Linearly Implicit Quantization–Based Integration Methods for Stiff Ordinary Differential Equations.

Gustavo Migoni[a,b], Mario Bortolotto[a,b], Ernesto Kofman[a,b,*], François E. Cellier[c]

[a] *Departamento de Control. FCEIA. Universidad Nacional de Rosario. Riobamba 245 bis, (2000) Rosario, Argentina*
[b] *CIFASIS – CONICET. Argentina*
[c] *Department of Computer Science, ETH Zurich. Switzerland*

Abstract

In this paper, new integration methods for stiff ordinary differential equations (ODEs) are developed. Following the idea of quantization–based integration (QBI), i.e., replacing the time discretization by state quantization, the proposed algorithms generalize the idea of linearly implicit algorithms. Also, the implementation of the new algorithms in a DEVS simulation tool is discussed. The efficiency of these new methods is verified by comparing their performance in the simulation of two benchmark problems with that of other numerical stiff ODE solvers. In particular, the advantages of these new algorithms for the simulation of electronic circuits are demonstrated.

Keywords: Ordinary differential equation, Stiff systems, State quantization, Quantized state systems, PowerDEVS

1. Introduction.

The digital simulation of dynamic systems is an area of ongoing development both in theory and applications. On the one hand, the significant developments in computer technology (both hardware and software) in recent decades gave rise to the profusion of a large number of numerical methods

*Corresponding author

Email addresses: migonig@fceia.unr.edu.ar (Gustavo Migoni), mariob@fceia.unr.edu.ar (Mario Bortolotto), kofman@fceia.unr.edu.ar (Ernesto Kofman), fcellier@inf.ethz.ch (François E. Cellier)

for solving ordinary differential equations (ODEs), as modern computing environments allow ever more complex models to be simulated efficiently and effectively [4, 7, 8, 20]. On the other hand, the increasing complexity of models that describe modern engineering systems present new challenges for these numerical algorithms.

In this paper, we shall discuss problems related to the efficient simulation of stiff and discontinuous systems.

Many dynamical systems of practical relevance, both in science and engineering, are stiff. That is, they have Jacobian matrices with eigenvalues, the real parts of which are widely separated along the negative real axis of the complex plane.

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 [4], 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 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.

In return, implicit methods have higher computational cost than explicit ones, because they call for iterative algorithms in each step to calculate the next value. This is unacceptable in real–time applications, since it is impossible to predict beforehand, how many (Newton) iterations it will take to converge to an acceptably accurate solution, i.e., it cannot be known in advance, how much real time each simulation step will consume.

A totally different kind of ODE solvers can be formulated by replacing the time discretization by state quantization. This idea led to the *quantization based integration* (QBI) methods that approximate the differential equations by *discrete event systems* in terms of the DEVS formalism [22].

Based on this idea, originally proposed by Zeigler [21, 23], the first general purpose QBI solver developed was QSS1 (Quantized State Systems) [13], which performs a first–order approximation. Based on similar principles, second–order (QSS2) [9] and third–order (QSS3) [11] methods were also developed that offer better accuracy without increasing the number of calculations significantly.

Due to their asynchronous nature, QSS methods show important advantages when simulating discontinuous ODEs [10], and they exhibit nice stability and error bound properties.

QSS methods were later extended to deal with stiff systems. A first–order accurate method, called *Backward QSS* (BQSS) was proposed in [17]. BQSS, in spite of its backward formulation, does not call for iterations, and the method preserves most of the advantages of the explicit QSS solvers.

While BQSS could not be extended to higher order approximations, it inspired the formulation of a new family of methods that were introduced in [16], with algorithms of orders 1 and 2. This family, called *Linearly Implicit QSS* (LIQSS), combines the principles of QSS methods with those of linearly implicit classic algorithms [4].

In this article, we reformulate the definition of LIQSS methods given in [16], extending them to a generic N–th order of accuracy and providing a general algorithm for its computer coding. We also study the main stability and error bound properties of this new family of solvers.

We shall show that these algorithms are very efficient for the simulation of some classes of systems that are simultaneously stiff and discontinuous.

The paper is organized as follows. After an introduction to the QSS family of solvers provided in Section 2, the LIQSS methods are presented and defined in Section 3. Section 4 then analyzes the theoretical properties of these methods, and Section 5 discusses two simulation examples and compares the results obtained by LIQSS against those found using classic algorithms. Finally, Section 6 concludes the article.

2. Quantization Based Integration

In this section, we present the family of Quantized State System (QSS) methods. We first introduce the QSS methods of orders 1 to 3. Since papers detailing these methods have been previously published [9, 11, 13], we shall keep the discussion of these methods short and limit it to those aspects that are necessary for the understanding of the remainder of this article. We then introduce the LIQSS methods of orders 1 and 2 for stiff systems. A preliminary paper describing these methods had also been previously published [16]; however, the methods have evolved since the publication of that article. They have become more efficient and are now capable of simulating additional classes of stiff simulation models.

2.1. First–Order Quantized State Systems Method (QSS1).

2.1.1. QSS1 Definition

Given the system:

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

with analytical solution $\mathbf{x}_{\mathbf{a}}(t)$, the QSS1 approximates it by

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

Here, $\mathbf{q}$ is the *quantized state vector*. Its entries are componentwise related with those of the state vector $\mathbf{x}$ by the following *hysteretic quantization function*:

$$q_j(t) = \begin{cases} x_j(t) \text{ if } |x_j(t) - q_j(t^-)| \geq \Delta Q_j \\ q_j(t^-) \text{ otherwise} \end{cases} \tag{3}$$

where ΔQ_j is called *quantum*.

It can be seen easily that $q_j(t)$ follows a piecewise constant trajectory that only changes its value when the difference between $q_j(t)$ and $x_j(t)$ is equal to the quantum. After each change in the quantized variable, it results that $q_j(t) = x_j(t)$.

2.1.2. QSS1 and Stiff Systems

The following examples illustrate the behavior of QSS1 and its poor performance when simulating a stiff system.

Given the system

$$\begin{aligned} \dot{x}_{a1}(t) &= 0.01\, x_{a2}(t) \\ \dot{x}_{a2}(t) &= -100\, x_{a1}(t) - 100\, x_{a2}(t) + 2020 \end{aligned} \tag{4}$$

with eigenvalues $\lambda_1 \approx -0.01$ and $\lambda_2 \approx -99.99$. Consequently, the system is stiff.

The QSS method approximates it by

$$\begin{aligned} \dot{x}_1(t) &= 0.01\, q_2(t) \\ \dot{x}_2(t) &= -100\, q_1(t) - 100\, q_2(t) + 2020 \end{aligned} \tag{5}$$

Considering initial conditions $x_1(0) = 0$ and $x_2(0) = 20$ together with the quanta $\Delta Q_1 = \Delta Q_2 = 1$, the QSS numerical ODE solver performs the following steps:

- At $t = 0$, we set $q_1(0) = 0$ and $q_2(0) = 20$. Then, $\dot{x}_1(0) = 0.2$ and $\dot{x}_2(0) = 20$. This situation remains unchanged until $|q_i - x_i| = \Delta Q_i = 1$.
- The next change in q_1 is thus scheduled at $t = 1/0.2 = 5$, whereas the next change in q_2 is scheduled at $t = 1/20 = 0.05$.
- Hence a new step is performed at $t = 0.05$. After this step, it results that $q_1(0.05) = 0$, $q_2(0.05) = 21$, $x_1(0.05) = 0.01$, $x_2(0.05) = 21$. The derivatives are $\dot{x}_1(0.05) = 0.21$ and $\dot{x}_2(0.05) = -80$.
- The next change in q_1 is rescheduled to occur at $0.05 + (1 - 0.01)/0.21 = 4.764$, whereas the next change in q_2 is scheduled at $0.05 + 1/80 = 0.0625$. Hence, the next step is performed at $t = 0.0625$.
- At $t = 0.0625$, it results that $q_1(0.0625) = 0$, $q_2(0.0625) = x_2(0.0625) = 20$, $x_1(0.0625) \approx 0.0126$, and the derivatives coincide with those found at time $t = 0$.
- This behavior is cyclicly repeated until a change in q_1 occurs. That change occurs at $t \approx 4.95$, after 158 changes in q_2 oscillating back and forth between 20 and 21.
- The simulation continues in the same way.

Figure 1 shows the evolution of $q_1(t)$ and $q_2(t)$ across 500 units of simulated time.

It can be seen that fast oscillations occur in q_2 that cause a total of 15,995 transitions in this variable, whereas q_1 undergoes only 21 changes. Ultimately, more than 16,000 steps are needed to complete the simulation (comparable to the 25,000 steps performed by the forward Euler method to achieve a stable solution).

Although the results are qualitatively correct, the QSS method is unable to integrate the system of Eq.(4) efficiently.

2.2. Backward QSS Method (BQSS).

The BQSS method [17] attempts to prevent the appearance of fast oscillations observed in QSS. The main idea behind the BQSS method is inspired by the classic implicit methods that evaluate the state derivatives at future instants of time (which requires iteration to solve implicit equations). However, in BQSS no iterations are performed.

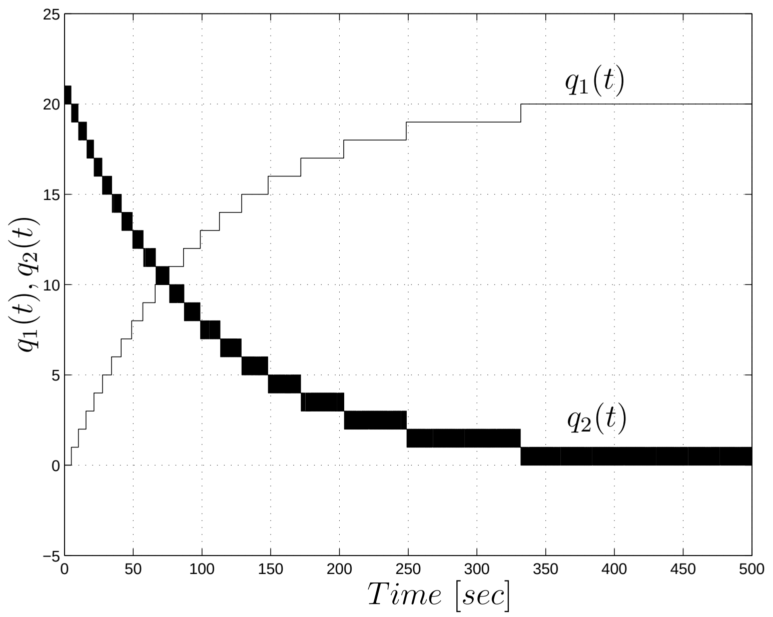

Figure 1: QSS solution of the stiff system of Eq.(4).

Like in QSS1, the quantized states q_j follow piecewise constant trajectories. Also, the state derivatives $\dot{x}_j$ are computed depending on the quantized states, as Eq.(2) shows.

In order to evaluate the state derivatives at future values of the state, each quantized state variable in BQSS contains a future value of the corresponding state variable. This is, q_j is selected so that x_j goes towards it.

Thus, whenever x_j reaches q_j, a new step is performed selecting $q_j(t) = x_j(t) \pm \Delta Q_j$ according to the sign of $\dot{x}_j$.

In other words, if during an event $\dot{x}_j > 0$, the new quantized state value is set to $q_j = x_j + \Delta Q_j$. Otherwise if $\dot{x}_j < 0$, it takes the value $q_j = x_j - \Delta Q_j$. In either case, the subsequent change in q_j is scheduled for the instant when x_j reaches q_j.

It could happen that, at a certain point, taking $q_j = x_j + \Delta Q_j$ provokes that $\dot{x}_j < 0$ and taking $q_j = x_j - \Delta Q_j$ leads to $\dot{x}_j > 0$. Here, the mean value theorem ensures that q_j can adopt a value $\widetilde{q}_j$ near x_j at which the condition $\dot{x}_j = 0$ is met.

In this case, the BQSS method does not actually search the value $q_j = \widetilde{q}_j$, but it keeps x_j constant as if its time derivative $\dot{x}_j$ were zero.

With this idea, the introductory example of Eq.(4) can be simulated in only 43 steps using the same quantum that was previously employed in the simulation using QSS.

A precise specification of the BQSS method, its implementation, theoretical properties, and main features are discussed in [15, 17].

The main limitation of BQSS is that it performs a first–order approximation only. Thus, we cannot obtain accurate results without accepting a significantly increased number of steps.

Higher–order QSS approximations can be obtained from QSS1, as we shall show below, whereas higher–order BQSS methods can unfortunately not be obtained [15].

2.3. Linearly Implicit QSS Methods

In BQSS, we choose q_j so that x_j evolves towards q_j. When this is not possible, we know that there must exist a point $\widetilde{q}_j$ near x_j, for which $\dot{x}_j = 0$. So, we enforce that condition without actually calculating $\widetilde{q}_j$.

We know that if we set $q_j = \widetilde{q}_j$, then $\dot{x}_j = f_j(\mathbf{q}, \mathbf{u}) = 0$. However, as we do not calculate $\widetilde{q}_j$, there is no way of computing higher–order derivatives of x_j. Consequently, we cannot obtain higher–order approximations. This is the reason why higher–order BQSS methods cannot be obtained.

In order to overcome this problem, the LIQSS1 method was defined in [16], where the value of $\widetilde{q}_j$ is approximated in a linearly implicit way.

In order to illustrate how LIQSS1 works, we shall simulate the system of Eq.(4) step by step from the same initial conditions and using the same quantum as before.

- At $t = 0$, we can choose either $q_2 = 19$ or $q_2 = 21$ according to the sign of $\dot{x}_2(t)$. In both cases, $\dot{x}_1(0) > 0$ so the *future* quantized value of x_1 will be $q_1(0) = 1$.

- If we choose $q_2(0) = 21$, it results that $\dot{x}_2(0) = -180 < 0$, and consequently, x_2 does not evolve towards q_2. On the other hand, choosing $q_2(0) = 19$ implies that $\dot{x}_2(0) = 20 > 0$, and once again, x_2 does not evolve towards q_2.

  Hence it is not possible to choose q_2 so that x_2 moves towards q_2.

  However, the fact that the sign of $\dot{x}_2$ differs for $q_2 = 19$ and $q_2 = 21$ implies that there must exist a point, $\widetilde{q}_2$, in between those two values, for which $\dot{x}_2 = 0$.

The LIQSS1 algorithm calculates the value for $\widetilde{q}_2(0)$ by solving a linear equation:

$$\dot{x}_2(0) = -100q_1(0) - 100\widetilde{q}_2(0) + 2020 = 0$$

from which we obtain

$$\widetilde{q}_2(0) = 20.2 - q_1(0) = 19.2$$

- Then, the state derivatives obtained for $t = 0$ are $\dot{x}_1(0) = 0.192$ and $\dot{x}_2(0) = 0$.
- The next change in q_1 is scheduled at $t = 1/0.192 \approx 5.2083$, whereas the next change in q_2 is scheduled at $t = \infty$.
- The next step takes place at $t = 5.2083$, i.e., when x_1 reaches q_1.

The calculations continue in the same way. Figure 2 shows the evolution of $q_1(t)$ and $q_2(t)$ across 500 units of simulated time.

It can be seen that, using this method, no fast oscillations are present. During this simulation, q_1 changes 21 times, and q_2 changes 25 times, resulting in 46 state changes or 46 simulation steps, which constitutes a decent result for a stiff system.

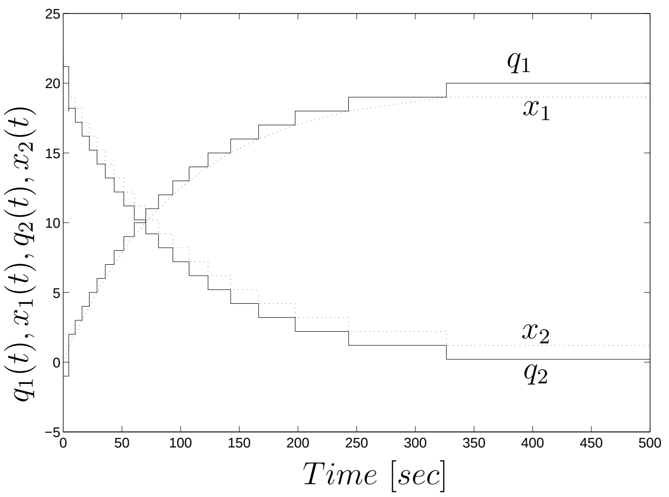

Figure 2: LIQSS solution of the stiff system of Eq.(4). Variables x_i and q_i are the states and quantized states of the numerical solution.

2.4. Higher–Order QSS and LIQSS Methods

Based on QSS1, higher–order methods where developed. The algorithms of QSS2 [9] and QSS3 [11], performing second– and third–order approximations, are outlined below.

2.4.1. Second– and Third–Order Quantized State Methods (QSS2/3).

The QSS2 method is based on the same principles as QSS1, but it replaces the zero–order quantization function of Eq.(3) by a first–order quantization function.

The input–output behavior of a first–order quantization function is shown in Fig.3.

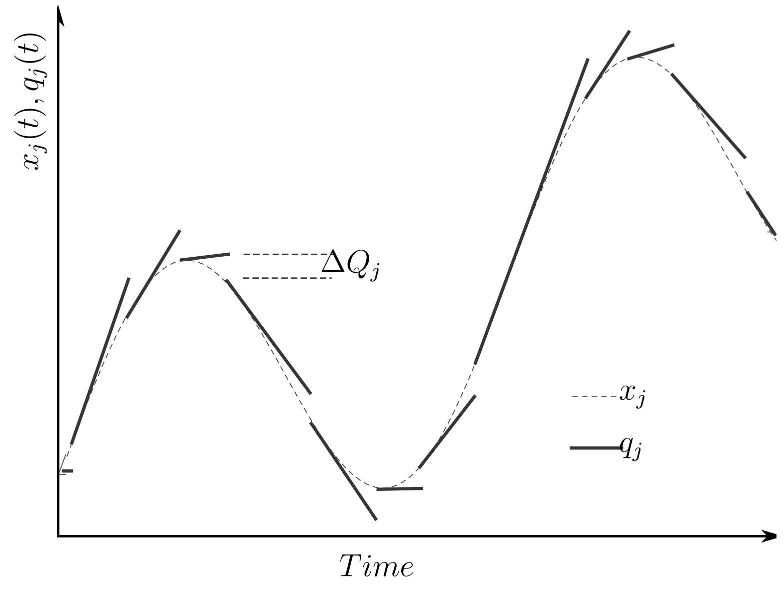

Figure 3: State and quantized variable in a first–order quantization function.

The output trajectory is piecewise linear, and each segment starts with a value and slope equal to those of the input. When the input and output trajectories differ by ΔQ_j, a new output segment begins.

The definition of the QSS2 method is identical to that of QSS1. That is, QSS2 approximates Eq.(1) by Eq.(2), except for using first–order quantization to relate x_j and q_j.

The third–order accurate QSS3 method follows the same idea but uses second–order quantization functions that compute piecewise parabolic quantized state trajectories.

QSS2 and even more so QSS3 are efficient for the simulation of discontinuous non–stiff ODEs. However, neither of these solvers can deal efficiently with stiff systems. Both solvers lead to high–frequency oscillations when confronted with stiff systems, just as QSS1 does.

2.4.2. Second–Order LIQSS Method

The second–order accurate linearly implicit QSS method (LIQSS2) combines the ideas of QSS2 and LIQSS1.

Like in QSS2, the trajectories of the quantized variables are piecewise linear instead of piecewise constant. We aspire that the state and quantized trajectories have the same slope at the moment of change.

In order to avoid oscillations in LIQSS1, we choose q_j so that x_j evolves towards it. This condition is equivalent to

$$(q_j - x_j) \cdot \dot{x}_j \geq 0$$

and it prevents oscillations in the derivative $\dot{x}_j$ around 0.

In LIQSS2, we choose q_j to avoid changes in the sign of the second derivative $\ddot{x}_j$. This translates to the following condition:

$$(q_j - x_j) \cdot \ddot{x}_j \geq 0$$

That is, when the second derivative $\ddot{x}_j$ is positive, we choose a piecewise linear quantized trajectory from above x_j. Otherwise, we choose the trajectory from below.

Similarly to LIQSS1, when the upper trajectory results in a negative value for $\ddot{x}_j$ and the lower trajectory results in a positive value for $\ddot{x}_j$, we know that there exists an intermediate trajectory, for which $\ddot{x}_j = 0$. Thus, we search for that trajectory in a linearly implicit way.

The simulation of the system of Eq.(4) from the same initial conditions ($x_1(0) = 0$ and $x_2(0) = 20$) using a quantum 10 times smaller than before ($\Delta Q_1 = \Delta Q_1 = 0.1$) with LIQSS2 takes a total of 40 steps.

Had we used LIQSS1 or BQSS with a quantum of 0.1, the entire simulation would have undergone at least 200 state transitions in each variable, since the two trajectories evolve from 20 to 0 and from 0 to 20, respectively. The much smaller number of steps encountered in the LIQSS2 simulation was achievable because LIQSS2 is a second–order accurate algorithm.

2.5. QSS Implementation

Although QSS methods can be coded as stand alone solvers, the simplest way of implementing them is by means of the DEVS formalism.

In a DEVS implementation, an n–th order system is split into n *static functions* and n *quantized integrators*.

The j–th static function computes a piecewise constant trajectory $\dot{x}_j(t)$ from the piecewise constant trajectories of the quantized states q_i according to $\dot{x}_j = f_j(\mathbf{q}, t)$. These trajectories are represented by sequences of events, according to the DEVS formalism.

Similarly, the j–th quantized integrator integrates the piecewise constant trajectory $\dot{x}_j$ calculated by the corresponding static function and computes the piecewise constant trajectory of q_j.

The atomic DEVS models for static functions and quantized integrators are quite simple and have been specified in [4, 11]. Coupling these atomic DEVS models, we obtain a new model that can be used to exactly simulate the behavior of the QSS1 approximation of the original ODE.

Higher–order QSS methods can also be implemented in the same way. To this end, the events of quantized integrators and static functions take vector values representing the coefficients of the corresponding piecewise polynomial segments.

The whole family of QSS methods, including the methods introduced in the next section, were implemented in PowerDEVS [2], 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 Modelica [1] and in *Open Source Physics*, a Java–based simulation tool [6], and implementations of the first–order QSS methods can also be found in CD++ [5] and VLE [18].

3. *N*–th Order LIQSS Methods

This section reformulates and extends the methods of LIQSS, originally defined in [16]. We first explain the basic idea of an LIQSS method of order N. Subsequently, we provide a formal definition. Then after discussing the differences between the new and the previously published formulations of LIQSS, we present an implementation algorithm for the methods. Finally, we briefly explain how LIQSS algorithms can be implemented using a DEVS simulation engine.

3.1. Basic Idea

LIQSS1 selects the value of $q_j = x_j \pm \Delta Q_j$ so that x_j approaches q_j. Whenever q_j reaches x_j, a new segment of q_j starts. This implies that $(q_j - x_j) \cdot \dot{x}_j > 0$.

When this condition cannot be met, the mean value theorem ensures that there exists a value $\widetilde{q}_j$ near x_j for which the time derivative of the state $\dot{x}_j(t)$ equals zero.

In this case using a linear approximation, the algorithm estimates the value $\widetilde{q}_j(t)$ for which $\dot{x}_j(t) = 0$.

In a N–th order method, x_j and q_j follow piecewise polynomial trajectories of orders N and $N-1$, respectively. The idea that x_j goes towards q_j (assuming that both trajectories start out with parallel sections) leads to the condition:

$$(q_j(t) - x_j(t)) \cdot \frac{\mathrm{d}^N x_j(t)}{\mathrm{d}t^N} = (q_j(t) - x_j(t)) \cdot x_j^{(N)}(t) > 0$$

Figure 4 illustrates this idea for the third–order case.

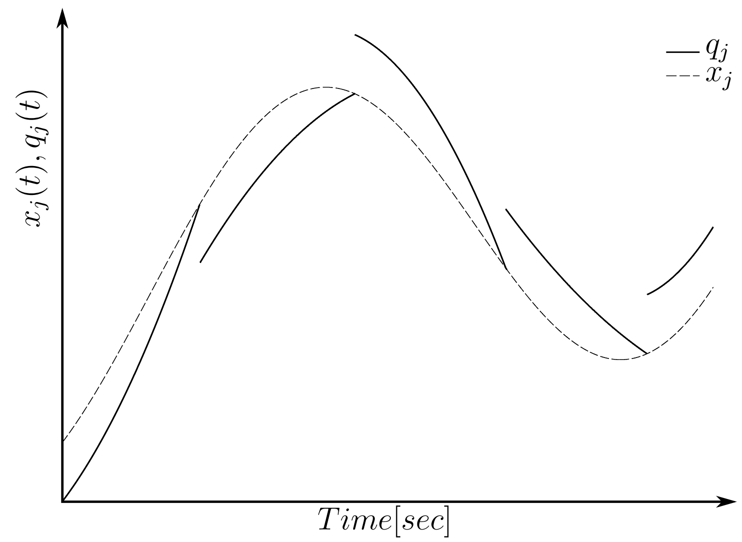

Figure 4: State and quantized trajectories in LIQSS3 method.

Like in LIQSS1 when the condition $(q_j(t) - x_j(t)) \cdot x_j^{(N)}(t) > 0$ cannot be met, the algorithm estimates the value $\widetilde{q}_j(t)$ for which $x_j^{(N)}(t) = 0$.

Actually in LIQSS1, the condition $(q_j - x_j) \cdot \dot{x}_j > 0$ is not verified using the actual value of $\dot{x}_j$, as this would imply performing an extra function evaluation. Instead, we make use of a linear approximation of the j–th component of the ODE:

$$\dot{\hat{x}}_j(t) = A_{jj}\hat{x}_j(t) + v_j(t)$$

where $\hat{x}_j(t)$ is the approximate state, A_{jj} is the j–th diagonal entry of the Jacobian matrix and $v_j(t)$ is a term that approximates the expression $f_j(\hat{\mathbf{x}}, t) - A_{jj}\hat{x}_j(t)$.

This is the same linear approximation that we use to compute $\widetilde{q}_j(t)$.

Similarly in LIQSS_N, we estimate the N–th derivative of the state using successive differentiations of the previous linear approximation, and we compute $\widetilde{q}_j$ from that expression.

3.2. Formal Definition

Given the ODE of Eq.(1), the N–th order accurate LIQSS_N method approximates it by the quantized state system

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

where each component q_j of $\mathbf{q}$ is represented by the following piecewise polynomial function

$$\begin{aligned} q_j(t) &= q_j^{[0]}(k) + q_j^{[1]}(k)(t - t_j^k) + \cdots + q_j^{[N-1]}(k)(t - t_j^k)^{N-1} \\ &= \sum_{i=0}^{N-1} q_j^{[i]}(k)(t - t_j^k)^i \end{aligned} \tag{7}$$

for $t_j^k \le t \le t_j^{k+1}$.

Here $q_j^{[i]}(k)$ is the i–th coefficient of the polynomial at the k–th instant of change t_j^k.

Let $x_j^{(i)}(t)^-$ be the i–th left time derivative[1] of $x_j(t)$ and let $\hat{x}_j^{(N)}(\hat{q}_j, t_j^k)$ be an estimate of the N–th time derivative of x_j at time t_j^k subject to $q_j(t) =$

[1]The i-th time derivative of $x_j(t)$ is obtained by differentiation of the j–th component of Eq.(6) exploiting the fact that the components of $\mathbf{q}(t)$ are sections of polynomials.

$\hat{q}_j(t)$. Then the coefficients $q_j^{[i]}(k)$ can be written as:

$$q_j^{[0]}(k) \triangleq \begin{cases} x_j(t_j^k) + \Delta Q_j & \text{if } \hat{x}_j^{(N)}(\overline{q}_j, t_j^k) > 0 \wedge \hat{x}_j^{(N)}(\underline{q}_j, t_j^k) > 0 \\ x_j(t_j^k) - \Delta Q_j & \text{if } \hat{x}_j^{(N)}(\overline{q}_j, t_j^k) < 0 \wedge \hat{x}_j^{(N)}(\underline{q}_j, t_j^k) < 0 \\ \widetilde{q}_j^{[0]}(k) & \text{otherwise} \end{cases} \tag{8}$$

$$q_j^{[i]}(k)\,(i = 1...N-1) \triangleq \begin{cases} \dfrac{x_j^{(i)}(t_j^k)^-}{i!} & \text{if } q_j^{[0]}(k) \neq \widetilde{q}_j^{[0]}(k) \\ \widetilde{q}_j^{[i]}(k) & \text{otherwise} \end{cases} \tag{9}$$

where $\overline{q}_j(t_j^k) = x_j(t_j^k) + \Delta Q_j$ and $\underline{q}_j(t_j^k) = x_j(t_j^k) - \Delta Q_j$.

The estimate $\hat{x}_j^{(N)}$ and the values $\widetilde{q}_j^{[i]}(k)$ are obtained from a linear approximation of the ODE of Eq. (6). The j–th component of this approximation for $t_j^k \le t < t_j^{k+1}$ can be written as

$$\begin{aligned} \dot{x}_j(t) &= A_{jj}(k) q_j(t) + v_j(t) \\ &= \sum_{i=0}^{N-1} A_{jj}(k) q_j^{[i]}(k)(t - t_j^k)^i + \sum_{i=0}^{N-1} v_j^{[i]}(k)(t - t_j^k)^i \end{aligned} \tag{10}$$

where $A_{jj}(k)$ is the j–th diagonal entry of the Jacobian matrix $A = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}$ at $t = t_j^k$ that can be estimated as

$$A_{jj} = \frac{x_j^{(1)}(t_j^k)^+ - x_j^{(1)}(t_j^k)^-}{q_j^{[0]}(k) - q_j((t_j^k)^-)} \tag{11}$$

and the coefficients of the input term $v_j(t)$ can be obtained from successive differentiations of Eq.(10) as:

$$v_j^{[i]}(k) = \frac{x_j^{(i+1)}(t_j^k)^-}{i!} - A_{jj}(k) \sum_{m=i}^{N-1} \binom{m}{i} q_j^{[m]}(k-1)(t_j^k - t_j^{k-1})^{(m-i)} \tag{12}$$

Then, the estimate of the N–th time derivative of x_j when $q_j = \hat{q}_j$ can be computed by successive differentiations of the linear model $\dot{x}_j(t) = A_{jj}(k) q_j(t) + v_j(t)$ and taking $\dot{q}_j = \dot{x}_j$. In this fashion, we find

$$\hat{x}_j^{(N)}(\hat{q}_j(t), t) = A_{jj}^N(k)\hat{q}_j(t) + \sum_{i=1}^{N} A_{jj}^{i-1}(k) v_j^{(N-i)}(t) \tag{13}$$

Letting $t = t_j^k$, the previous expression becomes

$$\hat{x}_j^{(N)}(\hat{q}_j, t_j^k) = A_{jj}^N(k)\hat{q}_j + \sum_{i=1}^{N} A_{jj}^{i-1}(k) v_j^{[N-i]}(k)(N-i)! \tag{14}$$

Integrating Eq.(10), we find

$$\begin{aligned} \hat{x}_j(t) &= x_j(t_j^k) + \sum_{i=0}^{N-1} \underbrace{\frac{A_{jj}(k) q_j^{[i]}(k) + v_j^{[i]}(k)}{i+1}}_{\hat{x}_j^{[i+1]}(k)} (t - t_j^k)^{i+1} \\ &= x_j(t_j^k) + \sum_{i=0}^{N-1} \hat{x}_j^{[i+1]}(k)(t - t_j^k)^{i+1} \qquad t_j^k < t < t_j^{k+1} \end{aligned} \tag{15}$$

and we can compute the coefficients $\widetilde{q}_j^{[i]}(k)$ so that $\hat{x}_j^{[N]}(k) = 0$ and $\hat{x}_j^{[i]}(k) = q_j^{[i]}(k)$. Then from Eq.(15), it results that

$$\widetilde{q}_j^{[i]}(k) = \begin{cases} \dfrac{-\, v_j^{[N-1]}(k)}{A_{jj}(k)} & \text{for } i = N-1 \\ \dfrac{(i+1)\widetilde{q}_j^{[i+1]}(k) - v_j^{[i]}(k)}{A_{jj}(k)} & \text{for } i = 0 \cdots N-2 \end{cases} \tag{16}$$

Finally, the sequence of values t_j^k $(k = 0, 1, ...)$ is defined so that t_j^{k+1} is the minimum $t > t_j^k$ where

$$|x_j(t) - \check{q}_j(t)| = \Delta Q_j \tag{17a}$$

or

$$\hat{x}_j^{(N)}(q_j(t), t) = 0 \tag{17b}$$

with

$$\check{q}_j(t) \triangleq x_j(t_j^k) + \sum_{i=1}^{N-1} q_j^{[i]}(k)(t - t_j^k)^i \tag{18}$$

3.3. Differences with previous definitions of LIQSS1 and LIQSS2

In the definition of LIQSS1 given in [16], the condition $(q_j(t) - x_j(t)) \cdot \dot{x}_j(t) > 0$ was verified using the actual value of $\dot{x}_j$, which implied an extra function evaluation per step. Similarly in LIQSS2, the verification of condition $(q_j(t) - x_j(t)) \cdot \ddot{x}_j(t) > 0$ implied the usage of two extra function evaluations per step.

In the new definition, the previous conditions are replaced by $(q_j(t) - x_j(t)) \cdot \hat{x}_j^{(N)}(t) > 0$, where the estimate $\hat{x}^{(N)}$ does not require any additional function evaluation.

Another change is introduced in LIQSS2 by Eq.(17b). This condition, which was not included in the previous definition of LIQSS2, states that a new step is performed whenever the estimate $\hat{x}_j^{(N)}(t)$ changes its sign. Since q_j is not constant for methods of orders higher than 1, it could happen that $q_j(t)$ reaches the value $\widetilde{q}_j$ (i.e., the value at which $\hat{x}_j^{(N)}(t) = 0$) before meeting the condition $q_j(t) = x_j(t)$. When the step condition $q_j(t) = x_j(t)$ is finally met, $\widetilde{q}_j$ could be far away from x_j (a distance greater than ΔQ_j), and we could not use that value for q_j, as this would introduce a large error. In the previous definition of LIQSS2, this case led to fast oscillations in some cases.

3.4. LIQSS_N Simulation Algorithm

The definition of LIQSS_N can be implemented by the following simulation algorithm:

a. When at $t = t_j^k$ Eq.(17) is verified (i.e., $|x_j(t) - \check{q}_j(t)| = \Delta Q_j$ or $\hat{x}_j^{(N)}(q_j(t), t) = 0$) then

1. Update the state x_j and its derivatives $x_j^{(i)}$ to the current time t using its polynomial representation

$$x_j(t_j^k) = x_j(t_j^{k-1}) + \sum_{i=1}^{N} \frac{x_j^{(i)}(t_j^{k-1}) \cdot (t_j^k - t_j^{k-1})^i}{i!}$$
$$x_j^{(1)}(t_j^k) = x_j^{(1)}(t_j^{k-1}) + \sum_{i=1}^{N-1} \frac{x_j^{(i+1)}(t_j^{k-1}) \cdot (t_j^k - t_j^{k-1})^i}{i!}$$
$$\vdots$$

   Update also the quantized state polynomial coefficients to the current time t with an analogous formula.

2. Store the actual values of the quantized state $q_j(t^-)$, the state derivative $x_j^{(1)}(t^-)$, and set $\check{q}_j(t_j^k) = x_j(t)$.

3. If $x_j^{(N)}(t_j^k) > 0$ then take $\hat{q}_j = x_j(t) + \Delta Q_j$ else take $\hat{q}_j = x_j(t) - \Delta Q_j$

4. Estimate $\hat{x}_j^{(N)}(\hat{q}_j, t)$ from Eq.(14).

5. If $\hat{x}_j^{(N)}(\hat{q}_j, t) \cdot x_j^{(N)}(t_j^k) > 0$ or $A_{jj}(k) = 0$, then from Eqs.(8)–(9)

- Take $q_j^{[0]}(k) = \hat{q}_j$.
- Calculate $q_j^{[i]}(k) = \dfrac{x_j^{(i)}(t_j^k)}{i!}$.

6. Otherwise according to Eq.(16)
   - Compute $q_j^{[N-1]}(k) = \dfrac{-\, v_j^{[N-1]}(k)}{A_{jj}(k)}$
   - Calculate $q_j^{[i]}(k) = \dfrac{(i+1)q_j^{[i+1]}(k) - v_j^{[i]}(k)}{A_{jj}(k)}$ starting from $i = N-2$ down to $i = 0$.
7. Compute the instant $t = t_j^{k+1}$ when Eq.(17) is satisfied (i.e., $|x_j(t) - \check{q}_j(t)| = \Delta Q_j$ or $\hat{x}_j^{(N)}(q_j(t), t) = 0$).
8. For each i so that $f_i(\mathbf{x}, t)$ explicitly depends on x_j
   - Evaluate $x_i^{(1)}(t) = f_i(\mathbf{q}, t)$ and the higher order derivatives $x_i^{(m)}(t)$ with $m = 2, \cdots, N$.
   - if $i = j$ then recompute $A_{jj}(k)$ from Eq.(11) as
     $$A_{jj} = \frac{x_j^{(1)}(t^-) - x_j^{(1)}(t)}{q_j(t^-) - q_j^{[0]}(k)}$$
   - Estimate the input coefficients $v_i^{[m]}(k)$ using Eq.(12)
   - Recompute the next instant of change $t = t_i^{k+1}$ when Eq.(17) is satisfied (i.e., $|x_i(t) - \check{q}_i(t)| = \Delta Q_i$ or $\hat{x}_i^{(N)}(q_i(t), t) = 0$).

b. Advance the simulation time t to the smallest future value of time when any state variable undergoes a transition and go back to the beginning. The usage of this algorithm in a simple example is illustrated in Appendix A.

Remark 1. *The definition of LIQSS_N and the corresponding algorithm given above are valid for an arbitrary order N. However, LIQSS methods of orders greater than 4 are impractical. The reason is that the computation of the next instant of change at Eq.*(17) *requires finding the roots of a N–th order polynomial. Thus, an LIQSS_N method of order 5 or greater requires iterations.*

Beside from the aforementioned difficulty, LIQSS_N methods are primarily intended for the simulation of systems with frequent discontinuities. In such systems, methods of very high order do not offer advantages, as the integration step sizes are limited by the occurrence of discontinuities. Thus, algorithms of order greater than 4 are rarely used in these types of systems.

3.5. DEVS Implementation of LIQSS Algorithms

The LIQSS methods of orders 1 to 3 were implemented in PowerDEVS [2]. The static functions are the same as those of the QSS methods of orders 1 to 3 as they compute functions of piecewise polynomial trajectories.

The LIQSS_N quantized integrators are similar to those of QSS_N. The main difference is that they compute the coefficients of $q_j(t)$ not only depending on the coefficients of $x_j(t)$ but also on the estimate of the N–th derivative $\hat{x}_j^{(N)}(t)$ as described in points 1 to 5 of the algorithm above. They also estimate the values of A_{jj} and the coefficients of $v_j(t)$ as described in point 7 of the algorithm.

The LIQSS_N quantized integrators can be used with parameter values identical to those of the QSS_N and BQSS integrators allowing the usage of logarithmic quantization. With logarithmic quantization [12] the quantum size is proportional to the state magnitude, and its use implies intrinsic control of the relative error.

A logarithmic quantization function is specified defining a *relative quantum*, ΔQ_{rel}, and a *minimum quantum*, ΔQ_{min}. Then, the quantum changes with the quantized state, q_j, according to the equation:

$$\Delta Q_j = \max(\Delta Q_{rel} \cdot |q_j|, \Delta Q_{min})$$

4. Theoretical Properties of the LIQSS Methods

In this section, we study the order, stability and accuracy properties of LIQSS methods.

4.1. Perturbed Representation

Since LIQSS methods have the same quantized system representation of Eq.(2) as QSS methods, they also share their perturbed representation:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t) + \boldsymbol{\Delta}\mathbf{x}(t), \mathbf{u}(t)) \tag{19}$$

where the perturbation term is defined as:

$$\boldsymbol{\Delta}\mathbf{x}(t) \triangleq \mathbf{q}(t) - \mathbf{x}(t)$$

Notice that Eq.(17a) establishes that when the condition $|\check{q}_j(t) - x_j(t)| = \Delta Q_j$ is met, a new step is performed. Since each section of $\check{q}_j(t)$, defined in Eq.(18), starts at $x_j(t)$, it follows that $\check{q}_j(t)$ and $x_j(t)$ never differ from each other by more than ΔQ_j.

From Eq.(18), the polynomial of $\check{q}_j(t)$ only differs from the polynomial of $q(t)$ in its first coefficient, so $q(t)$ and $\check{q}(t)$ run parallel to each other.

At each step, the value of $q_j(t)$ is taken as $q_j(t) = x_j(t) \pm \Delta Q_j$ or $q_j(t) = \widetilde{q}_j$ where $|\widetilde{q}_j - x_j(t)| < \Delta Q_j$. Thus, q_j does not differ from x_j by more than ΔQ_j at the time of the transition, and consequently, $q_j(t)$ never differs from $\check{q}_j(t)$ by more than ΔQ_j.

It follows that

$$\begin{aligned} |q_j(t) - x_j(t)| &= |q_j(t) - \check{q}_j(t) + \check{q}_j(t) - x_j(t)| \\ &\le |q_j(t) - \check{q}_j(t)| + |\check{q}_j(t) - x_j(t)| \le 2\Delta Q_j \end{aligned} \tag{20}$$

Thus, LIQSS_N methods simulate an approximate system that only differs from the original system of Eq.(1) due to the presence of the bounded state perturbation.

4.2. Stability and Global Error Bound

Based on the perturbed representation of Eq.(19) and the fact that

$$|q_j(t) - x_j(t)| \le \Delta Q_j \quad \forall t \ge 0 \tag{21}$$

it was proven that

- Assuming that $\mathbf{f}$ is locally Lipschitz, the numerical solution obtained by the QSS1 solver converges to the analytical solution [13].
- Provided that the original ODE has an asymptotically stable equilibrium point, the QSS1 solution is ultimately bounded around that equilibrium point [13].

As LIQSS1 only replaces Eq.(21) by (20) it is straightforward to prove that both properties are also satisfied by the first order LIQSS1 method.

Theoretical properties for higher–order QSS methods were only studied for Linear Time Invariant (LTI) systems, and they can be extended to LIQSS methods as follows.

Given the LTI system

$$\dot{\mathbf{x}}_a(t) = A\mathbf{x}_a(t) + B\mathbf{u}(t) \tag{22}$$

where A is a Hurwitz matrix with Jordan canonical form $\Lambda = V^{-1}AV$. Any LIQSS approximation simulates the system:

$$\dot{\mathbf{x}} = A(\mathbf{x}(t) + \Delta\mathbf{x}(t)) + B\mathbf{u}(t) \tag{23}$$

Defining the error as $\mathbf{e}(t) \triangleq \mathbf{x}(t) - \mathbf{x}_a(t)$, it follows that

$$\dot{\mathbf{e}} = A(\mathbf{e}(t) + \Delta\mathbf{x}(t)) \tag{24}$$

where, according to Eq. (20), $|\Delta x_j| \leq 2\Delta Q_j$ for $j = 1, \cdots, n$.

In QSS1, QSS2 and QSS3 methods, starting from Eq.(24) and knowing that $|\Delta x_j| \leq \Delta Q_j$, it was proven in [9, 11] that

$$|\mathbf{e}(t)| \preceq |V||\mathbb{R}\mathrm{e}(\Lambda)^{-1}\Lambda||V^{-1}|\mathbf{\Delta Q}$$

where $\mathbf{\Delta Q}$ is the vector of quanta, the symbol '$|\cdot|$' computes the elementwise absolute value of a matrix or vector, and '$\preceq$' represents a componentwise inequality.

Following an identical reasoning, it is straightforward to prove that in LIQSS methods

$$|\mathbf{e}(t)| \preceq |V||\mathbb{R}\mathrm{e}(\Lambda)^{-1}\Lambda||V^{-1}|2\mathbf{\Delta Q} \tag{25}$$

which is twice the error bound of the QSS methods.

For the case of logarithmic quantization, the conclusions are identical to those of the QSS methods given in [12], again except for a factor of 2.

4.3. Order of the Approximation

According to Eq.(7) in the definition of LIQSS_N, each component $q_j(t)$ of the quantized state $\mathbf{q}(t)$ follows a piecewise polynomial trajectory of order $N-1$.

Equation (9) defines that the i–th coefficient (for $1 \leq i \leq N-1$) of each section of polynomial $q_j^{[i]}$ is equal to either $\tilde{q}_j^{[i]}$ or the i–th coefficient of the Taylor expansion of the state $x_j(t)$. Considering also that the coefficients $\tilde{q}_j^{[i]}$ are computed so that $\tilde{q}_j$ goes parallel to x_j, it follows that in both cases each section of the quantized state q_j starts parallel to that of the state $x_j(t)$.

After a new polynomial section starts, the next change is scheduled for the time instant, at which the difference between x_j and q_j becomes equal to a given value. Since both trajectories only differ on the initial value and the

N–th coefficient of their polynomial representation (which is 0 for q_j), then the time for the next change is scheduled at

$$x_j^{[N]}(t_k) \cdot (t - t_k)^N = c$$

where $x_j^{[N]}$ is the N–th coefficient of the polynomial representation of $x_j(t)$ and c is a constant depending on the quantum and the initial difference between $x_j(t_k)$ and $q_j(t_k)$. Thus, the time of the next step is computed as

$$t = t_k + \sqrt[N]{\frac{c}{x_j^{[N]}(t_k)}}$$

This expression shows that the time between events changes with the N–th root of the quantum, which is proportional to the error bound of the numerical solution. Then, the number of steps performed by LIQSS_N grows with the N–th root of the accuracy.

In this way, when we use the first–order accurate LIQSS1 method, the number of steps grows linearly with the accuracy. In LIQSS2 it grows with the square root of the accuracy, and so on.

For instance, if we wish to improve a simulation result to get a solution that is 10^6 times more accurate, we can expect 10^6 times as many steps in LIQSS1, 10^3 times as many steps in LIQSS2, but only 10^2 as many steps in LIQSS3. It must, however, be mentioned that these numbers are only approximations as, in presence of frequent discontinuities, a fixed amount of additional steps is performed that does not depend on the accuracy settings.

Thus, regarding the order, LIQSS methods are similar to variable step classic algorithms. The order does not define the accuracy (it is a parameter in both cases), but it affects the number of steps performed.

5. Examples

This section presents two simulation examples where the performance of LIQSS methods is compared against that of classic algorithms.

5.1. Buck Converter - Motor Speed Control

The first example is a DC motor fed by a buck converter (Fig.5), where the output voltage is controlled so that the motor speed ω follows a reference signal Ω_{ref} .

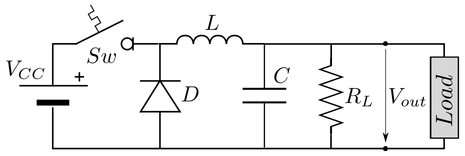

Figure 5: Electric schematic of the buck converter.

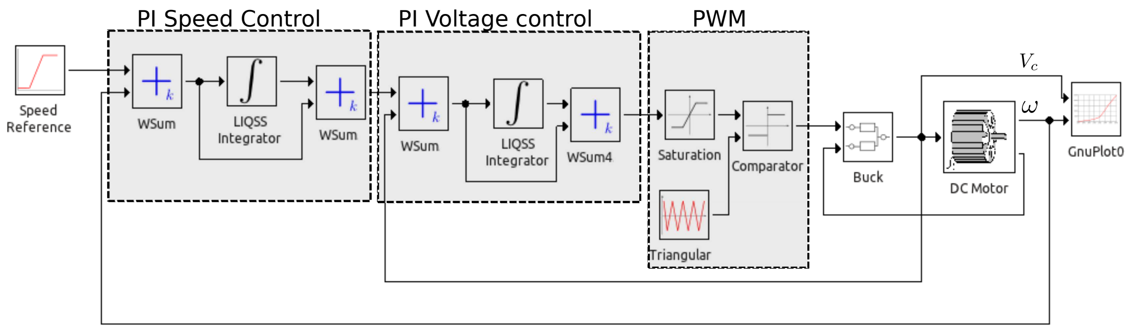

Figure 6: PowerDEVS speed control schematic of a DC Motor with buck converter.

Figure 6 represents the complete schematic of the system as shown by the PowerDEVS graphic user interface.

The system implements a proportional integral (PI) controller for the speed and another PI controller for the buck converter output voltage.

The signal that controls the buck switch state (open/close) is obtained from a pulse width modulator (PWM) that generates a square wave signal. High or low states are obtained by comparing the signal $x(t)$ with a triangular wave of 10kHz. If the value $x(t)$ is greater than the triangular, we set $\xi = 1$, otherwise we set $\xi = -1$. Figure 7 shows the behavior described in the PWM block.

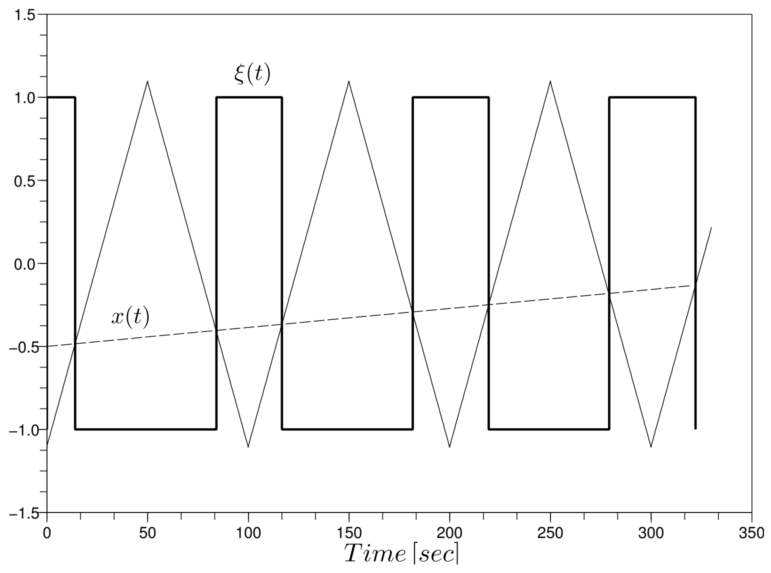

Figure 7: Behavior of PWM block diagram of Fig.6.

The equations that describe the system are:

$$\text{DC Motor equations} \quad \begin{cases} \dot{I}_a &= \frac{1}{L_a}\left(V_C - R_a \cdot I_a - K_m \cdot \omega\right) \\ \dot{\omega} &= \frac{1}{J}\left(I_a \cdot K_m - b \cdot \omega + T_c\right) \end{cases} \tag{26}$$

$$\text{Buck converter} \quad \begin{cases} \dot{V}_C &= \frac{I_L}{C} - \frac{V_C}{RC} - \frac{I_a}{C} \\ \dot{I}_L &= \frac{V_D - V_C}{C} \\ V_D &= R_D\left(\frac{V_{CC}-V_D}{R_{LL}} - I_L\right) \end{cases} \tag{27}$$

$$\text{Speed controller} \quad \begin{cases} e_\omega(t) &= \omega_{ref}(t) - \omega(t) \\ V_{ref}(t) &= k_{p\omega} \cdot e_\omega(t) + k_{i\omega} \int_0^t e_\omega(t)dt \end{cases} \tag{28}$$

$$\text{Converter output voltage controller} \quad \begin{cases} e_{V_{out}}(t) &= V_{ref}(t) - V_{out}(t) \\ x(t) &= k_{pv} \cdot e_{V_{out}}(t) + k_{iv} \int_0^t e_{V_{out}}(t)dt \end{cases} \tag{29}$$

where

$$R_{Sw} = \begin{cases} R_{Sw-on} & \text{if } \xi = 1 \ (Sw \text{ closed}) \\ R_{Sw-off} & \text{if } \xi = -1 \ (Sw \text{ open}) \end{cases} \tag{30}$$

$$R_D = \begin{cases} R_{D-off} & \text{if } V_D > 0 (I_D \cdot R_D > 0) \\ R_{D-on} & \text{if } V_D \le 0 (I_D \cdot R_D \le 0) \end{cases} \tag{31}$$

Equations (30)–(31) exhibit the discontinuous nature of the system. Stiffness is related to the large value of R_{D-off} representing the large resistance of the diode in its *off* state.

The system was simulated using the parameter values shown in Tables 1 and 2.

Parameter	**Value**
R	1Ω
C	10^{-4} F
L	10^{-4} H
R_{D-on} R_{D-off}	10^{-6} 10^{6}
R_{SW-on} R_{SW-off}	10^{-6} 10^{6}
Switching frequency	10 kHz
Vcc	24V

Table 1: Parameter values for the buck converter.

Parameter	**Value**
Armature Resistance (R_a)	1.73 Ω
Armature inductor (L_a)	2.54 mH
Rotor mass moment of inertia (J)	$1.62 \cdot 10^{-5}$ Nm/s^2
Electromotive force constant(K_m)	0.0551
Mechanical system's damping ratio(b)	$1.12 \cdot 10^{-5}$
$k_{p\omega}$	1.5
$k_{i\omega}$	5
k_{pv}	0.5
k_{iv}	0.5

Table 2: Parameter values of the DC motor and controller.

The speed reference signal starts at $t = 0$ with a value $\omega_{ref}(0) = 0$ and evolves with a constant slope of 54.2 until $t = 2$, when it reaches the value $\omega_{ref} = 108.4$. From that moment on, it remains constant.

The model was simulated in PowerDEVS setting a final simulation time of $t_f = 3$sec. All initial conditions are set equal to zero. The simulations were performed using the LIQSS2 and LIQSS3 solvers for different sets of relative and minimum quantum values, ΔQ_{rel} and ΔQ_{min}.

Then under identical conditions, the system was simulated with different solvers implemented in Dymola [3] for different accuracy settings. Here, the best results were obtained using the 'esdirk23a' algorithm[2]. The esdirk23a solver turned out to be more efficient than the default 'dassl' solver when simulating this model.

Figures 8–10 plot the motor speed, the converter output voltage, and the diode current obtained by LIQSS3 in PowerDEVS using quantization values of $\Delta Q_{min} = \Delta Q_{rel} = 10^{-4}$ for all state variables.

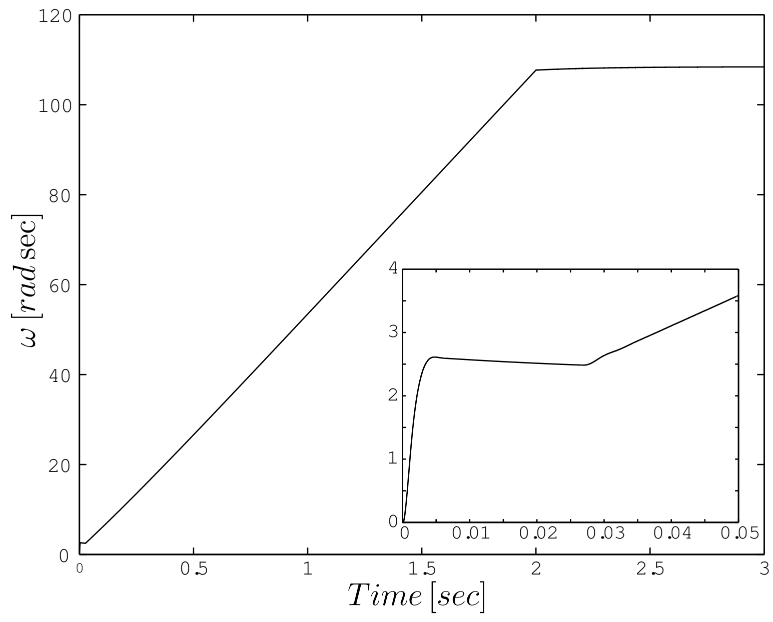

Figure 8: Simulated trajectory of the motor speed.

Table 3 summarizes the simulation performance of the LIQSS and esdirk23a solvers in PowerDEVS and Dymola, respectively.

The error was computed comparing the results with the solution given by DASSL using a very small error tolerance. More precisely, it was computed as

$$Err = \frac{\sum_{k=1}^{M}(\omega(t_k) - \omega_{ref}(t_k))^2}{M} \tag{32}$$

[2]esdirk23 stands for Explicit Singly Diagonally Implicit Runge Kutta 2–3 (i.e., a diagonally implicit RK algorithm with explicit first stage) [14]

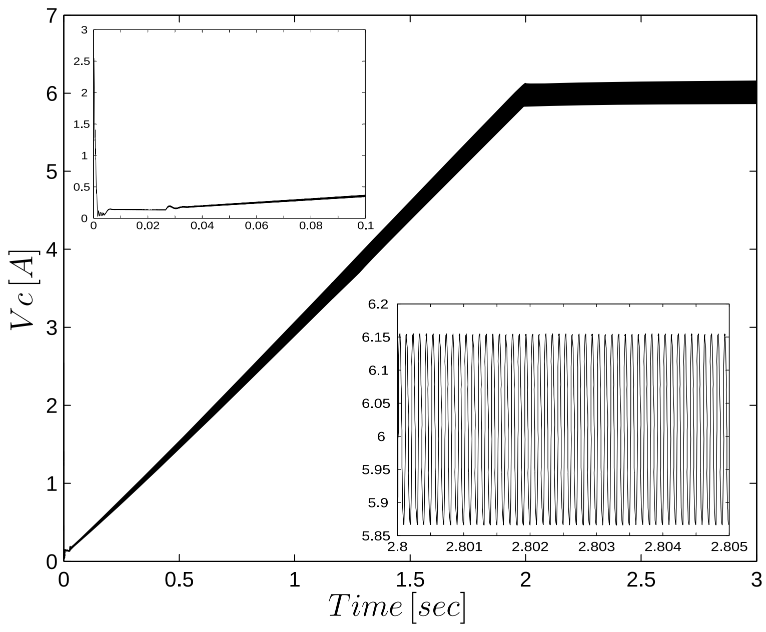

Figure 9: Simulated buck converter output voltage.

where $\omega(t_k)$ are the samples of the numerical solution for the motor speed and $\omega_{ref}(t_k)$ are the samples of the reference solution.

In order to compare the number of function evaluations, we took into account that each LIQSS step does not involve full function evaluations. It only performs evaluations at those components of $\mathbf{f}$ that explicitly depend on the quantized variable that changes or on a discontinuity taking place.

For instance, the simulation using LIQSS3 with $\Delta Q_{min} = \Delta Q_{rel} = 10^{-4}$ consumed a total of 895,660 integration steps, namely 148,793 at V_C, 592,790 at I_L, 7539 at ω, 134,984 at I_a, 11,004 at the voltage controller, and 339 at the speed controller. In addition, 60,035 discontinuities were detected in the switch, and 84,297 occurred in the diode. Consequently, 1,922,289 scalar function evaluations were needed altogether.

Dymola using the esdirk23a solver with a relative error tolerance of $1\cdot10^{-4}$ involved 705,264 accepted and 96,671 rejected integration steps; in addition, 89,972 true and 60,000 temporary state events were detected. The simulation involved a total of 4,923,076 full function evaluations. Taking into account that the system is of order 6, esdirk23a performed 29,538,456 scalar function evaluations.

Besides function evaluations, there are other factors that increase CPU

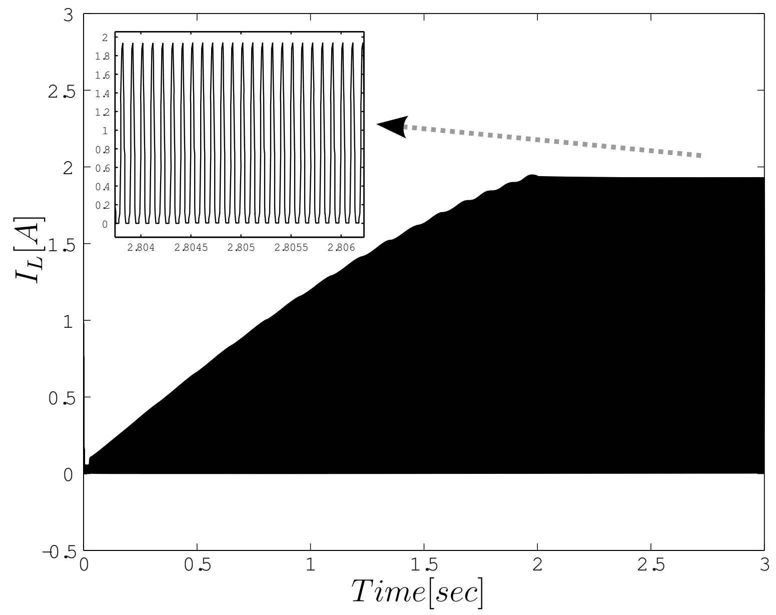

Figure 10: Simulated diode current at the buck converter.

time in both methods. On the one hand, LIQSS3 has to solve a cubic equation at each step, and the DEVS engine of PowerDEVS performs some tasks related to event scheduling and propagation, which can be computationally expensive. In contrast, esdirk23a must calculate a Jacobian matrix, solve a linear system of equations involving that Jacobian matrix, perform Newton iterations, and search for discontinuities.

The Dymola simulations ran more accurately than the LIQSS simulations, and the simulation results turned out to be more accurate than requested. The esdirk23a step size and thereby the simulation accuracy is controlled in this example primarily by the frequently occurring discontinuities and not so much by the requested error tolerance. For a low error tolerance of 10^{-2}, Dymola attempts a too large initial step, and the simulation dies before it ever takes off due to numerical stability problems.

In contrast, the LIQSS simulations of this system are more robust. The LIQSS solver never turns numerically unstable, and a user who requests less accurate results can get those and be rewarded by faster simulation runs. This feature can be essential in simulations that are running under real–time constraints, for example. The fastest simulation obtained by Dymola ran 3.3 times slower than the fastest simulation obtained by PowerDEVS.

5.2. Logical Inverter Chain

A logical inverter performs a logical operation over a signal. When the input signal assumes a high level, the inverter outputs a low level, and vice-versa.

Logical inverters are implemented by electrical circuits. They exhibit a non–ideal response since the rise and fall time of the signal output is limited by physical characteristics of the inverter circuit, and thus, the correct output level is not immediately obtained, i.e., it is delayed.

An inverter chain is a concatenation of several inverters, where the output of each inverter acts as the input to the next one. Making use of the aforementioned limitations, inverter chains can be used to obtain delayed signals.

Integration method	**N° of Steps**	**Scalar function evaluations f_j**	**Error**	**CPU time [sec]**
LIQSS2 ($\Delta Q_{min,rel} = 10^{-2}$)	400,774	819,738	0.8043	8.21
LIQSS2 ($\Delta Q_{min,rel} = 10^{-3}$)	931,348	1,936,255	0.1132	16.12
LIQSS2 ($\Delta Q_{min,rel} = 10^{-4}$)	2,695,093	5,675,314	$7.5 \cdot 10^{-3}$	43.41
LIQSS3 ($\Delta Q_{min,rel} = 10^{-3}$)	539,114	1,162,301	0.072	11.6
LIQSS3 ($\Delta Q_{min,rel} = 10^{-4}$)	895,660	1,922,289	$7.2 \cdot 10^{-3}$	17.9
LIQSS3 ($\Delta Q_{min,rel} = 10^{-5}$)	1,640,678	3,558,047	$8.6 \cdot 10^{-5}$	30.18
Dymola esdirk23a Tolerance:1e-5	1,276,767	53,392,584	$1.8 \cdot 10^{-6}$	62.3
Dymola esdirk23a Tolerance:1e-4	705,264	29,538,456	$1.9 \cdot 10^{-5}$	53.9
Dymola esdirk23a Tolerance:1e-3	350,108	14,098,782	$2.6 \cdot 10^{-5}$	27.21
Dymola esdirk23a Tolerance:1e-2	simulation fails	simulation fails	simulation fails	simulation fails

Table 3: Performance comparison for different integration methods and software when simulating Eqs.(26)–(29).

We consider here a chain of m inverters according to the inverter model given in [19] that is characterized by the following equations:

$$\begin{cases} \dot{\omega}_1(t) & = U_{op} - \omega_1(t) - \Upsilon g\left(u_{in}(t), \omega_1(t)\right) \\ \dot{\omega}_j(t) & = U_{op} - \omega_j(t) - \Upsilon g\left(\omega_{j-1}(t), \omega_j(t)\right) \quad j = 2, 3, .., m \end{cases} \tag{33}$$

where

$$g(u, v) = \left(max(u - U_{thres}, 0)\right)^2 - \left(\max\left(u - v - U_{thres}, 0\right)\right)^2 \tag{34}$$

We used the set of parameter values proposed in [19]: $m = 500$ inverters, $\Upsilon = 100$ (which results in a very stiff system), $U_{thres} = 1$, and $U_{op} = 5$.

The initial conditions are, as in the cited reference, $\omega_j = 6.247 \cdot 10^{-3}$ for odd values of j and $\omega_j = 5$ for even values of j. Finally, the input signal is given by:

$$u_{in}(t) = \begin{cases} t - 5 & \text{if } 5 \leq t \leq 10 \\ 5 & \text{if } 10 < t \leq 15 \\ \frac{5}{2}(17 - t) & \text{if } 15 < t \leq 17 \\ 0 & \text{otherwise} \end{cases} \tag{35}$$

A block diagram of a logic inverter built in PowerDEVS is provided in Fig.11. The inverter chain consist in a vector model that contains m sub–models implementing the individual inverters.

Figure 12 shows some components of the solution computed by the LIQSS2 solver using $\Delta Q_{min} = \Delta Q_{rel} = 0.001$ for all state variables.

Although the final simulation time was set to 500 sec, the simulation ended at $t = 128.47$ sec, because an equilibrium situation was reached at that point. Under the simulation conditions mentioned before, the simulation takes $178,595$ steps and 2.72 sec of CPU time. Each integrator performed between 350 and 370 steps, i.e., changes in their quantized state.

The simulation was repeated with different tolerance settings using LIQSS2 and LIQSS3. It was also simulated using different Matlab and Dymola algorithms and tolerances. In all cases, we used a final simulation time of 130 sec.

The results, including also those reported in [19], are summarized in Table 4.

This example shows the true power of the LIQSS solvers. The LIQSS simulations run a thousand time faster than the Dymola simulations; they run several hundreds of times faster than the Matlab simulations; and they run

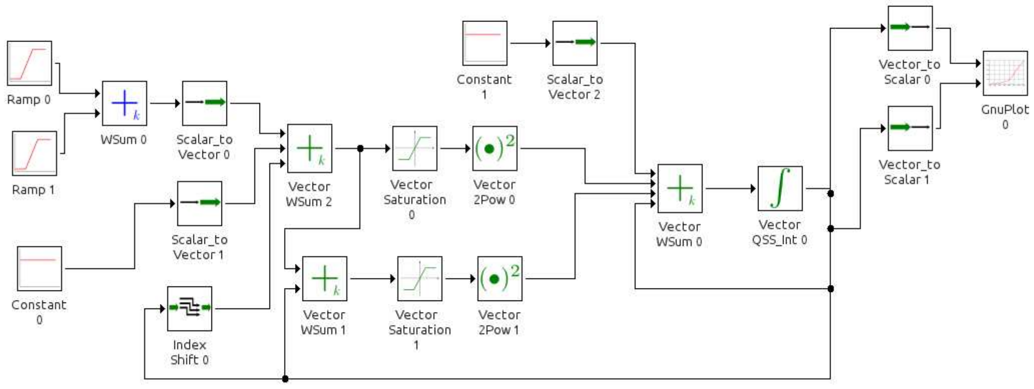

Figure 11: PowerDEVS block diagram of the inverter chain system.

even faster than the multi–rate simulations that were specifically designed for this type of problem.

The dramatic difference in the performance can be explained by the asynchronous nature of discrete event integration. At each step, a discrete time method, like esdirk23a or ode15s, evaluates all components of the vector function $\mathbf{f}$, in this case at least 500 scalar function evaluations. In contrast, LIQSS methods only evaluate those components that explicitly depend on the quantized variable that undergoes a change.

Another reason is that discrete time integration methods for stiff systems are implicit solvers. They need to perform a Newton iteration during each integration step. To this end, they need to solve in each iteration step a linear system of n equations, where n is the order of the system. This fact also adds significantly to the computational load. This is not a huge issue for a 6^{th} order system, but it is a big issue for a 500^{th} order system.

The large error values reported for all simulations of this example are deceiving [3]. They are caused by the large gradients during switching times. A tiny phase shift in the switching time of an inverter leads to a huge error in the signal.

[3]The error was computed as in the previous example, substituting ω by ω_{500} in Eq.(32).

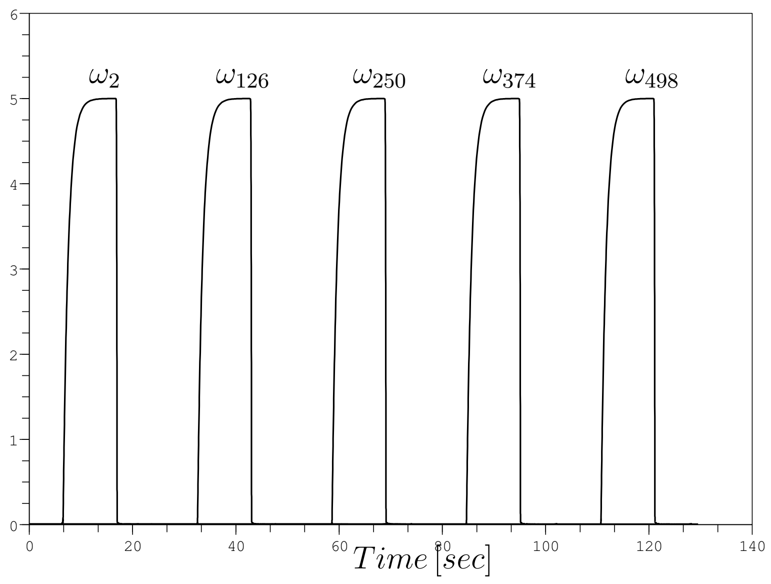

Figure 12: Selected ω_j–components (for j even) of the inverter chain PowerDEVS simulation.

In order to analyze the dependence of the required CPU time on the system order, the inverter chain model was simulated across $t_f = 130$ sec, while varying the number of logical inverters. The results of this experiment are shown in Fig.13.

Notice that the CPU time consumed by both LIQSS2 and LIQSS3 grows linearly with the system order, whereas the CPU time needed by ode15s and esdirk23a grows cubically with the system order. For a small number of inverters, the performance of LIQSS is similar to that of esdirk23a or ode15s, but for a large number of inverters, the LIQSS solvers turn out to be much more efficient.

Multi–rate methods, such as those reported in [19], exhibit a similar performance to that of LIQSS methods. They share the principle of trying to perform larger steps in variables that don't undergo rapid changes. However, those methods discard several function evaluations when the tolerance settings are not met. Consequently, the number of function evaluations is noticeable larger. Moreover to the best of our knowledge, the problem of discontinuity detection has not been addressed in the context of those methods.

Integration method	**Error tolerance**	**N^o of Steps/ Events**	**Scalar f_i eval.**	**Error**	**CPU time [sec]**
PowerDEVS LIQSS2 ($\Delta Q_{min} = \Delta Q_{rel}$)	$\Delta Q_{rel} = 10^{-2}$	178,595 ev.	714,380	0.61	2.72
	$\Delta Q_{rel} = 10^{-3}$	259,591 ev.	1,038,364	0.022	3.81
	$\Delta Q_{rel} = 10^{-4}$	533,200 ev.	2,132,800	0.00023	8.04
PowerDEVS LIQSS3 ($\Delta Q_{min} = \Delta Q_{rel}$)	$\Delta Q_{rel} = 10^{-2}$	81,377 ev.	488,262	1.343	1.51
	$\Delta Q_{rel} = 10^{-3}$	165,931 ev.	995,586	0.443	2.99
	$\Delta Q_{rel} = 10^{-4}$	286,806 ev.	1,720,836	0.119	4.95
	$\Delta Q_{rel} = 10^{-5}$	368,857 ev.	2,213,142	0.0133	6.49
	$\Delta Q_{rel} = 10^{-6}$	626,693 ev.	3,760,158	0.00205	11.1
Dymola esdirk23a	$e_{rel} = 10^{-1}$	7875 steps	$> 37,748,000$	0.062	1786.59
	$e_{rel} = 10^{-2}$	8664 steps	$> 43,937,000$	0.046	2005.64
	$e_{rel} = 10^{-3}$	10,005 steps	$> 51,935,000$	0.037	2147.1
Matlab Ode15s	$e_{rel} = 10^{-2}$	13,220 steps	$> 33,050,000$	0.041	651.37
Multirate II	$e_{rel} = 10^{-4}$	-	4,795,878	-	6.36*
Multirate II	$e_{rel} = 10^{-5}$	-	17,358,472	-	21.65*

Table 4: Performance comparison when simulating the model of Eq.(33) using different methods. The CPU time for multi–rate methods was not obtained on the same computer, but has been added as reported in [19].

6. Conclusions

We presented a new family of QSS solvers based on linearly implicit principles of orders 1 to 3 that are suitable for the simulation of some classes of stiff systems.

We demonstrated that the LIQSS methods preserve most of the theoretical and practical characteristics of the non–stiff QSS solvers. They ensure stability and a global error bound in linear systems. Also, they handle discontinuities in a highly efficient manner.

Compared with classic discrete time methods, LIQSS exhibit important advantages when simulating stiff systems experiencing frequent discontinuities. We illustrated this feature in the simulation of a complex system involving a power electronic device.

Another advantage is related to the simulation of some classes of large stiff sparse systems. Like in the case of the non–stiff QSS solvers, LIQSS steps perform local calculations in those states only that undergo a change.

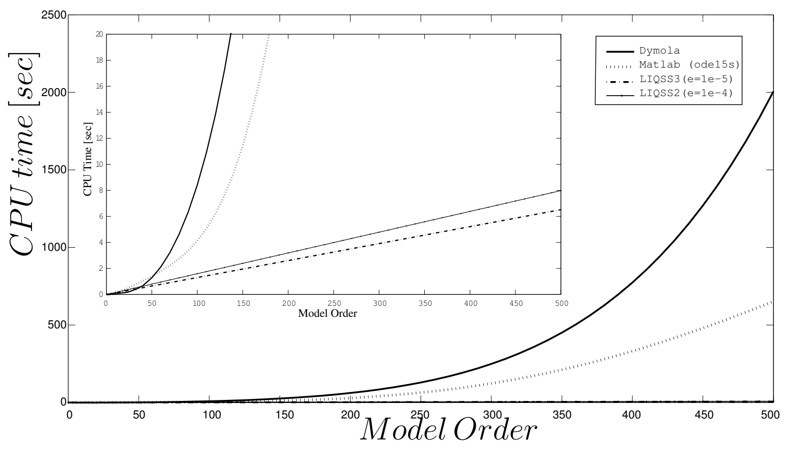

Figure 13: CPU time *vs.* number of logical inverters.

In large systems experiencing activities in a few states only at any given moment of time, LIQSS only computes functions that are directly related to those *active* states. This advantage was demonstrated in the simulation of the logical inverter chain.

It was mentioned that LIQSS algorithms do not perform matrix inversions. Unfortunately, this fact imposes a severe limitation on this class of solvers. LIQSS solvers avoid fast oscillations in x_j by searching for the point $\hat{q}_j$, at which the state derivative $\dot{x}_j$ changes its sign. This search is done by looking at the diagonal elements A_{jj} of the Jacobian matrix only. When stiffness is not due to the diagonal terms of the Jacobian matrix, but caused by some structural feature, LIQSS solvers may be unable to deal with the stiffness inherent in such a model, and oscillations may still occur. The same limitation applies to classical linearly implicit algorithms.

In power electronic circuits, where stiffness is mainly due to the large or small resistance values of the switching devices in their *off* or *on* states, LIQSS algorithms usually work well. However in other systems, such as those resulting from spatial discretization of a diffusion equation by the method of lines, for example, where the stiffness is not reflected in large diagonal elements of the Jacobian matrix [4], the LIQSS methods do not perform any better than their non–stiff QSS brethren.

We have not studied yet what conditions a stiff system must satisfy for LIQSS methods to be successful in their simulation. How can stiff systems that are not solvable efficiently by LIQSS solvers be recognized? Without such an analysis, LIQSS solvers cannot replace the classical dassl solver as default simulation method in a general purpose modeling and simulation environment, such as Dymola. Thus, it is important to research this question. We conjecture that, in power electronics circuits with non–ideal switching devices, such as those used in the Modelica standard library, LIQSS methods will always work. At least, we have not come across any counterexample until now. However, we have not yet been able to prove this conjecture.

7. Acknowledgments

We wish to acknowledge support by CONICET under grant PIP-2009/2011-00183

References

[1] T. Beltrame, F. Cellier, Quantised state system simulation in Dymola/Modelica using the DEVS formalism, in: Proceedings of the Fifth International Modelica Conference, volume 1, Vienna, Austria, pp. 73–82.

[2] 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 87 (2011) 113–132.

[3] D. Brück, H. Elmqvist, S. Mattsson, H. Olsson, Dymola for multi-engineering modeling and simulation, in: Proceedings of the Second International Modelica Conference, pp. 55.1–55.8.

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

[5] M. D'Abreu, G. Wainer, M/CD++: Modeling continuous systems using Modelica and DEVS, in: Proceedings of MASCOTS 2005, Atlanta, GA, pp. 229 – 236.

[6] F. Esquembre, Easy Java Simulations: a software tool to create scientific simulations in Java, Computer Physics Communications 156 (2004) 199–204.

[7] E. Hairer, S. Nørsett, G. Wanner, Solving Ordinary Dfferential Equations I. Nonstiff Problems, Springer, Berlin, 2nd edition, 1993.

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

[9] E. Kofman, A second order approximation for DEVS simulation of continuous systems, Simulation 78 (2002) 76–89.

[10] E. Kofman, Discrete event simulation of hybrid systems, SIAM Journal on Scientific Computing 25 (2004) 1771–1797.

[11] E. Kofman, A third order discrete event simulation method for continuous system simulation, Latin American Applied Research 36 (2006) 101–108.

[12] E. Kofman, Relative error control in quantization based integration, Latin American Applied Research 39 (2009) 231–238.

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

[14] A. Kværnø, Singly diagonally implicit Runge-Kutta methods with an explicit first stage, BIT Numerical Mathematics 44 (2004) 489–502.

[15] 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.

[16] G. Migoni, E. Kofman, Linearly implicit discrete event methods for stiff ODEs, Latin American Applied Research 39 (2009) 245–254.

[17] G. Migoni, E. Kofman, F. Cellier, Quantization-based new integration methods for stiff ODEs., Simulation: Transactions of the Society for Modeling and Simulation International 88 (2012) 387–407.

[18] G. Quesnel, R. Duboz, E. Ramat, M. Traoré, Vle: a multimodeling and simulation environment, in: Proceedings of the 2007 Summer Computer Simulation Conference, San Diego, California, pp. 367–374.

[19] V. Savcenco, R. Mattheij, A multirate time stepping strategy for stiff ordinary differential equations, BIT Numerical Mathematics 47 (2007) 137–155.

[20] L. Shampine, M. Reichelt, The matlab ODE suite., SIAM Journal on Scientific Computing 18 (1997) 1–22.

[21] B. Zeigler, DEVS representation of dynamical systems: Event-based intelligent control, Proceedings of the IEEE 77 (1989) 72–80.

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

[23] B. Zeigler, H. Sarjoughian, H. Praehofer, Theory of quantized systems: Devs simulation of perceiving agents, Cybernetics and Systems 31 (2000) 611–648.

Appendix A. Step–by–step behavior of the LIQSS Algorithm

Here, we illustrate the behavior of the LIQSS_N simulation algorithm introduced in Section 3.4. To this end, we consider the simulation with the LIQSS1 method of the first order system

$$\dot{x}(t) = -x(t) + 1 \triangleq f(x(t)) \tag{A.1}$$

from the initial state $x(0) = 0$ using quantum $\Delta Q = 0.4$.

At the initialization (which was not described in Section 3.4) we set $t^0 = 0$, $A_{11}(1) = 0$, $v^{[1]} = 0$, and evaluate $\dot{x}(t^0)$ at $f(x(t^0) \pm \Delta Q$ in order to decide the initial value for $q(t)$. Since $f > 0$ in both cases, we take $q(t^0) = x(t^0) + \Delta Q = 0.4$.

Then, it results that $x^1(t^0) = f(q(t^0)) = 1 - 0.4 = 0.6$, $x(t) = 0 + 0.6 \cdot t$, for $t^0 < t < t^1$ (with t^1 still unknown), $\check{q}(t^0) = x(t^0) = 0$ and we start the algorithm from this situation.

a. At $t = t^1 = 0.4/0.6 = 2/3$, Eq.(17) is verified (since $|x(t^1) - \check{q}(t^1)| = 0.4 = \Delta Q$). Then

1. We store the actual values of the quantized state $q(t^-) = 0.4$, the state derivative $x^{(1)}(t^-) = 0.6$, and set $\check{q}(t^1) = x(t) = 0.4$.
2. Since $x^{(1)}(t^1) > 0$, we take $\hat{q} = x(t) + \Delta Q = 0.8$.
3. We estimate $\hat{x}^{(1)}(\hat{q}, t)$ from Eq.(14), obtaining $\hat{x}^{(1)}(\hat{q}, t) = 0$ (at the first step we do not have an estimate of the linear model).

4. Since $A_{11}(1) = 0$, then from Eqs.(8)–(9)
    - Take $q^{[0]}(1) = \hat{q}_j = 0.8$.
5. Compute the instant $t = t^2$ when $|x(t) - \check{q}(t)| = \Delta Q$. Here we obtain $t^2 = t^1 + 0.4/0.6 = 4/3$.
6. As $f(x)$ depends explicitly on x (we have only one state), we continue as follows:
    - Evaluate $x^{(1)}(t) = f(q,t) = -0.8 + 1 = 0.2$.
    - Recompute $A_{11}(2)$ from Eq.(11) as
    $$A_{11} = \frac{x^{(1)}(t^-) - x^{(1)}(t)}{q(t^-) - q^{[0]}(k=1)} = \frac{0.6 - 0.2}{0.4 - 0.8} = -1.$$
    - Estimate the input coefficient using Eq.(12) as $v^{[0]}(1) = 0.6 + 0.4 = 1$.
    - Recompute the next instant of change $t = t^2$ when Eq.(17) is satisfied (i.e., $|x(t) - \check{q}(t)| = \Delta Q$, which with the new state derivative results at $t^2 = t^1 + 0.4/0.2 = 8/3$.

b. Advance the simulation time t to the smallest future value of time when any state variable undergoes a transition, so we set $t = t^2 = 8/3$, and go back to the beginning. The second step is then performed as follows
a. At $t = t^2 = 8/3$ Eq.(17) is verified (since $|x(t^1) - \check{q}(t^1)| = 0.4 = \Delta Q$). Then

1. Store the actual values of the quantized state $q(t^-) = 0.8$, the state derivative $x^{(1)}(t^-) = 0.2$, and set $\check{q}(t^k) = x(t) = 0.8$.
2. Since $x^{(1)}(t^2) > 0$, we take $\hat{q} = x(t) + \Delta Q = 1.2$.
3. We estimate $\hat{x}^{(1)}(\hat{q},t)$ from Eq.(14), obtaining $\hat{x}^{(1)}(\hat{q},t) = -0.2$.
4. The condition $\hat{x}^{(1)}(\hat{q},t) \cdot x^{(1)}(t^2) > 0$ or $A_{11}(2) = 0$ is not met, so we proceed from the next step.
5. According to Eq.(16)
    - Compute $q^{[0]}(2) = \dfrac{-\,v^{[0]}}{A_{11}} = \dfrac{-\,1}{-1} = 1$.
6. Compute the instant $t = t^3$ when $|x(t) - \check{q}(t)| = \Delta Q$. Here we obtain $t^2 = t^2 + 0.4/0.2 = 14/3$.
7. As $f(x)$ depends explicitly on x (we have only one state), we continue as follows:
    - Evaluate $x^{(1)}(t) = f(q,t) = -1 + 1 = 0$.
    - Recompute $A_{11}(3)$ from Eq.(11) as
    $$A_{11} = \frac{x^{(1)}(t^-) - x^{(1)}(t)}{q(t^-) - q^{[0]}(k=2)} = \frac{0.2 - 0}{0.8 - 1} = -1.$$
    - Estimate the input coefficient using Eq.(12) as $v^{[0]}(1) = 0.2 + 0.8 = 1$.

- Recompute the next instant of change $t = t^3$ when Eq.(17) is satisfied (i.e., $|x(t) - \check{q}(t)| = \Delta Q$. As the new state derivative is zero, it results that $t^3 = \infty$.

b. Advance the simulation time t to the smallest future value of time when any state variable undergoes a transition, so we set $t = \infty$, which finishes the simulation at the equilibrium point.