Quantization Based Integration Methods for Delay Differential Equations

Rodrigo Castro[a,c], Ernesto Kofman[b,c], François E. Cellier[d]

[a] *Departamento de Computación. Universidad de Buenos Aires. Argentina*
[b] *Departamento de Control. FCEIA. Universidad Nacional de Rosario. Argentina*
[c] *CIFASIS – CONICET. Argentina*
[d] *Department of Computer Science, ETH Zurich. Switzerland*

Abstract

This paper introduces a new class of numerical delay differential equation solvers based on state quantization instead of time slicing. The numerical properties of these algorithms, i.e., stability and convergence, are discussed, and a number of benchmark problems are being simulated and compared with the state-of-the-art solutions to these problems as they have been previously reported in the open literature.

Keywords: Delay differential equation, Numerical DDE solver, State quantization, Quantized state system, PowerDEVS

1. Introduction

The numerical solution of models of dynamic systems has been of interest for many years. Many physical systems can be approximated by sets of ordinary differential equations (ODEs), and consequently, the digital simulation of such models has caught the interest of engineers and applied mathematicians since the invention of the digital computer. These efforts have been summarized in many textbooks [4, 5, 8, 9, 12].

However, there are also a significant number of systems from both science and engineering that require, in their models, the inclusion of delays [1]. In spite of their importance, the numerical simulation of models described by sets of delay differential equations (DDEs) has been covered by much fewer publications, and also, the state of the art of software for dealing with such models is much less highly developed.

Good solvers for simulating DDE models were developed recently by Shampine, Thompson, and co–workers. These include a numerical DDE solver encoded in Matlab, called dde23 [17, 18]. They also include a numerical DDE solver encoded in Fortran, called ddesolver [19]. Both solvers are classical solvers in the sense that they are based on the classical time–slicing algorithms used throughout the numerical ODE, DDE, and DAE (differential algebraic equation) literature.

In this article, we wish to advance the state of the art of numerical DDE solvers by introducing a new class of DDE solvers, called DQSS, that are based on state quantization in place of time discretization. The solvers have been implemented in a modeling and simulation environment called PowerDEVS [3]. PowerDEVS simplifies significantly the use of these solvers.

Numerical ODE solvers based on quantized state systems (QSS) for non–stiff systems were previously described in [5, 11]. Currently available are non-stiff ODE solvers of orders 1..4 (QSS1, QSS2, QSS3, and QSS4).

A first–order accurate numerical ODE solver for stiff systems, called BQSS (backward QSS), was previously described in [6]. A more formal discussion of the stability and convergence features of BQSS is currently in preparation. A class of higher-order linearly implicit stiff system solvers has meanwhile also been developed. This work has been reported in [13, 14]. Currently available are linearly implicit stiff ODE solvers of orders 1..3 (LIQSS1, LIQSS2, and LIQSS3).

A first–order accurate numerical ODE solver for marginally stable systems, called CQSS (centered QSS), was previously described in [6]. The CQSS method is symmetric and symplectic and preserves ρ–reversibility [7].

A discussion of discontinuity handling (root solving) in QSS solvers can be found in [10].

QSS methods turn out to be very useful for the simulation of DDE models. There are a number of reasons for this observation. QSS methods offer dense output, and they preserve that information not only during the current step, but even across the entire simulation. QSS trajectories consist in sequences of polynomial segments. The coefficients of these polynomial segments change only at event times, i.e., when a state variable changes its quantization level. It suffices to store the event times and the polynomial coefficients valid during the immediately following time interval until the subsequent event takes place in order to reconstruct the trajectory precisely. This feature can be exploited in the calculation (interpolation) of the delayed signals. The information is maintained for each state variable separately, since QSS methods

are naturally asynchronous, i.e., each state variable is updated independently from all others, whenever it crosses through the next quantization threshold.

The paper is organized as follows. After a brief introduction to state quantization, the basic idea behind QSS methods is presented in Section 2 of the paper. Section 3 discusses the generalization of the QSS concept to handling DDEs and explains the DQSS algorithms in detail. Section 4, together with the Appendices, discusses the numerical properties of the DQSS algorithms, i.e., stability and convergence. Section 5 presents the DQSS simulations of four separate DDE benchmark models found in the open literature [18, 19].

1.1. Intuitive Idea

We shall follow a step by step procedure to solve a typical DDE example from the literature [2] based on a state quantization approach. Assume we wish to simulate the system:

$$\begin{aligned} \dot{x}(t) &= x(t-1)\ ,\ t \geq 0 \\ x(t) &= 1\ ,\ -1 < t \leq 0 \end{aligned} \tag{1}$$

for $t \geq 0$, with initial history $x(t) = 1$. This is a first-order DDE containing a single constant delay of $\tau = 1$.

We shall consider the following quantization function:

$$Q(x) \triangleq floor[\frac{x}{\Delta Q}].\Delta Q \tag{2}$$

where ΔQ is a parameter called *quantum*.

In Figure 1, the relationship of Eq. (2) is depicted for a quantum of $\Delta Q = 0.5$.

Let us now approximate the original system of Eq. (1) as follows:

$$\begin{aligned} \dot{x}(t) &= Q(x(t-1)) \triangleq q(t-1)\ ,\ t \geq 0 \\ x(t) &= 1\ ,\ -1 < t \leq 0 \end{aligned} \tag{3}$$

i.e., we replaced $x(\cdot)$ by $q(\cdot) \triangleq Q(x(\cdot))$ on the right hand side of the state equation. The variable $q(t)$ is called *quantized variable*.

Notice that $Q(1) = 1$ and hence $q(t) = 1$ for $t \leq 0$. Thus, we can rewrite Eq. (3) as follows:

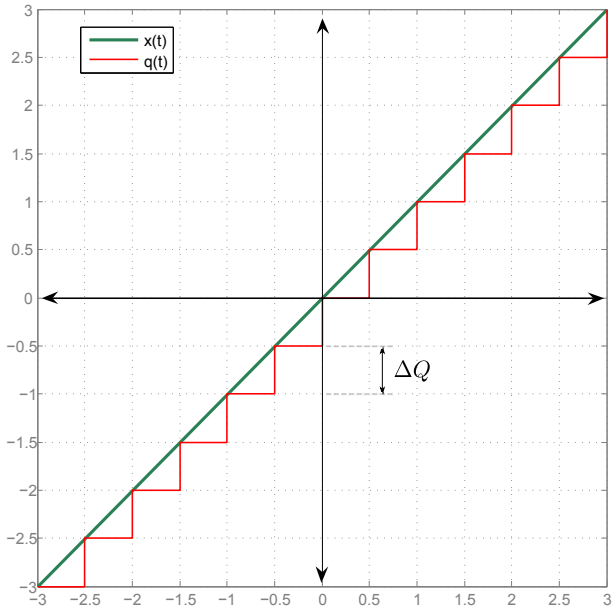

Figure 1: Quantization function $q(x)$ with quantum $\Delta Q = 0.5$

$$\begin{aligned} \dot{x}(t) &= q(t-1)\ ,\ t \geq 0 \\ q(t) &= 1\ ,\ -1 < t \leq 0 \end{aligned} \tag{4}$$

The modified system of Eq. (4) can be analytically solved.

We shall go through the solving procedure following Figure 2, where we plotted the solution $x(t)$ (dotted line), the quantized variable $q(t)$ (bold line), and its delayed version $q(t-1)$ (dash-dotted line).

Let us start the procedure at $t = t_0 = 0$ with $x(0) = q(0) = 1$. At this point, we have $\dot{x}(t=0) = q(-1) = 1$.

We may observe that $\dot{x}(t)$ does not change until $q(t-1)$ changes. Then, $x(t)$ evolves for a while growing with a constant slope of 1. Then at time $t = t_1 = 0.5$, we have $x(t = 0.5) = 1.5$ and the quantized variable $q(t)$ experiences its first change assuming a value of $q(t = 0.5) = 1.5$. This implies that $\dot{x}(t) = q(t-1)$ will remain unchanged until time $t = 1.5$.

At $t = t_2 = 1$, we have $x(t = 1) = 2$ and consequently, $q(t)$ changes its value again to $q(t = 1) = 2$. Then at $t = t_3 = 1.5$, the state reaches a value of $x(t = 1.5) = 2.5$, and the quantized state changes its value also to $q(t = 1.5) = 2.5$.

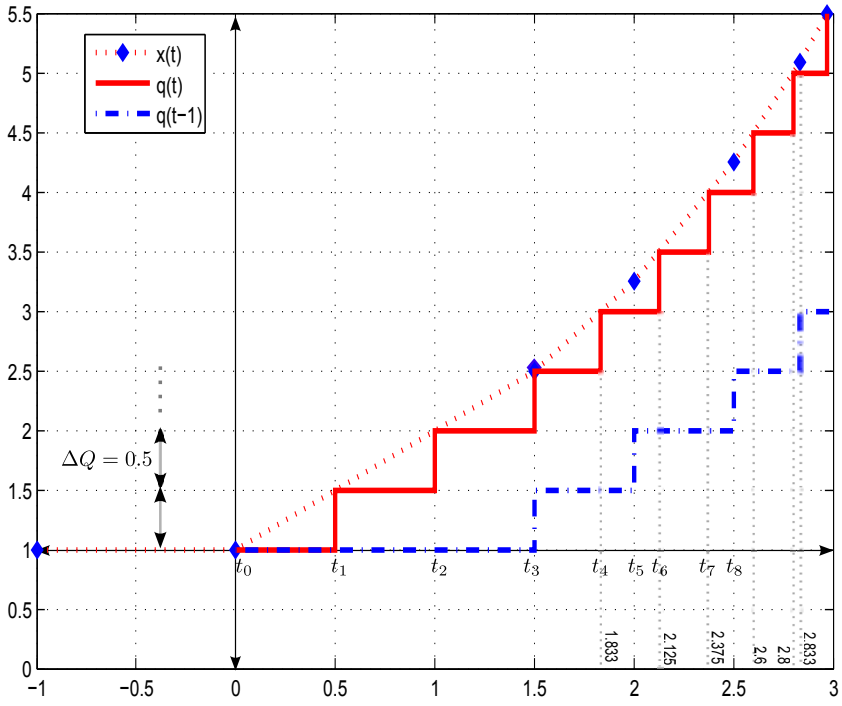

Figure 2: Solution of $x(t)$ for a DDE approximated by Eq. (4)

At time $t = 1.5$, the state derivative $\dot{x}(t)$ changes its value also, since $q(t)$ changed its value at $t = 1.5 - 1 = 0.5$). The new slope is $\dot{x}(t > 1.5) = q(0.5) = 1.5$, and now $x(t)$ grows faster.

After $\frac{\Delta Q}{\dot{x}(t_3)} = \frac{0.5}{1.5}$ units of time, i.e., at time $t_4 = t_3 + \frac{1}{3} = 1.8333$, the state reaches a value of $x(t_4) = 3$, and we have $q(t_4) = 3$.

The state derivative $\dot{x}(t)$ changes its value again at time $t = t_5 = 2$, and from that point on, we have $\dot{x}(t) = 2$, given that $q(2 - 1 = 1) = 2$. At time t_5, the state reaches a value of $x(t_5) = x(t_4) + 1.5 \cdot (t_5 - t_4) = 3.25$.

From this state value $x(t) = 3.25$ and with the new slope $\dot{x}(t) = 2$, the state reaches a value of $x(t = 2.125) = 3.5$ after $\frac{3.5-3.25}{2} = 0.125$ units of time, i.e., at time $t_6 = 2 + 0.125 = 2.125$. Then, $q(t)$ changes its value again, and the calculations continue in the same manner.

We can distinguish between two types of changes. At certain points in time: t_1, t_2, t_3, t_4, and t_6, the quantized state $q(t)$ changes its value. At other points in time: t_3 and t_5, it is the state derivative $\dot{x}(t)$ that changes its value. At time t_3, both types of changes occur simultaneously.

The observed behavior suggests that the approximated system of Eq. (4) can be described using a discrete-event model.

Although the procedure for simulating the approximated system of Eq. (4) is straightforward, it remains to be seen, how well the solution obtained in this way approximates the solution of the original system of Eq. (1). Moreover, although simulating this particular system was quite easy, we need to discuss if and if so how the proposed approach can be generalized to tackling broader classes of DDE systems.

In the next section, we shall introduce the Quantized State Systems (QSS), a family of numerical integration methods that formalize and generalize the intuitive procedure presented in this section.

2. Quantization–based Integration

We shall first introduce the idea and main characteristics of QSS methods applied to Ordinary Differential Equations (ODE). Then, we shall show that the systems approximated by QSS can be represented within the DEVS (Discrete Event Specification System) formalism framework.

After that in Section 3, we shall return to DDEs and study the requirements that arise when we try to extend the application of QSS methods from ODEs to DDEs. Whereas we can introduce the QSS methods for ODEs informally as these have already been introduced formally in other publications [11], the requirements and limitations of QSS methods for DDEs shall be introduced formally and rigorously in this paper.

2.1. QSS Methods for ODEs

Consider a set of time-invariant ODEs represented in their state-space form:

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)) \tag{5}$$

where $\mathbf{x}(t) \in \Re^n$ is the state vector, and $\mathbf{u}(t) \in \Re^w$ is a vector of known piecewise constant input trajectories.

The first order accurate QSS method (QSS1) simulates an approximated version of Eq. (5), which is called Quantized State System (QSS):

$$\dot{\mathbf{x}}(t) = f(\mathbf{q}(t), \mathbf{u}(t)) \tag{6}$$

where $\mathbf{q}(t)$ is a vector of quantized states resulting from the quantization of the state variables $x_j(t)$.

Each component of $\mathbf{q}(t)$ follows a piecewise constant trajectory, related with the corresponding component of $\mathbf{x}(t)$ by a hysteretic quantization function defined by:

$$q_j(t) = \begin{cases} x_j(t) & \text{if } |q_j(t^-) - x_j(t)| = \Delta Q_j \\ q_j(t^-) & \text{otherwise} \end{cases} \tag{7}$$

with $q_j(t_0) = x_j(t_0)$. Then, $q_j(t)$ results in a piecewise constant approximation of $x_j(t)$ that changes its value only when both trajectories differ by $\pm\Delta Q_j$. The magnitude ΔQ_j is called *quantum*. The relationship between x_j and q_j is depicted in Figure 3 (left), which differs from the quantization relationship presented in Fig.1 due to the introduction of hysteresis.

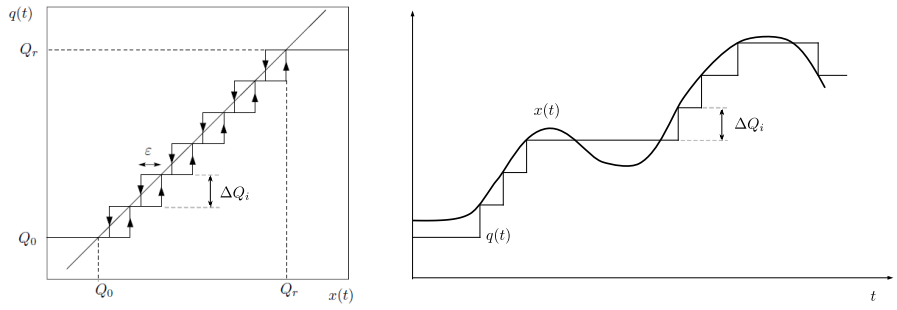

Figure 3: Quantization function with hysteresis (left) and hysteretic quantization with hysteresis width $\varepsilon = \Delta Q$ (right)

In Figure 3 (right), we show an example of the quantization of an arbitrary trajectory $x(t)$ applying the relationship of Eq. (7). The introduction of hysteresis in QSS1 avoids the appearance of infinitely fast oscillations that would prevent the simulation form advancing [5].

The QSS1 method can simulate *any* system with the structure of Eq. (5), offering the following properties:

- It preserves numerical stability (without involving any implicit formulae). The quantization process can be treated as a bounded perturbation on the original ODE, so the nonlinear stability can be studied by means of Lyapunov functions [11].

- It offers a global error bound, which guarantees that the numerical solution of a linear time–invariant analytically stable system will never

differ from its analytical solution by an amount larger than a computable finite value [5].

- It is intrinsically asynchronous: the different states update their values independently of each other at different time instants. This offers a significant advantage in terms of performance when dealing with sparse systems.

- It provides dense output, a feature particularly useful for asynchronous methods.

- It is very efficient at simulating across heavily discontinuous models due to the simplicity of the required root solving procedures when dense output is available. Also, due to the asynchronous features of the method, each discontinuity occurrence can be efficiently handled.

While classical numerical ODE solvers lead to discrete-time approximations, i.e., sets of difference equations, the QSS1 method does not. The QSS approximation of Eq. (6) results in a discrete-event system that can be modeled using the DEVS formalism [21, 22].

2.2. DEVS Formalism

A DEVS model processes an input event trajectory, and, based on that trajectory and the initial state, produces an output event trajectory.

The behavior of an *atomic* DEVS model is formally defined by the structure:

$$M = (X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta) \tag{8}$$

where

- X is the input event set, i.e., the set of all possible input event values;

- Y is the output event set;

- S is the state value set;

- δ_{int}, δ_{ext}, λ, and ta are functions that define the model dynamics.

Each possible state $s \in S$ has a time advance function $ta(s) : S \rightarrow \Re_0^+$ associated with it that indicates, how long the state will retain its current value in absence of external input events. Assuming that the state s assumes a value of s_1 at time t_1, the model undergoes an *internal transition* $ta(s_1)$ time units later, i.e., at time $t_1 + t_a(s_1)$, changing the value of the state to $s_2 = \delta_{int}(s_1)$. The function $\delta_{int}(s) : S \rightarrow S$ is called *internal transition function*. While the model undergoes an internal transition, it also produces an output event $y_1 \in Y$ calculated with the *output function* $\lambda : S \rightarrow Y$. The functions δ_{int}, ta, and λ define the *autonomous behavior* of a DEVS model. If an input event $x_1 \in X$ arrives while the state has been at a value of s_2 for e seconds, the model undergoes an external transition, changing the value of the state instantaneously to $s_3 = \delta_{ext}(s_2, e, x_1)$. The function $\delta_{ext}(s, e, x) : S \times \Re_0^+ \times X \rightarrow S$ is called *external transition function*. It is a function of the previous state value s, of the elapsed time (the time spent in that state already) e, and the value of the input event that triggered the transition. During an external transition, no output event is produced.

Any system that undergoes a finite number of state changes within any finite time interval can be modeled using an atomic DEVS model.

DEVS models can be coupled by interconnecting their inputs and outputs in a block diagram fashion. Since the DEVS formalism is closed under coupling, coupled DEVS models can be hierarchically coupled with other coupled and/or atomic DEVS models to represent yet more complex systems.

2.3. QSS and DEVS

Figure 4 shows the block diagram representation of Eq. (6). In grey, the figure shows the block diagram divided into two types of subsystems: The F_i (static functions) and the HQI (hysteretic quantized integrator) blocks. The F_i blocks correspond to the functions that compute the right hand side of Eq. (6), while each HQI block is composed by an integrator and a hysteretic quantization function.

Each F_i block has piecewise constant input trajectories, q_j and u_j, and it computes a piecewise constant output trajectory, $\dot{x}_j$. Similarly, the HQI blocks have piecewise constant input trajectories, $\dot{x}_j$, and calculate piecewise constant output trajectories, q_j.

Since piecewise constant trajectories can be represented as sequences of events, and recalling that atomic DEVS models can represent any system that receives and produces these types of sequences, each block in Fig.4 can be represented by a DEVS atomic model.

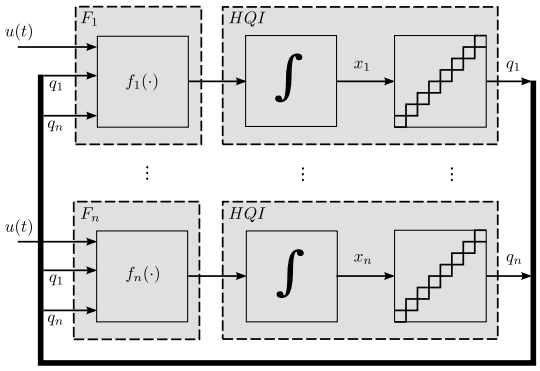

Figure 4: Block diagram representation of a QSS. Arrow references: thin (single component signal), bold (multiple component bus).

The DEVS models for static functions, F_i, and hysteretic quantized integrators, HQI, are quite simple and have been specified in [5].

Then, coupling the corresponding DEVS atomic models of F_i and HQI in accordance with the block diagram of Fig.4, we obtain a coupled DEVS model equivalent to Eq. (6). This DEVS model can be used to exactly simulate the behavior of the QSS1 approximation of the original ODE.

2.4. Higher–order QSS

The QSS1 mechanism described so far is a first–order approximation algorithm showing a linear relationship between the desired precision and the number of steps needed to simulate a given system. It exhibits poor performance for levels of accuracy as they are required for most practical (engineering) applications.

For this reason, higher–order QSS methods have also been developed, in particular a second–order QSS (QSS2) and a third–order QSS (QSS3) method. The basic idea behind higher–order QSS algorithms is to change the quantization function that calculates $q_j(t)$ to preserve higher–order information about the original signals $x_j(t)$.

In the QSS1 quantization, all higher time derivatives are discarded, and a piecewise constant $q_j(t)$ approximation is obtained. In QSS2, the second time derivative of $x_j(t)$ is preserved by producing piecewise–linear $q_j(t)$ outputs, while in QSS3, the third time derivative of $x_j(t)$ is preserved by producing piecewise–parabolic $q_j(t)$ outputs.

The quantization $q_j(t)$ for an arbitrary input signal $x_i(t)$, using a quantum ΔQ_j, is depicted in Figure 5 for the QSS2 and QSS3 methods.

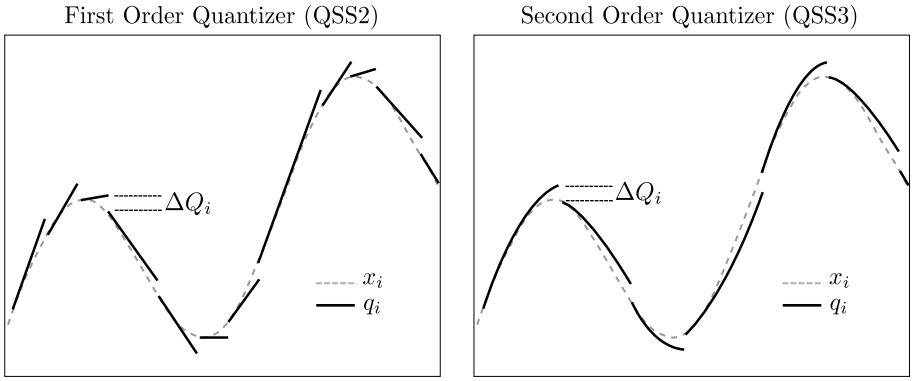

Figure 5: First order quantization function (QSS2, left) and second order quantization function (QSS3, right)

The formal definitions of the QSS2 and QSS3 methods are the same as that of QSS1, i.e., these integrators approximate the ODE system of Eq. (5) by the QSS of Eq. (6), but now the quantized states q_j are computed from x_j as shown in Fig.5.

In terms of the performance concerns mentioned earlier, higher–order QSS methods show an important property: for QSS2, the number of steps grows with the square root of the accuracy, and for QSS3, the number of steps grows with the cubic root of the precision. As a remark, QSS1, QSS2, and QSS3 share the same theoretical properties, at least for linear systems.

The DEVS implementation of QSS2 and QSS3 is analogous to that of QSS1. They follow the block diagram structure of Fig.4, but the atomic DEVS models for the static functions F_i and the quantized integrators HQI are different, as they take into account higher–order derivatives of their input and ouput trajectories.

For instance in QSS2, the blocks F_i and HQI have piecewise–linear input and output trajectories. Consequently, the DEVS models for F_i and HQI receive and produce events carrying two coefficients representing the values and slopes of the sections of the corresponding piecewise–linear trajectories.

3. QSS Methods for Delay Differential Equations

In this section, we present the Delay Quantized State System (DQSS) methods for solving DDEs.

DDEs are characterized by their ability to reference in the model past

values of state variables. DDEs express past information by means of one or more *delay functions* of the general form $\tau_j(\cdot)$. The delays may be either constant with $\tau_j(\cdot) = \tau_j$ (constant–delay DDEs), or they may be functions of time with $\tau_j(\cdot) = \tau_j(t)$ (time–dependent DDEs), and finally, they may also depend on system states with $\tau_j(\cdot) = \tau_j(x_1(t), ..., x_n(t), t)$ (state–dependent DDEs).

We can reformulate the general ODE representation in Eq. (5) to incorporate such delayed information and then apply a quantization function to the state variables, thus obtaining what we shall call *DQSS methods*.

3.1. DQSS Definition

Consider the following DDE in its State Equation System (SES) vectorial representation:

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{x}(t - \tau_1(\mathbf{x}, t)), ..., \mathbf{x}(t - \tau_m(\mathbf{x}, t)), \mathbf{u}(t)) \tag{9}$$

where $\tau_j(\mathbf{x}, t) \geq 0$ for all $t, \mathbf{x}$ and for all j, and the rest of the variables and functions are defined like in the ODE case of Eq. (5).

We also consider the initial history:

$$\mathbf{x}(t) = \mathbf{S}(t) \quad ; \quad -\tau_{max} < t < 0 \tag{10}$$

where τ_{max} is a positive constant that satisfies $t - \tau_i(\mathbf{x}, t) > -\tau_{max}$ for all $t, \mathbf{x}$ and for all i.

The DQSSn method then simulates the delayed quantized state system:

$$\dot{\mathbf{x}}(t) = f(\mathbf{q}(t), \mathbf{q}(t - \tau_1(\mathbf{q}, t)), ..., \mathbf{q}(t - \tau_m(\mathbf{q}, t)), \mathbf{u}(t)) \tag{11}$$

where each component of the quantized state vector $\mathbf{q}(\mathbf{t})$ is related to the corresponding component of the state vector $\mathbf{x}(\mathbf{t})$ by a hysteretic quantization function of order n.

3.2. DEVS representation of DQSS

Figure 6 shows the block diagram representation of the delayed quantized state system of Eq. (11).

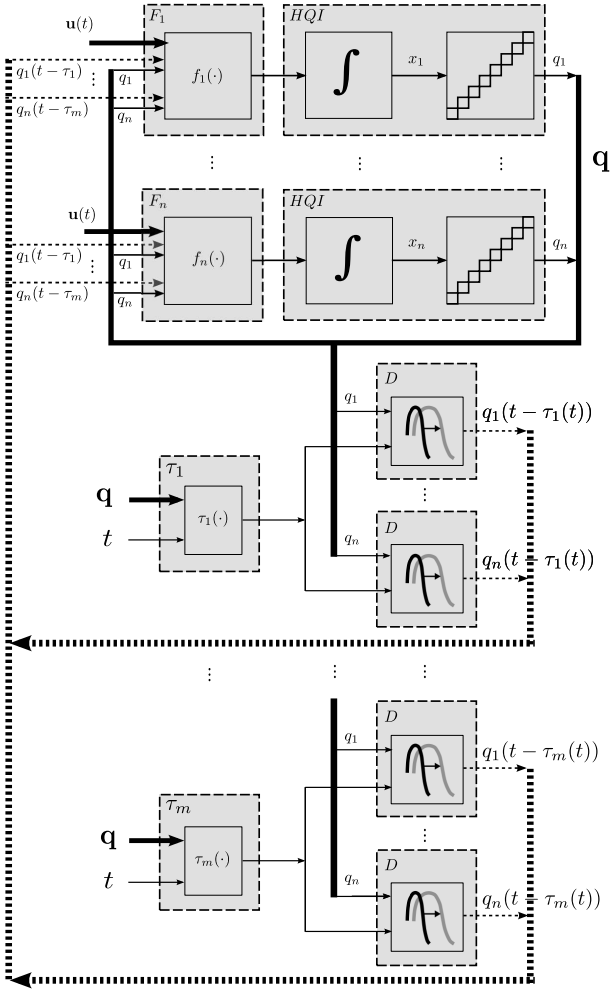

Figure 6: Block diagram representation of a DQSS. Box references: continuous (regular block), dotted (DEVS equivalent model). Arrow references: thin (single component signal), bold (multiple component bus), continuous (non-delayed), dotted (delayed)

In order to implement this method using DEVS, we can proceed following the same idea as in the ODE case, i.e. we build a coupled DEVS model with DEVS equivalents of the different system blocks.

The block diagram of Figure 6 shows now three types of DEVS models: *Hysteretic Quantized Integrators HQI*, which compute q_j; *Static Functions* F_j and τ_i, which compute $\dot{x}_j$ and τ_i, respectively; and the *Delays* D, which compute $\mathbf{q}(t - \tau_i)$.

We already have available DEVS models for quantized integrators and static functions. All we still have to do is to find a DEVS representation for the delay blocks D. The first step towards this aim is to define mathematically the operation of a generic delay block (Figure 7, left). After that, we shall provide the DEVS equivalent model D (Figure 7, right) that implements the obtained mathematical definition.

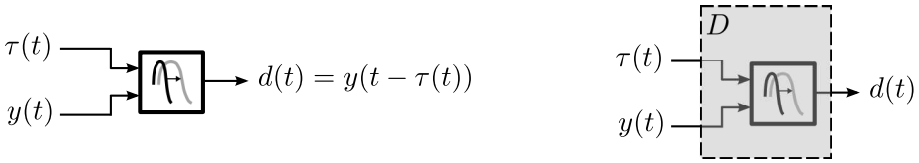

Figure 7: Delay block (left). Delay DEVS model D (right)

3.3. Analytical Calculation of Delayed Signal

We wish to compute $d(t) = y(t - \tau(t))$, where $y(t)$ and $\tau(t)$ are piecewise polynomial trajectories.

We can express $y(t)$ as a sequence of polynomial segments valid on adjacent time intervals given by:

$$\begin{aligned} &y(t) = y_{0,k} + y_{1,k}(t - t_k^y) + y_{2,k}(t - t_k^y)^2 + \dots \\ &t_k^y \leq t < t_{k+1}^y \end{aligned} \tag{12}$$

where t_k^y is a sequence of time instants with $t_k^y < t_{k+1}^y$, and $y_{0,k}, y_{1,k}, y_{2,k}, \dots$ are the corresponding sequences of polynomial coefficients.

Note that under this representation, the polynomial coefficients can change their values only at the beginning of each interval (t_k^y). The coefficients remain constant throughout any time interval $t \in [t_k^y, t_{k+1}^y)$.

Similarly, we express $\tau(t)$ as the following sequence:

$$\begin{aligned} &\tau(t) = \tau_{0,j} + \tau_{1,j}(t - t_j^\tau) + \tau_{2,j}(t - t_j^\tau)^2 + \dots \\ &t_j^\tau \leq t < t_{j+1}^\tau \end{aligned} \tag{13}$$

where t_j^τ is a sequence of time instants with $t_j^\tau < t_{j+1}^\tau$, and $\tau_{0,j}, \tau_{1,j}, \tau_{2,j}, \ldots$ are the corresponding sequences of polynomial coefficients.

The delayed signal $d(t)$ will also follow a piecewise polynomial trajectory that can be written as:

$$\begin{aligned} &d(t) = d_{0,\ell} + d_{1,\ell}(t - t_\ell^d) + d_{2,\ell}(t - t_\ell^d)^2 + \ldots \\ &t_\ell^d \leq t < t_{\ell+1}^d \end{aligned} \tag{14}$$

and we need to compute the time sequences t_ℓ^d and the coefficient sequences $d_{0,\ell}$, $d_{1,\ell}$, $d_{2,\ell}$, ...

Note that there is not predefined relationship between the time instants t_k^y and t_j^τ, which can evolve independently. The resulting timing sequence t_ℓ^d will depend on t_k^y and t_j^τ in a manner that we shall now describe.

Let t^* be an arbitrary instant of time, for which we want to find the expression of $d(t^*)$. Let j be the integer number that satisfies

$$t_j^\tau \leq t^* < t_{j+1}^\tau$$

and let k be the integer number satisfying

$$t_k^y \leq t^* - \tau(t^*) < t_{k+1}^y$$

It should be noted that, whereas the DDE makes use of values of state variables from the past, the delay time $\tau(t)$ is calculated now, i.e., at the time when the past state value is being accessed, not when it was computed.

Note further that $d(t^*)$ will be related to both the j–th polynomial section of $\tau(t)$ and the k–th polynomial section of $y(t)$. The beginning of the ℓ–th polynomial section of $d(t)$ can correspond to either a section change in $\tau(t)$, or a section change in $y(t)$, or both simultaneously. Thus, t_ℓ^d will be related to either t_j^τ, t_k^y, or t_{k+1}^y.

Let us define:

$$\begin{aligned} t_a &= \max\left(t \mid (t - \tau(t) = t_k^y) \wedge (t_j^\tau < t \leq t^*) \right) \\ t_b &= \max\left(t \mid (t - \tau(t) = t_{k+1}^y) \wedge (t_j^\tau < t \leq t^*) \right) \end{aligned} \tag{15}$$

in case the equations $t - \tau(t) = t_k^y$ and/or $t - \tau(t) = t_{k+1}^y$ have solution in the given interval. Otherwise, we set $t_a = -\infty$ and/or $t_b = -\infty$.

Then, we can compute the starting time $t_\ell^d < t^*$ of the ℓ–th polynomial section to which $d(t^*)$ belongs as:

$$t_\ell^d = \max(t_a, t_b, t_j^\tau) \tag{16}$$

The next adjacent section of $d(t)$ will start at $t_{\ell+1}^d$. In order to find its value, let us define:

$$\begin{aligned} t'_a &= \min\left(t \mid (t - \tau(t) = t_k^y) \wedge (t^* < t \leq t_{j+1}^\tau)\right) \\ t'_b &= \min\left(t \mid (t - \tau(t) = t_{k+1}^y) \wedge (t^* < t \leq t_{j+1}^\tau)\right) \end{aligned} \tag{17}$$

in case the equations $t - \tau(t) = t_k^y$ and/or $t - \tau(t) = t_{k+1}^y$ have solution in the given interval. Otherwise, we set $t'_a = \infty$ and/or $t'_b = \infty$.

The next point in time $t_{\ell+1}^d > t^*$ belonging to the sequence of the delayed signal $d(t)$ can be computed as:

$$t_{\ell+1}^d = \min(t'_a, t'_b, t_{j+1}^\tau) \tag{18}$$

Following these definitions, for any time instant belonging to the continuous interval $t \in [t_\ell^d, t_{\ell+1}^d)$, the trajectories $y(t - \tau(t))$ and $\tau(t)$ will not have discontinuities, and $d(t)$ will satisfy Eq. (14).

In Figure 8, we depict the main timing components and polynomial sequences discussed so far that uniquely define $d(t) = y(t - \tau(t))$. From now on, we shall refer to sets of polynomial coefficients that define a signal in a given interval as column vectors. Also, taking into account that we are interested in methods up to third order (QSS3), we shall only handle polynomial coefficients up to quadratic terms.

Thus, we shall deal with vectors

$$\mathbf{d}_\ell = \begin{bmatrix} d_{0,\ell} \\ d_{1,\ell} \\ d_{2,\ell} \end{bmatrix}, \quad \mathbf{y_k} = \begin{bmatrix} y_{0,k} \\ y_{1,k} \\ y_{2,k} \end{bmatrix}, \text{ and } \tau_\mathbf{j} = \begin{bmatrix} \tau_{0,j} \\ \tau_{1,j} \\ \tau_{2,j} \end{bmatrix}$$

Either Eq. (16) or Eq. (18) can be used to define the time sequence t_ℓ^d of the delayed function $d(t)$. To complete its analytical expression, we need

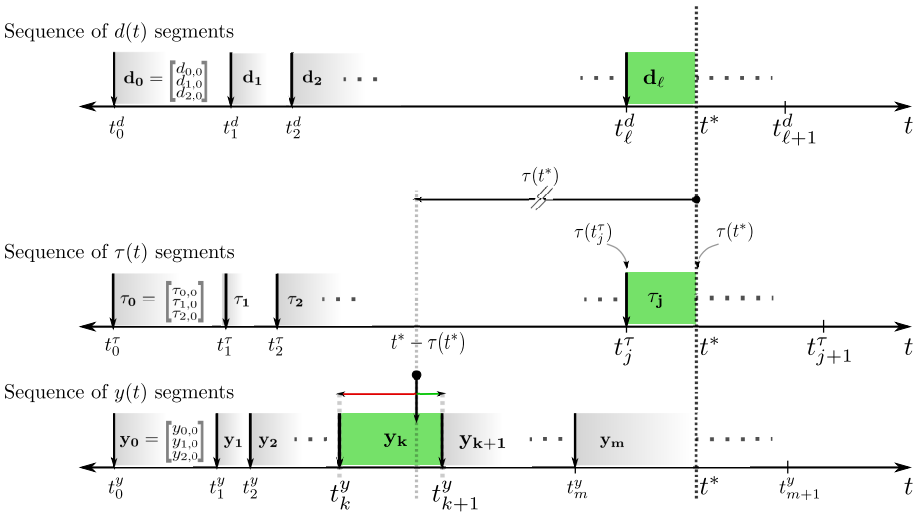

Figure 8: DQSS algorithm based on polynomial segments. Sequences of time instants and polynomial coefficients.

to also compute the sequence of coefficients $d_{0,0} \dots d_{0,\ell}$, $d_{1,0} \dots d_{1,\ell}$, and $d_{2,0} \dots d_{2,\ell}$. From Eqs. (12), (13), and (14), we can write:

$$
\begin{aligned}
d(t) &= d_{0,\ell} + d_{1,\ell}(t - t_\ell^d) + d_{2,\ell}(t - t_\ell^d)^2 + \dots = \\
&= y(t - \tau(t)) = \\
&= y_{0,k} + y_{1,k}[(t - \tau(t)) - t_k^y] + y_{2,k}[(t - \tau(t)) - t_k^y]^2 + \dots = \\
&= y_{0,k} + y_{1,k}[(t - (\tau_{0,j} + \tau_{1,j}(t - t_j^\tau) + \tau_{2,j}(t - t_j^\tau)^2 + \dots)) - t_k^y] + \\
&\qquad + y_{2,k}[(t - (\tau_{0,j} + \tau_{1,j}(t - t_j^\tau) + \tau_{2,j}(t - t_j^\tau)^2 + \dots)) - t_k^y]^2 + \dots
\end{aligned}
$$

Removing all coefficients of orders higher than 2 and eliminating the j, k, and ℓ sub-indices of the corresponding sequences to avoid notational clutter, we obtain the following expressions for the coefficients:

$$
\begin{aligned}
d_0 =& y_0 - y_1[\tau_0 + \tau_1(t^d - t^\tau) + \tau_2((t^\tau)^2 - 2t^\tau t^d + (t^d)^2) + t^y - t^d] + \\
& y_2[\tau_0^2 - \tau_0(2\tau_1(t^\tau - t^d) - \tau_2(2(t^\tau)^2 - 4t^\tau t^d + 2(t^d)^2) - 2t^y + 2t^d) + \\
& \tau_1^2 t^\tau - (t^d)^2 + 2\tau_1(t^d - t^\tau)(\tau_2((t^\tau)^2 - 2t^\tau t^d + (t^d)^2) + t^y - t^d) + \\
& \tau_2^2((t^\tau)^2 - 2t^\tau t^d + (t^d)^2)^2 + 2\tau_2((t^\tau)^2 - 2t^\tau t^d + (t^d)^2)(t^y - t^d) + \\
& (t^y)^2 - 2t^y t^d + (t^d)^2] \\
d_1 =& - y_1[\tau_1 + 2\tau_2(t^d - t^\tau) - 1] + \\
& y_2 2[\tau_0 + \tau_1(t^d - t^\tau) + \tau_2((t^\tau)^2 - 2t^\tau t^d + (t^d)^2) + t^y - t^d] \cdot \\
& [\tau_1 + 2\tau_2(t^d - t^\tau) - 1] \\
d_2 =& - y_1\tau_2 + y_2[2\tau_0\tau_2 + \tau_1^2 - \tau_1(6\tau_2(t^\tau - t^d) + 2) + 6\tau_2^2((t^\tau)^2 - 2t^\tau t^d + (t^d)^2) + \\
& 2\tau_2(2t^\tau + t^y - 3t^d) + 1]
\end{aligned}
\tag{19}
$$

3.4. DEVS–based Algorithm to Obtain $d(t) = y(t - \tau(t))$

Due to the event–like nature of quantized state systems, the task of delaying segments is based on a queueing–like activity. Incoming $y(t)$ events (polynomial representations of segments) are received, enqueued, eventually queried (as many times as required according to what the evolution of $\tau(t)$ demands), processed by an algorithm, and sent out as $d(t)$ events.

In Section 1.1, we presented an informal iterative procedure using a simple backward lookup to obtain $q(t-\tau)$ from $q(t)$. The procedure was fairly simple to follow and did not present algorithmic complexities. The system solved in that example is a very simple one, given that it features a single constant delay and that the integration method used to solve it was of first order.

Although there are many models of practical interest that involve constant delays, models with time– and state–dependent delays are very important also, and they turn out to be the most challenging DDE problems for classical time–slicing methods.

The DEVS Delay models D must accept variable delay information and must implement the required mechanism to search dynamically back in time over the stored $y(t)$ segments in accordance with $\tau(t)$ as discussed in Section 3.3.

We shall now describe a general implementation of a DEVS Delay model D that follows the mathematical specification provided in Section 3.3. The

main activities of D will be mapped to the description of Figure 8 and can be summarized as follows:

- **DEVS external transitions**[1]:
  - Upon receiving a new $y(t)$ segment at time t:

    Enqueue the new segment as a vector $\mathbf{y_m}$ with time-stamp $t_m^y = t$. It contains the n polynomial coefficients that describe the m–th arrived segment of the trajectory $y(t)$.
  - Upon receiving a new $\tau(t)$ segment at time t:

    Update the previously stored vector $\tau_\mathbf{j}$ with the new segment and time-stamp t_j^τ. It contains the n polynomial coefficients that describe the j–th arrived segment of the trajectory $\tau(t)$.

    Lookup backward in the queue the k–th segment $\mathbf{y_k}$ with time-stamp t_k^y "pointed at" by $\tau(t)$, i.e., satisfying $t_k^y \leq t - \tau(t) < t_{k+1}^y$.
  - After having processed all of the external events that arrived at time t:

    Calculate the next time instant $t_{\ell+1}^d$ according to Eq. (18) at which the validity of the current output $\mathbf{d}_\ell$ expires.

    Schedule an internal transition to take place at time $t = t_{\ell+1}^d$ for calculating $t_{\ell+2}^d$, assuming that no external transition interfered.

- **DEVS internal transitions**:
  - Upon expiration of the current output segment $\mathbf{d}_\ell$ at time $t = t_{\ell+1}^d$:

    Recalculate and Output a new vector $\mathbf{d_{\ell+1}}$ with time-stamp $t_{\ell+1}^d$ and polynomial coefficients as defined by Eq. (19).

[1]Upon receiving multiple new segments $y(t)$ and/or $\tau(t)$ simultaneously, consecutive external transitions will be triggered to process segments independently, one at a time. The processing order will be that of the segments' arrival order at model D, which will in turn be determined by the priorities of the models generating them. The resulting final state for model D will correspond to the last processed segment. This mechanism does not affect the correctness of the calculated output segments $d(t)$.

Calculate and Schedule a new internal transition to take place at time $t = t^d_{\ell+2}$ for calculating $t^d_{\ell+3}$, assuming that no external transition interfered.

The pseudo–algorithm just described can be easily mapped onto a DEVS atomic model. In the next section, we shall provide details of our implementation of the Delay model using the PowerDEVS tool.

3.5. PowerDEVS Implementation.

We developed a new block named *Delay* for the *Continuous Library* of PowerDEVS. It consists of an atomic DEVS model that implements the algorithm depicted in Figure 8.

The Delay block has two input ports and one output port. The top input port receives the variable delay information $\tau(t)$, and the bottom input port accepts the signal $y(t)$ to be delayed. The output port emits the resulting delayed signal $d(t) = y(t - \tau(t))$.

In Figure 9 we show the PowerDEVS tool with a Delay block and its user interface window for parametrization.

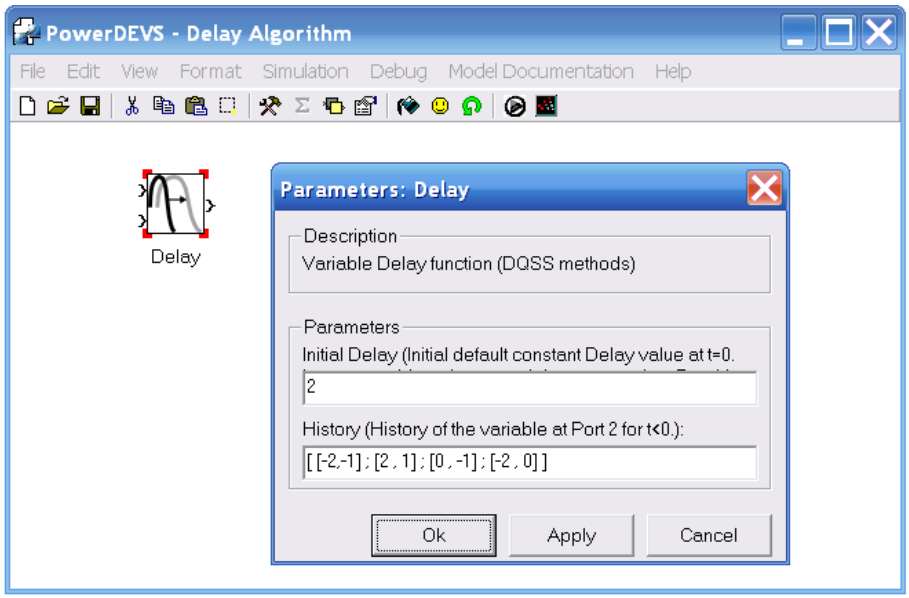

Figure 9: PowerDEVS Delay Block user interface.

Two parameters can be configured: *Initial Delay* and *History*. Both parameters use Scilab expressions or variables.

By configuring *Initial Delay*, we provide the block with the value for $\tau_0 = \tau(t_0)$ at the beginning of the simulation, in case there is no signal present at the top input port at time t_0. For instance in the example of Figure 9, the block is configured with $\tau_0 = 2$, and consequently, it will start emitting $d(t_0) = y(t_0 - \tau_0) = y(-2)$.

Values for $y(t \leq 0)$ are entered through the *History* parameter. The history information is entered as a matrix $H(r, c)$ of size $r \times c$ with r rows and c columns. Each column defines a polynomial segment and its time stamp. The first row $H(1, \cdot)$ represents the sequence of negative time stamps for each segment. For instance in the example of Figure 9, we have $H(1, \cdot) = [-2, -1]$, meaning that the block is provided with two history segments starting at $t = -2$ and $t = -1$.

For a given column $H(\cdot, c)$, the first row defines the time stamp, and the rows from 2 to r represent the polynomial coefficients of orders $0, 1, 2, \ldots$.

For instance in the example of Figure 9, the oldest segment at $t = -2$ has polynomial coefficients $2, 0, -2$ and will remain valid until the beginning of the next segment, starting at $t = -1$, with polynomial coefficients $1, -1, 0$. With these parameters, the block will start emitting $d_0 = y(-2) = 2$ at t_0.

Note that, it might happen that the *Initial Delay* parameter and/or $\tau(t)$ point at an instant t_{old} in the past, for which there is no segment defined in the *History* parameter, i.e., when $t_{old} < t_{min} = min(h_i | h_i \in H(1, i), i = 1, \ldots, c)$. Under that condition, the Delay block extrapolates the oldest segment defined in H backward in time, so that its new time stamp t'_{min} coincides with t_{old}. Each time this particular situation occurs, a warning message is written to a log file.

The Delay block can be connected to any PowerDEVS block from the Continuous Library and/or Hybrid Library to build complex hybrid systems with delays.

4. Theoretical Properties of QSS Methods for DDEs.

4.1. Stability and Convergence. Constant Delays.

With the next theorem, we provide the conditions to guarantee the applicability of DQSS methods to solve general constant–delay DDEs.

We prove that, if the original system is stable, then the simulation of the DQSS equivalent system will preserve numerical stability, and also the numerical solution can be made as accurate as desired by making the selected

quantum $\mathbf{\Delta Q}$ approach zero. We only require that the original system has an asymptotically stable solution for zero initial conditions.

Theorem 1. *Consider the following DDE:*

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + \sum_{i=1}^{m} A_i \mathbf{x}(t - \tau_i) \tag{20}$$

and assume that the analytical solution $\phi_{\mathbf{a}}(t)$ for the trivial initial condition $\phi_{\mathbf{a}}(t) = \mathbf{x}(t) = 0$, $t \leq 0$ is asymptotically stable. Then,

1. *The error committed by any QSS method in the simulation of Eq.* (20) *is bounded for all $t \geq 0$, for any quantum $\mathbf{\Delta Q}$ adopted, and for any initial condition $\mathbf{x}(t \leq 0)$.*
2. *The QSS approximation converges to the analytical solution, i.e., the global error goes to zero when the quantum goes to zero.*

Proof:. See Appendix B

4.2. Stability and Convergence. Time and State Dependent Delays

With the next theorem, we extend the class of DDEs that can be simulated with DQSS methods to those with time– and state–dependent delays, guaranteeing the applicability of DQSS methods to solve general DDEs of these classes.

As before, we prove that the simulation of the DQSS equivalent of an originally stable system will preserve numerical stability, and also the numerical solution can be made as accurate as desired by only making the selected quantum $\mathbf{\Delta Q}$ approach zero.

But in this case, we require a more stringent property to the original system. We ask the DDE system to be an *Input to State Stable (ISS)*[2] system. This means that, if the system is perturbed with an arbitrary bounded input, then the evolution of all states will be also bounded.

Theorem 2. *Consider the DDE*

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{x}(t - \tau_1(\mathbf{x}, t)), t) \tag{21}$$

[2]See Appendix A for a definition of Input to State Stability

where function $\mathbf{f}$ *is locally Lipschitz in the first two arguments and piecewise continuous on* t*, and* $\tau_1(\cdot)$ *is locally Lipschitz in* $\mathbf{x}$ *and piecewise continuous on* t*. Let* $\phi_a(t)$ *be the analytical solution from a given initial history* $\phi_a(t<0)$*. Suppose that the analytical solution is bounded, and assume also that the forced system*

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{x}(t-\tau_1(\mathbf{x},t)), t) + \mathbf{u}(t) \tag{22}$$

is Input to State Stable along the solution $\phi_a(t)$*.*

Let $\phi(t)$ *be the solution of the DQSS approximation*

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{q}(t), \mathbf{q}(t-\tau_1(\mathbf{q},t)), t) \tag{23}$$

with initial history $\phi(t<0) = \phi_a(t<0)$*.*

Then, provided that the quanta ΔQ_j*,* $j = 1, \ldots, n$ *are sufficiently small, the error* $\mathbf{e}(t) \triangleq \phi(t) - \phi_a(t)$ *is bounded for all* $t > 0$*. Moreover,* $\mathbf{e}(t) \to 0$ *as* $\Delta Q_j \to 0$*.*

Proof:. See Appendix C

Remark:. Theorem 2 holds for any DDE of the form of Eq.(21), which has a single delay function τ_1. However, it can be extended in a straightforward way to DDEs with multiple delay functions, such as Eq.(9).

5. Experimental Results

We tested and bench-marked the solution of some typical DDE systems taken from the open literature by performing simulations using the new DQSS–based algorithms and comparing those to the results obtained using discrete–time based solvers developed and published previously by other authors [18, 19].

We implemented the Delay block described in Section 3.3 as a DEVS model in PowerDEVS, a DEVS–based simulation platform specially adapted to simulating hybrid systems based on QSS methods. PowerDEVS has a graphical user interface similar to that of Matlab/Simulink, which permits editing block diagrams. The atomic DEVS description is programmed in C++ and can be easily edited with the PowerDEVS atomic editor tool. The Delay block is the only new model required for simulating DDEs with PowerDEVS, given that the tool already comes with static functions and hysteretic quantized integrators. The new Delay block is now part of the standard PowerDEVS library of continuous models.

5.1. Example 1

Let us consider, as a first example, the system reported as *Example 1* in [18] characterized by the DDE of Eq. (24). The DDE is to be simulated for $t \in [0, 5]$ with history $x_1(t) = 1$, $x_2(t) = 1$, and $x_3(t) = 1$ for $t \leq 0$.

$$\begin{aligned} \dot{x}_1(t) &= x_1(t-1) \\ \dot{x}_2(t) &= x_1(t-1) + x_2(t-0.2) \\ \dot{x}_3(t) &= x_3(t) \end{aligned} \tag{24}$$

This system has three state variables x_1, x_2, and x_3 and two constant delays $\tau_1 = 1, \tau_2 = 0.2$. It represents a linear DDE system with constant history.

In order to model the DQSS approximation of Eq. (24), we used the block–oriented graphical user interface of PowerDEVS, as depicted in Figure 10. It can be seen that two Delay blocks were needed, one for obtaining $x_1(t - \tau_1)$, and another for obtaining $x_2(t - \tau_2)$. Constant generator blocks are used to produce the constant continuous delay signals, which are connected to the top input port of the Delay blocks.

The simulation results are shown in Figure 11.

The system is analytically unstable, i.e., the trajectories grow exponentially over time.

Average performance results for sets of 30 simulation runs are provided in Table 1. DQSS3, the third-order accurate QSS solver coded in PowerDEVS is compared with dde23, another third-order accurate discrete-time algorithm coded in Matlab. Identical error tolerance values were applied in both cases. A relative accuracy of 10^{-3} was requested. For values of state variables very close to zero, $|x_i| < 10^{-6}$, the relative error tolerance is replaced by an absolute error tolerance of 10^{-6}.

We compared both the number of function evaluations and the execution times. The number of function evaluations cannot be compared one to one, since DQSS3 is an asynchronous algorithm, i.e., performs function evaluations independently for each state variables, whereas dde23 is a synchronous algorithm that performs function evaluations on all state variables together. Consequently, three separate function evaluations of DQSS are needed when simulating a third-order model to update all three state variables, whereas only one function evaluation is needed when using dde23. For this reason, we multiplied the reported number of function evaluations of the dde23 and

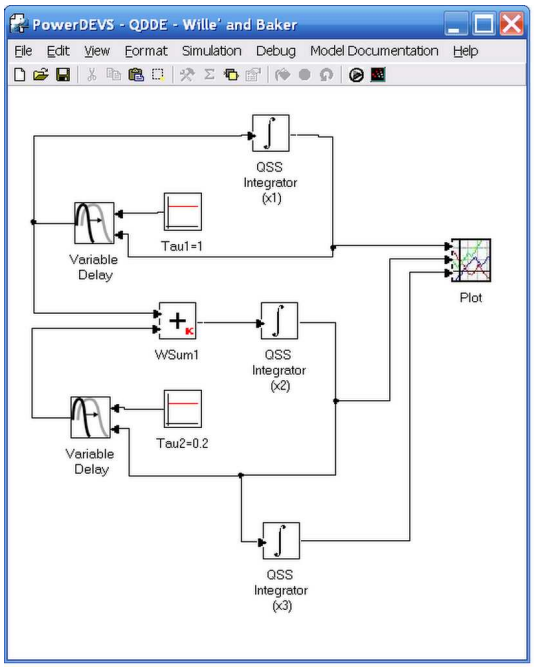

Figure 10: Model of a DDE implemented in the PowerDEVS tool

dde_solver algorithms by the order of the system in the tables to get comparable numbers.

Both PowerDEVS and Matlab perform partial compilations of the solver algorithms and/or the models. However, neither environment generates a flat program that is compiled and executed. Consequently, both programs experience similar overheads, and therefore, the execution times can be meaningfully compared to each other.

In PowerDEVS, the algorithm itself is so fast that its weight relative to the initialization process turned out to be negligible in this example. The 0.0158 seconds of execution time reported correspond to the initialization process in all cases. However, since the system to be simulated in analytically unstable, the simulation time cannot be extended much further without causing overflow.

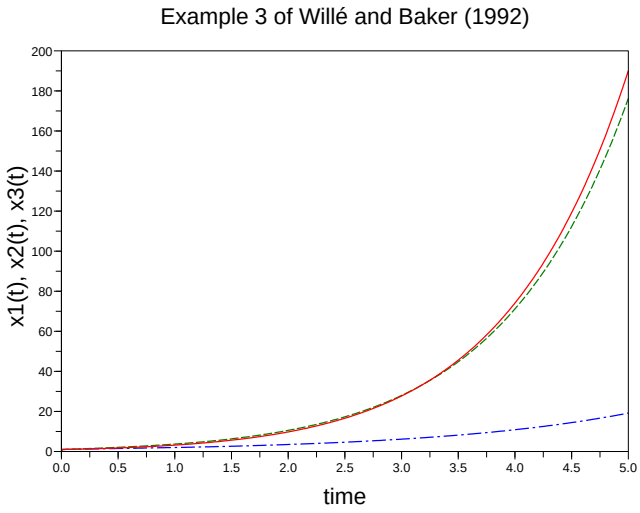

Figure 11: Solution of Example 1 [20]

5.1.1. Error Analysis

There is an important difference between DQSS and dde23 in how they interpret the error tolerance values. Whereas dde23 interprets these values as a measure of the local integration error in a single integration step, DQSS interprets these values as a measure of the global integration error across the entire simulation.

For this reason, it is important to compare both solutions to a reference solution and measure the true simulation errors generated by the solvers.

In order to obtain a reference solution, we ran the simulation using both solvers with accuracy requirements that were more stringent than the ones used before by three orders of magnitude. Comparing the results obtained in this way, we could indeed confirm that they are in agreement with an accuracy that is much better than the one requested in our original simulations.

We now report the absolute errors obtained, i.e., the deviations of the solutions of the original simulations from the reference solution. We report in Table 2 the maximum absolute error (max), and the average squared error (rms).

Let us interpret these results. $x_1(t)$ grows, during the simulation, to a value of approximately 20. Thus, with a relative error tolerance of 10^{-3},

Simulation Time	Method	# Function Evals.	Execution Time	Speedup
5 s (reported)	DQSS3	74	0.0158	2.57
	dde23	264	0.0406	
10 s	DQSS3	162	0.0157	3.08
	dde23	498	0.0484	
15 s	DQSS3	246	0.0157	3.54
	dde23	723	0.0557	
20 s	DQSS3	332	0.0155	4.24
	dde23	948	0.0656	
RelTol:1E-3, AbsTol:1E-6				

Table 1: Speedup analysis for DQSS3 vs. dde23 solving Eq.(24)

Simulation Time	Method	# Function Evaluations	Execution Time	Speedup	Rms Error $\times 10^{-6}$	Max Error $\times 10^{-6}$
5 s	DQSS3†	74	0.0158	2.57	2680.7	9020.5
	dde23†	264	0.0406		27.4	86.0
	DQSS3‡	284	0.0158	2.57	43.0	128.9
	DQSS3*	598	0.0336	1.20	4.4	17.7
Error measurements for variable: x_1 † RelTol:1E-3, AbsTol:1E-6 ‡ RelTol:1E-5, AbsTol:1E-6 * RelTol:1E-6, AbsTol:1E-6						

Table 2: Error and Speedup analysis for DQSS3 vs. dde23 solving Eq.(24)

we would allow the maximum absolute error, which occurs at the end of the simulation period, to be of the order of $20 \cdot 10^{-3}$. DQSS3 attained a maximum error of $9 \cdot 10^{-3}$. For reasons unknown to us, dde23 obtains, for this example, an absolute error that is much smaller than the one requested.

In order to compare the two solvers better, we thus squeezed the error tolerances of DQSS3 down to 10^{-5} and even 10^{-6}. With an error tolerance of 10^{-5}, we would allow an absolute error of $20 \cdot 10^{-5}$. DQSS3 attained a maximum error of $12.89 \cdot 10^{-5}$. With an error tolerance of 10^{-6}, we would allow an absolute error of $20 \cdot 10^{-6}$. DQSS3 attained a maximum error of $17.7 \cdot 10^{-6}$.

It can be seen that DQSS3 performs simulations that are as accurate as requested, but not much more accurate. In all of the simulations, DQSS3

turned out to be faster than dde23.

5.2. Example 2

In this example, we study the scalar DDE reported as *Example 3* in [18], whose DDE is defined by Eq. (25). The DDE model is to be simulated for $t \in [0, 20]$ with history $x(t) = t$ for $t \leq 0$.

$$\dot{x}(t) = -\lambda \cdot x(t-1) \cdot (1 + x(t)) \tag{25}$$

This system has a single delay $\tau_1 = 1$, is non–linear, and has a time-varying initial history. The simulation results are shown in Figure 12.

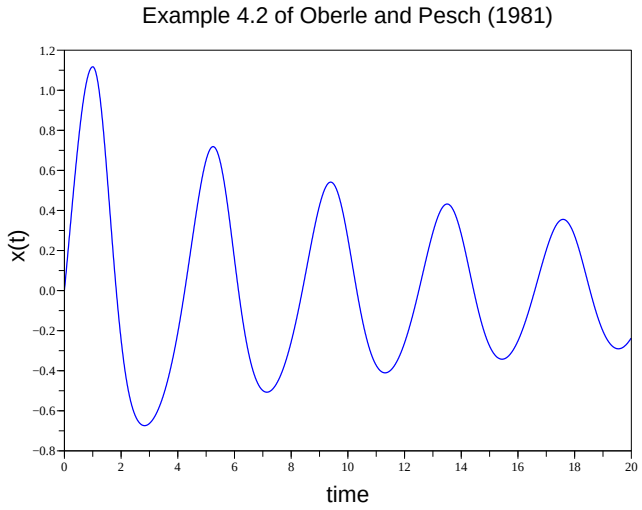

Figure 12: Solution of Example 2 solved for $\lambda = 1.5$ [15]

Average performance results for sets of 30 simulation runs are provided in Table 3.

In this example, DQSS3 always requires slightly more function evaluations than dde23. Since the system to be simulated is characterized by a first-order model, the number of function evaluations needed can be compared better between the two solvers. Thus, we might expect dde23 to execute a

Simulation Time	Method	# Function Evals.	Execution Time	Speedup
20 s (reported)	DQSS3	3825	0.2185	4.22
	dde23	3679	0.9219	
40 s	DQSS3	7379	0.4359	4.83
	dde23	7372	2.1063	
60 s	DQSS3	11121	0.6561	5.62
	dde23	11092	3.6891	
80 s	DQSS3	14931	0.8908	6.26
	dde23	14800	5.5755	
$\lambda = 1.5$ RelTol:1E-6, AbsTol:1E-10				

Table 3: Speedup analysis for DQSS3 vs. dde23 solving Eq.(25)

little faster than DQSS3. However, due to the much simpler nature of the DQSS3 solver in comparison with the algorithm used by dde23, DQSS3 still outperforms dde23.

5.2.1. Error Analysis.

As in Example 1, we wish to compare the errors that were actually obtained using the two solvers. We used the same methodology explained in the discussion of example 1. We requested here a relative error tolerance of 10^{-6}. Since the solution itself is in the order of 1.0, this means that also the absolute error should be of the order of 10^{-6}.

Simulation Time	Method	# Function Evaluations	Execution Time	Speedup	Rms Error $\times 10^{-6}$	Max Error $\times 10^{-6}$
20 s	DQSS3†	3825	0.2185	4.22	0.9	3.2
	dde23†	3679	0.9219		2.7	5.7
	DQSS3‡	8133	0.4378	2.1	0.1	0.5
$\lambda = 1.5$ † RelTol:1E-6, AbsTol:1E-10 ‡ RelTol:1E-7, AbsTol:1E-10						

Table 4: Error and Speedup analysis for DQSS vs. dde23 solving Eq.(25)

To understand the simulation errors even better, we plotted the absolute errors, i.e., the deviations of the simulation trajectories from the reference solution, as functions of time.

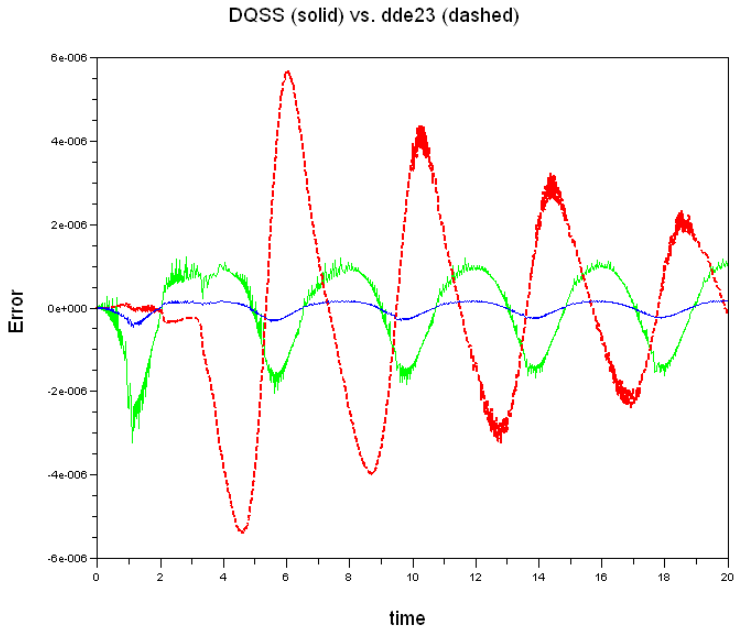

Figure 13: Simulation errors for example 2

The largest errors are those incurred by dde23. The error reaches a maximum value of $5.7 \cdot 10^{-6}$ after roughly 6 seconds of simulated time. The simulation error committed by dde23 is approximately proportional to the solution itself, i.e., the error is large and positive when the solution itself is large and positive.

The medium error curve depicts the errors committed by DQSS3 with a relative error tolerance of 10^{-6}. The error reaches a maximum absolute value of $3.2 \cdot 10^{-6}$ after roughly 1 second of simulated time. The error oscillations are phase–shifted in comparison with those obtained by dde23. Whereas dde23 seems to compute a solution with an amplitude that is, on average, a bit too large, DQSS3 seems to compute a solution with an amplitude that is, on average, a bit too small.

The most accurate results are obtained by DQSS3 with a relative error tolerance of 10^{-7}. The error reaches a maximum absolute value of $5 \cdot 10^{-7}$ after roughly 1 second of simulated time. The error curves for the two DQSS3

simulations look qualitatively similar, except that the errors of the more accurate simulation are, on average 9 times smaller.

In order to obtain these better simulation results, the number of function evaluations has grown by a factor of $\frac{8133}{3825} = 2.126 \approx \sqrt[3]{10}$, which is what we expect of a third-order accurate DQSS method.

5.3. Example 3

We shall now proceed to a slightly more complex DDE system, reported as *Example 4* in [18], whose DDE is defined by Eq. (26).

$$\begin{aligned} \dot{x}_1(t) &= -x_1(t)x_2(t-1) + x_2(t-10) \\ \dot{x}_2(t) &= x_1(t)x_2(t-1) - x_2(t) \\ \dot{x}_3(t) &= x_2(t) - x_2(t-10) \end{aligned} \tag{26}$$

This DDE model is to be simulated for $t \in [0, 40]$ with initial history $x_1(t) = 5$, $x_2(t) = 0.1$, and $x_3(t) = 1$ for $t \leq 0$. This is a third order non–linear system with multiple constant delays of $\tau_1 = 1$ and $\tau_2 = 10$ and constant initial history trajectories. The simulation results are presented in Figure 14.

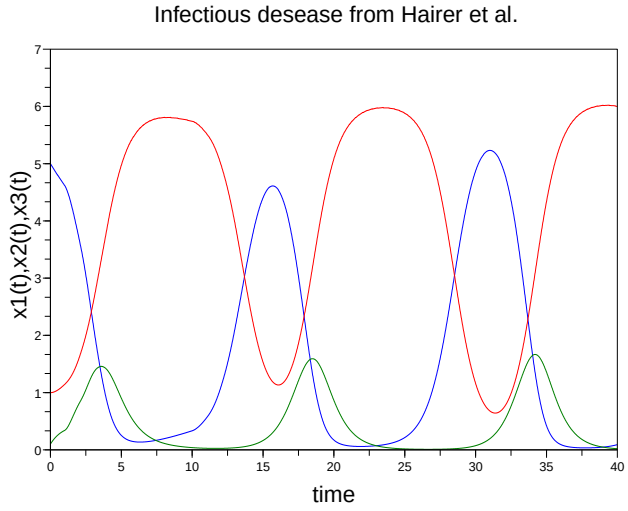

Figure 14: Solution of Example 3 [8]

Average performance results for sets of 30 simulation runs are provided in Table 5.

Simulation Time	Method	# Function Evals.	Execution Time	Speedup
40 s (reported)	DQSS3	1104	0.0470	3.68
	dde23	1353	0.1734	
80 s	DQSS3	2328	0.0470	6.99
	dde23	2568	0.3286	
RelTol:1E-3, AbsTol:1E-6				

Table 5: Speedup analysis for DQSS3 vs. dde23 solving Eq.(26)

While simulating this model, DQSS performs slightly fewer function evaluations than dde23. The speedup factor is similar to that of the previous examples.

5.3.1. Error Analysis.

We repeated the error analysis also for this example. We requested here a relative error tolerance of 10^{-3}. Since the solution itself is in the order of 6.0, this means that the absolute error should be of the order of $6 \cdot 10^{-3}$.

Simulation Time	Method	# Function Evaluations	Execution Time	Speedup	Rms Error $\times 10^{-3}$	Max Error $\times 10^{-3}$
40 s	DQSS3†	1104	0.047	3.68	1.032	6.28
	dde23†	1353	0.1734		10.701	43.75
	DQSS3‡	703	0.0314	5.52	13.599	59.0
Error measurements for variable: x_1 † RelTol:1E-3, AbsTol:1E-6 ‡ RelTol:1E-2, AbsTol:1E-6						

Table 6: Error and Speedup analysis for DQSS3 vs. dde23 solving Eq.(26)

DQSS3 obtained almost exactly the error requested. Its maximum absolute error during the 40 seconds of simulation was $6.28 \cdot 10^{-3}$. In contrast, dde23 generated a maximum absolute error of $43.75 \cdot 10^{-3}$, i.e., an error that is approximately seven times larger. This error is still acceptable, since dde23

controls only the local error, and it is usually expected that the global error may be roughly 10 times larger than the local error.

In order to be able to compare the two solvers a bit better, we repeated the DQSS3 simulation with a tenfold relaxed tolerance of 10^{-2}. We now would expect an error of $6 \cdot 10^{-2}$, whereas DQSS3 generated an error of $5.9 \cdot 10^{-2}$.

While relaxing the tolerance by a factor of 10, the number of function evaluations was reduced by a factor of $\frac{1104}{703} = 1.57$ only, i.e., a little less than $\sqrt[3]{10}$.

Both DQSS3 simulations were more efficient than the dde23 simulation.

To understand the simulation errors better, we once again plotted the absolute errors, i.e., the deviations of the simulation trajectories from the reference solution, as functions of time. In Figure 15 we plotted the errors of the dde23 simulation together with the errors of the DQSS3 simulation with the more stringent accuracy requirements.

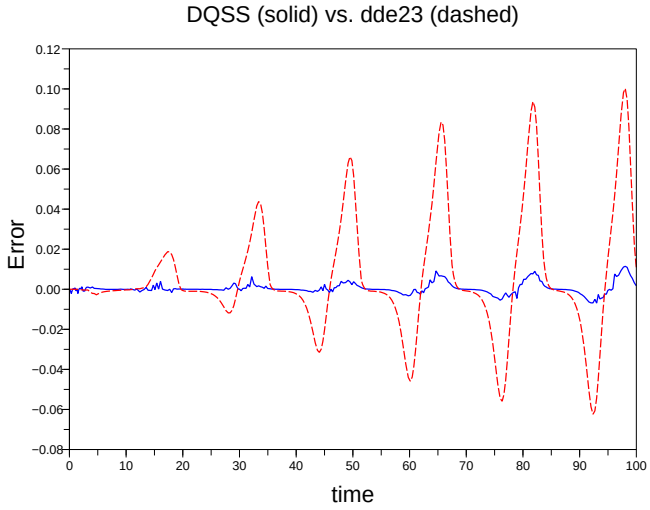

Figure 15: Simulation errors for Example 3

The simulations seem to be mildly unstable, as the absolute errors are growing. However, this is an artifact. The system exhibits a limit cycle, i.e., it is marginally stable. The numerical solutions accumulate over time a small phase error that represents itself, in the error plots, in the form of growing absolute errors.

This becomes evident when looking at the trajectories of x_1 in Figure 16. The dde23 trajectory leads the more accurate DQSS3 trajectory by a small Δt.

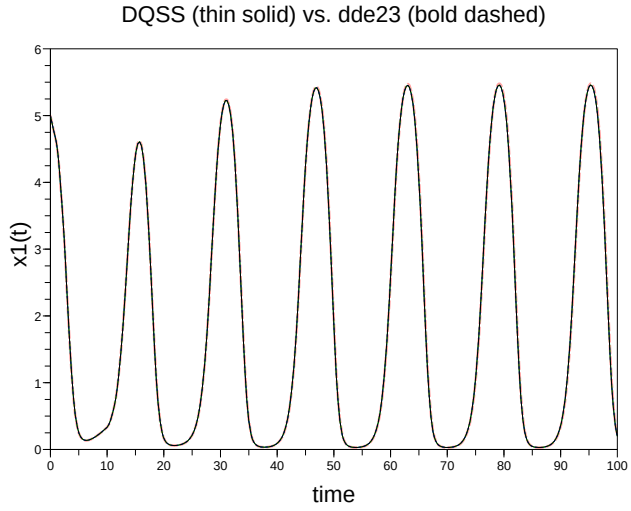

Figure 16: Trajectories of x_1 for Example 3

5.4. Example 4

In this last example, we deal with a much more demanding DDE system. The system is a time–varying DDE with impulses, reported as *Example 3.1* in [19]. The DDE model is defined by Eq. (27).

$$
\begin{aligned}
\dot{x}_1(t) =& -6x_1(t) + sin(2t)f(x_1(t)) + cos(3t)f(x_2(t)) \\
&+ sin(3t)f\left(x_1\left(t - \frac{1+cos(t)}{2}\right)\right) + sin(t)f\left(x_2\left(t - \frac{1+sin(t)}{2}\right)\right) + 4sin(t) \\
\dot{x}_2(t) =& -7x_2(t) + \frac{cos(t)}{3}f(x_1(t)) + \frac{cos(2t)}{2}f(x_2(t)) \\
&+ cos(t)f\left(x_1\left(t - \frac{1+sin(t)}{2}\right)\right) + cos(2t)f\left(x_2\left(t - \frac{1+sin(t)}{2}\right)\right) + 2cos(t)
\end{aligned}
$$

$$
f(x) = \frac{|x+1| - |x-1|}{2} \tag{27}
$$

The initial history is given by $x_1(t) = -0.5$ and $x_2(t) = 0.5$. At each impulse time, $t_k = 2k$, a time–dependent solution impulse is applied with $x_1(t_k)$ being replaced by $1.2x_1(t_k)$, and $x_2(t_k)$ being replaced by $1.3x_2(t_k)$. The simulation results are presented in Figure 17.

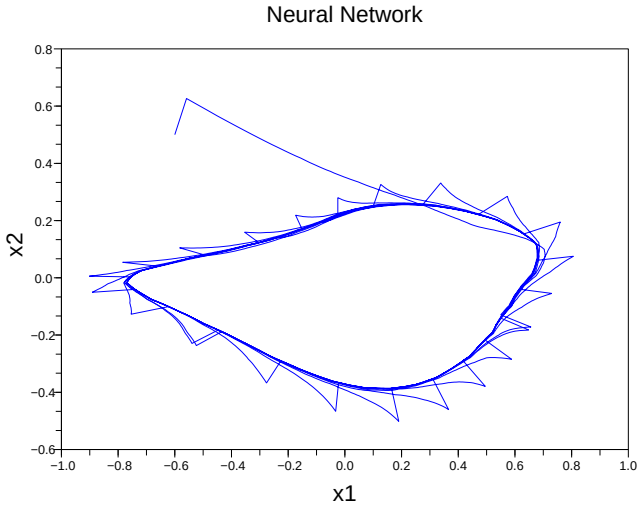

Figure 17: Phase plane of a neural network DDE with time–dependent impulses

This example includes delays and discontinuities in the states. The number of function evaluations of $f()$ in PowerDEVS does not include the number

of times that the two integrators were reset due to the spikes. The total number of resets is 52.

Average performance results for sets of 30 simulation runs are provided in Table 7.

Simulation Time	Method	# Function Evaluations	Execution Time	Slowdown	Rms Error $\times 10^{-6}$
50 s	DQSS3†	15003	0.636	4.07	1154.4
	ddesolver†	15822	0.156		9899.1
	DQSS3‡	11512	0.579	3.51	3062.9
Error measurements for variable: x_1 † RelTol:1E-3, AbsTol:1E-6 ‡ RelTol:1E-2, AbsTol:1E-6					

Table 7: Error and Speedup analysis for DQSS3 vs. ddesolver solving Eq.(27)

For this example, we did not have any Matlab (dde23) code available, but only compiled Fortran code (ddesolver). For this reason, the execution times of DQSS3 and ddesolver are not comparable. The compiled Fortran code is more efficient than the interpreted PowerDEVS code, which is not overly surprising. The number of function evaluations afforded by DQSS is significantly smaller for this example than that used by ddesolver.

We requested here a relative error tolerance of 10^{-3}. Since the solution itself is in the order of 1.0, this means that the absolute error should be of the order of 10^{-3} as well.

5.4.1. Error Analysis.

For this example, we did not provide the maximal absolute errors in the above table, but only the average squared errors. The reason is that this example contains discontinuities. Since the time instants, at which the discontinuities are being executed, varies slightly between the original simulation and the reference solution, the error plot contains spikes that are of the order of the magnitude of the signal itself.

Looking at the above plots, we see that the absolute largest errors for DQSS3(rel.tol=10^{-3}) under exclusion of the spikes are approximately 10^{-3} as requested, whereas the absolute largest errors for DQSS3(rel.tol=10^{-2}) are

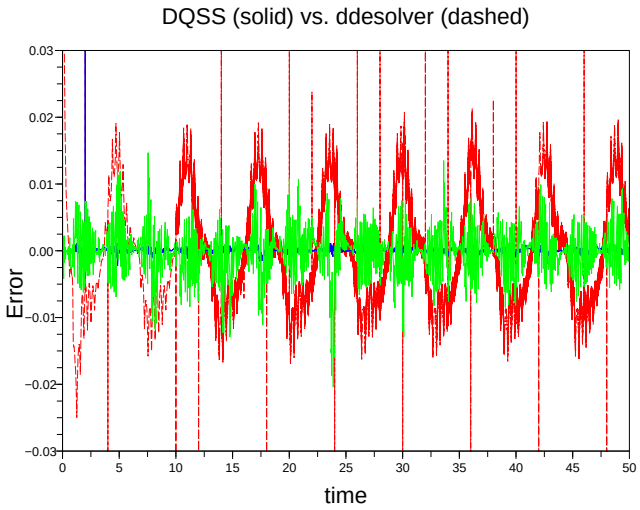

Figure 18: Simulation errors for Example 4

approximately 10^{-2}. Thus, DQSS3 is computing solutions that are almost exactly as accurate as they are supposed to be. The absolute largest errors for ddesolver(rel.tol=10^{-3}) are approximately $2 \cdot 10^{-2}$, i.e., about 20 times as large as requested. However, this is acceptable considering that ddesolver controls the local integration error only, whereas DQSS3 controls the global integration error.

6. Conclusions

In this article, a new class of numerical delay differential equation (DDE) solvers has been introduced, and their use has been demonstrated by means of a number of benchmark problems from the open literature.

Numerical DDE solvers that are based on state quantization instead of time slicing offer some striking and unparalleled properties that make these algorithms particularly well suited for the task at hand.

First, quantized state system (QSS) methods are naturally asynchronous. These are variable–step methods, whereby each state variable is updated at its own pace. State variables with larger gradients are updated more frequently than state variables with little ongoing activity. For this reason, the QSS–based solvers exploit implicitly and intrinsically the sparsity of the model to be simulated.

Second, we were able to show that an asymptotically stable linear time–invariant DDE system with constant delays leads to a QSS approximation that remains numerically stable for any quantum ΔQ. The granularity of the quantization influences the accuracy of the simulation results, but not the numerical stability.

Third, we were able to show that an input–to–state–stable non–linear time-varying DDE system with variable and even state–dependent delays leads to a QSS approximation that remains numerically stable for any sufficiently small quantum ΔQ. For any such system, there exists a quantum $\Delta Q_{\max}$ larger zero, such that its QSS approximation remains numerically stable for all $\Delta Q \leq \Delta Q_{\max}$.

Fourth, we showed that, as ΔQ approaches zero in such a system, the numerical simulation trajectories converge to the analytical trajectories of the original DDE system.

Fifth, whereas the step–size control algorithms of time–slicing based solvers usually control the local integration error only, state–quantization based solvers control the global integration error. Hence state–quantization based solvers are more robust than their time–slicing based competitors.

Sixth, it was demonstrated by means of four different benchmark problems from the open literature that state–quantization based DDE solvers are frequently significantly more efficient than their time–slicing based competitors. The reason for their higher efficiency is partly, because they naturally exploit the sparsity in the models to be simulated, and partly, because the algorithms used for interpolating the delayed states are simpler and can thus be implemented more effectively.

Seventh, the DQSS codes as implemented in PowerDEVS, a modeling and simulation environment with a graphical user interface similar to that of Simulink, are much easier to use than the dde23 and dde_solver codes that were previously available.

7. Acknowledgments

We wish to acknowledge support by CONICET under grant PIP–2009/2011–00183 and Fundación Repsol–YPF under fellowship Estenssoro 2006. We also wish to express our gratitude to Sylvester (Skip) Thompson for his willingness to share the dde23 (Matlab) and dde_solver (Fortran) codes with us.

References

[1] C. Baker, C. Paul, D. Willé, A Bibliography on the Numerical Solution of Delay Differential Equations., Technical Report Numerical Analysis Report No. 269, University of Manchester, U.K., 1995.

[2] B. Balachandran, K.N. T., G.D. Eds., Delay Differential Equations - Recent Advances and New Directions, Springer Science+Business Media, 2009.

[3] F. Bergero, E. Kofman, PowerDEVS. A Tool for Hybrid System Modeling and Real Time Simulation, Simulation: Transactions of the Society for Modeling and Simulation International (2010). In press.

[4] J. Butcher, The Numerical Analysis of Ordinary Differential Equations: Runge–Kutta and General Linear Methods, John Wiley, Chichester, United Kingdom, 1987. 512p.

[5] F. Cellier, E. Kofman, Continuous System Simulation, Springer, New York, 2006.

[6] F. Cellier, E. Kofman, G. Migoni, M. Bortolotto, Quantized state system simulation, in: Proceedings of SummerSim 08 (2008 Summer Simulation Multiconference), Edinburgh, Scotland.

[7] E. Hairer, C. Lubich, G. Wanner, Geometric Numerical Integration Structure-Preserving Algorithms for Ordinary Differential Equations, Springer, 2002.

[8] E. Hairer, S. Nørsett, G. Wanner, Solving Ordinary Differential Equations I: Nonstiff Problems, volume 8 of *Series in Computational Mathematics*, Springer–Verlag, Berlin, Germany, 2^{nd} edition, 2000. 528p.

[9] E. Hairer, G. Wanner, Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems., Springer, Berlin, 1991.

[10] E. Kofman, Discrete Event Simulation of Hybrid Systems, SIAM Journal on Scientific Computing 25 (2004) 1771–1797.

[11] E. Kofman, S. Junco, Quantized state systems. a devs approach for continuous system simulation., Transactions of SCS 18 (2001) 123–132.

[12] J. Lambert, Numerical Methods for Ordinary Differential Systems: The Initial Value Problem, John Wiley & Sons, 1991.

[13] G. Migoni, Simulación por Cuantificación de Sistemas Stiff, Ph.D. thesis, Facultad de Ciencias Exactas, Ingeniería y Agrimensura. Universidad Nacional de Rosario, Rosario, Argentina, 2010.

[14] G. Migoni, E. Kofman, Linearly Implicit Discrete Event Methods for Stiff ODEs, Latin American Applied Research 39 (2009) 245–254.

[15] H. Oberle, H. Pesch, Numerical Treatment of Delay Differential Equations by Hermite Interpolation, Numerische Mathematik 37 (1981) 235–255.

[16] P. Pepe, Z.P. Jiang, A Lyapunov−Krasovskii methodology for ISS and $iISS$ of time-delay systems., Systems Control Letters. 55 (2006) 1006–1014.

[17] L. Shampine, S. Thompson, Solving Delay Differential Equations with dde23., Technical Report, Southern Methodist University, 2000.

[18] L. Shampine, S. Thompson, Solving DDEs in Matlab, Applied Numerical Mathematics 37 (2001) 441 – 458.

[19] S.P. Corwin, S. Thompson and S.M. White, Solving ODEs and DDEs with Impulses, JNAIAM J. Numer. Anal. Indust. Appl. (2008) 139–149.

[20] D. Willé, C. Baker, DELSOL – A Numerical Code for the Solution of Systems of Delay–Differential Equations, Applied Numerical Mathematics 9 (1992) 223–234.

[21] B. Zeigler, Theory of Modeling and Simulation, John Wiley & Sons, New York, 1976.

[22] B. Zeigler, T. Kim, H. Praehofer, Theory of Modeling and Simulation. 2nd. edition, Academic Press, New York, 2000.

Appendix A. Input to State Stability

We define here some tools that are used along the theorems and their proofs.

Class $\mathcal{K}$ function. A continuous real valued function $\alpha : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ is said to belong to class $\mathcal{K}$ if it is strictly increasing and $\alpha(0) = 0$.

Class $\mathcal{L}$ function. A continuous real valued function $\psi : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ is said to belong to class $\mathcal{L}$ if it is strictly decreasing and $\lim_{t\to\infty} \psi(t) = 0$.

Class $\mathcal{KL}$ function. A continuous real valued function $\beta : \mathbb{R}_0^+ \times \mathbb{R}_0^+ \to \mathbb{R}_0^+$ is said to belong to class $\mathcal{KL}$ if it is class $\mathcal{K}$ with respect to the first argument and class $\mathcal{L}$ with respect to the second one.

Input to State Stability. Let $\phi_a(t)$ be the solution of the DDE

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{x}(t - \tau_1(\mathbf{x}, t)), ..., \mathbf{x}(t - \tau_m(\mathbf{x}, t)), \mathbf{u}(t)) \tag{A.1}$$

for $\mathbf{u}(t) = 0$ with a given initial history $\phi_a(t < 0)$, and let $\phi(t)$ be the solution for a given input $\mathbf{u}(t)$ with initial history $\phi(t < 0)$.

Define

$$\Delta\phi_0 = \sup_{t<0} \|\phi(t) - \phi_a(t)\| \tag{A.2}$$

and

$$u_m = \sup_{t\geq 0} \|\mathbf{u}(t)\| \tag{A.3}$$

We say that the system of Eq.(A.1) is input to state stable along $\phi_a(t)$ if a class $\mathcal{KL}$ function β and a class $\mathcal{K}$ function α exist such that

$$\|\phi(t) - \phi_a(t)\| \leq \beta(\Delta\phi_0, t) + \alpha(u_m) \tag{A.4}$$

When the initial history of $\phi(t)$ is identical to that of $\phi_a(t)$, we have $\Delta\phi_0 = 0$ and the ISS condition becomes:

$$\|\phi(t) - \phi_a(t)\| \leq \alpha(u_m) \tag{A.5}$$

In this last case, the ISS property implies that a bounded input provokes a bounded difference between the trajectories of the unforced and the forced system. Additionally, as the upper bound of the input becomes smaller, the upper bound of the difference between both trajectories also becomes smaller.

Appendix B. Proof of Theorem 1

The QSS approximation of Eq. (20) is given by

$$\dot{\mathbf{x}}(t) = A\mathbf{q}(t) + \sum_{i=1}^{m} A_i \mathbf{q}(t - \tau_i) \tag{B.1}$$

Let $\phi_a(t)$ be the analytical solution of Eq.(20) from an arbitrary initial history and let $\phi(t)$ be the solution of Eq. (B.1) from the same initial condition $\phi(t \leq 0) = \phi_a(t \leq 0)$.

Define the perturbation introduced in the states by the quantization of Eq. (B.1) as

$$\mathbf{\Delta x(t)} \triangleq \mathbf{q}(t) - \mathbf{x}(t) \tag{B.2}$$

Now, rewrite Eq. (B.1) as

$$\dot{\mathbf{x}}(t) = A(\mathbf{x}(t) + \mathbf{\Delta x}(t)) + \sum_{i=1}^{m} A_i(\mathbf{x}(t - \tau_i) + \mathbf{\Delta x}(t - \tau_i)) \tag{B.3}$$

This implies that $\phi(t)$ will also be the solution of Eq. (B.3), verifying:

$$\dot{\phi}(t) = A(\phi(t) + \mathbf{\Delta x}(t)) + \sum_{i=1}^{m} A_i(\phi(t - \tau_i) + \mathbf{\Delta x}(t - \tau_i)) \tag{B.4}$$

As $\phi_a(t)$ is a solution of Eq. (20), the following will hold:

$$\dot{\phi}_a(t) = A\phi_a(t) + \sum_{i=1}^{m} A_i \phi_a(t - \tau_i) \tag{B.5}$$

Let us now define the error committed by the QSS approximation as

$$\mathbf{e}(t) \triangleq \phi(t) - \phi_a(t) \tag{B.6}$$

Then, substracting Eq.(B.5) from Eq.(B.4) the error has the following dynamics:

$$\dot{\mathbf{e}}(\mathbf{t}) = A\mathbf{e}(t) + \sum_{i=1}^{m} A_i \mathbf{e}(t - \tau_i) + A\mathbf{\Delta x}(t) + \sum_{i=1}^{m} A_i \mathbf{\Delta x}(t - \tau_i) \tag{B.7}$$

with $\mathbf{e}(t \leq 0) = 0$.

By defining $B \triangleq I$ and

$$\mathbf{u}(t) \triangleq A\boldsymbol{\Delta}\mathbf{x}(t) + \sum_{i=1}^{m} A_i \boldsymbol{\Delta}\mathbf{x}(t - \tau_i) \tag{B.8}$$

the error dynamics can be rewritten as

$$\dot{\mathbf{e}}(t) = A\mathbf{e}(t) + \sum_{i=1}^{m} A_i \mathbf{e}(t - \tau_i) + B\mathbf{u}(t) \tag{B.9}$$

When $\mathbf{u}(t) = 0$, this last equation is identical to Eq. (20) with trivial intial condition. Then, according to the initial hypothesis, the solution of (B.9) for $\mathbf{u}(t) = 0$ is asymptotically stable. Following Prop. 2.5 of [16], this condition implies that Eq. (B.9) is Input to State Stable (ISS).

This is, there exist a class $\mathcal{K}$ function α so that $\|\mathbf{e}(t)\| \leq \alpha(\sup_{t \geq 0}(\|\mathbf{u}(t)\|))$. Then, taking into account that

$$\begin{aligned} \|\mathbf{u}(t)\| &= \|A\boldsymbol{\Delta}\mathbf{x}(t) + \sum_{i=1}^{m} A_i \boldsymbol{\Delta}\mathbf{x}(t - \tau_i)\| \\ &\leq \|A\| \cdot \|\boldsymbol{\Delta}\mathbf{x}(t)\| + \sum_{i=1}^{m} \|A_i\| \cdot \|\boldsymbol{\Delta}\mathbf{x}(t - \tau_i)\| \\ &\leq \|A\| \cdot \|\boldsymbol{\Delta}\mathbf{Q}\| + \sum_{i=1}^{m} \|A_i\| \cdot \|\boldsymbol{\Delta}\mathbf{Q}\| \\ &\leq (\|A\| + \sum_{i=1}^{m} \|A_i\|) \cdot \|\boldsymbol{\Delta}\mathbf{Q}\| \end{aligned} \tag{B.10}$$

being $\boldsymbol{\Delta}\mathbf{Q}$ the quantum vector, it results that

$$\|\mathbf{e}\| \leq \alpha((\|A\| + \sum_{i=1}^{m} \|A_i\|) \cdot \|\boldsymbol{\Delta}\mathbf{Q}\|) \triangleq \gamma(\|\boldsymbol{\Delta}\mathbf{Q}\|) \tag{B.11}$$

where γ is a class $\mathcal{K}$ function.

Then, the error is bounded for all $t \geq 0$. This completes the proof of item 1).

Moreover, when the quantum goes to zero, i.e., $\|\boldsymbol{\Delta}\mathbf{Q}\| \to 0$, it results that $\|\mathbf{e}\| \to 0$ (this is a property of any class $\mathcal{K}$ function). This proves the convergence condition claimed in item 2). ■

Appendix C. Proof of Theorem 2

Proof:. Let e_m be an arbitrary positive number. Let $t_f > 0$ be the first instant of time at which the norm of the error $\|\mathbf{e}(t)\|$ reaches the value e_m. Then, we have $\|\mathbf{e}(t)\| < e_m$ for all $t < t_f$.

Define

$$\boldsymbol{\Delta}\mathbf{x}(t) \triangleq \mathbf{q}(t) - \mathbf{x}(t) \tag{C.1}$$

and

$$\boldsymbol{\Delta}\mathbf{x}_\tau(t) \triangleq \mathbf{x}(t - \tau_1(\mathbf{q}, t)) - \mathbf{x}(t - \tau_1(\mathbf{x}, t)) \tag{C.2}$$

Then, we can rewrite the DQSS approximation of Eq.(23) as

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t) + \boldsymbol{\Delta}\mathbf{x}(t), \mathbf{x}(t - \tau_1(\mathbf{x}, t)) + \boldsymbol{\Delta}\mathbf{x}_\tau(t) + \boldsymbol{\Delta}\mathbf{x}(t - \tau_1(\mathbf{q}, t)), t) \tag{C.3}$$

Define now

$$\begin{aligned}\mathbf{u}(t) \triangleq &\mathbf{f}(\mathbf{x}(t) + \boldsymbol{\Delta}\mathbf{x}(t), \mathbf{x}(t - \tau_1(\mathbf{x}, t)) + \boldsymbol{\Delta}\mathbf{x}_\tau(t) + \boldsymbol{\Delta}\mathbf{x}(t - \tau_1(\mathbf{q}, t)), t) \\ &- \mathbf{f}(\mathbf{x}(t), \mathbf{x}(t - \tau_1(\mathbf{x}, t)), t)\end{aligned} \tag{C.4}$$

Then, the DQSS of Eq.(C.3) becomes identical to the system of Eq.(22), which is ISS along the analytical solution $\phi_a(t)$.

We also know that

$$\|\boldsymbol{\Delta}\mathbf{x}(t)\| \leq \delta Q \triangleq \|\Delta Q\| \tag{C.5}$$

for all t. Notice also that, since we are assuming that $\|\phi(t) - \phi_a(t)\| < e_m$ for all $t < t_f$, then function $\mathbf{f}$ in the DQSS of Eq.(23) is being evaluated in a bounded region, since

$$\|\mathbf{q}(t)\| \leq \phi_{a_{max}} + e_m + \delta Q \tag{C.6}$$

for all $t < t_f$, with $\phi_{a_{max}}$ being an upper bound for the norm of the analytical solution $\phi_a(t)$.

Taking into account that $\mathbf{f}$ is locally Lipschitz, a constant L exists for the given region so that

$$\|\mathbf{u}(t)\| \leq L \cdot (\|\boldsymbol{\Delta}\mathbf{x}(t)\| + \|\boldsymbol{\Delta}\mathbf{x}_\tau(t) + \boldsymbol{\Delta}\mathbf{x}(t - \tau_1(\mathbf{q}, t))\|) \tag{C.7}$$

Using also Eq.(C.5) in this last equation it results that

$$\|\mathbf{u}(t)\| \leq L \cdot (2\delta Q + \|\boldsymbol{\Delta}\mathbf{x}_\tau(t)\|) \tag{C.8}$$

Function $\tau_1(\cdot)$ is also locally Lipschitz and it is being evaluated in the same bounded region. So, a constant L_τ exists in that region such that

$$|\tau_1(\mathbf{q}(t), t) - \tau_1(\mathbf{x}(t), t)| \leq L_\tau \cdot \|\mathbf{q}(t) - \mathbf{x}(t)\| \leq L_\tau \cdot \delta Q \tag{C.9}$$

From Eq.(C.2) we know that

$$\|\mathbf{\Delta x}_\tau(t)\| \leq \max_{t<t_f}(\|\dot{\mathbf{x}}(t)\|) \cdot |\tau_1(\mathbf{q}(t), t)) - \tau_1(\mathbf{x}(t), t))| \tag{C.10}$$

Recalling that $\mathbf{f}(\cdot)$ is being evaluated in a bounded region defined by Eq.(C.6) and it is locally Lipschitz, it results bounded, and then a positive constant f_m exists so that $(\|\dot{\mathbf{x}}(t)\|) \leq f_m$ for $t < t_f$. Then,

$$\|\mathbf{\Delta x}_\tau(t)\| \leq f_m \cdot L_\tau \cdot \delta Q \tag{C.11}$$

Plugging this last inequality in Eq.(C.8), we obtain that

$$\|\mathbf{u}(t)\| \leq (2 \cdot L + f_m \cdot L_\tau) \cdot \delta Q \tag{C.12}$$

Since the forced system of Eq.(22) is ISS and $\phi(t) - \phi_a(t) = 0$ for $t \leq 0$, then the last equation implies that a class $\mathcal{K}$ function α exists so that

$$\|\mathbf{e}(t)\| \leq \alpha((2 \cdot L + f_m \cdot L_\tau) \cdot \delta Q) \tag{C.13}$$

for all $t < t_f$.

Then, by taking a sufficiently small quanta δQ, we can make

$$\|\mathbf{e}(t)\| \leq \alpha((2 \cdot L + f_m \cdot L_\tau) \cdot \delta Q) < \rho \cdot e_m \tag{C.14}$$

for any $0 < \rho < 1$.

This contradicts the initial assumption that the error reaches the value e_m at time t_f. Thus, by taking the quantum such that Eq.(C.14) is accomplished, the error will never reach e_m.

Moreover, if we take an arbitrary small value for e_m, we can always find δQ such that Eq.(C.14) is verified. This proves convergence when $\delta Q \to 0$. ■