Improving Linearly Implicit Quantized State System Methods

Franco Di Pietro*[1], Gustavo Migoni[1,2], and Ernesto Kofman[1,2]

[1] *CIFASIS–CONICET, 27 de febrero 210 bis, S2000EZP Rosario, Argentina*
[2] *Departamento de Control, FCEIA, Universidad Nacional de Rosario, Riobamba 245 bis, (2000) Rosario, Argentina*

Abstract

In this article we propose a modification to Linearly Implicit Quantized State System Methods (LIQSS), a family of methods for solving stiff ODE's that replace the classic time discretization by the quantization of the state variables. LIQSS methods were designed to efficiently simulate stiff systems but they only work when the system has a particular structure. The proposed modification overcomes this limitation allowing the algorithms to efficiently simulate stiff systems with more general structures. Besides describing the new methods and their algorithmic descriptions, the article analyzes the algorithms performance in the simulation of some complex systems.

Keywords Quantized state systems, numerical simulation, stiff systems

1 Introduction

The simulation of continuous time models requires the numerical integration of the Ordinary Differential Equations (ODEs) that represent them. The literature on numerical methods for ODEs [1, 2, 3] contains hundreds of algorithms with different features that make them suitable for solving different types of problems.

*Corresponding author:
Address: 27 de febrero 210 bis, S2000EZP Rosario, Argentina
Phone: +54 341 4815569
E-mail: dipietro@cifasis-conicet.gov.ar

Some ODE systems exhibit certain characteristics that pose difficulties to numerical ODE solvers. The presence of simultaneous fast and slow dynamics, known as stiffness, is one of these cases. Due to stability reasons, these systems enforce the usage of implicit ODE solvers that must perform expensive iterations over sets of nonlinear equations at each time step. The presence of discontinuities is another difficult case, where the ODE solvers must detect their occurrence using iterative procedures, restarting the simulation after each event.

ODE models coming from power electronics, spiking neural networks, multi-particle collision dynamics, and several other technical areas, exhibit very frequent discontinuities and, sometimes, stiffness. Consequently, the simulation of these systems becomes very expensive.

In the last years, a new family of numerical ODE solvers that can efficiently handle discontinuities was developed. These algorithms, called Quantized State System (QSS) [1, 4], replace the time discretization performed by classic ODE solvers by the quantization of the state variables. Regarding stiffness, a family of Linearly Implicit QSS (LIQSS) solvers was recently developed [5], that can efficiently simulate some of these systems.

A limitation of LIQSS algorithms is that they require that the stiffness is due to the presence of large entries on the main diagonal of the Jacobian matrix of the system. Otherwise, spurious oscillations may appear on the simulated trajectories undermining performance.

In this article, extending the preliminary results on the first order accurate LIQSS1 algorithm presented in [6], we propose a modification of LIQSS methods of order 1 and 2 that overcomes that limitation. Although the modified LIQSS (mLIQSS) algorithms do not cover all stiff structures, we show that they work in several practical cases.

Besides introducing the new algorithms, we analyze their properties, we describe their implementation in the stand-alone QSS solver [7] and we present simulation results, comparing the performance of the new methods with that of the original LIQSS algorithms as well as that of classic solvers like DASSL (Differential Algebraic System Solver) and DOPRI. (Dormand-Prince solver).

The paper is organized as follows: Section 2 introduces the previous concepts and definitions used along the rest of the work. Then, Section 3 describes the new algorithms and their implementation. Finally, Section 4 presents the simulation results and Section 5 concludes the article.

Glossary

The different algorithms described along the article use the following notation:

---

t = current simulation time
t_f = final time of simulation
x = state vector (x_i refers to its i−th component)
$\dot{x}$ = first time derivative vector
$\ddot{x}$ = second time derivative vector
q = quantized state vector
$\dot{q}$ = quantized state slope vector
t^x = array containing the last update time of each state
t^η = array containing the next update time of each quantized state
ΔQ = array containing the quantum of each state
e = elapsed time (auxiliary variable)
A = Jacobian matrix
u = affine coefficients matrix
$\dot{u}$ = affine coefficients slopes matrix
$\dot{x}_i^+$ = estimation of the future value of $\dot{x}_i$
q_i^- = previous value of the quantized state q_i
$\dot{x}_i^-$ = previous value of the state derivative $\dot{x}_i$
h = step size (auxiliary variable)
t^q = array containing the last update time of each quantized state
$\ddot{x}_i^+$ = estimation of the future value of $\ddot{x}_i$

---

2 Background

This section provides the background required to tackle the rest of the article. Starting with a brief description of the problems suffered by classic ODE solvers when dealing with discontinuous and stiff systems, the family of QSS solvers is presented.

2.1 Numerical Integration of Stiff and Discontinuous ODEs

Many dynamical systems of practical relevance, both in science and engineering, are stiff. Integration of these systems using traditional numerical methods based on time discretization requires the use of implicit algorithms, because all explicit methods must necessarily restrict the integration step to ensure numerical stability.

The reason is that the numerical stability domains of all explicit numerical ODE solvers invariably bend into the left–half complex $\lambda \cdot h$ plane, and algorithms with stability domains looping in the left–half plane force small step sizes, h, on the numerical ODE solver, in order to capture all eigenvalues, λ_i, of a stiff system inside the numerically stable region. The only way to avoid that the integration step size be limited by numerical stability is using implicit algorithms, the stability domains of which bend into right–half complex plane, a characteristic that can be observed in some, but not all, implicit ODE solvers. Such solvers are referred to as stiff ODE solvers. Consequently, implicit methods have higher computational cost than explicit ones, because they call for iterative algorithms in each step to calculate the next value.

Regarding discontinuities, it must be taken into account that classic algorithms are based, either explicitly or implicitly, on *Taylor series expansions* [1] that express the solution at the next time t_{k+1} as polynomials in the step size h around the current time t_k. As discontinuous trajectories cannot be represented by polynomials, the numerical algorithms usually introduce unacceptable errors when a discontinuity occurs between time t_k and t_{k+1}.

To avoid this problem, ODE solvers must detect the exact instant at which the discontinuity occur, advance the simulation until that time, and restart the simulation from the new conditions. This strategy, known as zero crossing detection and event handling, is expensive in terms of computational costs as the zero crossing location usually involves iterations.

2.2 Quantized State System Methods

QSS methods replace the time discretization of classic numerical integration algorithms by the quantization of the state variables.

Given a time invariant ODE in its State Equation System (SES) representation:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t), t) \tag{1}$$

where $\mathbf{x}(t) \in \mathbb{R}^n$ is the state vector, the first order Quantized State System (QSS1) method [4] analytically solves an approximate ODE called Quantized State System:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{q}(t), t) \tag{2}$$

Here, $\mathbf{q}(t)$ is a *quantized state* vector whose components are constrained to follow piecewise constant trajectories. Each quantized state $q_i(t)$ is related

to the corresponding state $x_i(t)$ by a *hysteretic quantization function*:

$$q_i(t) = \begin{cases} x_i(t) & \text{if } |q_i(t^-) - x_i(t)| = \Delta Q_i \\ q_i(t^-) & \text{otherwise} \end{cases}$$

This is, $q_i(t)$ only changes when it differs from $x_i(t)$ by a magnitude ΔQ_i called *quantum*. When the difference between the state and the quantized state reaches the quantum, the quantized state $q_i(t)$ takes the value of the state $x_i(t)$. Figure 1 plots a state trajectory and the corresponding quantized state trajectory.

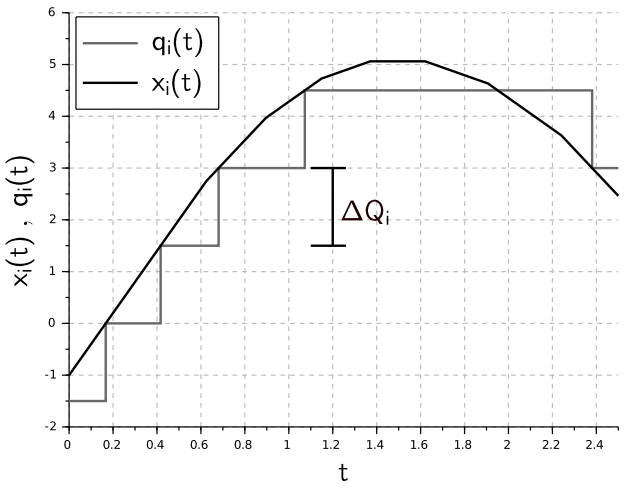


Figure 1: QSS1 hysteretic quantization function.

Since the quantized state trajectories $q_i(t)$ are piecewise constant, then, the state derivatives $\dot{x}_i(t)$ also follow piecewise constant trajectories and, consequently, the states $x_i(t)$ follow piecewise linear trajectories.

Due to the particular form of the trajectories, the numerical solution of Eq. (2) is straightforward and can be easily translated into a simple simulation algorithm.

For $j = 1, \ldots, n$, let t_j^η denote the next time at which $|q_j(t) - x_j(t)| = \Delta Q_j$. Then, the QSS1 simulation algorithm works as follows[1]:

[1]Algorithm 1 describes the behaviour of QSS1 during quantized state updates. Steps due to discontinuities or to significant changes in input trajectories are not described here as they are identical in the different QSS algorithms.

Algorithm 1: QSS1.

```
1  while(t < t_f) // simulate until final time tf
2    t = min(t_j^η) // adavance simulation time
3    i = argmin(t_j^η) // the i-th quantized state changes first
4    e = t − t_i^x // elapsed time since last xi update. (t^x is an
         array containing the last update time of each state.)
5    x_i = x_i + ẋ_i · e // update i-th state value
6    q_i = x_i // update i-th quantized state
7    t_i^η = min(τ > t) subject to |q_i − x_i(τ)| = ΔQ_i // compute the time
         of then next change in the i-th quantized state
8    for each j ∈ [1, n] such that ẋ_j depends on q_i
9      e = t − t_j^x // elapsed time since last xj update
10     x_j = x_j + ẋ_j · e // update j-th state value
11     if j ≠ i then t_j^x = t // last xj update
12     ẋ_j = f_j(q, t) // recompute j-th state derivative
13     t_j^η = min(τ > t) subject to |q_j − x_j(τ)| = ΔQ_j // recompute the
           time of the next change in the j-th quantized state
14   end for
15   t_i^x = t // last xi update
16 end while
```

QSS1 has very nice stability and error bound properties: the simulation of a stable system provides stable results [4] and the maximum error (in the simulation of a linear time invariant system) is bounded by a linear function of the quantum size ΔQ.

Since the quantized states follow piecewise constant trajectories, the instant of times at which they cross a given threshold can be directly computed allowing the straightforward detection of discontinuities[2]. Moreover, when a discontinuity occurs, it will eventually change some state derivatives in the same way a change in a quantized variable does during a normal step. That way, the simulation does not need to be restarted. In conclusion, the detection and handling of a discontinuity does not take more computational effort than that of a single step. Thus, the QSS1 method is very efficient to simulate discontinuous systems [8].

In spite of this advantage and the fact that it has some nice stability and error bound properties [1], QSS1 performs only a first order approximation and it cannot obtain accurate results without significantly reducing ΔQ_i thereby increasing the required number of steps. This accuracy limitation was improved with the definition of the second and third-order accurate QSS methods called QSS2 [9] and QSS3 [10], respectively.

[2]In QSS simulation tools, discontinuities are detected using the quantized states q_i. Alternatively, the states x_i can be used. In that case, the zero crossing detection can be more accurate but in presence of nonlinear threshold conditions that detection can be more involved.

QSS2 and QSS3 have the same definition of QSS1 except that the components of $\mathbf{q}(t)$ are calculated to follow piecewise linear and piecewise parabolic trajectories, respectively.

The simulation algorithm for QSS2 is similar to that of QSS1, except that it also computes the quantized state slopes $\dot{q}_i(t)$ and the state second derivative $\ddot{x}_i(t)$ as it is sketched below:

Algorithm 2: QSS2.

```
1  while(t < t_f) // simulate until final time tf
2    t = min(t_j^η) // adavance simulation time
3    i = argmin(t_j^η) // the i-th quantized state changes first
4    e = t − t_i^x // elapsed time since last xi update
5    // update i-th state value and its derivative
6    x_i = x_i + ẋ_i · e + 0.5 · ẍ_i · e^2
7    ẋ_i = ẋ_i + ẍ_i · e
8    // update i-th quantized state
9    q_i = x_i
10   q̇_i = ẋ_i
11   t_i^η = min(τ > t) subject to |q_i(τ) − x_i(τ)| = ΔQ_i // compute the
         time of then next change in the i-th quantized state
12   for each j ∈ [1, n] such that ẋ_j depends on q_i
13     e = t − t_j^x // elapsed time since last xj update
14     // update j-th state value and its derivatives
15     x_j = x_j + ẋ_i · e + 0.5 · ẍ_j · e^2
16     ẋ_j = f_j(q(t), t) // recompute state derivative
17     ẍ_j = f_j(q(t), t) // recompute state second derivative
18     t_j^η = min(τ > t) subject to |q_j(τ) − x_j(τ)| = ΔQ_j // recompute the
           time of the next change in the j-th quantized state
19     if j ≠ i then t_j^x = t // last xj update
20   end for
21   t_i^x = t // last xi update
22 end while
```

QSS2 steps are more expensive than those of QSS1. In particular, there are two scalar function evaluations to compute $\dot{x}_j$ and $\ddot{x}_j$ (lines 16–17) and the calculation of the next quantized state change in line 18 involves solving a quadratic equation. This additional cost is compensated by the fact that QSS2 can perform much larger steps achieving better error bounds.

QSS2 and QSS3 share the same advantages and properties of QSS1, i.e., they satisfy stability and error bound properties and they are very efficient to simulate discontinuous systems.

2.3 Linearly Implicit QSS Methods

In spite of these advantages, QSS1, QSS2 and QSS3 methods are inefficient to simulate stiff systems. In presence of simultaneous slow and fast dynamics, these methods introduce spurious high frequency oscillations that provoke a large number of steps with its consequent computational cost [1].

To overcome this problem, the family of QSS methods was extended with a set of algorithms called Linearly Implicit QSS (LIQSS) which are appropriate to simulate some stiff systems [5]. LIQSS methods combine the principles of QSS methods with those of classic linearly implicit solvers. There are LIQSS algorithms that perform first, second and third-order accurate approximations: LIQSS1, LIQSS2, and LIQSS3, respectively.

The main idea behind LIQSS methods is inspired by classic implicit methods that evaluate the state derivatives at future instants of time. In classic methods, these evaluations require iterations and/or matrix inversions to solve the resulting implicit equations. However, taking into account that QSS methods know the future value of the quantized state (it is $q_i(t) \pm \Delta Q_i$), the implementation of LIQSS algorithms is explicit and does not require iterations or matrix inversions.

LIQSS methods share with QSS methods the definition of Eq. (2), but the quantized states are computed in a more involved way, taking into account the sign of the state derivatives.

In LIQSS1 the idea is that $q_i(t)$ is set equal to $x_i(t) + \Delta Q_i(t)$ when the future state derivative $\dot{x}_i(t^+)$ is positive. Otherwise, when the future state derivative is negative $q_i(t)$ is set equal to $x_i(t) - \Delta Q_i(t)$. Then, when x_i reaches q_i, a new step is taken. That way, the quantized state is a future value of the state and the derivatives in Eq. (2) are computed using a future state value, as in classic implicit algorithms.

In order to predict the sign of the future state derivative the following linear approximation for the i–th state dynamics is used:

$$\dot{x}_i(t) = A_{i,i} \cdot q_i(t) + u_{i,i}(t) \tag{3}$$

where $A_{i,i} = \frac{\partial f_i}{\partial x_i}$ is the i-th main diagonal entry of the Jacobian matrix and $u_{i,i}(t) = f_i(\mathbf{q}(t), t) - A_{i,i} \cdot q_i(t)$ is an affine coefficient.

It could happen that $A_{i,i} \cdot (x_i(t) + \Delta Q_i) + u_{i,i}(t) < 0$, i.e., when we propose to use $q_i(t) = x_i(t) + \Delta Q_i$ the derivative $\dot{x}_i(t^+)$ becomes negative. It can also happen that $A_{i,i} \cdot (x_i(t) - \Delta Q_i) + u_{i,i}(t) > 0$. Thus, $q_i(t)$ cannot be chosen as a future value for $x_i(t)$. However, in that case, q_i can be chosen such that $\dot{x}_i(t) = 0$. That equilibrium value for q_i can be calculated from Eq.(3) as

$$q_i = -\frac{u_{i,i}}{A_{i,i}} \tag{4}$$

Then, the LIQSS1 simulation algorithm works as follows:

Algorithm 3: LIQSS1.

```
1  while(t < t_f) // simulate until final time tf
2    t = min(t_j^η) // adavance simulation time
3    i = argmin(t_j^η) // the i-th quantized state changes first
4    e = t − t_i^x // elapsed time since last xi update
5    x_i = x_i + ẋ_i · e // update i-th state value
6    q_i^− = q_i //store previous value of qi
7    ẋ_i^− = ẋ_i //store previous value of dxi/dt
8    ẋ_i^+ = A_{i,i} · (x_i + sign(ẋ_i) · ΔQ_i) + u_{i,i} // future state derivative
         estimation using last linear approximation stored in
         arrays A and u.
9    if (ẋ_i · ẋ_i^+ >0) //the state derivative keeps its sign
10     q_i = x_i + sign(ẋ_i) · ΔQ_i
11   else //the state changes its direction
12     q_i = −u_{i,i}/A_{i,i} // choose qi such that dxi/dt = 0
13   end if
14   t_i^η = min(τ > t) subject to x_i(τ) = q_i // compute the time of then
         next change in the i-th quantized state
15   for each j ∈ [1, n] such that ẋ_j depends on q_i
16     e = t − t_j^x // elapsed time since last xj update
17     x_j = x_j + ẋ_j · e // update j-th state value
18     if j ≠ i then t_j^x = t// last xj update
19     ẋ_j = f_j(q, t) // recompute j-th state derivative
20     t_j^η = min(τ > t) subject to x_j(τ) = q_j or |q_j − x_j(τ)| = 2ΔQ_j //
           recompute the time of the next change in the j-th
           quantized state
21   end for
22   // update linear approximation coefficients
23   A_{i,i} = (ẋ_i − ẋ_i^−)/(q_i − q_i^−) // Jacobian diagonal entry
24   u_{i,i} = ẋ_i − A_{i,i} · q_i // affine coefficient
25   t_i^x = t // last xi update
26 end while
```

It can be seen that LIQSS1 requires only a few additional calculations relative to QSS1. In particular, LIQSS1 estimates the future state derivative using a linear model (line 8) and it estimates the Jacobian main diagonal entry $A_{i,i}$ and the affine coefficient (lines 23–24)[3].

Notice also that in line 20 the algorithm checks the additional condition $|q_j - x_j(\tau)| = 2\Delta Q_j$, as a change in variable q_i can change the sign of the state derivative $\dot{x}_j(t)$ so that x_j no longer approaches q_j. In this case, we

[3]Notice that $A_{i,i}$ is computed using two values of $\dot{x}_i = f_i(\mathbf{q}, t)$. The first one is previous to the change in q_i and the second one is obtained after that change. The fact that the first value might be obtained for a different time t should not affect the calculations since changes in f_i due to the time advance are treated as input steps.

still ensure that the difference between x_j and q_j is bounded (by $2\Delta Q_j$). However, we shall see then that the fact that x_j does not always approach q_j may result in inefficient solution of some stiff systems.

LIQSS1 shares the main advantages of QSS1 and it can efficiently integrate stiff systems provided that the stiffness is due to the presence of large entries in the main diagonal of the Jacobian matrix. Like QSS1, it cannot achieve good accuracy, and higher order LIQSS methods have been proposed.

The second order accurate LIQSS2 combines the ideas of QSS2 and LIQSS1. In this case, like in QSS2, the quantized states follow piecewise linear trajectories. This algorithm will be explained later.

2.4 Implementation of QSS Methods

The easiest way of implementing QSS methods is by building an equivalent DEVS model, where the events represent changes in the quantized variables. Based on this idea, the whole family of QSS methods were implemented in PowerDEVS [11], a DEVS–based simulation platform specially designed for and adapted to simulating hybrid systems based on QSS methods. In addition, the explicit QSS methods of orders 1 to 3 were also implemented in a DEVS library of Modelica [12] and implementations of the first–order QSS1 method can also be found in CD++ [13] and VLE [14].

In spite of being simple, DEVS implementations of QSS algorithms are inefficient since the computations involved in each simulation step are performed by different DEVS atomic models exchanging events. The problem is that the coordination and handling of the corresponding messages is more computationally expensive than the computations required by the simulation steps. For this reason, the complete family of QSS methods was implemented in a *stand–alone QSS solver* [7] that improves DEVS–based simulation times by more than one order the magnitude. From a DEVS perspective, it can be thought that the stand–alone QSS solver replaces the coupled DEVS model by an equivalent single atomic model.

3 Modified LIQSS Algorithms

In this section, we first analyze the main limitation of LIQSS algorithms concerning the appearance of fast oscillations in systems where the stiffness is not due to large entries on the main diagonal of the Jacobian matrix. Then, we propose an idea to overcome this problem, and using this approach, we propose a first and second order accurate modified LIQSS methods.

3.1 LIQSS limitations

The simulation of a stable first order system with QSS1 algorithm produces a result that usually finishes with the state trajectory oscillating around the equilibrium point [1]. These oscillations are the reason why QSS1 is not efficient to simulate stiff systems.

That problem is solved by LIQSS1, that prevents the oscillations by taking the quantized state as a future value of the state, i.e. $q_i = x_i + \text{sign}(\dot{x}_i) \cdot \Delta Q_i$. When it is not possible, i.e. when proposing $q_i = x_i + \Delta Q_i$ produces that $\dot{x}_i < 0$ and proposing $q_i = x_i - \Delta Q_i$ produces that $\dot{x}_i > 0$, LIQSS1 realizes that is is near an equilibrium point that can be computed using a linear approximation as shown in Eq.(4).

However, LIQSS1 cannot ensure that q_i is always the future value of x_i because, after computing q_i, $\dot{x}_i$ can change its sign due to a change in some other quantized variable q_j. In such case, then it can also happen that the next change in q_i triggers a change in the sign of $\dot{x}_j$. This situation may lead to oscillations involving states x_i and x_j.

We shall illustrate this behaviour with a simple example. Consider the following system

$$\begin{aligned} \dot{x}_1 &= -x_1 - x_2 + 0.2 \\ \dot{x}_2 &= x_1 - x_2 + 1.2 \end{aligned} \tag{5}$$

with initial conditions $\mathbf{x_0} = [-4 \quad 4]^T$ and quantum $\Delta Q_{1,2} = 1$.

The successive simulation steps of this system with LIQSS1 algorithm can be seen on Table 1. It can be noticed that at time $t_6 = 7.01$, the states and quantized states are identical to those at time $t_2 = 2.95$. Thus, instead of reaching the equilibrium, the simulation exhibits an oscillation with period $t_6 - t_2 = 4.06$, as depicted in Fig.2.

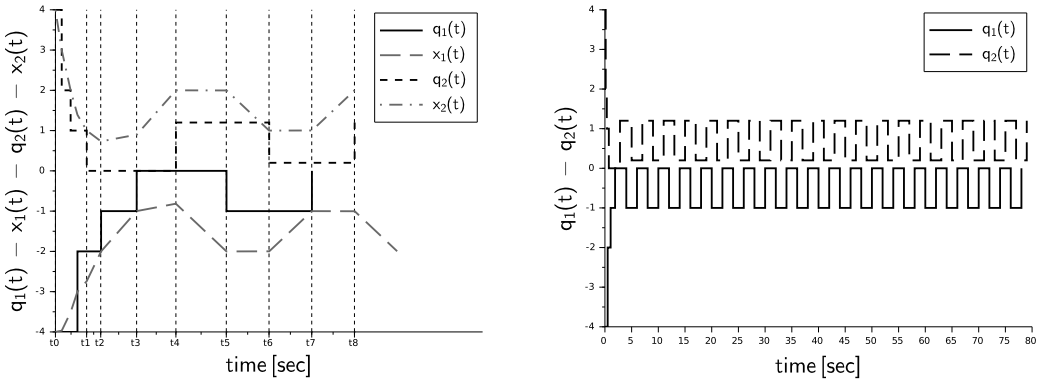
If Eqs.(5) were part of a larger system with slower dynamics, those spurious oscillations would provoke an unnecessary large number of simulation steps. That way, LIQSS1 cannot efficiently simulate stiff systems having this type of structure.

3.2 Basic Idea

In order to avoid the oscillations between pairs of variables, we propose to check whether a quantized state update produces a change in the sign of some other state derivative value. If so, we additionally check whether an eventual update of the second quantized state would cause a change in the sign of the previous state derivative. Under this situation, we expect that both variables

	time	q_1	q_2	$\dot{x}_1$	$\dot{x}_2$	next time
t_0	0	-4	4	0.2	-6.8	0.29
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
t_1	1.65	0	0	0	1.2	2.95
t_2	2.95	0	1.2	-1	0	4.21
t_3	4.21	-1	1.2	0	-1	5.01
t_4	5.01	-1	0.2	1	0	6.01
t_5	6.01	0	0.2	0	1	7.01
t_6	7.01	0	1.2	-1	0	8.01
t_7	8.01	-1	1.2	0	-1	9.01
t_8	9.01	-1	0.2	1	0	10.01

Table 1: Evolution of the simulation with LIQSS1 algorithm.



(a) With $t_f = 10sec.$ (b) With $t_f = 100sec.$

Figure 2: Simulation of system given by Eqs.(5) with LIQSS1 algorithm

experience spurious oscillations, and, in order to prevent them, we apply a simultaneous change in both quantized states using a linearly implicit step.

While this strategy may not solve general stiff structures, it will avoid the appearance of oscillations between pairs of variables, which covers several practical cases.

In order to deal with simultaneous changes in pairs of variables, we shall exploit the resemblance of LIQSS1 steps with Backward Euler steps. In fact, a change in a quantized state q_i can be alternatively computed as

$$q_i = x_i + h \cdot (A_{i,i} \cdot q_i + u_{i,i}) \tag{6}$$

where h is the largest step size such that

$$|q_i - x_i| \leq \Delta Q_i \tag{7}$$

Notice that, provided that h is finite, then $q_i = x_i + sign(\dot{x}_i) \cdot \Delta Q_i$ as stated in line 10 of Algorithm 3. Otherwise, when h is infinite, then $(A_{i,i} \cdot q_i + u_{i,i}) = 0$ as stated in line 12.

In other words, the calculation of q_i can be done using a Backward Euler step on the linear model approximation[4] of Eq.(3) with a step size h defined as the maximum step size that accomplishes the inequality of Eq.(7).

Also notice that this formulation does not require to check the sign of the state derivatives.

3.3 Modified LIQSS1 (mLIQSS1)

Based on the idea expressed above, the modification introduced to the LIQSS1 algorithm consists in checking an additional condition to verify that after changing a quantized state the other state derivatives do not change their signs. To check this condition, for each pair of state variables x_i, x_j, such that each directly influences the derivative of the other, a linear approximation of the form

$$\begin{aligned} \dot{x}_i &= A_{ii} \cdot q_i + A_{ij} \cdot q_j + u_{ij} \\ \dot{x}_j &= A_{ji} \cdot q_i + A_{jj} \cdot q_j + u_{ji} \end{aligned} \tag{8}$$

is used. Here, $A_{i,j} = \frac{\partial f_i}{\partial x_j}(\mathbf{q}, t)$ is the i, j entry of the Jacobian matrix, and $u_{ij} = f_i(\mathbf{q}, t) - A_{ii} \cdot q_i - A_{ij} \cdot q_j$ is an affine coefficient.

If the new value of q_i does not change the sign of any state derivative computed with the linear approximation of Eq.(8), the algorithm works identically to LIQSS1. Otherwise, we propose a new value for q_j in the new direction of x_j. Then, we check if that proposed value for q_j changes back the sign of $\dot{x}_i$. If it does not, we ignore the change in q_j and the algorithm follows identical steps to those of LIQSS1. In other case, we know that an oscillation may appear between states x_i and x_j, so we compute both quantized states q_i and q_j using a Backward Euler step on the linear model of Eq.(8).

Defining

$$\mathbf{q}_{ij} \triangleq \begin{bmatrix} q_i \\ q_j \end{bmatrix} \ , \ \mathbf{x}_{ij} \triangleq \begin{bmatrix} x_i \\ x_j \end{bmatrix} \ , \ \dot{\mathbf{x}}_{ij} \triangleq \begin{bmatrix} \dot{x}_i \\ \dot{x}_j \end{bmatrix} \tag{9}$$

[4]Performing implicit steps on the linear approximation is similar to what classic Linearly Implicit algorithms do. The name of *Linearly Implicit* QSS is due to this reason.

and

$$\mathbf{A}_{ij} = \begin{bmatrix} A_{ii} & A_{ij} \\ A_{ji} & A_{jj} \end{bmatrix} \ , \ \mathbf{u}_{ij} \triangleq \begin{bmatrix} u_{ij} \\ u_{ji} \end{bmatrix} \tag{10}$$

the backward Euler step is given by the equation

$$\mathbf{q}_{ij} = \mathbf{x}_{ij} + h \cdot \dot{\mathbf{x}}_{ij} = \mathbf{x}_{ij} + h \cdot (\mathbf{A}_{ij} \cdot \mathbf{q}_{ij} + \mathbf{u}_{ij}) \tag{11}$$

that can be rewritten as

$$\mathbf{q}_{ij} = (\mathbf{I} - h \cdot \mathbf{A}_{ij})^{-1} \cdot (\mathbf{x}_{ij} + h \cdot \mathbf{u}_{ij}) \tag{12}$$

where h is computed as the maximum step size so that the difference between the states and the quantized states is bounded by the quantum.

Given an integration step h, Eq.(12) allows to compute the quantized states q_i and q_j. The value of h should be such that the quantized states do not differ from the states in a quantity that exceeds the quantum, i.e.,

$$|q_i - x_i| \leq \Delta Q_i \ , \ \ |q_j - x_j| \leq \Delta Q_j \tag{13}$$

The maximum value of h that accomplishes both inequalities can be found analytically by solving Eq.(12) for h under $q_i = x_i \pm \Delta Q_i$ and $q_j = x_j \pm \Delta Q_j$ and taking the minimum among the different solutions. Alternatively, h can be found by iterations on Eq.(12). In either case, Eq.(12) with the restrictions imposed by Eq.(13), implicitly define a function that computes the maximum value of h satisfying those restrictions, i.e.,

$$h_{\max} = \text{max\_be\_step}(\mathbf{x}_{ij}) \triangleq \max(h : |q_i - x_i| \leq \Delta Q_i \wedge |q_j - x_j| \leq \Delta Q_j) \tag{14}$$

Thus, the mLIQSS1 simulation algorithm is identical to that of LIQSS1 until line 14. Afterwards it continues as follows:

Algorithm 4: mLIQSS1.

```
15  for each j ∈ [1,n] such that (i≠j and A_ij · A_ji ≠ 0)
16    e = t − t_j^x // elapsed time since last xj update
17    x_j^aux = x_j + ẋ_j · e // store updated j−th state value on
          auxiliary variable
18    u_ji = u_jj − A_ji · q_i^− // affine coefficient
19    ẋ_j^+ = A_ji · q_i + A_jj · q_j + u_ji // next j−th state der. est.
20    if (ẋ_j · ẋ_j^+ < 0) // update in qi => change of sign in dxj/dt
21      q_j^+ = x_j^aux + sign(ẋ_j^+) · ΔQ_j // update qj in future xj's
            direction
22      u_ij = u_ii − A_ij · q_j // affine coefficient
23      ẋ_i^+ = A_ii · q_i + A_ij · q_j^+ + u_ij // next i−th state der. est.
24      if (ẋ_i · ẋ_i^+ < 0) // update in qj => change of sign in dxi/dt
```

```
25          // presence of oscillations
26          q_j^- = q_j //store previous value of qj
27          ẋ_j^- = ẋ_j //store previous value of dxj/dt
28          h = MAX_BE_STEP_SIZE(x_i, x_j^aux) // maximum BE step size
                as defined in Eq.(14)
29          [q_i, q_j] = BE_step(x_i, x_j^aux, h) // qi and qj are computed
                using a BE step size h from xi and xj
30          t_j^q = t // last qj update
31          t_j^η = min(τ > t) subject to x_j(τ) = q_j or |q_j − x_j(τ)| = 2ΔQ_j //
                compute the time of the next change in the j-th
                quantized state
32          for each k ∈ [1, n] such that ẋ_k depends on q_j
33            e = t − t_k^x // elapsed time since last xk update
34            x_k = x_k + ẋ_k · e // update k-th state value
35            ẋ_k^- = ẋ_k //store previous value of dxk/dt
36            ẋ_k = f_k(q, t) // recompute k-th state derivative
37            t_k^η = min(τ > t) subject to x_k(τ) = q_k or |q_k − x_k(τ)| = 2ΔQ_k
                  // recompute the time of the next change in the
                  k-th quantized state
38            A_{k,j} = (ẋ_k − ẋ_k^-)/(q_j − q_j^-) // Jacobian
39            if k ≠ j then t_k^x = t // last xk update
40          end for
41          A_{j,j} = (ẋ_j − ẋ_j^-)/(q_j − q_j^-) // Jacobian diagonal entry
42          u_{j,j} = ẋ_j(t) − A_{j,j} · q_j // affine coefficient
43        end if
44      end if
45    end for
46    for each j ∈ [1, n] such that ẋ_j depends on q_i
47      e = t − t_j^x // elapsed time since last xj update
48      x_j = x_j + ẋ_j · e // update j-th state value
49      ẋ_j^- = ẋ_j //store previous value of dxj/dt
50      ẋ_j = f_j(q, t) // recompute j-th state derivative
51      t_j^η = min(τ > t) subject to x_j(τ) = q_j or |q_j − x_j(τ)| = 2ΔQ_j //
            recompute the time of the next change in the j-th
            quantized state
52      A_{j,i} = (ẋ_j − ẋ_j^-)/(q_i − q_i^-) // Jacobian
53        if j ≠ i then t_j^x = t // last xj update
54    end for
55    // update linear approximation coefficients
56    A_{i,i} = (ẋ_i − ẋ_i^-)/(q_i − q_i^-) // Jacobian diagonal entry
57    u_{i,i} = ẋ_i(t) − A_{i,i} · q_i // affine coefficient
58    t_i^x = t // last xi update
59  end while
```

Notice that this new algorithm adds the calculation of a simultaneous step on states x_i and x_j (lines 28–29), but this only takes place under the occurrence of oscillations. In other case, the algorithm only has some additional calculations to detect changes in the signs in the state derivatives, which

requires estimating them (lines 19 and 23) and estimating also the complete Jacobian matrix (lines 38, 41, 52 and 56) and different affine coefficients.

Let us now return to the previous example, now simulating the system of Eqs. (5) with mLIQSS1 algorithm using the same initial conditions.

As we discussed earlier, the algorithm behaves identically to LIQSS1 until the oscillation condition is detected. This situation occurs at time t_2, when updating variable q_2 would provoke that $\dot{x}_1$ goes from $\dot{x}_2(t_2^-) = 0$ to $\dot{x}_j(t_2^+) = -1$ (as it occurred in LIQSS1).

Here, the modified algorithm detects the situation and checks if the subsequent change in q_1 provokes a change of sign on $\dot{x}_2$. By proposing $q_1(t_2^+) = x_1 - \Delta Q$, it results that the state derivative $\dot{x}_2(t_2^+) = -2$ changes its sign. Thus, a future oscillation between both variables is predicted and a simultaneous Backward Euler step is performed in both variables. As the states are close to an equilibrium point, the Backward Euler step size is until final time t_f and the quantized state take the equilibrium values $q_1 = -0.5 - q_2 = 0.7$. With these values, both state derivatives are null, so the simulation finishes without any oscillations, as depicted in Fig.3.

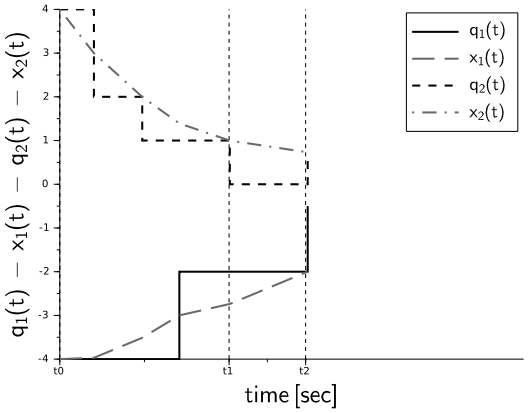


Figure 3: Simulation of system (5) with the modified LIQSS1 algorithm.

3.4 An Improved Version of LIQSS2

Before presenting the modified LIQSS2 algorithm, we shall introduce a change to the original LIQSS2 formulation of [5]. This change, which is necessary to define the mLIQSS2 algorithm, will improve the performance of LIQSS2 algorithm even in simpler cases where oscillations between pairs of variables do not appear.

The original definition of LIQSS2 combined the ideas of QSS2 and LIQSS1, where the quantized states $q_i(t)$ follow piecewise linear trajectories such that, at the end of each segment, they reach the states, i.e., $q_i(t+h) = x_i(t+h)$. However, the quantized state slopes were chosen such that they coincide with the state derivatives at the beginning of the step, i.e., $\dot{q}_i(t) = \dot{x}_i(t)$.

In order to extend the idea of mLIQSS1, we shall need to reformulate LIQSS2 such that the quantized state slopes and the state derivatives coincide at the end of the step, i.e., $\dot{q}_i(t+h) = \dot{x}_i(t+h)$.

For that goal, q_i and $\dot{q}_i$ are computed in order to verify the following equations:

$$\begin{aligned} \dot{q}_i &= \dot{x}_i + h \cdot \ddot{x}_i = A_{i,i} \cdot q_i + u_{i,i} + h \cdot (A_{i,i} \cdot \dot{q}_i + \dot{u}_{i,i}) \\ q_i + h \cdot \dot{q}_i &= x_i + h \cdot \dot{x}_i + \frac{h^2}{2} \cdot \ddot{x}_i \\ &= x_i + h \cdot (A_{i,i} \cdot q_i + u_{i,i}) + \frac{h^2}{2} \cdot (A_{i,i} \cdot \dot{q}_i + \dot{u}_{i,i}) \end{aligned} \tag{15}$$

where $\dot{u}_{i,i}$ is the affine coefficient slope. Notice that the first equation says that the quantized state slope $\dot{q}_i$ is equal to the state derivative $\dot{x}_i$ at time $t+h$. The second equation says that the quantized state q_i is equal to the state x_i at time $t+h$.

Here, h is computed as the maximum step size on Eq.(15) such that $|q_i - x_i| \leq \Delta Q_i$. Similarly to mLIQSS1, this value of h can be found analytically (i.e., by solving Eq.(15) for h using $q_i = x_i \pm \Delta Q_i$) or numerically.

Then, the LIQSS2 simulation algorithm works as follows:

Algorithm 5: LIQSS2.

```
1  while(t < t_f) // simulate until final time tf
2    t = min(t_j^η) // adavance simulation time
3    i = argmin(t_j^η) // the i-th quantized state changes first
4    e = t - t_i^x // elapsed time since last xi update
5    // update i-th state value and its derivative
6    x_i = x_i + ẋ_i · e + 0.5 · ẍ_i · e^2
7    ẋ_i = ẋ_i + ẍ_i · e
8    ẋ_i^- = ẋ_i //store previous value of dxi/dt
9    u_ii = u_ii + e_xi · u̇_ii // affine coefficient projection
10   e = t - t_i^q // elapsed time since last qi update
11   q_i^- = q_i + e · q̇_i //store previous value of qi projected
12   h = MAX_2ND_ORDER_STEP_SIZE(x_i)
13   [q_i, q̇_i] = 2ND_ORDER_step(x_i, h)
14   t_i^q = t // last qi update
15   t_i^η = min(τ > t) subject to x_i(τ) = q_i(τ)// compute the time of
         then next change in the i-th quantized state
```

```
16    for each j ∈ [1,n] such that ẋ_j depends on q_i
17      e = t − t_j^x // elapsed time since last xj update
18      // update j−th state value and its derivatives
19      x_j = x_j + ẋ_j · e + 0.5 · ẍ_j · e^2
20      ẋ_j = f_j(q(t), t) // recompute state derivative
21      ẍ_j = ḟ_j(q(t), t) // recompute state second derivative
22      t_j^η = min(τ > t) subject to x_j(τ) = q_j(τ) or |q_j(τ) − x_j(τ)| = 2ΔQ_j
            // recompute the time of the next change in the j−th
            quantized state
23      if j ≠ i then t_j^x = t // last xj update
24    end for
25    // update linear approximation coefficients
26    A_{i,i} = (ẋ_i − ẋ_i^−) / (q_i − q_i^−) // Jacobian diagonal entry
27    u_{i,i} = ẋ_i − A_{i,i} · q_i // affine coefficient
28    u̇_{i,i} = ẍ_i − A_{i,i} · q̇_i // affine coefficient slope
29    t_i^x = t // last xi update
30  end while
```

Here, we can see that LIQSS2 steps add a few calculations to those of QSS2. It calculates the maximum second order step size h (line 12) and uses it to compute the quantized state and its derivative (line 13). It also calculates the Jacobian main diagonal entry $A_{i,i}$ and the affine coefficients $u_{i,i}$ and $\dot{u}_{i,i}$ (lines 26–28).

Like in LIQSS1, the algorithm checks a condition to ensure that the difference between x_j and q_j is still bounded (by $2\Delta Q_j$) if a change in the quantized state value q_i causes that x_j no longer approaches q_j (line 22).

3.5 Modified LIQSS2 (mLIQSS2)

The modified LIQSS2 algorithm combines the ideas of LIQSS2 and the mLIQSS1 methods. That is, the algorithm works identically to LIQSS2 until a chain of significant changes in pairs of state derivatives x_i and x_j is detected[5]. We say that there is a chain of significant changes when the update of a quantized state q_i produces an abrupt change in $\dot{x}_j$ that triggers a change in q_j that in turn produces back an abrupt change in $\dot{x}_i$.

Under this situation, a simultaneous change in the quantized states q_i and q_j and their slopes $\dot{q}_i$ and $\dot{q}_j$ is applied following a second order accurate backward formula.

[5]In the second order accurate algorithm we not only check for changes in the sign, but also for significant changes in the values of the state derivatives as they would possibly lead to sequences of small steps. The reason is that when a change in the quantized state q_i provokes an abrupt change in some state derivative $\dot{x}_j$, this will provoke that x_j and q_j split from each other, what soon provokes a new step in q_j. If this new step also provokes a large change in $\dot{x}_i$, then a fast oscillation appears.

In order to predict the future first state derivatives, $\dot{x}_i$ and $\dot{x}_j$, the linear approximation of Eq.(8) is used. Additionally the future second state derivatives, $\ddot{x}_i$ and $\ddot{x}_j$ are estimated by differentiating the linear approximation of Eq.(8), obtaining the following system of equations.

$$\begin{aligned} \ddot{x}_i &= A_{ii} \cdot \dot{q}_i + A_{ij} \cdot \dot{q}_j + \dot{u}_{ij} \\ \ddot{x}_j &= A_{ji} \cdot \dot{q}_j + A_{jj} \cdot \dot{q}_j + \dot{u}_{ji} \end{aligned} \tag{16}$$

where two new coefficients, $\dot{u}_{ij}$ and $\dot{u}_{ji}$ appear, corresponding to the affine coefficient slopes.

When a new value of the quantized state q_i does not cause a significant change in any other state derivative, the algorithm works identically to LIQSS2. However, when the new value of q_i provokes that $\dot{x}_j$ or $\ddot{x}_j$ changes significantly, we propose a new value for q_j in the new direction of x_j. Then, we check if that proposed value for q_j changes $\dot{x}_i$ or $\ddot{x}_i$ as well. If it does not, we forget about the change in q_j and the algorithm follows identical steps to those of LIQSS2. Otherwise, we know that an oscillation may appear between states x_i and x_j, so we compute both quantized states q_i and q_j and their slopes simultaneously using a second order accurate backward step on Eq.(8).

Using definitions of Eqs.(9)–(10), and also defining:

$$\dot{\mathbf{q}}_{ij} \triangleq \begin{bmatrix} \dot{q}_i \\ \dot{q}_j \end{bmatrix} \ , \ \ddot{\mathbf{x}}_{ij} \triangleq \begin{bmatrix} \ddot{x}_i \\ \ddot{x}_j \end{bmatrix} \ , \ \dot{\mathbf{u}}_{ij} \triangleq \begin{bmatrix} \dot{u}_{ij} \\ \dot{u}_{ji} \end{bmatrix}$$

the backward step is given by the equations

$$\dot{\mathbf{q}}_{ij} = \dot{\mathbf{x}}_{ij} + h \cdot \ddot{\mathbf{x}}_{ij} = \mathbf{A}_{ij} \cdot \mathbf{q}_{ij} + \mathbf{u}_{ij} + h \cdot (\mathbf{A}_{ij} \cdot \dot{\mathbf{q}}_{ij} + \dot{\mathbf{u}}_{ij}) \tag{17a}$$

$$\begin{aligned} \mathbf{q}_{ij} + h \cdot \dot{\mathbf{q}}_{ij} &= \mathbf{x}_{ij} + h \cdot \dot{\mathbf{x}}_{ij} + \frac{h^2}{2} \cdot \ddot{\mathbf{x}}_{ij} \\ &= \mathbf{x}_{ij} + h \cdot (\mathbf{A}_{ij} \cdot \mathbf{q}_{ij} + \mathbf{u}_{ij}) + \frac{h^2}{2} \cdot (\mathbf{A}_{ij} \cdot \dot{\mathbf{q}}_{ij} + \dot{\mathbf{u}}_{ij}) \end{aligned} \tag{17b}$$

Notice that Eq. (17a) says that both quantized state slopes $\dot{\mathbf{q}}_{ij}$ are equal to the corresponding first state derivatives $\dot{\mathbf{x}}_{ij}$ at time $t + h$. Then Eq. (17b) says that the quantized states $\mathbf{q}_{ij}$ are equal to the states $\mathbf{x}_{ij}$ at time $t + h$.

Here, h is computed as the maximum step size on Eqs.(17) such that $|q_i - x_i| \leq \Delta Q_i$ and $|q_j - x_j| \leq \Delta Q_j$. Similarly to mLIQSS1, this value of h can be found analytically (i.e., by solving Eqs.(17) for h using $q_i = x_i \pm \Delta Q_i$ and $q_j = x_j \pm \Delta Q_j$) or numerically.

Then, the resulting algorithm for mLIQSS2 is identical to that of LIQSS2 until the point at which we update a quantized state and we now need to check if an oscillation between two variables may occur. Thus, after line 15 of Algorithm 5, the modified method continues as follows:

Algorithm 6: mLIQSS2.

16 **for each** $j \in [1,n]$ such that (i≠j and $A_{ij} \cdot A_{ji} \neq 0$)
17 $e = t - t_j^x$ // elapsed time since last xj update
18 $x_j^{aux} = x_j + \dot{x}_j \cdot e + 0.5 \cdot \ddot{x}_j \cdot e^2$ // store updated j−th state value on auxiliary variable
19 $u_{j,j} = u_{j,j} + e \cdot \dot{u}_{j,j}$ // affine coefficient projection
20 $e = t - t_j^q$ // elapsed time since last qj update
21 $q_j^- = q_j + e \cdot \dot{q}_j$ //store previous value of qj projected
22 $\dot{x}_j^- = \dot{x}_j$ //store previous value of dxj/dt
23 $u_{j,i} = u_{j,j} - A_{j,i} \cdot q_i^-$ // affine coefficient
24 $\dot{x}_j^+ = A_{j,i} \cdot q_i + A_{j,j} \cdot q_j^- + u_{j,i}$ // next j−th state der. est.
25 $\dot{u}_{j,i} = \dot{u}_{j,j} - A_{j,j} \cdot q_i^-$ // affine coefficient
26 $\ddot{x}_j^+ = A_{j,i} \cdot \dot{q}_i + A_{j,j} \cdot \dot{q}_j + \dot{u}_{j,i}$ // next j−th state 2nd der. est.
27 **if** $(|\dot{x}_j - \dot{x}_j^+| > |\dot{x}_j + \dot{x}_j^+|/2$ or $|\ddot{x}_j - \ddot{x}_j^+| > |\ddot{x}_j + \ddot{x}_j^+|/2)$ // update in qi => significant change in dxj/dt or ddxj/dt
28 $q_j^+ = x_j^{aux} - sign(\ddot{x}_j^+) \cdot \Delta Q_j$ // update qj in future xj's direction
29 $\dot{q}_j^+ = (A_{j,i} \cdot (q_i + h \cdot \dot{q}_i) + A_{j,j} \cdot q_j^+ + u_{j,i} + h \cdot \dot{u}_{j,i})/(1 - h \cdot A_{j,j})$
30 $u_{i,j} = u_{i,i} - A_{i,j} \cdot q_j^-$ // affine coefficient
31 $\dot{x}_i^+ = A_{i,i} \cdot q_i + A_{i,j} \cdot q_j^+ + u_{i,j}$ // next i−th state der. est.
32 $\dot{u}_{i,j} = \dot{u}_{i,i} - A_{i,j} \cdot \dot{q}_j^-$ // affine coefficient
33 $\ddot{x}_i^+ = A_{i,i} \cdot \dot{q}_i + A_{i,j} \cdot \dot{q}_j^+ + \dot{u}_{i,j}$ // next i−th state 2nd der. est.
34 **if** $(|\dot{x}_i - \dot{x}_i^+| > |\dot{x}_i + \dot{x}_i^+|/2$ or $|\ddot{x}_i - \ddot{x}_i^+| > |\ddot{x}_i + \ddot{x}_i^+|/2)$ // update in qj => significant change in dxi/dt or ddxi/dt
35 // presence of oscillations
36 $h = \text{MAX\_2ND\_ORDER\_STEP\_SIZE}(x_i, x_j^{aux})$
37 $[q_i, \dot{q}_i, q_j, \dot{q}_j] = \text{2ND\_ORDER\_step}(x_i, x_j^{aux}, h)$
38 $t_j^q = t$ // last qj update
39 $t_j^\eta = \mathbf{min}(\tau > t)$ subject to $x_j(\tau) = q_j(\tau)$ or $|q_j(\tau) - x_j(\tau)| = 2\Delta Q_j$ // compute the time of the next change in the j−th quantized state
40 **for each** $k \in [1,n]$ such that $\dot{x}_k$ depends on q_j
41 $e = t - t_k^x$ // elapsed time since last xk update
42 // update k−th state value and its derivatives
43 $x_k = x_k + \dot{x}_k \cdot e + 0.5 \cdot \ddot{x}_k \cdot e^2$
44 $\dot{x}_k^- = \dot{x}_k$ //store previous value of dxk/dt
45 $\dot{x}_k = f_j(\mathbf{q}(t), t)$ // recompute state derivative
46 $\ddot{x}_k = \dot{f}_k(\mathbf{q}(t), t)$ // recompute state second derivative
47 $t_k^\eta = \mathbf{min}(\tau > t)$ subject to $x_k(\tau) = q_k(\tau)$ or $|q_k(\tau) - x_k(\tau)| = 2\Delta Q_j$ // recompute the time of the next change in the k−th quantized state
48 $A_{k,j} = (\dot{x}_k - \dot{x}_k^-)/(q_j - q_j^-)$ // Jacobian
49 **if** $k \neq j$ **then** $t_k^x = t$ // last xk update
50 **end for**
51 // update linear approximation coefficient
52 $A_{j,j} = (\dot{x}_j - \dot{x}_j^-)/(q_j - q_j^-)$ // Jacobian diagonal entry

```
53          u_{j,j} = ẋ_j − A_{j,j} · q_j // affine coefficient
54          t_j^x = t // last xj update
55        end if
56      end if
57    end for
58    for each j ∈ [1,n] such that ẋ_j depends on q_i
59      e = t − t_j^x // elapsed time since last xj update
60      // update j−th state value and its derivatives
61      x_j = x_j + ẋ_j · e + 0.5 · ẍ_j · e^2
62      ẋ_j^− = ẋ_j //store previous value of dxj/dt
63      ẋ_j = f_j(q(t), t) // recompute state derivative
64      ẍ_j = f̈_j(q(t), t) // recompute state second derivative
65      t_j^η = min(τ > t) subject to x_j(τ) = q_j or |q_j − x_j(τ)| = 2ΔQ_j //
            recompute the time of the next change in the j−th
            quantized state
66      A_{j,i} = (ẋ_j − ẋ_j^−)/(q_i − q_i^−) // Jacobian
67      if j ≠ i then t_j^x = t // last xj update
68    end for
69    // update linear approximation coefficients
70    A_{i,i} = (ẋ_i − ẋ_i^−)/(q_i − q_i^−) // Jacobian diagonal entry
71    u_{i,i} = ẋ_i − A_{i,i} · q_i // affine coefficient
72    u̇_{i,i} = ẍ_i − A_{i,i} · q̇_i // affine coefficient slope
73    t_i^x = t // last xi update
74  end while
```

Compared with LIQSS2, the modified algorithm adds the calculation of a simultaneous step on states x_i and x_j (lines 36–37), but these calculations only take place under the prediction of oscillations. In other case, the algorithm only has a few additional calculations to predict significant changes in the state derivatives[6], what requires estimating them (lines 24, 26, 31 and 33) and estimating also the complete Jacobian matrix (lines 48, 52, 66 and 70) as well as the different affine coefficients.

3.6 Properties of modified LIQSS methods

Like the original algorithms, the modified versions of LIQSS1 and LIQSS2 ensure that the difference between each state x_i and the corresponding quantized state q_i are bounded by $2\Delta Q_i$. Thus, the new algorithms provide an analytic solution of Eq.(2), that, defining $\Delta\mathbf{x}(t) \triangleq \mathbf{q}(t) - \mathbf{x}(t)$, can be rewrit-

[6]We predict a significant change in the state derivative under the condition $|\dot{x}_i - \dot{x}_i^+| > |\dot{x}_i + \dot{x}_i^+|/2$. This condition is equivalent to consider a change in the sign or a change where the new derivatives is at least three times larger or smaller than the old one. A different condition can be used, but this one establishes a good trade off between actually detecting changes that lead to oscillations and not performing unnecessary simultaneous state updates.

ten as

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t) + \Delta\mathbf{x}(t), t) \tag{18}$$

Notice that the original system of Eq.(1) only differs from the approximation of Eq.(18) by the presence of a bounded *perturbation* term $\Delta\mathbf{x}(t)$, with each component of this perturbation being bounded as $|\Delta x_i(t)| \leq 2\Delta Q_i$.

Since the original properties of LIQSS1 and LIQSS2 are based on the presence of these bounded perturbations, the modified algorithms satisfy exactly the same properties:

- Convergence: Assuming that $\mathbf{f}(\mathbf{x}, t)$ is Lipschitz in $\mathbf{x}$ and piecewise continuous in t, then, the approximate solution of Eq.(18) goes to the analytical solution of Eq.(1) when the quantum in all variables ΔQ_i goes to zero [4].

- Practical Stability. Assuming that the analytical solution of Eq.(1) is asymptotically stable around an equilibrium point, the usage of a quantum small enough ensures that the approximate solution of Eq.(18) finishes inside an arbitrary small region around that equilibrium point [4].

- Global Error Bound. In a Linear Time Invariant case, i.e., when

  $$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \tag{19}$$

  provided that matrix $\mathbf{A}$ is Hurwitz (i.e., the LTI system is asymptotically stable), the maximum error committed in a simulation is bounded by

  $$|\mathbf{e}(t)| \preceq |\mathbf{V}||\text{Real}(\mathbf{\Lambda})^{-1}\mathbf{\Lambda}||\mathbf{V}^{-1}|\mathbf{\Delta Q}$$

  where $\mathbf{\Lambda} = \mathbf{V}^{-1}\mathbf{A}\mathbf{V}$ is the Jordan canonical decomposition of $\mathbf{A}$, $\mathbf{\Delta Q}$ is the vector of quanta, the symbol '$|\cdot|$' computes the elementwise absolute value of a matrix or vector, and '$\preceq$' represents a componentwise inequality [5].

3.7 Implementation of Modified LIQSS methods

The modified algorithms where implemented in the Stand Alone QSS Solver [7]. For that purpose, the pseudo codes of Algorithms 4 and 6 were programmed as plain C functions of the QSS solver. The corresponding codes are available at `https://sourceforge.net/projects/qssengine/`.

In our implementation, the value of h according to Eq.(14) is found using iterations, as they are faster than finding the analytical solution. Anyway,

those iterations do not attempt to find the exact value, but a sufficiently large value for h verifying the inequalities of Eq.(13). For that purpose, we first check if the inequalities are accomplished using $h = t_f - t$, where t_f is the final time of the simulation. If it is not the case, we try with a step size with value $h = \Delta Q_i / |\dot{x}_i|$ (this is the step size $QSS1$ would calculate). If this step size does not verify the inequalities, then we reduce h using a linear approximation of the dependence of $|q_i - x_i|$ on h. If it still fails, the algorithm just perform a single LIQSS1 step.

The improved LIQSS2 and mLIQSS2 use a very similar approach.

4 Examples and Results

This section shows simulation results, comparing the performance of the modified algorithms with that of their former versions and classic solvers (DOPRI and DASSL).

In order to perform this comparison, we run a set of experiments on different models according to the conditions described below:

- The simulations were performed on an AMD A4-3300 APU@2.5GHz PC under Ubuntu OS.
- In all cases, we measured the CPU time, the number of scalar function evaluations and the relative error, computed as:

$$e_{rr} = \sqrt{\frac{\sum (out[k] - out_{REF}[k])^2}{\sum out_{REF}[k]^2}} \tag{20}$$

  where the reference solution $out_{REF}[k]$ was obtained using DASSL with a very small error tolerance (10^{-9}).

4.1 1D Advection-Reaction-Diffusion (ADR) problem

Advection-diffusion equations provide the basis for describing heat and mass transfer phenomena as well as processes of continuum mechanics, where the physical quantity of interest could be temperature in heat conduction or concentration of some chemical substance. In several applications these phenomena occur in presence of chemical reactions, leading to the ADR equation, a problem frequently found in many areas of environmental sciences as well as in mechanical engineering.

ADR problems discretized with the method of lines lead to large stiff systems of ODEs where the use of LIQSS methods have shown important advantages over classic discrete time algorithms [15].

The following set of ODEs, taken from [15], corresponds to the spatial discretization of a 1D ADR problem:

$$\frac{\mathrm{d}u_i}{\mathrm{d}t} = -a \cdot \frac{u_i - u_{i-1}}{\Delta x} + d \cdot \frac{u_{i+1} - 2 \cdot u_i + u_{i-1}}{\Delta x^2} + r \cdot (u_i^2 - u_i^3)$$

for $i = 1, \ldots, N-1$ and

$$\frac{\mathrm{d}u_N}{\mathrm{d}t} = -a \cdot \frac{u_N - u_{N-1}}{\Delta x} + d \cdot \frac{2 \cdot u_{N-1} - 2 \cdot u_N}{\Delta x^2} + r \cdot (u_N^2 - u_N^3)$$

where $N = 1000$ is the number of grid points, and $\Delta x = \frac{10}{N}$.

We consider parameters $a = 1$, $d = 0.001$, $r = 1000$, and initial conditions

$$u_i(x, t = 0) = \begin{cases} 1 & \text{if } i \in [1, N/5] \\ 0 & \text{otherwise} \end{cases}$$

We simulated this system until a final time $t_f = 10$ using DASSL and DOPRI classic solvers as well as the original and the modified versions of LIQSS2. A result of these simulations is shown in Fig. 4. In all cases, we simulated using two different tolerance settings. The results are reported in Table 2.

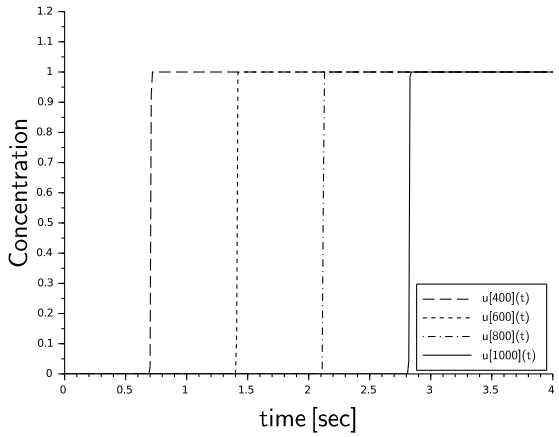


Figure 4: 1D ADR problem simulation results.

As it was already reported in [15], LIQSS2 overperforms DOPRI and DASSL. However, the modified version of LIQSS2 presented in this work is more than two times faster than the original one, extending the previously

Integration Method		Relative Error	Function f_i Evaluations	CPU [msec]
DASSL	$tol = 1 \cdot 10^{-3}$	$8.28 \cdot 10^{-}2$	539,685	42,870
	$tol = 1 \cdot 10^{-5}$	$4.95 \cdot 10^{-}4$	35,186	31,237
DOPRI	$tol = 1 \cdot 10^{-3}$	$8.86 \cdot 10^{-}4$	36,184	1,032
	$tol = 1 \cdot 10^{-5}$	$1.69 \cdot 10^{-}4$	41,890	1,289
old LIQSS2	$\Delta Q_i = 1 \cdot 10^{-3}$	$1.59 \cdot 10^{-}3$	405,104	55
	$\Delta Q_i = 1 \cdot 10^{-5}$	$4.35 \cdot 10^{-}5$	2,764,382	362
mLIQSS2	$\Delta Q_i = 1 \cdot 10^{-3}$	$2.82 \cdot 10^{-}3$	140,812	22
	$\Delta Q_i = 1 \cdot 10^{-5}$	$1.98 \cdot 10^{-}5$	1,084,484	157

Table 2: 1D Advection-Reaction-Diffusion problem results comparison.

reported advantages. That way, for the *standard* tolerance of 10^{-3} mLIQSS2 is almost 50 times faster than DOPRI and about 2000 times faster than DASSL. When a more stringent tolerance is used (10^{-5}) the advantages becomes less noticeable. This is due to the fact that mLIQSS2 is only second order accurate while DASSL and DOPRI are fifth order algorithms.

In this case, the advantage of mLIQSS2 over the original LIQSS2 is not due to the structure of the Jacobian matrix. The reason is that mLIQSS2 uses not only the future state value but also the future state slope to compute the quantized state trajectory.

4.2 Interleaved Ćuk Converter

The simulation of switching power converters is another case where LIQSS methods have important advantages over classic numerical integration algorithms. Here, the efficient treatment of discontinuities and stiffness is the main reason of the advantages [16]. However, as it was reported in the cited reference, when large entries appear at both sides of the main diagonal of the Jacobian matrix, LIQSS methods fail due to the appearance of fast oscillations between pairs of variables. An example of this occurs in the Ćuk Converter.

In order to verify that the modified LIQSS algorithms solve this problem, we simulated the circuit corresponding to a four-stage Ćuk interleaved converter shown in Fig.5.

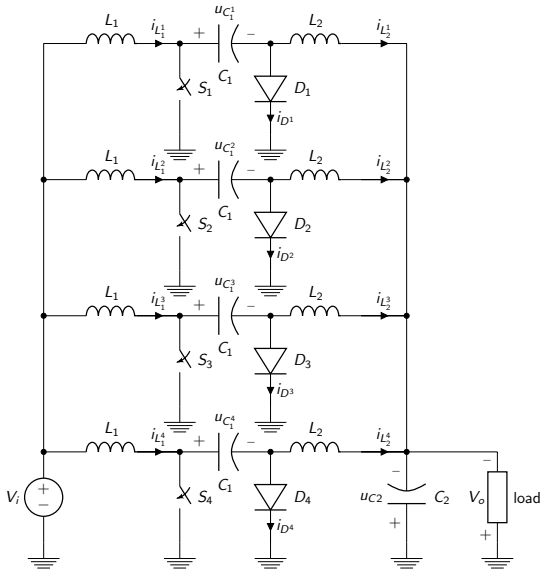


Figure 5: Four-stage Ćuk interleaved converter circuit.

The state equations of the j–th stage of the power electronic converter can be written as follows:

$$\frac{\mathrm{d}i_{L_1^j}}{\mathrm{d}t} = \frac{U - u_{C_1^j} - i_{D^j} \cdot R_{S^j}}{L_1}$$
$$\frac{\mathrm{d}i_{L_2^j}}{\mathrm{d}t} = \frac{-u_{C_2} - i_{D^j} \cdot R_{S^j}}{L_2}$$
$$\frac{\mathrm{d}u_{C_1^j}}{\mathrm{d}t} = \frac{i_{D^j} - i_{L_2^j}}{C_1}$$

with

$$i_{D^j} = \frac{(i_{L_1^j} + i_{L_2^j}) \cdot R_{S^j} - u_{C_1^j}}{R_{S^j} + R_{D^j}}$$

where R_{S^j} and R_{D^j} are the resistances of switches and diodes correspondingly, which can all take one of two values, whether R_{ON} or R_{OFF} depending on their state. Finally, the output voltage obeys to the following equation:

$$\frac{\mathrm{d}u_{C_2}}{\mathrm{d}t} = \frac{\sum_{j=1}^{N} i_{L_2^j} - \frac{u_{C_2}}{R_o}}{C_2}$$

The model was simulated with the following set of parameters:

- Input source voltage: $U = 24V$
- Capacities: $C_1 = 10^{-4}F$ and $C_2 = 10^{-4}F$
- Inductances $L_1 = 10^{-4}Hy$ and $L_2 = 10^{-4}Hy$
- Load resistance: $R_o = 10\Omega$
- Switch and diode On-state resistance: $R_{ON} = 10^{-5}\Omega$
- Switch and diode On-state resistance: $R_{OFF} = 10^{5}\Omega$
- Switch control signal period: $T = 10^{-4}sec$
- Switch control signal duty cycle: $DC = 0.25$

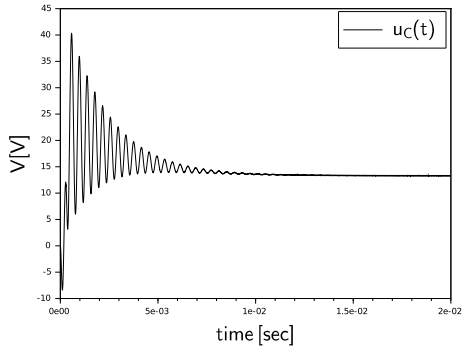
The system was simulated until a final time $t_f = 0.02sec$, and the results are shown in Fig.6.

The performance comparison of the different algorithms is reported in Table 3 for two different tolerance settings. DOPRI results were not reported as the system is too stiff and it failed to provide results in a reasonable CPU time. Additional results obtained with the first order accurate methods, LIQSS1 and mLIQSS1, can be found in [6].

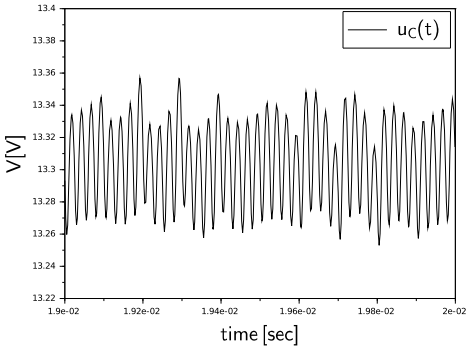
Integration Method		**Relative Error**	**Function f_i Evaluations**	**CPU [msec]**
DASSL	$tol = 1 \cdot 10^{-3}$	$1.74 \cdot 10^{-}2$	$5,697,757$	285
	$tol = 1 \cdot 10^{-5}$	$1.52 \cdot 10^{-}4$	$7,056,634$	390
old LIQSS2	$\Delta Q_i = 1 \cdot 10^{-3}$	$1.96 \cdot 10^{-}3$	$216,086,632$	$33,747$
	$\Delta Q_i = 1 \cdot 10^{-5}$	$3.10 \cdot 10^{-}5$	$335,675,912$	$49,069$
mLIQSS2	$\Delta Q_i = 1 \cdot 10^{-3}$	$1.23 \cdot 10^{-}3$	$341,928$	60
	$\Delta Q_i = 1 \cdot 10^{-5}$	$1.64 \cdot 10^{-}5$	$2,021,402$	349

Table 3: 4-Stage Interleaved Ćuk converter results comparison.

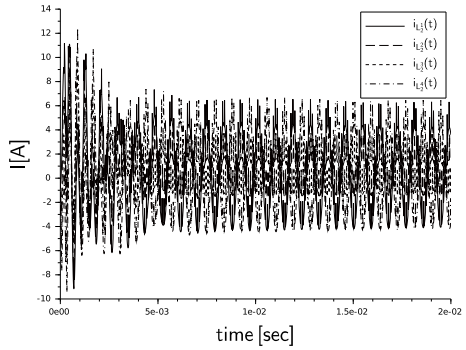
Here we can see that, as expected, LIQSS2 shows a very poor performance. However, mLIQSS2 is faster and more accurate than DASSL for both error tolerances. Moreover, for the *standard* tolerance of 10^{-3}, mLIQSS2 is more than four times faster and ten times more accurate. Then, for a tolerance of 10^{-5} mLIQSS2 is still faster and more accurate but, as in the previous



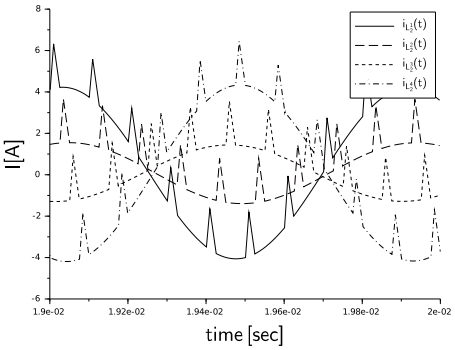
(a) Output voltage.



(b) Output voltage detail.



(c) Currents $i_{L_2^j}$.



(d) Currents $i_{L_2^j}$ detail.

Figure 6: Four-stage Ćuk converter simulation results.

example, the difference is less noticeable. Again, the reason is that the tolerance of 10^{-5} is not appropriate for a second order accurate algorithm.

In order to check how the computational costs grow with the circuit complexity, we also simulated the model varying the size from 4 to 32 stages. In order to perform a fair comparison, we set the tolerance of each solver so that the measured error are the same. The results are shown in Fig.7. There, mLIQSS2 CPU times grow about linearly with the number of stages, while DASSL times grow more than quadratically.

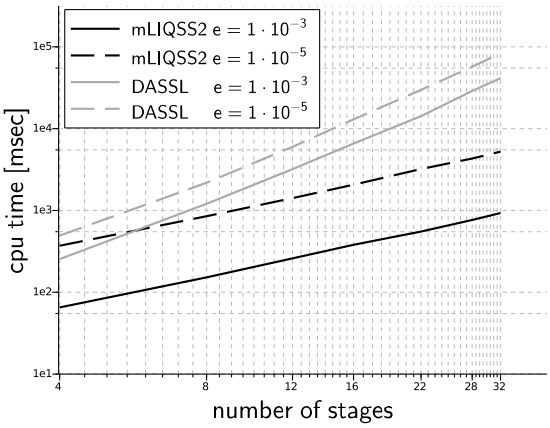


Figure 7: Simulation Time comparison: Four-stage Ćuk interleaved converter for tolerances $1 \cdot 10^{-3}$ and $1 \cdot 10^{-5}$.

4.3 Tyson model: Cdc2 and cyclin interactions

Another application where LIQSS methods exhibit advantages over classic discrete time algorithms is that of certain biological models [17]. However, in some of these models, the Jacobian matrix contains large entries at both sides of the main diagonal and LIQSS methods provoke spurious oscillations. A case where this happens is the classic Tyson model of *cdc2 and cyclin interactions*, presented in [18], that represents the creation and degradation of cyclin in a cell.

In order to verify that mLIQSS algorithms work in this case, we considered a model composed of 100 individual cells starting from different initial conditions.

The equations for a single cell are the following ones:

$$\begin{aligned}
\frac{\mathrm{d}C2}{\mathrm{d}t} &= k_6 \cdot M - k_8 \cdot [P] \cdot C2 + k_9 \cdot CP \\
\frac{\mathrm{d}CP}{\mathrm{d}t} &= -k_3 \cdot CP \cdot Y + k_8 \cdot [P] \cdot C2 - k_9 \cdot CP \\
\frac{\mathrm{d}pM}{\mathrm{d}t} &= k_3 \cdot CP \cdot Y - pM \cdot F(M) + k_5 \cdot [P] \cdot M \\
\frac{\mathrm{d}M}{\mathrm{d}t} &= pM \cdot F(M) - k_5 \cdot [P] \cdot M - k_6 \cdot M \\
\frac{\mathrm{d}Y}{\mathrm{d}t} &= k1 \cdot [aa] - k_2 \cdot Y - k_3 \cdot CP \cdot Y \\
\frac{\mathrm{d}YP}{\mathrm{d}t} &= k_6 \cdot M - k_7 \cdot YP
\end{aligned}$$

where $[P]$ and amino acids $[aa]$ concentrations are assumed to be constant. There are six state variables : the concentrations of cc2 ($C2$), cdc2-p (CP), preMPF = P-cyclin-cdc2-P (pM), active MPF = P-cyclin-cdc2 (M), cyclin (Y) and cyclin-P (YP). $F(M)$ is a function that describes the autocatalytic feedback of active MPF on its own production. All constants definitions $k_{1,\ldots,9}$ and further explanations can be found in [18].

As in the previous cases, the system was simulated with error tolerances of 10^{-3} and 10^{-5}, until a final time $t_f = 50$ using the different solvers. DOPRI results were not reported because it failed to complete the simulations due to the model stiffness.

The results are reported in Table 4. Output results of these simulations are shown in Fig. 8.

From Table 4, we can see that both modified algorithms mLIQSS1 and mLIQSS2 are significantly faster than their original versions. Moreover, for the error tolerance 10^{-3}, mLIQSS2 is more than 25 times faster than DASSL obtaining a similar error. Then, with the error tolerance 10^{-5}, mLIQSS2 is still faster obtaining also better accuracy. As expected, the original LIQSS2 algorithm was very slow.

Regarding mLIQSS1, as it is only first order accurate, it only obtains decent results for large error tolerances. Anyway, using a standard error tolerance of 10^{-3}, it is still as fast as DASSL.

5 Conclusions

A modification for the first and second order accurate Linearly Implicit Quantized State System Methods was proposed, allowing them to efficiently simulate stiff systems with more general structures than those having large entries

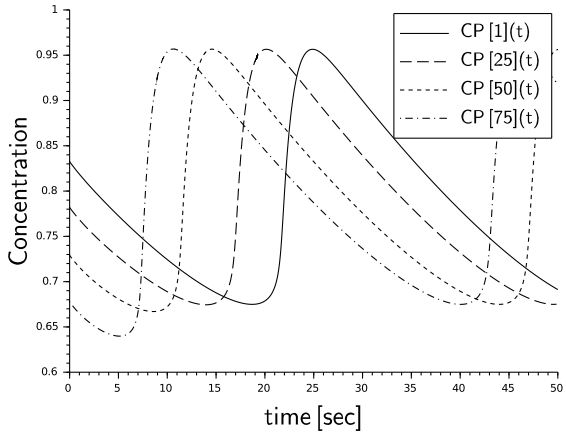


Figure 8: Tyson model simulation results.

	Integration Method	Relative Error	Function f_i Evaluations	CPU [msec]
DASSL	$tol = 1 \cdot 10^{-3}$	$3.34 \cdot 10^{-}3$	$44,062,800$	$1,648$
	$tol = 1 \cdot 10^{-5}$	$5.18 \cdot 10^{-}4$	$28,478,400$	$2,257$
old LIQSS1	$\Delta Q_i = 1 \cdot 10^{-3}$	$7.44 \cdot 10^{-}3$	$43,127,543$	$5,704$
	$\Delta Q_i = 1 \cdot 10^{-5}$	$2.11 \cdot 10^{-}5$	$134,002,162$	$20,741$
mLIQSS1	$\Delta Q_i = 1 \cdot 10^{-3}$	$7.10 \cdot 10^{-}3$	$8,672,317$	$1,704$
	$\Delta Q_i = 1 \cdot 10^{-5}$	$1.00 \cdot 10^{-}5$	$56,204,686$	$11,112$
old LIQSS2	$\Delta Q_i = 1 \cdot 10^{-3}$	$6.17 \cdot 10^{-}3$	$55,720,144$	$9,945$
	$\Delta Q_i = 1 \cdot 10^{-5}$	$1.36 \cdot 10^{-}5$	$64,628,190$	$11,376$
mLIQSS2	$\Delta Q_i = 1 \cdot 10^{-3}$	$7.53 \cdot 10^{-}3$	$185,970$	42
	$\Delta Q_i = 1 \cdot 10^{-5}$	$1.21 \cdot 10^{-}5$	$1,471,838$	320

Table 4: 100 cells - Cdc2 and cyclin interactions results comparison.

restricted to one side of the main diagonal of the Jacobian matrix. In the case of the second order algorithm, the modification also included computing the quantized state slope according to the future values of the state slope,

what allowed to obtain better results even in cases with simpler structures.

Both mLIQSS algorithms were implemented in the Stand Alone QSS Solver and tested in the simulation of some stiff systems comparing their performance with that of their original versions and those of classic solvers. The performance analysis showed that, in those cases, mLIQSS methods do not suffer the appearance of spurious oscillations exhibited by the original LIQSS algorithms. Consequently, the new algorithms are significantly faster than their predecessors and they exhibit advantages over classic algorithms like DASSL and DOPRI.

Besides the advantages demonstrated in the cases of study, the proposed methods have a remarkable feature of mixing LIQSS and discrete time implicit algorithms. When they predict that LIQSS may lead to oscillations on a certain sub-model, they apply backward steps on the corresponding state variables. That way, mLIQSS algorithms constitute the first approach to effectively combine Quantized State and classic discrete time ODE solvers.

Regarding potential applications, QSS algorithms are particularly efficient to simulate models with frequent discontinuities (like Power Electronics Models) as well as large sparse models. Thus, we expect that mLIQSS ideas lead to the efficient simulation of Power Electronic circuits where LIQSS fail: the already mentioned Ćuk converter, the different Z source topologies (that have a similar structure to that of the Ćuk), as well as more general switching converters under the presence of parasitic inductances and capacitances.

The main limitation of mLIQSS1 and mLIQSS2 is that they are only first and second order accurate respectively, so they are not efficient under low tolerance settings. Thus, we are currently working on developing higher order versions.

Besides extending mLIQSS to higher orders, future research should study the performance of this approach in a wider variety of applications.

The models of the different examples presented here are part of the distribution of the Stand Alone QSS Solver, that can be downloaded from `https://sourceforge.net/projects/qssengine/`.

Funding

This work was partially funded by UNR grant PID-ING 512.

References

[1] F. Cellier and E. Kofman, *Continuous System Simulation*. New York: Springer, 2006.

[2] E. Hairer, S. Nørsett, and G. Wanner, *Solving Ordinary Dfferential Equations I. Nonstiff Problems*. Berlin: Springer, 2nd ed., 1993.

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

[4] E. Kofman and S. Junco, "Quantized State Systems. A DEVS Approach for Continuous System Simulation," *Transactions of SCS*, vol. 18, no. 3, pp. 123–132, 2001.

[5] G. Migoni, M. Bortolotto, E. Kofman, and F. Cellier, "Linearly Implicit Quantization-Based Integration Methods for Stiff Ordinary Differential Equations," *Simulation Modelling Practice and Theory*, vol. 35, pp. 118–136, 2013.

[6] F. Di Pietro, G. Migoni, and E. Kofman, "Improving a Linearly Implicit Quantized State System Method," in *Proceedings of the 2016 Winter Simulation Conference*, (Arlington, Virginia, USA), 2016.

[7] J. Fernández and E. Kofman, "A Stand-Alone Quantized State System Solver for Continuous System Simulation.," *Simulation: Transactions of the Society for Modeling and Simulation International*, vol. 90, no. 7, pp. 782–799, 2014.

[8] E. Kofman, "Discrete Event Simulation of Hybrid Systems," *SIAM Journal on Scientific Computing*, vol. 25, no. 5, pp. 1771–1797, 2004.

[9] E. Kofman, "A Second Order Approximation for DEVS Simulation of Continuous Systems," *Simulation: Transactions of the Society for Modeling and Simulation International*, vol. 78, no. 2, pp. 76–89, 2002.

[10] E. Kofman, "A Third Order Discrete Event Simulation Method for Continuous System Simulation," *Latin American Applied Research*, vol. 36, no. 2, pp. 101–108, 2006.

[11] F. Bergero and E. Kofman, "PowerDEVS. A Tool for Hybrid System Modeling and Real Time Simulation," *Simulation: Transactions of the Society for Modeling and Simulation International*, vol. 87, no. 1–2, pp. 113–132, 2011.

[12] T. Beltrame and F. E. Cellier, "Quantised state system simulation in dymola/modelica using the devs formalism," in *Proceedings 5th International Modelica Conference*, pp. 73–82, 2006.

[13] M. C. D'Abreu and G. A. Wainer, "M/cd++: modeling continuous systems using modelica and devs," in *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2005. 13th IEEE International Symposium on*, pp. 229–236, IEEE, 2005.

[14] G. Quesnel, R. Duboz, É. Ramat, and M. K. Traoré, "Vle: a multimodeling and simulation environment," in *Proceedings of the 2007 summer computer simulation conference*, pp. 367–374, Society for Computer Simulation International, 2007.

[15] F. Bergero, J. Fernández, E. Kofman, and M. Portapila, "Time Discretization versus State Quantization in the Simulation of a 1D Advection-Diffusion-Reaction Equation.," *Simulation: Transactions of the Society for Modeling and Simulation International*, vol. 92, no. 1, pp. 47–61, 2016.

[16] G. Migoni, F. Bergero, E. Kofman, and J. Fernández, "Quantization-Based Simulation of Switched Mode Power Supplies.," *Simulation: Transactions of the Society for Modeling and Simulation International*, vol. 91, no. 4, pp. 320–336, 2015.

[17] R. Assar and D. J. Sherman, "Implementing biological hybrid systems: Allowing composition and avoiding stiffness," *Applied Mathematics and Computation*, vol. 223, pp. 167–179, 2013.

[18] J. J. Tyson, "Modeling the cell division cycle: cdc2 and cyclin interactions.," *Proceedings of the National Academy of Sciences*, vol. 88, no. 16, pp. 7328–7332, 1991.

Author Biographies

Franco Di Pietro received his BS degree in electronic engineering in 2013 from the National University of Rosario, Argentina. He is currently a PhD student at the French Argentine International Center for Information and Systems Sciences (CIFASIS) and an adjunct professor at the Department of Mathematics of the National University of Rosario (FCEIA–UNR). His research interests include numerical simulation of hybrid systems and power electronics.

Gustavo Migoni received his BS degree in electronic engineering in 2004 and a PhD degree in automatic control in 2010, both from the National University of Rosario. He is an adjunct professor at the Department of Control of the National University of Rosario (FCEIA–UNR) and holds a research position at the National Research Council of Argentina (CIFASIS–CONICET). His research interests include numerical simulation of hybrid systems and automatic control theory.

Ernesto Kofman received his BS degree in electronic engineering in 1999 and his PhD degree in automatic control in 2003, all from the National University of Rosario. He is an adjunct professor at the department of control of the National University of Rosario (FCEIA–UNR). He also holds a research position at the National Research Council of Argentina (CIFASIS–CONICET). His research interests include automatic control theory and numerical simulation of hybrid systems. He coauthored a textbook on continuous system simulation in 2006 with Springer, New York.