Applications

Simulation

Simulation: Transactions of the Society for Modeling and Simulation International
2016, Vol. 92(1) 47–61

DOI: 10.1177/0037549715616683
sim.sagepub.com
SAGE SCS

Time discretization versus state quantization in the simulation of a one-dimensional advection–diffusion–reaction equation

Federico Bergero, Joaquín Fernández, Ernesto Kofman and Margarita Portapila

Abstract

In this article, we study the effects of replacing the time discretization by the quantization of the state variables on a one-dimensional (1D) advection–diffusion–reaction (ADR) problem. For that purpose the 1D ADR equation is first discretized in space using a regular grid, to obtain a set of time-dependent ordinary differential equations (ODEs). Then we compare the simulation performance using classic discrete time algorithms and using quantized state systems (QSS) methods. The performance analysis is done for different sets of diffusion and reaction parameters and also changing the space discretization refinement. This analysis shows that, in advection–reaction-dominated situations, the second-order linearly implicit QSS method outperforms all of the conventional algorithms (DOPRI, Radau and DASSL) by more than one order of magnitude.

Keywords

advection–diffusion–reaction equation, quantization-based integration methods, numerical simulation

1. Introduction

Advection–diffusion equations provide the basis for describing heat and mass transfer phenomena as well as processes of continuum mechanics, where the physical quantity of interest $u(x, t)$ could be temperature in heat conduction or concentration of some chemical substance. In several applications these phenomena occur in presence of chemical reactions, leading to the advection–diffusion–reaction (ADR) equation, a problem frequently found in many areas of environmental sciences as well as in mechanical engineering. The ADR problem includes a wide range of configurations encompassing variable velocity fields, variable reaction coefficients, steady and transient problems, in one, two and three dimensions.[1–4]

The ADR equation poses several challenges to numerical integration algorithms. First, as in most partial differential equations (PDEs), the space discretization usually leads to large systems of equations which require an efficient treatment. Also, when the diffusivity is small in comparison with the advection field and the reaction coefficient (i.e. when the Péclet and Damköhler numbers are high) the problem often develops sharp fronts that are nearly shocks where numerical solutions are difficult to obtain. In addition, chemical reactions take place on very small time scales compared with the long-term effects considered for the advection–diffusion transport. For stability reasons, the presence of fast and slow dynamics (called *stiffness* in the numerical ordinary differential equation (ODE) literature) enforces the usage of implicit numerical integration algorithms. These algorithms have a high computational cost, particularly when the system dimension is large.

In all cases, obtaining numerical solutions of PDEs such as the ADR problem involves discretization in space and time. In some techniques such as the method of lines (MOL),[5,6] this discretization is only performed in space, transforming the PDE into a large set of ODEs. The resulting time dependent set of ODEs can then be solved with numerical integration algorithms such as Euler's or Runge–Kutta's methods,[6,7] or through algebraic differential equation solvers such as DASSL[8,9] among others.

Laboratorio de Sistemas Dinámicos, FCEIA - UNR, CIFASIS–CONICET, Argentina

Corresponding author:
Federico Bergero, Laboratorio de Sistemas Dinámicos, FCEIA - UNR, CIFASIS–CONICET, 27 de febrero 210 bis - (S2000EZP) Rosario, Argentina.
Email: bergero@cifasis-conicet.gov.ar

An alternative way to solve the resulting set of ODEs is given by the quantized state systems (QSS) methods,[6,10] that replace the time discretization by state quantization. These algorithms are characterized by performing only local steps when and where changes occur. In consequence, QSS methods are efficient when dealing with large sparse systems where only some parts of the system experience changes at a given time, a very common situation in ADR problems. Taking also into account that linearly implicit QSS (LIQSS) methods[11] are able to tackle certain stiff systems, these algorithms appear as promising candidates for integrating the ODEs resulting from the space discretization of ADR equations.

In this article, we provide a first analysis regarding the usage of QSS methods in ADR problems by comparing the performance of LIQSS methods against classic time discretization algorithms (DASSL, DOPRI and RADAU) in the simulation of a one-dimensional (1D) ADR problem previously discretized in space by the MOL. The comparison is performed under different parameter and grid refinement settings, showing that in advection–reaction-dominated ADR problems, LIQSS methods are more than 10 times faster than discrete time algorithms. We also briefly analyze the extension of these results to a 2D ADR equation.

The article is organized as follows. Section 2 introduces the main concepts used in the rest of the paper and describes some related work in the field. Section 3 discusses the implementation of the model in a QSS solver, studying also the error bounds of the approximation from a theoretical perspective. Section 4 shows numerical results of the performance of LIQSS methods in advection–diffusion–reaction models, comparing these results against classical integration methods. Finally, Section 5 presents the article conclusions and discusses how the state quantization can be extended to more general ADR problems in two and three dimensions.

2. Background

2.1. Motivating example

Consider the following ODEs

$$\begin{aligned} \dot{u}_1(t) &= 3 - u_1(t) \\ \dot{u}_2(t) &= u_1(t) - u_2(t) \\ \dot{u}_3(t) &= u_2(t) - u_3(t) \end{aligned} \tag{1}$$

with initial conditions: $u_1(0) = 3$, $u_2(0) = u_3(0) = 0$. Equations (1), that can be solved analytically, may represent a rough MOL approximation of the pure advection equation

$$\frac{\partial u(x,t)}{\partial t} = -a\frac{\partial u(x,t)}{\partial x}$$

for given parameters and boundary conditions.

Instead of solving (1) using a classic time discretization approach, we shall modify it substituting $u_i(t)$ by its integer part $q_i(t) \stackrel{\Delta}{=} \text{floor}[u_i(t)]$ at the right-hand side of each equation:

$$\begin{aligned} \dot{u}_1(t) &= 3 - \text{floor}[u_1(t)] = 3 - q_1(t) \\ \dot{u}_2(t) &= \text{floor}[u_1(t)] - \text{floor}[u_2(t)] = q_1(t) - q_2(t) \\ \dot{u}_3(t) &= \text{floor}[u_2(t)] - \text{floor}[u_3(t)] = q_2(t) - q_3(t) \end{aligned} \tag{2}$$

Let us solve this last set of equations.

- At time $t_0 = 0$ we have $q_1(t_0) = 3$, $q_2(t_0) = q_3(t_0) = 0$.
  - Initially, according to (2), we have $\dot{u}_1(t_0) = \dot{u}_3(t_0) = 0$ and $\dot{u}_2(t_0) = 3$. These derivatives will remain unchanged until some $u_i(t)$ changes its integer part.
  - Since $\dot{u}_1(t_0) = \dot{u}_3(t_0) = 0$, neither q_1 nor q_3 will change now.
  - The next change in $q_2(t)$ occurs when $u_2(t) = 1$. Since $u_2(t_0) = 0$ and its derivative is $\dot{u}_2(t_0) = 3$, it will reach the value 1 at time $t_1 = 1/3$.
- At time $t_1 = 1/3$ the result is $q_2(t_1) = u_2(t_1) = 1$.
  - According to (2) the result is $\dot{u}_2(t_1) = 2$ and $\dot{u}_3(t_1) = 1$.
  - The next change in $q_2(t)$ occurs at time $t_2 = t_1 + 1/2$ while the following change in q_3 would occur at time $t_1 + 1/1$.
- At time $t_2 = t_1 + 1/2 = 5/6$ the result is $q_2(t_2) = u_2(t_2) = 2$, while $u_3(t_2) = u_3(t_1) + (t_2 - t_1)\,\dot{u}_3(t_1) = 1/2$.
  - According to (2) the derivatives are now $\dot{u}_2(t_2) = 1$ and $\dot{u}_3(t_2) = 2$.
  - Then, the upcoming change in $q_2(t)$ would occur at time $t_2 + 1$ while the next change in q_3 should be recomputed to occur at time $t_3 = t_2 + 0.5/2$.
- At time $t_3 = t_2 + 1/4 = 13/12$ the result is $q_3(t_3) = u_3(t_3) = 1$.
  - According to (2) we have now $\dot{u}_3(t_3) = 1$.
  - Then, the subsequent change in $q_3(t)$ would occur at time $t_3 + 1$.
- At time $t_4 = t_2 + 1 = 11/6$ we have $q_2(t_4) = u_2(t_4) = 3$ and $u_3(t_4) = u_3(t_3) + (t_4 - t_3)\dot{u}_3(t_3) = 7/4$.
  - According to (2) the derivatives are now $\dot{u}_2(t_2) = 0$ and $\dot{u}_3(t_2) = 2$.
  - Then, $q_2(t)$ will not change again and the next change in $q_3(t)$ can be recomputed to occur at time $t_5 = t_4 + 0.25/2$.
- At time $t_5 = t_4 + 1/8 = 47/24$ we have $q_3(t_5) = u_3(t_5) = 2$.
  - According to (2) the derivative is now $\dot{u}_3(t_2) = 1$.

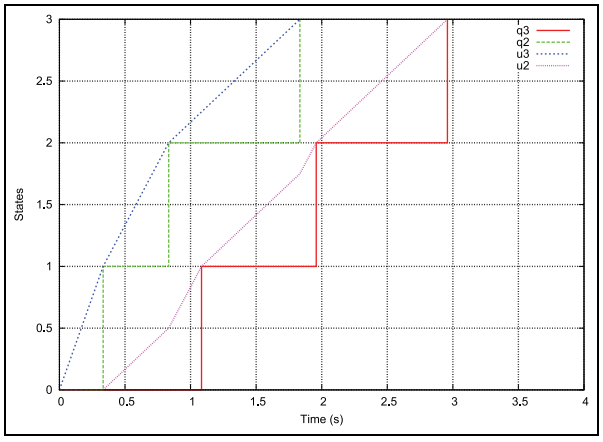

Figure 1. Solution of Equation (2).

  - Then, the next change in q_3 occurs at time $t_6 = t_5 + 1$.
- At time $t_6 = t_5 + 1 = 71/24$ we have $q_3(t_6) = u_3(t_6)$.
  - All of the derivatives are equal to zero and no further changes occur after t_6.

The trajectories of this solution are depicted in Figure 1. Variables $u_1(t)$ and $q_1(t)$, that remain unchanged for all t, are not drawn.

This example shows that replacing a variable $u_i(t)$ by its integer part $\text{floor}[u_i(t)]$ at the right-hand side of an ODE seems to provide a way to integrate the equation. Note that under this principle, we are replacing the time discretization by the quantization of the system states. This is indeed the basic idea behind the family of QSS methods.

The following remarks must be taken into account in connection with the procedure followed above.

- After the startup, the simulation took a total of six steps.
- Each step was local, related to a change in the integer part of a state: In t_1, t_2 and t_4 the change occurred in $q_2(t)$ while in t_3, t_5 and t_6 the change occurred in $q_3(t)$. As $q_1(t)$ was already at equilibrium, it never changed.
- Changes in $q_2(t)$ prompted the evaluation of $\dot{u}_2$ and $\dot{u}_3$. Changes in $q_3(t)$ provoked that only $\dot{u}_3$ was evaluated. Thus, after the startup, $\dot{u}_1$ was never computed, $\dot{u}_2$ was evaluated three times and $\dot{u}_3$ was evaluated six times.
- The previous analysis shows that computations are only performed where and when changes occur, which leads to a very efficient sparsity exploitation.
- The results plotted in Figure 1 show very coarse steps, with jumps of 1 unit between successive values of each state. More accurate results can be obtained replacing the *quantization function* $\text{floor}[u_i(t)]$ by $\Delta Q \cdot \text{floor}[u_i(t)/\Delta Q]$. The parameter ΔQ is called *quantum*.
- If the first line of (1) is replaced by $\dot{u}_1(t) = 2.5 - u_1(t)$, then the first line of (2) becomes $\dot{u}_1(t) = 2.5 - q_1(t)$. In this case, the procedure fails. Initially we have $q_1(0) = 3$ and then $\dot{u}_1(0) = -0.5$. Thus, immediately we have $u_1(0^+) < 3 \Rightarrow q_1(0^+) = 2$ and then $\dot{u}_1(0^+) = +0.5$. Therefore, we are back to the initial situation $u_1(0^{++}) = 3$. This cyclic behavior provokes an infinitely fast oscillation and the simulation cannot advance beyond the initial time.

This drawback is solved with the usage of *hysteresis* in the quantization function, which leads to the definition of the QSS algorithm.

2.2. QSS methods

QSS methods are inspired in the ideas explained above, replacing the time discretization of classic numerical integration algorithms by the quantization of the state variables.

Given the ODE

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

the first-order QSS method (QSS1)[10] approximates it by

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{q}(t), t) \tag{4}$$

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

$$q_j(t) = \begin{cases} x_j(t) & \text{if} |x_j(t) - q_j(t^-)| \geqslant \Delta Q_j \\ q_j(t^-) & \text{otherwise} \end{cases} \tag{5}$$

where ΔQ_j is called *quantum* and $q_j(t^-)$ denotes the left-sided limit of q_j at time t.

Equation (5) says that the quantized state $q_j(t)$ only changes when its difference with the state $x_j(t)$ becomes equal to the quantum ΔQ_j. When this condition is reached, the quantized state starts a new segment with the value of the state, i.e. $q_j(t) = x_j(t)$.

Since the quantized state trajectories $q_j(t)$ are piecewise constant then, the state derivatives $\dot{x}_j(t)$ also follow piecewise constant trajectories and, consequently, the states $x_j(t)$ follow piecewise linear trajectories. Figure 2 shows typical QSS1 trajectories.

Due to the particular form of the trajectories, the analytical solution of (4) is straightforward and can be obtained following the ideas used to solve (2). These ideas can be generalized by the following procedure.

For $j = 1, \ldots, n$, let t_j denote the next time at which $|q_j - x_j| = \Delta Q_j$. Then we use the following procedure.

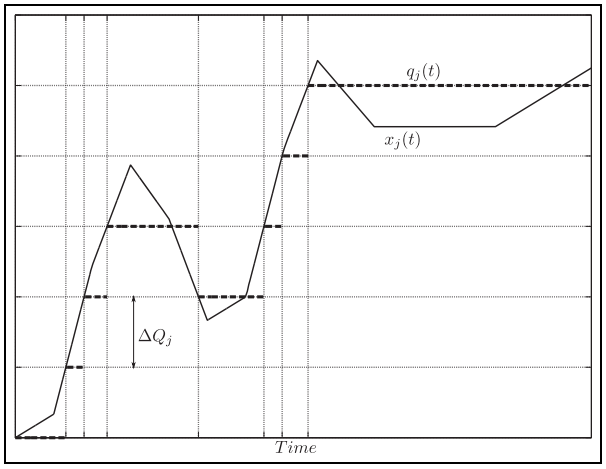

Figure 2. State and quantized trajectories in the QSS1 method.

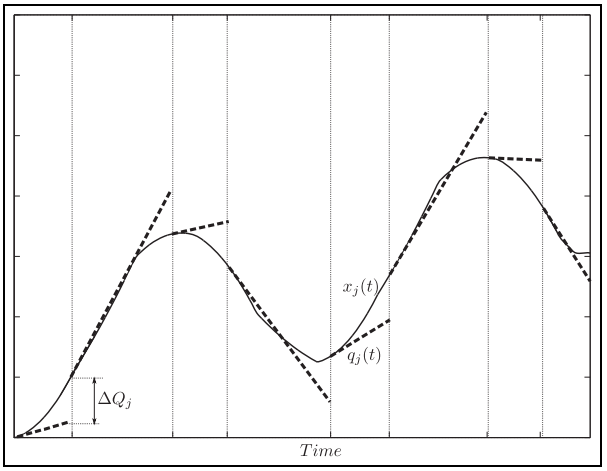

Figure 3. State and quantized trajectories in the QSS2 method.

1. Advance the simulation time t to the minimum t_j.
2. Recompute $x_j(t) = x_j(t_j^-) + \dot{x}_j(t_j^-) \cdot (t - t_j^-)$, where t_j^- was the last update time of x_j and $\dot{x}_j(t_j^-)$ was computed at time t_j^- from (4).
3. Take $q_j = x_j$ and recompute t_j (the next time at which $|q_j - x_j| = \Delta Q_j$).
4. For all i such that $\dot{x}_i$ explicitly depends on q_j, update $x_i(t) = x_i(t_i^-) + \dot{x}_i(t_i^-) \cdot (t - t_i^-)$, recompute $\dot{x}_i(t)$ and recalculate t_i (the next time at which $|q_i - x_i| = \Delta Q_i$).
5. Go back to step 1.

The QSS1 method has the following features.

- The difference between the state and quantized variables is never greater than the quantum ΔQ_j. This fact ensures stability and global error bound properties.[6,10] In stable linear systems, the global simulation error results linearly bounded by the quantum.
- The quantum ΔQ_j of each state variable can be chosen to be proportional to the state magnitude, leading to an intrinsic relative error control.[12]
- Each step is local to a single state variable x_j (the one which reaches the quantum change), and it only provokes evaluations of the state derivatives that explicitly depend on it. This fact implies that QSS1 performs intrinsic sparsity exploitation.
- If some state variables do not change significantly, they will not provoke any step or evaluation at all. This feature reinforces the efficient sparsity exploitation.
- The fact that the state variables follow piecewise linear trajectories makes very easy to detect discontinuities. Moreover, after a discontinuity is detected, its effects are not different to those of a normal step (because changes in q_j are discontinuous). Thus, QSS1 is very efficient to simulate discontinuous systems.[13]

The main limitations of QSS1 are as follows.

- It only performs a first-order approximation, and good accuracy cannot be obtained without a significant increment in the number of steps.
- It is not suitable to simulate stiff systems.

The first limitation was solved with the introduction of higher-order QSS methods such as QSS2[14] and QSS3.[15]

QSS2 has the same definition as (4) except that the quantization function of (5) is replaced by a different one, such that the quantized state variables $q_j(t)$ follow piecewise linear trajectories and the state variables $x_j(t)$ follow piecewise parabolic trajectories as shown in Figure 3. In that way, the algorithm performs larger steps preserving the difference between the state $x_j(t)$ and the quantized state $q_j(t)$ bounded by the quantum ΔQ_j.

QSS2 has the same theoretical properties and practical advantages of QSS1.

QSS3 is based on the same principles but with piecewise parabolic and piecewise cubic trajectories.

Regarding stiff systems, a first-order backward QSS method (BQSS) was introduced by Migoni et al.[16] This method, in spite of being backward, was explicit due to the following property. In QSS the next state value is always known as it should be either $q_j + \Delta Q_j$ or $q_j - \Delta Q_j$, according to the sign of $\dot{x}_j$. The unknown, that can be computed explicitly, is the instant of time at which the state reaches the next quantized value.

Unfortunately, BQSS cannot be extended to higher-order approximations. However, a family of LIQSS methods up to third order was proposed by Migoni et al.[11] Even when the formulation of LIQSS methods is implicit, their

implementations are explicit thanks to the same property explained above for the case of BQSS algorithm.

LIQSS methods share the advantages of QSS methods and, additionally, they are able to efficiently handle stiff systems, provided that the stiffness is due to the presence of large entries in the main diagonal of the system Jacobian matrix. Otherwise, when the stiffness is due to other reasons (the structure of semi-discretized diffusion problems,[6] for instance) LIQSS methods may provoke spurious oscillations and the efficiency is lost.

As a consequence, for sparse, discontinuous systems or those exhibiting the type of stiffness that is properly handled by LIQSS algorithms, the usage of quantized state solvers can offer a better performance than that of classic discrete time methods. Otherwise, the use of appropriate classical methods may be the best choice.

In the context of this work, the intrinsic sparsity exploitation and the explicit treatment of stiffness will provide the main advantages of LIQSS algorithms. Anyway, these advantages will disappear in the presence of large diffusion terms where the resulting stiffness cannot be efficiently handled by these methods.

2.3. Implementation of QSS methods

It was shown that the behavior of the QSS approximation of (4) can be described in terms of the discrete event system specification (DEVS) formalism.[17] Based on this property, the whole family of QSS methods was first implemented in PowerDEVS,[18] a DEVS-based simulation platform designed for simulating hybrid systems. In addition, the explicit QSS methods of orders 1–3 were also implemented in a DEVS library of Modelica[19] and implementations of the first-order QSS methods can also be found in CD++ [20] and VLE.[21]

DEVS-based implementations of QSS methods are simple but inefficient. DEVS simulation engines waste a large amount of computational effort passing messages and scheduling events that are not strictly necessary for the QSS algorithms. This fact motivated the development of stand-alone QSS solvers.

A first approach to a stand-alone version of QSS1 to QSS3 was implemented in the Java-based simulation tool *Open Source Physics*,[22] but that implementation was not more efficient than that of PowerDEVS and it required the user to provide the system structure information needed by QSS methods.

Recently, the complete family of QSS methods was implemented in a *stand-alone QSS solver* coded in plain C language.[23] This solver improves PowerDEVS simulation times by more than one order of magnitude, allowing the simulation of models described in a subset of the Modelica language[24] called μ-Modelica.

This is the tool we shall use in the rest of this article.

2.4. Related work

The goal of this article is to study the efficiency of QSS methods in the simulation of the ADR PDE semi-discretized using the MOL.

To the best of the authors' knowledge, this problem was never studied. However, there are several works that study the same PDE problem in the context of classic numerical integration algorithms, and there are some works that study the use of QSS methods in the simulation of other types of PDEs.

The combination of the MOL with classic numerical algorithms for the ADR PDE has been analyzed in several articles.[25–30]

In all of these works, the goal was to overcome the problem imposed by the stiffness associated to the reaction term, using variants of Runge–Kutta algorithms.

Savcenco et al.[31] study the use of multi-rate algorithms for stiff ODE problems, including a case resulting from the semi-discretization of an advection–reaction PDE. Multi-rate algorithms are somehow related to quantization-based integration methods in the sense that both use different time scales for different state variables.

The use of QSS methods in PDEs has not been yet studied in depth. Muzy et al.[32] showed the results of using QSS methods for a 1D diffusion problem. Hyperbolic PDEs representing lossless transmission lines were also simulated in the context of QSS methods in Migoni et al.,[14,16] including also a stiff load.

3. QSS approximation of the ADR model

In this section, we first introduce the 1D ADR model used along the work and its discretization with the MOL. We then perform a theoretical analysis to obtain an upper bound for the error introduced by the QSS approximation of the resulting ODE. Finally, we describe the implementation of this ODE in the QSS solver.

3.1. The ADR equation

Let $u(x,t)$ be the concentration of some species in the space coordinate x at time t. Then, the 1D advection and diffusion[33] process can be described by the following PDE:

$$\frac{\partial u(x,t)}{\partial t} + a\frac{\partial u(x,t)}{\partial x} = d\frac{\partial^2 u(x,t)}{\partial x^2} \tag{6}$$

Taking into account that the species undergoes a chemical reaction, we include a non-linear reaction term following Zeldovich's equation[34] as follows:

$$\frac{\partial u(x,t)}{\partial t} + a\frac{\partial u(x,t)}{\partial x} = d\frac{\partial^2 u(x,t)}{\partial x^2} + r(u(x,t)^2 - u(x,t)^3) \tag{7}$$

This is the model we shall work with in the rest of the article. Here a, d and r are parameters expressing the advection, diffusion and reaction coefficients, respectively.

We shall consider that the space domain is limited to the interval $0 \leqslant x \leqslant 10$ and that the boundary conditions are

$$u(x=0,t)=1; \quad \frac{\partial u(x=10,t)}{\partial x}=0 \tag{8}$$

For the simulations, we shall work with the following initial conditions:

$$u(x,t=0)=\begin{cases}1 & \text{if } x<2\\ 0 & \text{otherwise}\end{cases} \tag{9}$$

3.2. MOL discretization of the ADR model

In order to discretize the problem with the MOL, we shall use a regular grid of width

$$\Delta x=\frac{10}{N} \tag{10}$$

where N is the number of grid points.

The advection term of (7) $\frac{\partial u(x,t)}{\partial x}$ shall be replaced by a first-order upwind finite difference:

$$\frac{\partial u}{\partial x}(x=x_i,t)\approx\frac{u_i-u_{i-1}}{\Delta x} \tag{11}$$

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

$$u_i(t)\approx u(x_i,t) \tag{12}$$

is the ith state variable of the resulting ODE and

$$x_i=i\cdot\Delta x \tag{13}$$

is the ith grid point.

Taking into account the boundary condition of (8) at $x=0$, we also have $u_0=1$.

We shall discretize the diffusion term replacing the expression $\frac{\partial^2 u}{\partial x^2}$ by a second-order centered finite difference:

$$\frac{\partial^2 u}{\partial x^2}(x=x_i,t)\approx\frac{u_{i+1}-2u_i+u_{i-1}}{\Delta x^2} \tag{14}$$

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

For the last grid point, taking into account the symmetrical border condition of (8) at $x=10$, we can replace

$$\frac{\partial^2 u}{\partial x^2}(x=x_N,t)\approx\frac{u_{N-1}-2u_N+u_{N-1}}{\Delta x^2} \tag{15}$$

Replacing (11)–(15) into (7) we obtain the following set of ODEs:

$$\dot{u}_i=-a\frac{(u_i-u_{i-1})}{\Delta x}+d\frac{(u_{i+1}-2u_i+u_{i-1})}{\Delta x^2}+r(u_i^2-u_i^3) \tag{16}$$

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

$$\dot{u}_N=-a\frac{(u_N-u_{N-1})}{\Delta x}+d\frac{(2u_{N-1}-2u_N)}{\Delta x^2}+r(u_N^2-u_N^3) \tag{17}$$

3.3. Model structure

The Jacobian matrix of the system of (16) can be computed as

$$J=\begin{bmatrix}J_{11} & J_{12} & 0 & 0 & \cdots & 0\\ J_{21} & J_{11} & J_{12} & 0 & \cdots & 0\\ 0 & J_{21} & J_{11} & J_{12} & \cdots & 0\\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots\end{bmatrix} \tag{18}$$

where

$$\begin{aligned}J_{11}&=\frac{-a}{\Delta x}+\frac{-2d}{\Delta x^2}+r(2u_1-3u_1^2);\\ J_{12}&=\frac{d}{\Delta x^2}; \quad J_{21}=\frac{a}{\Delta x}+\frac{d}{\Delta x^2}\end{aligned} \tag{19}$$

Note that J is tridiagonal.

As was shown by Migoni et al.,[35] LIQSS methods efficiently handle stiffness due to large entries in the main diagonal. Thus, we expect that the stiffness due to the reaction term, which only appears in J_{11}, is efficiently handled. However, the stiffness due to the diffusion term may cause problems. Moreover, it is known that the stiffness ratio of the resulting ODE grows quadratically with the number of segments,[6] so those problems may become more important as the grid is refined.

3.4. Global error bounds of the QSS simulation of the ADR model

The global error bound properties of QSS methods[6,14] establish that the simulation with these algorithms of stable linear time invariant (LTI) systems gives numerical solutions that differ from the analytical solution in a quantity that is linearly bounded with the quantum.

In the absence of a reaction term, the system of (16)–(17) is LTI. However, for the pure advection problem, the analysis cited before[6,14] cannot be applied because the system cannot be diagonalized.

Thus, we analyze here the pure advection case in order to establish a theoretical upper bound for the error introduced by the QSS approximation of (16)–(17).

Defining $\mathbf{u}\stackrel{\Delta}{=}[u_1,u_2,\ldots,u_N]^{\mathrm{T}}$, the pure advection model can be written as

$$\dot{\mathbf{u}}(t) = A \cdot \mathbf{u}(t) + B \cdot u_0(t) \tag{20}$$

with

$$A = \frac{a}{\Delta x} \cdot \begin{bmatrix} -1 & 0 & 0 & \dots & 0 \\ 1 & -1 & 0 & \dots & 0 \\ 0 & 1 & -1 & \dots & 0 \\ \vdots & & & & \\ 0 & \dots & \dots & 1 & -1 \end{bmatrix}; \quad B = \frac{a}{\Delta x} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{21}$$

Any QSS or LIQSS method transforms (20) into

$$\dot{\mathbf{v}}(t) = A \cdot \mathbf{q}(t) + B \cdot u_0(t) \tag{22}$$

where $\mathbf{v}(t)$ is the numerical solution and $\mathbf{q}(t)$ is the quantized version of the state $\mathbf{v}(t)$.

Taking into account that differences between the components $q_i(t)$ and $v_i(t)$ cannot be larger than the quantum ΔQ, we can write

$$q_i(t) = v_i(t) + \Delta v_i(t) \tag{23}$$

with $|\Delta v_i(t)| < \Delta Q$. Then, we can rewrite the ith component of (22) as

$$\dot{v}_i(t) = -\frac{a}{\Delta x}(v_i(t) + \Delta v_i) + \frac{a}{\Delta x}(v_{i-1}(t) + \Delta v_{i-1}) \tag{24}$$

while the ith component of the original system of (20) is

$$\dot{u}_i(t) = -\frac{a}{\Delta x}u_i(t) + \frac{a}{\Delta x}u_{i-1}(t) \tag{25}$$

Defining the error $e_i(t) \stackrel{\Delta}{=} v_i(t) - u_i(t)$ and subtracting (25) from (24) we obtain the error dynamics as

$$\dot{e}_i(t) = -\frac{a}{\Delta x}(e_i(t) + \Delta v_i) + \frac{a}{\Delta x}(e_{i-1}(t) + \Delta v_{i-1}) \tag{26}$$

Taking into account that we have not quantified the boundary condition (i.e. $u_0 = v_0 = q_0$), then the dynamics of the first component of the error is

$$\dot{e}_1(t) = -\frac{a}{\Delta x} \cdot (e_1(t) + \Delta v_1(t)) \tag{27}$$

Note that if at certain time t_k this error is positive and reaches the quantum, i.e. $e_1(t_k) = \Delta Q$, recalling that $|\Delta v_1(t)| \leqslant \Delta Q$ it results that $e_1(t_k) \geqslant |\Delta v_1(t_k)|$ and therefore $e_1(t_k) + \Delta v_1(t_k) \geqslant 0$. Taking into account the negative sign in (27) the result is $\dot{e}_1(t_k) \leqslant 0$.

Similarly, if at certain time t_k this error is negative and reaches the quantum, i.e., $e_1(t : k) = -\Delta Q$, an analogous reasoning shows that $\dot{e}_1(t_k) \geqslant 0$.

In other words, whenever $e_1(t)$ reaches the value $+\Delta Q$, its derivative becomes negative or zero and whenever $e_1(t)$ reaches the value $-\Delta Q$, its derivative becomes positive or zero. Thus, if $|e_1(t_k)| \leqslant \Delta Q$, then $|e_1(t)| \leqslant \Delta Q$ for all $t > t_k$.

Taking into account that $e_1(t_0) = 0$ the result is that

$$|e_1(t)| \leqslant \Delta Q \tag{28}$$

for all $t \geqslant t_0$.

For the second component we have that

$$\dot{e}_2 = -\frac{a}{\Delta x} \cdot (e_2 + \Delta v_2 - e_1 - \Delta v_1) \tag{29}$$

where it can be easily seen that $|\Delta v_2(t) - e_1(t) - \Delta v_1(t)| \leqslant 3\Delta Q$. Hence, proceeding as before, we found that $|e_2| < 3 \cdot \Delta Q$.

Extending this analysis, we arrive to the error bound condition

$$|e_i(t)| < (2 \cdot i - 1) \cdot \Delta Q \tag{30}$$

This is a global upper bound on the error introduced by the QSS algorithms at the ith space point of the solution, which stands for any initial condition of the purely advective problem of (16)–(17).

This result establishes that the error bound grows linearly with the quantum ΔQ and with the space coordinate index $i = x_i/\Delta x$. Although it is a conservative result, its predictions will be corroborated in the following section.

The addition of a small diffusion term does not change significantly the results, leading to a more complex expression.

The presence of reaction terms leads to a more complex non-linear study that is out of the scope of this work.

3.5. The ADR model in the QSS solver

The ODE model of (16)–(17) can be described in the subset of Modelica language (μ-Modelica) used by the standalone QSS solver[23] as follows:

```
model adv_dif_reac
  constant Integer N=1000;
  parameter Real a=1;
  parameter Real d=1e-4;
  parameter Real r=10;
  parameter Real L=10;
  parameter Real dx=L/N;
  Real u[N];
initial algorithm
  for i in 1:0.2*N loop
    u[i]:=1;
  end for;
equation
  der(u[1])=-a*(u[1]-1)/dx + d*(u[2]-
  2*u[1] + 1)/(dx^2)+ r*(u[1]^2)*(1-u[1]);
  der(u[N])=-a*(u[N]-u[N-1])/dx + d*(u[N-
  1]-2*u[N] + u[N-1])/(dx^2) + r*(u[N]^2)*
  (1-u[N]);
  for i in 2:N-1 loop
```

```
    der(u[i])=-a*(u[i]-u[i-1])/dx +
    d*(u[i + 1]-2*u[i] + u[i-1])/(dx^2) +
    r*(u[i]^2)*(1-u[i]);
  end for;
end adv_dif_reac;
```

Note that in this case, we used parameters $a = 1$, $d = 10^{-4}$, $r = 10$ and performed the discretization over $N = 1000$ grid points. The solution for this parameter set, obtained with LIQSS2, is shown in Figure 4. There, $u[400]$ is the discretized version of $u(x = 4)$, $u[600]$ is the discretized version of $u(x = 6)$, and so on.

4. Results

In this section we compare the performance of different numerical integration methods on the ADR problem semi-discretized with the MOL. For that purpose, the resulting model of (16) is simulated for different parameter settings using LIQSS2, DASSL, Radau5 and DOPRI.

- DASSL results were computed using the Fortran code DASPK.[36]
- DOPRI and Radau5 results were computed using the C++ implementation available at Hairer's website `http://www.unige.ch/~hairer/software.html`, written by Blake Ashby.
- LIQSS2 results were obtained with the stand-alone QSS Solver.
- All of the simulations were performed on the same Intel i7-3770@3.40 GHz computer under a Linux operating system (Ubuntu).
- The errors in all cases are computed against reference trajectories obtained with a tight error tolerance (1×10^{-10}) using DOPRI. We consider the error on the state of the last grid point $u_N(t)$ since, as shown in Section 3.4, that point accumulates the error of all of the previous ones. The average error is computed on 5000 equidistant time points by $\sum_{i=1}^{5000} |u_{N_{ref}}(t_i) - u_{N_{sim}}(t_i)|/5000$ while the maximum error is $\max_i(\{|u_{N_{ref}}(t_i) - u_{N_{sim}}(t_i)|\})$ where $u_{N_{ref}}(t)$ is the ground truth reference and $u_{N_{sim}}(t)$ is the simulated solution.
- We did not compute consistency errors due to the MOL space discretization. We are only interested in the ODE integration error.
- In all scenarios (except for the error analysis case) we gave the numerical solver a relative tolerance of 1×10^{-3} and an absolute tolerance of 1×10^{-4}.
- The model was simulated up to $t = 10$ seconds. Before that time, the model always reaches an equilibrium condition.
- In all cases, the number of function evaluations reported corresponds to scalar components.

4.1. Error analysis

We first simulated the system of (16) with parameters $a = 1$, $r = d = 0$ (i.e. the pure advective case) using the LIQSS2 method for different quantum ΔQ and grid refinement N. The goal of this experiment was to compare the theoretical bound of (30) with the actual error introduced by the algorithm.

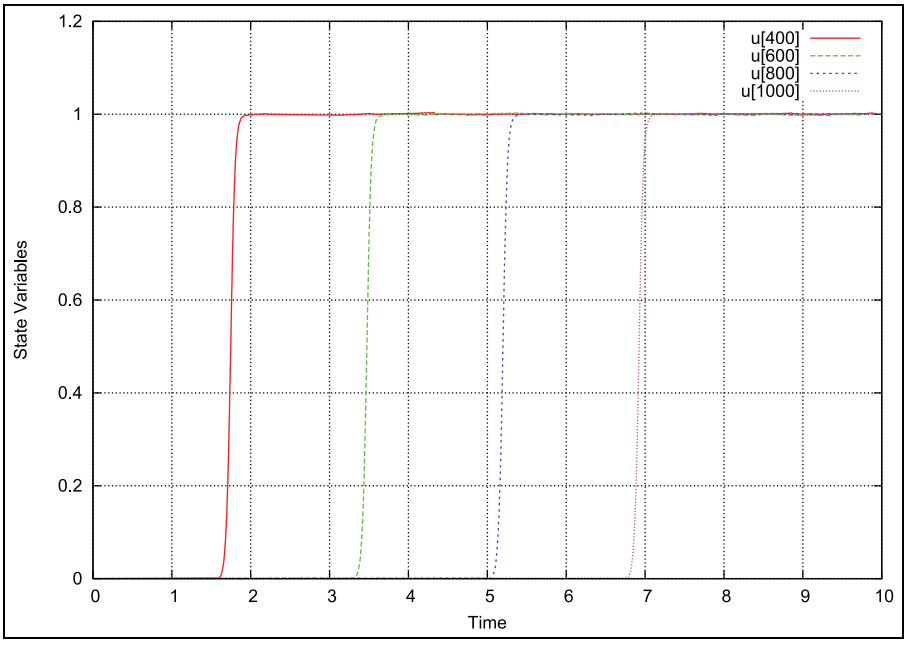

Figure 4. Simulation results for $a = 1$, $d = 1 \times 10^{-4}$, $r = 10$, $N = 1000$ using the LIQSS2 method.

Table 1. Maximum and average error for different values of ΔQ with $N = 100, 1000, 10{,}000$.

	$N = 100$		$N = 1000$		$N = 10{,}000$	
	Maximum	Average	Maximum	Average	Maximum	Average
$\Delta Q = 1 \times 10^{-3}$	8.5×10^{-3}	2.5×10^{-4}	7.7×10^{-3}	1.7×10^{-3}	2.5×10^{-2}	3.8×10^{-4}
$\Delta Q = 1 \times 10^{-4}$	9.9×10^{-4}	3.2×10^{-5}	7.1×10^{-4}	1.9×10^{-5}	1.4×10^{-3}	3.3×10^{-5}
$\Delta Q = 1 \times 10^{-5}$	1.8×10^{-4}	7.3×10^{-6}	7.3×10^{-5}	2.5×10^{-6}	2.5×10^{-4}	3.73×10^{-6}
$\Delta Q = 1 \times 10^{-6}$	5.4×10^{-5}	2.2×10^{-6}	1.6×10^{-5}	3.9×10^{-7}	3.0×10^{-5}	4.1×10^{-7}

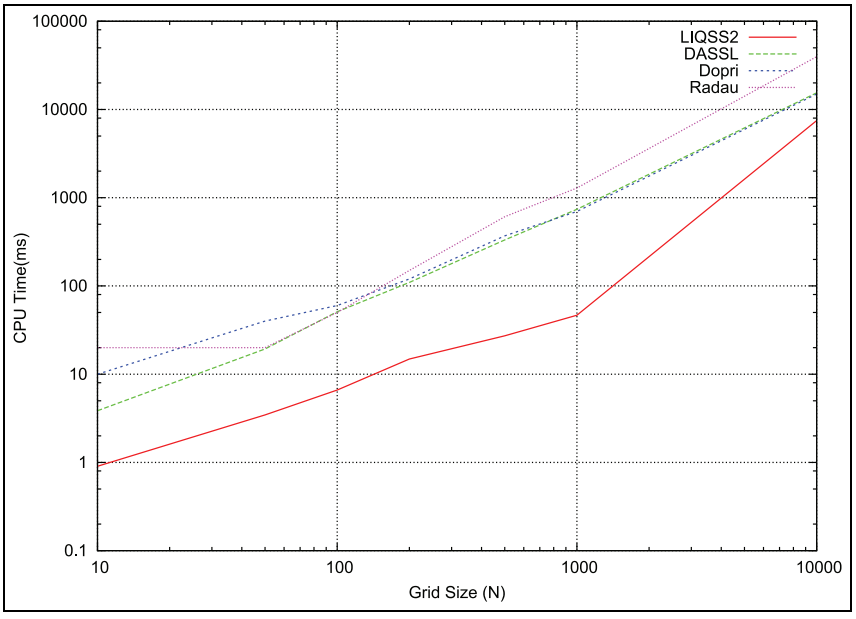

Figure 5. CPU time versus N with $a = 1$, $d = 1 \times 10^{-4}$, $r = 1000$.

The results are summarized in Table 1.

Analyzing these results, we make the following observations.

- The theoretical error bound computed by (30) holds for all cases.
- The theoretical error bound formula is very conservative. For instance, taking $N = 10,000$ and $\Delta Q = 1 \times 10^{-6}$, the theoretical error bound states that $|e_N(t)| < 1.9 \times 10^{-2}$ and Table 1 shows that the maximum error found is 3.0×10^{-5}.
- The reported errors grow linearly with the quantum in concordance with the theoretical error bound.
- While (30) establishes that the theoretical error bound grows linearly with the grid refinement N, results reported in Table 1 show that N does not seem to affect the maximum error significantly, since it remains of the same order.
- The main practical conclusion of the error analysis is that the measured error and the quantum have a similar order of magnitude.

4.2. *Variation of the grid size* Δx

In this scenario we study the computational cost and error introduced by the different algorithms for different number of points (N) in the grid. The remaining parameters are kept fixed, $a = 1$, $d = 1 \times 10^{-4}$, $r = 1000$. The resulting Péclet number is $a/d = 10,000$.

The goal of this experiment is to establish how efficient are ODE solvers at handling models resulting from more refined grids, which are used often to reduce the consistency error introduced by the MOL.

Figure 5 compares the CPU time of DASSL, DOPRI, Radau5 and LIQSS2 as N grows. Table 2 summarizes the results together with the number of scalar function evaluations.

Here LIQSS2 outperforms the other methods in all cases. Note that up to $N = 1000$, the CPU time grows sub-linearly with the size N for LIQSS2. With $N = 1000$ LIQSS2 is 15 times faster than DOPRI and DASSL, and 27 times faster than Radau.

However, at $N = 10,000$ the stiffness due to the diffusion term at (16) becomes relevant since, as was mentioned

Table 2. CPU time (ms) and number of function evaluations for different values of N with $a = 1$, $d = 1 \times 10^{-4}$, $r = 1000$.

	LIQSS2		DASSL		DOPRI		Radau5	
N	time	eval.	time	eval.	time	eval.	time	eval.
10	9.04×10^{-1}	5.99×10^{3}	3.85×10^{0}	8.47×10^{3}	1.00×10^{1}	1.88×10^{5}	2.00×10^{1}	1.02×10^{4}
50	3.45×10^{0}	2.79×10^{4}	1.93×10^{1}	1.02×10^{5}	4.00×10^{1}	1.06×10^{6}	2.00×10^{1}	1.74×10^{5}
100	6.62×10^{0}	5.38×10^{4}	5.11×10^{1}	3.29×10^{5}	6.00×10^{1}	2.45×10^{6}	5.00×10^{1}	5.02×10^{5}
200	1.48×10^{1}	1.17×10^{5}	1.10×10^{2}	8.85×10^{5}	1.20×10^{2}	5.17×10^{6}	1.50×10^{2}	1.57×10^{6}
500	2.73×10^{1}	3.16×10^{5}	3.33×10^{2}	2.61×10^{6}	3.70×10^{2}	1.73×10^{7}	6.10×10^{2}	6.06×10^{6}
1000	4.66×10^{1}	6.05×10^{5}	7.41×10^{2}	5.64×10^{6}	7.00×10^{2}	3.54×10^{7}	1.29×10^{3}	1.23×10^{7}
10,000	7.49×10^{3}	1.07×10^{8}	1.54×10^{4}	1.08×10^{8}	1.50×10^{4}	7.41×10^{8}	3.97×10^{4}	4.04×10^{8}

Table 3. Maximum and average error for different values of N with $a = 1$, $d = 1 \times 10^{-4}$, $r = 1000$.

	LIQSS2		DASSL		DOPRI		Radau5	
N	Maximum	Average	Maximum	Average	Maximum	Average	Maximum	Average
10	5.9×10^{-2}	2.8×10^{-3}	7.4×10^{-1}	7.9×10^{-4}	3.9×10^{-3}	8.7×10^{-4}	2.5×10^{-3}	2.7×10^{-6}
50	8.4×10^{-2}	8.1×10^{-4}	7.0×10^{-1}	6.8×10^{-4}	2.2×10^{-2}	1.9×10^{-3}	3.5×10^{-3}	6.7×10^{-6}
100	1.2×10^{-1}	1.7×10^{-4}	6.6×10^{-1}	6.1×10^{-4}	3.8×10^{-2}	2.5×10^{-3}	9.1×10^{-3}	2.9×10^{-5}
200	1.6×10^{-1}	1.8×10^{-3}	7.5×10^{-1}	7.6×10^{-4}	9.8×10^{-2}	3.0×10^{-3}	3.0×10^{-3}	1.3×10^{-5}
500	1.8×10^{-1}	1.1×10^{-3}	5.3×10^{-1}	4.0×10^{-4}	3.9×10^{-2}	3.8×10^{-3}	1.7×10^{-2}	1.4×10^{-5}
1000	2.1×10^{-1}	1.3×10^{-3}	3.4×10^{-2}	2.4×10^{-5}	5.8×10^{-2}	4.8×10^{-3}	4.9×10^{-2}	3.3×10^{-5}
10,000	5.9×10^{-1}	8.1×10^{-4}	1.0×10^{0}	1.4×10^{-3}	1.9×10^{-1}	6.6×10^{-3}	3.0×10^{-1}	1.3×10^{-4}

before, in diffusion problems discretized with the MOL the stiffness ratio grows quadratically with the number of grid points. We recall that this type of stiffness is not properly handled by LIQSS methods,[11] hence its performance is impoverished.

Although the presence of the reaction term makes the problem stiff, the explicit algorithm DOPRI is still able to simulate it in a reasonable time. In fact, it performs several function evaluations, but its low cost per step results in a similar performance to that of DASSL.

It must be mentioned that DASPK and Radau5 codes are suitable for large-scale models. Moreover, they exploit the knowledge of the tridiagonal structure of the Jacobian matrix for this particular case. Otherwise, their computational cost would grow cubically with N.

Table 3 shows the maximum and mean absolute errors obtained by the tested algorithms.

The average errors of LIQSS2, DASSL and DOPRI are similar, and they are consistent with the tolerance settings. Radau, however, is about two orders of magnitude more accurate. This is because the implementation is extremely conservative regarding the error tolerance.

The maximum absolute error is high for all algorithms (except for Radau). The reason is that the solution is a traveling wave with a large slope. Figure 4 illustrates the solution for $r = 10$. For $r = 1000$ the solution looks like a traveling step. Thus, a very small error in the wave speed causes a very large error in the value of u_i when the wave passes through the ith point of the grid.

4.3. Variation of the grid size Δx without diffusion

In this scenario we study the computational cost for different number of points N in the grid without diffusion term ($d = 0$), i.e. a purely advective–reactive problem. The remaining parameters were fixed as: $a = 1$, $r = 1000$. Errors are not reported as they are similar to those of the previous scenario.

Figure 6 compares the CPU time of DASSL, Radau5, DOPRI and LIQSS2. Table 4 summarizes the results together with the number of scalar function evaluations.

The results here are similar to those with $d = 1 \times 10^{-4}$, except that now LIQSS2 does not experience any problem as N grows. The absence of diffusion confines the stiffness to the main diagonal of the Jacobian matrix, a case that LIQSS2 efficiently handles.

Consequently, when $N = 10,000$, LIQSS2 is about 30 times faster than DOPRI, 38 times faster than DASSL and 98 times faster than Radau.

4.4. Variation of reaction term r

Now we consider the variation of r with the remaining parameters fixed at $a = 1$, $d = 1 \times 10^{-4}$, $N = 1000$.

Figure 7 compares the CPU time of DASSL, Radau5, DOPRI and LIQSS2 as r grows. Table 5 summarizes the results together with the number of scalar function evaluations. Errors are not reported as they are similar to the previous ones.

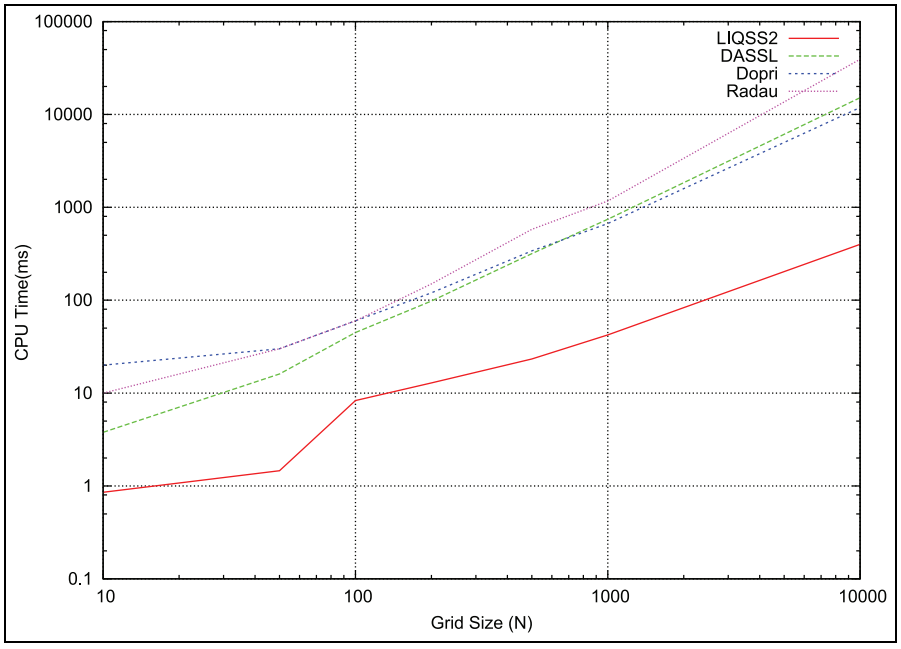

Figure 6. CPU time versus N with $a = 1$, $d = 0$, $r = 1000$.

Table 4. CPU time and number of function evaluations for different values of N with $a = 1$, $d = 0$, $r = 1000$.

	LIQSS2		DASSL		DOPRI		Radau5	
N	time	eval.	time	eval.	time	eval.	time	eval.
10	8.54×10^{-1}	6.14×10^{3}	3.78×10^{0}	8.47×10^{3}	2.00×10^{1}	1.88×10^{5}	1.00×10^{1}	1.02×10^{4}
50	1.46×10^{0}	2.81×10^{4}	1.61×10^{1}	1.02×10^{5}	3.00×10^{1}	1.06×10^{6}	3.00×10^{1}	1.74×10^{5}
100	8.30×10^{0}	5.92×10^{4}	4.49×10^{1}	3.12×10^{5}	6.00×10^{1}	2.46×10^{6}	6.00×10^{1}	5.02×10^{5}
200	1.28×10^{1}	1.04×10^{5}	9.79×10^{1}	8.70×10^{5}	1.20×10^{2}	5.16×10^{6}	1.50×10^{2}	1.57×10^{6}
500	2.33×10^{1}	2.70×10^{5}	3.17×10^{2}	2.74×10^{6}	3.40×10^{2}	1.65×10^{7}	5.80×10^{2}	6.06×10^{6}
1000	4.23×10^{1}	5.49×10^{5}	7.44×10^{2}	5.90×10^{6}	6.70×10^{2}	3.54×10^{7}	1.17×10^{3}	1.19×10^{7}
10,000	3.99×10^{2}	6.58×10^{6}	1.51×10^{4}	1.04×10^{8}	1.19×10^{4}	6.43×10^{8}	3.93×10^{4}	4.23×10^{8}

Table 5. CPU time and number of function evaluations for different values of r with $a = 1$, $d = 1 \times 10^{-4}$, $N = 1000$.

	LIQSS2		DASSL		DOPRI		Radau5	
r	time	eval.	time	eval.	time	eval.	time	eval.
100	3.35×10^{1}	5.93×10^{5}	3.53×10^{2}	1.94×10^{6}	1.10×10^{2}	5.38×10^{6}	5.80×10^{2}	5.50×10^{6}
500	4.34×10^{1}	5.45×10^{5}	4.79×10^{2}	3.68×10^{6}	3.10×10^{2}	1.61×10^{7}	9.90×10^{2}	9.18×10^{6}
1000	4.66×10^{1}	6.05×10^{5}	7.41×10^{2}	5.64×10^{6}	7.00×10^{2}	3.54×10^{7}	1.29×10^{3}	1.23×10^{7}
2000	4.49×10^{1}	6.51×10^{5}	1.05×10^{3}	1.00×10^{7}	1.21×10^{3}	6.37×10^{7}	2.51×10^{3}	2.41×10^{7}
5000	5.08×10^{1}	6.84×10^{5}	1.50×10^{3}	1.71×10^{7}	2.60×10^{3}	1.41×10^{8}	3.58×10^{3}	3.52×10^{7}
10,000	5.25×10^{1}	7.04×10^{5}	1.75×10^{3}	2.14×10^{7}	5.25×10^{3}	2.78×10^{8}	4.39×10^{3}	4.49×10^{7}
100,000	5.64×10^{1}	7.68×10^{5}	3.29×10^{3}	5.12×10^{7}	4.68×10^{4}	2.71×10^{9}	8.93×10^{3}	9.43×10^{7}

In this scenario LIQSS2 shows a noticeable advantage over the other methods as its performance is not affected at all by the growth of the reaction term r. When r grows the problem becomes more stiff, but this stiffness is due to a large entry in the main diagonal of the Jacobian matrix, which is efficiently handled by LIQSS2.

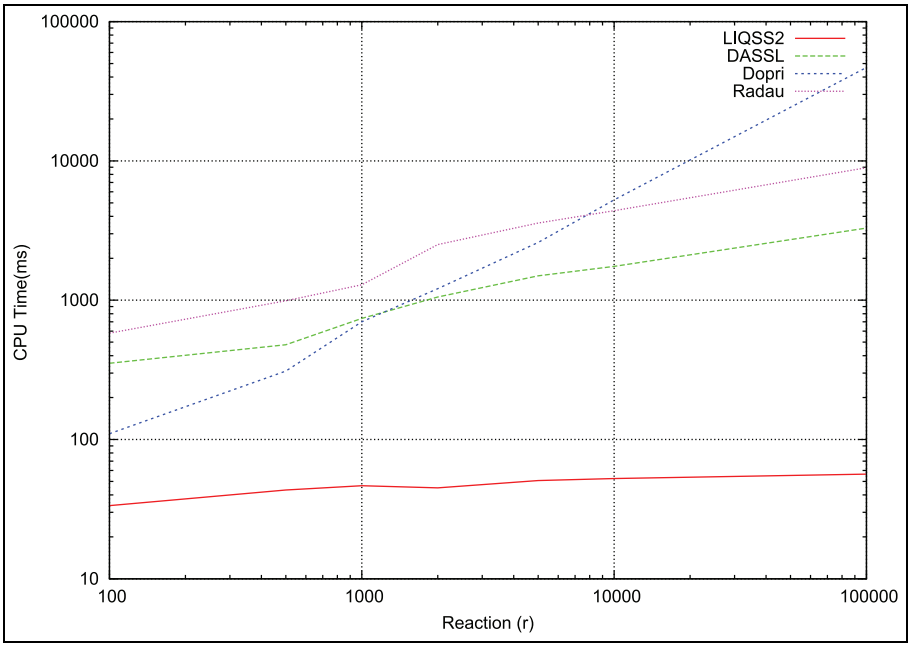

Figure 7. CPU time versus r with $a = 1$, $d = 1 \times 10^{-4}$, $N = 1000$.

However, the other methods present various drawbacks. DOPRI, being explicit, has its step size limited by the stability region which is reduced linearly with r. Thus, the computational cost grows linearly with r.

DASSL and Radau do not have stability issues, but the growth of r increases the non-linearity of the problem and the Newton iteration used by these implicit algorithms requires more steps to converge.

In conclusion, for the last case analyzed ($r = 100,000$), LIQSS2 is about 60 times faster than DASSL, 160 times faster than Radau and 830 times faster than DOPRI.

4.5. Variation of diffusion term d

In the last scenario we study the computational cost for different values of the diffusion term d while the remaining parameters are kept fixed ($a = 1$, $N = 1000$, $r = 1000$). Errors are similar to those of the first scenario so they are not reported.

Figure 8 plots the computational costs as a function of d while Table 6 summarizes the results together with the number of scalar function evaluations.

For low values of d, LIQSS2 again outperforms the other methods. However, as the diffusion term grows, LIQSS2 performance is soon degraded. The reason for this is the appearance of stiffness which is not reflected at the main diagonal of the Jacobian matrix. These stiff cases are not correctly handled by LIQSS algorithms, as is analyzed by Migoni et al.[11]

4.6 A simple 2D scenario

In this scenario we briefly analyze whether the results found before hold for 2D cases. For that purpose we consider a 2D MOL advection–reaction model given by the following equations:

$$\dot{u}_{i,j} = -a_x \frac{(u_{i,j} - u_{i,j-1})}{\Delta x} - a_y \frac{(u_{i,j} - u_{i-1,j})}{\Delta y} + r(u_{i,j}^2 - u_{i,j}^3) \tag{31}$$

for $i = 2, \ldots, N, j = 2, \ldots, N$,

$$\dot{u}_{i,1} = -a_x \frac{u_{i,1}}{\Delta x} - a_y \frac{(u_{i,1} - u_{i-1,1})}{\Delta y} + r(u_{i,1}^2 - u_{i,1}^3) \tag{32}$$

for $i = 2, \ldots, N$,

$$\dot{u}_{1,j} = -a_x \frac{(u_{1,j} - u_{1,j-1})}{\Delta x} - a_y \frac{u_{1,j}}{\Delta y} + r(u_{1,j}^2 - u_{1,j}^3) \tag{33}$$

for $j = 2, \ldots, N$ and finally

$$\dot{u}_{1,1} = -a_x \frac{u_{1,1}}{\Delta x} - a_y \frac{u_{1,1}}{\Delta y} + r(u_{1,1}^2 - u_{1,1}^3) \tag{34}$$

where the grid refinement is defined by $\Delta x = \Delta y = 10/N$.

We simulated this model for different grid refinement settings, obtaining the results summarized in Table 7.

We note that DASSL solver fails to perform from $N \times N = 100 \times 100$. In this case, DASSL must invert a huge matrix which is no longer tridiagonal.

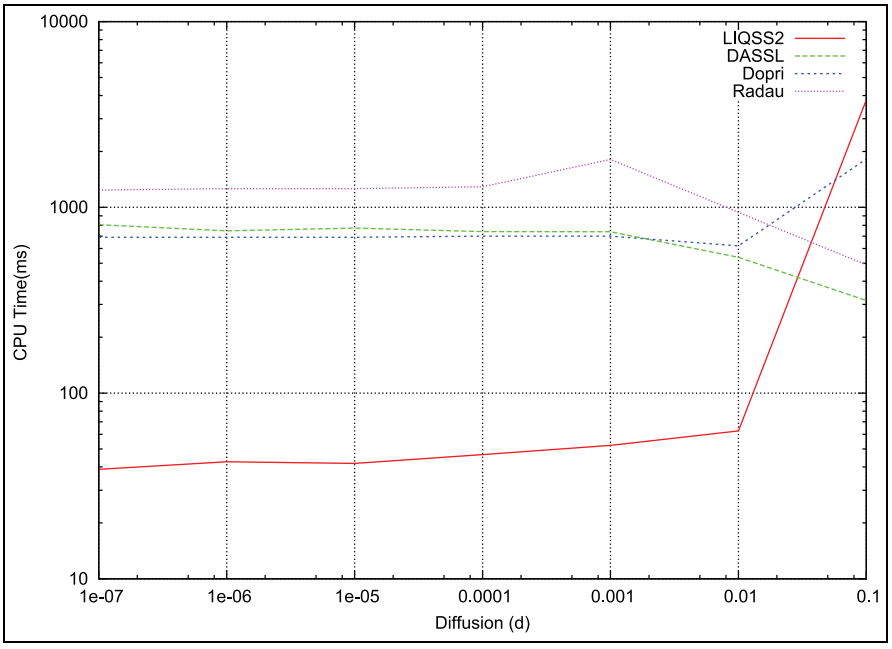

Figure 8. CPU time comparison for different magnitudes of diffusion d: $a = 1$, $N = 1000$, $r = 1000$.

Table 6. CPU time and number of function evaluations for different d: $a = 1$, $N = 1000$, $r = 1000$.

	LIQSS2		DASSL		DOPRI		Radau5	
d	time	eval.	time	eval.	time	eval.	time	eval.
1×10^{-1}	3.79×10^{3}	6.19×10^{7}	3.15×10^{2}	2.46×10^{6}	1.82×10^{3}	9.45×10^{7}	4.90×10^{2}	4.76×10^{6}
1×10^{-2}	6.25×10^{1}	8.68×10^{5}	5.38×10^{2}	4.04×10^{6}	6.20×10^{2}	3.16×10^{7}	9.40×10^{2}	9.00×10^{6}
1×10^{-3}	5.23×10^{1}	6.07×10^{5}	7.39×10^{2}	5.26×10^{6}	7.00×10^{2}	3.63×10^{7}	1.81×10^{3}	1.59×10^{7}
1×10^{-4}	4.66×10^{1}	6.05×10^{5}	7.41×10^{2}	5.64×10^{6}	7.00×10^{2}	3.54×10^{7}	1.29×10^{3}	1.23×10^{7}
1×10^{-5}	4.18×10^{1}	5.62×10^{5}	7.73×10^{2}	5.77×10^{6}	6.90×10^{2}	3.54×10^{7}	1.26×10^{3}	1.20×10^{7}
1×10^{-6}	4.27×10^{1}	5.48×10^{5}	7.47×10^{2}	5.45×10^{6}	6.90×10^{2}	3.54×10^{7}	1.26×10^{3}	1.19×10^{7}
1×10^{-7}	3.89×10^{1}	5.22×10^{5}	8.07×10^{2}	6.11×10^{6}	6.90×10^{2}	3.54×10^{7}	1.24×10^{3}	1.19×10^{7}

Table 7. CPU time (ms) for different grid refinement settings ($N \times N$): $a_x = a_y = 1$, $r = 1000$.

	LIQSS2		DOPRI		DASSL	
	time	eval.	time	eval.	time	eval.
10×10	5.70×10^{0}	4.86×10^{4}	1.37×10^{2}	2.39×10^{6}	8.64×10^{1}	1.28×10^{6}
50×50	1.82×10^{2}	1.17×10^{6}	3.02×10^{3}	7.16×10^{7}	3.76×10^{4}	4.69×10^{8}
100×100	8.59×10^{2}	4.68×10^{6}	1.19×10^{4}	2.91×10^{8}	—	—
500×500	5.69×10^{4}	1.21×10^{8}	4.03×10^{5}	9.40×10^{9}	—	—
1000×1000	4.35×10^{5}	4.96×10^{8}	1.91×10^{6}	4.24×10^{10}	—	—

As before, the LIQSS2 method exhibits a better performance than DOPRI and DASSL. We must mention that while the number of function evaluations of LIQSS2 grows linearly with the system size, the CPU time scales supra-linearly. This is due to the fact that the stage of the QSS solver that translates the μ-Modelica model into C language does not support 2D models yet, so we wrote the $N \times M$ matrix using M arrays. In consequence, the μ-Modelica model was inefficiently translated into C, with the right-hand side of the ODE containing M unnecessary

comparisons. As M grows, those unnecessary comparisons affect the overall performance.

5. Conclusions

In this work we studied the application of quantization-based integration methods for semi-discretized 1D ADR problems.

We compared the LIQSS2 method against widely used classic numerical integration methods implemented in solvers such as DASSL, Radau and DOPRI.

We make the following conclusions.

- LIQSS2 is a better option than classical numerical integration methods when the relation between the advection and the diffusion is large (i.e. large Péclet numbers). However, when the diffusion is higher, the stiffness introduced is not properly handled by LIQSS2 and classic methods are more efficient.
- Provided that the diffusion term is kept small, LIQSS2 shows an increasing advantage over the other methods while N grows since it scales sub-linearly with the grid refinement.
- Contrary to classic methods, LIQSS2 performance is not affected by the growth of the reaction term r. This is because (as stated in Section 2.2) LIQSS methods efficiently handle stiffness due to large entries in the main diagonal of the Jacobian matrix.
- In most cases, LIQSS2 performed at least 10 times faster than classic solvers.

We also have performed a theoretical analysis on the maximum error introduced by the LIQSS methods for purely advective cases. A simulation study showed that this error bound, in spite of being extremely conservative, still holds in the presence of diffusion and reaction terms.

We also extended the results to a simple 2D advection–reaction case, obtaining promising results regarding the usage of LIQSS methods in higher-dimensional problems, where they can still offer advantages over classic discrete time algorithms.

It is worth mentioning that in these two-dimensional studies, we are reporting simulation results with QSS methods on a system having up to one million states. To the best of the authors' knowledge, this is the first time QSS methods are applied to models of this size.

In spite of the advantages observed, we must recall that this work is limited to some special cases (1D ADR and 2D advection–reaction equations) with particular initial states and boundary conditions, and semi-discretized with the MOL using first-order finite differences. Thus, future work should corroborate these results on a more general context, considering the following.

- More sophisticated models, including 2D and 3D problems with realistic initial states and boundary conditions, such as environmental geochemistry, and pollutants transport in surface and ground water. Also adding native support for 2D models in the QSS tool is a must.
- The use of different space discretization methods, such as boundary integral methods or meshless methods.
- The usage of the MOL with higher-order finite differences.

It would be also of theoretical interest to extend the error analysis to the complete ADR model, including diffusion and reaction terms.

Funding

This work was funded by grants ANPCYT PICT 2012 - 0077 and CONICET PIP 2012-2014 00216.

References

1. Volker J and Schmeyer E. Finite element methods for time-dependent convection–diffusion–reaction equations with small diffusion. *Comput Meth Appl Mech Eng* 2008; 198(3–4): 475–494.
2. Theeraek P, Phongthanapanich S and Dechaumphai P. Solving convection–diffusion–reaction equation by adaptive finite volume element method. *Math Comput Sim* 2011; 82(2): 220–233.
3. Portapila M and Power H. A convergence analysis of the performance of the DRM-MD boundary integral approach. *Int J Numer Meth Eng* 2007; 71(1): 47–65.
4. Caruso N, Portapila M and Power H. Local regular dual reciprocity method for 2D convection-diffusion equation. In *34th international conference on boundary elements and other mesh reduction methods*, pp. 27–37.
5. Schiesser W. *The numerical method of lines: integration of partial differential equations*. New York: Academic Press, 1991.
6. Cellier F and Kofman E. *Continuous System Simulation*. New York: Springer, 2006.
7. Butcher J. *Numerical Methods for Ordinary Differential Equations*. New York: John Wiley & Sons, Ltd, 2005.
8. Brenan KE, Campbell SL and Petzold LR. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1995.
9. Petzold LR. A description of DASSL: a differential/algebraic system solver. In *Scientific computing*, Montreal, Quebec, 1982. New Brunswick, NJ: IMACS, 1983, pp. 65–68.
10. Kofman E and Junco S. Quantized state systems. A DEVS approach for continuous system simulation. *Trans SCS* 2001; 18(3): 123–132.
11. Migoni G, Bortolotto M, Kofman E, et al. Linearly implicit quantization-based integration methods for stiff ordinary

differential equations. *Sim Modell Practice Theory* 2013; 35: 118–136.
12. Kofman E. Relative error control in quantization based integration. *Latin Amer Appl Res* 2009; 39(3): 231–238.
13. Kofman E. Discrete event simulation of hybrid systems. *SIAM J Sci Comput* 2004; 25(5): 1771–1797.
14. Kofman E. A second order approximation for DEVS simulation of continuous systems. *Simulation* 2002; 78(2): 76–89.
15. Kofman E. A third order discrete event simulation method for continuous system simulation. *Latin Amer Appl Res* 2006; 36(2): 101–108.
16. Migoni G, Kofman E and Cellier F. Quantization-based new integration methods for stiff ODEs. *Simulation* 2012; 88(4): 387–407.
17. Zeigler B, Praehofer H and Kim TG. *Theory of Modeling and Simulation*, 2nd edn.New York: Academic Press, 2000.
18. Bergero F and Kofman E. PowerDEVS: a tool for hybrid system modeling and real time simulation. *Simulation* 2011; 87: 113–132.
19. Beltrame T and Cellier F. 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.
20. D'Abreu M and Wainer G. M/CD + + : Modeling continuous systems using Modelica and DEVS. In *Proceedings of MASCOTS 2005*, Atlanta, GA, pp. 229–236.
21. Quesnel G, Duboz R, Ramat E, et al. VLE: a multimodeling and simulation environment. In *Proceedings of the 2007 Summer computer simulation conference*, San Diego, California, pp. 367–374.
22. Esquembre F. Easy Java Simulations: a software tool to create scientific simulations in Java. *Comput Phys Commun* 2004; 156(1): 199–204.
23. Fernández J and Kofman E. A stand-alone quantized state system solver for continuous system simulation. *Simulation* 2014; in press.
24. Fritzson P and Engelson V. Modelica - a unified object-oriented language for system modelling and simulation. In *ECOOP*, pp. 67–90.
25. Wolke R and Knoth O. Implicit–explicit Runge–Kutta methods applied to atmospheric chemistry-transport modelling. *Environ Modell Software* 2000; 15(6): 711–719.
26. Sommeijer B, Shampine L and Verwer J. RKC: An explicit solver for parabolic {PDEs}. *J Computat Appl Math* 1998; 88(2): 315–326.
27. Verwer J, Sommeijer B and Hundsdorfer W. RKC time-stepping for advection–diffusion–reaction problems. *J Computat Phys* 2004; 201(1): 61–79.
28. Kleefeld B and Martn-Vaquero J. SERK2v2: A new second-order stabilized explicit Runge–Kutta method for stiff problems. *Numer Meth Partial Differ Eq* 2013; 29(1): 170–185.
29. Álvarez J and Rojo J. An improved class of generalized Runge–Kutta methods for stiff problems. Part I: The scalar case. *Applied Mathematics and Computation* 2002;.
30. Álvarez J and Rojo J. An improved class of generalized Runge–Kutta methods for stiff problems. Part II: The separated system case. *Appl Math Computat* 2004; 159(3): 717–758.
31. Savcenco V, Hundsdorfer W and Verwer J. A multirate time stepping strategy for stiff ordinary differential equations. *BIT Numer Math* 2007; 47(1): 137–155.
32. Muzy A, Jammalamadaka R, Zeigler B, et al. The activity-tracking paradigm in discrete-event modeling and simulation: The case of spatially continuous distributed systems. *Simulation* 2011; 87(5): 449–464.
33. Hundsdorfer W and Verwer JG. *Numerical Solution of Time-Dependent Advection–Diffusion–Reaction Equations*. New Yor: Springer, 2003.
34. Gilding BH and Kersner R. Travelling waves in nonlinear diffusion–convection–reaction. Memorandum 1585, Department of Applied Mathematics, University of Twente, Enschede, 2001.
35. Migoni G, Kofman E, Bergero F, et al. Quantization-based simulation of switched mode power supplies. *Simulation* 2015; 91(4): 320–336.
36. Brown P, Hindmarsh A and Petzold L. Using Krylov methods in the solution of large-scale differential-algebraic systems. *SIAM J Sci Comput* 1994; 15(6): 1467–1488.

Author biographies

Federico Bergero received a computer science degree in 2008 and a PhD in informatics in 2012, both from the UNR Argentina. He is currently a research assistant at CIFASIS and holds a professorial position at the UNR Argentina. His research interests include discrete event systems, real time and parallel simulation, modeling languages and simulation tools.

Joaquín Fernández received his BS in computer science in 2012 from the Universidad Nacional de Rosario, Argentina. He is currently a PhD student at the French Argentine International Center for Information and Systems Sciences (CIFASIS). His research interests include hybrid system simulation, real-time and parallel simulation, simulation tools, and modeling languages.

Ernesto Kofman received his BS in electronic engineering in 1999 and his PhD in automatic control in 2003, both from the National University of Rosario. He is an adjunct professor at FCEIA–UNR. He also holds a research position at the CIFASIS–CONICET. His research interests include automatic control theory and numerical simulation of hybrid systems.

Margarita Portapila is full professor at the School of Civil Engineering of National University of Rosario and head of the Computational Fluid Dynamics division at CIFASIS.