Politecnico di Milano
Scuola di Ingegneria Industriale e dell'Informazione

Corso di Laurea Magistrale in Ingegneria Matematica

Application of the Multirate TR-BDF2 method to the time discretization of nonlinear conservation laws

Ludovica Delpopolo Carciopolo

Relatore:
Prof. Luca Bonaventura
Correlatore:
Dott.ssa Anna Scotti

Anno accademico 2014-2015

A model must be wrong, in some respects, else it would be the thing itself. The trick is to see where it is right.

Henry Bent

Ai miei genitori...

Abstract

In this thesis we focused on the *multirate TR-BDF2 method* to discretize in time hyperbolic non linear conservation laws and non linear ordinary differential equations where components have different characteristic time scales. Multirate methods use different time steps for each component to reduce the computational costs. The study of numerical methods for the resolution of partial differential equations of non linear hyperbolic type represents a vast field of numerical mathematics. These equations are used to describe many physical phenomena. To discretize in space we use the Finite Volume method with the Rusanov numerical flux. A `C++` code has been developed to demonstrate the efficiency and accuracy of the multirate method. The algorithm is able to solve, with the single rate or the multirate TR-BDF2 method, zero-dimensional geochemistry problems. It is also able to compute the solution of conservation laws in the case they have a monotone flux.

Keywords: multirate TR-BDF2 method, conservation laws, geochemistry problems, Finite Volume methods.

Sommario

Questo lavoro di tesi ha come oggetto di studio il *metodo multirate TR-BDF2* per la discretizzazione temporale di leggi di conservazione iperboliche non lineari. I metodi multirate usano differenti passi temporali per ogni componente del sistema in modo da cercare di ridurre i costi computazionali. Lo studio di metodi numerici per la risoluzione di equazioni a derivate parziali iperboliche non lineari rappresenta un vasto campo della matematica numerica. Tali equazioni, infatti, sono usate per descrivere molte applicazioni in campo fisico. Per discretizzarle in spazio è stato utilizzato il metodo ai Volumi Finiti dove, come flusso numerico, è stato utilizzato il flusso di Rusanov. Il tutto è stato implementato in un codice `C++` per verificare l'efficienza e l'accuratezza del metodo multirate. L'algoritmo implementato è in grado risolvere, con il metodo multirate o single rate TR-BDF2, dei problemi zero dimensionali geo-chimici. È anche in grado calcolare la soluzione di leggi di conservazione nel caso in cui esse abbiano un flusso monotono.

Parole chiave: metodo multirate TR-BDF2, leggi di conservazione, problemi geo-chimici, metodi ai Volumi Finiti.

Contents

List of Figures

List of Tables

Introduction

There is an ever increasing need to develop numerical methods capable of solving large-scale real-world problems. Many of these problems arise in the geochemistry, biological and engineering sciences, such that we must use numerical models, because the problems cannot be solved analytically. In other words we seek to find numerical methods able to approximate the solution of a problem that can be modeled as a system of ordinary differential equations (ODE), or as a partial differential equation (PDE), or system of PDEs. With a standard single rate time integration approach the entire system evolves with the same time step. It is obvious that integrating the slow component on the same time scale as the fast component requires unnecessary computational effort. Our aim is to reduce these computational costs by using a multirate time stepping strategy that captures the different time scales of the components. Multirate methods use different time steps for each component while the coupling influences between the components are interpolated.

The idea of multirate methods was first proposed in [And79]. In the first approaches the system is partitioned a priori, based on the knowledge of the specific problem. In these cases the set of the slow components is integrated first using extrapolated values of the fast components. Afterwards the second set is integrated using the interpolated values of the first, as explained in [GW84]. In [SHV07] a self-adjusting time-stepping strategy is proposed. A tentative global step is taken for all the components and only the ones with errors that are larger than the tolerance are refined by halving the time-step. The components that are not refined, but that are required at the refinement stage of other components, are interpolated. In [Fok15] a multirate algorithm based on the 4th order accurate Runge Kutta method is proposed. It is assumed that the active components are coupled only to a small subset of the latent components. The partitioning and time step refinement strategy is based on the intrinsic error estimates of the Runge Kutta method. The multirate algorithm has been tested on some simple scalar PDEs. Automatic partitioning methods for multirate algorithms have also been analyzed in [VTB$^+$07]. Different techniques for optimizing the partitioning and time stepping strategy, based on the error estimates, have been discussed.

We will use the multirate TR-BDF2 method proposed by Ranade in [Ran16] where the system is integrated by means of the TR-BDF2 method, which is a second order, one step, L-stable, implicit method. Its appealing features also make it an interesting candidate for multirate approaches targeted at large scale stiff problems or for the time discretization of partial differential equations. We will focus on the analysis of two specific applications. First, a zero-dimensional geochemistry problem is considered. In a geochemistry system many different types of species are present and they can react with each others or stay constant. Some of them react very slowly, others very fast, so that in the system different times scales are present. Then, we will apply the self adjusting multirate TR-BDF2 method proposed by Ranade to some hyperbolic equations, comparing the single rate and the multirate methods with respect to the computational cost point of view and checking the behavior of the solutions with respect to positivity. The emphasis of the thesis is on the time-integration aspects of numerical modeling and for this reason we restrict our spatial discretization to simple Finite volume methods with uniform grids. We have implemented a `C++` code to test the multirate TR-BDF2 algorithm on our applications. A part of this work has been developed during a four-month internship at Istitute Français du Pétrole et Ènergies Nouvelles (IFPEN). The research group at IFPEN were particularly interested in testing the performance of Ranade's algorithm on geochemistry and hyperbolic problems. These kinds of problems can have many applications, such as the CO_2 geological storage. We tested the multirate method on a set of different complexity problem: from a linear advection problem to a Buckley Leverett equation, where a strong non linearity is present.

In Chapter 1 we review some basic concepts of ODE numerical integration theory, in particular, the stability, the accuracy and the monotonicity of numerical methods. We present the TR-BDF2 method with some details of implementation. In Chapter 2 we introduce shortly the properties of the conservation laws and of their solutions, focusing on nonlinear conservation laws. We also describe in a brief way the finite volume method used in this work. In Chapter 3 we explain the multirate algorithm, its time step refinement strategy and its partitioning criterion. In Chapter 4 we validate our code and we present the results with some considerations on the method used to solve a specific problems. It is also present, in Appendix A, the code structure of the multirate method for different problems.

Chapter 1

Numerical methods for ODE systems

Mathematical models of physical problems often consist in a system of ordinary differential equations (ODE) or systems of partial differential equations (PDE) that, after space discretization, give rise to a large ODE system. In this chapter, we review the standard notation and terminology that will be used in the rest of the thesis. We also introduce the basic theory of ODE and some different types of numerical methods employed for the numerical solution. For a more detailed introduction see e.g. [Ise96]-[HNW87]-[Lam91].

1.1 ODE systems

In this section we formulate the mathematical problem to be solved and we review the conditions for existence and uniqueness of solutions. The systems of ordinary differential equations are called initial values problems, also known as the Cauchy problem. These systems can be written as

$$\begin{cases} \mathbf{y}'(t) = \mathbf{f}(\mathbf{y}(t), t) & t \in [0, T] \\ \mathbf{y}(0) = \mathbf{y}_0 \end{cases} \tag{1.1}$$

where $\mathbf{y}(t) \in \mathbb{R}^d$ and $\mathbf{f}(\mathbf{y}, t) : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$.

Theorem 1. *The solution of* (1.1) *exists and is unique if the function* $\mathbf{f}(\mathbf{y}, t)$ *is continuous respect to* $t \in [0; T]$ *and Lipschitz continuous respect to* $\mathbf{y}$ *i.e.*

$$||\mathbf{f}(\mathbf{y}_1, t) - \mathbf{f}(\mathbf{y}_2, t)|| \leq L||\mathbf{y}_1 - \mathbf{y}_2||, \qquad \forall \mathbf{y}_1, \mathbf{y_2} \in \mathbb{R}^d, \quad t \in [0, T] \tag{1.2}$$

where $L > 0$ *is called the Lipschitz constant.*

It is well known that a sufficient condition for a function to be Lipschitz continuous with respect to a certain argument, is to be continuously differentiable with respect to that argument [Ise96].

1.2 Stability

Any numerical method must preserve the stability property. A problem with unique solution which continuously depends on the data is called well-posed problem [Lam91]. We consider a solution of problem (1.1) with a perturbation to the right hand side and the initial condition as follow

$$\begin{cases} \mathbf{z}'(t) = \mathbf{f}(\mathbf{z}(t), t) + \boldsymbol{\delta}(t), & t \in [0, T] \\ \mathbf{z}(0) = \mathbf{z}_0 + \boldsymbol{\delta} \end{cases} \tag{1.3}$$

Definition 1. *For two given perturbations to* (1.3) $(\boldsymbol{\delta}(t); \boldsymbol{\delta})$ *and* $(\boldsymbol{\xi}(t); \xi)$*, denote the corresponding solutions* $\mathbf{z}(t)$ *and* $\mathbf{x}(t)$*. Then, if there exists an* $S > 0$ *such that* $\forall t \in [0, T]$*:*

$$||\boldsymbol{\delta}(t) - \boldsymbol{\xi}(t)|| < \epsilon \qquad ||\boldsymbol{\delta} - \boldsymbol{\xi}|| < \epsilon$$

implies

$$||\mathbf{z}(t) - \mathbf{x}(t)|| < S\epsilon$$

for sufficiently small $\epsilon > 0$*, then the initial value problem* (1.1) *is said to be* ***totally stable****.*

It is important that the problem is totally stable because numerical methods introduce errors. These errors are equivalent to perturbations of the system, so that if the condition of stability is not satisfied the numerical method will not produce a reasonable solution. Another stability property is the Lyapunov stability, it is concerned with the stability of solutions near an equilibrium point.

Definition 2. ***(Lyapunov stability)*** *Given an autonomous system* $\mathbf{y}'(t) = \mathbf{f}(\mathbf{y}(t))$ *such that* $\mathbf{f}(0) = \mathbf{0} \in \mathbb{R}^d$*; the origin is called a stable solution if for any* $\epsilon > 0$ *there is a* $\delta > 0$ *such that* $||\mathbf{y}(0)|| \leq \delta$ *implies* $||\mathbf{y}(t)|| \leq \epsilon \quad \forall t \geq 0$*.*

For the linear, homogeneous, constant coefficient system $\mathbf{y}' = \mathbf{A}\mathbf{y}$ the origin is stable if and only if for all the eigenvalues of $\mathbf{A}$: $\mathcal{R}(\lambda_i(\mathbf{A})) < 0$. In this case of linear constant coefficient systems, the origin is not only stable, but attractive, in the sense that any solution of the system with any initial condition will tend asymptotically to zero.

1.3 Stiffness

ODE systems that describe dynamics with different time scales are often called stiff. A precise definition of this concept is not easy, a standard definition, given by [Lam91], is

Definition 3. *Consider a linear Cauchy problem* $\mathbf{y}' = \mathbf{A}\mathbf{y}$ *such that the eigenvalues* $\lambda(\mathbf{A}) = \mathcal{R}(\lambda) + i\mathcal{I}(\lambda)$ *have negative real part. The Cauchy problem is called* ***stiff*** *if*

$$\frac{\max_i |\mathcal{R}(\lambda_i)|}{\min_i |\mathcal{R}(\lambda_i)|} >> 1. \tag{1.4}$$

Explicit methods are very inefficient to the point of being unusable when they are used on stiff problems. One is forced to use implicit methods, which can solve the problem but are computationally more expensive per time step. Understanding the phenomenon of stiffness could, therefore, help to improve the efficiency of solvers. Several techniques for automatic detection of stiffness have been developed. The phenomenon of stiffness was first identified in the paper of Curtiss and Hirschfelder [CH52]. J.D. Lambert has devoted an entire chapter of his book [Lam91] to stiffness, in which he gives several definitions of stiffness along with counterexamples. There are also other authors who, rather than defining the phenomenon of stiffness, have stated that one of the effects of stiffness is that explicit methods do not work [WH91]. One of the most prolific author on the subject of stiffness has been L.F. Shampine. He had proposed an interesting idea for stiffness detection tests based on the stability region of different methods [Sha77]. The idea is to select two methods of which one has a larger stability region. Stiffness or the lack of it is detected by taking a test step and comparing the results. Another interesting paper is [SG79], in this paper Shampine and Gear make several interesting observations on stiffness. Firstly, they state that the stiffness of a problem can change with time; particularly, and perhaps somewhat counter-intuitively, according to them a problem is not stiff when there is a rapid transient. They have also discussed with some detail why explicit methods do not perform very well on stiff problems, using the forward Euler method to illustrate the issue. Continuing in this line of research, Shampine proposed type insensitive codes, which detect stiffness and automatically switch between Newton iteration and fixed-point iteration as the simulation progresses [Sha81]. The most interesting point in this paper is that a notion of stiffness for implicit A-stable methods is introduced. This so called IA-stiffness forces a code to use the more expensive Newton-iteration rather than the cheaper fixed point iteration. The paper also contains a discussion on the conditions for switching between the two iterations. The basic idea is that a problem is stiff at a certain point in its trajectory if the local Jacobian is large. A sufficient condition on the step size for an explicit method to avoid instability is

$$h||\mathbf{J}|| < 1 \tag{1.5}$$

In the following, we will use the above condition as an indication of stiffness.

1.4 Numerical methods for ODE systems

For the systems of ODEs numerical methods are used to find the solutions. Numerical methods are formulated to compute approximations of the solution at discrete points in time. The solution $\mathbf{y}(t)$ is approximated by values $\mathbf{u}^n$ at time instants t_n, $n = 0, 1, 2, ..., N$ with $t_N = T$.

There are different ways to classify numerical methods for ODEs system. One is the difference between explicit and implicit methods. If the formula for the computation of $\mathbf{u}^{n+1}$ only involves previously computed approximations such as $\mathbf{u}^n$; the method is called explicit. If the formula for the computation involves $\mathbf{u}^{n+1}$, then such methods are called implicit methods. In this case, to compute the solution we need, to solve a nonlinear system. Explicit methods are thus much cheaper per time step than implicit methods. However, implicit methods have in general much better stability properties which justify their use particularly for stiff ODEs.

Another distinction is between multistep and single step methods. If in the formula are present previously computed values $(t_{n-i}; \mathbf{u}^{n-i})$; $i = 1, 2, .., k$ to compute the approximate solution $\mathbf{u}^{n+1}$ then such methods are called multistep methods. They are presented as

$$\sum_{j=0}^{k} \alpha_j \mathbf{u}^{n+1-j} = h \sum_{j=0}^{k} \beta_j \mathbf{f}(t_{n+1-j}, \mathbf{u}^{n+1-j}) \tag{1.6}$$

It can be seen that if $\beta_0 \neq 0$ the method is implicit. Single step methods require only the solution at the previous time step t_n. They can be written as

$$\mathbf{u}^{n+1} = \mathbf{u}^n + h\boldsymbol{\psi}(t_n; \mathbf{u}^n; t_{n+1}; \mathbf{u}^{n+1}) \tag{1.7}$$

where the function $\boldsymbol{\psi}$ defines the numerical method. Also in this case, if the method involves the approximate solution at time t_{n+1} it will be an implicit method.

1.5 Consistency and convergence

The numerical methods are used to compute an approximate solution of the system and it is important to know how the numerical solution accurate is. During a simulation we can commit two types of errors: errors due to round-off and errors due to the discretization. For this reason it is important to introduce the notion of local truncation error and of global error.

Definition 4. *(**Local Truncation Error**) In a one-step method given by* (1.7) *from time t_n to time t_{n+1}, the local truncation error is defined as*

$$\epsilon_n = ||\mathbf{y}(t_{n+1}) - \mathbf{y}(t_n) - h\boldsymbol{\psi}(t_n; \mathbf{y}(t_n); t_{n+1}; \mathbf{y}(t_{n+1})||. \tag{1.8}$$

Local truncation error is the error in the solution computed by a single step of the numerical method for which the true solution values were plugged in.

Definition 5. *A numerical method is **consistent** if the local truncation error goes to zero when $h \to 0$.*

Definition 6. *(**Order of consistency**) A numerical method is said to have an order of consistency p if the local truncation error defined by* (1.8) *satisfies:*

$$\epsilon \leq Ch^{p+1} \tag{1.9}$$

where $C > 0$ is a constant.

Definition 7. *(**Global error**) The global error in a numerical solution $\mathbf{u}^n$ to the initial value problem* (1.1) *at time instants t_n, $n = 1, 2, ..., N$ is defined as*

$$\tau(h) = \max_{n=1,...,N} ||\mathbf{y}(t_n) - \mathbf{u}^n|| \tag{1.10}$$

Another important definition is the order of convergence that tells us how fast a method converges to the exact solution as the time steps decrease.

Definition 8. *(**Order of convergence**) A numerical method is said to have an order of convergence p, if the global error in the numerical solution as defined in* (1.10)*, when a numerical method is applied to* (1.1) *satisfies*

$$\tau(h) \leq Ch^p \tag{1.11}$$

with C>0 is a constant.

For multi-step methods, consistency does not imply convergence. For one-step methods, however, convergence is guaranteed if the method is consistent [Lam91]. Convergence of multistep method instead depends on zero-stability. We now introduce the notion of stability for numerical methods. There are many types of stability definitions depending on the problem and the corresponding requirements imposed on the numerical solution computed by the method.

Definition 9. *(**Zero stability of a numerical method**) Let $\mathbf{z}^n$ and $\mathbf{x}^n$ be two different numerical solution obtained by the numerical method* (1.6) *applied to the perturbed* *problem* (1.3) *for two perturbations* $\boldsymbol{\delta}^n$ *and* $\boldsymbol{\xi}^n$. *Then if there exists an* $S > 0$ *such that,* $\forall n$

$$||\boldsymbol{\delta}^n - \boldsymbol{\xi}^n|| < \epsilon$$

implies

$$||\mathbf{z}^n - \mathbf{x}^n|| < S\epsilon$$

for sufficiently small $\epsilon > 0$, *then the numerical method* (1.6) *is said to be* ***zero-stable***.

As we have seen before, a sufficient condition for a linear system to be stable is that the eigenvalues should have negative real part. If we introduce the following scalar linear problem

$$\begin{cases} y'(t) = \lambda y, & t \in [0, T] \\ y(0) = y_0 \end{cases} \tag{1.12}$$

with $\lambda \in \mathbb{C}$, the analytic solution is $y(t) = y_0 e^{\lambda t}$, therefore, for $\mathcal{R}(\lambda) < 0$, we have that $y(t) \to 0$ as $t \to \infty$. We can convert this into a property of numerical solution, in the following way:

Definition 10. *Let* u^n *a numerical solution for* (1.12), *the numerical method is* ***absolutely (linearly) stable*** *if, for a given* $h > 0$,

$$\lim_{n \to \infty} ||u^n|| = 0. \tag{1.13}$$

Definition 11. *Given the linear scalar problem and a single step numerical method with fixed time step* h, *the* ***linear stability function*** *is* $\mathcal{S}(z)$, $z \in \mathbb{C}$ *such that the application of the method to the scalar problem 1.12 yields*

$$u_n = \mathcal{S}(h\lambda)^n y_0 \tag{1.14}$$

If we consider the explicit Euler method:

$$\mathbf{u}^{n+1} = \mathbf{u}^n + h\mathbf{f}(t_n, \mathbf{u}^n) \tag{1.15}$$

Applying (1.15) to (1.12) yields

$$u^{n+1} = (1 + h\lambda)u^n \quad n \geq 0,$$

thus

$$u^n = (1 + h\lambda)^n u_0. \tag{1.16}$$

Consequently, the sequence $\{u^n\}_{n=0}^{\infty}$ will converge to zero if $|1+h\lambda| < 1$. This requirement is equivalent to

$$h\lambda \in \mathbb{C}^- \quad \text{e} \quad 0 < h < -\frac{2\mathcal{R}(\lambda)}{|\lambda|^2} \tag{1.17}$$

Let us define in (1.12) $z = h\lambda$; $z \in \mathbb{C}$. The set of z in the complex plane, for which the numerical method is absolutely stable, is known as the **region of absolute stability** of the method. The intersection of this region with the x-axis is known as the *interval of absolute stability.*

Definition 12. ***(A-stable method)*** *If the region of absolute stability of a numerical method contains the entire left half of the complex plane, the method is said to be A-stable.*

Definition 13. ***(L-stability)*** *A one step method is called* ***L-stable*** *if $S(z) \to 0$ as $z \to -\infty$.*

If methods with a small stability region are used, they will need small steps. For this reason, A-stable methods are preferred to compute solution of stiff problems.

A method that is L-stable is also A-stable. The above definitions of stability have as hypothesis the linearity of the problem. We are interested in nonlinear problems, for this reason in [PR74] another stability definition has been introduced. Prothero and Robinson linearize the problem, but this does not yield a general definition because the linearization of the problem is valid only locally. For more details, the reader is referred to [WH91].

1.6 Runge-Kutta methods

Runge-Kutta methods are a family of one-step methods. The well known forward-Euler, is the most basic of the Explicit Runge Kutta (ERK) methods. The idea of a Runge Kutta method is to approximate the derivative of the solution at that point to advance the solution in time. These methods only use the numerical solution $\mathbf{u}^n$ computed at time level n to compute the solution $\mathbf{u}^{n+1}$ at the next time level $n+1$, but in order to achieve higher order accuracy they perform a number of intermediate updates, called stages. A general Runge Kutta method $RK(\mathbf{A}, \mathbf{b})$ with s stages for a Cauchy problem (1.1), can be written as follow

$$\begin{aligned} \mathbf{u}^{n+1} &= \mathbf{u}^n + h_n \sum_{i=1}^{s} b_i \mathbf{k}_i \\ \mathbf{k}_i &= \mathbf{f}\left(t_n + c_i h_n, \mathbf{u}^n + \sum_{j=1}^{s} a_{ij}\mathbf{k}_j\right) \qquad i = 1, ..., s \end{aligned} \tag{1.18}$$

The structure of the matrix A determines if the method is implicit or explicit. If the matrix A is such that $a_{ij} = 0 \quad \forall j \geq i$ then the method is explicit. Each stage requires only the previously computed stages, that is used in other function evaluation, to compute the next stage. For an Implicit Runge-Kutta (IRK) method with a full coefficient matrix A, each step requires the solution of a nonlinear system. When A is a lower triangular matrix, each stage depends only on the previously computed stages and itself, it means that a step requires the solution of s different smaller nonlinear systems. These methods are known as Diagonally Implicit Runge Kutta (DIRK). The computation can be further simplified by letting all the diagonal elements be the same. In this case, the iteration matrix for each stage is the same and thus, this type of methods known as Singly Diagonally Implicit Runge Kutta (SDIRK) methods, avoids to repeat the assembly and factorization of iteration matrices. Also of interest are Singly Diagonally Implicit Runge Kutta methods with an Explicit first stage (ESDIRK).

The Runge Kutta parameters that identify a methods are:

$$\mathbf{c} = [c_1, c_2, ..., c_s]^T \qquad \mathbf{b} = [b_1, b_2, ..., b_s]^T \qquad A = \{a_{ij}\}$$

the methods can be represented with a table called *Butcher tableau.*

$$\begin{array}{c|cccc}
c_1 & a_{11} & a_{12} & \cdots & a_{1s} \\
c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\
\hline
 & b_1 & b_2 & \cdots & b_s
\end{array}$$

Table 1.1: Butcher tableau of a RK method

The Butcher tableaus of ERK, DIRK and SDIRK methods would have the form as shown in tables (1.2).

The coefficients in the Butcher tableau have to be chosen carefully to obtain a certain consistency order.

Theorem 2. ***(Order conditions for generic Runge Kutta method)*** *Any Runge Kutta method such that $\sum_j a_{ij} = c_j$ is*

1. *of order one if and only if $\sum_i b_i = 1$,*

2. *of order two if and only if 1. holds and $\sum_i b_i c_i = \frac{1}{2}$,*

3. *of order three if and only if 1. and 2. hold and $\sum_i b_i c_i^2 = \frac{1}{3}$ $\sum_{i,j} b_i a_{ij} c_j = \frac{1}{6}$.*

For higher orders more details are present in [But63].

$$\begin{array}{c|cccc}
0 & 0 & 0 & \cdots & 0 \\
c_2 & a_{21} & 0 & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
c_s & a_{s1} & a_{s2} & \cdots & 0 \\
\hline
 & b_1 & b_2 & \cdots & b_s
\end{array}$$

(a) ERK methods

$$\begin{array}{c|cccc}
c_1 & \gamma_1 & 0 & \cdots & 0 \\
c_2 & a_{21} & \gamma_2 & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
c_s & a_{s1} & a_{s2} & \cdots & \gamma_s \\
\hline
 & b_1 & b_2 & \cdots & b_s
\end{array}$$

(b) DIRK methods

$$\begin{array}{c|cccc}
c_1 & \gamma & 0 & \cdots & 0 \\
c_2 & a_{21} & \gamma & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
c_s & a_{s1} & a_{s2} & \cdots & \gamma \\
\hline
 & b_1 & b_2 & \cdots & b_s
\end{array}$$

(c) SDIRK methods

Table 1.2: Butcher tableaus for different types of RK methods

1.7 Monotonicity property of Runge-Kutta methods

In many applications, the solution is required to remain non negative: this is the case, for example, in chemistry modeling problems where negative solutions correspond to non-physical negative concentrations. In [Kra91] an analysis of the monotonicity properties for general Runge-Kutta methods has been done. Following [Kra91], we introduce for real z the quantities

$$\begin{aligned} \mathbf{A}(z) &= \mathbf{A}(\mathbf{I} - z\mathbf{A})^{-1} & \mathbf{b}^T(z) &= \mathbf{b}^T(\mathbf{I} - z\mathbf{A})^{-1} \\ \mathbf{e}(z) &= (\mathbf{I} - z\mathbf{A}^{-1})\mathbf{e} & \mathcal{S}(z) &= 1 + z\mathbf{b}^T(\mathbf{I} - z)\mathbf{A}^{-1}\mathbf{e} \end{aligned} \tag{1.19}$$

Definition 14. ***(Absolute monotonicity of RK)*** *An irreducible s-stage RK$(\mathbf{A}, \mathbf{b})$ is absolutely monotone at $z \in \mathbb{R}$ if $\mathbf{A} \geq 0$, $\mathbf{b} \geq 0$, $\mathbf{e} \geq 0$ and $\mathcal{S} \geq 0$ element wise.*

These notations are useful to introduce the radius of absolute monotonicity for a RK method.

Definition 15. ***(Radius of absolute monotonicity)*** *An s-stage RK with scheme $(\mathbf{A}, \mathbf{b})$ and $\mathbf{A} \geq 0$ and $\mathbf{b} \geq 0$ is characterized by its radius of absolute monotonicity defined for all z in $-r \leq z \leq 0$ as*

$$R(\mathbf{A}, \mathbf{b}) = \sup\{r : r \geq 0, \quad \mathbf{A}(z) \geq 0, \quad \mathbf{b}(z) \geq 0, \quad \mathbf{e}(z) \geq 0, \quad \mathcal{S}(z) \geq 0\} \tag{1.20}$$

In [Kra91] two results are derived to simplify practical investigation for the monotonicity property of the method. We introduce the incidence matrix $\mathbf{Inc}(\mathbf{A})$ as the matrix that contains 0 if the corresponding element in $\mathbf{A}$ is $a_{ij} = 0$, 1 if $a_{ij} \neq 0$.

Theorem 3. *For an irreducible RK $R(\mathbf{A}, \mathbf{b}) > 0$ if and only if $\mathbf{A} > 0$, $\mathbf{b} > 0$ and $\boldsymbol{Inc}(\mathbf{A}^2) \leq \boldsymbol{Inc}(\mathbf{A})$.*

Lemma 1. *For an irreducible RK $R(\mathbf{A}, \mathbf{b}) \geq r$ if and only if $\mathbf{A} \geq 0$ and $(\mathbf{A}, \mathbf{b})$ is absolutely monotone at $z = -r$.*

1.8 TR-BDF2 method

The TR-BDF2 method was originally proposed in [BCF+85]. It is a composite one step, two stages method, consisting of one stage of the trapezoidal method followed by another of the BDF2 method. The stages are so adjusted that both the trapezoidal and the BDF-2 stages use the same Jacobian matrix. This composite method has been reinterpreted in [HS96] as a one step Diagonally Implicit Runge Kutta (DIRK) method with two internal stages. The TR-BDF2 method is also in some sense the optimal method among all 2-stage DIRK methods, owing to the following properties:

1) it is first same as last (FSAL), so that only two implicit stages need to be evaluated;
2) the Jacobian matrix for both the stages is the same;
3) it has an embedded third order companion that allows for a cheap error estimator;
4) the method is strongly S-Stable;
5) all the stages are evaluated within the time step;
6) it is endowed with a cubic Hermite interpolation algorithm for dense output that yields globally C^1 continuous trajectories.

Features 3), 5) and 6) will play an important role in the multirate method proposed here. Furthermore, as shown in [GKC13], the method has an explicit second order companion with which it can be combined to form a second order implicit-explicit additive Runge Kutta method. Due to its favorable properties, it has been recently applied for efficient discretization of high order finite element methods for numerical weather forecasting in [GKC13], [TB15].

The TR-BDF2 method, considered as a composite method consisting of a step with the trapezoidal method followed by a step of the BDF2 method, can be written as

$$\begin{aligned} \mathbf{u}^{n+\gamma} &= \mathbf{u}^n + \frac{\gamma h_n}{2}\left(\mathbf{f}(t, \mathbf{u}^n) + \mathbf{f}(t, \mathbf{u}^{n+\gamma})\right) \\ \mathbf{u}^{n+1} &= \frac{1}{\gamma(2-\gamma)}\mathbf{u}^{n+\gamma} - \frac{(1-\gamma)_2}{\gamma(2-\gamma)}\mathbf{u}^n + \frac{1-\gamma}{2-\gamma}h_n\mathbf{f}(t, \mathbf{u}^{n+\gamma}) \end{aligned} \tag{1.21}$$

The TR-BDF2 method viewed as a DIRK method has the following Butcher tableau 1.3, where $\gamma = 2 - \sqrt{2}, d = \frac{\gamma}{2}$ and $w = \frac{\sqrt{2}}{4}$.

0	0	0	0
γ	d	d	0
1	w	w	d
	w	w	d
	$(1-w)/3$	$(3w+1)/3$	$d/3$

Table 1.3: The Butcher tableau of the TR-BDF2 method

The last row corresponds to the companion third order method that can be used to build a convenient error estimator. Here, the value $\gamma = 2 - \sqrt{2}$ is chosen for the two stages to have the same Jacobian matrix, as well as in order to achieve L-stability, as it will be shown shortly. It can be seen that the method has the First Same As Last (FSAL) property, i.e., the first stage of any step is the same as the last stage of the previous step. Thus, in any step, the first explicit stage need not be computed.

The implementation of the two implicit stages is done as suggested in [HS96]. The two stages according to the Butcher tableau are given by equation (1.21). Instead of iterating on the variable $\mathbf{u}^{n+1}$, we define another variable $\mathbf{z} = h\mathbf{f}(t, \mathbf{u})$ and solve for this variable by iteration. Thus, for the first implicit stage, we take $\mathbf{u}^{n+\gamma,k} = \mathbf{u}^n + d\mathbf{z}^n + d\mathbf{z}^{n+\gamma,k}$ and $\mathbf{z}^{n+\gamma,k}$ is computed by Newton iterations as

$$\begin{aligned}(\mathbf{I} - hd\mathbf{J}^n)\boldsymbol{\Delta}^k &= h_n\mathbf{f}(t_{n+\gamma}, \mathbf{u}^{n+\gamma,k}) - \mathbf{z}^{n+\gamma,k} \\ \mathbf{z}^{n+\gamma,k+1} &= \mathbf{z}^{n+\gamma} + \boldsymbol{\Delta}^k,\end{aligned}$$

where $\mathbf{J}^n = \mathbf{J}(t_n, \mathbf{u}^n)$ denotes the Jacobian of $\mathbf{f}$. Similarly, for the second implicit stage we take $\mathbf{u}^{n+1,k} = \mathbf{u}^n + w\mathbf{z}^n + w\mathbf{z}^{n+\gamma} + d\mathbf{z}^{n+1,k}$ and the Newton iteration is given by

$$\begin{aligned}(\mathbf{I} - dh\mathbf{J}^n)\boldsymbol{\Delta}^k &= h\mathbf{f}(t_{n+\gamma}, \mathbf{u}^{n+\gamma,k}) - \mathbf{z}^{n+\gamma,k} \\ \mathbf{z}^{n+1,k+1} &= \mathbf{z}^{n+1} + \boldsymbol{\Delta}^k.\end{aligned}$$

The reason for doing so is that, if the problem is stiff, a function evaluation will amplify the numerical error in the stiff components. Iterating on $\mathbf{z}$, however, ensures that it is computed to match the tolerance. For a more detailed discussion the interested reader is referred to [HS96]. An important point to be noted is that, although the TR-BDF2 method is L-stable, its 3rd order companion formula is not. Therefore, the error estimate at time level $n+1$, given by

$$\boldsymbol{\epsilon}^{n+1,*} = (b_1^* - b_1)\mathbf{z}^n + (b_2^* - b_2)\mathbf{z}^{n+\gamma} + (b_3^* - b_3)\mathbf{z}^{n+1}$$

cannot be expected to be accurate for stiff problems. To overcome this problem, it is suggested in [HS96] to modify the error estimate by considering as error estimator the quantity $\boldsymbol{\epsilon}^{n+1}$ defined as the solution of the linear system

$$(\mathbf{I} - dh\mathbf{J}^n)\boldsymbol{\epsilon}^{n+1} = \boldsymbol{\epsilon}^{n+1,*}. \tag{1.22}$$

This modification of the error estimate allows to improve it for stiff components, while preserving its accuracy in the limit of small time steps. Notice that the stability function of the TR-BDF2 method is given by

$$\mathcal{S}(z) = \frac{[1+(1-\gamma)^2]z + 2(2-\gamma)}{z^2(1-\gamma)\gamma + z(\gamma^2-2) + 2(2-\gamma)}, \tag{1.23}$$

whence it can be seen that the method is L-stable for $\gamma = 2 - \sqrt{2}$.

We report the monotonicity properties of the TR-BDF2 method studied in [HS96] as a DIRK according to the analysis of [BR15]. Following the definition (15), we find

$$\mathbf{A}(z) = \begin{bmatrix} 0 & 0 & 0 \\ \frac{\gamma}{2-\gamma z} & \frac{\gamma}{2-\gamma z} & 0 \\ \frac{1}{(2-\gamma)(2-\gamma z)\beta} & \frac{1}{(2-\gamma)(2-\gamma z)\beta} & \frac{1-\gamma}{(2-\gamma)\beta} \end{bmatrix}$$

$$\mathbf{b}^T(z) = \begin{bmatrix} \frac{2\beta - z\gamma}{4(2-\gamma)} \\ \frac{2\beta - z\gamma}{4(2-\gamma)} \\ \frac{(1-\gamma)\beta}{2\gamma} \end{bmatrix} \qquad \mathbf{e}(z) = \begin{bmatrix} 1 \\ \frac{2+\gamma z}{2-\gamma z} \\ \frac{[1+(1-\gamma)^2]z+2(2-\gamma)}{z^2(1-\gamma)\gamma+z(\gamma^2-2)+2(2-\gamma)} \end{bmatrix}$$

where $\beta = 1 - z(1-\gamma)/(2-\gamma)$. The conclusion is that the radius of absolute monotonicity of TR-BDF2 is $R(\mathbf{A}, \mathbf{b}) = \dfrac{2(2-\gamma)}{1+(1-\gamma)^2}$. It is maximized for $\gamma = 2 - \sqrt{2}$, the value that makes the TR-BDF2 method L-stable. If we substitute this value, we obtain that the radius of absolute monotonicity is $R(\mathbf{A}, \mathbf{b}) \approx 2.414$, it also the step size coefficient c for conditional monotonicity for any nonlinear problem as explained in [FS04]. The conclusion is that the TR-BDF2 method is not unconditionally monotone, in literature some variants are proposed to have unconditional monotonicity (see e.g. [BR15]).

Chapter 2

Numerical Methods for Conservation Laws

In this chapter we review some basic theory about conservation laws and some numerical methods to discretize them in space. For more details the reader is referred to [LeV92].

Hyperbolic equations arise naturally from the conservation laws of physics. In this chapter we will introduce a classical method for numerically solving such equations. These are time-dependent systems of partial differential equations (usually nonlinear). In one space dimension the equations are of this form

$$\frac{\partial}{\partial t}u(t,x) + \frac{\partial}{\partial x}f(u(t,x)) = 0 \tag{2.1}$$

Here $u : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is the conserved quantity. The flux is given by a function $f(u(t,x))$ and it is called the flux function for the conservation laws, so $f : \mathbb{R} \to \mathbb{R}$.

The simplest problem is the pure initial value problem, or Cauchy problem, in which (2.1) holds for $-\infty < x < \infty$ and $t \geq 0$. We have to specify the initial condition only

$$u(0,x) = u_0(x) \qquad -\infty < x < \infty \tag{2.2}$$

We separate the time and space discretization because we are interesting in applying multirate methods to discretize in time. The **method of lines** consist in applying a discretization scheme in space first. As a result,a system of ordinary differential equations of the form

$$\frac{d\mathbf{u}}{dt} = \mathbf{A}\mathbf{u} \tag{2.3}$$

is obtained, where $\mathbf{u}(t)$ is the vector whose components are $u_i(t)$ the unknown values at the grid point at any time t. Then one can use any Ordinary Differential Equation (ODE) solver for the time discretization.

2.1 Linear Advection Equation

We consider first the linear scalar advection equation

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} \qquad -\infty < x < \infty, \quad t \geq 0 \tag{2.4}$$

The Cauchy problem is defined by this equation together with initial conditions $u(0,x) = u_0(x)$ and the solution is

$$u(t,x) = u_0(x - at) \tag{2.5}$$

for $t \geq 0$. As time evolves, the initial data propagates unchanged to the right (if $a > 0$) or left (if $a < 0$) with velocity a. The solution is constant along each ray $x - at = x_0$, which are known as the **characteristics** of the equation. The characteristics are curve on the $t - x$ plain satisfying $x'(t) = a$, $x(0) = x_0$. Therefore, if we differentiate $u(t,x)$ along one of these curves along the characteristic, we find that

$$\frac{d}{dt}u(t,x(t)) = \frac{\partial}{\partial t}u(t,x(t)) + \frac{\partial}{\partial x}u(t,x(t))x'(t) = 0 \tag{2.6}$$

confirming that u is constant along the characteristics.

2.2 Nonlinear Conservation Laws

Now we focus on a generic scalar conservation law of the form

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = 0 \tag{2.7}$$

with initial condition $u(0,x) = u_0(x)$. Assuming $f \in \mathcal{C}^1(\mathbb{R})$, u also verify

$$\frac{\partial u}{\partial t} + f'(u)\frac{\partial u}{\partial x} = 0 \tag{2.8}$$

Also in this case we can define the characteristic curves associated to the conservation law as the solution of the differential equation. We deduce that $u(t,x(t)) = u(0;x(0)) = u_0(x(0))$ which is independent of t. This allows us to get the solution at a given point x and time t if the characteristic curve can be traced back in the t-x plane to the line $t = 0$. This is not always possible when f is nonlinear.

2.3 Weak solutions

In same cases, the characteristics associated to a non linear conservation law can cross, in which case the method of characteristics can non longer be used to find a solution. Actually when this happens the solution is no longer of class $\mathcal{C}^1$ and can even become discontinuous. Let us define the notion of weak solution:

Definition 16. *Let $u_0 \in L^\infty(\mathbb{R})$ the initial condition of the scalar conservation law (2.7), then $u \in L^\infty(\mathbb{R}^+ \times \mathbb{R})$ is* ***weak solution*** *if satisfies*

$$\int_0^T \int_{-\infty}^{+\infty} u\frac{\partial \varphi}{\partial t} + f(u)\frac{\partial \varphi}{\partial x} dt dx + \int_{-\infty}^{+\infty} u_0(x)\varphi(0,x)dx = 0 \qquad \forall \varphi \in \mathcal{C}_0^1([0,T) \times \mathbb{R}) \tag{2.9}$$

$\mathcal{C}_0^1$ is the space of function that are continuously differentiable with "compact support" it means that $\varphi(t,x)$ is identically zero outside of some bounded set, and so the support of the function lies in a compact set. The notion of weak solutions generalizes the notion of classical solutions because multiplying by φ and integriting by parts equation (2.7) we obtain (2.9).

2.4 The Rankine-Hugoniot condition

The Rankine-Hugoniot condition gives a constraint on the discontinuity along a line for the piecewise smooth solution to be a weak solution of the equation.

Theorem 4. *Assume the half space $\mathbb{R}^+ \times \mathbb{R}$ is split into two parts M_1 and M_2 by a smooth curve S parametrized by $(t, \sigma(t))$ with $\sigma \in \mathcal{C}^1(\mathbb{R}^+)$. We also assume that $u \in L^\infty(\mathbb{R}^+ \times \mathbb{R})$ and that $u_1 = u|_{M_1} \in \mathcal{C}^1(M_1)$ and $u_2 = u|_{M_2} \in \mathcal{C}^1(M_2)$ with u_1 and u_2 two classical solutions of our equation in respectively M_1 and M_2. Then u is a weak solution if and only if*

$$[u_1(t,\sigma(t)) - u_2(t,\sigma(t))]\sigma'(t) = f(u_1(t,\sigma(t))) - f(u_2(t,\sigma(t))) \qquad \forall t \in \mathbb{R}^+ \tag{2.10}$$

Relation (2.10) is called **Rankine-Hugoniot condition** where $\sigma'(t)$ is the the propagation speed of the discontinuity.

2.5 Entropy solution

There are situations in which the weak solution is not unique and an additional condition is required to capture the physically solution that instead is unique. There are different sufficient conditions to obtain the physical solution of the problem. The condition is that it should be the limiting solution of the viscous equation:

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} - \epsilon\frac{\partial^2 u}{\partial x^2} = 0 \tag{2.11}$$

for $\epsilon \to 0$. Associated to a smooth initial condition this equation has a unique smooth solution. And the physically correct solution of our conservation law, will be the unique limit of this solution (which depends on ϵ) when $\epsilon \to 0$ if the flux f is convex. This unique solution can be characterized using the notion of entropy. First of all we have to define an **entropy function** $\eta(u)$, for which another conservation law holds if the solution u is smooth. If we have a discontinuity solution the conservation laws becomes an inequality.

Suppose a function $\eta(u)$ satisfies a conservation law

$$\frac{\partial \eta(u)}{\partial t} + \frac{\partial \psi(u)}{\partial x} = 0 \tag{2.12}$$

for some **entropy flux** $\psi(u)$. For smooth u we obtain

$$\eta'(u)\frac{\partial u}{\partial t} + \psi'(u)\frac{\partial u}{\partial x} = 0 \tag{2.13}$$

If we write (2.7) as (2.8), multiply this by $\eta'(u)$ and compare with (2.13) we obtain

$$\psi'(u) = \eta'(u)f'(u) \tag{2.14}$$

For a scalar conservation law this equation admits many solutions $\psi(u)$ and $\eta(u)$. An additional condition we place on the entropy function is that it be convex, $\eta''(u) > 0$. The reason is that if we look at equation (2.11) we can derive the corresponding evolution equation for the entropy, multiplying it by $\eta'(u)$ we obtain

$$\frac{\partial \eta(u)}{\partial t} + \frac{\partial \psi(u)}{\partial x} = \epsilon\frac{\partial^2 u}{\partial x^2} \tag{2.15}$$

rewriting the right hand side

$$\frac{\partial \eta(u)}{\partial t} + \frac{\partial \psi(u)}{\partial x} = \epsilon\frac{\partial}{\partial x}\left(\eta'(u)\frac{\partial u}{\partial x}\right) - \epsilon\eta''(u)\frac{\partial u^2}{\partial x} \tag{2.16}$$

Integrating this equation over a control volume $[x_1, x_2] \times [t_1, t_2]$ gives

$$\begin{aligned}\int_{t_1}^{t_2}\int_{x_1}^{x_2} \frac{\partial \eta(u)}{\partial t} + \frac{\partial \psi(u)}{\partial x} dxdt =& \epsilon\int_{t_1}^{t_2} \eta'(u(t,x_2))\frac{\partial}{\partial x}(u(t,x_2) - \eta'(u(t,x_1))\frac{\partial}{\partial x}(u(t,x_1))dt + \\ & -\epsilon\int_{t_1}^{t_2}\int_{x_1}^{x_2} \eta''(u)\frac{\partial u^2}{\partial x}dxdt\end{aligned} \tag{2.17}$$

As $\epsilon \to 0$ the first term of the right hand side vanishes. For the other term, if the weak solution is discontinuous in the rectangle, it will not vanish. However, since $\epsilon > 0$, $\frac{\partial u^2}{\partial x} > 0$ and $\eta''(u) > 0$ thanks to the convexity assumption, we can conclude that the right hand side is not positive, therefore the weak solution satisfies

$$\int_{t_1}^{t_2}\int_{x_1}^{x_2} \frac{\partial \eta(u)}{\partial t} + \frac{\partial \psi(u)}{\partial x} dxdt \leq 0 \tag{2.18}$$

for all t_1, t_2, x_1 and x_2.

Proposition 1. *(**Entropy condition**) The function $u(t,x)$ is the entropy solution of* (2.7) *if, for all convex entropy functions and corresponding entropy fluxes, the inequality*

$$\frac{\partial \eta(u)}{\partial t} + \frac{\partial \psi(u)}{\partial x} \leq 0 \tag{2.19}$$

is satisfied in the weak sense.

*The **weak form of the entropy inequality** is*

$$\int_0^T \int_{-\infty}^{+\infty} \eta(u(t,x)) \frac{\partial \varphi(t,x)}{\partial t} + \psi(u(t,x)) \frac{\partial \varphi(t,x)}{\partial x} dx dt \leq - \int_{-\infty}^{+\infty} \eta(u_0(x)) \varphi(0,x) dx \tag{2.20}$$

for all $\varphi \in \mathcal{C}_0^1([0,T) \times \mathbb{R})$.

2.6 Inflow and outflow characteristics

We consider now the imposition of the boundary condition for the problem (2.7) with initial condition in restricted domain $x \in [0,R]$. The characteristics are straight lines of equation

$$x = f'(u_0(x_0))t + x_0, \tag{2.21}$$

where $f'(u(x,t))$ is the characteristic speed. To simplify the problem, we consider the linear advection equation $f(u(t,x)) = au(t,x)$ with a constant.

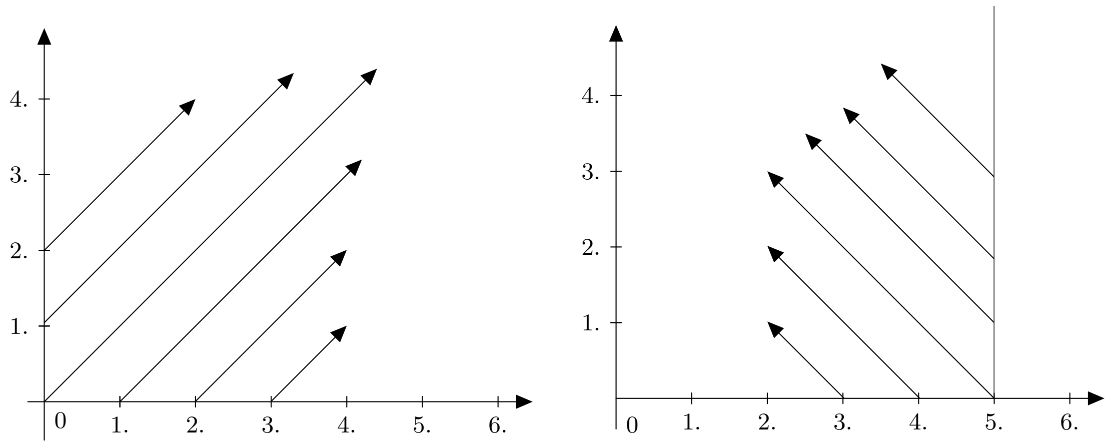

Figure 2.1: Outflow characteristics for $a > 0$ (left) and inflow characteristics for $a < 0$ (right).

As showed in Figure 2.1, if $a > 0$ therefore the characteristics get out from the axis x and t into the domain; in this case the boundary conditions are given in $x = 0$ and the characteristics are called *outflow characteristics.* If $a < 0$, instead the characteristics get out from x axis and the straight line $x = R$ into the domain, the the characteristics, in this case, are called *inflow characteristics.*

The boundary conditions are:

$$\begin{cases} u(t,0) & a > 0 \\ u(t,R) & a < 0 \end{cases} \tag{2.22}$$

2.7 Riemann problem

The Riemann problem for a generic conservation law is defined by assigning an initial condition given by the piecewise constant function:

$$u_0(x) = \begin{cases} u_l & x < 0 \\ u_r & x > 0 \end{cases} \tag{2.23}$$

For a convex flux, the form of the solution depends on the relation between u_l and u_r. We will give an example considering the *Burgers' equation* with flux function $f(u) = \frac{1}{2}u^2$.

First case: $\mathbf{u_l > u_r}$

In this case exist a unique weak solution

$$u(t,x) = \begin{cases} u_l & x < st \\ u_r & x > st \end{cases} \tag{2.24}$$

where $s = \dfrac{u_l + u_r}{2}$ is the shock speed, the speed at wich the discontinuity travels.

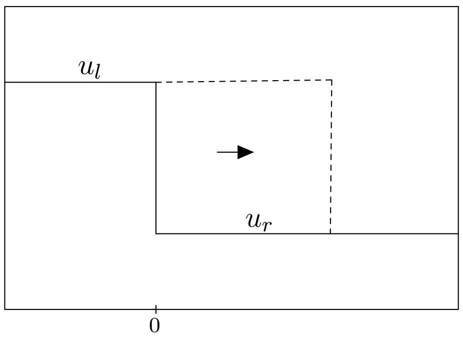

Figure 2.2: Shock wave.

Second case: $\mathbf{u_l < u_r}$

For this kind of problem there are infinitely many weak solutions, the weak solution physically correct is called *rarefaction wave* and corresponds to a series of characteristics emanating from the origin with continuous slopes between u_l and u_r:

$$u(t,x) = \begin{cases} u_l & x < u_l t \\ \dfrac{x}{t} & u_l t < x < u_r t \\ u_r & x > u_r t \end{cases} \tag{2.25}$$

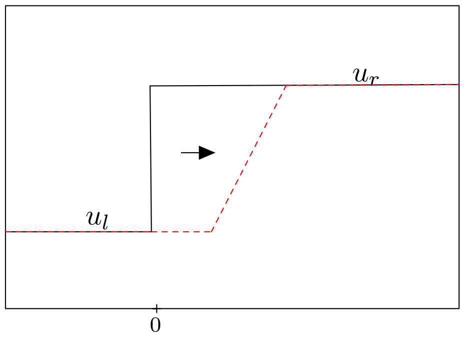

Figure 2.3: Rarefaction wave.

2.8 Buckley-Leverett equation

The Buckley-Leverett equation is a simple model for two phase fluid flow in a porous medium, as expleined in [LeV92]. An interesting application is to oil reservoir simulation. When an underground source of oil is tapped, a certain amount of oil flows out on its own due to high pressure in the reservoir. After the flow stops, there is a large amount of oil still in the ground. One standard method of "secondary recovery" is to pump water into the oil field through some wells, forcing oil out through others. In this case the two phases are oil and water, and the flow takes place in a porous medium of rock or sand. The Buckley-Leverett equations are a particularly simple scalar model that captures some features of this flow. In one space dimension the equation has the standard conservation law form (2.7) with

$$f(u) = \frac{u^2}{u^2 + a(1-u)^2} \tag{2.26}$$

where a is a constant. In this application u represents the saturation of water and so lies between 0 and 1. Figure 2.4 shows $f(u)$ and its derivative when $a = \frac{1}{2}$.

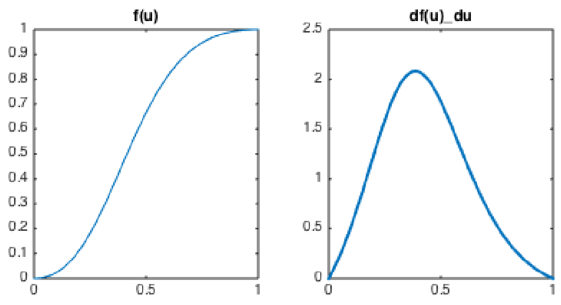

Figure 2.4: Flux function of the Buckley-Leverett equation with $a = \frac{1}{2}$.

The characteristics of this problem are present in Figure 2.5. If we consider as initial condition the Riemann problem with initial states $u_l = 1$ and $u_r = 0$, it models the flow of pure water if $u = 1$ into pure oil $u = 0$. In this case the Riemann solution involves both a shock and a rarefaction wave, in fact if $f(u)$ had more inflection points, the solution might involve several shocks and rarefactions.

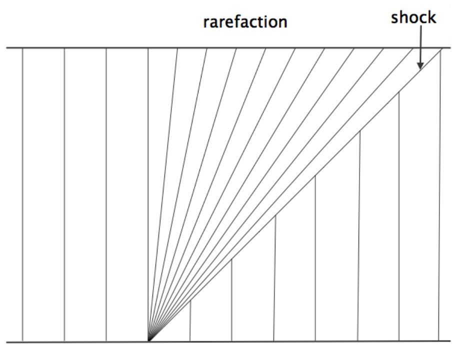

Figure 2.5: Characteristics gives by the Buckley-Leverett flux.

2.9 The Finite Volume method

In the Finite Volume method, the computational domain is divided into cells and the unknown quantity that is numerically computed is the cell average of u on each cell. This is in contrast to the Finite Difference method, for which the unknowns are the point values of u at the grid points. We need to number the cells. In 1D a convenient way to do it in order to avoid confusion with the grid points, is to assign half integers.

Let us denote by

$$u_i(t) = \frac{1}{\Delta x_i} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} u(t,x)dx \tag{2.27}$$

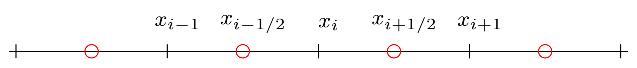

Figure 2.6: Mesh used for the space discretization.

where $\Delta x_i = x_{i+1} - x_{i-1}$.
Finite Volume numerical schemes is then obtained by integrating the original equation on each cell of the domain. As for the time scheme use the method of lines and separate space discretization from time discretization. So integrating (2.7) on the generic cell $[x_{i-1}, x_{i+1}]$ and dividing by Δx_i yields:

$$\frac{du_i(t)}{dt} = -\frac{1}{\Delta x_i}(f(u(t, x_{i+\frac{1}{2}})) - f(u(t, x_{i-\frac{1}{2}}))). \tag{2.28}$$

An important requirement, of our method is that it has to converge to a weak solution of the problem. For this reason the numerical method should be in **conservation form** which means it has form, where with c the numerical solution,

$$\begin{aligned}\frac{dc_i(t)}{dt} = -\frac{1}{\Delta x_i}[&F(c_{i-p}(t), c_{i-p+1}(t), \cdots, c_{i+q}(t)) + \\ &-F(c_{i-p-1}(t), c_{i-p}(t), \cdots, c_{i+q-1}(t))]\end{aligned} \tag{2.29}$$

for some function F of $p + q + 1$ arguments. F is called the **numerical flux function**. In the simplest case, p = 0 and q = 1 so that F is a function of only two variables and (2.29) becomes

$$\frac{dc_i(t)}{dt} = -\frac{1}{\Delta x_i}[F(c_i(t), c_{i+1}(t)) - F(c_{i-1}(t), c_i(t))] \tag{2.30}$$

This form satisfies the integral form of the conservation law if we view c_i as an approximation to the cell average defined by (2.27).

In the case of linear advection equation we use the **upwind method**, that in conservation form, it will be:

$$\frac{dc_i(t)}{dt} = -\frac{1}{\Delta x_i}\left(F_{i+\frac{1}{2}}(t) - F_{i-\frac{1}{2}}(t)\right) \tag{2.31}$$

with a numerical flux equal to:

$$F_{i+\frac{1}{2}}(t) = \begin{cases} ac_i(t) & a > 0 \\ ac_{i+1}(t) & a < 0 \end{cases}$$

The method is **consistent** if the numerical flux function reduces to the original flux in case of constant flow, that it means if $u(t, x) = \bar{u}$ then

$$F(\bar{u}, \bar{u}) = f(\bar{u}) \qquad \forall \bar{u} \in \mathbb{R} \tag{2.32}$$

For the upwind method it is easy to verify the the method is consistent, in fact for both the cases we obtain the exact flow of the conservation law that it is discretized in space. If we have to solve a nonlinear equation, the upwind method does not guarantee that weak solution obtained satisfy the entropy condition. An alternative numerical flux can be propose to approximate the scalar problem that gives the correct solution. They must be determined in order to ensure that the scheme preserves the monotonicity of the solution, in order to avoid spurious oscillations. Schemes satisfying this request are called monotone and as showed in [GR13] satisfy the entropy inequalities, and thus they converge to the unique entropy solution in the scalar case.

An example of monotone method in conservation form is given by (2.31) where we take as numerical flux the **Rusanov flux**, also called Local Lax Friedrichs flux, which is defined by

$$F_{i+\frac{1}{2}}(t) = \left[\frac{f(c_i(t)) + f(c_{i+1}(t))}{2} - \alpha \frac{(c_{i+1}(t) - c_i(t))}{2} \right] \tag{2.33}$$

where

$$\alpha = \max_{c \in [\beta, \gamma]} |f'(c)|$$

with $\beta = \min(c_i, c_{i+1})$ and $\gamma = \max(c_i, c_{i+1})$.

The idea behind this flux, instead of approximating the exact Riemann solver, is to recall that the entropy solution is the limit of viscous solutions and to take a centered flux to which some viscosity (with the right sign) is added. The only requirement is that the numerical flux must to be differentiable to evaluate the derivative of the flux.

Chapter 3

Multirate TR-BDF2 method

To calculate the numerical solution of an ODEs system there are a lot of methods that use different local time steps that are varying in time, but constant over the components. With the multirate methods we are able to have a big time step for the slow components while a smaller time step for the faster ones.

Given a system of ODEs with the notation present in equation (1.1), in a multirate approach, the system is partitioned into two subsystems, so that it is rewritten as

$$\begin{aligned} \mathbf{y}_a' &= \mathbf{f}_a(t, \mathbf{y}_a, \mathbf{y}_l) \\ \mathbf{y}_l' &= \mathbf{f}_l(t, \mathbf{y}_a, \mathbf{y}_l). \end{aligned} \tag{3.1}$$

The active components $\mathbf{y}_a$ are associated to phenomena on a fast time scale, while latent components are associated to slower phenomena. Given a global time step $h_n = t_{n+1} - t_n$, where the intervals $[t_n, t_{n+1}]$ are called time slabs, we compute the approximation of the solution at this new level for all components and, for those components that the error estimator is bigger than the tolerance, we recalculate the solution with a smaller time step, where the size of the new time step is selected with a self-adjusting strategy. In this smaller time step, for some components from the refinement set, we will need solution values of components not refined in that time, we use interpolation based on the information available at the time t_n and t_{n+1} to know the value of the latent components. Multirate methods avoid some computation that is necessary in the standard single rate approach. In other words, in the multirate approach, we are trying to use the most appropriate resolution for each state variable of the system. These algorithms can be very useful in problems having widely varying time scales.

3.1 A self adjusting implicit multirate approach

We consider multirate methods for the solution of the Cauchy problem

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y}) \qquad \mathbf{y}(0) = \mathbf{y}_0 \in \mathbb{R}^m, \qquad t \in [0, T]. \tag{3.2}$$

We will consider time discretizations defined by discrete time levels $t_n,\ n = 0, \dots, N$ such that $h_n = t_{n+1} - t_n$ and we will denote by $\mathbf{u}^n$ the numerical approximation of $\mathbf{y}(t_n)$. We will also denote by $\mathbf{u}^{n+1} = \mathcal{S}(\mathbf{u}^n, h_n)$ the implicitly defined operator whose application is equivalent to the computation of one step of size h_n of a given implicit, single step method. Furthermore, we will denote by $\mathbf{Q}(\mathbf{u}^{n+1}, \mathbf{u}^n, \zeta)$ an interpolation operator, that provides an approximation of the numerical solution at intermediate time levels $t_n + \zeta$, where $\zeta \in [0, h_n]$. Linear interpolation is often employed, but, for multistage methods, knowledge of the intermediate stages also allows the application of more accurate interpolation procedures without substantially increasing the computational cost.

The self adjusting multirate approach proposed by Ranade [Ran16], inspired by the method introduced in [SHV07], can be described as follows.

- Perform a tentative global (or macro) time step of size $h_n = h_n^{(0)}$ with the standard single rate method and compute

$$\hat{\mathbf{u}}^{n+1} = \hat{\mathbf{u}}^{n,0} = \mathcal{S}(\mathbf{u}^n, h_n^{(0)})$$

- Compute the error estimator and partition the state space into active and latent variables, based on the value of the error estimate. The projection into the subspace of the active variables is denoted by $\mathbf{P}_n = \mathbf{P}_n^{(0)}$. The specific way in which error control mechanisms are applied to the choice of the global step is crucial for the efficiency of the algorithm and will be discussed in detail in section 3.2. Here, we simply assume that $h_n^{(0)}$ has been chosen by an appropriate criterion.

- Set

$$\left(\mathbf{I} - \mathbf{P}_n^{(0)}\right) \mathbf{u}^{n+1} = \left(\mathbf{I} - \mathbf{P}_n^{(0)}\right) \hat{\mathbf{u}}^{n+1}$$

  and define $\mathbf{P}_n^{(0)} \mathbf{u}^{n,0} = \mathbf{P}_n^{(0)} \mathbf{u}^n$ and $t_{n,0} = t_n$.

- For $k = 0, \dots, k_{max}$

  - Choose a local (or micro) time step $h_n^{(k+1)}$ for the active variables, not larger than $t_{n+1} - t_{n,k}$.
  - At the intermediate time level $t_{n,k+1} = t_n + h_n^{(k+1)}$, compute by interpolation

$$(\mathbf{I} - \mathbf{P}_n^{(k)}) \hat{\mathbf{u}}^{n,k+1} = \mathbf{Q}\left((\mathbf{I} - \mathbf{P}_n^{(k)}) \mathbf{u}^{n+1}, (\mathbf{I} - \mathbf{P}_n^{(k)}) \mathbf{u}^{n,k}, h_n^{(k+1)}\right)$$

  - Compute

$$\mathbf{u}^{n,k+1} = \mathbf{P}_n^{(k)} \mathcal{S}(\mathbf{P}_n^{(k)} \mathbf{u}^{n,k}, h_n^{(k)}) + (\mathbf{I} - \mathbf{P}_n^{(k)}) \hat{\mathbf{u}}^{n,k+1}$$

  - Compute the error estimate and and partition the image of $\mathbf{P}_n^{(k)}$ into active and latent variables. Denote by $\mathbf{P}_n^{(k+1)}$ the projection onto the subspace of variables that need further refinement and repeat until $\sum_{l=1}^{k+1} h_n^l = h_n$.

Clearly, the effectiveness of the above procedure depends in a crucial way on the accuracy and stability of the basic ODE solver $\mathcal{S}$, as well as on the time step refinement and partitioning criterion. Furthermore, as well known in multirate methods, unconditional stability of the solver $\mathcal{S}$ does not necessarily entail that the same property holds for the derived multirate solver. The refinement and partitioning criterion will be described in detail in section 3.2. Here, we only remark that, as it will be shown by the numerical experiments in chapter 4, the approach outlined above is able to reduce significantly the computational cost with respect to the equivalent single rate methods, without a major reduction in stability for most of the envisaged applications. A linear stability analysis is present in [Ran16].

3.2 The time step refinement and partitioning criterion

We now describe in detail the time step refinement and partitioning strategy that we have used in the multirate algorithm described in section 3.1. Ranade's approach is based on the strategy proposed for an explicit Runge Kutta multirate method in [Fok15], where the time steps for refinement are obtained from the error estimates of the global step. The user specified tolerance plays an important role in the partitioning of the system. In [Fok15], a simple absolute error tolerance was considered. However, in most engineering system simulations, whenever the typical values of different components can vary greatly, the tolerance used is in general a combination of absolute and relative error tolerances, see e.g. [SW06]. Ranade has thus extended the strategy proposed in [Fok15] in order to employ such a combination of absolute and relative error tolerances.

More specifically, denote by τ_r, τ_a the user defined error tolerances for relative and absolute errors, respectively. Furthermore, assume that the tentative global step $\hat{\mathbf{u}}^{n+1} = \mathcal{S}(\mathbf{u}^n, h_n^{(0)})$ has been computed and that an error estimator $\boldsymbol{\epsilon}_{n+1}^0$ is available. The first task of the time step selection criterion is to asses whether the global time step $h_n^{(0)}$ has been properly chosen. Denoting by $\epsilon_i^{n+1,0}, \hat{u}_i^{n+1}, i = 1, \ldots, m$ the $i-$th components of the error estimator and of the tentative global step numerical solution, respectively, we define a vector $\boldsymbol{\eta}^{n+1,0}$ of normalized errors with components

$$\eta_i^{n+1,0} = \frac{\epsilon_i^{n+1,0}}{\tau_r|\hat{u}_i^{n+1}| + \tau_a}.$$

Since clearly the condition $\max_i \eta_i^{n+1,0} \leq 1$ has to be enforced, before proceeding to the partitioning into active and latent variables, this condition is checked and the global step is repeated with a smaller value of $h_n^{(0)}$ whenever it is violated. Numerical experiments have shown that, while an increase of efficiency with respect to the single rate version of the method is always achieved, independently of the choice of the tentative step, the greatest improvements are only possible if the global time step does not have to be repeated too often.

Once condition $\max_i \eta_i^{n+1,0} \leq 1$ is satisfied, the set of indices of the components flagged for the first level of time step refinement is identified by

$$\mathcal{A}_{n+1}^0 = \{i : \eta_i^{n+1,0} > \delta \|\boldsymbol{\eta}^{n+1,0}\|_\infty\}, \tag{3.3}$$

where $\delta \in (0,1)$ is a user defined coefficient. The smaller the value of δ, the larger is the fraction of components marked for refinement. Notice that, if δ is set to unity, the algorithm effectively operates in single rate mode. For each iteration $k = 1, \ldots, k_{max}$ of the algorithm described in section 3.1, active variable sets $\mathcal{A}_{n+1}^k$ are then defined analogously. In each iteration $k = 0, \ldots, k_{max}$, the time step $h_n^{(k)}$ is chosen:

$$h_n^{(k)} = \nu \min_{j \in \mathcal{A}_{n+1}^k} \left(\frac{\tau_r |u_i^n| + \tau_a}{\epsilon_i^{n+1,k}} \right)^{\frac{1}{p+1}}, \tag{3.4}$$

where p is the convergence order of the solver $\mathcal{S}$ and ν is an user defined safety parameter.

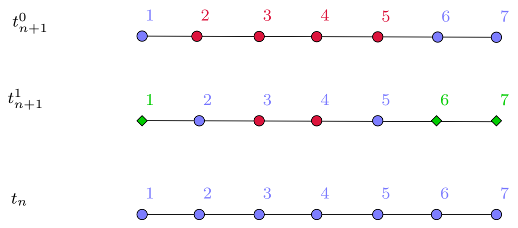

Figure 3.1: An example of partitioning, the flagged components are in red, the interpolated ones are in green.

3.3 The interpolation procedures

An essential component of any multirate algorithm is the procedure employed to reconstruct the values of the latent variables at those intermediate time levels for which only the active variables are computed. Self-adjusting multirate procedures based on implicit methods, such as the one proposed in [SHV07] and that presented in this paper, can avoid the use of extrapolation, thus increasing their overall stability. The simplest and most commonly used procedure is linear interpolation, that is defined for $\zeta \in [0, h_n]$ as

$$\mathbf{Q}(\mathbf{u}^{n+1}, \mathbf{u}^n, \zeta) = \frac{\zeta}{h_n}\mathbf{u}^{n+1} + \frac{(h_n - \zeta)}{h_n}\mathbf{u}^n. \tag{3.5}$$

An interesting feature of the TR-BDF2 method in the multirate framework is that the method, as explained in [HS96], is endowed with a cubic Hermite, globally C^1 interpolation algorithm for dense output. This interpolant allows to have accurate approximations of the latent variables without extra computational cost or memory storage requirements, since it employs the intermediate stages of the TR-BDF2 method in order to achieve higher order accuracy. using the notation of section 1.8, the Hermite cubic interpolant can be defined as

$$\begin{aligned}\mathbf{Q}(\mathbf{u}^{n+1}, \mathbf{u}^n, \zeta) &= (\boldsymbol{\alpha}_3 - 2\boldsymbol{\alpha}_2)\beta^3(\zeta) \\ &+ (3\boldsymbol{\alpha}_2 - \boldsymbol{\alpha}_3)\beta^2(\zeta) + \boldsymbol{\alpha}_1\beta(\zeta) + \boldsymbol{\alpha}_0,\end{aligned} \tag{3.6}$$

where, for $0 \leq \zeta \leq \gamma h_n$ one has

$$\boldsymbol{\alpha}_0 = \mathbf{u}^n, \quad \boldsymbol{\alpha}_1 = \gamma \mathbf{z}^n, \quad \boldsymbol{\alpha}_2 = \mathbf{u}^{n+\gamma} - \mathbf{u}^n - \gamma \boldsymbol{z}^n,$$
$$\boldsymbol{\alpha}_3 = \gamma(\boldsymbol{z}^{n+\gamma} - \boldsymbol{z}^n), \qquad \beta(\zeta) = \frac{\zeta}{\gamma h_n}$$

and for $\gamma h_n \leq \zeta \leq h_n$ one has instead

$$\boldsymbol{\alpha}_0 = \mathbf{u}^{n+\gamma}, \ \boldsymbol{\alpha}_1 = (1-\gamma)\mathbf{z}^{n+\gamma}, \ \boldsymbol{\alpha}_2 = \mathbf{u}^{n+1} - \mathbf{u}^{n+\gamma} - \boldsymbol{\alpha}_1,$$
$$\boldsymbol{\alpha}_3 = (1-\gamma)(\mathbf{z}^{n+1} - \mathbf{z}^{n+\gamma}) \qquad \beta(\zeta) = \frac{\zeta - \gamma h_n}{(1-\gamma)h_n}.$$

3.4 Error estimation

In the Ranade's algorithm, the error is estimated with a modified error estimator. First he compares the result of the second order formula to that of the embedded third order formula, then he takes as final result the solution of the linear system gives by equation 1.23, as proposed in [HS96]. As explained in section 1.8, this modification improve it for stiff components preserving its accuracy as $h \to 0$.

For a large PDE problem, solving at each time step another linear system could turn out to be very expensive. We propose other types of error estimator that are less expensive as they do not require to solve a linear system. At each time step, we know the active components values at times t_n and $t_{n+\gamma}$, so we can use an extrapolation technique. The simplest one is the linear extrapolation, given by

$$\hat{\mathbf{u}}_{lin}^{n+1} = \mathbf{u}^n + \frac{t_{n+1} - t_n}{t_{n+\gamma} - t_n}(\mathbf{u}^{n+\gamma} - \mathbf{u}^n). \tag{3.7}$$

Another estimator can be obtained by applying the Cubic Hermite extrapolation:

$$\begin{aligned}\hat{\mathbf{u}}_{cub}^{n+1} &= (\boldsymbol{\alpha}_3 - 2\boldsymbol{\alpha}_2)\beta^3 \\ &+ (3\boldsymbol{\alpha}_2 - \boldsymbol{\alpha}_3)\beta^2 + \boldsymbol{\alpha}_1\beta + \boldsymbol{\alpha}_0,\end{aligned} \tag{3.8}$$

where

$$\boldsymbol{\alpha}_0 = \mathbf{u}^n, \quad \boldsymbol{\alpha}_1 = \gamma \mathbf{z}^n, \quad \boldsymbol{\alpha}_2 = \mathbf{u}^{n+\gamma} - \mathbf{u}^n - \gamma \boldsymbol{z}^n,$$
$$\boldsymbol{\alpha}_3 = \gamma(\boldsymbol{z}^{n+\gamma} - \boldsymbol{z}^n), \qquad \beta = \frac{t_{n+1} - t_n}{\gamma h_n} \qquad .$$

Therefore, the error estimate , is given by:

$$\boldsymbol{\epsilon}^{n+1} = |\hat{\mathbf{u}}^{n+1} - \mathbf{u}^{n+1}| \tag{3.9}$$

where $\hat{\mathbf{u}}^{n+1}$ denotes either lines or cubic extrapolator.

Chapter 4

Numerical Experiments

A number of tests have been performed to assess the accuracy and efficiency of the propose multirate TR-BDF2 method. In Appendix A is present the structure of the used code.

4.1 Geochemistry ODE system

The first case test is an ODE system that is a model of geochemistry reactions. Given a system of species that are reacting with each other, the problem can be modeled, according to [BGG12], in the following way:

$$\frac{d\mathbf{n}}{dt} = \boldsymbol{R}(\mathbf{n}) \tag{4.1}$$

where

$$Rj = \sum_{l=1}^{N} S_{jl}\pi_l \qquad j = 1...n_{species} \tag{4.2}$$

- N is the total number of reactions that are present in the system,
- $\mathbf{S}$ is the stoichiometric matrix and

$$\pi_l = \mathcal{K}_l \left[\prod_{S_{jl}>0} \left(\frac{a_j(\mathbf{n})}{k_j}\right)^{|S_{jl}|} - \prod_{S_{jl}<0} \left(\frac{a_j(\mathbf{n})}{k_j}\right)^{|S_{jl}|} \right] \tag{4.3}$$

where $\mathcal{K}_l$ is the reaction speed, which could be a function of the temperature or a constant value. In this first approach to the problem, we have assumed it to be constant. a_j is the activity of a chemical species, which is modeled according to the the ideal activity law for every species. The constant k_j is the equilibrium constant for the j-th species.

The activity of a chemical species is modeled by the ideal law:

$$a_i = \begin{cases} x_i = \dfrac{n_{H_2O}}{\sum\limits_{i\in w} n_i} & i = H_2O \\ m_i = \dfrac{n_i}{n_{H_2O}} \dfrac{1}{18 \cdot 10^{-3}} & i \in w \neq H_2O \\ x_i = 1 & i \notin w. \end{cases} \tag{4.4}$$

Here,w denotes the aqueous phase, while:

- x_i is the molar fraction of the considered species in its phase,
- m_i represents the amount of a species expressed in moles/kg of solvent.

In the first test case, we consider a system formed of 12 species divided into 4 phases:

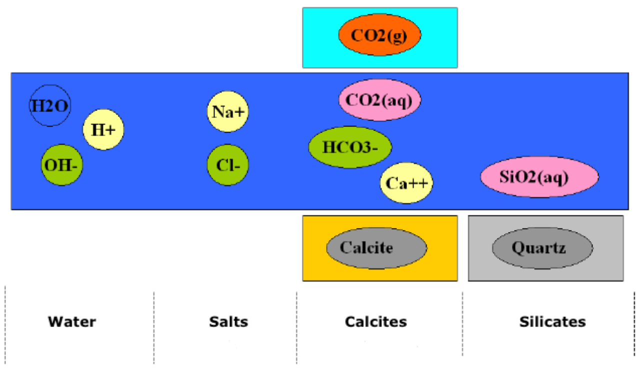

Figure 4.1: Structure of the system.

- Phase 1. Gas: $CO_2(g)$
- Phase 2. Aqueous solution: $H_2O, H^+, CO_2(aq), Cl^-, Na^+, Ca^{+2}, SiO_2(aq), HCO_3^-, OH^-$
- Phase 3. Calcite mineral: $CaCO_3(s)$
- Phase 4. Quartz mineral: $SiO_2(s)$

We consider the following three equilibrium reactions:

- Req 1. Hydrolysis of water

  $H_2O \rightleftharpoons H^+ + OH^-$

- Req 2. Dissolution of $CO_2(g)$ in water

  $CO_2(aq) \rightleftharpoons CO_2(g)$

- Req.3 Dissociation of $CO_2(aq)$

  $H_2O + CO_2(aq) \rightleftharpoons HCO_3^- + H^+$

We model dissolution-precipitation reactions by the following kinetics:

- Rkin 1. Dissolution of Calcite

  $CaCO_3(s) + H^+ \rightarrow Ca^{2+} + HCO_3^-$

- Rkin2. Precipitation of Calcite

  $CaCO_3(s) + H^+ \leftarrow Ca^{2+} + HCO_3^-$

- Rkin3. Dissolution of Quartz

  $SiO_2(s) \rightarrow SiO_2(aq)$

- Rkin 4. Precipitation of Quartz

  $SiO_2(s) \leftarrow SiO_2(aq)$

In Table 4.1 we can see the parameters of the species and in table 4.2 the reaction kinetic values that, in our model, are taken as constants.

The problem considered has initial time $t_0 = 0$ s and final time $t_{final} = 300s$, we take as Newton tolerance 1×10^{-11}; the tolerance for the error estimator is taken equal to $[1 \times 10^{-4}, 1 \times 10^{-5}]$. We denote here tolerance as $[\tau_r, \tau_a]$, where τ_r is the relative tolerance while τ_a is the absolute tolerance. The initial time step is equal to $1 \times 10^{-4}s$.

In Figure 4.2 and 4.3 we show the species concentrations, as computed with the multirate TR-BDF2 method and with the ode45 method implemented in MATLAB. The species amounts are in agreement with the qualitative evolution expected. The amount of $(CO_2)_g$ increases producing a loss of $(CO_2)_{aq}$, that is also present in reaction Req.3. For this reason also the H^+ and HCO_3^- amounts decrease, in order to try reach equilibrium. Another reaction is also involved in the process, that is the hydrolisis of water, the loss of H^+ amount causes the growth of OH^- and H_2O. H^+ and HCO_3^- are also present in the first kinetic reaction, they induce the precipitation of calcite thus reducing the Ca^{++} amount. The slow increment of $(SiO_2)_s$ is linear inasmuch the activity value of

Phase	Species	$log_{10}(k)$	initial concentration $[mol/m^3]$
Aqueous	H_2O	0	8071.20559
Aqueous	H^+	0	0.00377657
Aqueous	$CO_2(aq)$	0	119.23091
Aqueous	Cl^-	0	156.803187
Aqueous	Na^+	0	145.432629
Aqueous	Ca^{+2}	0	7.01164993
Aqueous	$SiO_2(aq)$	0	0.03770169
Aqueous	HCO_3^-	-6.2206340	2.62191679
Aqueous	OH^-	-13.235362	$3.091 \cdot 10^{-7}$
Mineral Calcite	$CaCO_3(s)$	-7.7454139	$4.33 \cdot 10^3$
Mineral Quartz	$SiO_2(s)$	3.5862160	$2.82 \cdot 10^4$
Gas	$CO_2(g)$	2.0861861	$4.27 \cdot 10^2$

Table 4.1: Thermodynamic parameters of the species

Reaction	Type	Name	$\mathcal{K}$
Req 1	Aqu	Hydrolisis of water	$1 \cdot 10^3$
Req 2	Gas-Aqu	Dissolution of $CO_2(g)$ in water	$1 \cdot 10^1$
Req 3	Aqu	Dissociation of $CO_2(aq)$	$1 \cdot 10^3$
Rkin 1	Min-Aqu	Dissolution of Calcite	$-1 \cdot 10^{-1}$
Rkin 2	Min-Aqu	Precipitation of Calcite	$1 \cdot 10^{-1}$
Rkin 3	Min-Aqu	Dissolution of Quartz	$-1 \cdot 10^{-1}$
Rkin 4	Min-Aqu	Precipitation of Quartz	$1 \cdot 10^{-1}$

Table 4.2: Thermodynamic parameters of the reactions

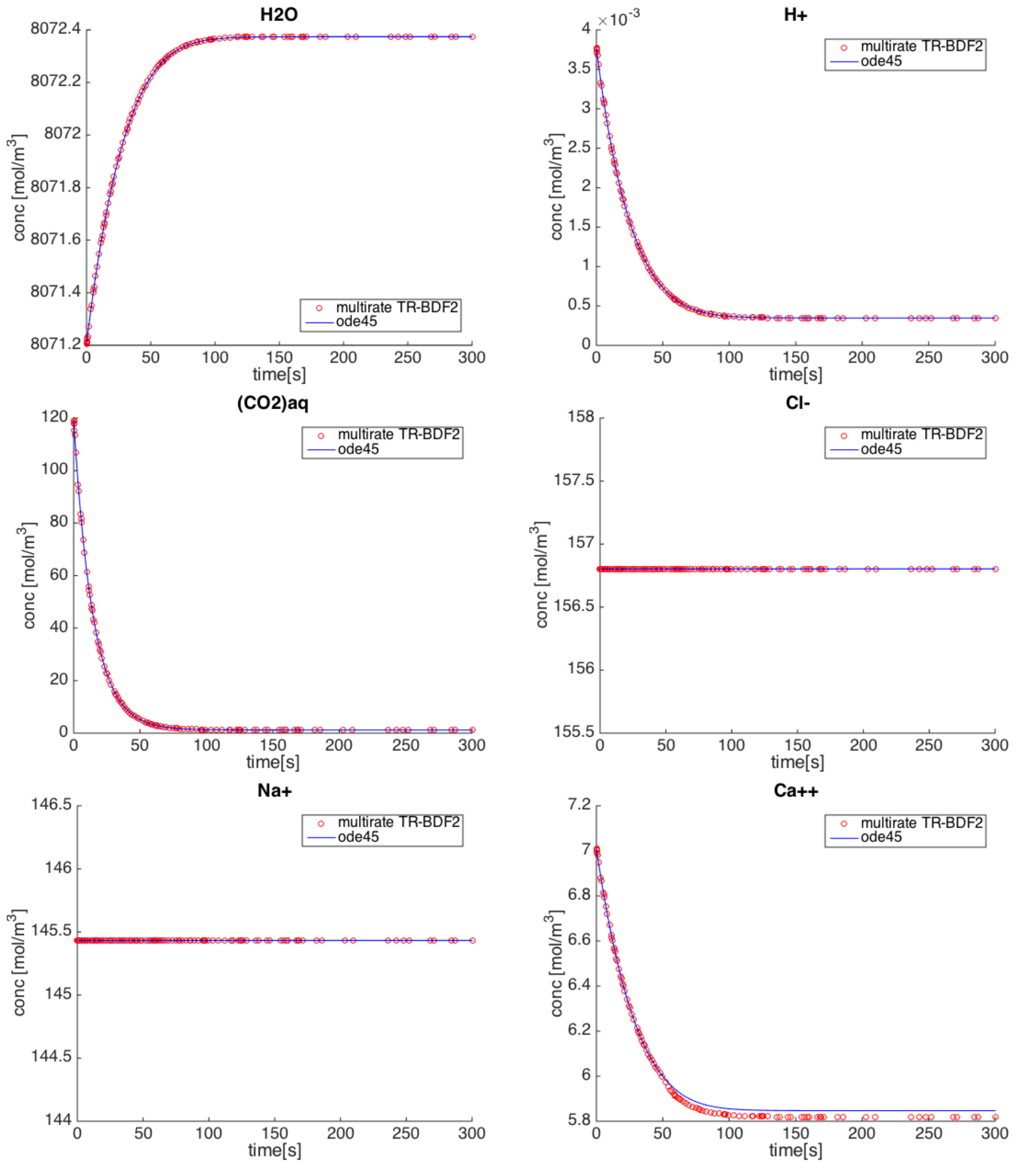

Figure 4.2: Amount (from the top to the bottom, from left to right) of H_2O, H^+, $(CO_2)_{aq}$, Cl^-, Na^+ and Ca^{++} computed with the multirate method (Cubic Hermite interpolation) and with the ode45 matlab method.

this species is equal to 1. The Cl^- and Na^+ are not present in any reaction, so their amounts stay constant for all the simulation.

The evolution of the Ca^{++} amount computed with the multirate TR-BDF2 method is not captured as well as the amount computed by the MATLAB solver. The reason of

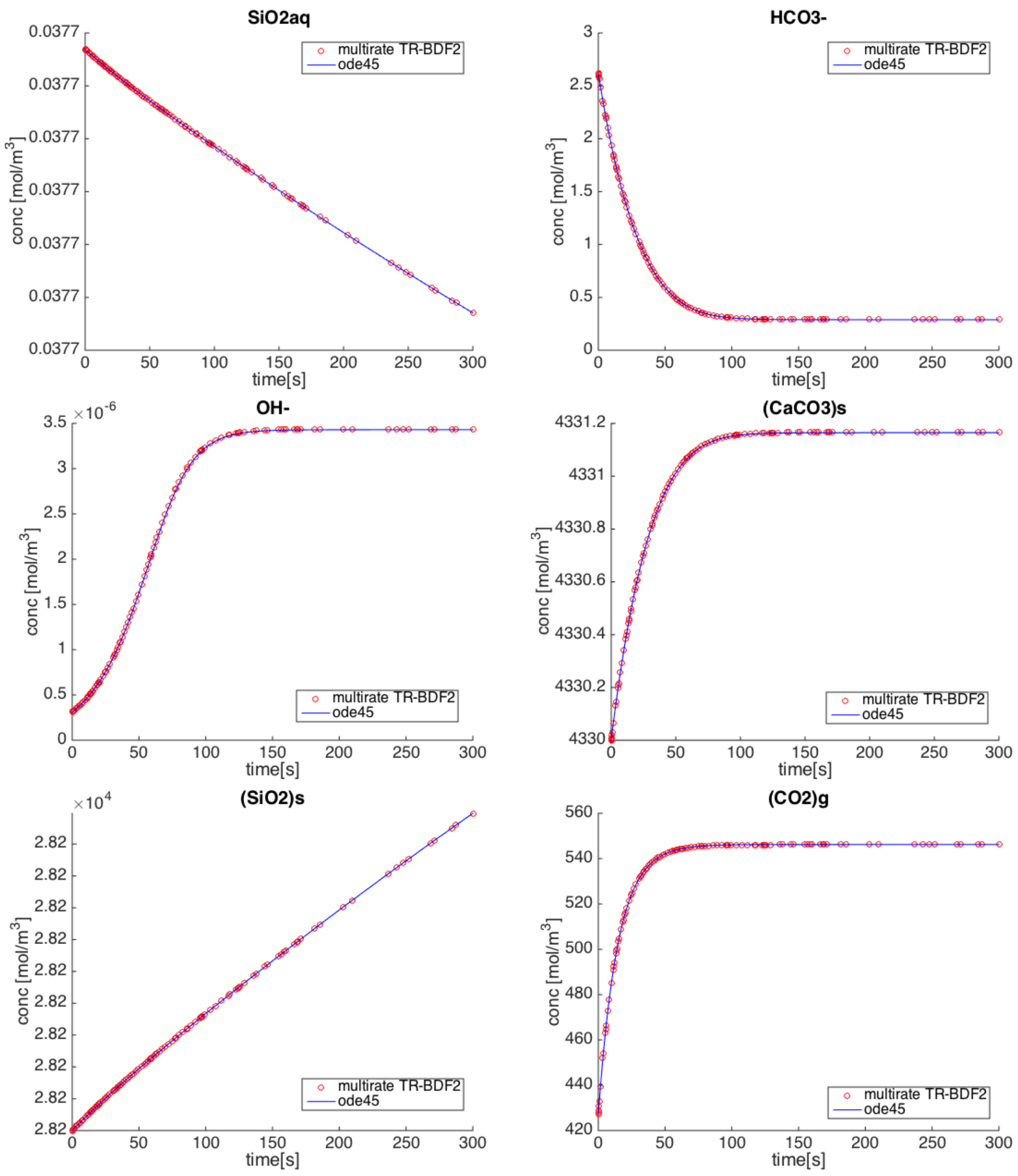

Figure 4.3: Amount (from the top to the bottom, from left to right) of $(SiO_2)_{aq}$, HCO_3^-, OH^-, $(CaCO_3)_s$, $(SiO_2)_s$ and $(CO_2)_g$ computed with the multirate method (Cubic Hermite interpolation) and with the ode45 matlab method.

this fact, is because the reaction where the species is present has a slow speed respect the other reactions, so that the Ca^{++} amount is always interpolated.
In section 1.8 we have explained that the TR-BDF2 method is not unconditionally monotone, so that the computed concentrations can become negative. In this system the only

species that goes to zero is the $(CO_2)_{aq}$, as showed in Figure 4.2, but its amount remains positive for every time step.

In Table 4.3 we report the relative error :

$$\frac{||u_j - n_j||_\infty}{||n_j||_\infty} \qquad \forall j = 1 \cdots n_{species} \tag{4.5}$$

where we consider for every species the infinity norm of the difference between the solution computed by our algorithms (u) and the solution computed by the MATLAB solver (n) divided by the infinity norm of the MATLAB solution. In the the first column the algorithm used is the multirate TR-BDF2 with a cubic interpolator that turn out to be better than the multirate TR-BDF2 method with the linear interpolator (second column), even if the single rate method with the same tolerance for the error estimator gives an extra order of accuracy. If we analyze the computational cost between the multirate and single-rate method presents in Table 4.4, we do not see a lot of difference because the system is not sufficiently big to underline the benefit of the multirate method.

4.2 Extended Geochemistry problem

We have seen a simple case test for a geochemistry problem. Now we want to test the multirate algorithm performance, we consider a larger chemistry system where a lot of species are not interacting with each other. Let us consider a bigger system with the same species and reactions of the previous case where we add 100 species that stay constant for all the simulation, they will be present in no reaction. We set the initial amount of these species equal to the Cl^- species value. In Table 4.6 are reported the computation time and the number of time step for the multirate TR-BDF2 method with cubic Hermite interpolation and the single rate TR-BDF2 method. We can see that in the multirate version we obtain a bigger computational cost respect the single rate method because, at each time step, the the last one computes the species amount for all the components. It also requests more time steps.

Another geochemistry problem that we want to analyze is like the previous one, but this time, the species interact with each other two by two. This means that, adding 100 of species at the base system, we have that:

$$\begin{aligned} s_1 + s_2 &\rightleftharpoons s_3 + s_4 \\ s_3 + s_4 &\rightleftharpoons s_5 + s_6 \\ &\vdots \\ s_{99} + s_{100} &\rightleftharpoons s_1 + s_2. \end{aligned}$$

Species	Cubic	Linear	Single rate
H_2O	2.11×10^{-7}	6.67×10^{-7}	3.31×10^{-8}
H^+	1.07×10^{-3}	4.24×10^{-3}	1.92×10^{-4}
$(CO_2)_{aq}$	2.74×10^{-4}	6.31×10^{-4}	1.12×10^{-5}
Cl^-	0	0	0
Na^+	0	0	0
Ca^{++}	9.09×10^{-3}	3.22×10^{-2}	3.80×10^{-5}
$SiO2_{aq}$	1.97×10^{-8}	3.97×10^{-8}	1.54×10^{-9}
HCO_3^-	5.08×10^{-4}	2.17×10^{-3}	2.03×10^{-4}
OH^-	4.40×10^{-3}	3.35×10^{-2}	4.78×10^{-4}
$(CaCO_3)_s$	3.92×10^{-7}	1.24×10^{-6}	6.15×10^{-8}
$(SiO_2)_s$	2.54×10^{-14}	5.27×10^{-14}	1.54×10^{-15}
$(CO_2)_g$	1.24×10^{-4}	5.17×10^{-4}	2.50×10^{-5}

Table 4.3: Relative error in infinity norm at a fixed time between the solution computed with ode45 and the solution computed with multirate method with cubic Hermite interpolator (first column), the solution computed with multirate method with linear interpolator (second column) and the solution computed with single-rate method (third column) for the geochemistry problem.

For all these reactions we take as kinetic parameter the same value of the slowest reaction, the precipitation of calcite. These species have not to be at the equilibrium so the equilibrium constant k_j in (4.3) is different for the 100 species, $log(k_{s1}) = log(k_{s2}) = 0$, $log(k_{s3}) = log(k_{s4}) = 2.0861861$, $log(k_{s5}) = log(k_{s6}) = 0$, $log(k_{s7}) = log(k_{s8}) = 2.0861861$ and so on.

In Figure 4.4 we can see the amount of the fist four new added species, for the other new species the plot is identically. The reactions, for the new species, have a slow speed, in fact at time $t = 300s$ they have not yet reach equilibrium. The amount of the species that were present in the first system have the same trend (Figures 4.2 and 4.3) and reach equilibrium at time $t = 150s$. After this time, as showed in Figure 4.5, the refinement strategy does not reject any more components and proposes every time a bigger global time step, where is not necessary any refinement.

Method	Computation time	Numb. time steps
Singlerate TR-BDF2	2.67s	256
Multirate TR-BDF2	1.93s	139

Table 4.4: Computation time and number of time steps for the geochemistry problem.

Method	Computation time	Numb. time steps
Singlerate TR-BDF2	8.7s	256
Multirate TR-BDF2	6.78s	139

Table 4.5: Computation time and number of time steps for the extended geochemistry problem with 100 *non interacting species.*

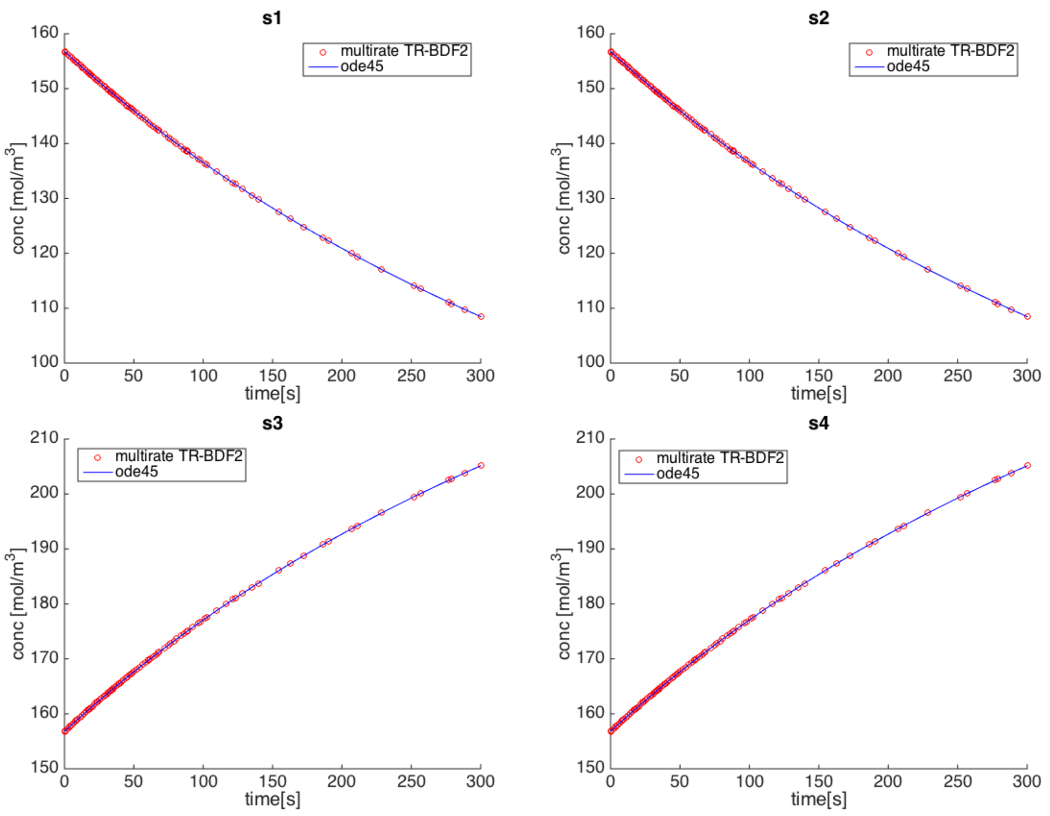

Figure 4.4: Amount of the first four new added species computed with the multirate method (Cubic Hermite interpolation) and with the ode45 matlab method.

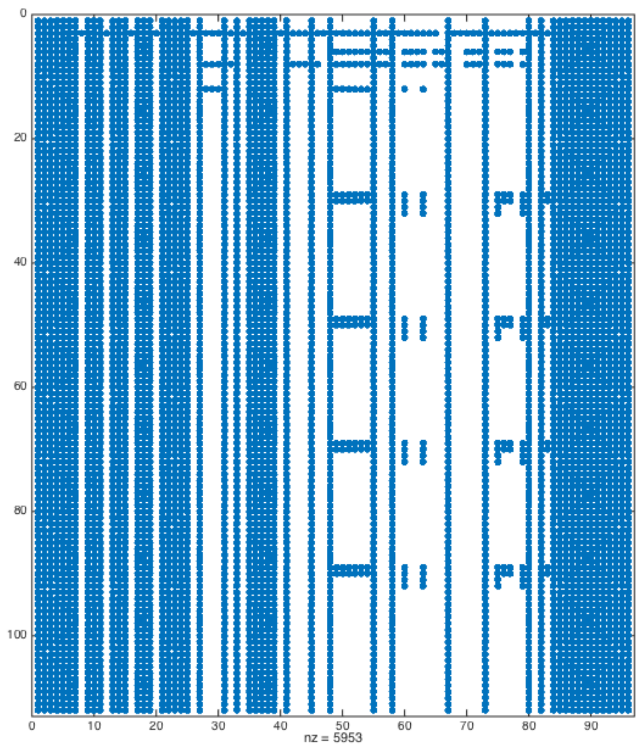

Figure 4.5: The components being computed at each time step by the multirate TR-BDF2 algorithm for the extended geochemistry ODE.

Method	Computation time	Numb. time steps
Singlerate TR-BDF2	$33.78s$	170
Multirate TR-BDF2	$27.15s$	144

Table 4.6: Computation time and number of time steps for the extended geochemistry problem with 100 *species that interact with pairwise.*

4.3 Linear Advection Equation

As first case test for the hyperbolic equations we consider a linear case

$$\begin{cases} \dfrac{\partial u}{\partial t} + \dfrac{\partial u}{\partial x} = 0, & x \in [-20, 20], \quad t > 0 \\ \\ u(x,0) = \exp(-x^2) & t > 0. \end{cases} \tag{4.6}$$

To discretize in space we used a first order upwind method. We set $a = 1$ if we consider the notation used in (2.4), so we use as boundary condition $u(x_0, t) = u_0(t_0)$. The interval time is $[0, 3]s$ with a number of cells equal to 400, the error tolerance is $[1 \times 10^{-6}, 1 \times 10^{-8}]$, where the first value is the relative error tolerance while the second is the absolute error tolerance. The initial size of the time step is equal to $1 \times 10^{-2}s$.

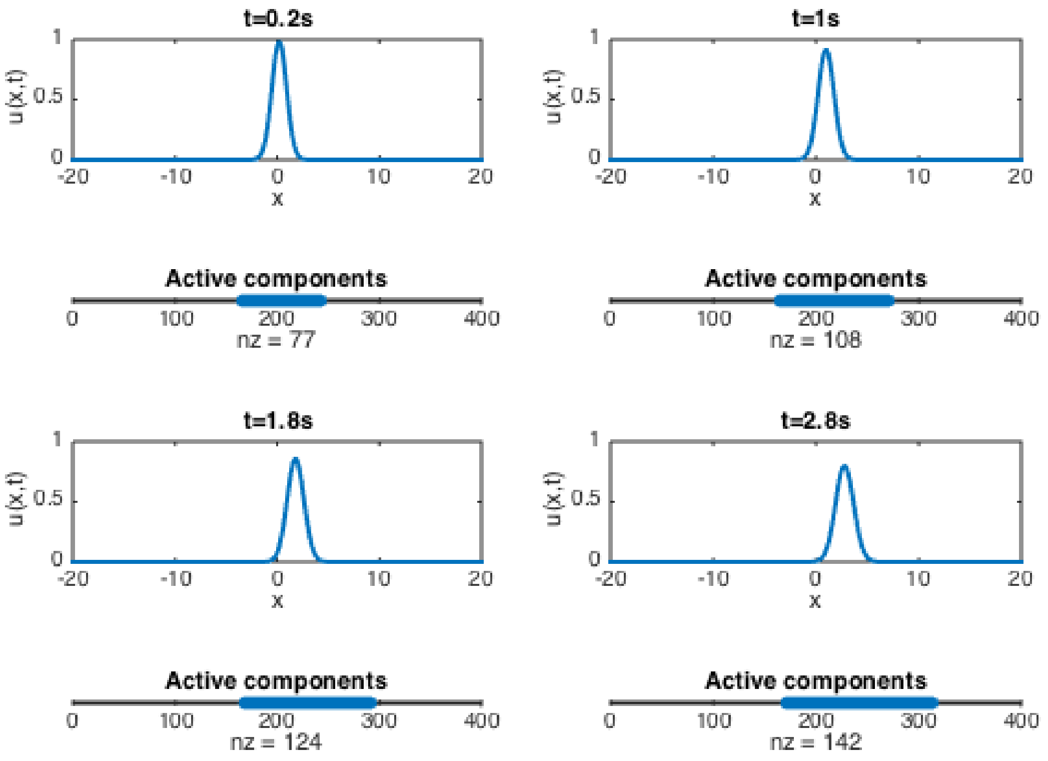

Figure 4.6: Multirate TR-BDF2 method with Cubic interpolation for the linear advection problem.

The upwind discretization has a diffusion term which is responsible for the spreading of the initial condition. This diffusion also ensures that the number of computed components increases as time progresses (Figure 4.6).

Method	Computation time	Numb. time steps
Single rate TR-BDF2	$21.41s$	323
Multirate TR-BDF2	$11.59s$	433

Table 4.7: Computation time and number of time steps for the linear advection equation.

time [s]	Cubic interp.	Linear Interp.	Single rate	Cubic extrap.
0.2	5.7×10^{-2}	5.71×10^{-2}	5.88×10^{-2}	5.92×10^{-2}
1	2.73×10^{-1}	2.74×10^{-1}	2.63×10^{-1}	2.74×10^{-1}
1.8	4.52×10^{-1}	4.53×10^{-1}	4.46×10^{-1}	4.54×10^{-1}
2.8	6.41×10^{-1}	6.43×10^{-1}	6.41×10^{-1}	6.44×10^{-1}

Table 4.8: Relative error in infinity norm at a fixed time between the exact solution and the solution computed with the multirate method with cubic Hermite interpolator (first column), the solution computed with multirate method with linear interpolator (second column), the solution computed with single-rate method (third column) and the solution computed with the multirate method with the cubic Hermite extrapolator as error estimator (fourth column) for the linear advection equation.

time [s]	Cubic interp.	Linear Interp.	Single rate	Cubic extrap.
0.2	1.41×10^{-6}	3.07×10^{-5}	1.38×10^{-6}	1.53×10^{-5}
1	5.45×10^{-6}	1.54×10^{-5}	4.76×10^{-6}	1.53×10^{-5}
1.8	8.78×10^{-6}	1.49×10^{-5}	8.80×10^{-6}	1.44×10^{-5}
2.8	1.24×10^{-5}	1.72×10^{-5}	1.02×10^{-5}	1.54×10^{-5}

Table 4.9: Relative error in infinity norm at a fixed time between the approximate solution computed with ode45 and the solution computed with the multirate method with cubic Hermite interpolator (first column), the solution computed with multirate method with linear interpolator (second column), the solution computed with single-rate method (third column) and the solution computed with the multirate method with the cubic Hermite extrapolator as error estimator (fourth column) for the linear advection equation.

time [s]	0.2	1	1.8	2.8
	5.88×10^{-2}	2.61×10^{-1}	4.43×10^{-1}	6.40×10^{-1}

Table 4.10: Relative error in infinity norm at a fixed time between the exact solution and the solution computed by the MATLAB solver for the linear advection equation.

To compute the relative errors we used the exact solution (Table 4.8) and the approximate solution with ode45 MATLAB's method set to a suitably stringent tolerance (Table 4.9). We used the last one to not consider the error caused by the upwind space discretization. To not assess the errors caused by the space discretization we have also report the relative errors between the exact solution and the solution computed by the MATLAB solver with the ode45 method. As Figure 4.6 confirms, the method uses the interpolation for constant component values, and we can not see a relevant difference in terms of time errors between the multirate and single rate method.
In the multirate TR-BDF2 algorithm we can choose the components to refine using the error estimator given by (1.23). As explained in Section 3.4, this approach can be expensive from a computational cost point of view. We first used the linear extrapolation to estimate the solution at time t_{n+1} with bad results. The method executes 12826 time steps so it rejects the micro time steps more times of the due. With the Cubic Hermite extrapolation we obtain better results, 503 time steps are necessary, the execution time is equal to 13.21 s. In Tables 4.8 and 4.9 we report the relative errors in infinity norm. In this last case the multirate method with Cubic Hermite extrapolator as error estimator is as accurate as the one with the original error estimator. In Figure 4.7 we have computed the CPU times at different cell numbers for the single rate method, multirate method with Cubic interpolation and original error estimator and multirate method with Cubic interpolation and Cubic extrapolator as error estimator. The multirate algorithms performed better than the single rate method. In this case we have not any benefit with the new error estimator because to solve the linear systems a direct method is used and the matrix, that is at each time step always the same, is already factored. If we use a iterative method this would not be true and we could see a computational gain.

As explained in Section 1.8 the TR-BDF2 method can not preserve the positivity of the solution. To obtain a violation of the monotonicity propriety we increased the number of cells to obtain a higher Courant number for each time step. The threshold to obtain a negative solution is a Courant number major than 2.5. We use 1000 cells and a tolerance equals to $[1 \times 10^{-2}, 1 \times 10^{-3}]$. In Figure 4.8 we have reported in red the components with negative values for the multirate (left) and for the single rate (right) algorithms. The number of components less than zero is major for the multirate method because the latent components do not require a small time step to be computed and so it is easer to violate the property.

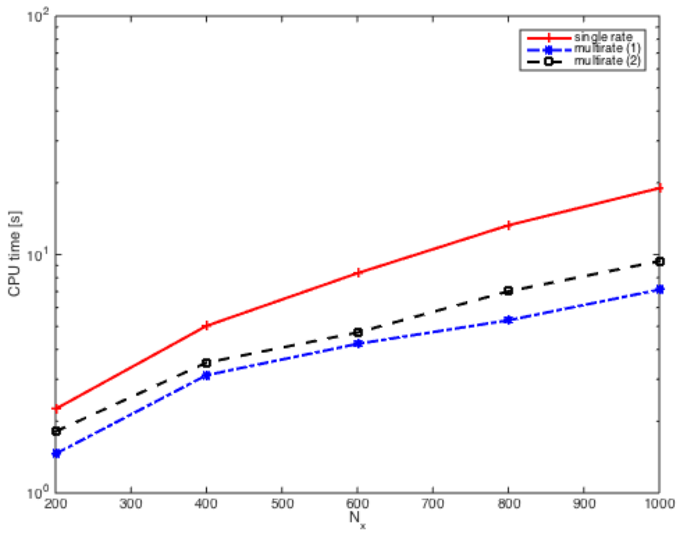

Figure 4.7: CPU times for the multirate method with the embedded error estimator (blue), with the Cubic extrapolator (black) and for the single rate method (red) for the linear advection problem.

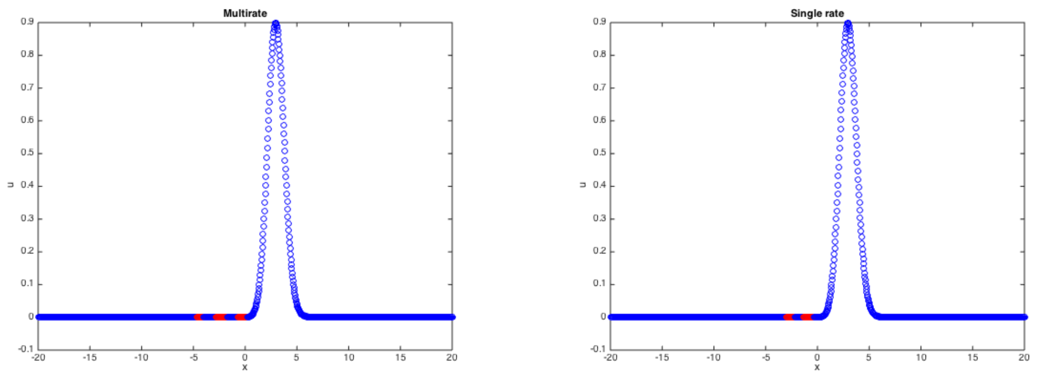

Figure 4.8: Solution at time $t = 3$, in red are marked the components with value less than 0 computed with the multirate algorithm (left) and with the single rate algorithm (right).

time [s]	Cubic interp.	Linear interp.	Single rate	Cubic extrap.
0.2	4.85×10^{-1}	4.83×10^{-1}	4.81×10^{-1}	4.87×10^{-1}
0.5	4.86×10^{-1}	4.86×10^{-1}	4.83×10^{-1}	4.88×10^{-1}
0.8	4.81×10^{-1}	4.82×10^{-1}	4.80×10^{-1}	4.87×10^{-1}
0.99	5.27×10^{-1}	5.31×10^{-1}	5.24×10^{-1}	5.3×10^{-1}

Table 4.11: Relative error in infinity norm at a fixed time between the exact solution and the solution computed with the multirate method with cubic Hermite interpolator (first column), the solution computed with multirate method with linear interpolator (second column), the solution computed with single-rate method (third column) and the solution computed with the multirate method with the cubic Hermite extrapolator as error estimator (fourth column) for the shock wave.

4.4 Burgers' equation

In this section we consider the Riemann problems applied to inviscid Burgers' equation with piecewise constant initial data. A Riemann problem is simply the conservation law together with particular initial data consisting of two constant states separated by a single discontinuity.

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}\left(\frac{1}{2}u^2\right) = 0, \qquad x \in [-1, 3], \quad t > 0 \tag{4.7}$$

As initial data we take:

$$u_0(x) = \begin{cases} u_l & x < 0 \\ u_r & x > 0 \end{cases} \tag{4.8}$$

First case: $\mathbf{u_l > u_r}$

In this case to discretize in space we have used the Finite Volume method with the Rousanov numerical flux. The interval time is $[0, 1]s$ with a number of cells equal to 400, the error tolerance is $[1 \cdot 10^{-4}, 1 \cdot 10^{-6}]$, the tolerance for the Newton solver is $1 \cdot 10^{-8}$. We took the initial size of the time step equal to $1 \cdot 10^{-2}s$. For the Riemann problem we used in this first case u_l equals to 1 while u_r equals to 0. As boundary condition we have $u(-1) = u_l$.

Figure 4.13 shows that the solution travels forward in space as time progresses. Also the interval of active components moves forward, but in this case the range stays constant and does not increase. Inside a global step, all fast components are then solved with

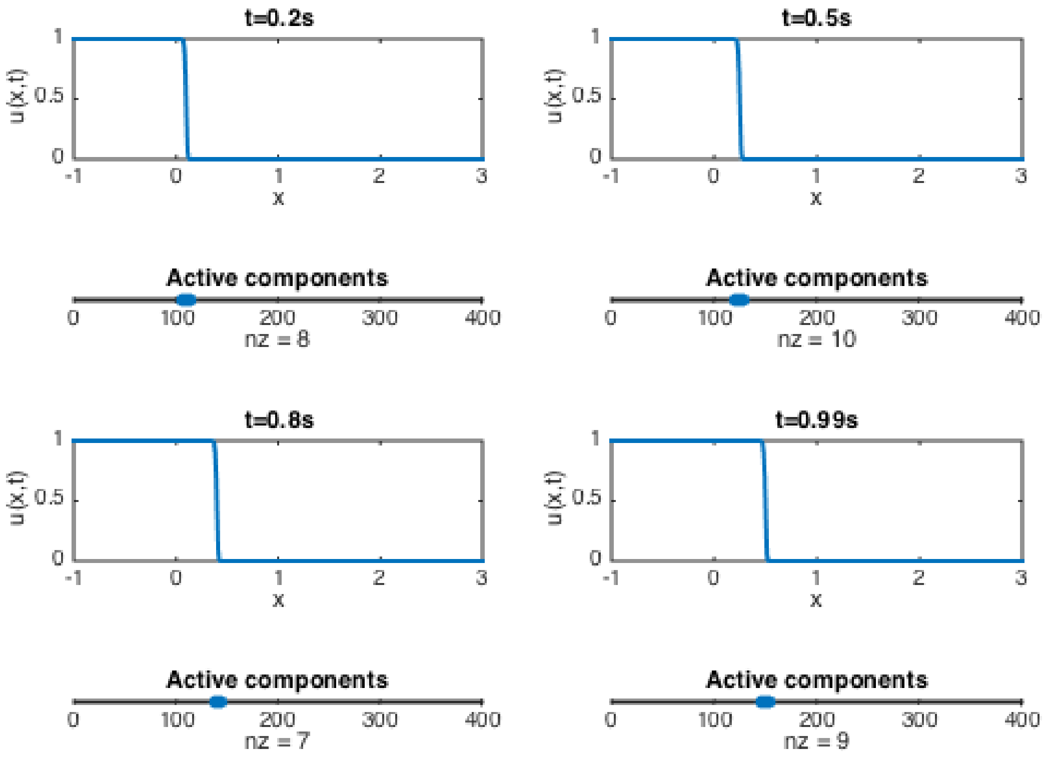

Figure 4.9: Multirate TR-BDF2 integration with Cubic interpolation for the shock wave.

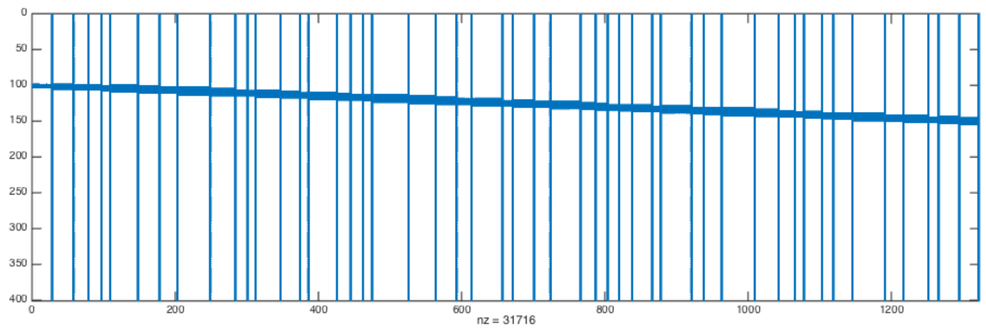

Figure 4.10: The components being computed at each time step by the multirate TR-BDF2 algorithm for the burgers equation that generates a shock wave.

time [s]	Cubic interp.	Linear interp.	Single rate	Cubic extrap.
0.2	4.37×10^{-5}	1.48×10^{-4}	1.34×10^{-5}	1.23×10^{-3}
0.5	2.76×10^{-1}	6.08×10^{-4}	3.5×10^{-5}	1.28×10^{-3}
0.8	6.65×10^{-4}	1.08×10^{-3}	3.63×10^{-5}	1.55×10^{-3}
0.99	9.18×10^{-4}	1.2×10^{-3}	3.59×10^{-5}	1.5×10^{-3}

Table 4.12: Relative error in infinity norm at a fixed time between the solution computed with the matlab solver and the solution computed with the multirate method with cubic Hermite interpolator (first column), the solution computed with multirate method with linear interpolator (second column), the solution computed with single-rate method (third column) and the solution computed with the multirate method with the cubic Hermite extrapolator as error estimator (fourth column) for the shock wave.

smaller step sizes. In this way a factor of 3.2 of improvement in execution time is obtained in comparison to the single rate scheme, as showed in Table 4.13. We denote by Multirate TR-BDF2 (1) the multirate algorithm with the error estimate proposed in [Ran16], and by Multirate TR-BDF2 (2) the multirate algorithm with the Cubic Hermite extrapolator as error estimate. Even if the second error estimator requires more time steps the execution time is lower. Anyway as the number of cells increases, the number of time steps increases too, so we do not have any computational gain, as showed in Figure 4.11.

We have reported the Courant numbers at each time step, computed with the following formula:

$$\max_{i=1,\dots,N_x} |f'(c_i^n)| \frac{h_n}{\Delta x} \qquad \forall n = 1, \cdots, N_T \tag{4.9}$$

where N_x is the total number of cells. In Figure 4.12 we represent in red the Courant values for the global steps. The latent components, where nothing happens, have a high Courant number. The active components instead have a very low Courant value because a small time step is required to capture the shock.

Method	Computation time	Numb. time steps
Singlerate TR-BDF2	$78.23s$	1008
Multirate TR-BDF2 (1)	$24.25s$	1378
Multirate TR-BDF2 (2)	$22.99s$	1558

Table 4.13: Computation time and number of time steps for shock wave .

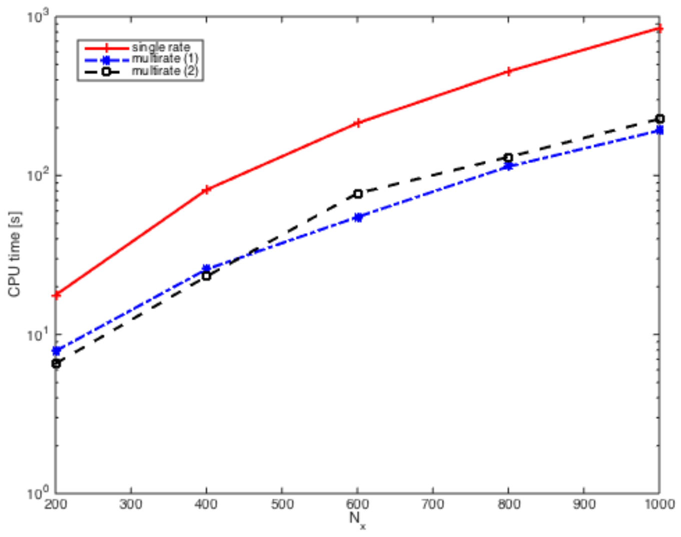

Figure 4.11: CPU times for the multirate with the embedded error estimator (blue), with the Cubic extrapolator (black) and single rate method (red) for the shock wave.

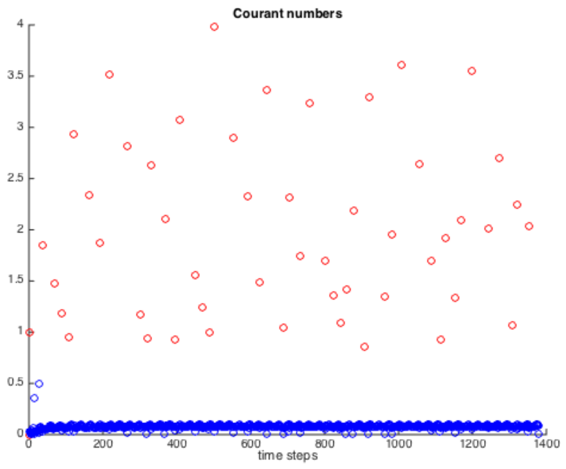

Figure 4.12: Courant numbers of each time step for the shock wave (with the red color are indicated the Courant numbers for the global time steps).

Second case: $\mathbf{u_l > u_r}$

To obtain a rarefaction wave we take $u_l = 0$ and $u_r = 1$. The boundary condition is $u(-1) = u_l$, while the other parameters are the same of the previous case.

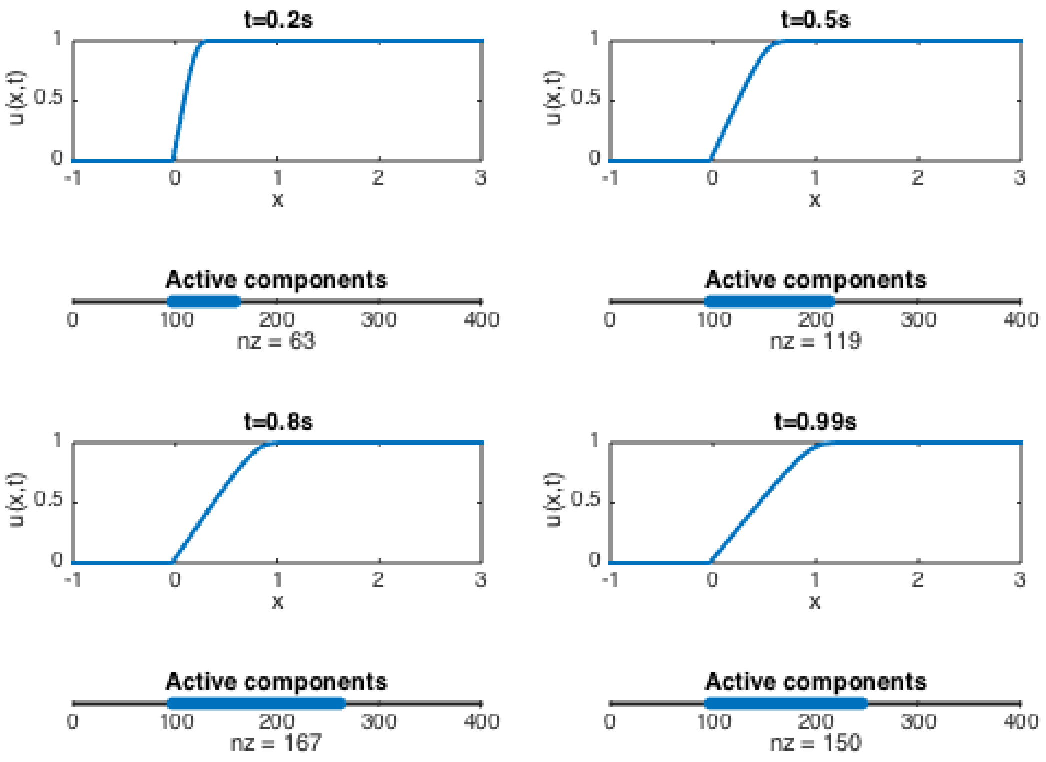

Figure 4.13: Multirate TR-BDF2 integration with Cubic interpolation for the rarefaction wave.

If we compare the results obtained with the shock and the rarefaction waves, we can see that for the same cells number the rarefaction problem requires less time steps. At each global step two refinement levels are computed, as showed in Figure 4.14 where we have reported the fraction of components involved and the Courant numbers of each time step. The first sub-level involves more and more components as time progresses, this is because the size of the global step increases and so the number of active components is bigger than the previous one. In the second sub-level the number of refined components is constant. In Figure 4.15 we have reported the CPU times varying the number of cells. For lower values the single rate method is comparable with the multirate algorithms, but as the number of cells increase, the multirate methods have better performance in terms of computational time.

time [s]	Cubic interp.	Linear interp.	Single rate	Cubic extrap.
0.2	3.59×10^{-1}	3.59×10^{-1}	3.59×10^{-1}	3.60×10^{-1}
0.5	3.16×10^{-1}	3.16×10^{-1}	3.14×10^{-1}	3.41×10^{-1}
0.8	2.93×10^{-1}	2.95×10^{-1}	2.85×10^{-1}	3.27×10^{-1}
0.99	2.84×10^{-1}	2.92×10^{-1}	2.83×10^{-1}	3.18×10^{-1}

Table 4.14: Relative error in infinity norm at a fixed time between the exact solution and the solution computed with the multirate method with cubic Hermite interpolator (first column), the solution computed with multirate method with linear interpolator (second column), the solution computed with single-rate method (third column)and the solution computed with the multirate method with the cubic Hermite extrapolator as error estimator (fourth column) for the rarefaction wave.

time [s]	Cubic interp.	Linear interp.	Single rate	Cubic extrap.
0.2	3.13×10^{-4}	4.42×10^{-4}	1.92×10^{-4}	6.93×10^{-4}
0.5	5.53×10^{-4}	9.91×10^{-4}	6.99×10^{-4}	2.10×10^{-2}
0.8	7.57×10^{-4}	1.75×10^{-3}	7.25×10^{-4}	1.33×10^{-2}
0.99	7.48×10^{-4}	1.25×10^{-3}	6.64×10^{-4}	1.10×10^{-2}

Table 4.15: Relative error in infinity norm at a fixed time between the solution computed by the MATLAB solver with ode45 method and the solution computed with the multirate method with cubic Hermite interpolator (first column), the solution computed with multirate method with linear interpolator (second column), the solution computed with single-rate method (third column) and the solution computed with the multirate method with the cubic Hermite extrapolator as error estimator (fourth column) for the rarefaction wave.

Method	Computation time	Numb. time steps
Singlerate TR-BDF2	$7.13s$	79
Multirate TR-BDF2	$2.99s$	169

Table 4.16: Computation time and number of time steps for the rarefaction wave.

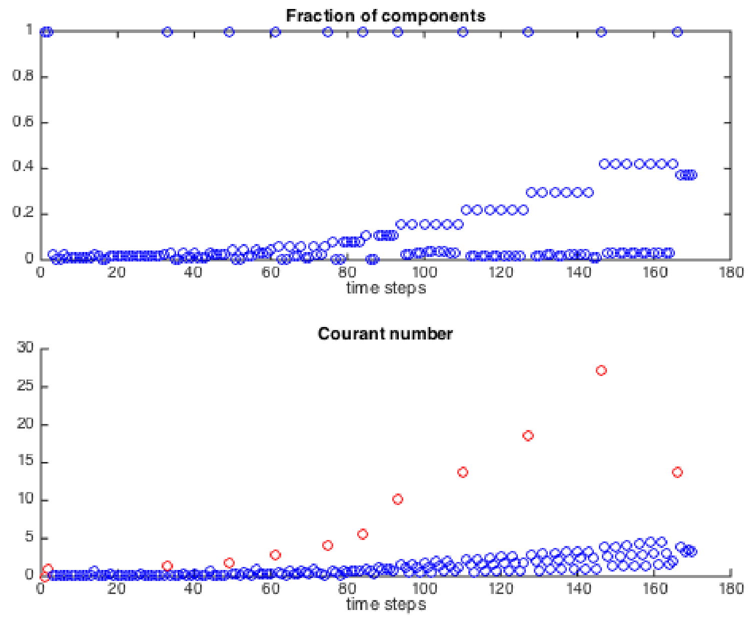

Figure 4.14: Fraction of components and Courant numbers of each time step for the rarefaction wave (with the red color are indicated the Courant numbers for the global time steps).

4.5 Buckley-Leverett equation

A more complex conservation law is the Buckley-Leverett equation:

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}\left(\frac{u^2}{u^2 + a(1-u)^2}\right) \qquad x \in [-1, 2], \quad t > 0 \tag{4.10}$$

where we take $a = \frac{1}{2}$.

To discretize in space we use the Finite Volume method with the Rusanov numerical flux. As initial condition we used the Riemann problem with $u_l = 1$ and $u_r = 0$. The interval time is $[0, 1]s$ with a number of cells equal to 300, the error tolerance is $[1 \times 10^{-6}, 1 \times 10^{-8}]$, the Newton tolerance is equal to 1×10^{-8}. The initial size of the time step is taken equal to $1 \cdot 10^{-2}s$.

Note the physical interpretation of the solution shown in Figure 4.16 for the appli-

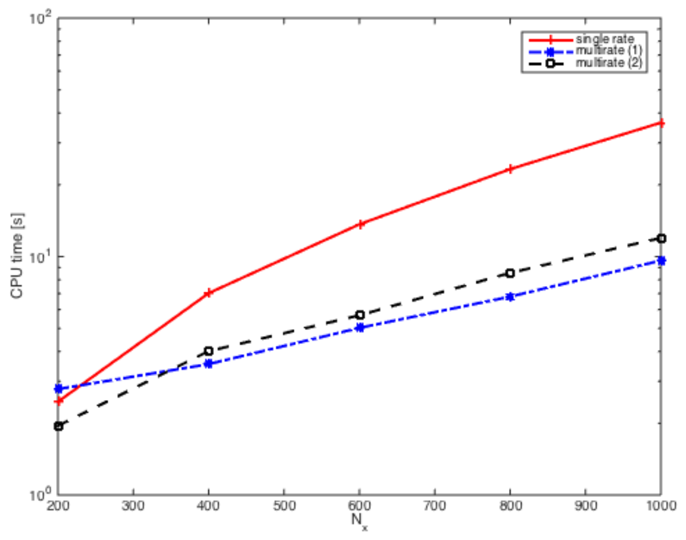

Figure 4.15: CPU times for the multirate method with the embedded error estimator (blue), with the Cubic extrapolator (black) and for the single rate method (red) for the shock wave.

cation described in Section 2.8. As the water moves in, it displaces a certain fraction u of the oil immediately. Behind the shock, there is a mixture of oil and water, with less and less oil as time goes on. At a production well, one obtains pure oil until the shock arrives, followed by a mixture of oil and water with diminishing returns as time goes on. It is impossible to recover all of the oil in finite time by this technique.

To compute the solution more time steps are necessary respect the others cases, as showed in Table 4.17 where the number of time steps and the computational time are reported.

In Figure 4.19 we plotted the execution time for different sizes of the space discretization. We pass from a factor of 2.7 of improvement with 200 cells to a factor of 5.9 with 500 cells if we compare the multirate algorithm with the single rate.

Method	Computation time	Numb. time steps
Singlerate TR-BDF2	$267.73s$	3371
Multirate TR-BDF2	$79.78s$	4780

Table 4.17: Computation time and number of time steps for the Buckley-Leverett equation.

We want to know if the multirate is a mass conservative method. At each time step, if the step is accepted and if it does not have to be refined, we check:

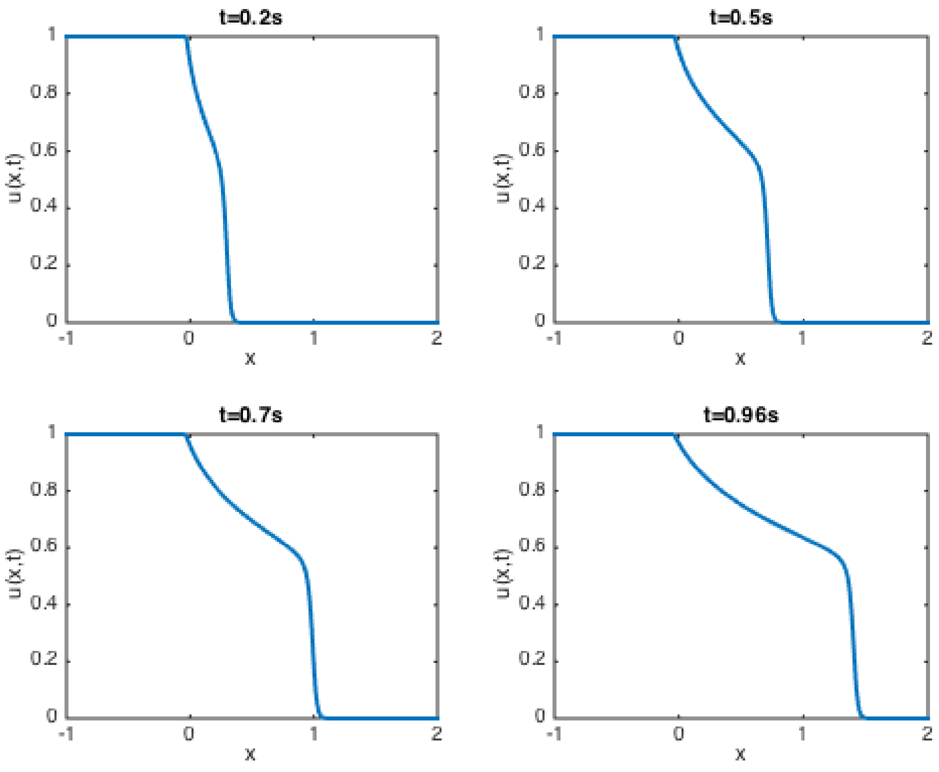

Figure 4.16: Solution computed with the multirate TR-BDF2 method for the Buckley-Leverett equation.

$$\left| \Delta x \sum_{i=1}^{N_x} c_i^{n+1} - \Delta x \sum_{i=1}^{N_x} c_i^n - (f(c_0(t)) - f(c_{N_x+1}(t)))h_n \right| < tol \quad \forall n = 0, \cdots, N_T \quad (4.11)$$

where N_x is the cells number, f is the flux function of the conservation law and *tol* is the prescribed Newton tolerance. We indicate with c_0 and c_{N_x+1} the boundary condition values. In this test case we have not mass conservation because for some time steps we interpolate the solution where it is not constant and the method does not control the flow exchanges between a cell and its neighbors. The maximum value where the inequality (4.11) is not verified is equals to 2.772×10^{-3}.

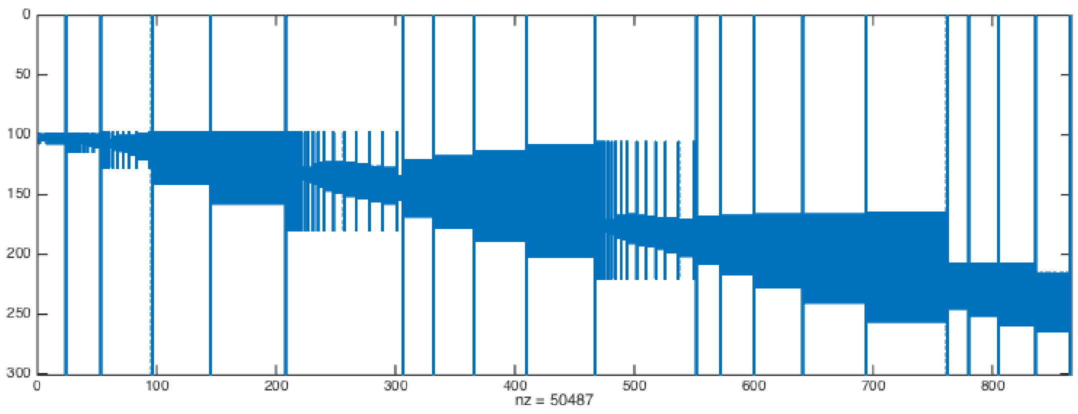

Figure 4.17: The components being computed at each time with the TR-BDF2 method for the Buckley-Leverett equation.

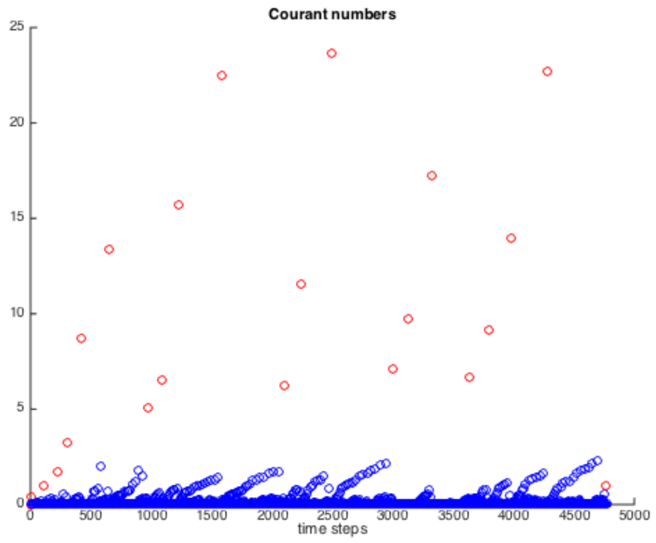

Figure 4.18: Courant numbers of each time step for the Buckely-Leverett equation (with the red color are indicated the Courant numbers for the global time steps).

time [s]	Cubic interp.	Linear interp.	Single rate	Cubic extrap.
0.2	4.11×10^{-6}	2.63×10^{-5}	6.14×10^{-6}	4.45×10^{-5}
0.5	5.82×10^{-6}	6.01×10^{-5}	7.09×10^{-6}	8.48×10^{-5}
0.8	6.27×10^{-6}	9.62×10^{-5}	7.34×10^{-6}	1.38×10^{-4}
0.99	1.05×10^{-5}	1.14×10^{-4}	7.33×10^{-6}	1.69×10^{-4}

Table 4.18: Relative error in infinity norm at a fixed time between the approximate solution computed with ode45 method and the solution computed with the multirate method with cubic Hermite interpolator (first column), the solution computed with multirate method with linear interpolator (second column), the solution computed with single rate method (third column) and the solution computed with the multirate method with the cubic Hermite extrapolator as error estimator (fourth column) for the Buckley-Leverett equation.

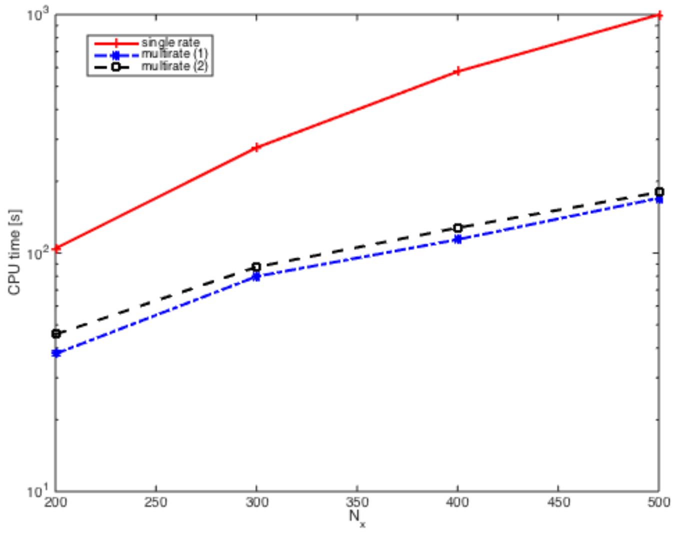

Figure 4.19: CPU times for the multirate method with the embedded error estimator (blue), with the Cubic extrapolator (black) and single rate method (red) for the Buckley-Leverett equation.

Conclusions

The objective of this thesis was to study the behavior of the multirate TR-BDF2 method when applied to nonlinear ODE systems and nonlinear conservation laws. This work was motivated by desire to improve the efficiency of numerical integration algorithms for very large systems of ordinary differential equations. A multirate method is characterized by three components: the numerical method to integrate in time, the interpolation scheme for the latent components and the time stepping strategy. We chose to use the self-adjusting partitioning strategy, which takes a global step for all the components. The system is then partitioned into a set of active and latent components based on the error estimate. The active components are refined with smaller time steps. The process is repeated until all components errors are within the specified tolerances. We have run the algorithm choosing different error estimators and investigated the accuracy of the method changing the interpolation scheme.

The multirate algorithm was tested on some applications with different time scale components and large size of the system. First of all we have tested the multirate algorithms on a geochemistry problem where the species react with different speeds. We have seen that in terms of accuracy the multirate method does not differ from the single rate method but we have not obtained a big computational gain. We have then extended our study to a larger geochemistry problem, with more species reacting with each other. In this case an improvement in terms of computational cost has been found. After that we have studied the multirate method for time discretization of conservation laws. First of all a linear case has been analyzed. The monotonicity property is violated in the multirate method, as in the single rate method, for high Courant numbers. Nonlinear case tests, where the method is effective, has been analyzed. In Burgers equations test, the multirate method has different behaviors and it seems to capture the difficulty of a shock wave: more time steps are necessary if we compare it to the rarefaction wave results. This involves smaller Courant numbers for the latent components of the shock wave, while the latent components of the rarefaction waves have higher Courant numbers. Finally, we have assessed the efficiency for the Buckley-Leverett equation where both a shock and a rarefaction wave are involved. In this test, we can see the benefits of the multirate method: as the number of cells increase the efficiency gains also improve.

The code can be improved in many aspects. We observed that monotone methods for scalar conservation laws satisfy a discrete entropy condition and they converge in a non oscillatory manner to the unique entropy solution. However, linear monotone methods are at most first order accurate, giving poor accuracy in smooth regions of the flow. It would be useful to study the conservation laws with higher order monotonized methods. Another possible further work is to select the set of active components relying not only on the accuracy of the solution, but also on the positivity violation. If some components do not respect the monotonicity property they can be recomputed with a smaller time step. Another way to obtain a positive solution is to use some unconditionally monotone variants of the TR-BDF2 method.

It would be very interesting to investigate also other different issues, for example, the Buckley-Leverett equation where at each cell, a geochemistry problem is present.

Appendix A

Code Structure

We describe the code used to solve a ODE with the multirate TR-BDF2 method. We explain the structure of the code and how it works. The code has been implemented completely in `C++`, to solve some linear algebra systems the Eigen library is used (`http://eigen.tuxfamily.org/index.php?title=MainPage`).

The code solves, with the multirate or single rate TR-BDF2 algorithm, an ODE system. Inside the code a geochemistry problem and three PDE are implemented (the case tests that we have analyzed in chapter 4). To generalize the code and to make it reliable, we have created some interfaces, for each one, more than one derived class has being implemented.

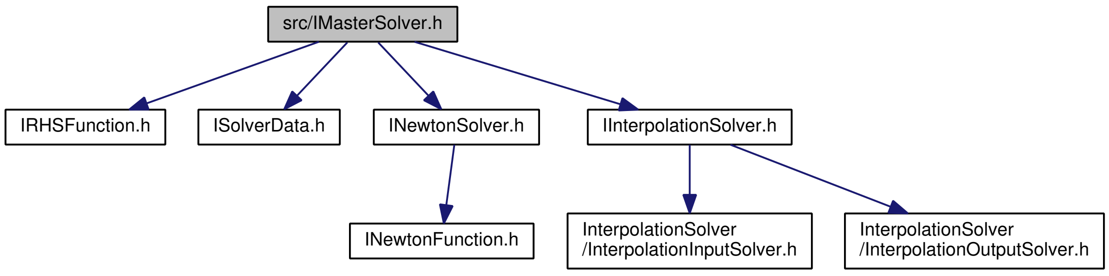

Figure A.1: Interfaces used inside the code to solve an ODE system.

The IMasterSolver interface allow to choose which algorithm we want to use to compute the solution. We can compare our multirate method with other time integration methods or other time stepping strategies. Its derived classes are the MultiRateTRBDF2Solver and the SingleRateTRBDF2Solver. As attributes of this interface there are some interfaces useful to solve the problem, and there are also some pure virtual methods that must to be implemented inside each derived class.

The INewtonSolver is used when the system is nonlinear and two possible different solver can be selected.

To store the output we implemented another interface named ISolverData. For both the methods, we created the respective derived classes. We need two different classes because we can store different objects for each master solver.

A.0.1 Multirate TR-BDF2 Solver

In this section we see in detail the multirate algorithm. The single rate class has the same structure, the only difference is that it does not require the interpolation. Every time we declare an object of this class, we have to set all the classical variables as the Newton tolerance or the initial and final times; but also: the right-hand side function of the problem (IRHSFunction), which Newton solver we want to use (INewtonSolver) and finally the interpolator for the latent components (IInterpolationSolver). Before to call the interpolator, we have to set its input that is the same for both the interpotators: the LinearInterpolation and the CubicHermiteInterpolation. The attributes present in the input interpolator class are:

- ref a boolean vector, where the dimension represents the number of components of the system. If a component has value equal to 0 it means that it must to be interpolated,
- y and z vectors at the initial and final times of the bigger time step,
- the time where the method has to compute the interpolation.

After the interpolation, the latent component values are stored inside an object of type InterpolationOutputSolver where is present the y vector. To advance by one time step, the class TRBDF2Solver is being implemented, inside there is a method that compute the solution for the active components with the TR-BDF2 method. Inside the class MultiRateTRBDF2Solver is present the method compute() with the time stepping strategy explained in section 3.2.

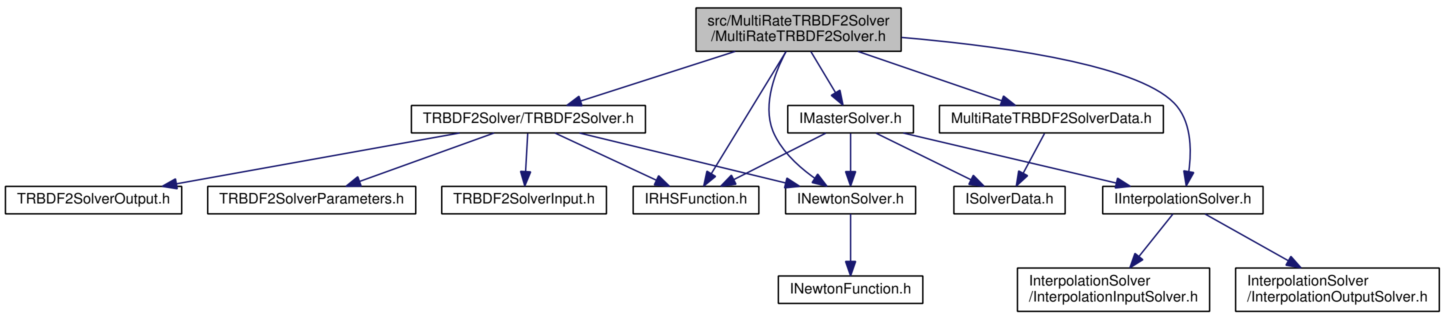

Figure A.2: Code structure for the multirate method.

The implementation of the multirate method can be outline with this algorithm:

Algorithm 1 MultiRateTRBDF2Solver.compute()

```
1: Initialization and declaration of vectors useful for the interpolation
2: initialization of the TR-BDF2 parameters
3: $n_{sub} = 0$
4: **while** $t_{current} < T_{final}[n_{sub}]$ **do**
5:     compute the Jacobian
6:     compute matrix-iteration $I - dhJ$
7:     compute mixed tolerance $\tau_r |u_{cur}| + \tau_a$
8:     **if** $n_{sub} > 0$ **then**
9:         call the interpolator
10:    **end if**
11:    call the TRBDF2Solver
12:    **if** Newton solver has converged **then**
13:        store the solution from TRBDF2SolverOutput in the output vector
14:        set $ref = Est > \delta mixed\_tol$
15:        **if** have to refine **then**
16:            initialize the input for the interpolation
17:            propose a new time step
18:            $n_{sub} += 1$
19:        **else if** refuse the time step **then**
20:            propose a new smaller time step
21:        **else**
22:            $t_{cur} = t_{new}$
23:            $u_{cur} = u_{new}$
24:            propose new time step and check $t_{cur} + h <= T_{final}[n_{sub}]$
25:        **end if**
26:    **else**
27:        propose a new smaller time step because Newton solver has failed
28:    **end if**
29:    **while** $|t_{cur} - T_{final}[n_{sub}]| < \epsilon_{mac}$ and $n_{sub} != 0$ **do**
30:        erase interpolator values for the current $n_{sub}$
31:        $n_{sub} -= 1$
32:    **end while**
33: **end while**
```

In the algorithm the Newton solver might not necessarily converge, in this case the method rejects the size of the time step and it proposes a smaller one.

At the end of the while loop, when we exit from a sub-level we have to pay attention at a special case: if the value t_{final} of the previous (or more) sub-level is equal to the current one, we have to set: the correct initial guess for the $\mathbf{z}$ variables, the new proposal time step and we have to exit from more than one sub-level. Every time that we compute the new size of the time step, even if we are in a sub-level, we must check that the time is not over the final time of the sub-level.

TRBDF2Solver

To solve a single step, the multirate class calls the method compute() of the TRBDF2Solver class. This method computes the solution for the active components, respect the $\mathbf{z}$ variables as explained in section 1.8. Inside the method we have to specify the Newton function that the Newton solver is going to solve.

INewtonSolver

Inside the INewtonSolver interface there is only a pure virtual method. We pass as reference the Jacobian that is a sparse matrix and the vector $\mathbf{y}$ that is the initial guess for the Newton solver. Inside the method, if it converges, it updates $\mathbf{y}$ with the new solution. The two derived classes are:

- BasicNewtonSolver. Its method, for each iteration, until the error is less than a prescribed tolerance, evaluates the Jacobian matrix and the right-hand side vector, calculates the error using the L^∞ norm, solves the system for the increment with the SparseLU solver and finally updates the solution by adding the increment.

- FixedJacobianNewtonSolver has inside a method that it is the same of the Basic-NewtonSolver, the only difference is that it uses the same Jacobian evaluation for all the iterations that are necessary.

A.0.2 The right-hand side function

In the code, to solving different types of problem, an interface has been implemented, its name is IRHSFunction. Inside the derived classes we have implemented the method that evaluates the right-hand side of the problem and another one that evaluates the Jacobian, it can be analytic or calculates with finite differences. In these classes is present the overloading of the two methods because in a sub-level we have to evaluate the right-hand side only for a smaller set of components and the method has to know the indices of the active components. In Figures A.3, A.4 and A.5 we have reported the code structure for the numerical experiments showed in chapter 4.

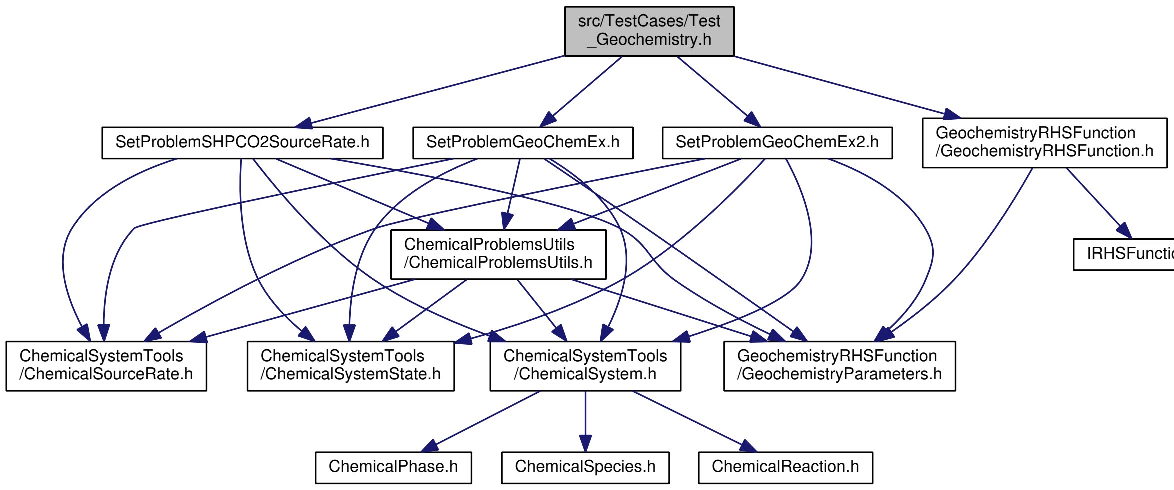
src/TestCases/Test _Geochemistry.h
SetProblemSHPCO2SourceRate.h
SetProblemGeoChemEx.h
SetProblemGeoChemEx2.h
GeochemistryRHSFunction /GeochemistryRHSFunction.h
ChemicalProblemsUtils /ChemicalProblemsUtils.h
IRHSFunction.h
ChemicalSystemTools /ChemicalSourceRate.h
ChemicalSystemTools /ChemicalSystemState.h
ChemicalSystemTools /ChemicalSystem.h
GeochemistryRHSFunction /GeochemistryParameters.h
ChemicalPhase.h
ChemicalSpecies.h
ChemicalReaction.h

Figure A.3: Code structure for the geochemistry problem.

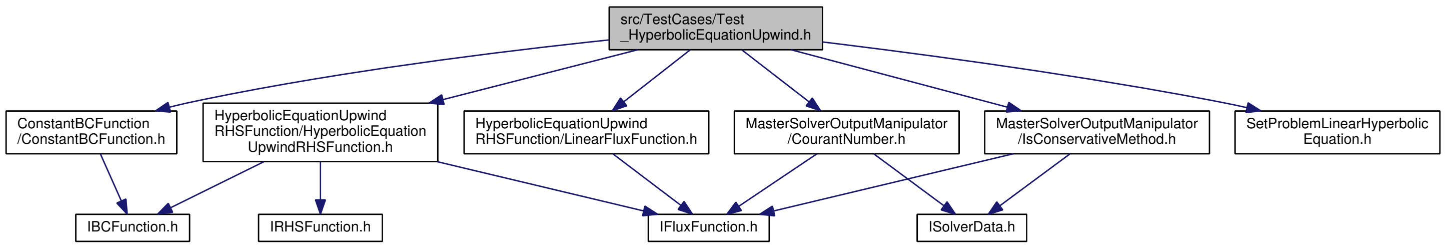
src/TestCases/Test _HyperbolicEquationUpwind.h
ConstantBCFunction /ConstantBCFunction.h
HyperbolicEquationUpwind RHSFunction/HyperbolicEquation UpwindRHSFunction.h
HyperbolicEquationUpwind RHSFunction/LinearFluxFunction.h
MasterSolverOutputManipulator /CourantNumber.h
MasterSolverOutputManipulator /IsConservativeMethod.h
SetProblemLinearHyperbolic Equation.h
IBCFunction.h
IRHSFunction.h
IFluxFunction.h
ISolverData.h

Figure A.4: Code structure for the linear conservation laws using the Upwind method.

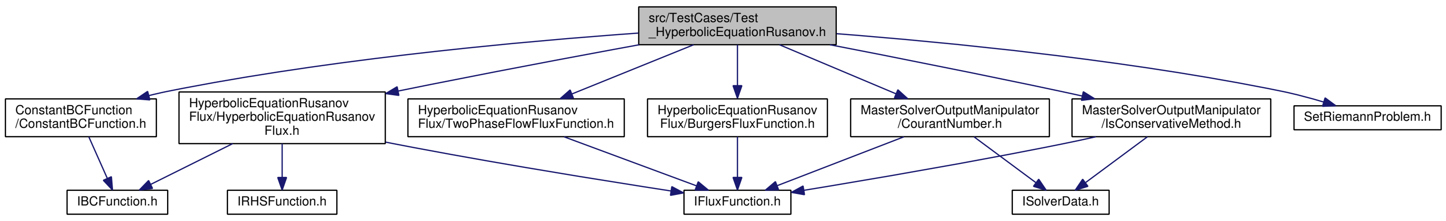
src/TestCases/Test _HyperbolicEquationRusanov.h
ConstantBCFunction /ConstantBCFunction.h
HyperbolicEquationRusanov Flux/HyperbolicEquationRusanov Flux.h
HyperbolicEquationRusanov Flux/TwoPhaseFlowFluxFunction.h
HyperbolicEquationRusanov Flux/BurgersFluxFunction.h
MasterSolverOutputManipulator /CourantNumber.h
MasterSolverOutputManipulator /IsConservativeMethod.h
SetRiemannProblem.h
IBCFunction.h
IRHSFunction.h
IFluxFunction.h
ISolverData.h

Figure A.5: Code structure for the the nonlinear conservation laws using the Rusanov numerical flux.

Bibliography

[And79] J.F. Andrus. Numerical solution of systems of ordinary differential equations separated into subsystems. *SIAM Journal of Numerical Analysis*, 16:605–611, 1979.

[BCF$^+$85] R.E. Bank, W.M. Coughran, W. Fichtner, E.H. Grosse, D.J. Rose, and R.K. Smith. Transient simulation of silicon devices and circuits. *IEEE Transactions on Electron Devices*, 32:1992–2007, 1985.

[BGG12] A. Bermùdez and L. M. Garcìa-Garcìa. Mathematical modeling in chemistry. application to water quality problems. *Applied Numerical Mathematics*, 62(4):305 – 327, 2012.

[BR15] L. Bonaventura and A. Della Rocca. Monotonicity, positivity and strong stability of the TR–BDF2 method and of its SSP extensions. *MOX-Report No. 56/2015*, 2015.

[But63] J.C. Butcher. Coefficients for the study of Runge-Kutta integration processes. *Journal of the Australian Mathematical Society*, 3(02):185–201, 1963.

[CH52] C.F. Curtiss and J.O. Hirschfelder. Integration of stiff equations. *Proc. Nat. Acad. Sci*, 38(235):1, 1952.

[Fok15] P.K. Fok. A linearly fourth order multirate Runge–Kutta method with error control. *Journal of Scientific Computing*, pages 1–19, 2015.

[FS04] L. Ferracina and M.N. Spijker. Stepsize restrictions for the total-variation-diminishing property in general Runge–Kutta methods. *SIAM journal on numerical analysis*, 42(3):1073–1093, 2004.

[GKC13] F.X. Giraldo, J.F. Kelly, and E.M. Constantinescu. Implicit-explicit formulations of a three-dimensional nonhydrostatic unified model of the atmosphere (NUMA). *SIAM Journal of Scientific Computing*, 35(5):1162–1194, 2013.

[GR13] E. Godlewski and P.A. Raviart. *Numerical approximation of hyperbolic systems of conservation laws*, volume 118. Springer Science & Business Media, 2013.

[GW84] C.W. Gear and D.R. Wells. Multirate linear multistep methods. *BIT Numerical Mathematics*, 24:484–502, 1984.

[HNW87] E. Hairer, S.P. Norsettl, and G. Wanner. Solving ordinary differential equation I: nonstiff problems. *Springer Ser. in Comput. Math*, 8, 1987.

[HS96] M.E. Hosea and L.F. Shampine. Analysis and implementation of TR-BDF2. *Applied Numerical Mathematics*, 20:21–37, 1996.

[Ise96] A. Iserles. *A first course in the numerical analysis of differential equations.* Cambridge texts in Applied Mathematics, 1996.

[Kra91] J.F.B.M. Kraaijevanger. Contractivity of Runge–Kutta methods. *BIT Numerical Mathematics*, 31(3):482–528, 1991.

[Lam91] J.D. Lambert. *Numerical methods for ordinary differential systems: the initial value problem.* Wiley, 1991.

[LeV92] R.J. LeVeque. *Numerical methods for conservation laws*, volume 132. Springer, 1992.

[PR74] A. Prothero and A. Robinson. On the stability and accuracy of one-step methods for solving stiff systems of ordinary differential equations. *Mathematics of Computation*, 28(125):145–162, 1974.

[Ran16] A. Ranade. *Multirate Algorithms Based On DIRK Methods For Large Scale System Simulation.* PhD thesis, Politecnico di Milano, 2016.

[SG79] L.F. Shampine and C.W. Gear. A user's view of solving stiff ordinary differential equations. *SIAM review*, 21(1):1–17, 1979.

[Sha77] L.F. Shampine. Stiffness and nonstiff differential equation solvers, ii: detecting stiffness with runge-kutta methods. *ACM Transactions on Mathematical Software (TOMS)*, 3(1):44–53, 1977.

[Sha81] L.F. Shampine. Type-insensitive ODE codes based on implicit A–stable formulas. *Mathematics of Computation*, 36(154):499–510, 1981.

[SHV07] V. Savcenco, W. Hundsdorfer, and J.G. Verwer. A multirate time stepping strategy for stiff ordinary differential equations. *BIT Numerical Mathematics*, 47:137–155, 2007.

[SW06] G. Söderlind and L. Wang. Evaluating numerical ODE/DAE methods, algorithms and software. *Journal of Computational and Applied Mathematics*, 185:244–260, 2006.

[TB15] G. Tumolo and L. Bonaventura. A semi-implicit, semi-Lagrangian, DG framework for adaptive numerical weather prediction. *Quarterly Journal of the Royal Meteorological Society*, 141:2582–2601, 2015.

[VTB$^+$07] A. Verhoeven, B. Tasić, T.G.J. Beelen, E.J.W. ter Maten, and R.M.M. Mattheij. Automatic partitioning for multirate methods. In *Scientific Computing in Electrical Engineering*, pages 229–236. Springer, 2007.

[WH91] G Wanner and E Hairer. *Solving ordinary differential equations II*, volume 1. Springer-Verlag, Berlin, 1991.

Ringraziamenti

Innanzitutto vorrei ringraziare il Professor Luca Bonaventura e la Dottoressa Anna Scotti per essere sempre stati disponibili e pazienti, per avermi guidato e consigliato saggiamente durante questo lavoro. Poi ringrazio il Professor Luca Formaggia per avermi proposto uno stage all'IFPEN, lì ho incontrato delle persone con grande passione per il loro lavoro.

Je remercie Thibault pour sa disponibilité et pour la passion qu'il m'a transmise pendant les quatre mois de stage, ainsi que Antony pour ses conseilles et sa patience. J'ai trouvé l'IFPEN une ambiance de travail vraiment accueillante et heureuse.

Un grandissimo *grazie* va ai miei genitori, loro mi hanno sempre sostenuta in tutte le mie scelte e anche sopportata durante le mie catastrofissime ansie. Grazie mamma per essere sempre presente, per mettere sempre me al primo posto davanti a tutto. Grazie papà per essere il mio punto di riferimento e per avermi dato dei preziosi consigli durante questi cinque anni di università. Vi voglio davvero un immenso bene!

Un grazie di cuore va a tutti gli Ing. Mat. Specialmente vorrei ringraziare i compagni dell'aula tesiti, Giorgio e Jacopo che, in questi 3 mesi, mi sono sempre stati accanto, insieme abbiamo condiviso risate, ansie, preoccupazioni ed errori di Latex. Lollo e Nicola, i due calcolisti che mi hanno sopportato durante le ore di lezione e che non mi hanno mai negato un aiuto. Anna, Cate, Marta e Vale per essere state delle amiche preziose durante questi anni di università. E ringrazio anche Tommy, non so per cosa, ma grazie! Infine ringrazio Giulia, non so come avrei fatto senza di te a Parigi!

Ringrazio tutti i miei amici di Caltanissetta, con loro sono cresciuta, con loro ho affrontato i momenti più felici e spensierati, in particolare ringrazio Anita per essere la mia amica di sempre, anche a distanza.

Infine ringrazio la persona che più di tutti ha creduto in me, più di quanto facessi io, sei stato la mia roccia. Anche a distanza, hai saputo tirarmi sù di morale nei momenti più difficili, e regalatomi dei sorrisi che porterò per sempre nel mio cuore. Tu sei *"la luce dei miei occhi"*!