Journal of Dynamics and Differential Equations (2023) 35:1047–1082
https://doi.org/10.1007/s10884-021-10081-7

A Fixed Point Approach to Simulation of Functional Differential Equations with a Delayed Argument

Vincenzo M. Isaia[1]

Received: 25 September 2020 / Revised: 2 September 2021 / Accepted: 9 September 2021 /
Published online: 7 October 2021

Abstract
A computational method is developed for a family of functional differential equations in one independent variable with a single deviating argument, assumed to be a delay. These equations are mapped into a finite system of ODEs, a subsystem of which involves only the function controlling the delay. Key features include efficiency during method of steps, freedom from Jacobians and root finding techniques, and computing a continuous approximation. The nonlinear differential equations may be retarded, neutral or advanced. The method is established for state dependent delays, stiff equations, discontinuous initial history, a specific loss of monotonicity in the delay and extended naturally to distributed delays. When restricted to ODEs, the method extends naturally to PDEs; the hope is the delayed version here can eventually be extended to delayed PDEs. Conditions for convergence of the approximation are established, and results of numerical experiments are reported to indicate robustness of the implementation.

Keywords Functional differential equation · PSM · Picard iteration · Distributed delay · Nonmonotonic delay

Mathematics Subject Classification 65L03 · 34K07 · 34K40

1 Introduction

Functional differential equations arise when modeling situations in which earlier states of a system as well as its current state, are needed to direct the evolution. Examples range from control theory to disease transmission to the distribution of primes, see for example [1] and the references therein. While resembling ODEs, there can be substantial changes in a solution's behavior when a delay is introduced. It is also possible that derivatives of the solution may develop discontinuities, even if the vector field and initial data are smooth.

✉ Vincenzo M. Isaia
vincenzo.isaia@indstate.edu

[1] Department of Mathematics and Computer Science, Indiana State University, Terre Haute, IN 47809, USA

If there is an incompatibility between the vector field and initial data, then this results in a derivative discontinuity at the initial time, and would have potential to propagate in time.

The discussion begins with ODEs and a particular approach demonstrated in [2,3] and [4], known as the Power Series Method, *PSM*, which has very attractive features: it is capable of generating a Maclaurin expansion for the solution to IVPs involving ODEs. This is achieved without taking any derivatives of the vector field, and usually for a low computational cost, by leveraging Picard iteration with a polynomial environment. There is also an explicit a priori error bound independent of the vector field's derivatives.

The purpose here is to modify PSM to work in the presence of a single delayed argument, while attempting to retain the low cost, high accuracy attributes. Specifically, since coefficients for a polynomial are computed, à la finite elements, the method bypasses the need for dense output. The order of the method is flexible as well, and in practice it is to an extent controlled by a user input. Standard monomials may not be the most robust basis for particular situations, but when they are a viable option, they integrate very easily.

The deviating argument is assumed to be an explicit delay, as opposed to a delay of threshold type, e.g. see [5], and the label DDE is used for differential equations whose vector fields depend on arguments with an explicit delay. The initial focus is on a delay which enacts a discrete time sampling of the solution's history, including those with state dependence. A specific continuous time sampling, namely a distributed delay is also considered.

The method that is developed here is related to the one given in [6], in that it uses PSM as a motivation. However, there are significant differences in the execution, making them independent of each other. The method presented here applies Picard iteration up to a stopping iterate over a fixed time interval, then moves to the next interval and repeats. The approach in [6] applies each Picard iteration globally in time, i.e. over all possible time intervals.

This method, *PSMd*, is much more efficient numerically than the one in [6]. However, there is a more complicated situation concerning error bounds here than in [6]. The error generated in any particular time interval is transferred to the next time interval. Hence, stability of the vector field with respect to (wrt) the expansion errors is important to understand, while both PSM and the method in [6] have no stability considerations. Also, the method in [6] currently is better suited for multiple delays, while PSMd as developed here applies to a single delay.

As compared with other methods, PSMd as a MATLAB code does not require dynamic memory allocation and tends to run quickly. Versus codes written in FORTRAN, the codes for PSMd are not very cumbersome. Codes can be reused very easily with minor modifications to capitalize on problem specific advantages. Passing information from one section of a problem to another sometimes requires attention to detail on the programmer's part, as such the change of variables needed to develop PSMd are given across several steps to aid that situation.

The PSM philosophy combines polynomial vector fields via auxiliary variables with Picard iteration, see [3]. A brief summary of PSM will appear in Sect. 2 to give a framework for computational issues when the delayed argument is present. It also addresses the mild assumption, known as *projectively polynomial*, for the vector fields under consideration, which is not needed until Sect. 4. In addition to the benefits it provides for ODEs, the use of Picard iteration neutralizes the difficulty of approximating state dependent delays.

Existence and uniqueness of solutions are important when numerical simulations are concerned. Certain DDEs can have existence or uniqueness issues, even with smooth problem data. The content of Sect. 3 begins with the family of problems, along with a discussion of notation. An example concerning existence issues is presented, which also identifies how uniqueness issues can occur. Assuming compatibility and Lipschitz continuity in most arguments from the vector field and delay, a proof of existence and uniqueness is given. There are

no requirements on the Lipschitz constants. The proof provides the justification for PSMd. The primary goal of this paper is not to explore existence and uniqueness issues, however they also can not be ignored.

When applicable, the *method of steps* (MoS) solves a DDE over one subinterval at a time, and then uses that solution for the delayed terms in the next subinterval, see [7]. Over these subintervals, the DDE appears to be an ODE, since the delayed terms are considered known. Additionally, the previously mentioned jump discontinuities are possible in the solution's derivatives, but if the initial data, vector field, delay are all smooth, jumps can only appear at the MoS endpoints. Using this framework, there is a discussion of the propagation of jump discontinuities in the derivatives of the solution. This discussion and notation for MoS are the first half of Sect. 4. Smoothness is not mandatory for the initial data with PSMd, but is mandatory for the vector field and delay.

Formally, the subintervals are how far forward in time the solution can be determined based on the information that is known. Suppose information defined on $[-a, 0]$ is given for some $a \in \mathbb{R}^+$ and assume a delay which is increasing in time. The future time $t^* > 0$ which delays to 0 would represent the largest time for which the vector field's delayed terms still reference the given information. The DDE can then be solved or approximated over $(0, t^*)$, which would be the first MoS subinterval.

If the delay continues to increase, then the solution/approximation over $(0, t^*)$ can be used as the new given information. The time $t^{**} > t^*$ which delays back to t^* can be sought, and the problem can then be solved or approximated over (t^*, t^{**}). This process is then continued as long as the delay increases with time and doesn't ever equal the current time.

Extending a polynomial vector field to the delay case comes from a very specific change of variable, which is examined in the rest of Sect. 4. This change of variable, in the case of discrete time sampling, removes the delayed argument from the history terms and renders the DDE into a system of ODEs. It converts the delay into a single coefficient on the original vector field. The delay also generates another system of ODEs which compute the MoS endpoints without using root finding techniques along with the coefficient in the first system. In addition, both systems have a fixed number of components for all time.

The presentation of PSMd and its algorithm, an error analysis and discussion of convergence possibilities for PSMd begin Sect. 5. Generalizations of the differential equation and initial data are considered. In particular, the notion of subdividing the MoS subinterval is developed. Subdivisions play a major role in the approach's ability to handle problems involving stiff equations, nonhomogeneous (forcing) terms and discontinuous initial data. The accommodations needed for delays which decrease and in particular, the failure of MoS, and distributed delays, an integral form of continuous time sampling, conclude this section.

The results of a variety of numerical experiments are presented in Sect. 6 that demonstrate the robustness of PSMd. These experiments involve most of the situations covered in this paper. In this paper, $\mathbb{Z}^+$ represents $1, 2 \ldots$ and $\mathbb{Z}_0^+ = \mathbb{Z}^+ \cup \{0\}$.

2 PSM for ODEs

This subsection contains a summary of the PSM approach. The state of a system is denoted $u(t)$. The unique solution to an IVP with a nonlinear autonomous vector field is called *projectively polynomial* if the solution $u(t)$ is a component of the solution for some equivalent system with a polynomial vector field.

The change of variables, which convert the original vector field to an autonomous polynomial equivalent, is comprised of *auxiliary variables*. The solution to the original system along with the auxiliary variables make up the solution to the equivalent polynomial system. An example will demonstrate these ideas.

Example Suppose $\dot{u} = u \cos u$, and note that the vector field is not polynomial due to the cos term. Introducing $U = \cos u$ and $V = \sin u$, yields the system

$$\begin{cases} \dot{u} = uU \\ \dot{U} = -V\dot{u} = -uUV \\ \dot{V} = U\dot{u} = uU^2 \end{cases}$$

where the first component of $\mathbf{u} = [u, U, V]$ solves $\dot{u} = u \cos u$ and the vector field for the new system is polynomial, in fact cubic. Hence, the solution to $\dot{u} = u \cos u$ is projectively polynomial and U, V are auxiliary variables.

This vector field can be reduced to a quadratic vector field; one possibility is to introduce two more auxiliary variables: $W = uU$ and $X = uV$. This demonstrates that the scalar transcendental vector field $u \cos u$ is equivalent to a 5th order system of quadratic vector fields (or a 3rd order cubic system).

Finding a set of auxiliary variables that converts the original vector field into a polynomial vector field is not overly difficult and was given in a NASA technical report by Fehlberg [8]. Coupling this idea with Picard iteration and developing the resulting theory was done by Parker and Sochacki et al., see [4] and references therein.

For any given projectively polynomial solution, the equivalent polynomial vector field (not unique) may be transformed into a quadratic one if enough auxiliary variables are used. In the sequel, all polynomial vector fields are reduced to quadratic form, as far as any theoretical or presentation matters are concerned. For computational purposes, it may or may not be worth the effort to enact the reduction.

Vector fields which cannot be transformed into polynomial ones do exist, but they must be constructed carefully, see [2] for details. Only IVPs with projectively polynomial solutions are considered here, and this is the mild assumption mentioned in Sect. 1. Also, see the discussion following (14).

If the initial time for the IVP is zero, PSM produces the Maclaurin expansion for each component of the system. The initial data provides the constant term, the first iteration produces the linear term, and in general, the degree of the expansions increases by one for each Picard iteration performed. From a computational point of view, this generation of the Maclaurin expansion occurs without taking any derivatives of the vector field. All the claims made in this paragraph follow from the results in [3].

The key ingredient to the efficiency of PSM is that in any given iteration, it only computes terms that produce new information, and these terms will subsequently be invariant with increasing iteration. The relevant terms are discrete convolutions of the polynomial coefficients. This structure for the expansion is retained during the move to PDEs, see [9].

Looking forward, an issue that arises from a delay is the presence of mixed terms t^i and powers of the delay, whose product may require significant work to integrate. This is circumvented by the change of variable in Sect. 4.

2.1 Computational Considerations

The computational aspect and its efficiency are now examined to shed light on some implementation details that are relevant for the deviating argument case. The focus will be on one equation out of a system, with a single term quadratic vector field: $\dot{w} = uv$.

Let $u(t) = \sum_{i=0}^{d} U_i t^i$ be polynomial in t and represent $u(t)$ by $[U_0, U_1, \ldots, U_d]$. Similarly let $[V_0, \ldots, V_d]$ represent $v(t)$ and $[W_0, \ldots, W_d]$ represent $w(t)$. The focus is on one iteration, which computes the 'next' coefficient of W's expansion.

For illustrative purposes, consider the case $d = 3$ and a 2D array using the monomials $U_i t^i$ and $V_i t^i$ as row and column headings, respectively. If the 2D array's entries are the product of the row and column heading, then summing the array's entries would represent the product $u(t)v(t)$.

Figure 2 contains a visual representation of the product $u(t)v(t)$ for the case $d = 3$. The choice of row and column headings is for presentation purposes: this array, when viewed as a matrix, has its diagonals containing the same power of t in the product.

In particular, the trace is the discrete convolution of the polynomials currently representing the expansion

$$U_0 V_3 + U_1 V_2 + U_2 V_1 + U_3 V_0$$

and it produces the coefficient on the t^3 term in the product $u(t)v(t)$, i.e. $a_3 = \sum_{i=1}^{3} U_i V_{3-i}$ where $\sum a_i t^i$ is the expansion of the product uv.

The role of integration is to transfer the given coefficient information to the coefficient in the next power, i.e. U_i and V_i with $i = 0 \ldots 3$ are used to obtain W_4. Computing W_4 reduces to computing a_3 and integrating (which amounts to dividing by an appropriate integer). In addition, only the main diagonal is needed to compute W_4, see [3] for a proof.

Since the initial data are invariant, so is the linear term in the approximation for each component. This can be extended via induction, to show that the underlined terms in the array are also invariant: if the given coefficients, U_i, V_i, $i = 0 \ldots 3$, are invariant wrt further Picard iteration, then so is the coefficient for the next power. Hence, once these coefficients are computed, they need not be recomputed during later iterations.

Individual terms below the main diagonal are also invariant wrt further Picard iteration, but that particular convolution is not invariant, since all of the terms which would contribute to that convolution have not been introduced yet. For example, the terms $V_4 t^4$ and $U_4 t^4$ are not available, but are needed to compute the t^5 coefficient.

This is the key ingredient for efficiency with PSM: in a given iteration, it computes only the terms in the array that produce new information, and will be invariant with increasing iteration. These are the terms that make up the current main diagonal, i.e. just compute the trace.

2.2 Integration Algorithm

PSM can be quantified as follows: let $N \in \mathbb{Z}^+$ be the size of a given IVP's system when it has a polynomial vector field, and let its components be labeled u^λ, with $\lambda \in \{1, \ldots, N\}$. For presentation convenience, if the vector field contains linear terms, a dummy component of $u^0 = [1, 0, \ldots, 0]$ is used so that all terms appear to be quadratic. Suppose an arbitrary component to a system has a d degree polynomial representation already known. Denote this by $u^\lambda(t) = \sum_{i=0}^{d} U_i^\lambda t^i$ with representation $[U_0^\lambda, \ldots, U_d^\lambda]$. The method would then expand the degree to $d + 1$ by computing the next coefficient U_{d+1}^λ.

In algorithm form, given any IVP whose solution is projectively polynomial

1. Convert the IVP via auxiliary variables to have an autonomous, polynomial vector field with initial time zero. Write the ODE for each component u^λ with $\lambda \in \{1, \ldots, N\}$ as

$$\dot{u}^\lambda = C^\lambda + \sum_{\lambda_1=0}^{N} \sum_{\lambda_2=\lambda_1}^{N} C^\lambda_{\lambda_1\lambda 2} u^{\lambda_1} u^{\lambda_2}$$

   where the λ_1, λ_2 entry in the coefficient matrix C^λ is zero if the pairing of components λ_1 and λ_2 does not contribute to the vector field.
2. For each $\lambda \in \{1, \ldots, N\}$, enact the Picard iteration
   a. Compute trace for each pair (λ_1, λ_2): $c^\lambda_{\lambda_1\lambda_2} = \sum_{i=0}^{d} C^\lambda_{\lambda_1\lambda_2} U_i^{\lambda_1} U_{d-i}^{\lambda_2}$
   b. Sum over pairs: $c^\lambda = \sum_{\lambda_1} \sum_{\lambda_2 \geq \lambda_1} c^\lambda_{\lambda_1\lambda_2}$
   c. Integrate and update: $\dfrac{c^\lambda}{d+1} = U^\lambda_{d+1}$
3. $d \mapsto d + 1$ and repeat steps 2-3 until $d = d_{\text{stop}}$.

and this encapsulates the PSM approach. The list in step 2 will be dubbed the *integration algorithm*, and ultimately, this is coupled with a different step 1 for the delay case. The determination of auxiliary variables when a delay is involved is covered in more detail in Sect. 4.

In closing, note that the order of the Maclaurin expansion is equal to the number of Picard iterations performed. The accuracy in the PSMd approximation can be improved, to a certain extent, by increasing the number of iterations, although the approximation is constrained by being a Maclaurin polynomial, which can converge very slowly in some cases.

3 DDEs and Successive Approximation

For convenience, a scalar functional differential equation of arbitrary order with a delayed argument is used as the starting point, although it will be apparent that this approach is applicable to systems as well. In addition to assuming that the deviating argument is a delay, assume $(p, q) \in \mathbb{R}^2$ with Euclidean metric and that $\Delta(p, q)$ is uniformly Lipschitz continuous wrt the second variable with Lipschitz constant C_Δ and continuous in the first variable.

The state dependent deviating argument would be given by $\Delta(t, u(t))$, which is relabeled $\Delta_u(t)$ and the condition that this be a delay implies $\Delta_u(t) \leq t$ for all t of interest. So as not to confuse Δ_u with a partial derivative, which is needed later, partials of Δ are indicated with ∂.

Let $L_1, L_2 \in \mathbb{Z}^+$, $t_0 \in \mathbb{R}$ and D be the differential operator, which is used until arbitrary order derivatives are not required. The family of problems under consideration, with $f : (0, T) \times \mathbb{R}^{L_1} \times \mathbb{R}^{L_2} \to \mathbb{R}$ and $\Delta : (0, T) \times \mathbb{R} \to \mathbb{R}$, is given by

$$\begin{cases} D^{L_1+1} u(t) = f\left(t, D^{\mathbf{L}_1} u(t), D^{\mathbf{L}_2} u\left(\Delta_u(t)\right)\right) & t > t_0 \\ \qquad u(t) = \Psi(t) & t \in [a, t_0] \end{cases} \tag{1}$$

with the lists $\mathbf{L}_1, \mathbf{L}_2$ on D indicating an ordered list of derivatives, where the entries specify which derivatives are present. For example, $D^{[0,2,4]}u(t)$ would indicate that f has $u(t)$, $D^2u(t)$ and $D^4u(t)$ as its arguments. For convenience during analysis, take $\mathbf{L}_1 = [0, \ldots, L_1]$ and let f absorb unused terms. Similarly, take $\mathbf{L}_2 = [0, \ldots, L_2]$, with L_2 independent of L_1.

Even though the data Ψ, referred to as *initial history*, only needs to defined over $[a, t_0]$ in the DDE, the upcoming existence and uniqueness proof requires $\Psi : [a, T) \to \mathbb{R}$ where T is the right end point for the domain of the DDE's solution. The word 'history' unqualified is a generic label identifying what known terms represent the delayed argument terms that occur at times away from the initial time. The left endpoint of the initial history's domain, denoted a, depends on both Δ_u and Ψ. It is based on the minimum value achieved by $\Delta_u(t)$ over the solution's domain, which may not be known a priori. Assuming the delay is increasing, then $a = \Delta_\Psi(t_0)$.

One can arrive at (1) from the more general form

$$F\left(t, D^{\mathbf{L}_1+1}(t), D^{\mathbf{L}_2}\left(\Delta_u(t)\right)\right) = 0$$

provided the inverse function theorem holds, allowing $D^{L_1+1}u(t)$ to be isolated. This situation can be approximated with proper modifications to PSMd, but this is not undertaken here.

Following [10], equations can be classified by comparing the highest derivative without a delayed argument to the highest derivative with a delayed argument. In particular, if $L_1+1 > L_2$ the equation is considered to be *retarded*, if $L_1+1 = L_2$, then the equation is considered to be *neutral*, while if $L_1+1 < L_2$ the equation is considered to be *advanced*. This classification effects the propagation of any discontinuities which may be present in derivatives of the solution, which is discussed further in the next subsection.

In expectation of future notation, superscripts will be used for vector components. Define $L \equiv \max\{L_1, L_2\}$ and denote $\mathbf{u} = \left(u^l\right)_{l=0}^{L}$ as well as $\boldsymbol{\Psi} = \left(\Psi^l\right)_{l=0}^{L}$ with $u^l = D^l u$ and $\Psi^l = D^l\Psi$. This provides the standard change of variables to convert (1) to a first order system. Letting $\mathbf{f}$ absorb unused arguments, (1) can be written

$$\begin{cases} \dot{\mathbf{u}}(t) = \mathbf{f}\left(t, \mathbf{u}(t), \mathbf{u}(\Delta_u(t))\right), & t > t_0 \\ \mathbf{u}(t) = \boldsymbol{\Psi}(t), & t \in [a, t_0] \end{cases} \tag{2}$$

where Δ_{u^0} has been shortened to Δ_u since the component u^0 of $\mathbf{u}$ in (2) is the solution to (1). Note that the system version (2) contains retarded, neutral and advanced problems in one framework.

3.1 Breaking Points

This subsection highlights the need for a definition of a solution to (2). The definition of interest here is based on the one given in [11] (see references therein), but the presentation is postponed until some notation is developed in Sect. 4.1.

To discuss existence and uniqueness, an understanding of how jump discontinuities in a derivative of the solution interact with the DDE is in order. The derivative that has the jump discontinuity can be of any order. *Breaking points*, see [12], are times at which a jump discontinuity appears in some derivative of the solution. The order of a breaking point equals the highest order derivative of the solution which is still continuous at the breaking point.

In the sequel, an nth order *jump* implies a jump discontinuity in the $n+1$ order derivative of the solution. Note that a jump of a certain order implies all derivatives with higher orders do not exist at the breaking point. However, those higher order derivatives may not necessarily jump; the discontinuity could be removable. The concern here is always with the lowest order derivative that has a jump.

If $\mathbf{f}$, $\boldsymbol{\Psi}$ and Δ_u are all smooth, then the chain rule shows that breaking points can only occur at the MoS endpoints as was mentioned in Sect. 1. Because of the jumps, these endpoints are also the possible locations for existence and uniqueness issues. Failure of existence is *termination* of the solution; the incompatibility of $\mathbf{f}$ and Δ_u prevents extension of the current solution, at least locally. Another possibility is that more than one solution can be constructed and there is a *loss of uniqueness*.

A breaking point t_b is classified as *resolved* if there exists a unique solution over $[t_b, A]$ for some $A > t_b$. When either termination or loss of uniqueness occurs at a breaking point, it is classified as *unresolved*. Classifying breaking points, determining termination vs loss of uniqueness, and if termination occurs, whether there is reappearance of the solution at later times, are all part of a decision making process that would sit on top of PSMd, which itself is just the computational engine.

Concerning resolved breaking points, taking derivatives of the vector field can be used to show the effect of retarded, neutral and advanced equations on jumps. As a general rule, a retarded equation tends to smooth out a jump that occurs, by propagating it to the next breaking point, but at a higher derivative order. A neutral equation will propagate the jump at the same order, while for an advanced equation, the order of the jump will decrease until the jump reaches $\dot{u}$.

For a quick view of the mechanism of an unresolved breaking point, this modification to a problem from [11] is presented.

Example Consider

$$\dot{u}(t) = -\dot{u}\left(u(t) - 2\right), \quad t > 0, \qquad u(t) = 1 - t, \quad t \leq 0$$

which has solution $u(t) = 1 + t$ over $t \in (0, 1)$ and this is all that is needed here. The delay $\Delta_u(t) = u(t) - 2$ along with the initial history of $1 - t$ are needed to compute the left endpoint $t = -1$ for the initial history's domain and to determine the next breaking point at $t = 1$. What must be confronted now is the state dependency of the delay past the breaking point with the solution still unknown.

On the right side of the breaking point, $\dot{u}(1^+) = -\dot{u}(u(1^+) - 2)$. If $u(t) - 2 > 0$ as $t \to 1^+$, then $\dot{u}(1^+) = -\dot{u}(0^+) = -1$. But the condition $u(t) - 2 > 0$ imposes a condition on $\dot{u}(1^+)$: since $u(1) = 2$ and u needs to increase to make $u(t) - 2 > 0$, then $\dot{u}(1^+) > 0$ which is a contradiction.

If there was existence and uniqueness for this problem, one would find that upon looking at $u(t) - 2 < 0$, this would be compatible with the vector field and give a value for $\dot{u}(1^+)$ that is consistent. But the assumption $u(t) - 2 < 0$ produces $\dot{u}(1^+) = -\dot{u}(0^-) = -(-1) = 1$, which contradicts u decreasing from 2.

Since neither option is viable, the solution terminates at $t = 1$. Sometimes, incompatibility is temporary and at a later time, a solution can be defined again. It is also possible that $\mathbf{f}$ and Δ_u are 'too' compatible and that both conditions produce consistent results, which would generate two viable solutions. Note that if these double solutions are subject to a breaking point in the future, there is the possibility of having more than two solutions.

PSMd will compute the MoS endpoints, and hence the breaking points. Resolved breaking points pose no issue for PSMd as they are not integrated over. When unresolved breaking

points occur, only testing for loss of uniqueness or termination (and possible reappearance) of the solution are the required modifications to PSMd. This is a codification of the analysis in the example, which tests whether the Picard computation is meaningful.

As opposed to unresolved breaking points, more attention will be given to situations when MoS fails, which need to be handled more carefully. This is seen in the problem of Castleton and Grimm, see [13], and for which numerical results are given in Sect. 6.8 on page 38.

3.2 Existence and Uniqueness Proof

In order to ensure existence and uniqueness, $\mathbf{f}$ and Δ_u need to be *compatible* see [11] in particular, the inequality at the bottom of page 352. The theorem will apply up to the first unresolved breaking point, as the Picard iteration does not detect termination or loss of uniqueness unless analysis similar to the example is performed on the computed polynomial.

Under these circumstances, the assumptions needed for the existence and uniqueness proof are now collected. Suppose $\mathbf{f}$ in (2) is continuous with respect to t and uniformly Lipschitz continuous in the remaining variables on $[t_0, T^*] \times [-U, U]^{2L}$, with fixed U, $T^* < \infty$. Such vector fields will be referred to as *admissible*. Denote by C_1^l the Lipschitz constant of an admissible $\mathbf{f}$ with respect to component $u^l(t)$ of $\mathbf{u}(t)$, and denote by C_2^l the Lipschitz constant for component $u^l(\Delta_u(t))$ of $\mathbf{u}(\Delta_u(t))$. In addition, define

$$C_f \equiv \sum_{l=0}^{L} C_1^l + C_2^l \quad \text{and} \quad M_f \equiv \max_l \max_{[t_0,T^*]} \mathbf{f} \tag{3}$$

where the T^* dependence in M_f has been suppressed.

The assumption needed for Ψ as far as the existence and uniqueness result is concerned is a bound on height difference. In particular, given an admissible $\mathbf{f}$, for any fixed a, $T > 0$ such that $a < T < T^*$, define

$$M_\Psi \equiv \max_l \left(\max_{t\in[a,T]} \Psi^l(t) - \min_{t\in[a,T]} \Psi^l(t) \right)$$

where the T dependence has been suppressed. If $M_\Psi < \infty$ then Ψ is called *admissible*. If Ψ is continuous, then Ψ is admissible since $[a, T]$ is compact.

Existence and uniqueness can be extended to (2) over compact intervals in time prior to the first unresolved breaking point. The method is via the method of successive approximations: applying Picard iteration to the case of admissible vector fields and initial histories. Alternate notation $|u(t) - v(t)| = |u - v|(t)$ will be used when convenient.

Proposition 1 *For a given (2) with continuous Ψ and admissible $\mathbf{f}$ over the interval $[t_0, T^*]$, denote the smallest unresolved breaking point of (2) by t_b^*. If there is no unresolved breaking point, then $t_b^* = T^*$. Define the sequence $\mathbf{u}_k$ by $\mathbf{u}_0(t) = \boldsymbol{\Psi}(t)$ for $t \in [a, T^*]$ and for $k > 0$, define*

$$\mathbf{u}_{k+1}(t) = \begin{cases} \boldsymbol{\Psi}(t), & t \in [a, t_0] \\ \mathbf{u}_k(t_0) + \displaystyle\int_{t_0}^{t} \mathbf{f}\left(s, \mathbf{u}_k(s), \mathbf{u}_k(\Delta_{u_k}(s))\right)\, ds, & t > t_0 \end{cases} \tag{4}$$

The sequence $\mathbf{u}_k(t)$ converges uniformly for $t \in [a, T]$ for every $T < t_b^*$ and $\lim_{k\to\infty} \mathbf{u}_k(t)$, solves (2) uniquely whenever $\mathbf{u}(\Delta_u(t))$ is defined.

Proof For the entire proof, superscripts will denote components of $\mathbf{u}$, and subscripts will refer to iteration numbers.

For any fixed $T < T^*$, consider the compact interval $[t_0, T]$. Continuity of Ψ^l, the Fundamental Theorem of Calculus and induction yield a well defined $\mathbf{u}_k$ for all $k \in \mathbb{Z}_0^+$ and $t \in [t_0, T]$, and by hypothesis, $\mathbf{u}_k(t) = \boldsymbol{\Psi}(t)$ if $t \in [a, t_0]$.

Let $\mathbf{f}_k(t) \equiv \mathbf{f}(t, \mathbf{u}_k(t), \mathbf{u}_k(\Delta_{u_k}(t)))$ with components f_k^l for any $1 \leq l \leq L$. Consider (4) after the first iteration, and taking the maximum over all components produces

$$\max_l |u_1^l - u_0^l|(t) = \max_l \left| \Psi^l(t_0) + \int_{t_0}^t f_0^l(s)\, \mathrm{d}s - \Psi^l(t) \right| \leq M_\Psi + M_f(t - t_0)$$

for $t \in [t_0, T]$. For $t \in [t_0, T]$, then $\Delta_{u_0}(t) \in [a, \Delta_{u_0}(T)] \subset [a, T]$.

If $\Delta_{u_1}(t) \geq t_0$, then it follows that

$$\begin{aligned} \max_l \left| u_1^l\left(\Delta_{u_1}(t)\right) - u_0^l\left(\Delta_{u_0}(t)\right) \right| &= \max_l \left| \Psi^l(t_0) + \int_{t_0}^{\Delta_{u_1}(t)} f_0^l(s)\, \mathrm{d}s - \Psi^l(\Delta_{u_0}(t)) \right| \\ &\leq M_\Psi + M_f(\Delta_{u_1}(t) - t_0) \end{aligned}$$

thus $\max_l \left| u_1^l\left(\Delta_{u_1}(t)\right) - u_0^l\left(\Delta_{u_0}(t)\right) \right| \leq M_\Psi + M_f(t - t_0)$ since Δ_{u_k} is a delay for any k.

If $\Delta_{u_1}(t) < t_0$, then

$$\max_l \left| u_1^l\left(\Delta_{u_1}(t)\right) - u_0^l\left(\Delta_{u_0}(t)\right) \right| = |\Psi^l(\Delta_{u_1}(t)) - \Psi^l(\Delta_{u_0}(t))| \leq M_\Psi$$

and it again follows that $|u_1^l(\Delta_{u_1}(t)) - u_0^l(\Delta_{u_0}(t))| \leq M_\Psi + M_f(t - t_0)$ for $t > t_0$.

Defining

$$E_0(t) \equiv \max \left\{ \max_l \left| u_1^l - u_0^l \right|(t), \max_l \left| u_1^l(\Delta_{u_1}(t)) - u_0^l(\Delta_{u_0}(t)) \right| \right\}$$

it follows that the previous bounds on $|u_1^l - u_0^l|(t)$ and $|u_1^l(\Delta_{u_1}(t)) - u_0^l(\Delta_{u_0}(t))|$ can be summarized by the single bound $E_0(t) \leq M_\Psi + M_f(t - t_0)$.

In general, define

$$E_k(t) \equiv \max \left\{ \max_l \left| u_{k+1}^l - u_k^l \right|(t), \max_l \left| u_{k+1}^l(\Delta_{u_{k+1}}(t)) - u_k^l(\Delta_{u_k}(t)) \right| \right\}$$

and it will be shown that for each l, the differences in the vector field can be bound by the error: $|f_{k+1}^l - f_k^l|(t) \leq C_f E_k(t)$ with C_f given in (3).

To clarify the upcoming notation, a verbal description of the process is given first. The difference in the vector field is telescoped and regrouped. The telescoping terms that are introduced have the previous iteration's $(k-1)$ information in each argument until a certain argument is reached, after which the information comes from the next iterate (k). The terms are regrouped so that the subtractions involve vector field terms which differ in only one argument, and the result follows from the triangle inequality.

Introduce $\mathbf{u}_k^l = \{(u_i)_k^l\}_{i=1}^L$ where $l = 0, \ldots, L$. For a fixed l, define the components via $(u_i)_k^l = u_k^i$ when $l < i$, and $(u_i)_k^l = u_{k-1}^i$ when $i \leq l$. Inserting $u_k^0 = u_k$ in the delay then the difference $|f_{k+1}^l - f_k^l|(t)|$ can be written

$$|f^l(\mathbf{u}_{k+1}(t), \mathbf{u}_{k+1}(\Delta_{u_{k+1}}(t))) - f^l(\mathbf{u}_k(t), \mathbf{u}_k(\Delta_{u_k}(t)))|$$

$$= \left| \sum_{m=1}^{L} f^l(\mathbf{u}_{k+1}^{m-1}(t), \mathbf{u}_{k+1}(\Delta_{u_{k+1}}(t))) - f^l(\mathbf{u}_k^m(t), \mathbf{u}_{k+1}(\Delta_{u_{k+1}}(t))) \right.$$
$$\left. + \sum_{m=1}^{L} f^l(\mathbf{u}_k(t), \mathbf{u}_{k+1}^{m-1}(\Delta_{u_{k+1}}(t))) - f^l(\mathbf{u}_k(t), \mathbf{u}_k^m(\Delta_{u_k}(t))) \right|$$
$$\leq \sum_{m=1}^{L} C_1^m \left| u_{k+1}^m(t) - u_k^m(t) \right| + \sum_{m=1}^{L} C_2^m \left| u_{k+1}^m(\Delta_{u_{k+1}}(t)) - u_k^m(\Delta_{u_k}(t)) \right| \tag{5}$$

after applying the triangle inequality and the Lipschitz condition. This implies

$$|f_{k+1}^l - f_k^l|(t) \leq C_f E_k(t) \tag{6}$$

Generalizing the previous bounds, define $P_k(t) \equiv \left(M_\Psi \frac{(t-t_0)^k}{k!} + M_f \frac{(t-t_0)^{k+1}}{k+1!} \right)$. Denote $C_* = C_f(1 + C_\Delta M_f)$ and C_*^k as the kth power of C_*. Note that the bound $E_k(t) \leq C_*^k P_k(t)$, holds when $k = 0$. It will now be demonstrated via induction that $E_k(t) \leq C_*^k P_k$ for all k. Using this bound and (6), there is for each l

$$|u_{k+2}^l - u_{k+1}^l|(t) \leq \int_{t_0}^{t} |f_{k+1}^l - f_k^l|(s)\, \mathrm{d}s \leq \int_{t_0}^{t} C_f C_*^k P_k(s)\, \mathrm{d}s$$

and subsequently, there is the bound

$$|u_{k+2}^l - u_{k+1}^l|(t) \leq C_f C_*^k \left(M_\Psi \frac{(t-t_0)^{k+1}}{k+1!} + M_f \frac{(t-t_0)^{k+2}}{k+2!} \right) = C_f C_*^k P_{k+1}(t) \tag{7}$$

for each l as well.

A similar bound can be shown to hold for the difference between the delayed iterates. To this end, denote $t_* = \min\{\Delta_{u_{k+2}}(t), \Delta_{u_{k+1}}(t)\}$ and t^* the maximum. It then follows that

$$\left| u_{k+2}^l(\Delta_{u_{k+2}}(t)) - u_{k+1}^l(\Delta_{u_{k+1}}(t)) \right| \leq \left| \int_{t_0}^{\Delta_{u_{k+2}}(t)} f_{k+1}^l(s)\, \mathrm{d}s - \int_{t_0}^{\Delta_{u_{k+1}}(t)} f_k^l(s)\, \mathrm{d}s \right|$$
$$\leq \int_{t_0}^{t_*} \left| f_{k+1}^l(s) - f_k^l(s) \right| \mathrm{d}s + \int_{t_*}^{t^*} \left| f_*^l(s) \right| \mathrm{d}s \tag{8}$$

where $f_*^l = f_k^l$ or f_{k+1}^l depending on which iterate produces the larger Δ_u. In either case, the last integral may be bound as

$$\int_{t_*}^{t^*} |f_*^l(s)|\, \mathrm{d}s \leq M_f \left| \Delta_{u_{k+2}}(t) - \Delta_{u_{k+1}}(t) \right|$$
$$\leq M_f C_\Delta |u_{k+2}(t) - u_{k+1}(t)| \tag{9}$$

and using (6), (9) along with the induction argument, (8) then becomes

$$|u_{k+2}^l(\Delta_{u_{k+2}}(t)) - u_{k+1}^l(\Delta_{u_{k+1}}(t))| \leq (C_f C_*^k + M_f C_\Delta C_f C_*^k) P_{k+1}(t) = C_*^{k+1} P_{k+1}(t) \tag{10}$$

since $t^* < t$ and $P_k(t)$ is strictly increasing for $t > 0$. Since $C_f < C_*$ implies $C_f C_*^k < C_*^{k+1}$, then $E_{k+1}(t) \leq C_*^{k+1} P_{k+1}(t)$. And thus, $E_k(t) \leq C_*^k P_k(t)$ holds for all $k \geq 0$.

For each $1 \leq l \leq L$, it follows that $\sum_k |u_{k+1}^l - u_k^l|(t) \leq \sum_k C_*^k P_k(T)$ over the compact interval $[t_0, T]$. Since $\sum_k C_*^k P_k(T) \leq \max\{M_\Psi, M_f\} e^{2C_*(T-t_0)}$, it follows from the Weierstrass M-test that for each $1 \leq l \leq L$, the sequence $u_k^l(t)$ converges uniformly over $[t_0, T]$

to a continuous limit. This limit solves (2), since (2) and (4) are equivalent. In turn, $\Delta_{u_k}(t)$ converges as well since Δ_u is continuous wrt u. □

4 Polynomial Vector Fields for Delay Differential Equations

In this section, the original DDE (1) will undergo a series of changes of variable. The PSMd method itself only relies on the final system, after all changes have been enacted. The first change is the standard change to achieve autonomy, the second change, given by (14), is the crucial one for PSMd: it removes the delay from the original problem, and it moves the delay to the coefficient of the current vector field along with into its own subsystem. The final change is just to couple (14) and the delay subsystem after the delay subsystem has been developed.

The second change is based on the fact that the delay maps each interval from MoS into the previous interval. If the delay is composed with itself, then this can map any MoS interval back to the first one. So a change of variable involving compositions of the delay allows every MoS problem to be run over the same interval (or 'local clock'), mainly $(0, \Delta^{-1}(0))$. The benefits to linking each MoS problem to the same clock are seen in this section.

State independent delays are considered first, so the delay notation is shortened to $\Delta(t)$. Anticipating the ideas of PSM, a standard change of variable is invoked which shifts the initial time to the origin and makes the vector field autonomous. To this end, define $\underline{t} \equiv t - t_0$ and $\underline{\mathbf{u}}(\underline{t}) \equiv \mathbf{u}(\underline{t} + t_0) = \mathbf{u}(t)$.

In addition, append any auxiliary variables needed to make the vector field autonomous to $\underline{\mathbf{u}}$ as well as appending their evolution equations to $\mathbf{f}$. The time, the new variable list, the delay and the new vector field are relabeled as t, $\mathbf{u}$, $\Delta(t)$ and $\mathbf{f}$ since recovery of the original solution is simply to shift the first component of $\mathbf{u}$ by t_0.

Due to the derivative being invariant with respect to a time shift, (2) then becomes under these changes

$$\begin{cases} \dot{\mathbf{u}}(t) = \mathbf{f}\left(\mathbf{u}(t), \mathbf{u}\left(\Delta(t)\right)\right), \ t > 0 \\ \\ \mathbf{u}(t) = \boldsymbol{\Psi}(t + t_0), \qquad\qquad t \in [a - t_0, 0] \end{cases} \tag{11}$$

where the vector field is now autonomous and the initial time is 0. If $t_0 \neq 0$, then some computational effort must be exerted to put $\Psi(t + t_0)$ into standard polynomial form wrt t. For convenience, assume $t_0 = 0$ in the sequel.

Prior to examining the auxiliary variables for (11), MoS is reviewed in the next section. This process partitions the solution's domain into subintervals based on the given delay. MoS provides a convenient framework for discussing both propagation of jumps along with computational aspects. After that, auxiliary variables will be addressed in Sects. 4.2, 4.3 and 4.4 .

4.1 Method of Steps and its Failure

A state independent deviating argument $\Delta(t)$ will be called a *simple delay* if Δ is continuous, monotonically increasing and $\Delta(t) < t$ for all t in its domain. In particular, if $\Delta(t)$ is a simple delay then its inverse $\Delta^{-1}(t)$ exists. Define the *lag* via $\tau(t) \equiv t - \Delta(t) > 0$.

MoS is well defined in the case of a simple delay, and it uses the given delay to partition the domain into subintervals, over which the DDEs can be cast into an ODE form, which

gives ODE methods traction to be utilized. The delay maps each subinterval (except the first) into the preceding one.

Recalling the assumption that $t_0 = 0$, first consider the case $t_1 = \infty$, i.e. the solution's domain is $(0, \infty)$. Define $\tau_0 \equiv 0$ as well as the sets

$$\mathscr{T}_\Delta \equiv \{\tau_{m-1} \mid \Delta(\tau_m) = \tau_{m-1}, m \in \mathbb{Z}_0^+\}$$

and, after defining $I_m \equiv (\tau_m, \tau_{m+1})$,

$$\mathscr{I}_\Delta \equiv \{I_{m-1} \mid m \in \mathbb{Z}_0^+\}$$

which are well defined for any simple delay. Note that elements of $\mathscr{I}_\Delta$ are also the intervals over which the history does not change: over I_0, Ψ is the history, over I_1, the solution over (τ_0, τ_1) is the history etc.

If $t_1 < \infty$, then there are two cases to consider: there exists an $m^* \in \mathbb{Z}^+$ such that $\tau_{m^*} = t_1$, i.e. the right edge of the domain coincides with the right edge of an interval for MoS. If the solution's domain falls in the middle of an interval from MoS, i.e. there exists an $m^* \in \mathbb{Z}^+$ such that $\tau_{m^*-1} < t_1 < \tau_{m^*}$, then set $\tau_{m^*} = t_1$. In either of these cases, m would be an element of $\{0, 1, \ldots, m^*\}$ rather than $\mathbb{Z}_0^+$.

If any of $\mathbf{f}$, $\boldsymbol{\Psi}$ or Δ_u are not smooth, then the first occurrence of a jump would also be used to seed a set like $\mathscr{T}_\Delta$ and all of these sets should be unioned together to act as $\mathscr{T}_\Delta$ as the analysis for termination or loss of uniqueness needs to be checked at each of these points. This is considered in a little more detail in Sect. 5.4 for the case of initial history jumps.

It is also possible to present the definition of a solution to (11). As per [11] and the reference therein, $u(t)$ is a solution if $u(t)$ is continuous over (t_0, t_1), continuously differentiable and satisfies (11) over $\mathscr{I}_\Delta$. The information on the right side of a jump should be used to evaluate the vector field, in order to obtain $\dot{u}$ to the right of $\tau_m \in \mathscr{T}_\Delta$.

MoS can fail two ways: if there is a *vanishing lag* at $t^* < t_1$, i.e. $\Delta(t^*) = t^*$, then $\tau_m \to t^*$ but $t^* < t_1$. Another failure mechanism is for Δ to have a turning point so that Δ is not invertible. For a specific situation examined here, minor modifications to the increasing version are possible, such that MoS can be extended to include a change in monotonicity. The failures of traditional MoS are addressed in Sect. 6.8.

Proposition 1 implies a unique solution will exist for (11) with admissible $\mathbf{f}$ at least up to the first unresolved breaking point. Denote this unique solution over elements of $\mathscr{I}_\Delta$ by $\mathbf{u}_m(t) \equiv \mathbf{u}(t)$, $t \in I_m$ with $m \in \mathbb{Z}_0^+$. Define $\mathbf{u}_{m+1}(\tau_{m+1}) = \mathbf{u}_m(\tau_{m+1})$ so that by construction, the piecewise function made up of $\mathbf{u}_m$ is continuous if the individual $\mathbf{u}_m$ are continuous.

It is straightforward to modify the proof of Proposition 1 to handle the case when $\mathbf{u}_m(t)$ is considered known, and (4) is used to compute $\mathbf{u}_{m+1}(t)$, along with using the identification $\mathbf{u}_k(\Delta(s)) = (\mathbf{u}_m)_k(\Delta(s))$ in the vector field. Hence, by induction the piecewise function $\{(\mathbf{u}_m)_k \mid m \geq 0, k \geq 0\}$ computed from (4) approximates the solution to (11).

By definition of τ_m, it follows that $\Delta : I_m \to I_{m-1}$ for each $m \geq 0$. Consider compositions of the delay, denoted

$$\Delta_m(t) \equiv \Delta \circ \cdots \circ \Delta(t) \tag{12}$$

with $\Delta_1(t) = \Delta(t)$ and $\Delta_0(t) \equiv t = t - \tau_0$, and it follows that $\Delta_m : I_m \to I_0$. For the case of a constant lag, there is $\Delta(t) = t - \tau$ for some fixed $\tau > 0$, and $\tau_m = m\tau$ for all $m \geq 0$, while $\Delta_m(t)$ reduces to $t - m\tau$.

4.2 Vector Field Auxiliary Variables and Time Change

There are several issues to address concerning auxiliary variables and delays. In the upcoming example, the case of a constant lag is considered since this delay itself does not require auxiliary variables. The choice of auxiliary variables to convert the vector field into a polynomial one is the exact same as in the ODE case up to the delay terms, which can then be examined carefully. The next two subsections are devoted to the more general case of nonconstant lag.

Example Let $\dot{u}(t) = u(t)\cos(u(t-\tau)) + u(t-\tau)$ with $t > 0$ with some given initial history over $[-\tau, 0]$. The choice of auxiliary variables $v(t) = \cos(u(t-\tau))$ and $w(t) = \sin(u(t-\tau))$ would yield the following (neutral) system

$$\begin{cases} \dot{u}(t) = u(t)v(t) + u(t-\tau) \\ \dot{v}(t) = -w(t)\dot{u}(t-\tau) \\ \dot{w}(t) = v(t)\dot{u}(t-\tau) \end{cases} \tag{13}$$

and the question is how to compute the history terms.

One could use the currently generated approximation for u from the system above, which would be polynomial, and evaluate that at $t-\tau$. However, this poses a small problem, since the polynomial with arguments $t-\tau$ must be expanded and while this is theoretically feasible, computationally it is expensive compared to the trace convolutions.

Another possibility is to use more auxiliary variables, and then try to compute $U(t) = u(t-\tau)$ by finding a vector field for U's evolution and applying Picard iteration. However, this causes the system to not close, since this simply passes the issue to the next derivative. Specifically, $\dot{U} = \dot{u}(t-\tau)$ and so now an auxiliary variable is needed for $\dot{u}(t-\tau) = V(t)$, whose evolution requires knowledge of $\ddot{u}(t-\tau) = \dot{V}(t)$, and so on. Note that the presence of a nonconstant delay would compound the issues seen in both approaches mentioned so far.

Changing to a local time in each $I_m \in \mathscr{I}_\Delta$ manages to remove the computational issue entirely for history terms that appear in the vector field, so these details are now addressed. Bellman, Buell and Kalaba had proposed a method in [14] based on MoS. The change of variable there led to a system whose size increased as time increased.

Instead, consider the change of variable for time: for $t \in I_m$ with fixed $m \geq 1$, define $\underline{t} \equiv \Delta_m(t)$ so that $\underline{t} \in I_0$ for every $m \geq 1$. Now $\underline{t}$ becomes the local variable for each $I_m \in \mathscr{I}_\Delta$. Noting that

$$(\Delta^{-1})_m \equiv \Delta^{-1} \circ \Delta^{-1} \circ \cdots \circ \Delta^{-1} = (\Delta_m)^{-1}$$

implies Δ_m^{-1} is unambiguous, define

$$\mathbf{U}_m(\underline{t}) \equiv \mathbf{u}(\Delta_m^{-1}(\underline{t})) = \mathbf{u}_m(t)$$

Now (11) can be used when $m = 0$ and then, for each $m \geq 1$ such that $\tau_m < T$, replace (11) with

$$\begin{cases} \dot{\mathbf{U}}_m(\underline{t}) = \mathbf{f}_\Delta\left(\mathbf{U}_m(\underline{t}), \mathbf{U}_{m-1}(\underline{t})\right), \quad \underline{t} \in (\tau_0, \tau_1) \\ \mathbf{U}_m(\tau_0) = \mathbf{U}_{m-1}(\tau_1) \end{cases} \tag{14}$$

where $\mathbf{f}_\Delta = (\Delta_m^{-1})'(\underline{t})\mathbf{f}$, so that the new vector field is only a scalar multiple of the original vector field. Attention is called to the fact that u's clock is t (global time) and U and its

variants' clocks are $\underline{t}$ (local time), and that the prime notation looks better for Δ_m^{-1} than the dot notation.

Note that the system for Δ_u^{-1}, to be introduced in the next subsection, is a system of ODEs. In the original DDE, if $\Delta_u(t)$ were taken to be identically t, suppose the resulting vector field produces a *projectively polynomial* solution. If so, then the original DDE has a projectively polynomial solution as well. This shows that the choice for auxiliary variables needed to make the vector field polynomial is based on Δ_u, in particular Δ_m^{-1}, and the vector field but not on delayed terms $u(\Delta_u(t))$, $u'(\Delta_u(t))$ etc. themselves.

In particular, if the lag is constant, then $(\Delta_m^{-1})' = 1$ and $\mathbf{f}_\Delta = \mathbf{f}$. Note that the open interval implies the differential equation does not apply at τ_1 due to jump discontinuities. However, the solution is continuous and so U_m may be evaluated at $t = \tau_1$ to define $U_m(\tau_1)$.

Two items to address concerning (14) are discussed in both this and the next subsections. The items are the computational advantages of this particular form of the problem and the computation of $(\Delta_m^{-1})'$ along with other Δ related information.

With regards to efficiency, the important part of (14), especially for the computation, is that the $\mathbf{u}_{m-1}(\Delta(t))$ term in the vector field [from (2) when $t \in I_m$] appears to lose its delayed argument in (14). As such, (14) is amenable to the integration algorithm from Sect. 2.

The auxiliary variables' ability to remove the deviating argument is now examined. The apparent loss of the deviating argument is achieved by noting that t and $t^* = \Delta(t)$ each have their own local variable, and these local variables would be independent of each other. For each $t \in I_m$, and the associated $t^* = \Delta(t) \in I_{m-1}$, denote the local time $\underline{t}$ for time t and the local time $\underline{t}^*$ for t^*. Now consider for $t \in I_m$

$$\underline{t} = \Delta_m(t) = \Delta_{m-1}(\Delta(t)) = \Delta_{m-1}(t^*) = \underline{t}^*$$

and so t and $\Delta(t)$ have local times that are equal.

This formulation of putting each subinterval in $\mathscr{I}_\Delta$ on the same clock allows for rapid computation of $\mathbf{U}_m$ since $\mathbf{U}_{m-1}$ is used in its current form as the delay term in the vector field for $\mathbf{U}_m$.

Returning to the example, for $t \in I_m$ and $V_0 = \Psi'$, (13) would become

$$\begin{cases} \dot{u}_m = u_m v_m + u_{m-1} \equiv V_m \\ \dot{v}_m = -w_m V_{m-1} \\ \dot{w}_m = v_m V_{m-1} \end{cases} \tag{15}$$

i.e. the vector field computation for u'_m is retained for use during integration in the next subinterval I_{m+1}. The impact of state independent nonlinear delays on auxiliary variables is thus reduced to the auxiliary variables needed for $(\Delta_m^{-1})'(\underline{t})$.

4.3 Delay Auxiliary Variables: State Independent

Inspection of (14) shows that knowledge of $\Delta_m^{-1}(t)$ is needed for two reasons: to determine $\tau_m \in \mathscr{T}$, and also to build the scalar multiplier for the vector field in (14), $(\Delta_m^{-1})'$. While there are several ways to enact these ideas, computation of Δ_m^{-1} only is pursued here, since it is capable of rectifying both issues: a derivative would yield the scalar multiplier, and it also follows that for any $m \geq 1$, $\tau_m = \Delta_m^{-1}(\tau_0)$. PSMd uses $\Delta^{-1}(t)$ to compute τ_m without utilizing root finding techniques.

Attention is now placed on getting $(\Delta_m^{-1})'$ into a polynomial form that can be computed efficiently. If $\Delta_m^{-1}(\underline{t})$ is considered as an unknown function, and if a vector field for $(\Delta_m^{-1})'(\underline{t})$ can be found in terms of this unknown, then assuming the delay is projectively polynomial, PSM can be applied directly to this problem since the delay function is now the dependent variable, not a deviating argument. Thus, the Maclaurin expansion for $\Delta_m^{-1}(\underline{t})$ can be computed.

To this end, re-using the identity $\Delta_m'(t) = \Delta'(\Delta_{m-1}(t)) \cdot \Delta_{m-1}'(t)$ on the equation $(\Delta_m^{-1})'(\underline{t}) = \left((\Delta_m)'(\Delta_m^{-1}(\underline{t}))\right)^{-1}$ and the change of variable to local time, there is

$$\begin{aligned}(\Delta_m^{-1})'(\underline{t}) &= \left(\Delta'(\Delta_{m-1} \circ \Delta_m^{-1}(\underline{t})) \cdot \Delta_{m-1}'(\Delta_m^{-1}(\underline{t}))\right)^{-1} \\ &= \left(\Delta'(\Delta^{-1}(\underline{t})) \cdot \Delta'(\Delta_2^{-1}(\underline{t})) \cdots \Delta'(\Delta_m^{-1}(\underline{t}))\right)^{-1} \end{aligned} \tag{16}$$

and upon defining $W_j(\underline{t}) \equiv \Delta_j^{-1}(\underline{t})$ for $j \geq 1$ with W_m the current variable, there is

$$\dot{W}_m = \left(\prod_1^m \Delta'(W_i)\right)^{-1} = \left(\prod_1^{m-1} \Delta'(W_i)\right)^{-1} \cdot (\Delta'(W_m))^{-1} \tag{17}$$

and since the terms from $j = 1$ to $m-1$ have been computed before $t \in I_m$, their product is relabeled as $p_{m-1}(\underline{t})$.

Hence, (17) can be written $\dot{W}_m = p_{m-1}(\underline{t})\left(\Delta'(W_m)\right)^{-1} \equiv g(W_m)$, with $p_0(t) = 1$, so that $\dot{W}_1 = \left(\Delta'(W_1)\right)^{-1}$ allows W_m to be found from a dynamic programming approach. Note that the size of the system to be solved to determine Δ_m^{-1} does not change as m increases, unlike in [14].

Example [14] Let $\Delta(t) = t - 1 - e^{-t}$. Then using $(\Delta^{-1})'(t) = (\Delta'(\Delta^{-1}(t)))^{-1}$ and denoting $W(t) = \Delta^{-1}(t)$, there is $\dot{W} = (1+e^{-W})^{-1}$. This ODE for W can be expanded to a quadratic vector field using $X(t) = (1+e^{-W(t)})^{-1}$ and $Y(t) = X^2(t)$.

Choosing the DDE was $\dot{u}(t) = -u(t)u(\Delta(t))$, then for I_m with any $m \geq 0$, the vector field for the system would be, recalling $\tau_0 = 0$,

$$\begin{cases} \dot{U}_m = -p_{m-1}X_m U_m U_{m-1} & U_m(0) = U_{m-1}(\tau_1) \\ \dot{W}_m = p_{m-1}X_m & W_m(0) = W_{m-1}(\tau_1) \\ \dot{X}_m = Y_m(1 - X_m) & X_m(0) = \left(1 + e^{-W_m(0)}\right)^{-1} \\ \dot{Y}_m = 2Y_m(X_m - Y_m) & Y_m(0) = X_m^2(0) \end{cases} \tag{18}$$

where $p_{m-1} = \prod_{i=1}^{m-1} X_i$ or $p_{m-1} = X_{m-1}p_{m-2}$ with $p_0 = 1$. Denoting $\mathbf{W} = [W, X, Y]$, then in (τ_m, τ_{m+1}), there is $\dot{\mathbf{W}}_m = \mathbf{g}(\mathbf{W}_m)$, with $\mathbf{g}$ given by the right hand sides of the second, third and fourth lines of (18).

In general, the right hand side of the ODE for the delay (the W_m equation above) also appears as a multiplier in each equation from the original DDE. Working backwards from (18), the term $p_{m-1}X_m$ is both a right hand side (2nd equation) and a multiplier (1st equation). Note that p_{m-1} does not appear in the 3rd and 4th equations. Hence, 3rd and 4th equation are in the delay subsystem, along with the 2nd. So, only the 1st equation came from the DDE, hence the original DDE was scalar. Ignoring the $p_{m-1}X_m$ multiplier in the first equation, the

remaining terms are $U_m U_{m-1}$, which is the original vector field, with the $m-1$ indicating the delayed term.

For the case $m=0$, the scalar multiplier $(\Delta_0^{-1})' = 1$, and the systems uncouple. The term $p_{m-1}X_m$ is replaced with 1 in the equation for $\dot{U}$ and replaced with X_0 in the equation for $\dot{W}_0$. In addition, the proper time interval for the delay problem, the subsystem involving W_0, X_0 and Y_0, is (τ_{-1}, τ_0) rather than (τ_0, τ_1). In particular, the initial data for $W_0(\tau_{-1})$ is zero, since $\Delta(\tau_1) = \tau_0 \implies \Delta^{-1}(\tau_1) = \tau_0 = 0$.
Let the solution to (14) be relabeled as $\underline{\mathbf{U}}_m$ rather than $\mathbf{U}_m$. Appending Δ_m^{-1} to the end of the variable list in $\mathbf{u}$ from (14), along with any necessary auxiliary variables, denote this new list as $\mathbf{U}_m = [\underline{\mathbf{U}}_m, \mathbf{W}_m]$. Appending (17) to $\mathbf{f}_\Delta$, along with the ODEs for the auxiliary variables yields the following system for $m \geq 1$

$$\begin{cases} \dot{\mathbf{U}}_m(\underline{t}) = \mathbf{f}_W\left(\mathbf{U}_m(\underline{t}), \mathbf{U}_{m-1}(\underline{t})\right), \quad \underline{t} \in (\tau_0, \tau_1) \\ \\ \mathbf{U}_m(\tau_0) = \mathbf{U}_{m-1}(\tau_1) \end{cases} \tag{19}$$

with $\mathbf{f}_W = [\dot{W}_m \mathbf{f}, \mathbf{g}]$.

The result of the Δ_0^{-1} subsystem is used to find the value of τ_1, which is necessary to setup (19) when $m \geq 1$ from $\tau_1 = W_0(\tau_0)$. It would be convenient to integrate the delay subsystem first, get τ_1, then integrate the solution subsystem, since τ_1 would be known to transfer the information. Note that either system could be integrated first, but the transfer of $\mathbf{U}_0$'s information can not occur until τ_1 is known. However, if the delay were state dependent, it would be important to be able to integrate the solution subsystem before having to integrate the delay subsystem.

4.4 State Dependent Delays

A single state dependent delay is now considered and for presentation purposes with a delay whose state dependency preserves the properties of a simple delay, so that MoS does not fail. More specifically, for a fixed state u from some admissible set of functions, $\Delta_u(t) < t$ for all $t \geq 0$ and $\Delta_u(t)$ is increasing wrt t.

It remains to determine the scalar multiplier. For clarity, shorten the notation for the delay to $\delta(t) \equiv \Delta_u(t)$. Define $\delta_m(t) \equiv \delta \circ \cdots \circ \delta(t)$ for $m \geq 1$, and also define $\underline{t}_m \equiv \delta_m(t)$. Exchange D in favor of a prime.

Let $\Delta^{-1}(p, q)$ be such that $\Delta^{-1}(\delta(t), u(\delta(t))) = t$ and promptly shorten $\Delta^{-1}(\underline{t}, u(\underline{t}))$ to $\delta^{-1}(\underline{t})$. In addition, define $\delta_m^{-1}(\underline{t})$ as the composition $\delta^{-1} \circ \cdots \circ \delta^{-1}(\underline{t})$. Then the functional inverse of δ_m is δ_m^{-1}. Hence the state dependent version of the scalar multiplier from (19), for fixed u can be written $(\delta_m^{-1})' = (\delta_m' \circ \delta_m^{-1})^{-1}$. Suppressing $\underline{t}$ dependence, the analog of (4.6) can be written

$$(\delta_m^{-1})'(\underline{t}) = \prod_{i=1}^{m} \left(\frac{\partial \Delta}{\partial p}\left(\delta_i^{-1}(\underline{t}), u(\delta_i^{-1}(\underline{t}))\right) + \frac{\partial \Delta}{\partial q}\left(\delta_i^{-1}(\underline{t}), u(\delta_i^{-1}(\underline{t}))\right) \dot{u}(\delta_i^{-1}(\underline{t})) \right)^{-1}$$

and upon relabeling $W_i(\underline{t}) = \delta_i^{-1}(\underline{t})$ and considering the state dependent version of (19), one can write more compactly

$$\dot{W}_m = \left(\prod_{i=1}^{m} \frac{\partial \Delta}{\partial p}(W_i, U_i) + \frac{\partial \Delta}{\partial q}(W_i, U_i)\, \dot{U}_{i*} \right)^{-1}$$

$$= p_{m-1}\left(\frac{\partial \Delta}{\partial p}(W_m, U_m) + \frac{\partial \Delta}{\partial q}(W_m, U_m)\,\dot{U}_{m*}\right)^{-1} \tag{20}$$

with $p_{m-1} \equiv \left(\prod_{i=1}^{m-1} \frac{\partial \Delta}{\partial p}(W_i, U_i) + \frac{\partial \Delta}{\partial q}(W_i, U_i)\,\dot{U}_{i*}\right)^{-1}$ and the asterisk in U_{i*} indicates the derivative is on U only with no $(\delta_m^{-1})'$ term, in other words it is the derivative delayed, not the derivative of the delay term.

To illustrate this in more detail, the problem of Bellman et al. is extended to a state dependent case by taking $\delta(t) = t - 1 - e^{-u(t)}$ and it follows that this $\delta(t)$ is a simple delay if $\dot{u} > -e^u$, since $\frac{\mathrm{d}}{\mathrm{d}t}\Delta_u = 1 + e^{-u}\dot{u} > 0$. Using $\frac{\partial \Delta}{\partial p} = 1$ and $\frac{\partial \Delta}{\partial q} = e^{-u(t)}$, (20) would become

$$\dot{W}_m = p_{m-1}\left(1 + e^{-U_m}\dot{U}_{m*}\right)^{-1}$$

with $p_{m-1} = \prod_{i=1}^{m-1}\left(1 + e^{-U_i}\dot{U}_{i*}\right)^{-1}$ and this can be converted to a polynomial vector field. In particular, assuming the same DDE as before $\dot{u} = -uu_*$, the first two equations are the same as the first two in (18), the remaining would be, for $m \geq 1$

$$\begin{cases} \dot{X}_m = -Z_m X_m \Phi_m & X_m(0) = \left(1 + e^{-U_{m-1}(\tau_1)}\dot{U}_{m-1}(\tau_1)\right)^{-1} \\ \dot{Y}_m = -p_m Y_m U_m U_{m-1} & Y_m(0) = e^{-U_{m-1}(\tau_1)} \\ \dot{Z}_m = -Z_m^2 \Phi_m + p_m Z_m U_m U_{m-1} & Z_m(0) = X_{m-1}(\tau_1) Y_{m-1}(\tau_1) \end{cases} \tag{21}$$

for $X(t) = (1 + e^{-U(t)}\dot{U}(t))^{-1}$, $Y(t) = e^{-U(t)}$ and $Z(t) = X(t)Y(t)$, along with $p_m = p_{m-1}X_m$ and $\Phi_m = \ddot{U}_m - (\dot{U}_m)^2$. from this, one can see that the state dependency forces a slightly different set of auxiliary variables than was seen in the state independent version.

If $m = 0$, then $\dot{U}_0 = -U_0 U_{-1} = -U_0 \Psi$ with $U_0(0) = \Psi(0)$ and $\dot{W}_0 = X_0 \Phi_0$. The initial data should be: $W_0(0) = 0$, $X_0(0) = (1 + e^{-U_0(0)}\dot{U}_0(0))^{-1}$, $Y_0(0) = e^{-U_0(0)}$, and $Z_0(0) = X_0(0)Y_0(0)$.

4.5 Summary

The system in (19) achieves having a (multivariate) polynomial vector field in terms of components which are all evaluated at the same time (local clock). It also has a subsystem

Fig. 1 A product in the vector field as an array

V_3t^3	V_2t^2	V_1t	V_0	
$V_3U_0t^3$	$\underline{V_2U_0t^2}$	$\underline{V_1U_0t}$	$\underline{V_0U_0}$	U_0
	$V_2U_1t^3$	$\underline{V_1U_1t^2}$	$\underline{V_0U_1t}$	U_1t
		$V_1U_2t^3$	$\underline{V_0U_2t^2}$	U_2t^2
			$V_0U_3t^3$	U_3t^3

specific to the delay in the original problem. The degree of the vector field polynomials are all two. Each component is itself a standard polynomial in variable t whose degree grows with further Picard iterations.

With polynomials, Picard iteration becomes very easy to implement: the vector field polynomial and integration simply becomes a convolution of two expansion polynomials' coefficients (in t). The next coefficient in the Maclaurin expansion for a particular component is achieved by simply computing some traces (see Fig. 1) and then possibly summing some of those together (integration algorithm). This approach applied to (19) [or with (20)] works for a single simple delay.

5 PSMd

Having accomplished the task of making (1) become (19) so that it is amenable to the integration algorithm, the Picard formula may be put into an algorithm. After presenting the algorithm, an error analysis, which amounts to checking stability in error propagation by the vector field, is also considered. Major variants of the approach are then examined, which include applying the method to a subdivided $I_m \in \mathscr{I}_\Delta$ and both decreasing (which are handled by modifying MoS) and distributed delays.

5.1 Algorithm

Since both function evaluations in the vector field are not delayed, (19) has the appearance of an ODE system, and the integration algorithm from Sect. 2 may be used directly over (τ_0, τ_1). Note that the solution to (11) can be recovered from the solution to (19) by equating the value of $\mathbf{U}_m$ at local time $\underline{t}$ with the value of $\mathbf{u}$ at the (computed) global time $\Delta_m^{-1}(\underline{t})$. The solution to (2) is recovered from that by a subsequent time shift of t_0, and the first component of this result solves (1).

Assuming for presentation convenience that the number of Picard iterations to be performed remains constant in each I_m, and call that number k^*. Note that it is straightforward to allow k^* to be m dependent, i.e. using a different number of Picard iterations in each I_m (or even subdivisions).

The kth iteration of the *PSMd* approximation is denoted $\mathbf{U}_m^k(\underline{t})$, and is computed via Picard as

$$\mathbf{U}_m^{k+1}(\underline{t}) = \mathbf{U}_m^k(\tau_0) + \int_{\tau_0}^{\underline{t}} \mathbf{f}_W\left(\mathbf{U}_m^k(s), \mathbf{U}_{m-1}^{k^*}(s)\right) \mathrm{d}s \tag{22}$$

for $k \in \mathbb{Z}^+$ using the integration algorithm.

This can be put into algorithm form by denoting the initial history as $\mathbf{U}_{-1} = \mathbf{U}_{-1}^{k^*}$, *PSMd* becomes

1. Convert (1) into (19), which includes (17), via the necessary auxiliary variables.
2. Use $\mathbf{U}_{m-1}^{k^*}$ as initial history.
3. Run the integration algorithm on (19) over (τ_0, τ_1) to generate $\mathbf{U}_m^k$, $k = 1, \ldots, k^*$.
4. $m \mapsto m + 1$ and repeat steps 2-4 until $m = m_{\text{stop}}$.

Constructing the piecewise function from $\mathbf{U}_m$ would produce the computed approximation to the solution of (19) by PSMd.

5.2 Error Analysis

There is a price to pay for the efficiency in PSMd versus the version in [6]. When a finite number of Picard iterations is used in practice, in particular, for $\mathbf{U}_0$ with $t \in I_0$, this error propagates to I_1 when the vector field receives $\mathbf{U}_1^* = \mathbf{U}_0$ as the history. The evolution of the magnitude of this error as m increases is of interest, i.e. the stability of the vector field with respect to history errors.

A sufficient condition for stability will be given for a subclass of problems, and this can be extended to larger subclasses by considering specific vector fields. Recalling Proposition 1 notation, define $C_i = \sum_{l=1}^{L} C_i^l$ with $i = 1, 2$, so that $C_1 + C_2 = C_f$. Now in the face of stability wrt history errors, a condition is put on one of the Lipschitz constants.

Proposition 2 *For a given* $\mathbf{f}$ *and* Δ *from (2), suppose* $C_1\tau_1 < 1$. *Then, for a given* $m^* > 0$ *and* $\epsilon > 0$, *there exists* $\kappa \in \mathbb{Z}^+$ *such that if* $k_m^* = \kappa$ *for every* $m \le m^*$, *then the error in the approximation from PSMd to the solution of (19) is bounded by* ϵ, *i.e. for any* $1 \le k \le k_m^*$

$$\left|(U_m^k)^l - (U_m)^l\right|(\underline{t}) \le \epsilon$$

Proof Consider an idealized approach: apply (22) to (19) and let $k_m^* \to \infty$ for each $m < m^*$, while leaving k finite for $m = m^*$, so that the only error incurred is the error in I_m. Denote this approximation by $\mathbf{V}_m^k(\underline{t})$ over I_m.

Now consider the error between the non-idealized (i.e. k_m^* finite for all m) approximation and the solution to (19). Denoting $(U_m^k)^l$, $(V_m^k)^l$ as the components of $\mathbf{U}_m^k$, $\mathbf{V}_m^k$, respectively, the following inequality holds

$$\left|(U_m^k)^l - (U_m)^l\right|(\underline{t}) \le \left|(U_m^k)^l - (V_m^k)^l\right|(\underline{t}) + \left|(V_m^k)^l - (U_m)^l\right|(\underline{t})$$

for each component $l = 1, \ldots, L$. The second term on the right hand side of the inequality is the PSM error, assuming the initial data is error free. This particular error is quantified in [15] and it converges to zero with increasing k.

Concerning the first term, define the error at each grid point by

$$E_m^k \equiv \max_l \max_{[\tau_0,\tau_1]} |(U_m^k)^l - (V_m^k)^l|(\underline{t})$$

Using (22) with the pair $k, k-1$ rather than $k+1, k$ along with $\mathbf{U}_m, \mathbf{V}_m$, the triangle inequality and Lipschitz condition, there is

$$E_m^k \le E_{m-1}^\kappa(1 + C_2\tau_1) + C_1\tau_1 E_m^{k-1} \tag{23}$$

Noting that κ is fixed wrt k, there is

$$\begin{aligned} E_m^k &\le E_{m-1}^\kappa(1 + C_2\tau_1) + C_1\tau_1\left(E_{m-1}^\kappa(1 + C_2\tau_1) + C_1\tau_1 E_m^{k-2}\right) \\ &\le \sum_{i=0}^{k-1}(1 + C_2\tau_1)(C_1\tau_1)^i E_{m-1}^\kappa + (C_1\tau_1)^k E_m^0 \end{aligned} \tag{24}$$

By construction,

$$E_m^0 = \max_l \max_{[\tau_0,\tau_1]} \left|(U_m^0)^l - (V_m^0)^l\right|(\underline{t}) = \left|(U_{m-1}^\kappa)^l - (V_{m-1}^\kappa)^l\right|(\tau_m) \le E_{m-1}^\kappa$$

so that when $k = \kappa$, the inequality may be updated to

$$E_m^\kappa \le (1 + C_2\tau_1)\left(1 + \sum_{i=1}^{\kappa}(C_1\tau_1)^i\right)E_{m-1}^\kappa$$
$$\le (1 + C_2\tau_1)^m\left(1 + \sum_{i=1}^{\kappa}(C_1\tau_1)^i\right)^m E_0^\kappa \equiv M_\kappa^{f\tau}E_0^\kappa$$

when the inequality is iterated with respect to m, with E_0^κ being the PSM error in I_0.

The assumption $C_1\tau_1 < 1$ implies that $M_\kappa^{f\tau} < \left(\frac{1+C_2\tau_1}{1-C_1\tau_1}\right)^m$, which is independent of κ, so there exists a κ big enough so that $E_0^\kappa < \left(\frac{1-C_1\tau_1}{1+C_2\tau}\right)^m \epsilon$, which implies $E_m^\kappa \le \epsilon$ for any $\epsilon > 0$ and for any finite m, i.e. $m \le m^*$. □

Remarks If $m \to \infty$, then these bounds are too coarse to provide any useful information. In particular, the overly coarse assumption of using global values C_1 and τ_1 in (23) rather than local information is required because of the general nature of the vector field. For a specific vector field, sharper approximations to these multipliers may be possible in different I_m, thereby weakening the restriction imposed by $C_1\tau_1 < 1$.

5.3 Segments

The idea of resetting a PSM computation is now extended to PSMd. There are a few reasons to consider subdividing elements of $\mathscr{I}_\Delta$ over which PSMd computes its approximation. The differential equation may be stiff, or there may be a nonhomogeneous (forcing) term present with a support that is smaller than the lag, see Experiment 6 on page 29, or the initial history may have a discrete number of jump discontinuities. The constant lag case is considered in this subsection, since the notation and concepts are easier to present. In the next subsection, a nonlinear delay is considered.

Assume a constant lag, and subdivide I_0 in (14) into N *segments* of equal length δ, so that $N\delta = \tau_1$. Let $i = 0, \ldots, N$. In segment $(i\delta, (i+1)\delta) = (i, i+1)\delta$ of I_m, let $\mathbf{U}_{m,i}(\bar{t}) \equiv \mathbf{U}_m(\bar{t})$ for $\bar{t} \in (i, i+1)\delta$. Then (14) becomes

$$\begin{cases} \dot{\mathbf{U}}_{m,i}(\bar{t}) = \mathbf{f}_\Delta\left(\mathbf{U}_{m,i}(\bar{t}), \mathbf{U}_{m-1,i}(\bar{t})\right), \quad \bar{t} \in (i, i+1)\delta \\ \mathbf{U}_{m,i}(i\delta) = \mathbf{U}_{m,i-1}(i\delta) \end{cases}$$

To be able to use the integration algorithm, one more change of variable must be invoked after the first segment: resetting the time in each segment to begin at zero. Let $\underline{t} = \bar{t} - i\delta$ with $\underline{\mathbf{U}}_{m,i}(\underline{t}) = \mathbf{U}_{m,i}(\bar{t})$, which is independent of m. Hence, the previous IVP becomes

$$\begin{cases} \dot{\underline{\mathbf{U}}}_{m,i}(\underline{t}) = \mathbf{f}_\Delta\left(\underline{\mathbf{U}}_{m,i}(\underline{t}), \underline{\mathbf{U}}_{m-1,i}(\underline{t})\right), \quad \underline{t} \in (0, \delta) \\ \underline{\mathbf{U}}_{m,i}(0) = \underline{\mathbf{U}}_{m,i-1}(\delta) \end{cases} \tag{25}$$

and the original approximation can be found from $\mathbf{u}_m(t) = \underline{\mathbf{U}}_{m,i}(t - m\tau - i\delta)$. Note that (25) becomes (14) if $N = 1$. Computationally, with N segments, then N different sets of polynomials that make up the approximation from the previous iteration must be stored. For $m = 0$, the initial history must be partitioned into N segments as well. In the case of $N > 1$ segments, replace m with mN in $M_\kappa^{f\tau}$ in Theorem 2.

A small computational benefit can occur for short segment lengths. Whenever the initial history is not polynomial, it must be Taylor expanded, for which an error occurs. A smaller order approximation can be used to achieve the same accuracy, since segments would have a smaller length than I_m does.

5.4 Subdivisions

This approach is now extended to the case when Δ_u is nonlinear. Even if one starts with equal sized subintervals in (τ_{-1}, τ_0), a nonlinear Δ_u will map these to subintervals in (τ_0, τ_1) which are not uniform in length, and this will remain the case as m increases. The term *subdivisions* indicates they may be arbitrarily sized, while segments imply equal sized subdivisions.

Consider the case when the initial history has a finite number of isolated jumps. To this end, assume the initial history has been subdivided at discrete locations $\{\tau_{-1}^1, \ldots, \tau_{-1}^N\} \subset (\tau_{-1}, \tau_0)$ with $N \in \mathbb{Z}^+$. These locations can be used to generalize the set $\mathscr{T}_\Delta$. In particular,

$$\mathscr{T}_\Delta\left(\{\tau_{-1}^n\}\right) = \left\{\tau_m^n \mid \Delta(\tau_m^n) = \tau_{m-1}^n, \quad n = 1, \ldots, N, \quad m \in \mathbb{Z}_0^+\right\}$$

assuming the domain of u is $(0, \infty)$. Under this definition, Δ_u maps $(\tau_m^n, \tau_{m+1}^n) \equiv I_m^n$ into I_{m-1}^n for every $n = 1, \ldots, N$ and $m \geq 0$, i.e. Δ_u maps subdivisions of I_m into subdivisions of I_{m-1}. In addition, note that $\tau_m^1 = \tau_m$ as defined earlier.

For the case of a forcing term, the constraint on subdividing comes from $(0, \tau_1)$ rather than $(\tau_{-1}, 0)$, in which case the important aspect with subdividing a PSMd problem when Δ_u is nonlinear, is to make sure the subdivision endpoints are mapped by Δ_u into subdivision endpoints of the initial history.

5.5 Decreasing Delay

PSMd can be modified to work when the delay lacks monotonicity, and MoS is still appropriate (i.e. away from turning points or vanishing lags). Using a specific example that appears in the numerical experiments, consider $\Delta_u(t) = t \sin^2(t)$ over $t \in [0, \pi]$. During the decreasing section, the elements of $\mathscr{T}_\Delta$ are chosen to delay to elements in $\mathscr{T}_\Delta$ associated with the increasing section, but visited in reverse order.

In particular, denote by τ_m^1 elements in the increasing section, while τ_m^2 are in the decreasing section and assume that there are M subintervals from $\mathscr{I}_\Delta$ in the increasing section. As such, the relation between different points in $\mathscr{T}_\Delta$ would need to be updated in the decreasing section to look like $\Delta_u(\tau_{M-m-1}^2) = \tau_m^1$ with $m = 0, \ldots, M$, as opposed to the the relation $\Delta_u(\tau_m^1) = \tau_{m-1}^1$ found in the increasing section.

Concerning the $(\Delta_m^{-1})'$ term, the composition needed to put each subinterval on the same clock will have one term representing the decreasing section, while the remaining terms will come from the increasing section to delay back to (τ_0^1, τ_1^1). Hence the decreasing section is put on the same clock as the increasing section that preceded it. Using superscripts to represent sections, $\Delta_{m-1} \circ \Delta^2(t)$ would delay I_m^2 back to I_0^1: Δ^2 would delay I_m^2 back to I_{M-1-m}^1 while Δ_{m-1} would delay I_{M-1-m}^1 back to I_0^1.

Concerning any history, note that these need local variables which are centered on the right, rather than left, endpoint of the subinterval. So any terms involving the delay that are needed, like u_*, $(\Delta_m^{-1})'$ etc., must be re-expanded around the right endpoint as well as having the independent variable reverse direction. With these modifications, MoS can be used on the decreasing section of the delay.

5.6 Distributed Delays

Given an autonomous ODE $u'(t) = f(u(t))$, this will be extended to a distributed delay problem by the addition of the convolution of the solution at earlier times with a kernel. Consider the family of problems with $\tau > 0$ given arbitrarily

$$\dot{u}(t) = f(u(t)) + \int_0^\tau K(s)g(u(t-s))\,\mathrm{d}s \tag{26}$$

where K is real analytic and g does not inhibit the projectively polynomial property. As before, extending this approach to a system rather than a scalar equation is straightforward.

For convenience, the approach is presented for g composed with u only and not any of its derivatives, along with an exponential kernel $K(s) = \exp(-\alpha s)$ with $\alpha > 0$. Comments concerning a general analytic kernel are made afterward. Examination of the case when $\tau = \infty$ will not be undertaken here.

Introducing these with a given α, τ, and $b > 0$, (26) becomes

$$\dot{u}(t) = f(u(t)) + \int_0^\tau be^{-\alpha s}u(t-s)\,\mathrm{d}s \tag{27}$$

for $t > 0$ and this will be the model scalar problem. The form of the delay implies this problem acts like a constant lag problem: functional forms in the approximation will change across values $\tau_m = m\tau$ and similarly, these are times at which discontinuities in the derivatives may reside. Consider $t \in I_m$. Then the application of Picard iteration produces, for $0 \le t - \tau_m \le \tau$

$$\dot{u}_n(t) = f(u_{n-1}(t)) + \int_0^{t-\tau_m} be^{-\alpha s}u_{n-1}(t-s)\,\mathrm{d}s + \int_{t-\tau_m}^{\tau} be^{-\alpha s}u_{n-1}(t-s)\,\mathrm{d}s$$

and after invoking the change of variables $\underline{t} = t - \tau_m$ and $\underline{u}(\underline{t}) = u(t)$, there is

$$\dot{\underline{u}}_n(\underline{t}) = f(\underline{u}_{n-1}(\underline{t})) + \int_0^{\underline{t}} be^{-\alpha s}\underline{u}_{n-1}(\underline{t}-s)\,\mathrm{d}s + \int_{\underline{t}}^{\tau} be^{-\alpha s}\underline{u}_{n-1}(\underline{t}-s)\,\mathrm{d}s$$

Invoking a change of variable for the first integral, consider $S = \underline{t} - s$ with $\mathrm{d}S = -\mathrm{d}s$. In addition, for $s < \tau$ note that $\underline{u}_{n-1}(\underline{t}-s) = \underline{u}_{\mathrm{old}}(\underline{t}+\tau-s)$, where $\underline{u}_{\mathrm{old}}$ is the approximation computed over I_{m-1}. Invoking a different change of variable in the second integral, let $\underline{S} = \underline{t} + \tau - s$ with $\mathrm{d}\underline{S} = -\mathrm{d}s$. Using the fact that the kernel is exponential, so that the variables separate, there is

$$\dot{\underline{u}}_n(\underline{t}) = f(\underline{u}_{n-1}(\underline{t})) + e^{-\alpha\underline{t}}\int_0^{\underline{t}} be^{\alpha S}\underline{u}_{n-1}(S)\,\mathrm{d}S + e^{-\alpha(\underline{t}+\tau)}\int_{\underline{t}}^{\tau} be^{\alpha\underline{S}}\underline{u}_{\mathrm{old}}(\underline{S})\,\mathrm{d}\underline{S} \tag{28}$$

and this form can easily be computed with the following recipe:

1. Taylor expand all exponentials and include the next power to be computed.
2. Convolve the approximation with the integrand exponentials and then apply the integration algorithm.
3. Convolve the resulting polynomials with the coefficient exponentials and then apply the integration algorithm again.
4. Apply PSM to $\dot{u} = f(u(t))$, and then combine with step 3 and repeat steps 1-4.

Concerning the kernel, the properties of the exponential allow the variables to be separated in a multiplicative fashion: $K(t-s) = K(t)K(-s)$. In the case of a kernel which does not have this property, the general form of (28) is

$$\dot{\underline{u}}_n(\underline{t}) = f(\underline{u}_{n-1}(\underline{t})) + \int_0^{\underline{t}} K(\underline{t}-S)\underline{u}_{n-1}(S)\,\mathrm{d}S + \int_{\underline{t}}^{\tau} K(\underline{t}+\tau-\underline{S})\underline{u}_{\mathrm{old}}(\underline{S})\,\mathrm{d}\underline{S} \quad (29)$$

and now the kernel's Maclaurin expansion needs to accommodate the shifted argument $-\underline{s}+\underline{t}$, which can be dealt with several different ways. One option is to calculate the integration formula for terms of the form $s^i(-s+\tau)^j$ with arbitrary positive i and j and convolve the polynomials normally.

Another possibility is to consider Fourier series, so that an arbitrary kernel could be decomposed into complex exponentials. Yet another possibility is if K is known, then the expansion of $K(\underline{t}+\tau-\underline{S})$ into a polynomial in standard form wrt $\underline{t}$ could be performed at the beginning and would only need to be done once.

6 Numerical Experiments

The purpose of these experiments is to show that sufficient accuracy can be gained for feasible values of the parameters involved. The experiments were run in Octave with a processor speed of 2.4GHz and 8 Gb of RAM available. The experiments below had their numerical parameters, n_1 = number of Picard iterations and n_2 = number of subdivisions, initially chosen as $n_1 = 5$, and $n_2 = 1$ and the resulting errors were then computed.

If the resulting errors were not satisfactory, then two additional runs were performed, one with n_1 increased only, and one with n_2 increased only. If these resulting errors were not satisfactory, then the decision to increase n_1, n_2 (or both) was made by looking at the difference in errors and CPU time from the original run. The final choices for n_1 and n_2 are presented.

A detailed description of each experiment is given in the upcoming subsections. The names for the experiments are

1. Paul—a linear DDE with state dependent delay (comparison)
2. Modified Bellman—state dependent delay
3. Distributed Delay
4. Delayed Kuramoto—parameter dependent asymptotic behavior
5. Circuit problem—a stiff, neutral DDE
6. Blood, Spleen and Tumor model—nonanalytic vector field
7. Castleton and Grimm (CG)—vanishing lags and intervals of decreasing delay

Before these results are presented, a subsection discussing an a posteriori error calculation is introduced. These errors supplement the graphs of the approximations when the correct solution is not known, which is all the experiments except the last, and part of the first. Throughout this final section, the delayed argument will be represented by an asterisk, e.g. $u(\Delta(t)) = u_*(t)$.

6.1 Operator Error

Consider representing a DDE as a vector valued nonlinear operator acting on the solution: $\mathcal{N}\mathbf{U} = \mathbf{0}$. Change k^* from Sect. 5.1 to k and let $\mathbf{V}_k$ be the PSMd approximation using k

Picard iterations. It follows that $\mathscr{N}\mathbf{V}_k = \mathscr{N}(\mathbf{U} + \mathbf{e}_k)$ where $\mathbf{e}_k \equiv \mathbf{V}_k - \mathbf{U}$. Note that $\mathscr{N}\mathbf{V}_k$ can be computed once $\mathbf{V}_k$ has been computed over $I_m \in \mathscr{I}_\Delta$. In particular, it is the vector field without the $(\Delta_m^{-1})'$ multiplier subtracted from the approximation's derivative, but the derivative is over a time interval that reflects global time, not the local time $\underline{t}$.

Since the solution is unique, it follows that $\mathscr{N}\mathbf{V}_k \neq \mathbf{0}$ unless the PSMd approximation is exact. Assuming not, this difference from zero can be used to investigate the magnitude of $\mathbf{e_k}$. To distinguish $\mathbf{e}_k$ from this error in the vector field, $\mathscr{N}\mathbf{V}_k$ shall subsequently be referred to as the *operator error*.

It is easy to show that if $\mathbf{V}_k \overset{u}{\to} \mathbf{V}^*$ for some $\mathbf{V}^*$ and $\mathscr{N}\mathbf{V}_k \overset{u}{\to} \mathbf{0}$, then $\mathbf{V}_k \overset{u}{\to} \mathbf{U}$. Both uniform convergence of $\mathbf{V}_k$ to some limit $\mathbf{V}^*$ along with the operator error to $\mathbf{0}$ can be investigated using PSMd computations for various values of k.

In particular, denoting components of $\mathbf{V}_k$ as $(V_i)_k$, with $i = 0, \ldots, N$, bundle together the norms of the operator error for each component into a vector and define

$$||\mathscr{N}\mathbf{V}_k||_\infty \equiv \left\{\left|\left|(V_i)_k' - f_i\left((V_i)_k, (V_i)_{k*}\right)\right|\right|_\infty\right\}_{i=0}^N$$

where the $||\cdot||_\infty$ inside the brackets is the uniform norm for continuous functions.

This uniform norm is best suited to help indicate uniform convergence of $\mathscr{N}\mathbf{V}_k$ to 0. On the other hand, the one norm version of the operator error may be capable of representing the magnitude of $\mathbf{e}_k$. To see this, denote $t_1^* = \min\{t_1, s_1\}$, where t_1 are s_1 are the right boundaries for the domains of $\mathbf{U}$ and $\mathbf{V}_k$, respectively. Consider

$$\begin{aligned} ||\mathscr{N}\mathbf{V}_k||_1 &\geq \left| \int_0^{t_1^*} \mathbf{V}_k'(s) - \mathbf{f}(\mathbf{V}_k(s), \mathbf{V}_{k*}(s))\, \mathrm{d}s \right| \\ &= \left| \mathbf{V}_k(t_1^*) - \mathbf{V}_k(0) - \int_0^{t_1^*} \mathbf{f}(\mathbf{V}_k(s), \mathbf{V}_{k*}(s))\, \mathrm{d}s \right| \end{aligned}$$

where the integrals, absolute values and inequalities are understood to be applied componentwise. Substituting $\mathbf{V}_k = \mathbf{U} + \mathbf{e}_k$ yields

$$\begin{aligned} ||\mathscr{N}\mathbf{V}_k||_1 \geq \Bigg| \mathbf{e}_k(t_1^*) &- \int_0^{t_1^*} \mathbf{f}_\mathbf{U}(\underline{\mathbf{U}}(s), \underline{\mathbf{U}}_*(s))\mathbf{e}_k(s)\, \mathrm{d}s \\ &- \int_0^{t_1^*} \mathbf{f}_{\mathbf{U}_*}(\underline{\mathbf{U}}(s), \underline{\mathbf{U}}_*(s))\mathbf{e}_k(\Delta(s))\, \mathrm{d}s \Bigg| \end{aligned} \tag{30}$$

for any $t > 0$, where $\mathbf{f}_\mathbf{U}$, $\mathbf{f}_{\mathbf{U}_*}$ are matrices with a gradient for each component of the vector field for its rows, the gradient with respect to non-delayed terms and delay terms, and $\underline{\mathbf{U}}$ indicates the Lagrange form of the remainder for the 1st order Maclaurin expansion of $\mathbf{f}$. Define I by taking $||\mathscr{N}\mathbf{V}_k||_1 \geq \left|\mathbf{e}_k(t_1^*) - I\right|$ to be equivalent to (30).

If, for example, $I < 0$, then it would follow that $|\mathbf{e}_k(t_1^*)| \leq ||\mathscr{N}\mathbf{V}_k||_1$. Hence, a bound for the solution error would be computable. However, if $I > 0$, then care must be taken in interpreting the calculated value of $||\mathscr{N}\mathbf{V}_k||_1$. It is possible that $|||\mathscr{N}\mathbf{V}_k||_1$ is small, but $|\mathbf{e}_n(t_1^*)|$ is large, which can occur when I is close in value to $|\mathbf{e}_k(t_1^*)|$.

6.2 Paul

The DDE is $\dot{u} = u(u(t))$ for $t > 2$ with discontinuous initial history: $u(t) = \frac{1}{2}$ if $t < 2$ and $u(2) = 1$. However, the breaking point from this jump in the initial history occurs at t_0,

Table 1 Paul: uniform norm of operator error and breaking point errors

	(τ_0, τ_1)	(τ_1, τ_2)	(τ_2, τ_3)	
$\|\|u(t) - \phi(t)\|\|_\infty$	<machine error	4.3997e–9	8.8165e–8	
	(τ_0, τ_1)	(τ_1, τ_2)	(τ_2, τ_3)	(τ_3, τ_4)
$\|\|\mathscr{N}\mathbf{V}\|\|_\infty$	<machine error	<machine error	<machine error	1.6734e–8
	(τ_4, τ_5)	(τ_5, τ_6)	(τ_6, τ_7)	(τ_7, τ_8)
	8.3668e–8	8.0217e–8	2.7154e–8	5.3855e–9
	τ_1	τ_2	τ_3	τ_4
e_m^τ	2.2204e–16	2.8266e–8	6.1288e–8	7.9876e–8
	τ_5	τ_6	τ_7	τ_8
	8.5228e–8	8.6262e–8	8.6436e–8	8.6465e–8

the seed for $\mathscr{T}_\Delta$, so that the breaking points are also the MoS endpoints. The exact solution is known over the first three MoS subintervals. More information about the Paul problem is included in the help file for `RADAR5`, which can be found at [16], and the relevant pages are 33,34 and 39,40.

For comparison, `RADAR5` reports a CPU time of roughly 6.77e–2 seconds and an error of roughly 0.129e–7 for the Paul problem when a numerical parameter has a particular value. By decreasing this numerical parameter, `RADAR5` can achieve an error as small as 8.8e–14 (but without computational times listed). The domain used for the `RADAR5` version is (2,5.5) or equivalently (τ_0, τ_3).

The problem is considered using PSMd over (τ_0, τ_8) and this required 0.1 sec of CPU time. Each I_m had 10 subdivisions, and 10 Picard iterations were performed. Since the exact solution is known over (τ_0, τ_3), the sup norm of the error in the solution is computed. This is supplemented by the operator error over (τ_0, τ_8), along with the errors in the computed breaking points, in Table 1.

6.3 Modified Bellman: State Dependent Delay

This is problem (21) on page 22, with delay $t - 1 - e^{-u(t)}$ and DDE $\dot{u} = -uu_*$. The solution decreases when it is positive. If $u \to 0$ as t increases, then the lag would approach 2. This experiment required 4 sec of CPU time.

Computing 10 Picard iterations in each subdivision and 100 subdivisions in each I_m, the sup norm operator errors are listed in the Table 2. Note that the sup norm of the operator error is roughly 10^{-18}. The errors in breaking point calculations are absolute and their size indicates accuracy of the locations of the breaking points. The initial width between breaking points is roughly 1.37, then 1.86 and then always larger than 1.94 appearing to approach 2 as expected.

Table 2 Modified Bellman: uniform norm of operator error and breaking point errors

	(τ_0, τ_1)	(τ_1, τ_2)	(τ_2, τ_3)	(τ_3, τ_4)
$\|\|\mathscr{N}\mathbf{V}\|\|_\infty$	0.000e–0	8.3267e–17	6.9389e–18	2.1684e–18
	(τ_4, τ_5)	(τ_5, τ_6)	(τ_6, τ_7)	(τ_7, τ_8)
	1.7347e–18	1.3010e–18	1.0842e–18	1.0842e–18
	τ_1	τ_2	τ_3	τ_4
e_m^τ	5.5595e–4	7.3561e–3	8.0297e–3	8.3607e–3
	τ_5	τ_6	τ_7	τ_8
	8.6295e–3	8.8574e–3	9.0534e–3	9.2239e–3

Table 3 Distributed delay: uniform norm of operator error (no segments)

	(τ_0, τ_1)	(τ_1, τ_2)	(τ_2, τ_3)	(τ_3, τ_4)	(τ_4, τ_5)	(τ_5, τ_6)
$\|\|\mathscr{N}\mathbf{V}\|\|_\infty$	9.9834e–04	7.3530e–04	5.5316e–04	4.1559e–04	3.1244e–04	2.3485e–04
	(τ_6, τ_7)	(τ_7, τ_8)	(τ_8, τ_9)	(τ_9, τ_{10})	(τ_{10}, τ_{11})	(τ_{11}, τ_{12})
	1.7653e–04	1.3270e–04	9.9745e–05	7.4977e–05	5.6359e–05	4.2364e–05

6.4 Distributed Delay

Using a decaying exponential for the ODE part, the case of an exponential kernel with parameters α, b and τ all set to unity yields

$$\dot{u}(t) = -u(t) + \int_0^1 e^{-s} u(t-s)\, \mathrm{d}s$$

and this was coupled with an initial history which was also constant with height set equal to unity. Using 10 Picard iterations per I_m, the problem was run until time $t = 12$. None of the I_m were segmented, and Table 3 contains the sup norm of the operator error in each I_m. This experiment required 1 sec of CPU time to compute the PSMd approximation. The numerical integration to compute the error only used the trapezoidal rule with a mesh size of 0.01 and required 3 sec of CPU time.

6.5 Delayed Kuramoto

This experiment combines an unfrustrated deterministic *Kuramoto* model with a constant lag, see [17], whose notation is adopted here. In particular, the asymptotic behavior of the system is parameter dependent. The results show that PSMd captures that behavior properly.

For N oscillators, the system has the ith oscillator's phase θ_i as its states, with two parameters: the coupling strength K and the inherent (constant) frequency of each oscillator

Table 4 Delayed Kuramoto: synchronization; Uniform norm of operator error

	1	2	3	4	5	6
$\|\|\mathscr{N}\mathbf{V}\|\|_\infty$	1.8178e–3	3.1565e–3	6.4722e–3	8.0360e–3	7.4618e–3	4.9601e–3
	7	8	9	10	11	12
	5.2274e–4	3.3401e–3	5.9040e–3	8.0801e–3	7.2293e–3	4.7355e–3

ω_i. The delayed Kuramoto can be written

$$\dot{\theta}_i(t) = \omega_i + \frac{K}{N}\sum_1^N \sin(\theta_i(t-\tau) - \theta_i(t))$$

for $i = 1, \ldots, N$.

The vector field for the model is autonomous with $t_0 = 0$, so to achieve a polynomial vector field, define $\Theta_{ij} \equiv \sin(\theta_j(t-\tau) - \theta_i(t))$ and $\Psi_{ij} \equiv \cos(\theta_j(t-\tau) - \theta_i(t))$ with $i, j = 1, \ldots, N$. The delayed Kuramoto model can now be rewritten as

$$\begin{cases} \dot{\theta}_i = W_i \\ \dot{\Theta}_{ij} = \Psi_{ij}(\dot{\theta}_{j*} - W_i) \\ \dot{\Psi}_{ij} = -\Theta_{ij}(\dot{\theta}_{j*} - W_i) \end{cases} \tag{31}$$

with $W_i \equiv \omega_0 + \frac{K}{N}\sum_{j=1}^N \Theta_{ij}$ and $\dot{\theta}_{j*} \equiv \dot{\theta}_j(t-\tau)$ where $i, j = 1, \ldots, N$. In terms of auxiliary variables, this becomes a $2N^2 + N = 300$ component system when $N = 12$.

The oscillators were equally spaced around the circle and then perturbed slightly by a random amount, while each were given an initial angular velocity of $\frac{\pi}{2}$. When the coupling strength $K = 1$ and the lag $\tau = 1$, then synchronization should occur, i.e. all oscillators should have equal phases mod 2π. When $K = 1.25$ and $\tau = 2$, this should yield an incoherent state, i.e. the oscillators' phases are distinct.

For the incoherent case, 20 Picard iterations in each I_m, which were not segmented, were performed, i.e. the approximations were degree 20 Maclaurin polynomials, while only 10 iterations were needed for synchronization. The code for this problem took advantage of the structure of the auxiliary variables and is actually one of the shortest codes used in this section. These results required 23 sec of CPU time.

Table 4 contains the sup norm of the operator error at the final time for a particular set of initial positions of the 12 oscillators in the synchronization case. The sup norm of the operator errors are displayed over the interval (0, 50).

Graphical results are included in Fig. 2. The top subplot indicates synchronization, while the bottom indicates incoherence. The phase differences for oscillator i and oscillator $i+1$, i.e. $\theta_{i+1} - \theta_i$, for $i = 1, \ldots, 11$ are plotted, with all differences smaller than 10^{-3} (mod 2π) in the top subplot.

6.6 Circuit Problem

This experiment, dubbed the *circuit problem*, can be found in [18]. This problem has a linear, hence polynomial, autonomous vector field, and begins at $t_0 = 0$ and thus requires no auxiliary variables. This system is a three component system, which is both neutral as well as *moderately* stiff (eigenvalue ratio is $3.1135 \cdot 10^5$).

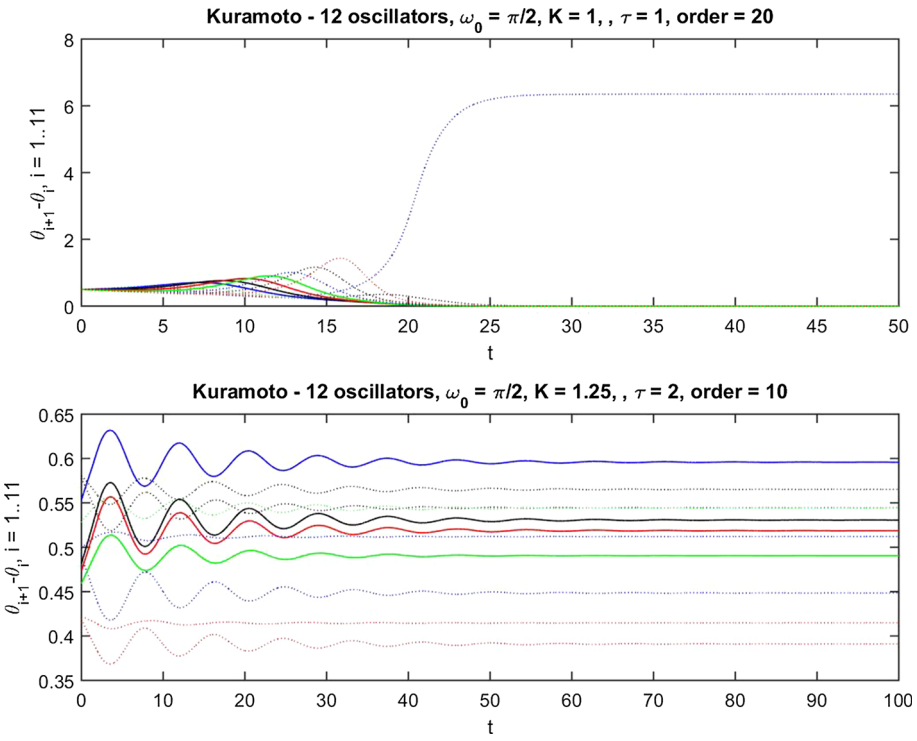

Fig. 2 Delayed Kuramoto—synchronization and incoherence

The system of equations can be written in vector form

$$\dot{\mathbf{u}} = \mathbf{L}\mathbf{u} + \mathbf{M}\mathbf{u}_* + \mathbf{N}\dot{\mathbf{u}}_*$$

where $\mathbf{u} = [\,u_1\ u_2\ u_3\,]^T$ and $\mathbf{u}_*$ denotes the deviating argument. The matrices $\mathbf{L}$, $\mathbf{M}$ and $\mathbf{N}$ are given by

$$\mathbf{L} = 100\begin{bmatrix} -7 & 1 & 2 \\ 3 & -9 & 0 \\ 1 & 2 & -6 \end{bmatrix} \quad \mathbf{M} = 100\begin{bmatrix} 1 & 0 & -3 \\ -0.5 & -0.5 & -1 \\ -0.5 & -1.5 & 0 \end{bmatrix} \quad \mathbf{N} = \frac{1}{72}\begin{bmatrix} -1 & 5 & 2 \\ 4 & 0 & 3 \\ -2 & 4 & 1 \end{bmatrix}$$

The initial history is comprised of sines and cubic approximations to these are used in the computation due to the small segment lengths. The lag $\tau = 1$ for all experiments.

Stiffness required 50,000 segments and 30 Picard iterations to generate the table entries in 4204 sec of CPU time, while only 2000 segments and 5 iterations were needed to achieve plotting accuracy, requiring 71 sec. While not done here, it is possible to change the number of segments in I_m as m changes, for a small computational cost, which may reduce overall CPU time.

Table 5 contains the operator errors. The operator error definitions were restricted to elements of $\mathscr{I}_\Delta$ to gain information about where and for how long large operator errors occur. The information around $t = 1$ shows where the stiffness of the problem is strongest. While the one norm errors are 10^{-4}, the sup norm errors are much larger. Investigation showed that these large sup norm errors only occurred in the first segment after $t = 1$.

Table 5 Circuit: uniform and one norm of operator error

$\|\|\mathscr{N}\mathbf{V}\|\|_\infty$	(τ_0, τ_1)	(τ_1, τ_2)	(τ_2, τ_3)	(τ_3, τ_4)	(τ_4, τ_5)
u_1	1.8918e–2	8.6370e–1	2.8144e–2	2.8326e–2	1.0620e–2
u_2	8.6678e–3	3.8240e–1	7.8971e–2	2.1371e–2	1.1100e–2
u_3	6.3071e–3	1.9481e–1	4.6027e–2	1.7421e–2	1.0262e–2
$\|\|\mathscr{N}\mathbf{V}\|\|_1$	(τ_0, τ_1)	(τ_1, τ_2)	(τ_2, τ_3)	(τ_3, τ_4)	(τ_4, τ_5)
u_1	5.556e–3	2.1180e–3	1.0049e–3	4.5843e–4	2.2150e–4
u_2	2.4763e–3	1.9130e–3	8.9902e–4	4.3155e–4	2.0654e–4
u_3	1.9791-3	1.6816e–3	7.9083e–4	3.8122e–4	1.8120e–4

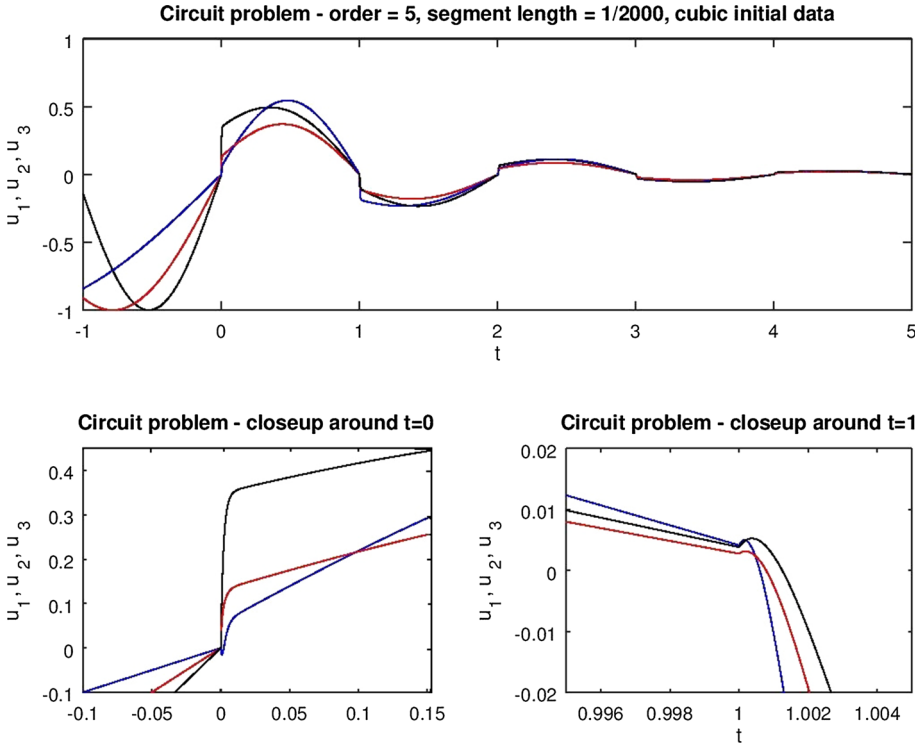

Fig. 3 Circuit—includes closeups of $t = 0$ and $t = 1$

Note that u_1's slope difference of roughly one over a run of 1/50000 implies an error in the dependent of roughly 2e–5. The one norm does not give any evidence that this large slope error persists for a long time interval. The vector field components evaluated at τ_1^+ are large in magnitude, so that the relative sup norm operator errors are small by comparison. A plot of each component's approximation superimposed is shown in Fig. 3, with u_1 in blue, u_2 in red and u_3 in black.

6.7 Blood Spleen and Tumor Model

A model for the growth of a melanoma tumor in the face of a vaccine supported immune response is considered for simulation. Numerical results were presented in [19] and the results of PSMd's replication are given here for comparison. See the appendix on page 41 for the full system of equations, although two relevant equations will be displayed below.

The plot in Fig. 4 is two distinct stages in one problem. The CPU time required for one parameter choice was 1880 sec (there are 4 in the plot). This was due to the large number of segments needed for the equation stiffness, rather than the vaccine's support along with the sizable time interval, which is ≈ 20 days.

The system has 9 components which measure the number of cells at time t for several agents: T, for tumor, dendritic cells in the blood, spleen and tumor, respectively D_B, D_S, and D_T. There are also two types of cytotoxic T lymphocytes (CTLs), activated and memory, for blood and spleen, with activated CTLs only for tumor: E_S^a, E_S^m, E_B^a, E_B^m and E_T^a.

The polynomial vector field used for computation was quartic and has 17 components after introducing some auxiliary variables, more info can be found in the Appendix. The equations in the system are all ODEs except for $\dot{E_S^a}$ which introduces a term with a constant lag. In particular,

$$\dot{E_S^a} = aE_B^a + b(D_S)E_S^a + cD_SE_S^m + e(d - E_S^a) + fE_S^a + g\frac{D_{S*}E_{S*}^a}{h + D_{S*}}$$

where $a, c, d, \ldots, h$ are constants and $b(D_s) = A + \frac{B}{1+CD_S}$ for constants A, B and C.

This problem also has a few features which makes it challenging. One aspect is that the parameters in the differential equations range from 9.42e–12 to 1e9. There is also a forcing function, which represents vaccination, and it has support that happens to be much smaller than the lag. These two issues require the use of segments.

There is a term in the vector field for T which is nonanalytic at $t = 0$:

$$\dot{T} = rT\left(1 - \frac{T}{k}\right) - dT\frac{\left(\frac{E_T^a}{T}\right)^{\frac{2}{3}}}{s + \left(\frac{E_T^a}{T}\right)^{\frac{2}{3}}}$$

with $E_T^a(0) = 0$ and r, k, d and s are parameters.

Rather than using a different method to advance the problem forward, and then handing that result off to PSMd to continue, the issue was circumvented by noting that the solution to $\dot{v} = v^{\frac{2}{3}}, \quad v(0) = \epsilon$ with $\epsilon \geq 0$ is continuous wrt ϵ. Hence, for ϵ small enough, $v \approx u$ where u solves $\dot{u} = u^{\frac{2}{3}}, \quad u(0) = 0$.

Further, the root can be Taylor expanded. Hence, the initial data of the auxiliary variable for $(E_T^a T^{-1})^{\frac{2}{3}}$ is ϵ. Values between 1e–4 and 1e–15 were used for ϵ, and essentially the same results were obtained in each case. In particular, $\epsilon = 1/6$e–5 was used for all experiments in this paper.

The effect of CTL kill factor d was replicated in this experiment. The problem involves vaccination (via bloodstream) on day 0 and day 7 prior to the tumor challenge with varying d. It was assumed that the injection time was a half hour and the lag is half of a day.

The error is only determined after the tumor challenge and is given for one value of the CTL kill factor: $d = 1$. In addition, Fig. 4 is a replication of the bottom panel of Fig. 9 found in [19]. The support of the forcing term (vaccination) is divided into 300 segments and for a 3rd degree polynomial, the errors are given in Table 6.

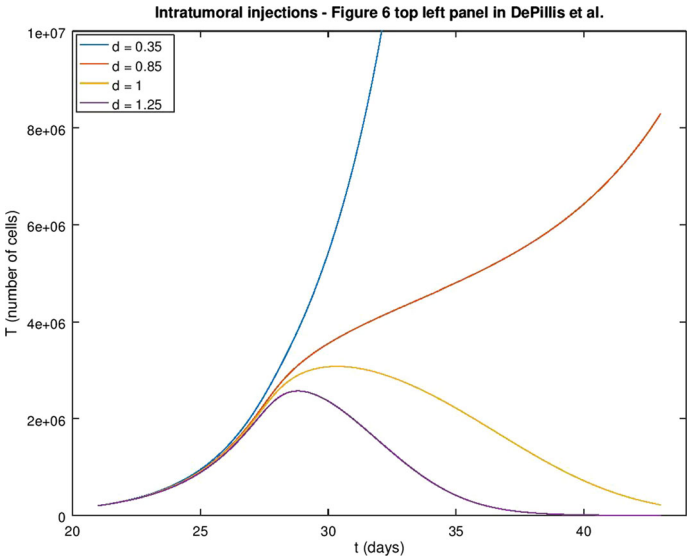

Fig. 4 Blood spleen tumor model, $\tau = \frac{1}{2}$

Table 6 Blood, spleen and tumor: uniform and one norm operator error

	T	DS	DB	DT	
$\|\|\mathscr{N}\mathbf{V}\|\|_\infty$	6.1039e0	1.5273e–8	1.4725e–7	3.8142e–7	
$\|\|\mathscr{N}\mathbf{V}\|\|_1$	3.5317e–4	6.7808e–9	2.7670e–9	1.1073e–6	
	EaT	EaB	EmB	EmS	EaS
$\|\|\mathscr{N}\mathbf{V}\|\|_\infty$	5.0350e–4	4.0322e–4	1.7735e–6	1.2408e–5	3.7590e–3
$\|\|\mathscr{N}\mathbf{V}\|\|_1$	3.5115e–4	1.6651e–4	9.6685e–7	9.9594e–6	2.0436e–3

In Table 6, the value of Δt for the error calculation is roughly 1e–4. Like the circuit problem, only one segment is responsible for the sup norm operator error for T. In particular, it is the segment right after τ_0. This is expected due to the use of ϵ to approximate the initial data for the nonanalytic term of the vector field. This error is largest at $t = \tau_0$.

6.8 MoS Failure and CG

A turning point more than likely will not delay back to a member of $\mathscr{T}_\Delta$ thereby disrupting MoS. In addition, a vanishing lag is a limit point but not a member of $\mathscr{T}_\Delta$. Hence, the set of turning points and set of vanishing lags should be included, and the union of all three sets is relabeled $\mathscr{T}_\Delta$.

Assume that $\mathscr{T}_\Delta$ is made up of isolated points. Vanishing lags and delay turning points are referred to as *failure points* of MoS. If a failure point occurs in $\mathscr{T}_\Delta$, then surround that with a local interval. MoS, and hence PSMd, by design, is successful outside these local intervals, but inside, a modified or different approach is necessary. The intervals around the failure points will be referred to as *extrapolation zones*.

The left endpoint of the extrapolation zone for a vanishing lag can be chosen, with a trade off between closeness to the vanishing lag and increasing CPU time. For a turning point denoted t^* that does not delay back to some $\tau_m \in \mathscr{T}$, the left endpoint for the extrapolation zone needs to be $\max\{\tau_m \in \mathscr{T}_\Delta | \tau_m < t^*\}$.

PSMd as developed so far needs to be coupled with a decision making process to determine if the approximation is approaching a failure point, and if so, what kind. The process also needs to determine what behavior occurs at breaking points as well. A claim to a decision making process that is robust and complete is not made here.

In particular, PSMd could be supplemented with another method for the extrapolation zone's computation. PSMd could provide as history, either a formula (polynomial) or a vector of outputs (polynomial evaluated at a vector of inputs). By design, PSMd only uses history outside the extrapolation zone, so the method that supplements only needs to supply just the height and time at the end of the extrapolation zone for PSMd to continue.

In the extrapolation zone, only the change of variable Δ_m is predicated on the MoS structure, and has to be disregarded. The choice that is made is to approximate (11) with polynomials and Picard's, rather than approximate (19). The burden is then to replicate the $u(\Delta_u(t))$ information in a form that is compatible.

A problem introduced in Castleton and Grimm [13], dubbed CG in the instructions for the code in [20], has a very simple solution $u(t) = \sin(t)$ so that direct errors may be computed. Via appropriate trigonometric identities and trading some sin terms for u terms to form the delay, one can get the equation for any $c \in \mathbb{R}$, with $\phi(t) \equiv t \sin^2(t)$

$$\dot{u}(t) = \cos(t)\left(1 + u\left(tu^2(t)\right) - \sin\left(\phi(t)\right)\right) + c\left(u(t)\dot{u}\left(tu^2(t)\right) - \sin(t)\cos\left(\phi(t)\right)\right)$$

with only an initial condition being needed, namely $u(0) = 0$. A polynomial form for this DDE is possible by using auxiliary variables for $\cos t$, $\sin t$, ϕ, $\cos\phi$ and $\sin\phi$.

This DDE can be a challenge to approximate, see also the delay in [21]. The problem has (what ultimately will be) a delay without monotonicity. Note that since PSMd relies on $(\Delta^{-1})'(t) \to \infty$ as $\Delta'(t) \to 0$, this can be used to signal that a turning point may be approaching.

This problem is considered over the interval of $[0, \pi]$. The problem begins with a vanishing lag. There is also little room between the vanishing lag at $t = \frac{1}{2}\pi$ and the turning point of $\Delta_u(t)$ occurring at $t \approx 1.8365968$, but this needs to be detected by the implementation.

There is a removable discontinuity in the DDE concerning $\dot{u}(t)$. If $c = 1$, then when $t = \frac{\pi}{2} = t_v$, the $\dot{u}(t)$ term cancels on both sides of the equal sign, so that $\dot{u}(t_v)$ cannot be computed using the DDE. If Δ_u is to remain a delay, then any time the lag vanishes, $\dot{u}(t) = 0$ if $u(s)$ is increasing for $s < t$, hence $\dot{u}(t_v)$ should be 0.

Over $[0, \pi]$, there are 5 failure points surrounding 4 MoS regions. The problem starts with a vanishing lag, so a linear expansion was generated over an interval chosen by the user, requiring only an evaluation of the vector field.

The first two MoS regions are dubbed Zone I and II, which occur over the increasing section of the delay. The two regions during which the delay is decreasing are (in order) Zone IId and Id. This is because Zone IId uses the approximation from Zone II for its history. The failure points are labeled (in order) t_v, between Zone I and II (vanishing lag), t_M between II and IId (turning point), t_v^s between IId and Id (delays to t_v) and t_m at the end (turning point). The approximate values for these failure points are $\frac{\pi}{2}$, 1.8365968, 2.0943951, π but are not known a priori.

This simple version of PSMd was effective for $c \in [-1, 0.9]$. The results for $c = 0.9$ are displayed in Table 7 and Fig. 5. A better extrapolation zone strategy would improve the

Table 7 CG: uniform norm and breaking point errors for $c = 0.9$

	Zone I	Zone II	Zone IId	Zone Id
$\|\|u - \sin\|\|_\infty$	1.2175e–4	1.0494e–4	2.5274e–4	2.0966e–3
	$\tau = t_v$	$\tau = t_M$	$\tau = t_v^s$	$\tau = t_m$
$\|\|\tau_m - \tau\|\|_\infty$	6.4335e–6	5.9910e–3	2.8834e–4	4.2641e–3

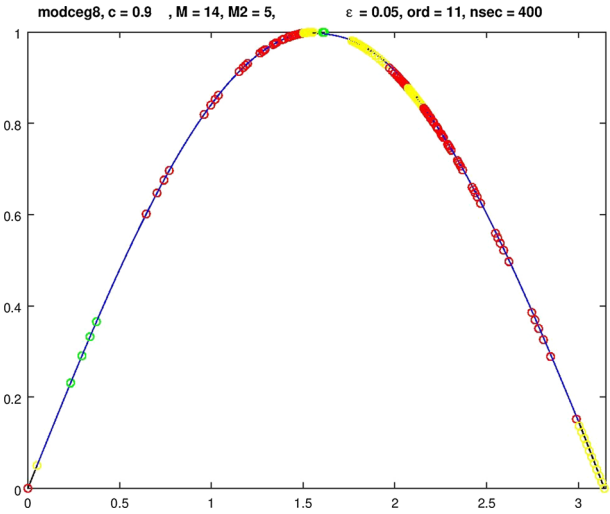

Fig. 5 CG with $c = 0.9$, exact solution in circles, extrapolation zones are yellow

overall errors. This experiment computed 11 Picard iterations, and used 400 segments per I_m, requiring 312 sec of CPU time. The case $c = -1$ only requires 104 sec of CPU time.

7 Conclusion

Working with a single smooth simple delay, a smooth vector field and smooth initial history, a change of variable involving compositions of the delay successfully converts IVPs involving retarded, neutral or advanced DDEs into equivalent ones now involving ODEs with a quadratic vector field. These equivalent problems also have some ODEs that are generated by the delay. The delay may be nonlinear and/or state dependent.

Having this conversion in place, it is very easy to implement Picard iteration to generate the Maclaurin expansion of the solution to the ODEs. This involves at each iteration a discrete convolution of the coefficients of the polynomials representing the system's components. This is done for each quadratic term in the vector field, and appropriately summed together. To integrate, the resulting vector of coefficients are shifted right and divided by the iteration number plus one.

This method of approximation by Picard iteration with polynomials, known as PSMd, can also handle delays which are not monotonic, or initial data that has a finite number of jumps. The method can be used on stiff problems by subdividing the basic computing unit.

With obvious modifications, the method can also approximate problems involving explicit distributed delays. Loosening the restrictions of smoothness for the vector field and delay can be considered for future work, but more importantly, it would be great to lift the single delay restriction for PSMd.

Acknowledgements The author would like to thank Dr. James Sochacki and Dr. Anthony Tongen for their contributions which enabled this manuscript to be produced. The author would also like to thank Dr. Lisette de Pillis, who made the author aware of the blood, spleen and tumor problem. This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors. The author would also like to thank the (hard working) referees for their helpful suggestions, which ultimately made this a better paper.

Declarations

Conflict of interest The author declares that they have no conflict of interest.

8 Appendix

For a full discussion of the BST model, see [19] and references therein. This is the list of 9 components from the full system of DEs in the BST problem, see page 35, which are all functions of t

$$T, D_B, D_S, D_T, E_S^a, E_S^m, E_B^a, E_B^m, E_T^a$$

This is the list of parameters (all constant) from the same system:

$$\mu_B, \mu_{TB}, \mu_{BB}, \max D, \mu_{BS}, a_D, b_{DE}, \mu_{BSE}, b_a, a_{E_aS}, E_{naive}, r_{am}, b_p, \theta_D, a_{E_m}, a_{E_aT}, c, r, k, q, m, d, s, l, \alpha, \mu_{SB}^{normal}, \mu_{SB}^*$$

The functions $v_{blood}(t)$ and $v_{tumor}(t)$ are nonhomogeneous (forcing) terms. This is a list of known functions in the same system:

$$\mu_{SB}(D_S) \equiv \mu_{SB}^* + \frac{\mu_{SB}^{normal} - \mu_{SB}^*}{1 + \frac{D_S}{\theta_{shut}}} \qquad DC_{on} \equiv \begin{cases} 0 & \text{if } D_S(t) = 0 \\ 1 & \text{if } D_S(t) > 0 \end{cases}$$

$$\mu_{BTE} \equiv \mu_{BB}\left(\frac{T}{\alpha + T}\right) \qquad \mathscr{D} \equiv d\frac{\left(\frac{E_T^a}{T}\right)^l}{s + \left(\frac{E_T^a}{T}\right)^l}$$

The full system of equations, using the above and prime for derivative, is

$$\begin{cases} D_B' = -\mu_B D_B + \mu_{TB} D_T + v_{blood}(t) \\ (E_B^a)' = \mu_{SB}(D_S)E_S^a - \mu_{BB}E_B^a \\ (E_B^m)' = \mu_{SB}(D_S)E_S^m - \mu_{BB}E_B^m \end{cases}$$

$$\begin{cases} D_S' = \max D\left(1 - e^{\frac{-\mu_{BS}D_B}{\max D}}\right) - a_D D_S - b_{DE}E_S^a D_S \\ (E_S^a)' = \mu_{BSE}E_B^a - \mu_{SB}(D_S)E_S^a + b_a D_S E_S^m + a_{E_aS}(DC_{on}E_{naive} - E_S^a) \\ \qquad -r_{am}E_S^a + b_p\frac{D_S(t-\tau_D)E_S^a(t-\tau_D)}{\theta_D + D_S(t-\tau_D)} \\ (E_S^m)' = r_{am}E_S^a - (a_{E_m} + b_a D_S + \mu_{SB}(D_S))E_S^m + \mu_{BSE}E_B^m \end{cases}$$

$$\begin{cases} (E_T^a)' = \mu_{BTE}(T)E_B^a - a_{E_aT}E_T^a - cE_T^aT \\ T' = rT(1-\frac{T}{k}) - \mathscr{D}T \\ D_T' = \frac{mT}{q+T} - (\mu_{TB}+a_D)D_T + v_{tumor}(t) \end{cases}$$

Auxiliary variables were used for $\mu_{SB}, \mu_{BTE}, E_T^aT^{-1}, \mathscr{D}, 1-e^{\frac{-\mu_{BS}D_B}{\max D}}, (\theta_D+D_S(t-\tau_D))^{-1}$, $(q+T)^{-1}$, and $(\cdot)^l$ since $l=2/3$, to polynomial ones.

References

1. Hale, J.K., Lunel, S.M.V.: Introduction to Functional Differential Equations. Springer-Verlag, New York (1993)
2. Carothers, D.C., Parker, G.E., Sochacki, J.S., Warne, P.G.: Some properties of solutions to polynomial systems of equations. Electron. J. Differ. Equ. **2005**(40), 1–17 (2005)
3. Parker, G.E., Sochacki, J.S.: Implementing the Picard iteration. Neural Parallel Sci. Comput. **4**(1), 97–112 (1996)
4. Sochacki, J.S.: Polynomial ODEs—examples, solutions and properties. Neural Parallel Sci. Comput. **18**, 441–449 (2010)
5. Hu, Q.: A model of regulatory dynamics with threshold-type state-dependent delay. Math. Biosci. Eng. **15**(4), 863–882 (2018)
6. Isaia, V.M.: Nonlinear differential equations with deviating arguments and their approximation via a Parker–Sochacki approach. Electron. J. Differ. Equ. **2017**(68), 1–23 (2017)
7. Baker, C.T.H., Paul, C.A.H., Willé, D.R.: Issues in the numerical solution of evolutionary delay differential equations. Adv. Comput. Math. **3**, 171–196 (1995)
8. Fehlberg, E.: Numerical Integration of Differential Equations by Power Series Expansions, Illustrated by Physical Examples. Technical Report NASA-TN-D-2356, NASA (1964)
9. Parker, G.E., Sochacki, J.S.: A Picard–Maclaurin theorem for initial value PDEs. Abstr. Appl. Anal. **5**(1), 47–63 (2000)
10. Norkin, S.B.: Differential Equations of the Second Order with Retarded Argument. Translations of Mathematical Monographs (L. J. Grimm) vol. 31, AMS, Providence (1972)
11. Bellen, A., Guglielmi, N.: Solving neutral delay differential equations with state-dependent delays. J. Comput. Appl. Math. **229**, 350–362 (2009)
12. Bellen, A., Zennaro, M.: Numerical Methods for Delay Differential Equations. Oxford University Press, Oxford (2003)
13. Castleton, R.N., Grimm, L.J.: A first order method for differential equations of neutral type. Math. Comput. **27**(123), 571–577 (1973)
14. Bellman, R.E., Buell, J.D., Kalaba, R.E.: Numerical integration of a differential-difference equation with a decreasing time-lag. Commun. ACM **8**(4), 227–228 (1965)
15. Warne, P.G., Warne, D.A.P., Sochacki, J.S., Parker, G.E., Carothers, D.C.: Explicit a-priori error bounds and adaptive error control for approximation of nonlinear initial value differential systems. Comput. Math. Appl. **52**, 1695–1710 (2006)
16. http://www.unige.ch/~hairer/software.html
17. Yeung, M.K.S., Strogatz, S.H.: Time delay in the Kuramoto model of coupled oscillators. Phys. Rev. Lett. **82**(3), 648–651 (1996)
18. Bellen, A., Guglielmi, N., Ruehli, A.E.: Methods for linear systems of circuit delay differential equations of neutral type. IEEE Trans. Circuits Syst. I Fundam. Theory Appl. **46**(1), 212–215 (1999)
19. de Pillis, L., Gallegos, A., Radunskaya, A.: A model of dendritic cell therapy for melanoma. Front. Oncol. **3**(56), 1–14 (2013)
20. Guglielmi, N., Hairer, E.: Computing breaking points in implicit delay differential equations. Adv. Comput. Math. **29**(3), 229–247 (2008)
21. Bellman, R.E., Buell, J.D., Kalaba, R.E.: Mathematical experimentation in time-lag modulation. Commun. ACM **9**(10), 752–754 (1966)