Asociación Argentina de Mecánica Computacional

Mecánica Computacional Vol XXXII, págs. 1103-1119 (artículo completo)
Carlos G. García Garino, Aníbal E. Mirasso, Mario A. Storti, Miguel E. Tornello (Eds.)
Mendoza, Argentina, 19-22 Noviembre 2013

QUANTIZED STATE SIMULATION OF ADVECTION-DIFFUSION-REACTION EQUATIONS

Federico Bergero[a,b], **Joaquín Fernández**[a], **Ernesto Kofman**[a,b] **and Margarita Portapila** [a,b]

[a] *CIFASIS–CONICET, Ocampo y Esmeralda (S2000EZP) Rosario, Argentina, Phone: +54 (341) 4237248 Ext. 370, bergero@cifasis-conicet.gov.ar, http://www.fceia.unr.edu.ar/~fbergero/*

[b]*Facultad de Ciencias Exactas, Ingeniería y Agrimensura - UNR - Rosario, Argentina*

Keywords: Advection-Diffusion-Reaction Equation, Quantization Based Integration Methods, Numerical Simulation.

Abstract. Time-dependent Advection-Diffusion-Reaction (ADR) equations are used in areas such as chemistry, physics and engineering. These areas include chemical reactions, population dynamics, flame propagation, and the evolution of concentrations in environmental and biological processes.

Each of the three phenomena (advection, diffusion, and reaction) evolves in a different time scale, thus the model shows a stiff behavior.

This equation is usually discretized along the spatial variables using a grid, converting it into a large sparse set of ordinary differential equations (ODEs) that can be then solved using numerical integration methods that discretize the time variable.

An alternative way is the usage of Quantized State Systems (QSS) methods, a family of numerical integration algorithms that replace the time discretization by the quantization of the state variables. Some QSS algorithms can efficiently integrate sparse stiff ODEs, which makes them promising candidates for the ADR problem.

In this article we study the use of QSS methods for ADR models semi–discretized with the Method Of Lines. We compare the performance and the quality of the solutions obtained by these algorithms with those of conventional methods, such as DASSL, Radau and DOPRI.

Analyzing simulation times we show that, in most situations, the second order linearly implicit QSS method (LIQSS2) outperforms all the conventional algorithms in more than one order of magnitude.

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. It is well known that the advection-dominated diffusion problem often develops sharp fronts that are nearly shocks. Therefore, it is not easy to construct an effective numerical method for solving such a problem. Besides, there may also be a change in $u(x,t)$ due to chemical reactions, combining the three effects of advection, diffusion and chemical reaction, leading to the advection-diffusion-reaction (ADR) equation.

Reactive multicomponent flows are studied in environmental sciences as well as in mechanical engineering. In applications of practical interest, the number of unknowns may be large such that fast and efficient methods are needed.

Moreover, it should be considered that the chemical reactions take place on very small time scales compared to the long term effects considered for the advection-diffusion transport. Each of the three phenomena (advection, diffusion, and reaction) evolving in a different time scale.

Systems that exhibit simultaneous fast and slow dynamics are called stiff[1] (Cellier and Kofman, 2006). Due to numerical stability issues, these systems enforce the usage of implicit numerical integration algorithms which have a high computational cost, particularly when the system dimension is large.

When we say that the advection-diffusion-reaction equation is advection-reaction-dominated, means that the diffusivity is relatively small compared with the module of the advection field or the reaction coefficient, i.e. we will have high Péclet number (Pe) and high Damköhler number. In physical terms, the Péclet number represents the ratio of convective forces to diffusive forces, and is a non-dimensional quantity. A larger Péclet number represents an increasingly advection-dominated situation.

As for the numerical solution of the ADR equation, when the Pe increases numerical solutions produce low accuracy or suffer from instabilities. Advection-Diffusion-Reaction problems have been examined in the literature, with a wide range of configurations encompassing variable velocity fields, variable reaction coefficients, steady and transient problems, in one, two and three dimensions (John and Schmeyer, 2008; Theeraek et al., 2011; Portapila and Power, 2007; Caruso et al., 2012).

The numerical solution of a partial differential equation (PDE) such as the ADR problem involves discretization in space and time coordinates. While some methods perform the simultaneous discretization in space and time, some techniques discretize only in space transforming the PDE into a set of ODEs that are then solved by numerical integration algorithms (Cellier and Kofman, 2006). The Method of Lines (MOL) (Schiesser, 1991; Cellier and Kofman, 2006) is one of these *semi-discretization* techniques.

The idea of the MOL is to use a grid over space and compute the solution in the grid nodes replacing the partial derivatives by finite differences. The resulting model is then a large sparse set of ODEs where the state variables of a grid node are only related to the state variables of some neighbor nodes.

The resulting ODE can be then simulated with numerical integration methods such as Euler's, Runge-Kutta (Butcher, 2005; Cellier and Kofman, 2006), DASSL (Brenan et al., 1995;

[1]In the context of this work, the term stiffness is used in the mathematical sense, where it is associated to the presence of large and small eigenvalues in the Jacobian matrix. This concept of stiffness is not related to that of structural stiffness.

Petzold, 1983), which solve the equations performing a time discretization.

An alternative way to solve the ODE is given by the QSS methods (Kofman, 2006; Cellier and Kofman, 2006), that replace the time discretization by state quantization. QSS methods are efficient when dealing with discontinuous and sparse systems, and there are Linearly Implicit QSS (LIQSS) methods (Migoni et al., 2013) that are also able to tackle stiff systems.

Thus, LIQSS methods appear as promising candidates for integrating the ODEs resulting from the space discretization of Advection-Diffusion-Reaction problems.

In this article we study the use of LIQSS methods in the simulation of Advection-Diffusion-Reaction problems semi–discretized with the MOL comparing its performance with that of classic algorithms such as DASSL and Runge–Kutta. The comparison is performed over different sets of parameters and under different grid refinement conditions.

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. Then, Section 3 briefly discusses the implementation of the model in a QSS solver and Section 4 shows the main results of the the usage of QSS methods in Advection-Diffusion-Reaction models, making performance comparisons with classical numerical integration methods. Finally, Section 5 concludes the article and discusses some lines of future work.

2 BACKGROUND

In this section, we introduce the basic concepts that are then used along the article.

We first describe the Advection-Diffusion-Reaction model, its discretizations for space and time, and finally we explain the Quantization Based Integration Methods.

2.1 The Advection-Diffusion-Reaction Equation

Let $u(x,t)$ be the concentration of some species in the space coordinate x at time t. Then, the one dimensional Advection and Diffusion (Hundsdorfer and Verwer, 2003) process can be described by the following PDE:

$$\frac{\partial u(x,t)}{\partial t} + a\frac{\partial u(x,t)}{\partial x} = d\frac{\partial^2 u(x,t)}{\partial^2 x} \tag{1}$$

where a and d are adimensional parameters expressing the amount of advection and diffusion respectively.

Taking into account that the species undergoes a chemical reaction, we include a reaction term following Zeldovich's equation (Gilding and Kersner, 2001) as follows:

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

where r is an adimensional parameter expressing the amount of reaction.

This is the model we shall work with along the rest of the article.

2.2 Space Discretization - Method of Lines

A PDE like that of Eq.(2) can be numerically solved using different approaches.

A popular technique to solve it is known as the *Method of Lines*, that replaces the space derivatives by finite difference approximations (Cellier and Kofman, 2006). That way, the problem is converted into a set of ODEs which can be solved using conventional numerical integration algorithms.

The replacement of space derivatives by finite differences can be seen as a *spatial discretization*, which is usually performed over a regular grid. For instance, the use of first order finite differences leads to replacements like

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

where x_i is the space position of the i–th grid point, Δx is the grid width and $u_i(t) \approx u(x_i,t)$.

The replacement of all space derivatives then leads to a system of pure ODEs. Naturally, this discretization provokes what is know as the *consistency error*, which can be minimized with the use of smaller grid widths or using higher order finite difference approximations.

2.3 Classical Numerical Integration

After applying the Method of Lines, the original PDE becomes an ODE of the form

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

where $\mathbf{x}(t)$ is the state vector.

Explicit single–step numerical algorithms transform Eq.(3) into a difference equation of the form:

$$\mathbf{x}(t_{k+1}) = \mathbf{F}(\mathbf{x}(t_k),t_k) \tag{4}$$

In multi–step algorithms the right hand side of the previous equations depends also on past values of the state vector $\mathbf{x}(t_{k-1})$, $\mathbf{x}(t_{k-2})$, etc.

A limitation of explicit numerical integration algorithms is that their numerical solutions become unstable as the step size $h = t_{k+1} - t_k$ grows. The simulation of stiff systems (i.e., systems with simultaneous fast and slow dynamics) (Hairer and Wanner, 1991; Cellier and Kofman, 2006) requires that the methods preserve numerical stability independently of the step size, a feature that can be only achieved by some implicit algorithms. These methods lead to approximations of the form

$$\mathbf{F}(\mathbf{x}(t_{k+1}),\mathbf{x}(t_k),t_k) = 0 \tag{5}$$

where $\mathbf{x}(t_{k+1})$ is generally obtained through iterative procedures such as the Newton iteration. These methods, which in the case of multi–step algorithm also depend on past values of the state, are called *stiff stable*.

The literature on numerical integration of ODEs contains many single-step, multi-step, explicit and implicit algorithms (Hairer and Wanner, 1991; Hairer et al., 1993; Cellier and Kofman, 2006).

In this work, we shall focus on three particular methods: The explicit embedded Runge Kutta method of Dormand–Prince (DOPRI) (Dormand and Prince, 1980), the implicit RK method of Radau5 (Hairer and Wanner, 1991) and the popular solver DASSL (Petzold, 1982), based on variable step and variable order series of Backward Difference Formulae (BDF) algorithms (Cellier and Kofman, 2006).

Variants of DOPRI, DASSL and Radau are the default solvers of most ODE and Differential Algebraic Equations (DAE) simulation tools.

2.4 Quantization State System Methods

Quantized State System (QSS) methods replace the time discretization of classic numerical integration algorithms by the quantization of the state variables.

Given the ODE of Eq.(3), the first order Quantized State System method (QSS1) (Kofman and Junco, 2001) approximates it by

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

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*:

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

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

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

The QSS1 method has the following features:

- In the solution, the quantized states $q_j(t)$ follow piecewise constant trajectories.
- The state variables $x_j(t)$ follow piecewise linear trajectories.
- The state and quantized variables never differ more than the quantum ΔQ_j. This fact ensures stability and global error bound properties (Kofman and Junco, 2001; Cellier and Kofman, 2006).
- 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 (Kofman, 2009).
- Each step is local to a 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 (Kofman, 2004).

The main limitations of QSS1 are the following:

- It only performs a first order approximation, and a good accuracy cannot be obtained without a significant increment in the number of steps.
- It is not suitable to simulate stiff systems. In this cases, the appearence of fast oscillations limits the step size due to stability issues.

The first limitation was solved with the introduction of higher order QSS methods like QSS2 (Kofman, 2002), where the quantized state follow piecewise linear trajectories, and QSS3 (Kofman, 2006) where the quantized state follow piecewise parabolic trajectories.

Regarding stiff systems, a first order backward QSS method (BQSS) was introduced in (Migoni et al., 2012). This method, in spite of being backward, was explicit due to the following property. In BQSS the next state value is always known as it should be $q_j \pm \Delta Q_j$. The unknown is the time at which the state reaches it which can be explicitly computed.

In BQSS, the oscillations mentioned above dissapear and the step size is no longer limited by stability concerns. That way, BQSS integrates with small steps the fast variables and with large steps the slow ones.

Unfortunately, BQSS cannot be extended to higher order approximations. However, a family of linearly implicit QSS methods (LIQSS) of order 1 to 3 was also proposed in (Migoni et al., 2013). LIQSS methods, like BQSS, are also explicit algorithms.

LIQSS methods have the same advantages of QSS methods, and they are able to efficiently handle many stiff systems, provided that the stiffness is due to the presence of large entries in the main diagonal of the Jacobian matrix.

In the context of this work, the efficient sparsity exploitation and the explicit treatment of stiffness will provide the main advantages of LIQSS. In presence of diffusion, stiffness appears without showing large entries anywhere in the Jacobian matrix and that will impose an important limitation. Nevertheless, it must be considered that for the ADR equation the numerical difficulties arise generally when the Péclet number increases, and consequently the advection term dominates over diffusion.

2.5 Implementation of QSS Methods

It was shown that the behavior of the QSS approximation of Eq.(6) can be described as a Discrete EVent System (DEVS)(Zeigler et al., 2000). Thus, the easiest was of implementing these algorithm is through their equivalents on a DEVS simulation engine.

The whole family of QSS methods were implemented in PowerDEVS (Bergero and Kofman, 2011), a DEVS–based simulation platform specially designed for and adapted to simulating hybrid systems based on QSS methods. In addition, the explicit QSS methods of orders 1 to 3 were also implemented in a DEVS library of Modelica (Beltrame and Cellier, 2006) and implementations of the first–order QSS methods can also be found in CD++ (D'Abreu and Wainer, 2005) and VLE (Quesnel et al., 2007).

DEVS–based implementations of QSS methods are simple but they are not efficient. The problem is that the DEVS simulation engines waste a large amount of the computational load attending the DEVS simulation mechanism. This fact motivated the development of stand alone QSS solvers.

A first approach to a stand–alone version of QSS1 to 3 was implemented in the Java–based simulation tool *Open Source Physics* (Esquembre, 2004), but that implementation was not more efficient that 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 (Fernandez and Kofman, 2012). This solver improves PowerDEVS simulation times in more than one order the magnitude, and can simulate models described in a subset of the Modelica language (Fritzson and Engelson, 1998), called μ-Modelica (Bergero et al., 2012).

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

2.6 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 (Wolke and Knoth, 2000; Sommeijer et al., 1998; Verwer et al., 2004; Kleefeld and Martín-Vaquero, 2013; Álvarez and Rojo, 2002, 2004).

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

In (Savcenco et al., 2007) Savcenco et al. 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. (Muzy et al., 2011) showed the results of using QSS methods for a one dimensional diffusion problem. Hyperbolic PDEs representing lossless transmission lines were also simulated in the context of QSS methods in (Kofman, 2002; Migoni et al., 2012), including also a stiff load.

3 THE ADR MODEL IN A QSS SOLVER

In this section, we first explain the way we proceeded to discretize the ADR model of Eq.(2) using the MOL and then we show how we described in the QSS solver the resulting set of ODEs.

3.1 ADR PDE Model

The ADR model we shall work with is described by the PDE

$$\frac{\partial u(x,t)}{\partial t} + a\frac{\partial u(x,t)}{\partial x} = d\frac{\partial^2 u(x,t)}{\partial^2 x} + r(u(x,t)^2 - u(x,t)^3) \tag{8}$$

The space domain is limited to the interval $0 \leq x \leq 10$ and we shall consider the following boundary conditions

$$u(x=0,t) = 1; \quad \frac{\partial u(x=10,t)}{\partial x} = 0; \tag{9}$$

The initial condition is given by

$$u(x,t=0) = \begin{cases} 1 & \text{if } x < 2 \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

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

$$\Delta x = \frac{10}{N} \tag{11}$$

where N is the number of grid points.

The advection term of Eq.(8) $\frac{\partial u(x,t)}{\partial x}$ shall be replaced by a first order upwind finite difference:

$$\left.\frac{\partial u}{\partial x}\right|_{x=x_i} \approx \frac{u_i - u_{i-1}}{\Delta x} \tag{12}$$

for $i = 1, \cdots, N$, where $u_i(t) \approx u(x_i, t)$ is the i–th state variable of the resulting ODE and $x_i = i \cdot \Delta x$ is the i–th grid point.

Taking into account the boundary condition of Eq.(9) at $x = 0$, we have also $u_0 = 1$.

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

$$\left.\frac{\partial^2 u}{\partial^2 x}\right|_{x=x_i} \approx \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2} \tag{13}$$

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

For the last grid point, taking into account the symmetrical border condition of Eq.(9) at $x = 10$, we can replace

$$\left.\frac{\partial^2 u}{\partial^2 x}\right|_{x=x_N} \approx \frac{u_{N-1} - 2u_N + u_{N-1}}{\Delta x^2} \tag{14}$$

Replacing Eqs (12)–(14) into Eq. (8) we get the following set of ODEs:

$$\dot{u_i} = -a\frac{(u_i - u_{i-1})}{\Delta x} + d\frac{(u_{i+1} - 2u_i + u_{i-1})}{\Delta x^2} + r(u_i^2 - u_i^3) \tag{15}$$

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

$$\dot{u_N} = -a\frac{(u_N - u_{N-1})}{\Delta x} + d\frac{(2u_{N-1} - 2u_N)}{\Delta x^2} + r(u_N^2 - u_N^3) \tag{16}$$

3.3 The ADR Model in the QSS Solver

The ODE model of Eqs.(15)–(16) can be described in the subset of Modelica language (μ-Modelica) of the QSS Solver 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;
```

Notice 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 Fig.1. There, $u[400]$ is the discretized version of $u(x = 4)$, $u[600]$ $u(x = 6)$, etc.

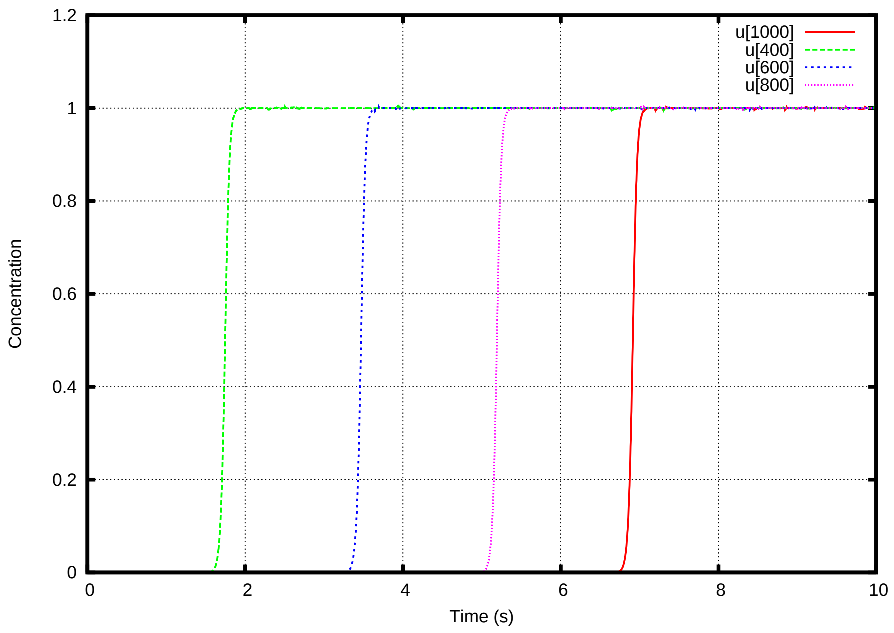

Figure 1: Simulation results for $a = 1, d = 1 \cdot 10^{-4}, r = 10, N = 1000$ points using LIQSS2 method.

4 RESULTS

In this section we compare the performance of different numerical integration methods on the ADR problem semidiscretized with the MOL. For that purpose, the resulting model of Eq.(15) is simulated for different parameter settings using LIQSS2, DASSL, Radau5 and DOPRI.

- DASSL results were computed using the Fortran code DASPK described in (Brown et al., 1994).
- 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 the simulations were performed on the same Intel i7-3770@3.40GHz computer under a Linux Operating System (Ubuntu).
- The error in all cases were computed comparing the results with reference results obtained using a very small error tolerance ($1 \cdot 10^{-10}$) with DOPRI.
- We did not compute consistency errors due to the MOL space discretization. We are only interested in the ODE integration error.
- In all scenarios we gave the numerical solver a relative tolerance of $1 \cdot 10^{-3}$ and an absolute tolerance of $1 \cdot 10^{-4}$.
- DASSL, DOPRI and Radau5 are variable step solvers that adapt the step size in order to meet the tolerance.
- LIQSS2 can also be considered a variable step solver. Moreover, the step size is adapted differently for each state variable.

- Taking into account that the step size is not constant, we report the number of scalar function evaluations in each case.

- The model was simulated up to $t = 10$ second. Before that time, the model always reaches an equilibrium condition.

4.1 First scenario – Variation of the grid size Δx

In this first scenario we study the computational cost and errors for different number of points N in the grid. The remaining parameters were fixed, $a = 1, d = 1 \cdot 10^{-4}, r = 1000$. The resulting Péclet Number is $a/d = 10000$.

Figure 2 compares the CPU time of DASSL, DOPRI, Radau5 and LIQSS2 as N grows while Table 1 summarizes the results together with the number of scalar function evaluations.

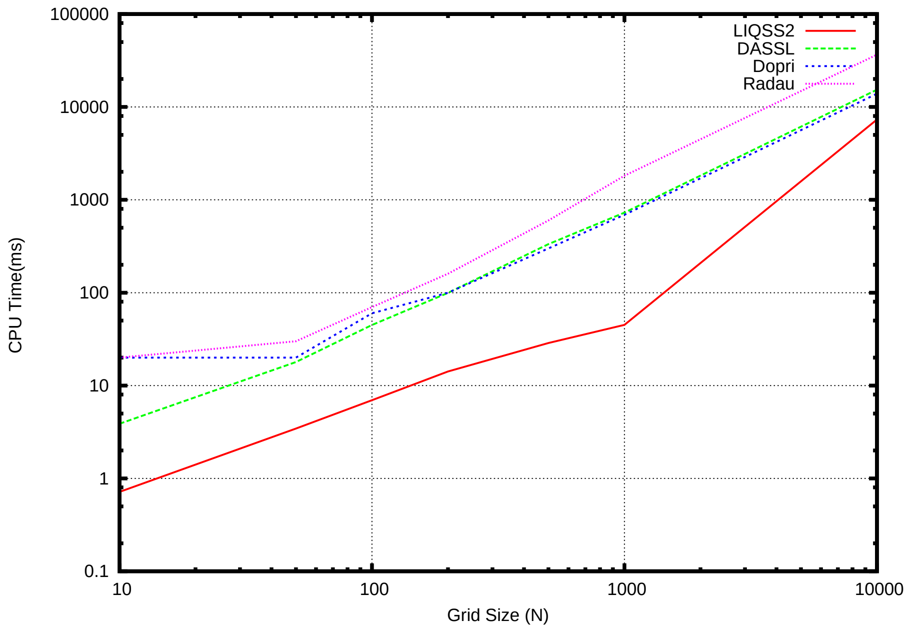

Figure 2: CPU time(ms) vs. N (number of points in the grid) with $a = 1, d = 1 \cdot 10^{-4}, r = 1000$

	LIQSS2		DASSL		DOPRI		Radau5	
N	time	fun. eval.	time	fun. eval.	time	fun. eval.	time	fun. eval.
10	$9.04 \cdot 10^{-1}$	$5.99 \cdot 10^{3}$	$3.85 \cdot 10^{0}$	$8.47 \cdot 10^{3}$	$1.00 \cdot 10^{1}$	$1.88 \cdot 10^{5}$	$2.00 \cdot 10^{1}$	$1.02 \cdot 10^{4}$
50	$3.45 \cdot 10^{0}$	$2.79 \cdot 10^{4}$	$1.93 \cdot 10^{1}$	$1.02 \cdot 10^{5}$	$4.00 \cdot 10^{1}$	$1.06 \cdot 10^{6}$	$2.00 \cdot 10^{1}$	$1.74 \cdot 10^{5}$
100	$6.62 \cdot 10^{0}$	$5.38 \cdot 10^{4}$	$5.11 \cdot 10^{1}$	$3.29 \cdot 10^{5}$	$6.00 \cdot 10^{1}$	$2.45 \cdot 10^{6}$	$5.00 \cdot 10^{1}$	$5.02 \cdot 10^{5}$
200	$1.48 \cdot 10^{1}$	$1.17 \cdot 10^{5}$	$1.10 \cdot 10^{2}$	$8.85 \cdot 10^{5}$	$1.20 \cdot 10^{2}$	$5.17 \cdot 10^{6}$	$1.50 \cdot 10^{2}$	$1.57 \cdot 10^{6}$
500	$2.73 \cdot 10^{1}$	$3.16 \cdot 10^{5}$	$3.33 \cdot 10^{2}$	$2.61 \cdot 10^{6}$	$3.70 \cdot 10^{2}$	$1.73 \cdot 10^{7}$	$6.10 \cdot 10^{2}$	$6.06 \cdot 10^{6}$
1000	$4.66 \cdot 10^{1}$	$6.05 \cdot 10^{5}$	$7.41 \cdot 10^{2}$	$5.64 \cdot 10^{6}$	$7.00 \cdot 10^{2}$	$3.54 \cdot 10^{7}$	$1.29 \cdot 10^{3}$	$1.23 \cdot 10^{7}$
10000	$7.49 \cdot 10^{3}$	$1.07 \cdot 10^{8}$	$1.54 \cdot 10^{4}$	$1.08 \cdot 10^{8}$	$1.50 \cdot 10^{4}$	$7.41 \cdot 10^{8}$	$3.97 \cdot 10^{4}$	$4.04 \cdot 10^{8}$

Table 1: CPU time(ms) and number of function evaluations for different values of N (number of points in the grid) with $a = 1, d = 1 \cdot 10^{-4}, r = 1000$

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

However, at $N = 10000$ the diffusion term at Eq.(15) becomes relevant, since d is divided by Δx^2 while a is only divided by Δx. This situation leads to a type of structural stiffness that is not properly handled by LIQSS methods (Migoni et al., 2013), and 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. It in fact performs several function evaluations, but its low cost per step gives it a similar performance to that of DASSL.

It must be mentioned that DASPK and Radau5 codes in use 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.

	LIQSS2		DASSL		DOPRI		Radau5	
N	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.
10	$5.9 \cdot 10^{-2}$	$2.8 \cdot 10^{-3}$	$7.4 \cdot 10^{-1}$	$7.9 \cdot 10^{-4}$	$3.9 \cdot 10^{-3}$	$8.7 \cdot 10^{-4}$	$2.5 \cdot 10^{-3}$	$2.7 \cdot 10^{-6}$
50	$8.4 \cdot 10^{-2}$	$8.1 \cdot 10^{-4}$	$7.0 \cdot 10^{-1}$	$6.8 \cdot 10^{-4}$	$2.2 \cdot 10^{-2}$	$1.9 \cdot 10^{-3}$	$3.5 \cdot 10^{-3}$	$6.7 \cdot 10^{-6}$
100	$1.2 \cdot 10^{-1}$	$1.7 \cdot 10^{-4}$	$6.6 \cdot 10^{-1}$	$6.1 \cdot 10^{-4}$	$3.8 \cdot 10^{-2}$	$2.5 \cdot 10^{-3}$	$9.1 \cdot 10^{-3}$	$2.9 \cdot 10^{-5}$
200	$1.6 \cdot 10^{-1}$	$1.8 \cdot 10^{-3}$	$7.5 \cdot 10^{-1}$	$7.6 \cdot 10^{-4}$	$9.8 \cdot 10^{-2}$	$3.0 \cdot 10^{-3}$	$3.0 \cdot 10^{-3}$	$1.3 \cdot 10^{-5}$
500	$1.8 \cdot 10^{-1}$	$1.1 \cdot 10^{-3}$	$5.3 \cdot 10^{-1}$	$4.0 \cdot 10^{-4}$	$3.9 \cdot 10^{-2}$	$3.8 \cdot 10^{-3}$	$1.7 \cdot 10^{-2}$	$1.4 \cdot 10^{-5}$
1000	$2.1 \cdot 10^{-1}$	$1.3 \cdot 10^{-3}$	$3.4 \cdot 10^{-2}$	$2.4 \cdot 10^{-5}$	$5.8 \cdot 10^{-2}$	$4.8 \cdot 10^{-3}$	$4.9 \cdot 10^{-2}$	$3.3 \cdot 10^{-5}$
10000	$5.9 \cdot 10^{-1}$	$8.1 \cdot 10^{-4}$	$1.0 \cdot 10^{0}$	$1.4 \cdot 10^{-3}$	$1.9 \cdot 10^{-1}$	$6.6 \cdot 10^{-3}$	$3.0 \cdot 10^{-1}$	$1.3 \cdot 10^{-4}$

Table 2: Max. and Avg. Error for different values of N (number of points in the grid) with $a = 1, d = 1 \cdot 10^{-4}, r = 1000$

Table 2 shows the maximum and mean absolute errors committed by the different 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 over-conservative about 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 1 illustrates the solution for $r = 10$. For $r = 1000$ the solution look like a traveling step. Thus, a very small error in the wave speed provokes a very large error in the value of u_i when the wave passes through the i–th point of the grid.

4.2 Second scenario – Variation of the grid size Δx without diffusion

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

Figure 3 compares the CPU time of DASSL, Radau5, DOPRI and LIQSS2. Table 3 summarizes the results together with the number of scalar function evaluations.

The results here are similar to those with $d = 1 \cdot 10^{-4}$, 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 handles in a very efficient way.

Consequently, when $N = 10000$, LIQSS2 is about 30 times faster than DOPRI, 38 times faster than DASSL and 98 times faster than Radau.

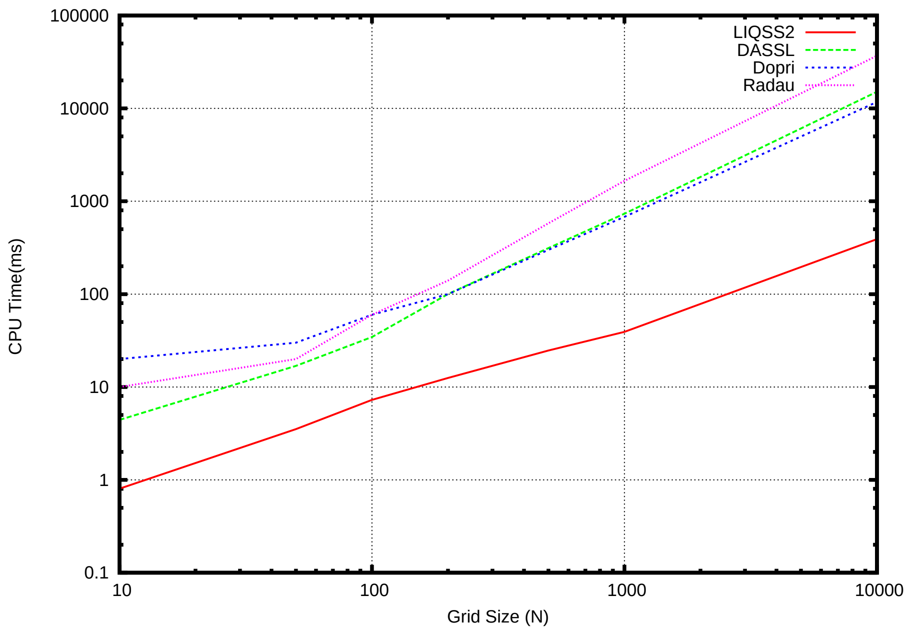

Figure 3: CPU time (ms) vs N (number of points in the grid) with $a = 1, d = 0, r = 1000$

	LIQSS2		DASSL		DOPRI		Radau5	
N	time	fun. eval.	time	fun. eval.	time	fun. eval.	time	fun. eval.
10	$8.54 \cdot 10^{-1}$	$6.14 \cdot 10^3$	$3.78 \cdot 10^0$	$8.47 \cdot 10^3$	$2.00 \cdot 10^1$	$1.88 \cdot 10^5$	$1.00 \cdot 10^1$	$1.02 \cdot 10^4$
50	$1.46 \cdot 10^0$	$2.81 \cdot 10^4$	$1.61 \cdot 10^1$	$1.02 \cdot 10^5$	$3.00 \cdot 10^1$	$1.06 \cdot 10^6$	$3.00 \cdot 10^1$	$1.74 \cdot 10^5$
100	$8.30 \cdot 10^0$	$5.92 \cdot 10^4$	$4.49 \cdot 10^1$	$3.12 \cdot 10^5$	$6.00 \cdot 10^1$	$2.46 \cdot 10^6$	$6.00 \cdot 10^1$	$5.02 \cdot 10^5$
200	$1.28 \cdot 10^1$	$1.04 \cdot 10^5$	$9.79 \cdot 10^1$	$8.70 \cdot 10^5$	$1.20 \cdot 10^2$	$5.16 \cdot 10^6$	$1.50 \cdot 10^2$	$1.57 \cdot 10^6$
500	$2.33 \cdot 10^1$	$2.70 \cdot 10^5$	$3.17 \cdot 10^2$	$2.74 \cdot 10^6$	$3.40 \cdot 10^2$	$1.65 \cdot 10^7$	$5.80 \cdot 10^2$	$6.06 \cdot 10^6$
1000	$4.23 \cdot 10^1$	$5.49 \cdot 10^5$	$7.44 \cdot 10^2$	$5.90 \cdot 10^6$	$6.70 \cdot 10^2$	$3.54 \cdot 10^7$	$1.17 \cdot 10^3$	$1.19 \cdot 10^7$
10000	$3.99 \cdot 10^2$	$6.58 \cdot 10^6$	$1.51 \cdot 10^4$	$1.04 \cdot 10^8$	$1.19 \cdot 10^4$	$6.43 \cdot 10^8$	$3.93 \cdot 10^4$	$4.23 \cdot 10^8$

Table 3: CPU time(ms) and number of function evaluations for different values of N (number of points in the grid) with $a = 1, d = 0, r = 1000$

4.3 Third scenario – Variation of reaction term r

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

Figure 4 compares the CPU time of DASSL, Radau5, DOPRI and LIQSS2 as r grows. Table 4 summarizes the results together with the number of scalar function evaluations. Errors are not reported as they are similar to those of the first scenario.

In this scenario LIQSS2 shows a noticeable advantage over the 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.

However, the other methods have problems. 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 requires more steps to converge.

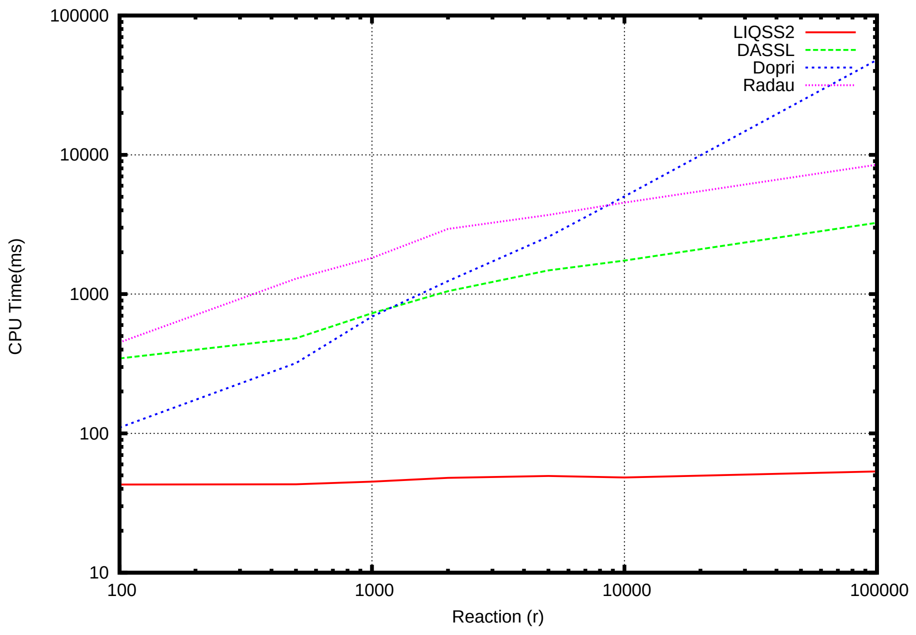

Figure 4: CPU time (ms) vs. r with $a = 1, d = 1 \cdot 10^{-4}, N = 1000$ points

	LIQSS2		DASSL		DOPRI		Radau5	
r	time	fun. eval.	time	fun. eval.	time	fun. eval.	time	fun. eval.
100	$3.35 \cdot 10^1$	$5.93 \cdot 10^5$	$3.53 \cdot 10^2$	$1.94 \cdot 10^6$	$1.10 \cdot 10^2$	$5.38 \cdot 10^6$	$5.80 \cdot 10^2$	$5.50 \cdot 10^6$
500	$4.34 \cdot 10^1$	$5.45 \cdot 10^5$	$4.79 \cdot 10^2$	$3.68 \cdot 10^6$	$3.10 \cdot 10^2$	$1.61 \cdot 10^7$	$9.90 \cdot 10^2$	$9.18 \cdot 10^6$
1000	$4.66 \cdot 10^1$	$6.05 \cdot 10^5$	$7.41 \cdot 10^2$	$5.64 \cdot 10^6$	$7.00 \cdot 10^2$	$3.54 \cdot 10^7$	$1.29 \cdot 10^3$	$1.23 \cdot 10^7$
2000	$4.49 \cdot 10^1$	$6.51 \cdot 10^5$	$1.05 \cdot 10^3$	$1.00 \cdot 10^7$	$1.21 \cdot 10^3$	$6.37 \cdot 10^7$	$2.51 \cdot 10^3$	$2.41 \cdot 10^7$
5000	$5.08 \cdot 10^1$	$6.84 \cdot 10^5$	$1.50 \cdot 10^3$	$1.71 \cdot 10^7$	$2.60 \cdot 10^3$	$1.41 \cdot 10^8$	$3.58 \cdot 10^3$	$3.52 \cdot 10^7$
10000	$5.25 \cdot 10^1$	$7.04 \cdot 10^5$	$1.75 \cdot 10^3$	$2.14 \cdot 10^7$	$5.25 \cdot 10^3$	$2.78 \cdot 10^8$	$4.39 \cdot 10^3$	$4.49 \cdot 10^7$
100000	$5.64 \cdot 10^1$	$7.68 \cdot 10^5$	$3.29 \cdot 10^3$	$5.12 \cdot 10^7$	$4.68 \cdot 10^4$	$2.71 \cdot 10^9$	$8.93 \cdot 10^3$	$9.43 \cdot 10^7$

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

In consequence, in the last case analyzed ($r = 100000$), LIQSS2 is about 60 times faster than DASSL, 160 times faster than Radau and 830 times faster than DOPRI.

4.4 Fourth scenario – Variation of diffusion term d

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

Figure 5 shows the computational costs for different diffusion terms d while Table 5 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 of 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 it is analyzed in (Migoni et al., 2013).

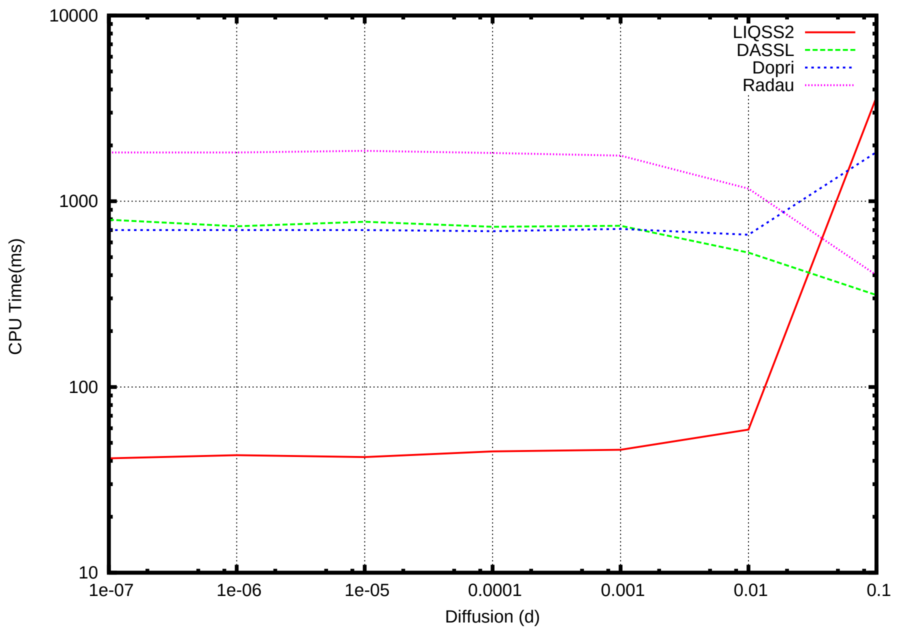

Figure 5: CPU time(ms) comparison for different magnitudes of diffusion d - $a = 1, N = 1000$ points, $r = 1000$

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

Table 5: CPU time(ms) and number of function evaluations for different d - $a = 1, N = 1000$ points, $r = 1000$

5 CONCLUSIONS

In this work we studied the application of Quantization Based Integration methods for semi–discretized Advection-Diffusion-Reaction (ADR) problems.

We compared the second order Linearly Implicit QSS (LIQSS2) method against widely used classic numerical integration methods such as DASSL, Radau and DOPRI.

We conclude that:

- LIQSS2 is a better option than classic numerical integration methods when the relation between the advection and the diffusion is large (i.e., large Péclet Numbers).
- Provided that the diffusion term is small, LIQSS2 shows an increasing advantage over the other methods as the size N grows as 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.
- In most cases, LIQSS2 performed at least 10 times faster than classic solvers.

Taking into account these remarks, we corroborated that LIQSS2 is a good alternative to integrate ADR equations.

However, this work was limited to a special case of a one dimensional ADR equation, with particular initial states and border conditions, and semidiscretized with the MOL using first order finite differences.

Future work should corroborate these results on a more general context, considering:

- More sophisticated models, including two and three dimensional problems with realistic initial states and boundary conditions, such as environmental geochemistry, groundwater and pollution.

- 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.

REFERENCES

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.

Álvarez J. and Rojo J. An improved class of generalized Runge-Kutta methods for stiff problems. Part II: The separated system case . *Applied Mathematics and Computation*, 159(3):717 – 758, 2004. ISSN 0096-3003. doi:10.1016/j.amc.2003.09.023.

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, pages 73–82. Vienna, Austria, 2006.

Bergero F., Floros X., Fernández J., Kofman E., and Cellier F.E. Simulating Modelica models with a Stand–Alone Quantized State Systems Solver. In *9th International Modelica Conference*. 2012.

Bergero F. and Kofman E. PowerDEVS: A Tool for Hybrid System Modeling and Real Time Simulation. *Simulation*, 87:113–132, 2011. ISSN 0037-5497. doi:http://dx.doi.org/10.1177/0037549710368029.

Brenan K.E., Campbell S.L., and Petzold L.R. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. Society for Industrial and Applied Mathematics, 1995. doi:10.1137/1.9781611971224.

Brown P.N., Hindmarsh A.C., and Petzold L.R. Using Krylov methods in the solution of large-scale differential-algebraic systems. *SIAM Journal on Scientific Computing*, 15(6):1467–1488, 1994.

Butcher J. *Numerical Methods for Ordinary Differential Equations*, pages i–xiv. John Wiley & Sons, Ltd, 2005. ISBN 9780470868270. doi:10.1002/0470868279.fmatter.

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*, pages 27–37. 2012.

Cellier F. and Kofman E. *Continuous System Simulation*. Springer, New York, 2006.

D'Abreu M. and Wainer G. M/CD++: Modeling continuous systems using Modelica and DEVS. In *Proceedings of MASCOTS 2005*, pages 229 – 236. Atlanta, GA, 2005.

Dormand J. and Prince P. A family of embedded Runge-Kutta formulae . *Journal of Computational and Applied Mathematics*, 6(1):19 – 26, 1980. ISSN 0377-0427.

Esquembre F. Easy Java Simulations: a software tool to create scientific simulations in Java. *Computer Physics Communications*, 156(1):199–204, 2004.

Fernandez J. and Kofman E. Implementación autónoma de métodos de integración numérica QSS. Technical Report, FCEIA - UNR, Rosario, Argentina, 2012.

Fritzson P. and Engelson V. Modelica - A Unified Object-Oriented Language for System Modelling and Simulation. In *ECOOP*, pages 67–90. 1998.

Gilding B.H. and Kersner R. Travelling waves in nonlinear diffusion-convection-reaction. Memorandum 1585, Department of Applied Mathematics, University of Twente, Enschede, 2001.

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

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

Hundsdorfer W. and Verwer J.G. *Numerical Solution of Time-Dependent Advection-Diffusion-Reaction Equations*. Springer, 2003.

John V. and Schmeyer E. Finite element methods for time-dependent convection-diffusion-reaction equations with small diffusion. *Computer Methods in Applied Mechanics and Engineering*, 198(3-4):475 – 494, 2008. ISSN 0045-7825. doi:http://dx.doi.org/10.1016/j.cma.2008.08.016.

Kleefeld B. and Martín-Vaquero J. SERK2v2: A new second-order stabilized explicit Runge-Kutta method for stiff problems. *Numerical Methods for Partial Differential Equations*, 29(1):170–185, 2013. ISSN 1098-2426. doi:10.1002/num.21704.

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

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

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

Kofman E. Relative Error Control in Quantization Based Integration. *Latin American Applied Research*, 39(3):231–238, 2009.

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

Migoni G., Bortolotto M., Kofman E., and Cellier F.E. Linearly implicit quantization-based integration methods for stiff ordinary differential equations . *Simulation Modelling Practice and Theory*, 35:118 – 136, 2013.

Migoni G., Kofman E., and Cellier F. Quantization-Based New Integration Methods for Stiff ODEs. *Simulation: Transactions of the Society for Modeling and Simulation International*, 88(4):387–407, 2012.

Muzy A., Jammalamadaka R., Zeigler B.P., and Nutaro J.J. The Activity-tracking paradigm in discrete-event modeling and simulation: The case of spatially continuous distributed systems. *Simulation*, 87(5):449–464, 2011.

Petzold L.R. Description of DASSL: A differential/algebraic system solver. Technical Report, Sandia National Labs., Livermore, CA (USA), 1982.

Petzold L.R. A description of DASSL: a differential/algebraic system solver. In *Scientific computing (Montreal, Quebec, 1982)*, pages 65–68. IMACS, New Brunswick, NJ, 1983.

Portapila M. and Power H. A convergence analysis of the performance of the drm-md boundary integral approach. *International Journal for Numerical Methods in Engineering*, 71(1):47–65, 2007. ISSN 1097-0207. doi:10.1002/nme.1936.

Quesnel G., Duboz R., Ramat E., and Traoré M. VLE: a multimodeling and simulation environment. In *Proceedings of the 2007 Summer Computer Simulation Conference*, pages 367–374. San Diego, California, 2007.

Savcenco V., Hundsdorfer W., and Verwer J. A multirate time stepping strategy for stiff ordinary differential equations. *BIT Numerical Mathematics*, 47(1):137–155, 2007.

Schiesser W. *The numerical method of lines: integration of partial differential equations*. Academic Press, 1991. ISBN 9780126241303.

Sommeijer B., Shampine L., and Verwer J. Rkc: An explicit solver for parabolic pdes. *Journal of Computational and Applied Mathematics*, 88(2):315 – 326, 1998. ISSN 0377-0427. doi: 10.1016/S0377-0427(97)00219-7.

Theeraek P., Phongthanapanich S., and Dechaumphai P. Solving convection-diffusion-reaction equation by adaptive finite volume element method. *Mathematics and Computers in Simulation*, 82(2):220 – 233, 2011. ISSN 0378-4754.

Verwer J., Sommeijer B., and Hundsdorfer W. RKC time-stepping for advection-diffusion-reaction problems. *Journal of Computational Physics*, 201(1):61 – 79, 2004. ISSN 0021-9991. doi:10.1016/j.jcp.2004.05.002.

Wolke R. and Knoth O. Implicit–explicit Runge–Kutta methods applied to atmospheric chemistry-transport modelling. *Environmental Modelling & Software*, 15(6):711–719, 2000.

Zeigler B., Praehofer H., and Kim T.G. *Theory of Modeling and Simulation - Second Edition*. Academic Press, 2000.