# Mixed–Mode State–Time Discretization in ODE Numerical Integration.

Franco Di Pietro$^{a,b*}$, Joaquín Fernández$^{a*}$, Gustavo Migoni$^{a,b*}$, Ernesto Kofman$^{a,b*}$

$^{a}$ *CIFASIS-CONICET, Argentina*
$^{b}$ *FCEIA-UNR, Argentina*

**Abstract**

This article introduces the joint usage of Quantized State System (QSS) methods and classic numerical integration algorithms in the simulation of continuous time systems described by systems of Ordinary Differential Equations (ODEs). The proposed mixed–mode scheme consists of splitting an ODE, using QSS algorithms where they perform better than classic algorithms (i.e., in presence of frequent discontinuities or stiffness under certain particular sparse structures) and using classic algorithms where they are a better choice.

Besides describing the methodology –where the key issue is the interface between both algorithms– the article studies some properties of the resulting scheme including convergence and numerical stability.

In addition, the performance of the proposed mixed–mode algorithm is analyzed in the simulation of two large systems with heterogeneous dynamics, showing an important reduction of the simulation times –more than one order of magnitude– compared to the most efficient QSS and classic approaches.

## 1. Introduction

The presence of stiffness and discontinuities are two major issues in the numerical integration of ODEs. In the first case, the simultaneous presence of slow and fast dynamics enforces the usage of implicit numerical solvers as explicit algorithms become numerically unstable except for very small values of the step size [1, 2]. Implicit numerical solvers implement iterative algorithms that can be computationally expensive, in particular when systems are large. Regarding the second issue, the numerical approximations are not valid in presence of discontinuities on the right hand side of an ODE, and, in consequence, the solvers must be equipped with zero crossing detection and event handling routines that increase their computational costs [2].

Motivated by these problems, a new family of numerical algorithms for ODEs was developed in recent years. These algorithms, called Quantized State Systems (QSS) methods [3, 2], replace the time discretiza-

*Corresponding authors
*Email addresses:* `dipietro@cifasis-conicet.gov.ar` (Franco Di Pietro$^{a,b}$), `kofman@cifasis-conicet.gov.ar` (Ernesto Kofman$^{a,b}$)

tion of classic ODE solvers by the quantization of the state variables obtaining an asynchronous *discrete event approximation* that verifies strong stability and error bound properties. Due to its intrinsic discrete event nature, QSS methods are very efficient in the presence of discontinuities [4] since their occurrence is straightforwardly treated. In addition, there are backward QSS (BQSS) and linearly implicit QSS (LIQSS) methods [5, 6] that can efficiently integrate some particular classes of stiff systems. A noticeably property of these algorithms is that, in spite of their backward formulation, BQSS and LIQSS algorithms are explicit in practice.

The condition for LIQSS algorithms to efficiently integrate a stiff system is that the stiffness is due to the presence of large entries on at only one side of the main diagonal of the Jacobian matrix. Although this restriction was recently relaxed [7] and there are several practical stiff problems in which LIQSS are efficient, in most cases these algorithms are clearly outperformed by classic methods. One example of this occurs in the simulation of parabolic equations (the heat equation, for instance) discretized with the Method of Lines (MOL), where a stiff system of ODEs is obtained but there are no large entries in the Jacobian matrix.

In general, QSS algorithms are more efficient than classic methods in presence of frequent discontinuities and in the simulation of large systems that exhibit inhomogeneous activity, i.e., when only few variables have significant changes during a given interval of time. This is explained by the fact that QSS methods only perform calculations in the state variables that experience significant changes. On the contrary, in systems with homogeneous activity and without frequent discontinuities classic ODE solvers are usually more efficient [8].

There are systems that combine subsystems where QSS algorithms perform better than classic ODE solvers with other subsystems in which classic solvers are the best choice. Motivated by these cases, and following the idea of multi–rate and mixed–mode or multi–method algorithms [9, 2, 10, 11, 12, 13], this work proposes a novel and general approach to use QSS to integrate some subsystems and classic solvers for the remaining subsystems.

Besides proposing a general methodology for combining QSS and classic solvers, this work also studies the convergence and stability properties of the resulting approximation and describes a particular implementation that combines LIQSS2 with CVODE solver. In addition, two simulation examples are presented, the first one corresponding to a model that represents the heating around a switching mode power electronic converter and the second one corresponding to a one dimensional advection–diffusion–reaction problem where the diffusion coefficient changes with the space.

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 algorithm, studies its convergence and stability, and describes the implementation in a simulation tool. Finally, Section 4 presents the simulation results and Section 5 concludes the article.

## 2. Background

This section provides an introduction to QSS algorithms and their implementation.

### *2.1. 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 [3] solves an approximate ODE called Quantized State System:

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

Here, $\mathbf{q}(t)$ is the *quantized state* vector. Each component of the quantized state $q_i(t)$ follows a piecewise constant trajectory that only changes when its difference with the corresponding state $x_i(t)$ reaches the quantum $\Delta Q_i$. Denoting $t_1, t_2, \ldots, t_k, \ldots$ the times at which the piecewise constant trajectory $q_i(t)$ changes , the quantized state trajectory is related to the corresponding state trajectory $x_i(t)$ as follows:

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

for $t_k < t \leq t_{k+1}$, where $t_{k+1}$ is the first time after $t_k$ at which $|x_i(t) - q_i(t_k)| = \Delta Q_i$. In addition, we consider that initially $\mathbf{q}(t_0) = \mathbf{x}(t_0)$. This defines an *hysteretic quantization function* generating trajectories like those depicted in Figure 1.

Since the quantized state trajectories $q_i(t)$ are piecewise constant, then, provided that the system is autonomous (or that $\mathbf{f}(\cdot, t)$ is piecewise constant with $t$), the state derivatives $\dot{x}_i(t)$ also follow piecewise constant trajectories and, consequently, the states $x_i(t)$ follow piecewise linear trajectories. In non autonomous systems Eq.(2) can be rewritten as

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

for some *input trajectories* $\mathbf{u}(t)$ that are approximated by piecewise constant trajectories $\mathbf{v}(t)$ such that the difference $v_i(t) - u_i(t)$ remains bounded by certain quantity (given by the input quantization). That way, the QSS1 approximation actually integrates the system

$$\dot{\mathbf{x}}(t) = \tilde{\mathbf{f}}(\mathbf{q}(t), \mathbf{v}(t))$$

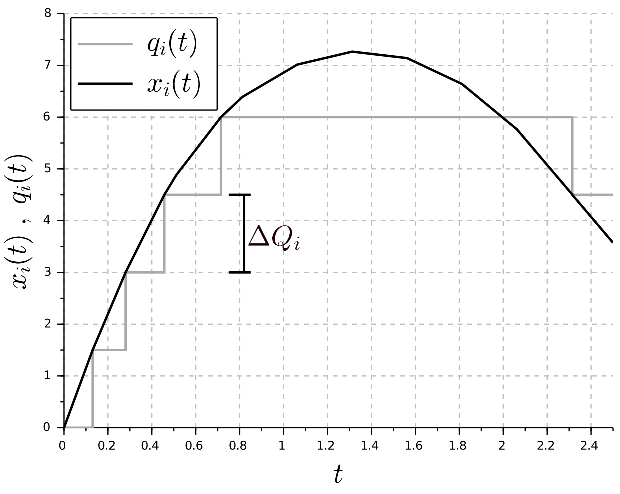

Figure 1: QSS1 hysteretic quantization function.

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, that for the case of autonomous system can be described as follows[1]:

**Algorithm 1**: QSS1.

```
//inputs: $t_i$ (initial time), $t_f$ (final time), $\mathbf{x_0}$ (initial state), $\mathbf{\Delta Q}$ (quantum vector)
//algorithm variables:
//$t$ is the current simulation time
//$\mathbf{x}$ is the continuous state vector computed by the algorithm (this is the approximate solution
    and thus the output of the algorithm)
//$\mathbf{q}$ is the quantized state vector
//$\dot{\mathbf{x}}$ is the state derivatives vector
//$t_j^\eta$ is the time of the next change of quantized state $q_j$
//$t_j^q$ is the time of the last change of quantized state $q_j$
//$t_j^x$ is the time of the last change of continuous state $x_j$
//Initialization
$t$ = $t_i$  //initial time ti
$\mathbf{x}$ = $\mathbf{x_0}$  //initial state
$\mathbf{q}$ = $\mathbf{x}$  //initial quantized state
for each $j \in [1, n]$
  $\dot{x}_j$ = $f_j(\mathbf{q}, t)$  // compute j-th initial state derivative
```

[1]In presence of a piecewise constant input trajectories the algorithm first checks if the next change is due to a quantized state or to an input change. In the second case, it proceeds like in line 29 of the algorithm but updating on the states depending on that input.

```
  $t_j^\eta$ = $t$ + $\Delta Q_j/|\dot{x}_j|$  // compute the time of the next change in the j-th quantized state
  $t_j^x$ = $t$  // time of the last change in the j-th continuous state
  $t_j^q$ = $t$  // time of the last change in the j-th quantized state
end for
//simulation cycle
while($t < t_f$) // simulate until final time $t_f$
  $t$ = min($t_j^\eta$) // adavance simulation time.
  $i$ = argmin($t_j^\eta$) // the i-th quantized state changes first
  $e$ = $t - t_i^x$ // elapsed time since last xi update. ($t^x$ is an array containing the last update
      time of each state.)
  $x_i$ = $x_i + \dot{x}_i \cdot e$ // update i-th state value
  $q_i$ = $x_i$ // update i-th quantized state
  $t_i^q$ = $t$  // time of the last change in the i-th quantized state
  $t_i^\eta$ = $t$ + $\Delta Q_i/|\dot{x}_i|$ // compute the time of then next change in the i-th quantized state
  for each $j \in [1,n]$ such_that $\dot{x}_j$ depends_on $q_i$
    $e$ = $t - t_j^x$ // elapsed time since last xj update
    $x_j$ = $x_j + \dot{x}_j \cdot e$ // update j-th state value
    if $j \neq i$ then $t_j^x$ = $t$ // last xj update
    $\dot{x}_j$ = $f_j(\mathbf{q}, t)$ // recompute j-th state derivative
    $t_j^\eta$ = min($\tau > t$) subject_to $|q_j - x_j(\tau)| = \Delta Q_j$ // recompute the time of the next change in the
        j-th quantized state
  end for
  $t_i^x$ = $t$ // last xi update
end while
```

The simulation algorithm works as follows. It looks which is the quantized state $q_i$ that changes first and advances the simulation time until that event. Then, it advances the state value $x_i$ (using the fact that $\dot{x}_i$ is constant in the period) and computes the new quantized state $q_i = x_i$. This new value for $q_i$ will change some state derivatives $\dot{x}_j = f_j(\mathbf{q}, t)$ provided that $q_i$ explicitly appears in the expression of $f_j$. Then, the algorithm recomputes the corresponding states $x_j$ and the next time of change for the corresponding quantized states $q_j$.

The time of the next change $t_j^\eta$ is computed as the first time after $t$ at which the difference between the piecewise constant trajectory $q_j(t)$ and the piecewise linear trajectory $x_j(t) = x_j + \dot{x}_j \cdot (t - t_j^x)$ becomes equal to the quantum $\Delta Q_j$. This is,

$$|x_j + \dot{x}_j \cdot (t_j^\eta - t_j^x) - q_j| = \Delta Q_j$$

The solution to this is equation is given by

$$t_j^\eta = \begin{cases} t_j^x + \dfrac{q_j + \Delta Q_j - x_j}{\dot{x}_j} & \text{if } \dot{x}_j > 0 \\ t_j^x + \dfrac{q_j - \Delta Q_j - x_j}{|\dot{x}_j|} & \text{if } \dot{x}_j < 0 \\ \infty & \text{otherwise} \end{cases}$$

In order to illustrate better the algorithm, we consider the following second order system

$$\dot{x}_1 = 2 - x_1$$
$$\dot{x}_2 = 2 \cdot x_1 - x_2$$

and its QSS1 approximation

$$\dot{x}_1 = 2 - q_1$$
$$\dot{x}_2 = 2 \cdot q_1 - q_2$$

with initial states $x_1(0) = x_2(0) = 0$ and quantum $\Delta Q_1 = \Delta Q_2 = 1$. Then, the algorithm works as follows:

- At $t = 0$ it computes the quantized states $q_1 = x_1 = 0$, $q_2 = x_2 = 0$, the state derivatives $\dot{x}_1 = 2 - q_1 = 2$, $\dot{x}_2 = 2 \cdot q_1 - q_2 = 0$, the time of the next changes $t_1^\eta = \Delta Q_1/|\dot{x}_1| = 1/2$, $t_2^\eta = \Delta Q_2/|\dot{x}_2| = \infty$. It also sets the time of the last state updates $t_1^x = t_2^x = 0$.

- The next step is then performed in $t = t_1^\eta = 1/2$, computing the states $x_1 = 1$, $x_2 = 0$, the quantized state $q_1 = 1$, the state derivatives $\dot{x}_1 = 2 - q_1 = 1$, $\dot{x}_2 = 2 \cdot q_1 - q_2 = 2$, the time of the next changes $t_1^\eta = t + \Delta Q_1/|\dot{x}_1| = t + 1 = 3/2$, $t_2^\eta = t + 1/|\dot{x}_2| = t + 1/2 = 1$. It also sets the time of the last state updates $t_1^x = t_2^x = t = 1/2$.

- The next step is then performed in $t = t_2^\eta = 1$, computing the state $x_2 = 1$, the quantized state $q_2 = 1$, the state derivative $\dot{x}_2 = 2 \cdot q_1 - q_2 = 1$, the time of the next change in $q_2$ $t_2^\eta = t + \Delta Q_2/|\dot{x}_2| = t + 1 = 2$. It also sets the time of the last state update for $x_2$ as $t_2^x = t = 1$.

  Notice that this step does not involve calculations in the first state, as $\dot{x}_1$ does not depend on $x_2$.

- The next step is then performed in $t = t_1^\eta = 3/2$, computing the states $x_1 = 2$, $x_2 = x_2 + \dot{x}_2 \cdot e = 1 + 1/2 = 3/2$, the quantized state $q_1 = 2$, the state derivatives $\dot{x}_1 = 2 - q_1 = 0$, $\dot{x}_2 = 2 \cdot q_1 - q_2 = 3$, the time of the next changes $t_1^\eta = t + \Delta Q_1/|\dot{x}_1| = \infty$, $t_2^\eta = t + (1/2)/|\dot{x}_2| = t + 1/6 = 5/3$. It also sets the time of the last state updates $t_1^x = t_2^x = t = 3/2$.

- The next step is then performed in $t = t_2^\eta = 5/3$, computing the state $x_2 = 2$, the quantized state $q_2 = 2$, the state derivative $\dot{x}_2 = 2 \cdot q_1 - q_2 = 2$, the time of the next change in $q_2$, $t_2^\eta = t + \Delta Q_2/|\dot{x}_2| = t + 1/2 = 13/6$. It also sets the time of the last state update for $x_2$ as $t_2^x = t = 5/3$.

- The next step is then performed in $t = t_2^\eta = 13/6$, computing the state $x_2 = 3$, the quantized state $q_2 = 3$, the state derivative $\dot{x}_2 = 2 \cdot q_1 - q_2 = 1$, the time of the next change in $q_2$, $t_2^\eta = t + \Delta Q_2/|\dot{x}_2| = t + 1 = 19/6$. It also sets the time of the last state update for $x_2$ as $t_2^x = t = 13/6$.

- The last step is then performed in $t = t_2^\eta = 19/6$, computing the state $x_2 = 4$, the quantized state $q_2 = 4$, the state derivative $\dot{x}_2 = 2 \cdot q_1 - q_2 = 0$, the time of the next change in $q_2$, $t_2^\eta = t + \Delta Q_2/|\dot{x}_2| = \infty$, and the simulation finishes as $t_1^\eta = t_2^\eta = \infty$.

This algorithm is clearly more involved than simple classic algorithms: it requires using more memory to store the state and quantized state vectors, as well as the last and next time arrays, In addition, it requires the knowledge of the system incidence matrix in order to determine which state derivative should be computed after each quantized state changes.

However, these disadvantages are sometimes compensated by the fact that the algorithm only computes when and where changes occur. In addition, it has important advantages in presence of discontinuities and it performs an intrinsic control of the global error.

The fact that the difference between the states $x_i$ and the corresponding quantized states $q_i$ is bounded by the quantum $\Delta Q_i$ allows to rewrite Eq.(2) as

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t) + \mathbf{\Delta x}(t), t) \tag{3}$$

where $\mathbf{\Delta x}(t) \triangleq \mathbf{q}(t) - \mathbf{x}(t)$ is a perturbation term bounded by the quantum. In consequence, the use of QSS1 is equivalent to the addition of a bounded perturbation to the original system and several properties regarding convergence, stability, and global error bounds can be easily derived [3, 2] for linear and non-linear systems. One of those properties establishes that the use of QSS in stable linear time invariant systems of the form $\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$ produces a global error that can be bounded by the formula

$$|\mathbf{x}(t) - \mathbf{x_a}(t)| \leq |\mathbf{V}| \cdot |\mathbb{R}e\{\mathbf{\Lambda}\}^{-1} \cdot \mathbf{\Lambda}| \cdot |\mathbf{V}^{-1}| \cdot \mathbf{\Delta Q}$$

where $\mathbf{x}$ and $\mathbf{x_a}$ are the QSS and the analytical solutions, and $\mathbf{\Lambda} = \mathbf{V}^{-1}\mathbf{A}\mathbf{V}$ is the Jordan decomposition of matrix $\mathbf{A}$. That way, there is a linear dependence between the quantum and the global error bound.

For these reasons, the quantum plays an equivalent role to that of the tolerance in variable step size algorithms. Here, the step size is usually controlled in order to accomplish with a *relative* error tolerance and this can be also achieved in QSS algorithm by using a quantum that changes with the signal amplitude: $\Delta Q_i = \max(\Delta Q_i^{rel}|x_i|, \Delta Q_i^{abs})$, such that $\Delta Q_i^{rel}$ is the relative quantum (relative tolerance) and $\Delta Q_i^{abs}$ is the absolute quantum (absolute tolerance).

Regarding the occurrence of discontinuities, the fact that the quantized states follow piecewise constant trajectories allows the straightforward detection of zero crossings[2]. Moreover, when a discontinuity occurs,

[2]In QSS simulation tools, discontinuities are detected using the quantized states $q_i$. Alternatively, the states $x_i$ can be

it will provoke the same effect than a change in a quantized state so the simulation does not need to be restarted like in classic algorithms. 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 [4].

In spite of these advantages, QSS1 only performs a first order approximation. This is, the number of steps performed is inversely proportional to the quantum which is proportional to the global error bound. In consequence, the computational costs grow linearly with the required accuracy. This limitation was improved with the definition of the second and third-order accurate QSS methods called QSS2 [14] and QSS3 [15], respectively.

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. In consequence, in QSS2 the number of steps grows with the square root of the accuracy while in QSS3 it grows with the cubic root.

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.

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 produce a large number of steps with their consequent computational cost [2]. To overcome this problem, the family of QSS methods was extended with a set of algorithms called Backward QSS (BQSS) [5] and Linearly Implicit QSS (LIQSS) [6, 7] that are appropriate to simulate some particular classes of stiff systems. While BQSS algorithm is only first order accurate, there LIQSS methods of orders 1 to 3 (LIQSS1, LIQSS2, and LIQSS3) combining the principles of QSS methods with those of classic linearly implicit solvers.

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 some type of iterations in order 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.

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. Besides allowing the efficient integration of several classes of stiff systems, LIQSS methods share the main advantages of QSS1 regarding discontinuity handling, stability and error bounds.

These advantages regarding efficient stiffness and discontinuity handling allow LIQSS methods to obtain results around one order of magnitude faster than the best classic ODE solvers in some application domains,

used. In that case, the zero crossing detection could be more accurate but in presence of non-linear threshold conditions that detection can be more involved.

including power electronics [16] and advection–reaction equations [17].

A limitation of LIQSS methods is that they compute the quantized states based on local calculations. For this reason, they can only avoid the appearance of spurious oscillations in a state $x_i(t)$ if they are the consequence of changes in its own quantized state $q_i(t)$. Thus, LIQSS algorithms are only efficient when the stiffness is due to the presence of large entries in the main diagonal of the Jacobian matrix (or at only one side of it). Although this restriction was recently relaxed avoiding oscillations between pairs of states [7], the solution is not general and some cases –like the MOL discretization of diffusion equations– are not efficiently treated.

Taking also into account the presence of input signals in non-autonomous systems, the different QSS algorithms can be simulated by the following generic algorithm:

**Algorithm 2**: (LI)QSS(N).

```
//inputs: $t_i$ (initial time), $t_f$ (final time), $\mathbf{x}_0$ (initial state), $\mathbf{\Delta Q}$ (quantum vector), $\mathbf{v}_k$ (a
    known succession of input polynomials that change at times $tv_k$).
//outputs: $\mathbf{x}, \mathbf{q}$ (current vectors of polynomials representing continuous and quantized state
    trajectories).
//other variables: $t$ (current simulation time), $t_n$ (time of the next quantized state change), $i$
    (next quantized state that changes), $k$ (current input polynomial section).

$t = t_i$
$k = 0$
($\mathbf{x}, \mathbf{q}, t_n, i$)=QSS_init($\mathbf{x_0}, \mathbf{v}_k, t, \mathbf{\Delta Q}$)
while ($t < t_f$) //simulation cycle
  $t$=min($t_n, tv_k$)
  if $t_n \leq tv_k$ then //quantized state change
    ($\mathbf{x}$, $\mathbf{q}$, $t_n$, $i$)=QSS_step($\mathbf{x}, \mathbf{q}, \mathbf{v}_k, t, \mathbf{\Delta Q}, i$)
  else //input change
    k=k+1
    ($\mathbf{x}$, $\mathbf{q}$, $t_n$, $i$)=QSS_input($\mathbf{x}, \mathbf{q}, \mathbf{v}_k, t, \mathbf{\Delta Q}$)
  end if
end while

//functions used
QSS_init($\mathbf{x_0}, \mathbf{v}, t, \mathbf{\Delta Q}$)
  $\mathbf{q}$ = constpoly($\mathbf{x_0}, t$) //initial quantized state polynomial (constant)
  for each $j \in [1, n]$
    $x_j$ = Statepoly($x_{0,j}, f_j, \mathbf{q}, \mathbf{v}, t$) //initial polynomial of order $N$ for the $j$-th state
    $t_j^\eta$ = nTime($x_j, q_j, \Delta Q_j$) //compute time of next change in $q_j$ according to the QSS method used
  end for
  $t_n$ = min($t_j^\eta$)
  $i$ = argmin($t_j^\eta$)
  return x, q, $t_n$, $i$

QSS_step($\mathbf{x},\mathbf{q},\mathbf{v},t,\mathbf{\Delta Q},i$)
  $x_i$ = advpoly($x_i,t$) //advance state polynomial to current time
  $q_j$ = Quantized($x_i,\Delta Q_i,t$) // compute new quantized state polynomial according to the QSS
      method used
  $t_i^\eta$ = nTime($x_i,q_i,\Delta Q_i$) //time of next change in $q_i$
  for each $j \in [1,n]$ such that $\dot{x}_j$ depends_on $q_i$
    $xj_{aux}$ = polyval($x_j,t$) //evaluate state polynomial at current time
    $x_j$ = Statepoly($xj_{aux},f_j,\mathbf{q},\mathbf{v},t$) //compute new state polynomial for the $j$-th state
    $t_j^\eta$ = nTime($x_j,q_j,\Delta Q_j$) //recompute time of next change in $q_j$
  end for
  $t_n$ = min($t_j^\eta$)
  $i$ = argmin($t_j^\eta$)
  return x, q, $t_n$, $i$

QSS_input($\mathbf{x},\mathbf{q},\mathbf{v},t,\mathbf{\Delta Q}$)
  for each $j \in [1,n]$ such that $\dot{x}_j$ depends_on v
    $xj_{aux}$ = polyval($x_j$,t) //evaluate state polynomial at current time
    $x_j$ = Statepoly($xj_{aux},f_j,\mathbf{q},\mathbf{v},t$) //compute new state polynomial for the $j$-th state
    $t_j^\eta$ = nTime($x_j,q_j,\Delta Q_j$) //recompute time of next change in $q_j$
  end for
  $t_n$ = min($t_j^\eta$)
  $i$ = argmin($t_j^\eta$)
  return x, q, $t_n$, $i$
```

In Algorithm 2, function `Statepoly` calculates a polynomial for the state $x_j(t)$ using its current value and its time derivatives (computed from function $f_j$ an their time derivatives). Similarly, function `Quantized` computes the quantized state polynomial according to the QSS method used while function `nTime` calculates the time for the next change in a quantized state. The way of computing these functions for the different methods can be easily deduced from the definition of the different QSS algorithms [7].

### *2.2. Stand–Alone QSS Solver*

While several Discrete Event System Specification (DEVS) simulation tools have implementations of different QSS algorithms [18, 19, 8], the most efficient and complete QSS tool is the Stand–Alone QSS solver [20].

The models in this solver are described in a sub-set of the Modelica language [21], and the tool automatically translates it into a C language piece of code containing the set of ODEs with the corresponding zero crossing functions and event handlers for discontinuous cases. The tool also extracts structure information (incidence matrices) and produces the code for the symbolic evaluation of the Jacobian matrix. The C code produced is then linked to the different QSS algorithms (QSS and LIQSS of order one to three) or to classic ODE solvers like DOPRI, DASSL, CVODE and IDA.

The fact that it provides all the structure information and the symbolic evaluation of the Jacobian matrices implies that the results obtained by this tool using classic ODE solvers are noticeably faster than the results obtained by other interfaces. In addition, the fact that it uses the same piece of code for the ODEs in the different algorithms (QSS and classic ODE integrators) allows fair performance comparisons among them.

## 3. Quantization-based mixed–mode integration

This section introduces the proposed methodology, studies its main properties and describes a particular implementation in the Stand–Alone QSS solver.

### *3.1. Proposed scheme*

Given the ODE of Eq.(1), and following the idea of multi–rate and mixed–mode schemes, we first split it as the the coupling of two subsystems,

$$\dot{\mathbf{x}}_{\mathbf{1}}(t) = \mathbf{f}_{\mathbf{1}}(\mathbf{x}_{\mathbf{1}}(t), \mathbf{x}_{\mathbf{2}}(t), t) \tag{4a}$$

$$\dot{\mathbf{x}}_{\mathbf{2}}(t) = \mathbf{f}_{\mathbf{2}}(\mathbf{x}_{\mathbf{1}}(t), \mathbf{x}_{\mathbf{2}}(t), t) \tag{4b}$$

such that $\mathbf{x}(t) = [\mathbf{x}_{\mathbf{1}}(t), \mathbf{x}_{\mathbf{2}}(t)]^T$, and $\mathbf{f}(\cdot) = [\mathbf{f}_{\mathbf{1}}(\cdot), \mathbf{f}_{\mathbf{2}}(\cdot)]^T$.

Then, we propose to integrate $\mathbf{x}_{\mathbf{1}}(t)$ using a QSS method and $\mathbf{x}_{\mathbf{2}}(t)$ using some classic algorithm (same order of the QSS method) with variable step size $h_k = t_{k+1} - t_k$, that is,

$$\dot{\mathbf{x}}_{\mathbf{1}}(t) = \mathbf{f}_{\mathbf{1}}(\mathbf{q}_{\mathbf{1}}(t), \hat{\mathbf{x}}_{\mathbf{2}}(t), t) \tag{5a}$$

$$\mathbf{x}_{\mathbf{2}}(t_{k+1}) = \mathbf{F}_{\mathbf{2}}(\mathbf{q}_{\mathbf{1}}(t_k), \dot{\mathbf{q}}_{\mathbf{1}}(t_k), \ldots, \mathbf{x}_{\mathbf{2}}(t_k), t_k, h_k) \tag{5b}$$

where Eq. (5b) represents the case of a single-step explicit method. In a more general case, $\mathbf{x}_{\mathbf{2}}(t_{k+1})$ could be the result of

$$\tilde{\mathbf{F}}_{\mathbf{2}}(\mathbf{q}_{\mathbf{1}}(t_k), \dot{\mathbf{q}}_{\mathbf{1}}(t_k), \ldots, \mathbf{q}_{\mathbf{1}}(t_{k+1}), \dot{\mathbf{q}}_{\mathbf{1}}(t_{k+1}), \ldots, \mathbf{x}_{\mathbf{2}}(t_{k+1}), \mathbf{x}_{\mathbf{2}}(t_k), \mathbf{x}_{\mathbf{2}}(t_{k-1}), \ldots, t_k, h_k) = 0 \tag{6}$$

which can also represent a multi-step implicit method.

The interface between both algorithms is performed by sampling the quantized states $\mathbf{q}_{\mathbf{1}}(t)$ and its derivatives (up to the order of the quantized state) at the time steps $t_k$ of the classic algorithm and by the construction of a polynomial (of the order of the quantized state) providing values for $\mathbf{x}_{\mathbf{2}}(t)$ between successive steps, i.e., for $t_k < t < t_{k+1}$:

$$\hat{\mathbf{x}}_{\mathbf{2}}(t) = \mathbf{x}_{\mathbf{2}}(t_k) + \mathbf{f}_{\mathbf{2}}(\mathbf{q}_{\mathbf{1}}(t_k), \mathbf{x}_{\mathbf{2}}(t_k), t_k) \cdot (t - t_k) + \ddot{\mathbf{x}}_{\mathbf{2}}(t_k) \cdot \frac{(t - t_k)^2}{2!} + \ldots \tag{7}$$

for $t \in [t_k, t_{k+1})$. This polynomial has the same order than the quantized state of the corresponding QSS algorithm (i.e., 0 for QSS1 and LIQSS1, 1 for QSS2 and LIQSS2, etc.).

In Eq.(7) $\ddot{\mathbf{x}}_{\mathbf{2}}(t_k)$ represents a coefficient conveniently defined for obtaining a third order scheme. The remaining terms of the polynomial (expressed by $\ldots$) are for higher order schemes, which cannot be currently implemented in practice since QSS algorithms of order higher than 3 were never used (in fact, most applications where QSS methods are useful, as those with frequent discontinuities, do not require the use of higher order algorithms as the step size will be limited by the time between events).

Although we are not limiting the definition of the scheme, we leave open the problem of the construction of the polynomial of Eq.(7) for any order greater or equal than 3. In any case, we shall prove later that the scheme converges irrespective of the way this polynomial is constructed.

### *3.2. Step size control at the classic solver side*

The basic idea of the mixed–mode approach is very simple. In addition, any QSS algorithm can integrate the system of Eq.(5a) taking into account the changes in $\hat{\mathbf{x}}_{\mathbf{2}}(t)$ at times $t_k$ (these changes are regarded as discontinuities that are straightforwardly treated by QSS). However, the formulation does not take into account, so far, the effect of the asynchronous changes in the quantized variables $\mathbf{q}_{\mathbf{1}}(t)$ during the computation of $\mathbf{x}_{\mathbf{2}}(t_{k+1})$ by the classic solver. In order to solve this problem, we propose to modify the usual step–size control strategy.

A standard step–size control algorithm would experience the following situation. Suppose a given variable $q_i$ computed using the QSS method explicitly appears on the right side of Eq. (5b). Then, a change in $q_i$ produces a change in the states derivatives computed using the classic method. If those changes of $q_i$ do not take place on the instants that correspond to those established by the step size control, they will produce an additional integration error.

A simple solution to this issue would be to perform a step using the classic method whenever a change in $q_i$ takes place (similarly to discontinuity detection). However, this solution would be very inefficient as it might enforce the classic method to perform a large number of unnecessary steps.

A far more efficient alternative, that we propose next, consists on estimating the additional error that the changes on $q_i$ introduce to the numerical integration of $\hat{\mathbf{x}}_{\mathbf{2}}(t)$. Then, we use this error estimation in order establish an additional limit to the step–size in order to accomplish the error tolerance.

#### *3.2.1. Additional numerical error in the classic method*

We deduce next a simple formula for estimating the additional numerical error provoked by ignoring changes in a quantized state $q_i(t)$ between the steps performed by the classic solver. In order to simplify the analysis, we consider a linear approximation to Eq.(4b) given by:

$$\dot{\mathbf{x}}_{\mathbf{2}}(t) = A_{22} \cdot \mathbf{x}_{\mathbf{2}}(t) + A_{2_i} \cdot x_i(t) + \mathbf{u}_{\mathbf{2}}(t) \tag{8}$$

where $x_i(t)$ is some component of $\mathbf{x}_{\mathbf{1}}(t)$ and $\mathbf{u}_{\mathbf{2}}(t)$ is the remaining term of the linearization. We also consider the following two approximations of Eq.(8):

$$\dot{\mathbf{x}}_{\mathbf{2}}(t) = A_{22} \cdot \mathbf{x}_{\mathbf{2}}(t) + A_{2_i} \cdot q_i(t) + \mathbf{u}_{\mathbf{2}}(t) \tag{9a}$$

$$\dot{\tilde{\mathbf{x}}}_{\mathbf{2}}(t) = A_{22} \cdot \tilde{\mathbf{x}}_{\mathbf{2}}(t) + A_{2_i} \cdot \tilde{q}_i(t) + \mathbf{u}_{\mathbf{2}}(t) \tag{9b}$$

where $q_i(t)$ is the quantized state trajectory computed by the QSS method (corresponding to the state $x_i(t)$) while $\tilde{q}_i(t)$ is the quantized state trajectory observed by the classic algorithm (taking into account that it only receives the sampled values at times $t_k$).

The difference between $q(t)$ and $\tilde{q}_i(t)$ can be observed in Figure 2. The goal of the following analysis is then to find a maximum value for the step size $h$ that guarantees that the difference between $q_i(t)$ and $\tilde{q}_i(t)$ is such that the difference between $\mathbf{x}_{\mathbf{2}}(t)$ and $\tilde{\mathbf{x}}_{\mathbf{2}}(t)$ –the solutions of Eqs.(9)– does not exceed the error tolerance.

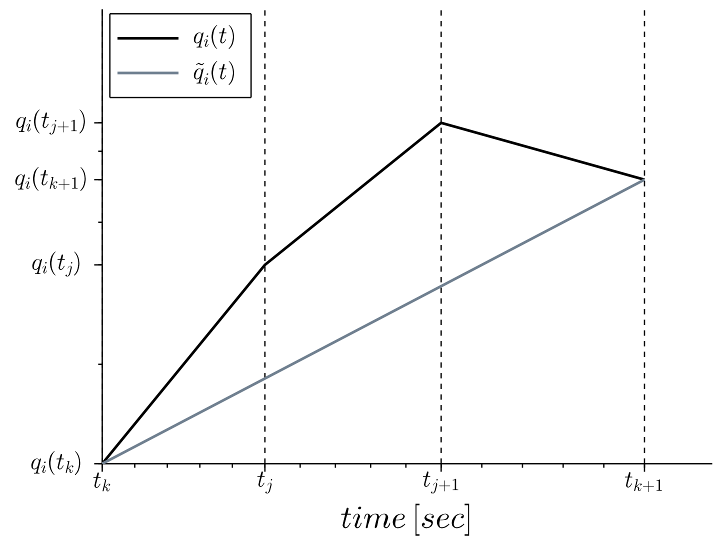

Figure 2: Computed and observed quantized state trajectory.

Defining then the error $\mathbf{e}(t) \triangleq \tilde{\mathbf{x}}_{\mathbf{2}}(t) - \mathbf{x}_{\mathbf{2}}(t)$ and subtracting Eq.(9a) from (9b), we obtain

$$\dot{\mathbf{e}}(t) = A_{22} \cdot \mathbf{e}(t) + A_{2_i} \cdot (\tilde{q}_i(t) - q_i(t))$$

This expression can be analytically solved, obtaining

$$\mathbf{e}(t) = e^{A_{22}\cdot(t-t_k)} \cdot \mathbf{e}(t_k) + \int_{t_k}^{t} e^{A_{22}\cdot(t-\tau)} \cdot A_{2_i} \cdot (\tilde{q}_i(\tau) - q_i(\tau))d\tau$$

We shall suppose too that, after a change in $q_i(t)$, the step size $h$ of the classic solver is limited by some constant $\bar{h}_{\max}$ such that there are no significant variations of the solution between successive integration steps. This is equivalent to consider $A_{22}\cdot(t-t_k)$ to be small, which implies that $e^{A_{22}\cdot(t-t_k)}$ is close to the identity matrix. In addition, as we are computing the error introduced during a step, we consider that it is initially zero, i.e., $\mathbf{e}(t_k) = 0$.

Then, the expression for the error can be approximated as

$$\mathbf{e}(t) \approx \int_{t_k}^{t} A_{2_i} \cdot (\tilde{q}_i(\tau) - q_i(\tau))d\tau = A_{2_i} \cdot \left( \int_{t_k}^{t} \tilde{q}_i(\tau)d\tau - \int_{t_k}^{t} q_i(\tau)d\tau \right) \tag{10}$$

That way, the error introduced by ignoring the changes in $q(t)$ between successive steps of the classic method is proportional to the difference between the integrals of $q_i(t)$ and $\tilde{q}_i(t)$. This information can be easily used to limit the step size such that $\mathbf{e}(t)$ lies within the prescribed tolerance.

### *3.2.2. Step size control in second order accurate algorithms*

If we consider, in particular, that both $q_i(t)$ and $\tilde{q}_i(t)$ follow piecewise linear trajectories (i.e., both algorithms are second order accurate schemes) as in Fig.2, then the previous analysis can be used to obtain an explicit formula for the maximum step size.

Let $t_j \in (t_k, t_{k+1})$ be an instant of change in $q_i$ and consider a time $t$ with $t_k < t_j < t < t_{k+1}$. Then,

$$\int_{t_k}^{t} q_i(\tau)d\tau = \int_{t_k}^{t_j} q_i(\tau)d\tau + \int_{t_j}^{t} q_i(\tau)d\tau = Q_i(t_j) + q_i(t_j)\cdot(t-t_j) + \dot{q}_i(t_j)\cdot\frac{(t-t_j)^2}{2} \tag{11}$$

where $Q_i(t_j) \triangleq \int_{t_k}^{t_j} q_i(\tau)d\tau$. Also,

$$\int_{t_k}^{t} \tilde{q}_i(\tau)d\tau = \frac{q_i(t_k) + q_i(t)}{2}\cdot(t-t_k) = \frac{q_i(t_k) + q_i(t_j) + \dot{q}_i(t_j)\cdot(t-t_j)}{2}\cdot(t-t_k) \tag{12}$$

Then replacing (11) and (12) in (10), we obtain:

$$\begin{aligned}
\mathbf{e}(t) &\approx A_{2_i}\cdot\left(\frac{q_i(t_k) + q_i(t_j) + \dot{q}_i(t_j)\cdot(t-t_j)}{2}\cdot(t-t_k) - Q_i(t_j) - q_i(t_j)\cdot(t-t_j) - \dot{q}_i(t_j)\cdot\frac{(t-t_j)^2}{2}\right)\\
&\approx A_{2_i}\cdot\left(\frac{q_i(t_k) - q_i(t_j) + \dot{q}_i(t_j)\cdot(t_j-t_k)}{2}\cdot(t-t_j) + \frac{q_i(t_k)+q_i(t_j)}{2}\cdot(t_j-t_k) - Q_i(t_j)\right)\\
&\approx A_{2_i}\left[\Delta_1(t_j,t_k)\cdot(t-t_j) + \Delta_2(t_j,t_k)\right]
\end{aligned} \tag{13}$$

where

$$\Delta_1(t_j,t_k) \triangleq \frac{q_i(t_k) - q_i(t_j) + \dot{q}_i(t_j)\cdot(t_j-t_k)}{2},\ \Delta_2(t_j,t_k) \triangleq \frac{q_i(t_k)+q_i(t_j)}{2}\cdot(t_j-t_k) - Q_i(t_j)$$

Then, the error at the end of the step $t_{k+1} = t_k + h_k$ can be computed as

$$\mathbf{e}(t_{k+1}) \approx A_{2_i} \left[\Delta_1(t_j, t_k) \cdot (h_k + t_k - t_j) + \Delta_2(t_j, t_k)\right] \tag{14}$$

whose $l$–th component can be computed as

$$e_l(t_{k+1}) \approx A_{2_{l,i}} \left[\Delta_1(t_j, t_k) \cdot (h_k + t_k - t_j) + \Delta_2(t_j, t_k)\right]$$

This value must be bounded, according to the tolerance settings, by certain quantity, i.e. $|e_l(t_{k+1})| \leq e_{\max,l}$, from which we obtain

$$|e_l(t_{k+1})| \approx |A_{2_{l,i}} \left[\Delta_1(t_j, t_k) \cdot (h_k + t_k - t_j) + \Delta_2(t_j, t_k)\right]| \leq e_{\max,l}$$

that must be accomplished for all $l$. This condition can be rewritten as

$$|\Delta_1(t_j, t_k) \cdot (h_k + t_k - t_j) + \Delta_2(t_j, t_k)| \leq r_i$$

where

$$r_i \triangleq \min_l \frac{e_{\max,l}}{|A_{2_{l,i}}|}$$

and finally, the step size $h_k$ can be bounded as $h_k < \min(\bar{h}_{\max}, h_{\max})$ where

$$h_{\max} \triangleq \begin{cases} \dfrac{r_i - \Delta_2(t_j, t_k)}{\Delta_1(t_j, t_k)} + t_j - t_k & \text{if } \Delta_1 > 0 \\ \dfrac{-r_i - \Delta_2(t_j, t_k)}{\Delta_1(t_j, t_k)} + t_j - t_k & \text{otherwise} \end{cases} \tag{15}$$

and $\bar{h}_{\max}$ is a parameter that ensures that $e^{A_{2,2}h}$ is close to the identity matrix so that Eq.(10) holds.

While this analysis was performed for second order accurate schemes, it can be easily extended to other orders.

## *3.3. Implementation of Mixed–Mode Algorithms*

The implementation of the proposed mixed–mode scheme only requires to coordinate a QSS and a classic solver, communicating between them the interface states (i.e., the components of $\mathbf{x_1}(t)$ that are used to compute $\dot{\mathbf{x}}_\mathbf{2}(t)$ and the components of $\mathbf{x_2}(t)$ that are used to compute $\dot{\mathbf{x}}_\mathbf{1}(t)$ in Eqs.(4a)–(4b)). The interface states must be also extrapolated to the current step times of both solvers.

The algorithm implementing the interface between both solvers is summarized below:

**Algorithm 3**: Mixed–Mode

```
//inputs: $t_i$ (initial time), $t_f$ (final time), [$\mathbf{x_1}_0$, $\mathbf{x_2}_0$] (partitioned initial state vector),
    $\mathbf{\Delta Q}$ (quantum vector), $\bar{h}_{\max}$ (maximum step size), $h_i$ (initial step size).
//variables:
```

```
//$\mathbf{x_1}$ is the vector of polynomials of continuous states computed by the QSS algorithm
//$\mathbf{q_1}$ is the vector of polynomials of quantized states computed by the QSS algorithm
//$\mathbf{x_2}$ is the vector of states computed by the CLASSIC method
//$\hat{\mathbf{x}}_\mathbf{2}$ is the vector of polynomials of interface states computed by the CLASSIC method defined
    by Eq.(7)
//$t_k$: is the classic method last step time
//$\hat{\mathbf{q}}_\mathbf{1}$ is a copy of the interface quantized state polynomials at time $t_k$
//CLASSIC.$t_n$ is the classic method next step time
//QSS.$t_n$ is the QSS next step time.

$t$ = $t_i$ //initial simulation time
$\mathbf{x_2}$ = $\mathbf{x_{20}}$ //initial values for CLASSIC states
$\hat{\mathbf{x}}_\mathbf{2}$ = constpoly(interface($\mathbf{x_{20}}$)) //initial polynomials for interface CLASSIC states
($\mathbf{x_1}$,$\mathbf{q_1}$,QSS.tn,i) = QSS_init($\mathbf{x_{10}}$,$\hat{\mathbf{x}}_\mathbf{2}$,$t$,$\mathbf{\Delta Q}$) //initialize QSS algorithm
$\hat{\mathbf{q}}_\mathbf{1}$ = interface($\mathbf{q_1}$) //initial polynomial for interface QSS states
$h$ = $h_i$ //initial classic step size
CLASSIC.$t_n$ = $t_i$ // the first step at the classic solver is at $t = t_i$
$t_k$ = $t_i$
while($t < t_f$) // simulate until final time tf
  $t$ = min(QSS.$t_n$,CLASSIC.$t_n$) //advance simulation time
  if (QSS.$t_n \leq$ CLASSIC.$t_n$) //QSS step
    $\hat{\mathbf{x}}_\mathbf{2}$ = advpoly($\hat{\mathbf{x}}_\mathbf{2}$,$t$) //advance state polynomials to current time
    $m$ = $i$ //the step is in the $m$--th state
    ($\mathbf{x_1}$,$\mathbf{q_1}$,QSS.$t_n$,$i$) = QSS_step($\mathbf{x_1}$,$\mathbf{q_1}$,$\hat{\mathbf{x}}_\mathbf{2}$,$t$,$\mathbf{\Delta Q}$,$m$) //QSS step
    if $m$-th state is_in_interface //the state that changed is in the interface
      $h_{\max}$ = compute_max_step_size($\mathbf{q_1}$,$\hat{\mathbf{q}}_\mathbf{1}$) //according to Eq.(15)
      if ($h > \min(h_{\max}, \bar{h}_{\max})$)
        $h$ = $\min(h_{\max}, \bar{h}_{\max})$
        CLASSIC.$t_n$ = $t_k$ + $h$
      end if
    end if
  else //CLASSIC step
    $\hat{\mathbf{q}}_\mathbf{1}$ = advpoly(interface($\mathbf{q_1}$),$t$) //advance interface quantized state polynomials to current
      time
    ($\mathbf{x_2}$,$h$) = CLASSIC_step($\hat{\mathbf{q}}_\mathbf{1}$,$\mathbf{x_2}$,$t$,$h$,$\bar{h}_{\max}$)
    CLASSIC.$t_n$=t+$h$
    $\hat{\mathbf{x}}_\mathbf{2}$ = genpoly(interface($\mathbf{x_2}$),$t$) //build interface state polynomials according to Eq.(7)
    ($\mathbf{x_1}$,QSS.tn,$i$) = QSS_input($\mathbf{x_1}$,$\mathbf{q_1}$,$\hat{\mathbf{x}}_\mathbf{2}$,$t$,$\mathbf{\Delta Q}$) //input change in QSS
    $t_k$ = $t$
  end if
end while
```

Algorithm 3, besides the functions from the generic QSS procedure (Algorithm 2), uses the following functions:

- `CLASSIC_step`: It implements Eqs.(5b) or (6) (according to the classic method used). For that goal, it considers the presence of the input polynomial $\hat{\mathbf{q}}_1$ in the right hand side function. In addition, this function recomputes the next step size $h$ (which can be then changed during QSS steps using the additional step size control).

- `interface`($\mathbf{x}$): It returns the subset of states that belongs to the interface. This is an optimization that avoids interpolating state variables that are computed by one method and are not used by the other algorithm.

A version of this algorithm was implemented in the Stand–Alone QSS Solver, combining LIQSS2 with the classic CVODE solver [22].

For that goal, the Stand–Alone QSS Solver was extended so that it produces a partitioned code for the given model containing the ODEs, zero crossing functions and event handlers that are used by each solver. In addition, the tool automatically computes the structure information regarding the interface between both solvers, so that the mixed–mode algorithm is able to know which states computed at one side are used to compute state derivatives (or zero crossing functions) at the other side.

From a user perspective, the implementation offers two choices for partitioning the model. The first option is the manual selection of the set of state variables that are computed by the CVODE solver. The second choice is an automatic partitioning method that uses LIQSS2 for all the state variables that are part of a zero crossing function and those that are affected by an event handler (i.e., the variables that are part of the discontinuous behavior) while it uses CVODE for the remaining variables. We say that a variable $x_i$ is affected by an event handler when the occurrence of the event can change the value of the $L^{th}$ derivative of $x_i$ where $L$ is a user defined parameter of the automatic partitioning algorithm.

More precisely, the Stand–Alone QSS Solver integrates systems of the form

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

where $\mathbf{d}(t)$ is a vector of discrete variables that only change under the occurrence of certain events triggered by some zero crossing functions of the form $ZC_i(\mathbf{x}(t), \mathbf{d}(t), t) = 0$. The changes of $\mathbf{d}(t)$ are dictated by the corresponding event handlers $\mathbf{d}(t^+) = H_i(\mathbf{x}(t), \mathbf{d}(t), t)$ where $\mathbf{d}(t^+)$ expresses the value after the event.

Then, a state $x_i$ is *affected* by a discontinuity provided that some component of $\mathbf{d}$ explicitly appears in the expression of $f_i(\mathbf{x}(t), \mathbf{d}(t), t)$.

The solver also has tools that automatically compute the incidence matrices from states to derivatives, from states to zero crossing functions, from event handlers to state derivatives and from event handlers to zero crossing functions. Making use of these incidence matrices, it is straightforward to find the sets of states affected by discontinuities. Then, looking to the variables that are affected by already affected variables, the set of variables affected in their second derivatives can be also obtained (and recursively for higher order derivatives)

In practice, the parameter $L$ should be chosen greater or equal than the order of the classic method in use. Otherwise, the discontinuities, correctly treated at the QSS side, may produce sudden derivative changes in the classic method which will reduce the step size affecting the performance of the overall scheme.

## *3.4. Convergence of the mixed–mode scheme*

The following theorem establishes sufficient conditions for the convergence of the solutions of the mixed–mode scheme to the analytical solution when the quantum of the QSS algorithm and the step size of the classic algorithm go simultaneously to zero.

**Theorem 1** (Convergence of the Mixed–Mode Scheme)**.** *Consider the system of Eq.*(4) *where* $\mathbf{f}(\mathbf{x},t) = [\mathbf{f_1}(\cdot),\mathbf{f_2}(\cdot)]^T$ *is locally Lipschitz in* $\mathbf{x}$ *on a compact set* $D \subset \mathbb{R}^n$*. Let* $\mathbf{x_a}(t)$ *be a solution of Eq.*(1) *from an initial condition* $\mathbf{x_a}(t_0)$ *such that* $\mathbf{x_a}(t)$ *remains in the strict interior of* $D$ *during the interval* $[t_0,t_f]$*.*

*Let* $\mathbf{x}(t) = [\mathbf{x_1}(t),\mathbf{x_2}(t)]^T$ *be the solution of the mixed mode scheme of Eqs.*(5)*–*(7) *from the same initial condition* $\mathbf{x}(t_0) = \mathbf{x_a}(t_0)$*, where the QSS algorithm uses quanta* $\Delta Q_i < \Delta Q_{\max}$ *and the classic algorithm is at least first order accurate and uses a bounded step size* $0 < h_k < h_{max}$*, and where*

$$\mathbf{x_2}(t) \triangleq \mathbf{x_2}(t_k) + \frac{\mathbf{x_2}(t_{k+1}) - \mathbf{x_2}(t_k)}{h_k}\cdot(t-t_k) \qquad t_k \le t < t_{k+1} \tag{16}$$

*Assume also that the coefficients of the extrapolation polynomial of Eq.*(7) *are bounded in* $D\times[t_0,t_f]$*.*

*Then,*

$$\lim_{\Delta Q_{\max},h_{\max}\to 0} \mathbf{x}(t) = \mathbf{x_a}(t) \tag{17}$$

*for all* $t \in [t_0,t_f]$*.*

The proof of this theorem can be found in Appendix A.

## *3.5. Stability of coupling QSS and classic methods*

We analyze next the stability of the mixed–mode scheme for a linear time invariant system of the form

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

Given that the presence of the input term $B\cdot\mathbf{u}(t)$ will not affect the stability conditions, we remove that term and split the system as before

$$\begin{bmatrix}\dot{\mathbf{x}}_\mathbf{1}(t)\\ \dot{\mathbf{x}}_\mathbf{2}(t)\end{bmatrix} = \begin{bmatrix}A_{11} & A_{12}\\ A_{21} & A_{22}\end{bmatrix}\begin{bmatrix}\mathbf{x_1}(t)\\ \mathbf{x_2}(t)\end{bmatrix}$$

We shall consider that a QSS method is used to compute $\mathbf{x_1}$ and that Backward Euler's algorithm is used to calculate $\mathbf{x_2}$. The analysis can be then easily extended to consider other classic algorithms.

The use of a QSS algorithm in $\mathbf{x_1}$ implies that it is computed as the solution of

$$\dot{\mathbf{x}}_\mathbf{1}(t) = A_{11} \cdot \mathbf{q_1}(t) + A_{12} \cdot \mathbf{x_2}(t_k) = A_{11} \cdot \mathbf{x_1}(t) + A_{12} \cdot \mathbf{x_2}(t_k) + A_{11} \cdot \mathbf{\Delta x_1}(t)$$

for $t \in [t_k, t_{k+1})$, where $\mathbf{\Delta x_1}(t) = \mathbf{q_1}(t) - \mathbf{x_1}(t)$.

The solution of the QSS approximation can be then computed for $t = t_{k+1}$ as

$$\begin{aligned}\mathbf{x_1}(t_{k+1}) &= e^{A_{11}\cdot h} \cdot \mathbf{x_1}(t_k) + \int_{t_k}^{t_{k+1}} e^{A_{11}\cdot(t-\tau)} \cdot A_{12} \cdot \mathbf{x_2}(t_k) d\tau + \underbrace{\int_{t_k}^{t_{k+1}} e^{A_{11}\cdot(t-\tau)} \cdot A_{11} \cdot \mathbf{\Delta x_1}(t) d\tau}_{\mathbf{\Delta}(t_k)} \\ &= e^{A_{11}\cdot h} \cdot \mathbf{x_1}(t_k) + A_{11}^{-1} \cdot e^{A_{11}\cdot h} \cdot A_{12} \cdot \mathbf{x_2}(t_k) + \mathbf{\Delta}(t_k)\end{aligned} \tag{18}$$

Then, the usage of Backward Euler's on the second subsystem yields

$$\mathbf{x_2}(t_{k+1}) = \mathbf{x_2}(t_k) + h \cdot A_{21} \cdot \mathbf{x_1}(t_k) + h \cdot A_{22} \cdot \mathbf{x_2}(t_{k+1})$$

that can be written as

$$\mathbf{x_2}(t_{k+1}) = (I - h \cdot A_{22})^{-1} \cdot h \cdot A_{21} \cdot \mathbf{x_1}(t_k) + (I - h \cdot A_{22})^{-1} \cdot \mathbf{x_2}(t_k) \tag{19}$$

Then, placing together Eqs. (18) and (19), we finally obtain

$$\mathbf{x}(t_{k+1}) = F \cdot \mathbf{x}(t_k) + G \cdot \mathbf{\Delta}(t_k) \tag{20}$$

where

$$F \triangleq \begin{bmatrix} e^{A_{11}\cdot h} & A_{11}^{-1} \cdot e^{A_{11}\cdot h} \cdot A_{12} \\ (I - h \cdot A_{22})^{-1} \cdot h \cdot A_{21} & (I - h \cdot A_{22})^{-1} \end{bmatrix} \qquad G \triangleq \begin{bmatrix} I \\ \mathbf{0} \end{bmatrix} \tag{21}$$

Notice that $\|\mathbf{\Delta x_1}\|$ is bounded by the maximum quantum used $\Delta Q_{\max}$. Then, the term $\mathbf{\Delta}(t)$ defined in Eq.(18) is also bounded by a quantity that is independent on the state $\mathbf{x}$. That way, provided that all the eigenvalues of matrix $F$ in Eq.(21) lie inside the unit circle, the numerical approximation has ultimately bounded solutions (i.e., it is practically stable) for any value of $h$ and $\Delta Q_{\max}$.

Since matrix $F$ only depends on $h$, the stability of the resulting scheme may depend on the step size used by the classic method, but it is independent on the maximum quantum $\Delta Q_{\max}$ used by the QSS algorithm.

## 4. Examples and Results

This section introduces two examples in which the proposed mixed–mode scheme shows noticeable advantages over classic and quantized state solvers. In both examples the reported results were obtained using the different algorithms (LIQSS, DASSL, CVODE, and the mixed–mode LIQSS2_CVODE) implemented in the Stand–Alone QSS Solver [3]. The different experiments were run on the same computer (an Intel Core i7

[3]The Stand–Alone QSS Solver is an open source project, available at https://github.com/CIFASIS/qss-solver. The models used in this article are part of the distribution and the results shown we obtained using git commit [9937e35].

CPU running a Linux Ubuntu 16.04 OS) and all the algorithms were used with the same error tolerances ($tol_{rel} = 1 \cdot 10^{-3}$ and $tol_{abs} = 1 \cdot 10^{-6}$ for classic solvers, and $\Delta Q_{rel} = 1 \cdot 10^{-3}$ and $\Delta Q_{abs} = 1 \cdot 10^{-6}$ for QSS based solvers).

## *4.1. DC-DC converter with heat sink*

The first example consists in a model of a buck converter (a widely used switched mode power electronic device) that takes into account both, the switched circuit and the thermal dynamics at the aluminum heat sink. The circuit is depicted in Fig.3, and it works alternating the switch position between *on* and *off* states at high frequency.

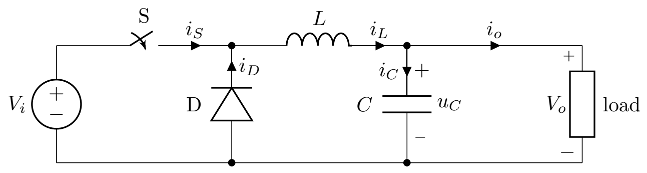

Figure 3: Buck DC-DC converter circuit.

The heat sink, in turn, is described by a heat equation

$$\frac{\partial u}{\partial t} = \sigma \cdot \frac{\partial^2 u}{\partial x^2} \qquad ; x \in [0, l] \; ; t \in [0, \infty)$$

where $l = 25$mm is the heat sink length and $\sigma$ is the diffusion coefficient given by

$$\sigma = \frac{\lambda}{\rho \cdot c}$$

where $\lambda = 209.3 \; J \, m^{-1} \, sec^{-1} \, K^{-1}$ is the aluminum thermal conductivity, $\rho = 2698.4 \; kg \, m^{-3}$ its density and $c = 900.0 \; J \, kg^{-1} \, K^{-1}$ its specific heat capacity.

The model considers the power losses during each change in the switch position, represented by power pulses as shown in Figure 4, that are transferred to the heat sink.

After discretizing the heat equation using the Methods of Lines with $N$ sections, the set of equations

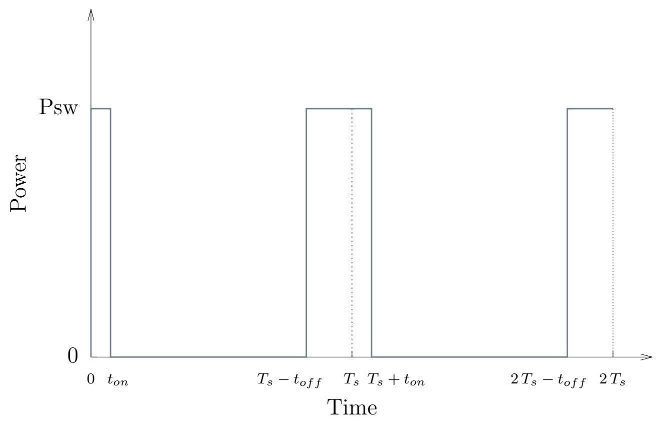

Figure 4: Power pulses in the switch.

describing the full model is the following one:

$$
\begin{aligned}
i_D &= \frac{i_L \cdot R_s - U}{R_s + R_d} \\
\dot{i}_L &= \frac{-\,i_D \cdot R_d - u_C}{L} \\
\dot{u}_C &= \frac{i_L - \frac{u_C}{R}}{C} \\
P &= \begin{cases} Psw & \text{if } t \in [T_s \cdot j - t_{off}, T_s \cdot j + t_{on}] \text{ where } j = 0, 1, \ldots \\ 0 & \text{otherwise} \end{cases} \\
\dot{u}_1 &= \sigma \cdot (P \cdot R_t - 2 \cdot u_1 + u_2) \cdot \frac{N^2}{l^2} \\
\dot{u}_N &= \sigma \cdot (298 - 2 \cdot u_N + u_{N-1}) \cdot \frac{N^2}{l^2} \\
\dot{u}_i &= \sigma \cdot (u_{i-1} - 2 \cdot u_i + u_{i+1}) \cdot \frac{N^2}{l^2} \quad \text{for } i = 2, N-1
\end{aligned}
\tag{22}
$$

The circuit was modeled using the following set of parameters: $U = 24\,V$ (input voltage), $C = 10\,mF$ (capacitor), $L = 10\,mH$ (inductor), $R = 10\,\Omega$ (load resistance), $f = 10\,kHz$ (switching frequency), $DC = 0.35$ (duty cycle, i.e., the fraction of time that the switch is in *on* state), $R_{\text{off}} = 10^5\,\Omega$ (off-state resistance), $R_{\text{on}} = 10^{-5}\,\Omega$ (on-state resistance), $R_t = 7.5K\,W^{-1}$ (heat sink thermal resistance), $t_{on} = 1.2\mu sec$ and $t_{off} = 5.5\mu sec.$

In order to analyze the performance of the proposed mixed–mode algorithm, we simulated this system using two different classic solvers (DASSL and CVODE), a QSS method (LIQSS2) and the mixed–mode LIQSS2_CVODE algorithm. This novel algorithm was used exploiting the automatic partitioning strategy with the $L$ parameter set to 7, that uses LIQSS2 for the circuit and the first 7 thermal stages, and CVODE for the remaining of the system. In all cases we considered initial conditions $i_L(t=0)=0$, $u_C(t=0)=0$ and $u_i(t=0)=298$.

We first considered a MOL discretization of the heat equation using 100 sections and simulated until a final time $t_f = 60$ obtaining the results reported in Table 1.

| **Integration Method** | **CPU [msec]** | **Error** |
|---|---|---|
| DASSL | $3.51 \cdot 10^6$ | $5.24 \cdot 10^{-4}$ |
| CVODE | $1.70 \cdot 10^7$ | $4.48 \cdot 10^{-3}$ |
| LIQSS2 | 11 131 | $6.82 \cdot 10^{-4}$ |
| LIQSS2_CVODE | 10 966 | $2.04 \cdot 10^{-4}$ |

Table 1: Solver performance comparison for the buck DC-DC converter with heat sink.

The errors reported are the mean of the errors in all state variables. In each state variable the error was computed as:

$$e_y = \sqrt{\frac{\sum_{k=1}^{5000}(y_{act_k} - y_{ref_k})^2}{\sum_{k=1}^{5000} y_{ref_k}^2}}$$

on 5000 sample points where the reference trajectory was obtained using LIQSS2 with tolerance settings of $\Delta Q_{rel} = 10^{-7}$ and $\Delta Q_{abs} = 10^{-9}$.

The results, showing a clear advantage of LIQSS2 and LIQSS2_CVODE over classic solvers, can be easily explained. The presence of frequent discontinuities in the power electronic converter enforces the classic solvers to perform very small steps on the whole system while LIQSS2 and LIQSS2_CVODE only perform those smalls steps in the variables involving discontinuities (i.e., those of the power electronic converter). In consequence, LIQSS2 and LIQSS2_CVODE are several orders of magnitude faster.

It is worth mentioning that most LIQSS2 simulation steps are performed at the power electronic converter variables, while the integration of the 100 variables representing the MOL discretization requires very few steps using either LIQSS2 or CVODE. Since LIQSS2 and LIQSS_CVODE use the same algorithm for the discontinuous parts (LIQSS2), the total computational cost is very similar and the fact that CVODE is faster than LIQSS2 in the remaining of the system does not change much the overall result.

We then compared both LIQSS2 and LIQSS2_CVODE algorithms as the number of sections $N$ increases. The results are depicted in Figure 5.

This time, as the number of sections $N$ grows, the mixed mode solver outperforms the pure LIQSS2 algorithm, and, for $N = 1000$, it is about 18 times faster. This can be explained by the fact that, as $N$ increases, the MOL discretization becomes more stiff [2], with a type of stiffness that LIQSS2 cannot

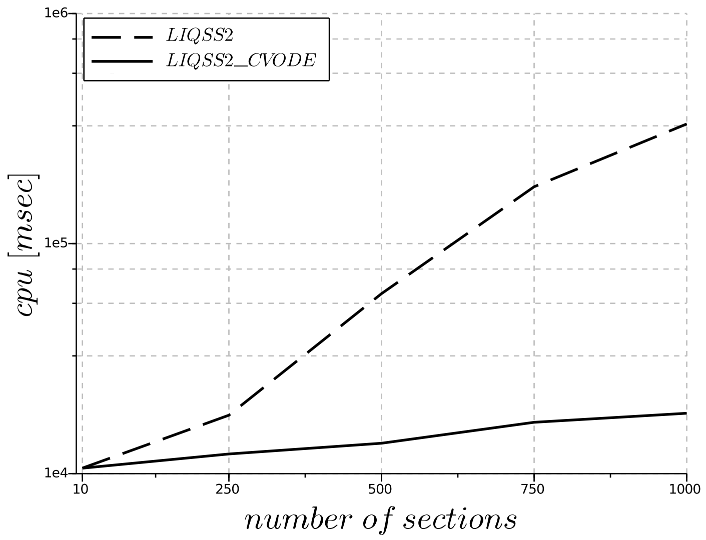

Figure 5: Comparison LIQSS2 vs LIQSS_CVODE as system size grows.

efficiently integrate and its performance is impoverished with the appearance of spurious oscillations. The mixed–mode LIQSS2_CVODE algorithm, however, uses CVODE to integrate the MOL approximation and this algorithm can efficiently integrate this part of the system.

Finally, Table 2 shows the number of individual state changes performed by the mixed–mode algorithm for different values of the error tolerance. As it can be seen, these experimental results verify that the order of convergence is, as expected, close to 2 (except for the lower accuracy settings, where the step size is more frequently limited by the occurrence of discontinuities than by the error tolerance). The order of convergence is computed as:

$$c_o = \frac{\log(tol_2/tol_1)}{\log(steps_1/steps_2)}$$

between successive experiments.

## *4.2. 1D Advection-Diffusion-Reaction (ADR) problem*

The following set of ODEs corresponds to the spatial discretization of a 1D ADR problem:

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

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

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

| **Relative Tolerance** | **Absolute Tolerance** | **Simulation steps** | **Simulation Error** | **Convergence Order** |
|---|---|---|---|---|
| $1 \cdot 10^{-2}$ | $1 \cdot 10^{-5}$ | 23 627 256 | $3.90 \cdot 10^{-3}$ | – |
| $1 \cdot 10^{-3}$ | $1 \cdot 10^{-6}$ | 399 03 067 | $2.04 \cdot 10^{-4}$ | 4.36 |
| $1 \cdot 10^{-4}$ | $1 \cdot 10^{-7}$ | 109 535 734 | $4.45 \cdot 10^{-5}$ | 2.30 |
| $1 \cdot 10^{-5}$ | $1 \cdot 10^{-8}$ | 324 167 990 | $1.43 \cdot 10^{-5}$ | 2.11 |
| $1 \cdot 10^{-6}$ | $1 \cdot 10^{-9}$ | 998 532 903 | $7.22 \cdot 10^{-7}$ | 2.05 |

Table 2: Simulation results using LIQSS2_CVODE with different error tolerances.

where $N$ is the number of grid points, $\Delta x = \frac{10}{N}$, and we consider parameters $a = 1$, $r = 1000$, and

$$d_i = \begin{cases} 1 \cdot 10^{-6} & \text{if } i \in [1, N/2 - 1] \\ 10 & \text{otherwise} \end{cases} \tag{23}$$

We also consider initial conditions $u_i(0) = 0$ for $i = 1, \dots, N$ and a boundary condition $u_0(t) = 1$.

MOL discretizations of ADR equations can be efficiently integrated by LIQSS methods provided that the diffusion coefficient $d_i$ is small compared to the advection coefficient $a$ [17]. Otherwise, BDF approximations like CVODE offer better results. In this case, the diffusion coefficient changes with the space in Eq.(23), so that we can expect that LIQSS2 is better at the first stages and CVODE is better in the last stages, and the mixed–mode algorithm can offer the best alternative.

In order to corroborate this, the system was first simulated for $N = 1000$ until a final time $t_f = 10$ using classic methods, as well as LIQSS2 and LIQSS2_CVODE. For the Mixed–Mode algorithm we used LIQSS2 in the first $N/2$ state variables (those with low diffusion coefficient) and CVODE in the remaining states.

We measured the errors using the same strategy as in the previous example but computing this time the reference values with DASSL and tolerances $tol_{rel} = 1 \cdot 10^{-7}$ and $tol_{abs} = 1 \cdot 10^{-10}$. The performance of the different algorithms is reported in Table 3.

| **Integration Method** | **Time [msec]** | **Error** |
|---|---|---|
| DASSL | 3 143 | $1.44 \cdot 10^{-3}$ |
| CVODE | 220 | $2.86 \cdot 10^{-4}$ |
| LIQSS2 | 2 023 | $8.33 \cdot 10^{-4}$ |
| LIQSS2_CVODE | 40 | $5.76 \cdot 10^{-4}$ |

Table 3: Comparison of simulation results for the ADR system with $N = 1000$

As expected, the mixed–mode algorithm provides the best results. This time LIQSS2_CVODE is more that 5 times faster than CVODE , more than 50 times faster than LIQSS2 and more than 75 times faster than DASSL. The poor performance of LIQSS2 alone is due to the same reason of the previous example (the appearance of spurious oscillations in the diffusion–dominated equation). In addition, the noticeable advantages of CVODE against DASSL can be explained by the fact that the former uses a sparse Jacobian and the system is large.

We also compared the performance of the different algorithms as the size of the problem ($N$) increases, reporting the results in Fig.6.

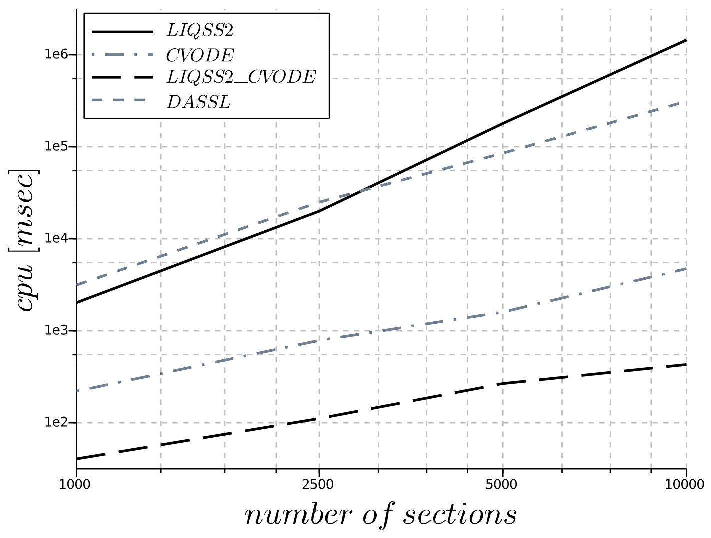

Figure 6: Simulation results of the ADR system varying parameter $N$.

The figure shows that the mixed–mode algorithm is the best choice for all values of $N$. As $N$ increases the advantage of LIQSS2_CVODE also increases and it becomes 11 times faster than CVODE for $N = 10000$.

## 5. Conclusions

This article introduced mixed–mode numerical integration algorithms that combine classic ODE integration methods with QSS methods and studied their convergence and numerical stability properties. The implementation of one algorithm of this type that combines LIQSS2 with CVODE was also described. Simulation results comparing the performance of this implementation with that of different ODE solvers exhibited noticeable advantages of the proposed mixed–mode scheme, reducing the CPU times in more than one order of magnitude.

The examples analyzed demonstrated that the novel approach is useful in presence of systems with heterogeneous dynamics, in which some subsystems are better suited for QSS methods while the remaining subsystems can be more efficiently integrated by classic ODE solvers.

Regarding future work, there is so far only one mixed–mode algorithm implementation (LIQSS2_CVODE). Combinations of different QSS algorithms with different classic solvers may lead to more efficient results in some problems. It is also important to study the use of these algorithms in different multi–domain applications where heterogeneous dynamics are usually encountered.

Another important improvement would be a relaxation in the assumption made on Section 3.2 about the parameter $\bar{h}_{\max}$ that limits the step size of the classic algorithm. That parameter was necessary to ensure the validity of Eq.(10) which allowed to establish a simple expression for the additional numerical error. A possible way to overcome this limitation would be to use an expression that approximates the exponential matrix $e^{A_{2,2}\cdot t}$ instead of approximating it by the identity matrix under the assumption that $t$ is small.

## Acknowledgements

This work was partially supported by grants ANPCYT PICT-2017-2436 and CONICET PIP 2017 0100485.

## 6. References

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

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

[3] E. Kofman, S. Junco, Quantized State Systems. A DEVS Approach for Continuous System Simulation, Transactions of SCS 18 (3) (2001) 123–132.

[4] E. Kofman, Discrete Event Simulation of Hybrid Systems, SIAM Journal on Scientific Computing 25 (5) (2004) 1771–1797.

[5] 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 (4) (2012) 387–407.

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

[7] F. Di Pietro, G. Migoni, E. Kofman, Improving linearly implicit quantized state system methods, Simulation: Transactions of the Society for Modeling and Simulation International 95 (2) (2019) 127–144.

[8] B. P. Zeigler, A. Muzy, E. Kofman, Theory of Modeling and Simulation: Discrete Event & Iterative System Computational Foundations, Academic Press, 2018.

[9] A. Bartel, M. Günther, A multirate w-method for electrical networks in state–space formulation, Journal of Computational and Applied Mathematics 147 (2) (2002) 411–425.

[10] V. Savcenco, Construction of a multirate rodas method for stiff odes, Journal of Computational and Applied Mathematics 225 (2) (2009) 323–337.

[11] E. M. Constantinescu, A. Sandu, Extrapolated multirate methods for differential equations with multiple time scales, Journal of Scientific Computing 56 (1) (2013) 28–44.

[12] B. Thiele, M. Otter, S. E. Mattsson, Modular multi-rate and multi-method real-time simulation, in: Proceedings of the 10th International Modelica Conference, 2014.

[13] F. Casella, A. Ranade, Efficient modelling and simulation of multi-domain smart grids using modelica and multi-rate integration algorithms, in: IECON 2016-42nd Annual Conference of the IEEE Industrial Electronics Society, IEEE, 2016, pp. 6285–6291.

[14] E. Kofman, A Second Order Approximation for DEVS Simulation of Continuous Systems, Simulation: Transactions of the Society for Modeling and Simulation International 78 (2) (2002) 76–89.

[15] E. Kofman, A Third Order Discrete Event Simulation Method for Continuous System Simulation, Latin American Applied Research 36 (2) (2006) 101–108.

[16] G. Migoni, F. Bergero, E. Kofman, J. Fernández, Quantization-Based Simulation of Switched Mode Power Supplies., Simulation: Transactions of the Society for Modeling and Simulation International 91 (4) (2015) 320–336.

[17] F. Bergero, J. Fernández, E. Kofman, 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 92 (1) (2016) 47–61.

[18] M. C. D'Abreu, 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, IEEE, 2005, pp. 229–236.

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

[20] J. Fernández, E. Kofman, A Stand-Alone Quantized State System Solver for Continuous System Simulation., Simulation: Transactions of the Society for Modeling and Simulation International 90 (7) (2014) 782–799.

[21] S. E. Mattsson, H. Elmqvist, M. Otter, Physical system modeling with modelica, Control Engineering Practice 6 (4) (1998) 501–510.

[22] S. D. Cohen, A. C. Hindmarsh, P. F. Dubois, Cvode, a stiff/nonstiff ode solver in c, Computers in physics 10 (2) (1996) 138–143.

## Appendix A. Proof of Theorem 1

*Proof.* Since $\mathbf{f}$ is locally Lipschitz in $D$, there exists a constant $L > 0$ such that

$$\| \mathbf{f}(\mathbf{x_c}, t) - \mathbf{f}(\mathbf{x_b}, t) \| < L \cdot \| \mathbf{x_c} - \mathbf{x_b} \| \tag{A.1}$$

for all $\mathbf{x_c}, \mathbf{x_b} \in D$ and for all $t \in [t_0, t_f]$. Here, the symbol $\|\mathbf{x}\|$ denotes the infinite norm of vector $\mathbf{x}$.

The condition of Eq.(A.1) implies that there exists a constant $F_{\max} > 0$ such that

$$\| \mathbf{f}(\mathbf{x}, t) \| < F_{\max} \tag{A.2}$$

for $t \in [t_0, t_f]$ and $\mathbf{x} \in D$.

Then, taking into account that the classic method is at least first order accurate, we can write:

$$\mathbf{x_2}(t_{k+1}) = \mathbf{x_2}(t_k) + h_k \cdot \mathbf{f_2}(\mathbf{q_1}(t_k), \mathbf{x_2}(t_k), t_k) + \frac{h_k^2}{2} \cdot \mathbf{r_h}(t_k) \tag{A.3}$$

where

$$\| \mathbf{r_h}(t_k) \| < r_{\max} \quad \forall t_k \in [t_0, t_f] \tag{A.4}$$

and where $r_{\max}$ is a constant that bounds the remainder term $\mathbf{r_h}(t_k)$ for any step size $h_k \leq h_{\max}$, and for any $[\mathbf{q_1}(t_k), \mathbf{x_2}(t_k)]^T \in D$. Notice that $r_{\max}$ may depend on the numerical method used, the expression of $\mathbf{f_2}$, and the maximum step–size bound $h_{\max}$.

Replacing the term $\mathbf{x_2}(t_{k+1}) - \mathbf{x_2}(t_k)$ in Eq.(16) with Eq.(A.3), we obtain

$$\mathbf{x_2}(t) = \mathbf{x_2}(t_k) + [\mathbf{f_2}(\mathbf{q_1}(t_k), \mathbf{x_2}(t_k), t_k) + h_k \cdot \mathbf{r_h}(t_k)](t - t_k),$$

an expression that can be also obtained after integrating both sides of the ODE

$$\dot{\mathbf{x}}_\mathbf{2}(t) = \mathbf{f_2}(\mathbf{q_1}(t_k), \mathbf{x_2}(t_k), t_k) + h_k \cdot \mathbf{r_h}(t_k) \tag{A.5}$$

in the interval $[t_k, t]$ with $t_k \leq t < t_{k+1}$. That way, $\mathbf{x_2}(t)$ defined in Eq.(16) is the solution of the ODE of Eq.(A.5) from the initial condition $\mathbf{x_2}(t_0)$.

In consequence, the system given by Eq. (5) has identical solution to the one given by

$$\dot{\mathbf{x}}_\mathbf{1}(t) = \mathbf{f_1}(\mathbf{q_1}(t), \tilde{\mathbf{x}}_\mathbf{2}(t), t) \tag{A.6a}$$

$$\dot{\mathbf{x}}_\mathbf{2}(t) = \mathbf{f_2}(\mathbf{q_1}(t_k), \mathbf{x_2}(t_k), t_k) + h_k \cdot \mathbf{r_h}(t_k) \tag{A.6b}$$

at discrete times $t_k$. Using this fact, we shall prove that $\mathbf{x}(t) = [\mathbf{x_1}(t), \mathbf{x_2}(t)]^T \to \mathbf{x_a}(t)$.

Defining

$$\mathbf{\Delta_{1,1}}(t) \triangleq \mathbf{q_1}(t) - \mathbf{x_1}(t), \quad \mathbf{\Delta_{1,2}}(t) \triangleq \tilde{\mathbf{x}}_\mathbf{2}(t) - \mathbf{x_2}(t)$$

$$\mathbf{\Delta_{2,1}}(t) \triangleq \mathbf{q_1}(t_k) - \mathbf{x_1}(t), \quad \mathbf{\Delta_{2,2}}(t) \triangleq \mathbf{x_2}(t_k) - \mathbf{x_2}(t)$$

we rewrite Eq.(A.6) as follows

$$\dot{\mathbf{x}}_\mathbf{1}(t) = \mathbf{f_1}(\mathbf{x_1}(t) + \mathbf{\Delta_{1,1}}(t), \mathbf{x_2}(t) + \mathbf{\Delta_{1,2}}(t), t) \tag{A.7a}$$

$$\dot{\mathbf{x}}_\mathbf{2}(t) = \mathbf{f_2}(\mathbf{x_1}(t) + \mathbf{\Delta_{2,1}}(t), \mathbf{x_2}(t) + \mathbf{\Delta_{2,2}}(t), t_k) + h_k \cdot \mathbf{r_h}(t_k) \tag{A.7b}$$

The perturbation term $\mathbf{\Delta_{1,1}}(t) = \mathbf{q_1}(t) - \mathbf{x_1}(t)$ can be bounded as

$$\|\mathbf{\Delta_{1,1}}(t)\| \leq \Delta Q_{\max} = c_{1,1} \Delta Q_{\max} \tag{A.8}$$

since QSS methods always ensure that the difference between the continuous and the quantized states they compute are bounded by the quantum (recall that they perform a step whenever the difference reaches the quantum size).

From Eqs.(7) and (16), we obtain

$$\mathbf{\Delta_{1,2}}(t) = [\mathbf{f_2}(\mathbf{q_1}(t_k), \mathbf{x_2}(t_k), t_k) - \frac{\mathbf{x_2}(t_{k+1}) - \mathbf{x_2}(t_k)}{h_k}] \cdot (t - t_k) +$$
$$+ \ddot{\mathbf{x}}_\mathbf{2}(t_k) \cdot \frac{(t - t_k)^2}{2!} + \dots$$

Then, using Eq.(A.3), it results

$$\begin{aligned}\mathbf{\Delta_{1,2}}(t) &= [\mathbf{f_2}(\mathbf{q_1}(t_k), \mathbf{x_2}(t_k), t_k) - \mathbf{f_2}(\mathbf{q_1}(t_k), \mathbf{x_2}(t_k), t_k) - \frac{h_k}{2} \cdot \mathbf{r_h}(t_k)] \cdot (t - t_k) + \\ &+ \ddot{\mathbf{x}}_\mathbf{2}(t_k) \cdot \frac{(t-t_k)^2}{2!} + \dots \\ &= [\ddot{\mathbf{x}}_\mathbf{2}(t_k) \cdot \frac{(t-t_k)}{2!} + \dots - \frac{h_k}{2} \cdot \mathbf{r_h}(t_k)] \cdot (t - t_k)\end{aligned}$$

Then, since the extrapolation polynomial coefficients are bounded in $D$, and recalling Eq.(A.4) and the fact that $t - t_k \le h_{\max} \le h\max$, it results that

$$\|\mathbf{\Delta_{1,2}}(t)\| \le c_{1,2} \cdot h_{\max} \tag{A.9}$$

where $c_{1,2}$ is an upper bound for $\|\ddot{\mathbf{x}}_\mathbf{2}(t_k) \cdot \frac{(t-t_k)}{2!} + \dots - \frac{h_k}{2} \cdot \mathbf{r_h}(t_k)\|$. Notice that this bound is valid under the assumption that $\mathbf{f_1}$ and $\mathbf{f_2}$ in Eq.(A.7) are evaluated inside $D$.

Regarding the bound for $\mathbf{\Delta_{2,1}}(t)$, we know that there exists $t^* \in [t_k, t]$ such that, according to the mean value inequality, it results that

$$\|\mathbf{x_1}(\mathbf{t}) - \mathbf{x_1}(\mathbf{t_k})\| \le \|\dot{\mathbf{x}}_\mathbf{1}(t^*)\| \cdot (t - t_k) \le F_{\max} \cdot h_k$$

and then, recalling that $\mathbf{\Delta_{2,1}}(t) = \mathbf{q_1}(t_k) - \mathbf{x_1}(t) = \mathbf{q_1}(t_k) - \mathbf{x_1}(t_k) + \mathbf{x_1}(t_k) - \mathbf{x_1}(t)$, we obtain

$$\|\mathbf{\Delta_{2,1}}(t)\| \le \Delta Q_{\max} + F_{\max} \cdot h_{\max} \le c_{2,1} \max(\Delta Q_{\max}, h_{\max}) \tag{A.10}$$

a bound that is also valid under the assumption that $\mathbf{f_1}$ and $\mathbf{f_2}$ in Eq.(A.7) are evaluated inside $D$.

The last perturbation term, $\mathbf{\Delta_{2,2}}(t)$, can be computed from Eqs.(A.3) and (16) as

$$\begin{aligned}\mathbf{\Delta_{2,2}}(t) = \mathbf{x_2}(t) - \mathbf{x_2}(t_k) &= \frac{\mathbf{x_2}(t_{k+1}) - \mathbf{x_2}(t_k)}{h_k}(t - t_k) = \\ &= [\mathbf{f_2}(\mathbf{q_1}(t_k), \mathbf{x_2}(t_k), t_k) + \frac{h_k}{2} \cdot \mathbf{r_h}(t_k)] \cdot (t - t_k)\end{aligned}$$

and then,

$$\|\mathbf{\Delta_{2,2}}(t)\| \le (F_{\max} + h_{\max} \cdot r_{\max}) \cdot h_{\max} = c_{2,2} \cdot h_{\max} \tag{A.11}$$

This inequality also assumes that $\mathbf{f_1}$ and $\mathbf{f_2}$ in Eq.(A.7) are evaluated inside $D$.

The fact that the analytical solution $\mathbf{x_a}(t)$ is in the strict interior of $D$ for $t \in [t_0, t_f]$ implies that there exists a constant $d > 0$ such that

$$\text{dist}(\mathbf{x_a}(t), \partial D) > d \quad \forall t \in [t_0, t_f] \tag{A.12}$$

Take $T \in (0, t_f - t_0]$ so that

$$T < \frac{d}{4 \cdot (F_{\max} + h_{\max} \cdot r_{\max})} \tag{A.13}$$

and let $\Delta Q_{\max}$ and $h_{\max}$ be small enough such that the right hand side of Eqs.(A.8),(A.9),(A.10), and (A.11) are all less than $d/4$.

Let $t_a$ be some instant of time at which $\operatorname{dist}(\mathbf{x}(t_a), \partial D) > d/2$, and let $t_e \in (t_a, t_a + T)$ be the first instant of time at which $\operatorname{dist}(\mathbf{x}(t_e), \partial D) \leq d/4$. Thus, $\mathbf{f_1}$ and $\mathbf{f_2}$ are both computed inside $D$ in Eq.(A.7) during the interval $[t_a, t_e)$ and the bounds given for $\|\mathbf{\Delta_{i,j}}(t)\|$ by Eqs.(A.8),(A.9),(A.10), and (A.11) are valid in that interval.

Then, according to the mean value inequality, there exists $t^* \in [t_a, t_e]$ such that

$$\|\mathbf{x}(t_e) - \mathbf{x}(t_a)\| \leq \|\dot{\mathbf{x}}(t^*)\| \cdot (t_e - t_a) \leq (F_{\max} + \bar{h} \cdot r_{\max}) \cdot T < \frac{d}{4}$$

Since $\operatorname{dist}(\mathbf{x}(t_a), \partial D) > d/2$, the last inequality implies that $\operatorname{dist}(\mathbf{x}(t_e), \partial D) > d/4$ contradicting the definition of $t_e$ and showing that the distance from the state $\mathbf{x}(t)$ to the boundary of $D$ is at least $d/4$ in $[t_a, t_a + T]$.

Let $\varepsilon > 0$ and consider some $\delta > 0$ small enough such that $\Delta Q_{\max} < \delta$ and $h_{\max} < \delta$ imply that the right hand side of the inequalities (A.8),(A.9),(A.10), and (A.11) are all less than $d/4$.

Assume now that there is some time $t_b \geq t_0$, with $t_b \leq t_f - T$, such that

$$\|\mathbf{x}(t) - \mathbf{x_a}(t)\| < \min(\varepsilon, d/4) \tag{A.14}$$

for all $t \in [t_0, t_b]$. We shall prove next that we can find $\delta > 0$ such that the conditions $\Delta Q_{\max} < \delta$ and $h_{\max} < \delta$ imply that $\|\mathbf{x}(t) - \mathbf{x_a}(t)\| < \varepsilon$ for all $t \in [t_0, t_b + T]$.

Notice that $\|\mathbf{x}(t_b) - \mathbf{x_a}(t_b)\| < d/4$ implies that $\operatorname{dist}(\mathbf{x}(t_b), \partial D) > d/2$, which in turn implies that $\operatorname{dist}(\mathbf{x}(t), \partial D) > d/4$ for all $t \in [t_0, t_b + T]$. Then, recalling that $\delta$ is small enough such that $\|\mathbf{\Delta_{i,j}}(t)\| < d/4$, it results that functions $\mathbf{f_1}$ and $\mathbf{f_2}$ in Eq.(A.7) are both computed inside $D$, which in turn implies the validity of Eqs.(A.8),(A.9),(A.10), and (A.11).

Recalling that $\mathbf{x_a}(t) = [\mathbf{x_{a,1}}(t), \mathbf{x_{a,2}}(t)]^T$ is the solution of Eq.(4), and subtracting Eq.(4a) from Eq.(A.7a), we obtain

$$\dot{\mathbf{x}}_\mathbf{1}(t) - \dot{\mathbf{x}}_\mathbf{a,1}(t) = \mathbf{f_1}(\mathbf{x_1}(t) + \mathbf{\Delta_{1,1}}(t), \mathbf{x_2}(t) + \mathbf{\Delta_{1,2}}(t), t) - \mathbf{f_1}(\mathbf{x_{a,1}}(t), \mathbf{x_{a,2}}(t), t)$$

or, equivalently,

$$\mathbf{x_1}(t) - \mathbf{x_{a,1}}(t) = \int_{t_0}^{t} [\mathbf{f_1}(\mathbf{x_1}(\tau) + \mathbf{\Delta_{1,1}}(\tau), \mathbf{x_2}(\tau) + \mathbf{\Delta_{1,2}}(\tau), \tau) - \mathbf{f_1}(\mathbf{x_{a,1}}(\tau), \mathbf{x_{a,2}}(\tau), \tau)] \mathrm{d}\tau$$

and, using the Lipschitz condition on $D$,

$$\begin{aligned}
\|\mathbf{x_1}(t) - \mathbf{x_{a,1}}(t)\| &\leq \int_{t_0}^{t} L \cdot [\|\mathbf{x_1}(\tau) + \mathbf{\Delta_{1,1}}(\tau) - \mathbf{x_{a,1}}(\tau)\| + \|\mathbf{x_2}(\tau) + \mathbf{\Delta_{1,2}}(\tau) - \mathbf{x_{a,2}}(\tau)\|] \mathrm{d}\tau \\
&\leq \int_{t_0}^{t} 2L \cdot \|\mathbf{x}(\tau) - \mathbf{x_a}(\tau)\| \mathrm{d}\tau + L[c_{1,1}\Delta Q_{\max} + c_{1,2} h_{\max}] \cdot (t - t_0) \\
&\leq \int_{t_0}^{t} 2L \cdot \|\mathbf{x}(\tau) - \mathbf{x_a}(\tau)\| \mathrm{d}\tau + L[c_{1,1} + c_{1,2}] \cdot \delta \cdot (t - t_0)
\end{aligned} \tag{A.15}$$

for all $t \in [t_0, t_b + T]$.

Proceeding in a similar way and subtracting now Eq.(4b) from Eq.(A.7b), we obtain

$$\begin{aligned}\dot{\mathbf{x}}_{\mathbf{2}}(t) - \dot{\mathbf{x}}_{\mathbf{a},\mathbf{2}}(t) =& \mathbf{f}_{\mathbf{2}}(\mathbf{x}_{\mathbf{1}}(t) + \boldsymbol{\Delta}_{\mathbf{2},\mathbf{1}}(t), \mathbf{x}_{\mathbf{2}}(t) + \boldsymbol{\Delta}_{\mathbf{2},\mathbf{2}}(t), t) + h_k \cdot \mathbf{r}_{\mathbf{h}}(t_k) \\ &- \mathbf{f}_{\mathbf{2}}(\mathbf{x}_{\mathbf{a},\mathbf{1}}(t), \mathbf{x}_{\mathbf{a},\mathbf{2}}(t), t)\end{aligned}$$

and,

$$\begin{aligned}\|\mathbf{x}_{\mathbf{2}}(t) - \mathbf{x}_{\mathbf{a},\mathbf{2}}(t)\| \leq& \int_{t_0}^{t} L \cdot [\|\mathbf{x}_{\mathbf{1}}(\tau) + \boldsymbol{\Delta}_{\mathbf{2},\mathbf{1}}(\tau) - \mathbf{x}_{\mathbf{a},\mathbf{1}}(\tau)\| + \|\mathbf{x}_{\mathbf{2}}(\tau) + \boldsymbol{\Delta}_{\mathbf{2},\mathbf{2}}(\tau) - \mathbf{x}_{\mathbf{a},\mathbf{2}}(\tau)\|]\mathrm{d}\tau + \\ &+ h_{\max} \cdot r_{\max}(t - t_0) \\ \leq& \int_{t_0}^{t} 2L \cdot \|\mathbf{x}(\tau) - \mathbf{x}_{\mathbf{a}}(\tau)\|\mathrm{d}\tau + L[c_{2,1} \max(\Delta Q_{\max}, h_{\max}) + c_{2,2} h_{\max}] \cdot (t - t_0) \\ &+ h_{\max} \cdot r_{\max} \cdot (t - t_0) \\ \leq& \int_{t_0}^{t} 2L \cdot \|\mathbf{x}(\tau) - \mathbf{x}_{\mathbf{a}}(\tau)\|\mathrm{d}\tau + [c_{2,1} + c_{2,2} + r_{\max}] \cdot \delta(t - t_0) \qquad \text{(A.16)}\end{aligned}$$

Then, from Eqs.(A.15) and (A.16), we obtain

$$\|\mathbf{x}(t) - \mathbf{x}_{\mathbf{a}}(t)\| \leq \int_{t_0}^{t} 4L \cdot \|\mathbf{x}(\tau) - \mathbf{x}_{\mathbf{a}}(\tau)\|\mathrm{d}\tau + L[c_{1,1} + c_{1,2} + c_{2,1} + c_{2,2} + r_{\max}] \cdot \delta(t - t_0)$$

Applying Grönwall–Bellman's inequality on the last expression, we obtain

$$\|\mathbf{x}(t) - \mathbf{x}_{\mathbf{a}}(t)\| \leq L[c_{1,1} + c_{1,2} + c_{2,1} + c_{2,2} + r_{\max}] \cdot \delta(t - t_0) \cdot e^{4L(t-t_0)}$$

That way, taking

$$\delta < \frac{\min(\varepsilon, d/4)}{L[c_{1,1} + c_{1,2} + c_{2,1} + c_{2,2} + r_{\max}] \cdot (t_f - t_0) \cdot e^{4L(t_f - t_0)}}$$

it results that

$$\|\mathbf{x}(t) - \mathbf{x}_{\mathbf{a}}(t)\| < \min(\varepsilon, d/4) \leq \varepsilon \qquad \text{(A.17)}$$

for all $t \in [t_0, t_b + T]$.

Since the condition of Eq.(A.14) is verified with $t_b = t_0$, and taking into account that it implies Eq.(A.17), this reasoning can be recursively applied with $t_b = t_0 + T$, $t_b = t_0 + 2T$, etc.. When $t_0 + m \cdot T > t_f - T$ we can take $t_b = t_f - T$ and apply for the last time the procedure concluding that Eq.(A.17) is verified for all $t \in [t_0, t_f]$.

That way, given $\varepsilon > 0$ we can find $\delta > 0$ such that $\Delta Q_{\max} \leq \delta$ and $h_{\max} \leq \delta$ imply that Eq.(A.17) is verified. This implies (17) concluding the proof. □