= \begin{cases} f_1(t,z_n(t)), & \text{for } g(t,z_n(t)) < 0 \\ f_2(t,z_n(t)), & \text{for } g(t,z_n(t)) \geq 0 \end{cases} \\ z_n(t_n) &= y_n. \end{aligned} \tag{5}$$

3 Techniques for the Efficient Simulation of Discontinuous IVPs

Suppose that a solver is trying to compute a numerical approximation to the solution of (4) 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/discontiniuity 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 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 [5]. 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 (5) 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 1), that is, $z_{[c]}$ is the solution of the local IVP,

$$\begin{aligned} z'_{[c]}(t) &= f_i(t, z_{[c]}(t)), \text{ for } t_n \le t \le t_{n+1} \\ z_{[c]}(t_n) &= y_n, \end{aligned} \tag{6}$$

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 $t_n \le t \le t_{n+1}$.

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

$$g_{[c]}(t) = g(t, z_{[c]}(t)), \tag{7}$$

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,

$$z_n(t) = z_{[c]}(t), \text{ for } t_n \le t < \lambda. \tag{8}$$

Then, it is not hard to see that,

$$g_{[c]}(t) = g(t, z_n(t)), \text{ for } t_n \leq t < \lambda. \tag{9}$$

Considering the limit case,

$$\lim_{t \nearrow \lambda} g_{[c]}(t) = \lim_{t \nearrow \lambda} g(t, z_n(t)), \tag{10}$$

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

$$g_{[c]}(\lambda) = g(\lambda, \lim_{t \nearrow \lambda} z_n(t)), \tag{11}$$

which gives us,

$$g_{[c]}(\lambda) = 0, \tag{12}$$

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

Equation (12) defines another function that crosses zero at λ. However, there is a big difference which makes Equation (12) 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 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 $\overline{z}_{[c]}(t)$, 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

$$\overline{g}_{[c]}(t) = g(t, \overline{z}_{[c]}(t)). \tag{13}$$

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

A similar idea has been used by Park and Barton [14] for handling transitions in hybrid systems of differential algebraic equations.

4 DDEs as Discontinuous IVPs

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

$$\begin{aligned} y'(t) &= f(t, y(t), y(\alpha(t, y(t)))), \text{ for } t \geq t_0 \\ y(t_0) &= y_0, \\ y(t) &= \phi(t), \text{ for } t < t_0 \end{aligned} \tag{14}$$

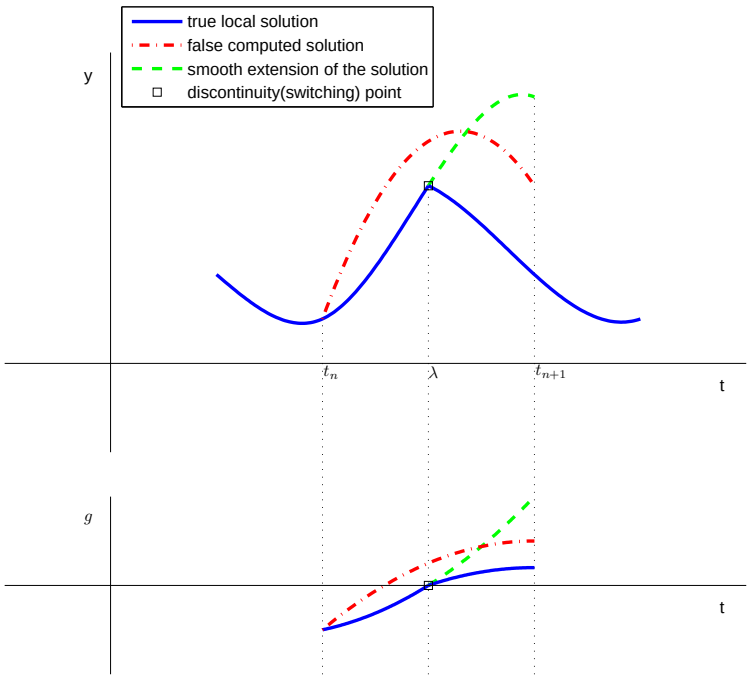

Figure 1: A typical situation for a state-dependent switching function. The true local solution refers to the exact solution of Equation (5); the false computed solution refers to the continuous approximate solution of Equation (5) (produced by a standard IVP method); and smooth extension refers to $z_{[c]}(t)$ (Equation (6)). 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

$$\cdots < \lambda_{-2} < \lambda_{-1} < \lambda_0 = t_0 < \lambda_1 < \lambda_2 < \cdots \tag{15}$$

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

$$g_i(t,y(t)) = \alpha(t,y(t)) - \lambda_i, \quad i = \ldots,-2,-1,0,1,2,\ldots \tag{16}$$

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

$$\lambda_r = \min\{\lambda > \lambda_{r-1} \colon \lambda \text{ is a root of odd multiplicity of } g_i(t,y(t)),\ i \leq r-1\}. \tag{17}$$

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,

$$\alpha(\lambda_r,y(\lambda_r)) = \lambda_{r-1},\ \ r \geq 1. \tag{18}$$

Using the explicit identification of all sources of non-smoothness, it is not hard to see that the solution of the system (14) also satisfies the following system of discontinuous IVPs,

$$\begin{aligned} y'(t) &= f_r(t,y(t)) = f(t,y(t),y_{[r]}(\alpha(t,y(t)))), \\ &\qquad \text{for } \ \lambda_r \leq \alpha(t,y(t)) < \lambda_{r+1} \\ y(t_0) &= y_0, \end{aligned} \tag{19}$$

where

$$y_{[r]}(\alpha) = \begin{cases} y(\alpha), & \text{for } \lambda_r \leq \alpha < \lambda_{r+1} \\ \text{smooth extension from } [\lambda_r,\lambda_{r+1}), & \text{for } \alpha < \lambda_r \text{ or } \alpha \geq \lambda_{r+1}. \end{cases} \tag{20}$$

The value of $y_{[r]}(\alpha)$ outside $[\lambda_r,\lambda_{r+1})$ is not required to be defined, as the right hand side of (19) switches if α goes outside this interval. Therefore, the smooth extension in (20) is only defined and used to facilitate the root-finding process during the numerical computations.

Now, using (16), Equation (19) can be rewritten in the standard form for discontinuous IVPs as

$$\begin{aligned} y'(t) &= f_r(t,y(t)), \\ &\qquad \text{for } g_r(t,y(t)) \geq 0 \text{ and } g_{r+1}(t,y(t)) < 0 \\ y(t_0) &= y_0. \end{aligned} \tag{21}$$

While (21) 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.

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), (21) can then be numerically integrated using the associated perturbed IVP,

$$f(t,y(t),y_{[r]}(\alpha(t,y(t)))) \approxeq f(t,y(t),\theta_{[r]}(\alpha(t,y(t)))). \tag{22}$$

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, (21),

$$\begin{aligned} &y'(t) = f_r(t,y(t)), \text{ for} \\ &\quad g_r(t,y(t)) \geq 0, \quad g_{r+1}(t,y(t)) < 0, \\ &\quad g_e^n(t,y(t)) \leq 0, \\ &y(t_n) = y_n. \end{aligned} \tag{23}$$

This formulation when combined with the approach for handling switching functions described in the previous section would impose the restriction that the stepsize $t_{n+1} - t_n$ 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 a constraint,

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

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

$$g_v(t,y(t)) = \alpha(t,y(t)) - t + \varepsilon, \tag{24}$$

and can be used to rewrite (23) as,

$$\begin{aligned} &y'(t) = f_r(t,y(t)), \text{ for} \\ &\quad g_r(t,y(t)) \geq 0, \quad g_{r+1}(t,y(t)) < 0, \\ &\quad g_e^n(t,y(t)) \leq 0, \\ &\quad g_v(t,y(t)) < 0, \\ &y(t_n) = y_n. \end{aligned} \tag{25}$$

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 (23) with,

$$\begin{aligned} &y'(t) = f_r(t,y(t)) = f(t,y(t),\theta_{[r]}(\alpha(t,y(t)))), \text{ for} \\ &\quad g_r(t,y(t)) \geq 0, \quad g_{r+1}(t,y(t)) < 0, \\ &\quad g_v(t,y(t)) > 0, \\ &y(t_n) = y_n, \end{aligned} \tag{26}$$

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 $\mathscr{P}_n$ in $[t_n, t_{n+1}]$.
2. Compute the solution using the interpolant $\mathscr{P}_n$ as a part of the history.
3. Update the interpolant $\mathscr{P}_n$ using the last computed solution.
4. If the sequence of updated interpolants has converged $\Rightarrow$ 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,

$$\begin{aligned} y'(t) &= \tilde{f}(t,y(t)) = f(t,y(t),(1-\xi)y(t_n)+\xi y(t)), \text{ for} \\ &\quad g_r(t,y(t)) \geq 0, \quad g_{r+1}(t,y(t)) < 0, \\ &\quad g_v(t,y(t)) > 0, \\ y(t_n) &= y_n, \end{aligned} \tag{27}$$

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 $\mathscr{P}_n$ in $[t_n, t_{n+1}]$ and switch back to our original equations (26) 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 $\mathscr{O}(h_n^2)$.

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 [3] for GLMs. The details for RK methods or LMMs can be derived by standard translations from GLMs (see [3] for details).

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

$$\begin{aligned} y'(t) &= f_r(t,y(t)) = f(t,y(t),y_{[r]}(\alpha(t,y(t)))), \text{ for} \\ & g_r(t,y(t)) \geq 0, \quad g_{r+1}(t,y(t)) < 0, \\ y(t_n) &= y_n. \end{aligned} \tag{28}$$

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

$$F_j = f_r(t_j, Y_j), \quad j = 1,2,\ldots,s \tag{29}$$

and

$$\begin{aligned} &\frac{\partial F_j}{\partial Y_k}, \quad j = 1,2,\ldots,s \\ &\qquad\quad k = 1,2,\ldots,s \end{aligned} \tag{30}$$

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.

5.2.1 Simultaneous Iterative Improvement

Assume that $\mathscr{P}_n$ 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

$$\theta_{[r]}(\alpha) = \begin{cases} \text{independent of } Y, & \text{for } \alpha < t_n \\ \mathscr{P}_n[Y, y^{[n]}](\alpha), & \text{for } \alpha \geq t_n \end{cases} \tag{31}$$

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 $(\frac{\partial \mathscr{A}}{\partial q})$ is used to indicate partial differentiation of a parametric multivariate function $\mathscr{A}$ w.r.t. a parameter or variable q, and $(\frac{d\mathscr{A}}{dq})$ is used to indicate the total derivative of such a function.

Computing F_j : In either case of (31) 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 (29) and using (19) and (22),

$$\begin{aligned}\frac{\partial F_j}{\partial Y_k} = &\frac{\partial f}{\partial y}(t_j, Y_j, \theta_{[r]}(\alpha(t_j, Y_j))) \times \delta_{jk} + \\ &\frac{\partial f}{\partial v}(t_j, Y_j, \theta_{[r]}(\alpha(t_j, Y_j))) \times \frac{d\theta_{[r]}(\alpha(t_j, Y_j))}{dY_k},\end{aligned} \tag{32}$$

where δ_{jk} denotes the Kronecker symbol.

Using (31),

$$\frac{d\theta_{[r]}(\alpha(t_j, Y_j))}{dY_k} = \begin{cases} \theta'_{[r]}(\alpha(t_j, Y_j)) \times \frac{\partial \alpha}{\partial y}(t_j, Y_j) \times \delta_{jk}, & \text{for } \alpha(t_j, Y_j) < t_n \\ \frac{d\mathscr{P}_n[Y, y^{[n]}](\alpha(t_j, Y_j))}{dY_k}, & \text{for } \alpha(t_j, Y_j) \geq t_n \end{cases} \tag{33}$$

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

$$\begin{aligned}\frac{d\mathscr{P}_n[Y, y^{[n]}](\alpha(t_j, Y_j))}{dY_k} &= \mathscr{P}'_n[Y, y^{[n]}](\alpha(t_j, Y_j)) \times \frac{\partial \alpha}{\partial y}(t_j, Y_j) \times \delta_{jk} + \\ &\quad \frac{\partial \mathscr{P}_n}{\partial Y_k}[Y, y^{[n]}](\alpha(t_j, Y_j)) \\ &= \theta'_{[r]}(\alpha(t_j, Y_j)) \times \frac{\partial \alpha}{\partial y}(t_j, Y_j) \times \delta_{jk} + \\ &\quad \frac{\partial \mathscr{P}_n}{\partial Y_k}[Y, y^{[n]}](\alpha(t_j, Y_j)),\end{aligned} \tag{34}$$

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

Combining (32), (33) and (34),

$$\begin{aligned}\frac{\partial F_j}{\partial Y_k} = &\left[\frac{\partial f}{\partial y}(t_j, Y_j, \theta_{[r]}(\alpha[j])) \right. \\ &\left. + \frac{\partial f}{\partial v}(t_j, Y_j, \theta_{[r]}(\alpha[j])) \times \theta'_{[r]}(\alpha[j]) \times \frac{\partial \alpha}{\partial y}(t_j, Y_j)\right] \times \delta_{jk} + \\ &\begin{cases} 0, & \text{for } \alpha[j] < t_n \\ \frac{\partial f}{\partial v}(t_j, Y_j, \theta_{[r]}(\alpha[j])) \times \frac{\partial \mathscr{P}_n}{\partial Y_k}[Y, y^{[n]}](\alpha[j]), & \text{for } \alpha[j] \geq t_n \end{cases}\end{aligned} \tag{35}$$

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

The first component of $\frac{\partial F_j}{\partial Y_k}$ in (35) 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 (35) 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.

5.3 Neutral Problems

For a system of NDDEs (3), 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 $\mathscr{P}'_n$ playing a similar role for $y'(\alpha)$ as $\mathscr{P}_n$ did for $y(\alpha)$. However, if we have to use Equation (27), 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 (32) 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 $\mathscr{P}_n$ should be replaced with $\theta'_{[r]}$ and $\mathscr{P}'_n$, respectively, in the subsequent Equations (33, 34, 35).

5.4 Extension to Multiple Delays

For a general system of DDEs with multiple delays (2), 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 (32).

6 Numerical Experiments

Based on the ideas presented here, we have developed an experimental code DDEM. The code 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 [6]).

6.1 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 (24) 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 [16]:

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

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 [12]:

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

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

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

for $t \leq 0$. 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 $n \geq 1$.
The exact solution of this problem is unknown.

Test Problem 3 [13]:

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

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

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

The exact solution to this problem is

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

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 [15]:

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

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

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

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 [19]:

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

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

$$y(0) = 0.$$

The exact solution to this problem is

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

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 [7]:

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

where

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

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

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

for $t \le 0$.
The exact solution of this problem is unknown.

6.2 Results

Here we present the numerical results for the chosen test problems. We include results for a new version [10] of RADAR5 (Guglielmi and Hairer [9]), and DDE_SOLVER (Thompson and Shampine [20]). We have set all absolute tolerances and the relative tolerance to TOL with TOL=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 :

Table 1: 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	$1.6 \cdot 10^{-8}$	$2.9 \cdot 10^{-7}$

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 2: 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	$2.1 \cdot 10^{-9}$	$3.8 \cdot 10^{-8}$

6.3 Discussions and Conclusions

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 [18], 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. The code DDEM and several driver programs are available at the address 'http://www.cs.toronto.edu/∼hzp'.

References

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

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

[3] Butcher, J.C.: Numerical Methods for Ordinary Differential Equations, 2nd Edition, J. Wiley, Chichester, (2008)

[4] Corwin, S. C. and Sarafyan, D. and Thompson, S.: 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)

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

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

[7] Genik, L. and Van Den Driessche P.: An Epidemic Model with Recruitment-Death Demographics and Discrete Delays. In: Ruan, S., Wolkowicz, G.S.K. and Wu, J. (eds.) Differential Equations with Applications to Biology, Fields Institute Communications, No. 21, 237–249, American Mathematical Society, Providence, RI, (1999)

[8] Grossman, R.L., Nerode, A., Ravn, A.P., and Rischel, H.: Hybrid Systems. Lecture Notes in Computer Science, Vol. 736, Springer-Verlag, New York, (1993)

[9] Guglielmi, N. and Hairer, E.: Implementing Radau II-A methods for stiff delay differential equations, Computing, 67, 1–12 (2001)

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

[11] Hayashi, H.: 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)

[12] Kuang, Y.: On Neutral Delay Logistic Gause-Type Predator-Prey Systems, Dynamics and Stability of Systems, 6, 173–189 (1991)

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

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

[15] Paul, C.A.H.: Runge-Kutta Methods for Functional Differential Equations, PhD. Thesis. University of Manchester (1992)

[16] Paul, C.A.H.: Developing a Delay Differential Equation Solver, Appl. Numer. Math., 9, 403–414 (1992)

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

[18] Paul C.A.H.: A Test Set of Functional Differential Equations, Numerical Analysis Report, Manchester Centre for Computational Mathematics, Manchester, England, No. 243, (1994)

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

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

[21] Willé, D.R. and Baker, C.T.H.: DELSOL - A Numerical Code for The Solution of Systems of Delay-Differential Equations, Appl. Numer. Math., 9, 223–234 (1992)