Partitioned Multirate Time Integration for Coupled Systems of Ordinary Differential Equations

Joost W. M. Kranenborg

Master's thesis
2018:E29

Lund Institute of Technology
Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Numerical Analysis

Centrum Scientiarum Mathematicarum

Partitioned multirate time integration for coupled systems of ordinary differential equations

Author: Joost Kranenborg

Abstract

In this thesis a new method for the numerical solution of coupled systems of ordinary differential equations is investigated. To understand this new method, the workings of existing Waveform iteration methods, in particular the Jacobi and Gauß-Seidel methods were explored first. These methods were introduced to be able to exploit multirate behaviour in systems and use an iterative procedure to successively approximate the solution of the problem. The new multirate time integration method is similar to these methods, but introduces a new way of parallelising the time integration, thereby improving performance over the existing methods. In this work an analysis of the performance of the Jacobi and Gauß-Seidel method when applied to linear systems is done, both for a singlerate setting as well as a multirate setting. This analysis is then compared to numerical simulations of the waveform iteration methods, as well as the new multirate time integration method. The results look promising for the new method, having a superior performance compared to the waveform iteration methods for almost all test cases when applied to the heat equation.

keywords: *Waveform iteration, waveform relaxation, multirate time integration, parallel computing*

Supervisors: As. prof. Philipp Birken, Msc. Peter Meisrimel
Examiner: Prof. Eskil Hansen

Acknowledgements

I would like to thank my supervisors Philipp Birken and Peter Meisrimel for motivating me to do my best on this work. During no previous semester have I learnt as much as I did during my thesis, which I attribute a great deal to the excellent support I have gotten from you.
I would also like to express my gratitude to everyone at F-sektionen, Teknologkåren vid LTH as well as Lundakarnevalen whom I have had the pleasure to meet and work with. My life as a student wouldn't have been half as interesting without these organisations and everyone I have met there.
Finally, I would like to thank my parents who completely unannounced came to my thesis defence. Their encouragement throughout my studies really means a great deal to me.

Contents

Chapter 1

Introduction

Many real world problems can be described by differential equations. A big part of numerical analysis revolves around finding solutions to these equations. Sometimes a system can be better described by multiple smaller subsystems working together. In this work we will explore the waveform iteration methods, designed for the numerical solution of these systems. Alongside two existing methods, the Jacobi and Gauß-Seidel method, a new multirate method proposed by Philipp Birken will be presented. The performance of this method will be compared to the Jacobi and Gauß-Seidel methods.

1.1 Motivation and problem description

When a system is made up of two or more subsystems, it can be beneficial to be able to use the different properties of the subsystems to design more efficient methods. Some of the properties that can be exploited include, but are not limited to, differing time scales between the problems or the requirement to use different existing codes for the different subsystems. In this thesis, we will consider a system of two coupled ODEs of the following form:

$$\begin{cases} \text{System A} & \dot{x}(t) = f(x(t), y(t)) & x(0) = x_0 \\ \text{System B} & \dot{y}(t) = g(x(t), y(t)) & y(0) = y_0 \end{cases} \quad t \in [t_0, t_e]. \tag{1.1}$$

Here $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$ are the system's state variables and $\dot{x} \in \mathbb{R}^n$,$\dot{y} \in \mathbb{R}^m$ the time derivatives of the state. The two maps $f : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^n$ and $g : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^m$ are the maps describing the derivatives of each subsystem. Both functions $x(t)$ and $y(t)$ are functions of time. For the rest of this thesis the systems will be referred to as system "A" and system "B" as described above.

Waveform iteration methods seek to exploit the properties of these coupled systems. Waveform relaxation methods, of which waveform iteration methods are a subclass, were first proposed in the PHD-thesis by Lelarasmee in 1982 [1] as a means of simulating integrated circuits. The methods work by an iterative procedure, generating sequences of functions, also called waveforms, that represent the solution of one of the subsystems. Combining the solution of the subsystems, we then have a numerical solution of the full system.

Some waveform iteration methods can be implemented in a parallel way. These methods are however not fully exploiting all possibilities regarding par-

allelisation. In this thesis a new method proposed by Philipp Birken is implemented and analysed. This new method introduces parallelisation on a lower level compared to conventional waveform iteration methods with the idea that parallelisation will decrease computational time. To be viable the method should fulfil the following criteria:

- **Partitioned**: Contrary to the monolithic approach, where the full system is discretised and solved together, we want to have a partitioned method separating the solvers for the different systems. The exchange of information between the systems is achieved by coupling.
- **Order of convergence ≥ 2**: Any less of a convergence order is not very useful in modern scientific computing.
- **Implicit**: We need the ability to solve stiff problems. Many interesting applications of these methods are stiff.
- **Multiscale**: Both systems should use independent time steps. This is again a property of the given problem we want to exploit to increase performance.
- **Parallel**: By parallelising the method we hope to decrease the time needed to solve the problem.
- **Time-adaptivity**: We would like the ability for the method to adapt the step size in order to control the error. This is however not considered in this thesis.

1.2 Overview of content

In chapter 2 we are going to lay out the basics of the numerical solution of ordinary differential equations and parallel computing which is needed in subsequent chapters. This chapter is not meant as an introduction to the topics for someone unfamiliar with numerical analysis , rather it should serve as a common ground for the rest of the thesis. In chapter 3 the basics of the waveform iteration methods will be discussed, including an analysis of the convergence of these methods when applied to linear systems. The new multirate method is introduced in chapter 4. In chapter 5 we compare the results from these three methods, comparing the waveform iteration methods to the analysis done in chapter 3 and comparing the new multirate method to the Jacobi and Gauß-Seidel methods. Finally in chapter 6 we present conclusions made based on the results and analysis. Also possible improvements and continuations of this work are proposed. There is also a short appendix containing the numerical integration methods used.

1.3 Literature review

This master thesis is based on different works in the field of waveform iteration methods. The main source is the book on waveform relaxation methods for parabolic problems by Vandewalle [2]. This book explores a new numerical method combining multigrid methods and waveform iteration methods. It

includes a comprehensive background on waveform relaxation methods and its applications. A thorough analysis of the convergence rates of waveform relaxation methods applied to linear systems is done in two papers by Nevanlinna in [3] and [4]. In the first paper the method is presented and analysed in the continuous case. Also the stopping criterion of the iteration is discussed. In the second paper the discrete case is analysed. Also some additional measures are proposed to speed up the computation of the solution. Finally the PHD thesis by White [5] gives great insight in the general convergence properties of waveform relaxation methods. Also the convergence of the methods in a multirate setting is analysed.

Chapter 2

Prerequisites

In this section we give a brief background on the numerical solution of ordinary differential equations as well as a short introduction on parallel computing.

2.1 Numerical solution of ODEs

In this section we give a short introduction to the numerical solution of ordinary differential equations (ODEs). This introduction is meant to introduce definitions used later in this thesis, not as an introduction to numerical analysis for someone unfamiliar with the subject. A more thorough explanation can be found in any introductory book on numerical analysis, for example the book by Arieh Iserles [6].

Ordinary differential equations can be written as

$$\dot{x} = f(x,t), \quad x(t_0) = x_0 \tag{2.1}$$

where $\dot{x} \in \mathbb{R}^n$ is the time derivative of the different state variables $x \in \mathbb{R}^n$. The map $f : (\mathbb{R}^n \times [t_0, t_e]) \mapsto \mathbb{R}^n$ describes the relation between the system's state and its derivatives. For completeness we also require f to be Lipschitz continuous. Most numerical methods for ODEs solve equation (2.1) by generating a sequence $x_i \approx x(t_i)$ approximating the real solution on a grid of specific points in time t_i. Between these grid points we can approximate the solution using interpolation. Using an equidistant grid, meaning a constant step size h, we can write down the simplest numerical method, called the explicit Euler method:

$$x_{i+1} = x_i + hf(x_i, t_i).$$

It is an explicit method, meaning defines the new point x_{i+1} only in terms of x_i, t_i or even earlier points in time. A method is called implicit if an equation needs to be solved, for instance in the implicit Euler method:

$$x_{i+1} = x_i + hf(x_{i+1}, t_{i+1}).$$

A list of the methods used throughout this thesis can be found in appendix A.

All numerical methods introduce an error in the solution. The error introduced by a single step, assuming the step starts at the exact solution: $x_i = x(t_i)$, is called the local truncation error and can be written as

$$\delta_{i+1}^h = x_{i+1} - x(t_{i+1}). \tag{2.2}$$

A method is called consistent if $\delta_i^h \to 0$ as $h \to 0$ with order p if $\delta_i^h = \mathcal{O}(h^{p+1})$. The global error of a method is the error e^h between the numerical solution and the real solution at the end of the time interval t_e:

$$e^h = x_e - x(t_e).$$

The global error for a method of order p decreases as $\mathcal{O}(h^p)$. Both the implicit Euler and explicit Euler methods are of order 1.

For a method to be useful it needs to be convergent. Convergence here means that

$$\lim_{h \to 0^+} \max_{i=0,1,\ldots,t^*/h} \|x_{i,h} - x(t_i)\| = 0$$

where $t^* > t_0$ and $x_{i,h}$ denotes the resulting solution on an equidistant grid with step size h [6]. A necessary condition for convergence is that the method is consistent with an order of $p > 0$. Consistency alone is not enough; a method also needs to be stable. For so called *stiff* problems, stability can be a real issue. This is where implicit methods come into play since they are generally more stable than explicit methods. For the rest of thesis, we assume that the methods used to solve the ODEs are stable for the given problem and time step.

2.2 Parallelisation

Thus far we have introduced basic numerical methods for solving ODEs on the form of equation (2.1). These methods are all sequential methods meaning all calculations are done in sequence. To decrease the time taken to calculate the solution, one would like to parallelise the computation, meaning multiple calculations are done simultaneously. In this section we give a short introduction to parallel computing. A more thorough introduction to parallel computing techniques can be found in the book on concurrent scientific computing by E. F. van de Velde [7]. To understand how parallelisation works we first need to have a rough understanding of how the hardware works. In most modern computers the processor consists of multiple cores. One core is able to execute instructions independently of the other cores. This means that, for instance a four core processor can simultaneously execute four different instructions. The same goes for large supercomputers which consist of many processors working together where very processor can execute instructions independently. The instructions being executed come from so called *processes*. A Process is a set of instructions, coming from a computer program, being processed by a processor. A computer program consists of one or more of these processes. If a program uses multiple processes, the processes can utilize different cores in the processor to execute instructions in parallel. Processes can communicate with each other using different means of communication. In numerical analysis we want to exploit multiprocessing techniques for parallelisation, meaning splitting the solution procedure of the problem into multiple independent instructions executed by different processes.

2.2.1 Communication between processes

Sometimes processes need to exchange data with each other. There are two main paradigms when it comes to data sharing. There is the shared memory

paradigm meaning both processes have access to the same memory, and there is the message passing paradigm where the processes have separate space in memory meaning all communication is done by sending messages between the processes. The results in this thesis are based on an implementation using message passing.

It is important to realise that whenever a sequential algorithm is parallelised the total amount of work needed to be done increases. This comes from the extra communication introduced in the parallelisation which was not there in the sequential algorithm. To parallelise code efficiently we want a high computation to communication ratio meaning that we want the time spent communicating be negligible compared to the time spent computing. This means that the parallel methods will be more efficient for more difficult problems, for instance coupled FEM problems, than simple problems like solving two coupled 1D ODEs.

Chapter 3

Waveform iteration

In this section we give a definition of the waveform iteration methods, in particular the Jacobi and Gauß-Seidel methods. Also an analysis of the convergence rates for the different methods is presented. Finally some details regarding the implementation are presented, as well as some alternatives for improving the waveform iteration methods that are not covered further in this thesis.

3.1 Method descriptions

All methods described in this thesis stem from a similar idea, namely to divide the whole time domain into so called *macrosteps*, in some literature also called *windows*. These macrosteps are needed as a common ground for the integration of each subsystem. Waveform iteration is a method for numerically approximating and iteratively approaching the subsystems' solutions on these macrosteps. The idea is to solve each subsystem using information from the other system available prior to integration. The solution for the subsystems at iteration $K+1$ are governed by the following equation:

$$\begin{cases} \dot{x}^{K+1} = f(x^{K+1}, y^*) \\ \dot{y}^{K+1} = g(x^*, y^{K+1}) \end{cases} . \tag{3.1}$$

Here the iterates x^{K+1} and y^{K+1} are all functions defined on $t \in [T_0, T_1]$; the domain of the macrostep. The functions $y^*(t) : \mathbb{R} \mapsto \mathbb{R}^m$ and $x^*(t) : \mathbb{R} \mapsto \mathbb{R}^n$ are functions on the whole macrostep as well. The successive application of this iteration scheme results in sequences of functions $\{x(t)_K\}_{K=1}^{\infty}$ and $\{y(t)_K\}_{K=1}^{\infty}$. We want these sequences to converge to some limit $x(t)$ and $y(t)$. For a general waveform iteration method these may be defined in many different ways. We will present two different methods here, the Jacobi method and the Gauß-Seidel method.

3.1.1 Jacobi iteration

The Jacobi method is obtained by setting $x^* = x^K$ and $y^* = y^K$. This results, given initial data x^0 and y^0, in the following iteration [2]:

$$\begin{cases} \dot{x}^{K+1} = f(x^{K+1}, y^K) \\ \dot{y}^{K+1} = g(x^K, y^{K+1}) \end{cases} . \tag{3.2}$$

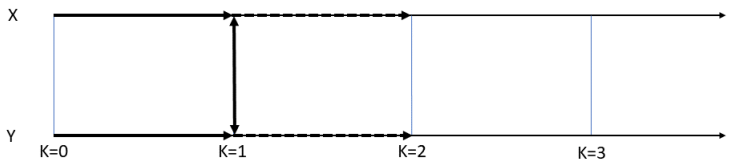

Figure 3.1: Diagram of the Jacobi iteration.

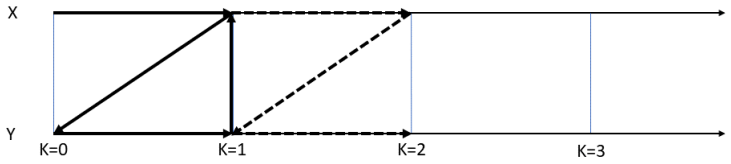

Figure 3.2: Diagram of the Gauß-Seidel iteration.

Looking at the equation we see that to update from iteration K to $K+1$ each subsystem only needs information from the previous iteration of the other subsystem. In an implementation of Jacobi, the subsystems are solved separately and information is exchanged at the end of the macrostep. This means that this method is inherently parallel and therefore easily implemented in a parallel fashion. This is depicted in figure 3.1. Also we recognize the similar structure between the Jacobi method for linear equation systems and equation (3.2), therefore we call this iteration scheme Jacobi iteration.

3.1.2 Gauß-Seidel iteration

For the Gauß-Seidel method we set $x^* = x^K$ and $y^* = y^{K+1}$. This results in the following iteration [2]:

$$\begin{cases} \dot{x}^{K+1} = f(x^{K+1}, y^K) \\ \dot{y}^{K+1} = g(x^{K+1}, y^{K+1}) \end{cases} . \tag{3.3}$$

Looking at the equation we see where the name comes from as the triangular structure of the system reminds us of the Gauß-Seidel iteration for solving linear systems. The difference in the definition of y^* compared to the Jacobi method changes the method significantly. The inherent parallelism of the Jacobi method is lost in the Gauß-Seidel approach. Figure 3.2 gives a schematic view of the iteration.

3.1.3 Comparing Gauß-Seidel and Jacobi

In a 2-dimensional system there is an interesting relationship between the Gauß-Seidel method and the Jacobi method, namely that the Jacobi method has twice the computational cost of the Gauß-Seidel method. To visualise this we rewrite equation (3.1) using only the independent variables with

$$\varphi(y^*) = f(x^K, y^*)$$

and

$$\psi(x^*) = g(x^*, y^K).$$

We denote the Gauß-Seidel and Jacobi iterates with the subscripts $_{GS}$ and $_J$ respectively. Given initial data we can write down the subsequent iterates for Gauß-Seidel,

$$x^1_{GS} = \varphi(y^0_{GS}), \quad x^2_{GS} = \varphi(y^1_{GS}), \quad x^3_{GS} = \varphi(y^2_{GS})$$
$$y^1_{GS} = \psi(x^1_{GS}), \quad y^2_{GS} = \psi(x^2_{GS}), \quad y^3_{GS} = \psi(y^3_{GS})$$

and for Jacobi:

$$x^1_J = \varphi(y^0_J), \quad x^2_J = \varphi(y^1_J), \quad x^3_J = \varphi(y^2_J)$$
$$y^1_J = \psi(x^0_J), \quad y^2_J = \psi(x^1_J), \quad y^3_J = \psi(y^2_J).$$

Comparing the first x iterates we see that

$$x^1_J = \varphi(y^0_J) = \varphi(y^0_{GS}) = x^1_{GS}.$$

Using substitution we see that

$$y^2_J = \psi(x^1_J) = \psi(x^1_{GS}) = y^1_{GS}.$$

We can do the same for

$$x^3_J = \varphi(y^2_J) = \varphi(y^1_{GS}) = x^2_{GS}.$$

Continuing we get

$$y^4_J = \psi(x^3_J) = \psi(x^2_{GS}) = y^2_{GS}$$

and

$$x^5_J = \varphi(y^4_J) = \varphi(y^2_{GS}) = x^3_{GS}.$$

From these equations we observe that

$$\begin{aligned} x^K_{GS} &= x^{2K-1}_J \\ y^K_{GS} &= y^{2K}_J \end{aligned} \tag{3.4}$$

for iteration K. This shows us that Jacobi takes twice the amount of iterations as Gauß-Seidel, up to a single iteration. This relation only holds for systems that are partitioned in to two subsystems. For three subsystems this does not hold anymore. Also if the problem is solved using multiple macrosteps this doesn't since the solutions produced by the methods are slightly different, giving subsequent macrosteps slightly different initial conditions.

3.1.4 Creating a function from discrete data points

The different methods above are rules for iterating over functions. However the numerical integration of an ODE does not give us a function, rather it returns discrete data points. We denote these point sets by $\{x_i\}_1^n$ and $\{y_i\}_1^m$ respectively. To transform these point sets to functions we resort to interpolation, for example polynomial spline interpolation where we can also determine the number of continuous derivatives. The functions x^* and y^* will then be defined as

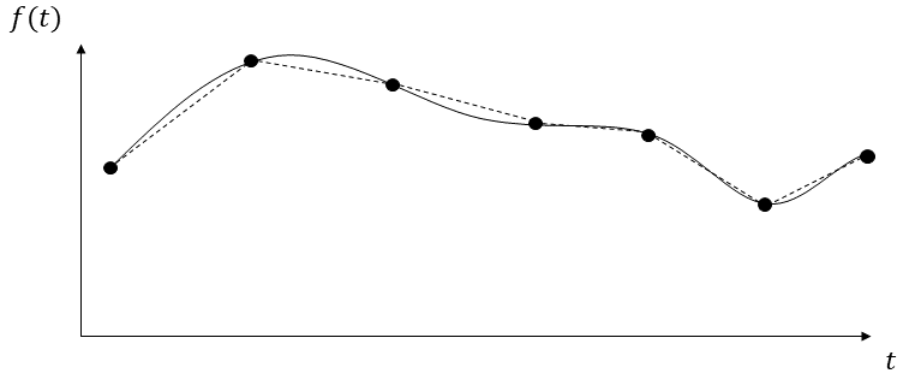

Figure 3.3: Figure sketching the idea of interpolation. We have discrete data points (the black dots) given from a numerical solution. Using different interpolations we can convert this discrete solution to a function in continuous time. The dotted line represents a linear interpolation and the continuous line a higher order interpolation.

$$\begin{aligned} x^* &= interp(\{x_i\}_1^n) \\ y^* &= interp(\{y_i\}_1^m) \end{aligned} \tag{3.5}$$

where *interp* is some interpolation procedure. A schematic example of how interpolation works can be seen in figure 3.3.

3.1.5 Singlerate vs multirate methods

Looking at the partitioning in equation (3.1) we see that, given a x^* and y^*, both systems can be solved independently from each other. This means that the solvers used for the different systems don't have to use the same time step length. When this property is utilized the solution method is called a multirate method. Here multirate refers to the fact that both subsystems are solved with different time steps, or in other words at different rates. If the different systems are solved with the same time step, the method is called a single rate method. Figure 3.4 shows schematically how the parallel time integration for a multirate system may look like.

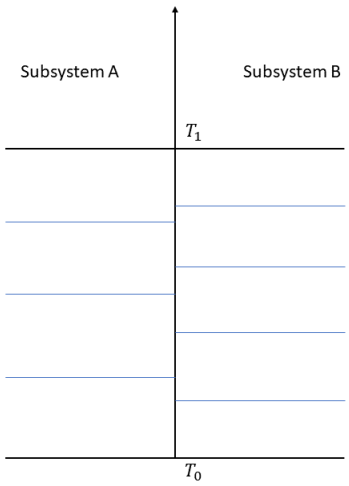

Figure 3.4: Diagram showing the workings of one macrostep. The subsystems have independant microsteps.

3.2 Convergence analysis for linear systems

To get an understanding of the behaviour of these methods we will analyse the convergence properties of the two methods defined above when applied to a 2-dimensional linear system. The analysis will also be limited to real valued functions on finite-time intervals. First we will analyse continuous time waveform iterations, meaning the iterates are functions defined on a continuous time domain. Thereafter we will analyse the discrete case where the iterates are the result of numerical time integration.

3.2.1 Convergence of iterative schemes

In the analysis below we will look at the behaviour of different waveform iteration methods. To understand how those iterative schemes work, we look at the more general case first. This also requires a small summary of topics from functional analysis. More background on this summary can be found in [2] or in any introductory book on functional analysis.

We define the norm of a bounded linear operator $\mathcal{A}$ in a normed space $\mathcal{X}$ in the following way:

$$\|\mathcal{A}\|_{\mathcal{X}} = \sup_{\|x\|_{\mathcal{X}}=1} \|\mathcal{A}x\|_{\mathcal{X}} . \tag{3.6}$$

We also define the spectrum of this operator $\mathcal{A}$, $\sigma(\mathcal{A})$ as the set of scalars λ for which the operator $(\lambda - \mathcal{A})^{-1}$ is not a bounded operator defined on a dense subset of $\mathcal{X}$. This leads us to the following definition of the spectral radius, valid when $\sigma(\mathcal{A})$ is non-empty and bounded:

$$\rho(\mathcal{A}) = \sup_{\lambda \in \sigma(\mathcal{A})} |\lambda|. \tag{3.7}$$

If $\mathcal{A}$ is a matrix, the spectral radius corresponds to the largest absolute value of the eigenvalues of $\mathcal{A}$. The spectral radius has an important property, namely

$$\rho(\mathcal{A}) = \lim_{n \to \infty} \sqrt[n]{\|\mathcal{A}^n\|_{\mathcal{X}}}. \tag{3.8}$$

An interpretation of this equation is that the sequence $x^i \in \mathcal{X}$ where $x^{i+1} = \mathcal{A}x^i$ has the asymptotic relationship

$$\left\|x^{i+1}\right\|_{\mathcal{X}} = \rho(\mathcal{A}) \left\|x^i\right\|_{\mathcal{X}} \tag{3.9}$$

between successive iterates.

This leads us to the final result, namely that the iteration

$$x^i = \mathcal{A}x^{i+1} + \varphi \tag{3.10}$$

is convergent if $\rho(\mathcal{A}) < 1$.

3.2.2 Linear systems and splittings

To be able to compare these results and derivations to existing texts we will in this chapter consider ordinary differential equations of the following form, consistent with the analysis done by Vandewalle in [2] and Nevanlinna in [3]:

$$\dot{u} + Au = f, \text{ where } u(0) = u_0, t \in [0, T] \tag{3.11}$$

where $A \in \mathbb{R}^{2\times 2}$ and $u \in \mathbb{R}^2$, $f \in \mathbb{R}^2$ are functions of time. For these systems there exists the direct formula

$$u(t) = e^{-tA}u_0 + \int_0^t e^{(s-t)A} f(s) ds \tag{3.12}$$

for the solution.

To arrive at an iterative scheme we use a splitting technique. By applying a splitting to the system with $A = P - Q$ we can rewrite the system to

$$\dot{u} + Pu = Qu + f, \text{ where } u(0) = u_0, t \in [0, T] \tag{3.13}$$

and introduce the iterative scheme

$$\dot{u^K} + Pu^K = Qu^{K-1} + f, \text{ where } u^K(0) = u_0, t \in [0, T] \tag{3.14}$$

given an initial starting point for the iteration: $u^0 = u_0 \quad \forall t \in [0, T]$.

Using this we can write the Jacobi and Gauß-Seidel iteration in terms of P, Q and the decomposition of $A = D - L - U$ into its strictly upper/lower triangular parts $-U, -L$ and its diagonal D:

$$\begin{array}{lll} \texttt{Jacobi:} & P = D & , \quad Q = L + U \\ \texttt{Gauß-Seidel:} & P = -L + D & , \quad Q = U \end{array} \tag{3.15}$$

3.2.3 Convergence for continuous time functions

We would like to rewrite equation (3.14) in a form with a single operator $\mathcal{K}$ operating on u^{K-1} and a rest term φ similar to (3.10):

$$u^K = \mathcal{K}u^{K-1} + \varphi. \tag{3.16}$$

This can be done by applying the solution formula (3.12) to the splitting in (3.14),

$$u^K(t) = e^{-tP}u_0 + \int_0^t e^{(s-t)P}Qu^{K-1}(s)ds + \int_0^t e^{(s-t)P}f(s)ds. \tag{3.17}$$

Then

$$\begin{array}{rcl} \mathcal{K}u & = & \int_0^t e^{(s-t)P}Qu(s)ds \\ \varphi & = & e^{-tP}u_0 + \int_0^t e^{(s-t)P}f(s)ds \end{array} \tag{3.18}$$

we can view this operator $\mathcal{K}$ as a convolution operator with a kernel k:

$$\mathcal{K}u(t) = k \star u(t) = \int_0^t k(t-s)u(s)ds \text{ with } k(t) = e^{-tP}Q. \tag{3.19}$$

Now the iterations can be thought of as successive applications of this convolution. Hence we can define the kernel at iteration K, k^{K*} with the following recursive definition:

$$k^{K*} = k \star k^{K-1*}, \text{ where } k^{1*} = k. \tag{3.20}$$

We also define $\mathcal{K}^K$ as K successive applications of the operator $\mathcal{K}$. To analyse the convergence rate we need to define a norm to measure in. We will use the max norm $\|.\|_T$ defined in the following way:

$$\|u\|_T = \max_{[0,T]} \|u(t)\|, \tag{3.21}$$

where $\|.\|$ denotes a suitable vector norm. We define the upper bound of k as

$$\|k\|_T = C. \tag{3.22}$$

Using this bound and the recursive definition for k^{K*} we can bound $k^{K*}(t)$ in the following way:

$$\left\|k^{K*}(t)\right\| = \left\|k(t) \star k^{(K-1)\star}(t)\right\| \leq C \int_0^t \left\|k^{(K-1)*}(s)\right\| ds \tag{3.23}$$

using that

$$\begin{array}{l} \left\|\int_0^t k(t-s)k^{(K-1)\star}(s)ds\right\| \leq \int_0^t \left\|k(t-s)k^{(K-1)\star}(s)\right\| ds \\ \leq \int_0^t C \left\|k^{(K-1)\star}(s)\right\| ds. \end{array}$$

This recursive definition results in a successive evaluation of integrals defined in (3.23):

$$\left\|k^{2\star}(t)\right\| \leq C\int_0^t C ds = C(Ct)$$

$$\left\|k^{3\star}(t)\right\| \leq C\int_0^t C(Cs) ds = C\frac{(Ct)^2}{2}$$

$$\left\|k^{4\star}(t)\right\| \leq C\int_0^t C\frac{(Cs)^2}{2} ds = C\frac{(Ct)^3}{6}$$

giving us the following estimate for the Kth kernel $k^{K*}(t)$:

$$\left\|k^{K*}(t)\right\| \leq C\frac{(Ct)^{K-1}}{(K-1)!}. \tag{3.24}$$

Now that we have an upper bound on the kernel, we need to translate this bound to an upper bound for the operator $\mathcal{K}$. This can be done using the estimate, given by Nevanlinna in [3],

$$\left\|\mathcal{K}^K\right\|_T \leq \int_0^T \left\|k^{K*}(t)\right\| dt \tag{3.25}$$

which after integrating becomes:

$$||\mathcal{K}^K||_T \leq \frac{(CT)^K}{K!}. \tag{3.26}$$

To get to the error estimate we again look at the iteration scheme in equation (3.16). If the spectral radius of $\mathcal{K}$, $\rho(\mathcal{K}) < 1$ this iteration converges [2].

A solution $\tilde{u}$ satisfies the following equation:

$$\tilde{u} = \mathcal{K}\tilde{u} + \varphi. \tag{3.27}$$

We have established in equation (3.26) that the operator $\mathcal{K}$ is bounded superlinearly meaning the spectral radius $\rho(\mathcal{K}) = 0$ in the max norm [2]. We want to get an explicit formula for the iterate u^K. By seeing what happens in successive applications of the iterative scheme in equation (3.16) we can find a general expression:

$$\begin{aligned} u^1 &= \mathcal{K}u^0 + \varphi \\ u^2 &= \mathcal{K}(\mathcal{K}u^0 + \varphi) + \varphi \\ u^3 &= \mathcal{K}(\mathcal{K}(\mathcal{K}u^0 + \varphi) + \varphi) + \varphi. \end{aligned} \tag{3.28}$$

We see the following relation for u^K:

$$u^K = \mathcal{K}^K u^0 + \sum_{i=0}^{K-1} \mathcal{K}^i \varphi. \tag{3.29}$$

Finally we arrive at the error estimate for the iterates:

$$||\tilde{u} - u^K||_T \leq \frac{(CT)^K}{K!}||\tilde{u} - u^0||_T. \tag{3.30}$$

Here we have a formula showing superlinear convergence for waveform iteration schemes as defined above. It is superlinear because of the limit

$$\lim_{K\to\infty} \frac{(CT)^K}{K!} \to 0.$$

It is evident that both the macrostep length T and the constant C coming from the splitting are greatly impacting the speed of convergence.

To get a value for C we analyse the T-norm of the kernel k. For simplicity we assume $T = 1$. For Gauß-Seidel we have

$$||k_{GS}||_T = \left\| e^{-t(D-L)}Q \right\|_T = \max_{t\in[0,1]} \left\| e^{-t(D-L)}U \right\| \tag{3.31}$$

and similarly for Jacobi:

$$||k_J||_T = \left\| e^{-t(D)}Q \right\|_T = \max_{t\in[0,1]} \left\| e^{-t(D)}(L+U) \right\|. \tag{3.32}$$

3.2.4 Singlerate convergence for discrete time functions

For the discrete case we first need some notation. We denote the set of the discrete points of the solution as $U = [u_1, u_2, ..., u_N]^T$. Here u_i denotes the solution at time step t_i. Our iterates are denoted as U^K for iterate K and u_i^K for the value of the iterate at the particular time t_i.

To simplify the analysis we use the implicit Euler method for time integration. A more general analysis for other multistep methods can be found in Vandewalle [2]. Using implicit Euler with step size τ we can discretise equation (3.14) in the following way:

$$\frac{1}{\tau}(u_{n+1}^K - u_n^K) + Pu_{n+1}^K = Qu_{n+1}^{K-1} + f(u_{n+1}). \tag{3.33}$$

To arrive to a similar error estimate as in equation (3.30), we rewrite equation (3.33) in terms of the iteration errors $e_n^K = u_n^K - u_n$,

$$\frac{1}{\tau}(e_{n+1}^K - e_n^K) + Pe_{n+1}^K = Qe_{n+1}^{K-1} \tag{3.34}$$

using the fact that the iteration doesn't change the exact solution, similar to equation (3.27). We rewrite this equation into

$$C_0 e_n^K + C_1 e_{n+1}^K = D_1 e_{n+1}^{K-1} \tag{3.35}$$

with $C_0 = -\frac{1}{\tau}I$, $C_1 = (\frac{1}{\tau}I + P)$ and $D_1 = Q$. Now forming the full solution $E^K = [e_1^K, e_2^K, ...e_N^K]$ we arrive at the waveform iteration scheme

$$CE^K = DE^{K-1} \Leftrightarrow E^K = C^{-1}DE^{K-1} \tag{3.36}$$

with $C, D \in \mathbb{R}^{(2N)\times(2N)}$ as block matrices defined as

$$C = \begin{bmatrix} C_1 & & & & \\ C_0 & C_1 & & & \\ & C_0 & C_1 & & \\ & & \ddots & \ddots & \\ & & & C_0 & C_1 \end{bmatrix}, \quad D = \begin{bmatrix} D_1 & & & \\ & D_1 & & \\ & & \ddots & \\ & & & D_1 \end{bmatrix}. \tag{3.37}$$

Recognizing the special block lower triangular structure, the spectral radius of $C^{-1}D$ can be expressed in P and Q:

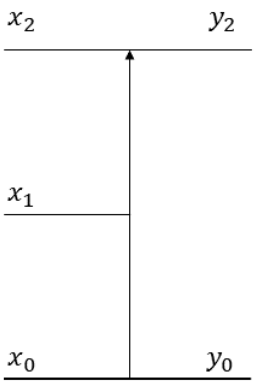

Figure 3.5: Figure showing the simple multirate case we are going to study. The central axis denotes advancing in time. For every step in the y system two steps are taking in the x system. We have no direct value of y_1 however we can assign a value using interpolation.

$$\rho(C^{-1}D) = \rho(C_1^{-1}D^1) = \rho((\frac{1}{\tau}I + P)^{-1}Q). \tag{3.38}$$

This means that the asymptotic convergence rate is not determined by the macrostep length T or the number of microsteps. It is the microstep length τ that plays a key role.

3.2.5 Multirate convergence for discrete time functions

To understand the multirate case we first analyse a simple setup, as seen in figure 3.5, where the two time steps are related by $\tau_y = 2\tau_x$. We define y_i and x_i to denote the state of both systems at time t_i. Also since every other point in system y, $y_{2n+1}, n \in \mathbb{N}$, is not given by the ODE integration method we define those points using interpolation. In this instance we use linear interpolation which, using a fix time step τ_x has the following form:

$$y_{2n+1} := \frac{1}{2}(y_{2n} + y_{2(n+1)}). \tag{3.39}$$

We also need some more notation. We split the matrices P and Q as

$$P = \begin{bmatrix} P_{xx} & P_{xy} \\ P_{yx} & P_{yy} \end{bmatrix} \tag{3.40}$$

and

$$Q = \begin{bmatrix} Q_{xx} & Q_{xy} \\ Q_{yx} & Q_{yy} \end{bmatrix}. \tag{3.41}$$

Denote the iteration errors of the subsystems as $\xi_i = x_i - x_i^K$ and $\eta_i = y_i - y_i^K$. Using this we can write down the equations for the two unknowns $\xi_{2(n+1)}$ and $\eta_{2(n+1)}$, similar to equation (3.34):

$$\begin{aligned} &\tfrac{1}{\tau_x}(\xi^K_{2(n+1)} - \xi^K_{2n+1}) + P_{xx}\xi^K_{2(n+1)} + P_{xy}\eta^K_{2(n+1)} \\ &= Q_{xx}\xi^{K-1}_{2(n+1)} + Q_{xy}\eta^{K-1}_{2(n+1)} \end{aligned} \tag{3.42}$$

and

$$\begin{array}{l} \frac{1}{\tau_y}(\eta^K_{2(n+1)} - \eta^K_{2n}) + P_{yy}\eta^K_{2(n+1)} + P_{yx}\xi^K_{2(n+1)} \\ = Q_{yy}\eta^{K-1}_{2(n+1)} + Q_{yx}\xi^{K-1}_{2(n+1)} \end{array} . \tag{3.43}$$

For step ξ^K_{2n+1} we have

$$\frac{1}{\tau_x}(\xi^K_{2n+1} - \xi^K_{2n}) + P_{xx}\xi^K_{2n+1} + P_{xy}\eta^K_{2n+1} = Q_{xx}\xi^{K-1}_{2n+1} + Q_{xy}\eta^{K-1}_{2n+1} \tag{3.44}$$

where we notice that the error η^K_{2n+1} doesn't exist because of the different rates of the subsystems. Therefore we resort to linear interpolation:

$$\eta^K_{2n+1} = \frac{1}{2}(\eta^K_{2n} + \eta^K_{2n+2}).$$

changing equation (3.44) into

$$\begin{array}{c} \frac{1}{\tau_x}(\xi^K_{2n+1} - \xi^K_{2n}) + P_{xx}\xi^K_{2n+1} + \frac{1}{2}P_{xy}(\eta^K_{2n} + \eta^K_{2(n+1)}) = \\ Q_{xx}\xi^{K-1}_{2n+1} + \frac{1}{2}Q_{xy}(\eta^{K-1}_{2n} + \eta^{K-1}_{2(n+1)}). \end{array} \tag{3.45}$$

Writing the error $e^K_{2(n+1)} = [\xi^K_{2n+1}, \xi^K_{2(n+1)}, \eta^K_{2(n+1)}]$ we can combine equations (3.42), (3.43) and (3.44) and write the iteration process for the error as

$$C_0 e^K_n + C_1 e^K_{n+1} = D_0 e^{K-1}_n + D_1 e^{K-1}_{n+1} \tag{3.46}$$

with

$$\begin{array}{l} C_0 = \begin{bmatrix} 0 & -\frac{1}{\tau_x} & \frac{1}{2}P_{xy} \\ 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{\tau_y} \end{bmatrix}, \quad C_1 = \begin{bmatrix} \frac{1}{\tau_x} + P_{xx} & 0 & \frac{1}{2}P_{xy} \\ -\frac{1}{\tau_x} & \frac{1}{\tau_x} + P_{xx} & P_{xy} \\ 0 & P_{yx} & \frac{1}{\tau_y} + P_{yy} \end{bmatrix} . \\ D_0 = \begin{bmatrix} 0 & 0 & \frac{1}{2}Q_{xy} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad D_1 = \begin{bmatrix} Q_{xx} & 0 & \frac{1}{2}Q_{xy} \\ 0 & Q_{xx} & Q_{xy} \\ 0 & Q_{yx} & Q_{yy} \end{bmatrix} \end{array} \tag{3.47}$$

We can again write this in terms of a matrix C, D and $E^K = [e^K_2, e^K_4, ..., e^K_{2N}]^T$,

$$CE^K = DE^{K-1}$$

with

$$C = \begin{bmatrix} C_1 & & & & \\ C_0 & C_1 & & & \\ & C_0 & C_1 & & \\ & & \ddots & \ddots & \\ & & & C_0 & C_1 \end{bmatrix}, \quad D = \begin{bmatrix} D_1 & & & & \\ D_0 & D_1 & & & \\ & D_0 & D_1 & & \\ & & \ddots & \ddots & \\ & & & D_0 & D_1 \end{bmatrix} \tag{3.48}$$

where $C, D \in \mathbb{R}^{(3N \times 3N)}$ and the spectral radius ρ again follows the relation

$$\rho(C^{-1}D) = \rho(C_1^{-1}D_1). \tag{3.49}$$

We can expand this analysis to fit any integer (q) amount of steps for system A (x) in a single step of system B (y), $q\tau_x = \tau_y$. We again write the iteration

in terms of the error according to equation (3.46) but now with $e^K_{q(n+1)} = [\xi^K_{qn+1}, \xi^K_{qn+2}, ..., \xi^K_{qn+q}, \eta^K_{qn+q}]$. For all points ξ^K_{qn+1} to ξ^K_{qn+q-1} we again need to use interpolation since the corresponding errors η^K_{qn+1} to η^K_{qn+q-1} do not exist. To show where interpolation is introduced we write down intermediate step p: ξ^K_{qn+p}, where we take an implicit euler step from x_{qn+p-1} to x_{qn+p}:

$$\begin{array}{l} \frac{1}{\tau_x}(\xi^K_{qn+p} - \xi^K_{qn+p-1}) + P_{xx}\xi^K_{qn+p} + P_{xy} interp(p, \eta^K_{qn}, \eta^K_{qn+q}) = \\ Q_{xx}\xi^{K-1}_{qn+p} + Q_{xy} interp(p, \eta^{K-1}_{qn}, \eta^{K-1}_{qn+q}) \end{array} \tag{3.50}$$

where

$$interp(p, \eta^{K-1}_{qn}, \eta^{K-1}_{qn+q}) = (1 - \frac{p}{q})(\eta^{K-1}_{qn}) + \frac{p}{q}(\eta^{K-1}_{qn+q})$$

is the linear interpolation of η^{K-1}_{qn} and η^{K-1}_{qn+q} at point $qn + p$. Substituting in this interpolation formula in equation (3.50) we get

$$\begin{array}{l} \frac{1}{\tau_x}(\xi^K_{qn+p} - \xi^K_{qn+p-1}) + P_{xx}\xi^K_{qn+p} + P_{xy}((1 - \frac{p}{q})\eta^K_{qn} + \frac{p}{q}\eta^K_{qn+q}) = \\ Q_{xx}\xi^{K-1}_{qn+p} + Q_{xy}((1 - \frac{p}{q})\eta^{K-1}_{qn} + \frac{p}{q}\eta^{K-1}_{qn+q}) \end{array} . \tag{3.51}$$

For η^K_{qn+q} we have a similar equation to (3.43):

$$\begin{array}{l} \frac{1}{\tau_y}(\eta^K_{qn+q} - \eta^K_{qn}) + P_{yy}\eta^K_{qn+q} + P_{yx}\xi^K_{qn+q} \\ = Q_{yy}\eta^{K-1}_{qn+q} + Q_{yx}\xi^{K-1}_{qn+q} \end{array} . \tag{3.52}$$

Now we can combine equations (3.51) and (3.52) into the same form as equation (3.46), now with

$$
\begin{aligned}
C_0 &= \begin{bmatrix} 0 & \cdots & 0 & -\frac{1}{\tau_x} & (1-\frac{1}{q})P_{xy} \\ 0 & \cdots & 0 & 0 & (1-\frac{2}{q})P_{xy} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & 0 & 0 \\ 0 & \cdots & 0 & 0 & -\frac{1}{\tau_y} \end{bmatrix} \\
C_1 &= \begin{bmatrix} \frac{1}{\tau_x}+P_{xx} & 0 & \cdots & 0 & \frac{1}{q}P_{xy} \\ -\frac{1}{\tau_x} & \frac{1}{\tau_x}+P_{xx} & 0 & 0 & \frac{2}{q}P_{xy} \\ 0 & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & -\frac{1}{\tau_x} & \frac{1}{\tau_x}+P_{xx} & P_{xy} \\ 0 & \cdots & 0 & P_{yx} & \frac{1}{\tau_y}+P_{yy} \end{bmatrix} \\
D_0 &= \begin{bmatrix} 0 & \cdots & 0 & (1-\frac{1}{q})Q_{xy} \\ \vdots & \ddots & & (1-\frac{2}{q})Q_{xy} \\ & & & \vdots \\ 0 & \cdots & 0 & 0 \\ 0 & \cdots & 0 & 0 \end{bmatrix} \\
D_1 &= \begin{bmatrix} Q_{xx} & 0 & \cdots & 0 & \frac{1}{q}Q_{xy} \\ 0 & \ddots & \ddots & & \frac{2}{q}Q_{xy} \\ \vdots & & & & \vdots \\ 0 & \cdots & 0 & Q_{xx} & Q_{xy} \\ 0 & \cdots & 0 & Q_{yx} & Q_{yy} \end{bmatrix}.
\end{aligned}
\tag{3.53}
$$

Here we can clearly see the structure in the matrices where the q first rows correspond to time stepping in system x. This is seen in the diagonal structure of C_1 as well as in the linear interpolation of system y in the right-most columns of all four matrices.

Again we combine the errors in a large error vector $E^K = [e_q^K, e_{2q}^K, ..., e_{qN}^K]^T$ and write the iteration as

$$CE^K = DE^{K-1}$$

with C and D defined in the same way as equation (3.48). We then also arrive at the same equation for the spectral radius:

$$\rho(C^{-1}D) = \rho(C_1^{-1}D_1).$$

3.2.6 Comments on the convergence analysis

There are different uses to the results of the continuous and the discrete analysis. The continuous analysis gives us a bound for the iteration error of a particular iterate. This is however only valid for the analytical solution. Still the continuous analysis gives us some insight in how the method performs during the first couple of iterates.

The discrete analysis does not tell us anything about a particular iteration. Rather it tells us the asymptotic convergence rate of the method when the equation is solved using implicit Euler.

From the discrete analysis we can also draw the conclusion that the number of time steps does not affect the asymptotic convergence rate. It is only the time step length τ that appears in the equation (3.38).

3.3 Stopping criteria

The sequence obtained by the iteration schemes is an infinite sequence. In general the limit will not be reached in finite time. In numerics we want to stop the iteration when the solution satisfies some error condition. We define the iteration error as

$$e^K = \left\| u - u^K \right\| \tag{3.54}$$

which measures the difference between the Kth iterate and the analytical solution. Since we generally don't have access to the real solution, we need a way of estimating the iteration error based on data we have access to. One way to do this is to use the absolute difference between iterations, defined as

$$\Delta e^K = \left\| u^K - u^{K+1} \right\| \tag{3.55}$$

which gives us an estimate of how much the solution changes each iteration. According to Nevanlinna [3] this is a good error estimate in the sense that

$$\left\| u - u^K \right\|_T \approx \left\| u^{K+1} - u^K \right\|_T . \tag{3.56}$$

Since our numerical schemes return points on a grid instead of continuous functions, we want to have an estimate based on these discrete numerical results. We know that in general $\|u_t - u(t)\|$ increases as $t > t_0$ increases, as described by [8]

$$\|u_t - u(t)\| \leq \frac{\max(\delta_i^h)}{L}(e^{L(t-t_0)} - 1) \tag{3.57}$$

where L is a positive constant depending on the numerical integration method and δ_i^h are the local errors defined in equation (2.2). Since the bound on the error increases with time, it makes sense to estimate the iteration error by comparing the iterates at the last grid point t_e:

$$e^K \approx \left\| u_e^K - u_e^{K+1} \right\| . \tag{3.58}$$

It is this estimate that will be used in the implementation of the methods.

3.3.1 Weighted Scaled Norm

The solution of the full system $u = [x, y]^T$ consists of both solutions x and y. To measure the error given by equation (3.55) we need to relate the errors in the subsystems to the error of the full system. This can be done by a weighted scaled norm.

Given that $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$ we can write down the weigted scaled norm based on the two-norm of the subsystems:

$$||u||_{wsn} = ||[x,y]^T||_{wsn} = \sqrt{\frac{1}{n+m}\Sigma\alpha_i u_i^2} \text{ where } \alpha_i = \begin{cases} n, i \leq n \\ m, i > n \end{cases} . \quad (3.59)$$

This leads to

$$||u||_{wsn} = \sqrt{\frac{n}{n+m}||x||_2^2 + \frac{m}{n+m}||y||_2^2} \quad (3.60)$$

expressing the norm in terms of the norms of the subsystems. Here we can see that if $n > m$ meaning system A is larger it is weighted more heavily than system B.

The weighted scaled norm defined above stems from using the two-norm. If the max norm is used instead we resort to using the max norm:

$$\|u\|_{wsn} = \left\|[x,y]^T\right\|_\infty . \quad (3.61)$$

3.4 Waveform relaxation

Thus far after each iteration we throw away old data and continue with the newly obtained iterate. There are ways to accelerate the convergence using relaxation. This means that we instead of only using the new iterate, create a weighted sum of the new iterate and the previous iterate:

$$u^{K+1} = \omega\tilde{u}^{K+1} + (1-\omega)u^K. \quad (3.62)$$

Here ω is the relaxation constant which is used to create the new iterate u^{K+1} as a convex combination of the old iterate u^K and the newly obtained unrelaxed iterate $\tilde{u}^{K+1}$.

Relaxation is a whole topic itself and analysis of finding good values of ω is difficult. For that reason we will stick to waveform iteration in this thesis.

3.5 The first iteration

The first iteration of a macrostep doesn't have any information available from previous iterations. The only piece of information that is known is the initial state of both systems, obtained from either initial conditions $u(t_0)$ (if this is the first macrostep) or from the previous macrostep. To start the iteration we need a way to define $y^{K=0}$. The simplest method only uses the state itself without gradient information. Using extrapolation we have an initial guess

$$\begin{aligned} x^{K=0}(t) &= x_0 \\ y^{K=0}(t) &= y_0 \end{aligned} \quad (3.63)$$

used as first iterate for the whole macrostep domain.

An alternative here is to include more information from the previous macrostep. One could include information about the gradient or even higher order derivatives. This could improve the performance of the method since the initial iterate $y^{K=0}$ will approximate more derivatives of the solution. This will however not be discussed in this thesis.

Chapter 4

The new multirate time integration method

Building on the principles of the discussed waveform iteration methods, we can see that both methods exchange information at predetermined places in time. This makes implementation easier but also severely limits flexibility. Ideally we would like both systems to use the most recent available information from each other. We describe this behaviour in the following equation:

$$\begin{aligned} \dot{x}^{K+1} &= f(x^{K+1}, y^*(y^K, y^{K+1})) \\ \dot{y}^{K+1} &= g(x^*(x^K, x^{K+1}), y^{K+1}) \end{aligned} \quad . \tag{4.1}$$

Here $x^*(x^K, x^{K+1})$ and $y^*(y^K, y^{K+1})$ denote representations of the subsystems that gradually build up during the parallel integration of both subsystems. This means that as each subsystem is being integrated, information is sent from one system to another whenever one system has advanced its solution in time. Figure 4.1 shows how this multirate integration works.

An important side effect of sending and retrieving information whilst integrating is that, contrary to Jacobi and Gauß-Seidel, the method is no longer deterministic. Underlying processes or other effects from the underlying operating system may result in a system finishing before the other for one run of a simulation, but finishing after the other when redoing the exact same simulation.

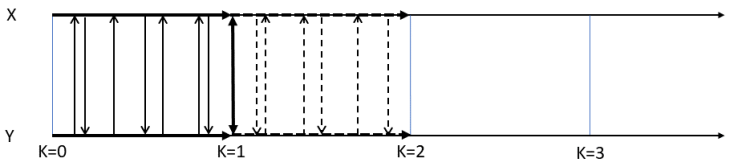

Figure 4.1: Diagram of the multirate integration scheme. Contrary to the Gauß-Seidel and Jacobi approach, information is exchanged whilst integrating the subsystems, not only after a full iteration.

4.1 Algorithm

The main workings of the method are condensed in algorithms 1 and 2. Algorithm 1 describes the workings of the macrostep iteration.

Algorithm 1 Multirate time integration, integrate macrostep

```
1: procedure INTEGRATE MACROSTEP
2:     for K ← 1, 2, 3... do                          ▷ Iteration loop
3:         Integrate subsystems in parallel           ▷ acc. to alrogithm 2
4:         Check stopping criterion                   ▷ If met, stop iteration
5:     end for
6: end procedure
```

Algorithm 2 describes the integration of one of the subsystems.

Algorithm 2 Multirate time integration, loop of system A

```
1: procedure SOLVE SUBSYSTEM x                        ▷ Analogous for y
2:     while not finished time integration do
3:         Update point set {y_i}_1^m.                ▷ Data read in from system B
4:         Create function y* = interp({y_i}_1^m) from point set.
5:         x_{n+1} ← ODEStep(x_n, y*)                 ▷ Take next microstep
6:         Send x_{n+1} to y system.
7:     end while
8: end procedure
```

4.2 Choosing function representations

The main difficulty with this method revolves around choosing a good way of representing the other system. If, for the current iteration and timestamp, system A has data about system B then the state of system B will be known by interpolation. However if A has advanced further in time than B this data is not available, thus one needs another way to represent the system. There are many different ways to do this. We could opt to only use simulation data from the current iteration and get the system's state using extrapolation. Although easy to implement, this can lead to great aproximation errors, meaning that in some cases the iteration might not converge. Instead we could also include data from previous iterations. Here we have several options.

4.2.1 The general case

In the general case we want to be able to use both information from the previous and the current iteration. We can write it down in the following form:

$$y^*(y^K, y^{K+1}) = \gamma(t_x, t_y)y^K + (1 - \gamma(t_x, t_y))y^{K+1}. \tag{4.2}$$

Here the function representation y^* of system B is generated as a combination between the new and old iterates using the function $\gamma(t_y, t_x)$ which is a function of t_x and y_y, which denote how far in time both subsystems have advanced their time integration.

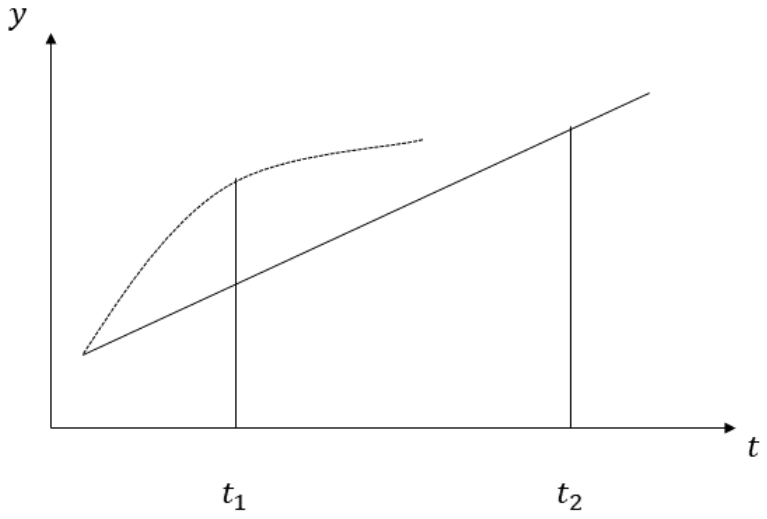

Figure 4.2: Figure schematically showing the mixed Jacobi and Gauß-Seidel method. The solid line shows the iterate K of system B. The dashed line shows the progress of iterate $K+1$. At time t_1 system B has information from the newest iterate available, therefore we choose $y^*(t_1) = y^{K+1}(t_1)$. At t_2 however we don't have this information available and resort to $y^*(t_2) = y^K(t_2)$.

4.2.2 Mixed Jacobi and Gauß-Seidel

One option is to always choose to use the most recent system state available. Lets assume that system A is ahead of system B, meaning that $t_x > t_y$. This means that system B can use information from the most recent iteration of system A. However, since system A has no information from system B at time t_x, the most recent information available at that time comes from the previous iteration. This leads to a kind of compromise between the Gauß-Seidel and the Jacobi approach. A schematic drawing of this process can be seen in figure 4.2. Using the notation from equation (4.2) we can opt for the following γ:

$$\gamma(t_x, t_y) = \begin{cases} 1 & \texttt{if} \quad t_y < t_x \\ 0 & \texttt{if} \quad t_y > t_x \end{cases} . \tag{4.3}$$

This can be formulated in terms of y^K and y^{K+1}:

$$y^* = \begin{cases} y^{K+1} & \texttt{if} \quad t_y < t_x \\ y^K & \texttt{if} \quad t_y > t_x \end{cases} . \tag{4.4}$$

All the results in this thesis are based on methods using this mixed Jacobi and Gauß-Seidel approach.

4.2.3 Other options

There are other options as well. We could use extrapolation for y^* when $t_y > t_x$. Here we could for instance use a weight w between the extrapolated state $\tilde{y}^{K+1}$

and y^K:

$$y^* = \begin{cases} y^{K+1} & \texttt{if} \quad t_y < t_x \\ wy^K + (1-w)\tilde{y}^{K+1} & \texttt{if} \quad t_y > t_x \end{cases}$$

where $\tilde{y}^K(t)$ is the extrapolation based on y^K. Instead of a constant one can also use a function $\omega(t_x, t_y)$ for weighting between the extrapolated state and the previous iterate:

$$y^* = \begin{cases} y^{K+1} & \texttt{if} \quad t_y < t_x \\ \omega(t_x,t_y)y^K + (1-\omega(t_x,t_y))\tilde{y}^{K+1} & \texttt{if} \quad t_y > t_x \end{cases}.$$

These options will not be explored more in this thesis, but are worth exploring in future work.

Chapter 5

Results

The results presented in this section are all produced using an implementation in Python version 3.6 using the SciPy and NumPy packages. With regard to parallelisation there are many libraries to choose from. In this implementation the built in package `multiprocessing` is used. The actual sending of the message is handled by the process itself in the background. This is important as this negatively affects the performance. In this thesis we do not look further into this except that it is worth noting that there might be gains in performance available using more efficient means of communication such as shared memory.

For all parallelised algorithms in this thesis two processes are allocated, one for subsystem A and one for subsystem B. These processes run for the full length of the simulation.

All simulations were run on computers with at least four core cpu's. This was done to minimize the impact other processes run by the operating system might have on the performance of the methods. Also whenever results are compared, they were done on the same machine.

Measuring computational time

When a program uses multiple cores or processors there are different ways of measuring the time taken to execute a task. We can measure the time in elapsed time for the user, called wall-clock time, which is the time from the start of execution until the task is complete. However when using multiple processes this doesn't reflect the amount of work done by the processor. It is also interesting to measure the total cpu time, meaning the sum of time consumed by all processes. For instance a program fully utilising two cores that takes one second to finish in wall clock time uses two seconds of total cpu time. All results involving computational time are presented using wall-clock time.

5.1 Verification of the methods

To verify the order of the methods we resort to solving a system of which we know the analytical solution. Comparing the numerical approximation to the real solution we can see how the order if affected by different parameters.

We use the following 2D system:

$$\begin{cases} \dot{x} = -x + y/2 & x(0) = 1 \\ \dot{y} = -x/2 - y & y(0) = 3 \end{cases} . \tag{5.1}$$

The solution to this system is given by the general solution for linear dynamical systems using the matrix exponential from equation (3.12).

5.1.1 Varying microstep length

The results in figures 5.1, 5.2 and 5.3 have been produced using the following solver parameters:

- Number of macrosteps: 1
- Macrostep length: 1
- Microstep length for system A: $\tau_x = \tau$, B: $\tau_y = 2\tau$, where τ is the base step length shown in the plot.
- Subsystem interpolation: Linear
- Macrostep iteration stopping tolerance: 10^{-14}, in 2 norm
- Subsystem solver: SDIRK2 (see appendix A.0.2)
- Representation of x^*,y^*: Mixed Jacobi, Gauß-Seidel, equation (4.4)

Order of convergence

To check the order the length of the microsteps in both the subsystems were varied. Then the obtained numerical approximation is compared to the analytical solution. The norm used to measure the error is the 2-norm. As the second order SDIRK2 method is used, it makes sense that the three waveform iteration methods have an order of accuracy of two. This can be seen in figure 5.1.

Number of iterations

As previously discussed, mixed Jacobi and Gauß-Seidel is a compromise between the two methods. If one of the processes is always ahead of the other, we essentially have a parallelised version of Gauß-Seidel. Therefore the multirate method is expected to take a similar amount of iterations as Gauß-Seidel. This can be seen in figure 5.2.

Execution time

Here we measure the execution time of the different methods. The methods were timed from the initialisation of the methods, for example just before starting a second process, until after returning the result. The results presented are in wall-clock time and can be seen in figure 5.3.

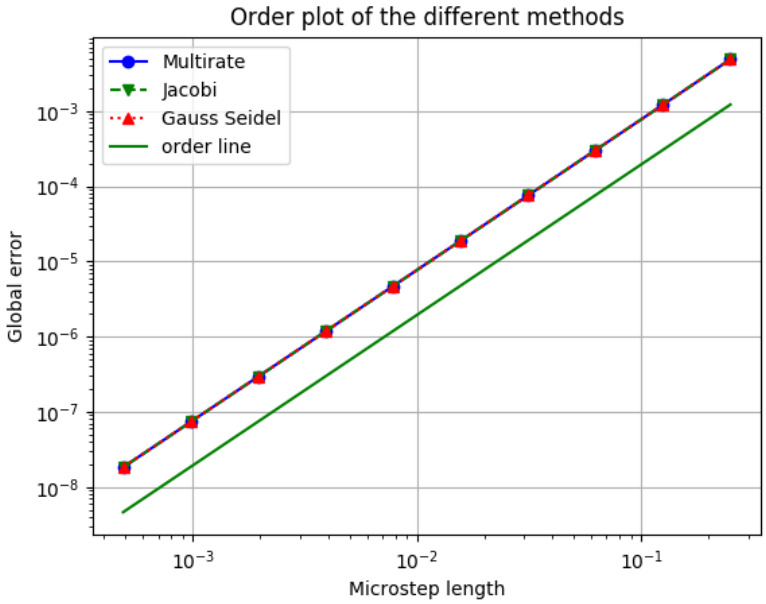

Figure 5.1: Order plot of the three methods solving the system from equation (5.1). The green line is for comparing with a slope of order 2.

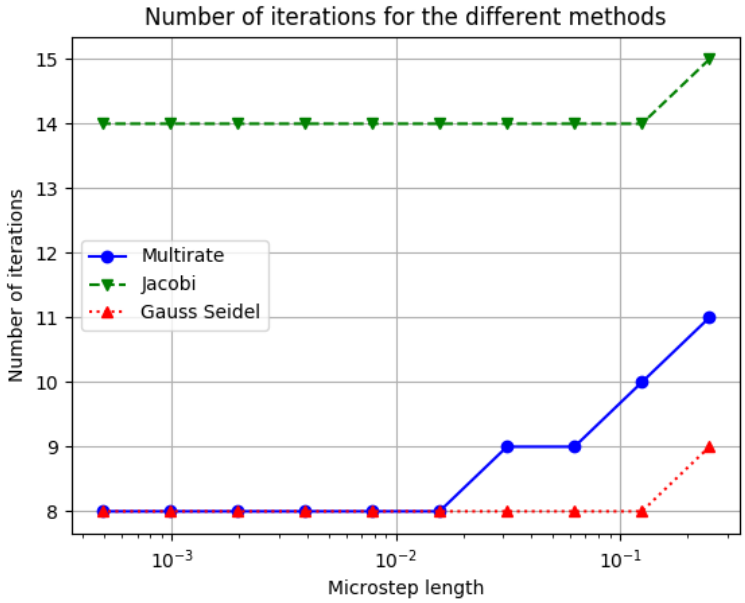

Figure 5.2: Number of iterations done solving the system from equation (5.1)

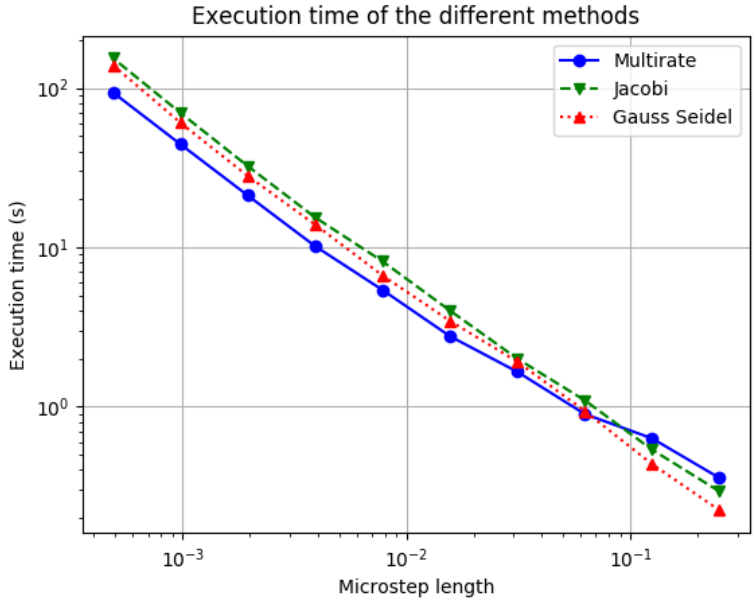

Figure 5.3: Execution time of the different methods solving equation (5.1)

Comments

The three figures above reveal some behaviour of the methods. In figure 5.1 it is clearly seen that the order of the time integration method used is preserved by the iterative scheme. Also the results from figures 5.2 and 5.3 show a minor improvement in performance for the new multirate method. Since these results are from a very simple 2D system where the communication to computation ratio is relatively bad, this small improvement makes the new method look promising.

5.1.2 Varying number of macrosteps

Contrary to the results above, now we use a fix microstep length and vary the amount of macrosteps taken. The results in figure 5.4, 5.5 and 5.6 have been produced using the following solver parameters:

- Macrosteps: Varied
- Microstep length: Both systems held constant at $\tau_x = 0.1\frac{1}{2^9}$, $\tau_x = 0.05\frac{1}{2^9}$.
- Subsystem interpolation: Linear
- Macrostep iteration stopping tolerance: 10^{-14}, in 2 norm
- Subsystem solver: SDIRK2 (see appendix A.0.2)
- Representation of x^*,y^*: Mixed Jacobi, Gauß-Seidel, equation (4.4)

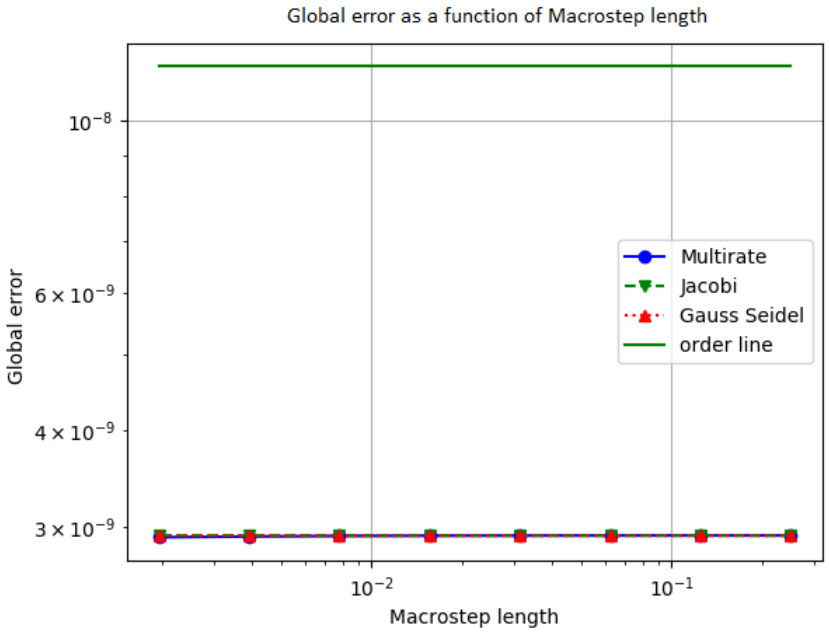

Figure 5.4: Global error as a function of macrostep length for the three methods solving the system from equation (5.1). The green line is for comparing with a constant slope of 0.

Global error

Here we again investigate how the global error is affected by the method. This time we measure what happens when the number of macrosteps used is changed. Again we use the 2-norm to compute the error. Here it is expected that the macrostep length does not affect the global error since the microsteps and the stopping criterion on the iterations are held constant. This behaviour is clearly seen in figure 5.4.

Number of iterations

Here we again see how many iterations are taken by the method. It is expected to take more iterations the longer the macrosteps are. What is different from when we varied microstep length in figure 5.2 is that since we use multiple macrosteps, we here display the average number of iterations done over all macrosteps, as seen in figure 5.5. Also since multiple macrosteps are used, the relationship between the Jacobi and Gauß-Seidel methods described in equation (3.4) doesn't hold anymore.

Execution time

In figure 5.6 we show the execution time of the different methods. Again the methods were timed from just before initialisation until it returned a result, and the time is presented in wall-clock time. It is expected to look similarly to

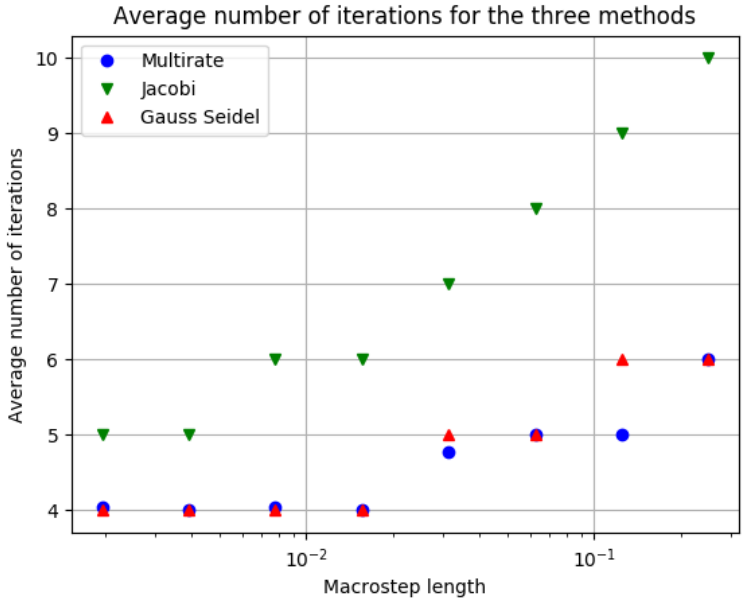

Figure 5.5: Number of iterations done solving the system from equation (5.1)

figure 5.5 since the microstep length is constant, the amount of iterations is the only other factor affecting execution time. Also again the multirate method has slightly better performance compared to the other two methods.

Notes on the results

Here we have seen that the methods preserve the numerical order of the underlying time integration method. Also we saw that the new multirate method shows promising performance results. It is also shown that the number of macrosteps used to solve the problem does affect the performance. We are however not going to investigate this further in this thesis. A more thorough analysis of how to choose the macrostep length can be found in Nevanlinna [4].

5.2 Comparison with analysis of linear systems

In this section we compare the methods' performance to the analytical results from chapters 3.2.3 and 3.2.4. We analyse the number of iterations taken by the methods when solving 2D linear systems with an error tolerance of 10^{-12} according to (3.58) and compare it with a prediction based on the estimates from equation (3.30). For some choices of parameters the number of iterations required are so high that it is practically impossible to calculate this. Therefore we don't investigate what happens after 1000 iterations, since this many iterations is impractical in any application anyways. Comparing with the discrete analysis we analyse how well the prediction of the spectral radius corresponds to the convergence factor from the numerical results. Since we are after the asymp-

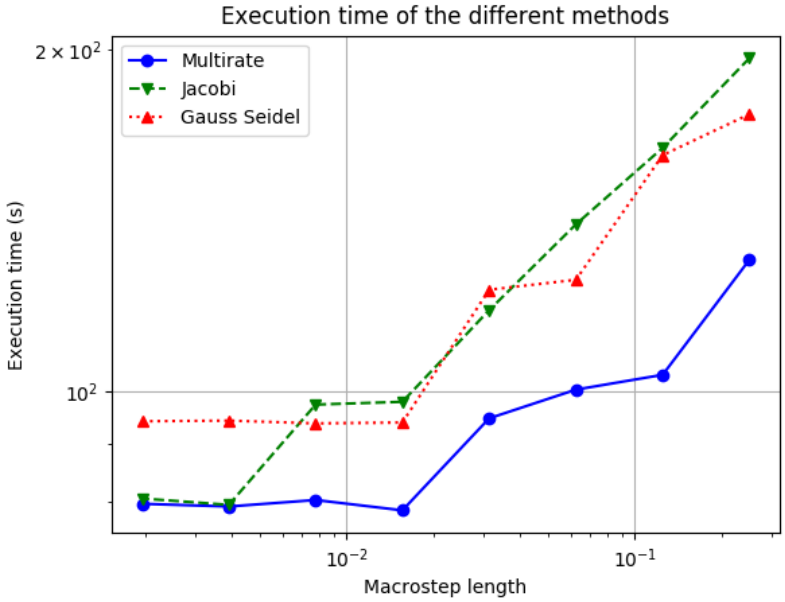

Figure 5.6: Execution time of the different methods solving equation (5.1)

totic convergence rate, the results are calculated based on the convergence rate between the last three iterates before termination. We also complement the Jacobi and Gauß-Seidel results with results from the new multirate method.

All numerical results in this section are produced using the implicit Euler method with a time step of $\tau = 0.01$ for both systems. Since the Multirate method is non-deterministic the results show the average amount of iterations taken over three samples. To compare the numerical results, a reference solution was computed using the Gauß-Seidel method with stopping criterion tolerance of 10^{-14}, close to machine precision. This reference solution was then used to calculate the iteration errors. All the errors are calculated in the max norm $\|.\|_\infty$.

The systems are of the form of equation (3.11) which we solve on a time domain of $t \in [0, 1]$ with initial conditions $[x, y] = [1, 1]$. The different systems' matrices for the test cases are given below. The dots $(*)$ represent the matrix entries that are varied.

Influence of main diagonal terms

Here we test the system

$$A = \begin{bmatrix} -* & 1 \\ 1 & * \end{bmatrix} \tag{5.2}$$

where we vary the two diagonals with different signs.

From figures 5.7, 5.8, 5.9 and 5.10 it is immediately apparent that the entry A_{11} has a significant impact on the performance of the methods compared to entry A_{22}. This can be explained by the difference in signs of the entries where

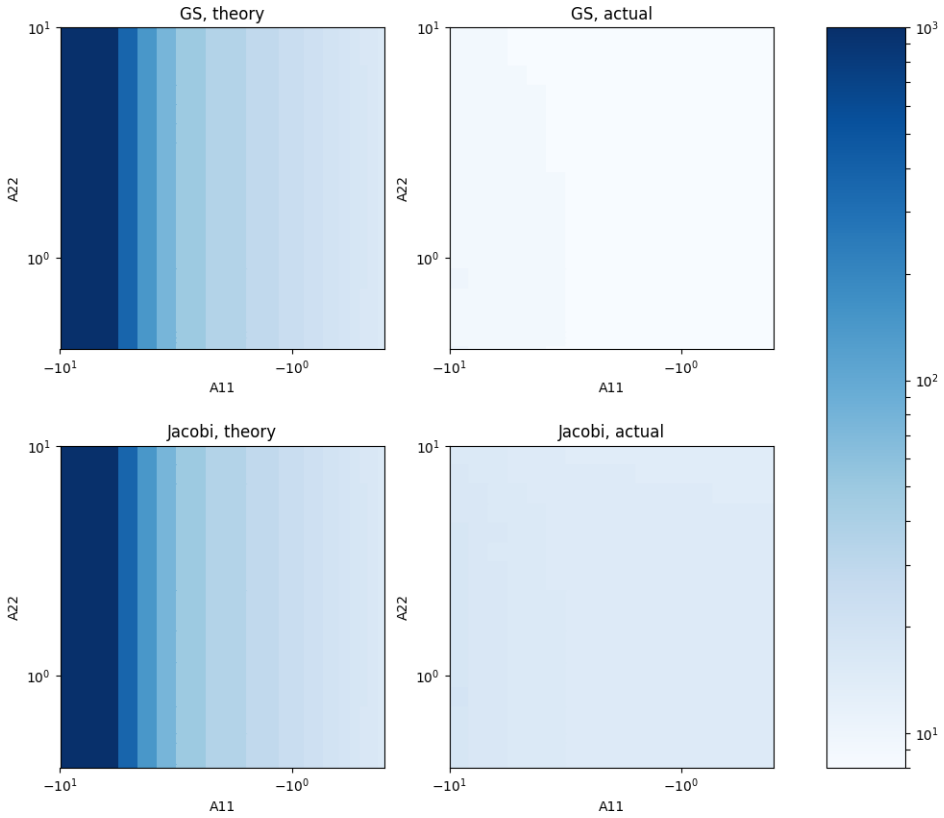

Figure 5.7: Number of iterations done by the Jacobi and Gauß-Seidel methods compared to the estimated amount.

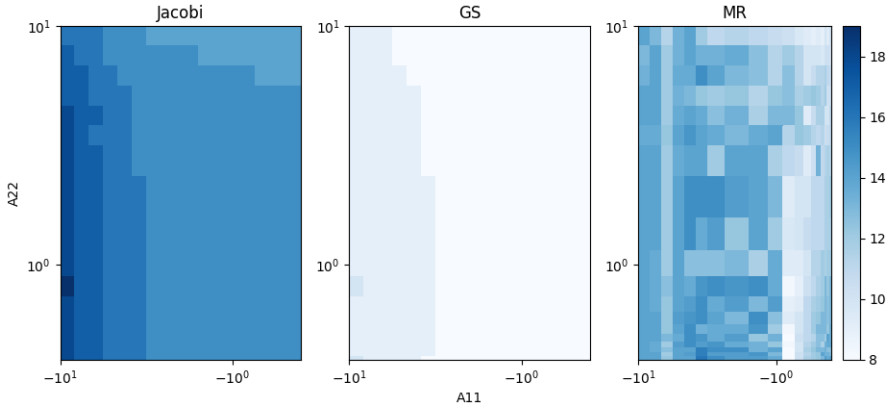

Figure 5.8: Number of iterations done by the three methods.

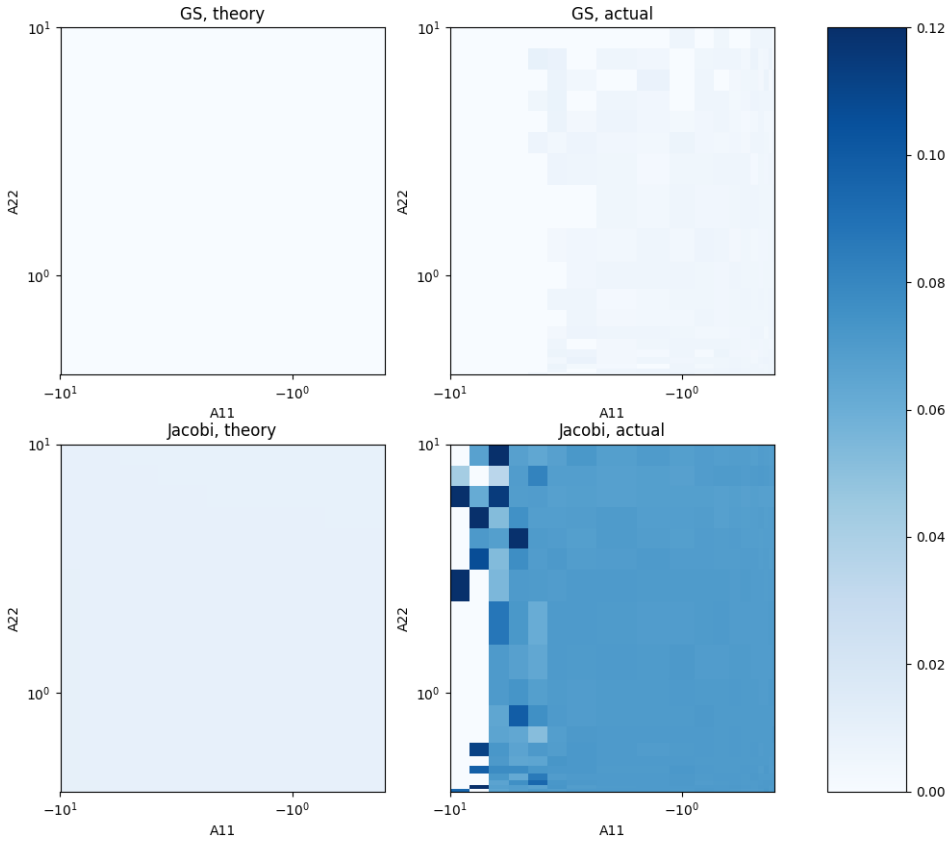

Figure 5.9: Comparison between the spectral radius given by the discrete analysis and the convergence rate obtained through numerical experiments.

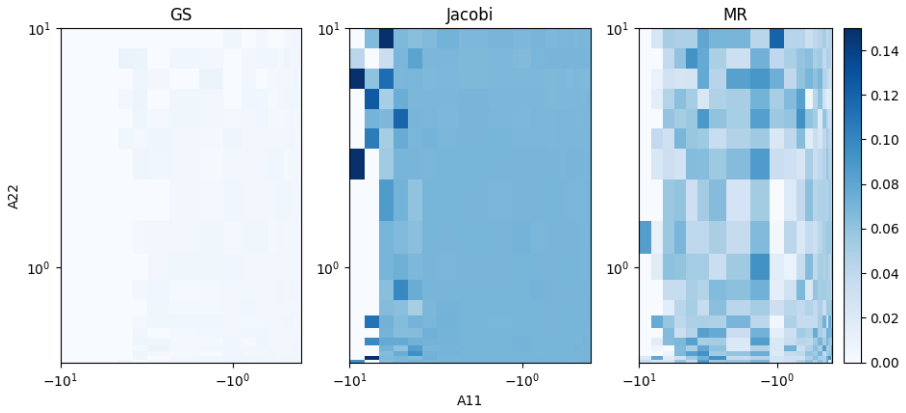

Figure 5.10: Comparison of convergence factor of the three methods.

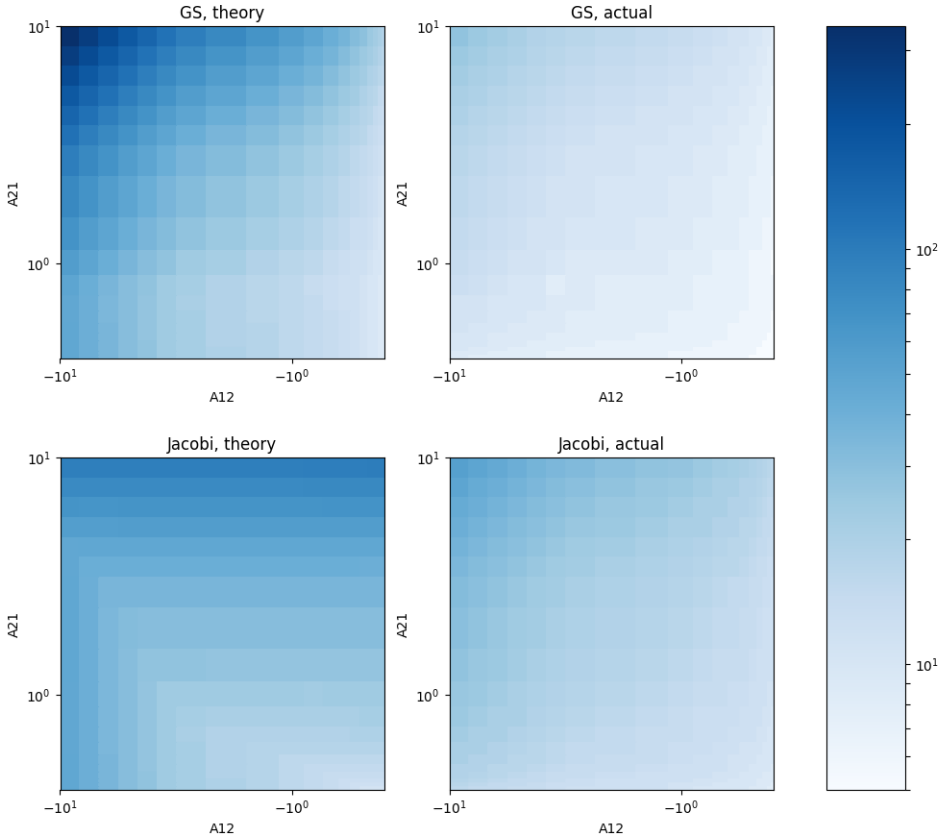

Figure 5.11: Number of iterations done by the Jacobi and Gauß-Seidel methods compared to the estimated amount.

the minus in front of A_{11} leads to exponential growth whereas the positive entry A_{22} leads to exponential decay. Even though the analytical estimate predicts this behaviour, it greatly overestimates the number of iterations needed.

Varying the two couplings

Here we are testing the system

$$A = \begin{bmatrix} 1 & -* \\ * & -1 \end{bmatrix} \tag{5.3}$$

where we vary the two coupling factors with different signs. The results can be seen in figure 5.11, 5.12, 5.13 and, 5.14.

Here the analysis seems to better predict the results than in the previous examples. Also it is evident that the different signs of the coupling constants have little to no impact on the iterations needed.

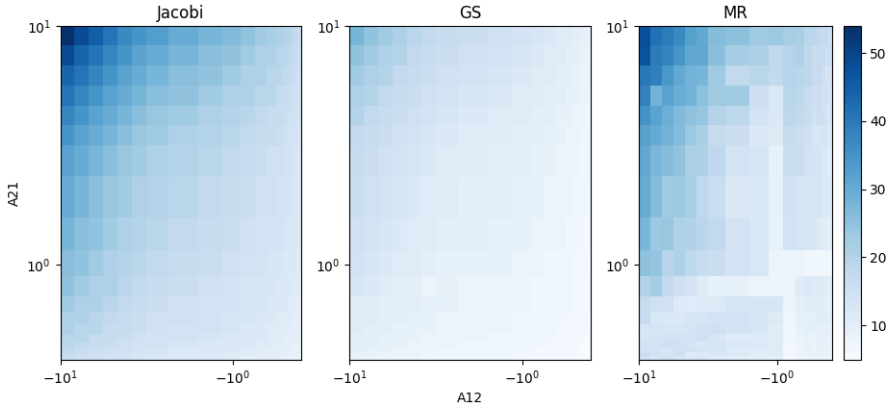

Figure 5.12: Number of iterations done by the three methods.

Here it is quite evident that both coupling factors directly affect the performance significantly.

General remarks

From the results above we can see that the iterations needed in the prediction from equation (3.30) greatly overestimates the importance of the growth factor. This could be explained by the estimate of the norm of the kernel in equation (3.22) being a very crude way to estimate the decay of iterates. Figure 5.11 suggests that the estimate of the influence of the couplings is much more accurate than the estimate of the exponential growth terms.

In figures 5.8 and 5.12 it is evident that Gauß-Seidel required the least amount of iterations of the methods. Jacobi seems to require the most amount of iterations to get to the same accuracy. The Multirate method has due to its non-deterministic nature not as clear figures as the other methods. However the general trend is that the amount of iterations needed is close to the amount of iterations Jacobi takes.

Figures 5.9 and 5.10 show a strange kind of noise for the Jacobi method. It is difficult to say where this comes from. This behaviour is not seen in figures 5.13 and 5.14. This suggests that the iterates from the Jacobi method behave wildly when the method is applied to a problem with a large exponential component A_{11}. This phenomenon will however not be further investigated in this thesis.

The predicted convergence rates are generally better for Gauß-Seidel than Jacobi which is seen in figures 5.9 and 5.13. Gauß-Seidel also seems to always have a better convergence rate compared to Jacobi. When it comes to convergence rates, the new multirate method behaves similar to Jacobi. The non-deterministic property of the new multirate method is also clearly seen in these plots.

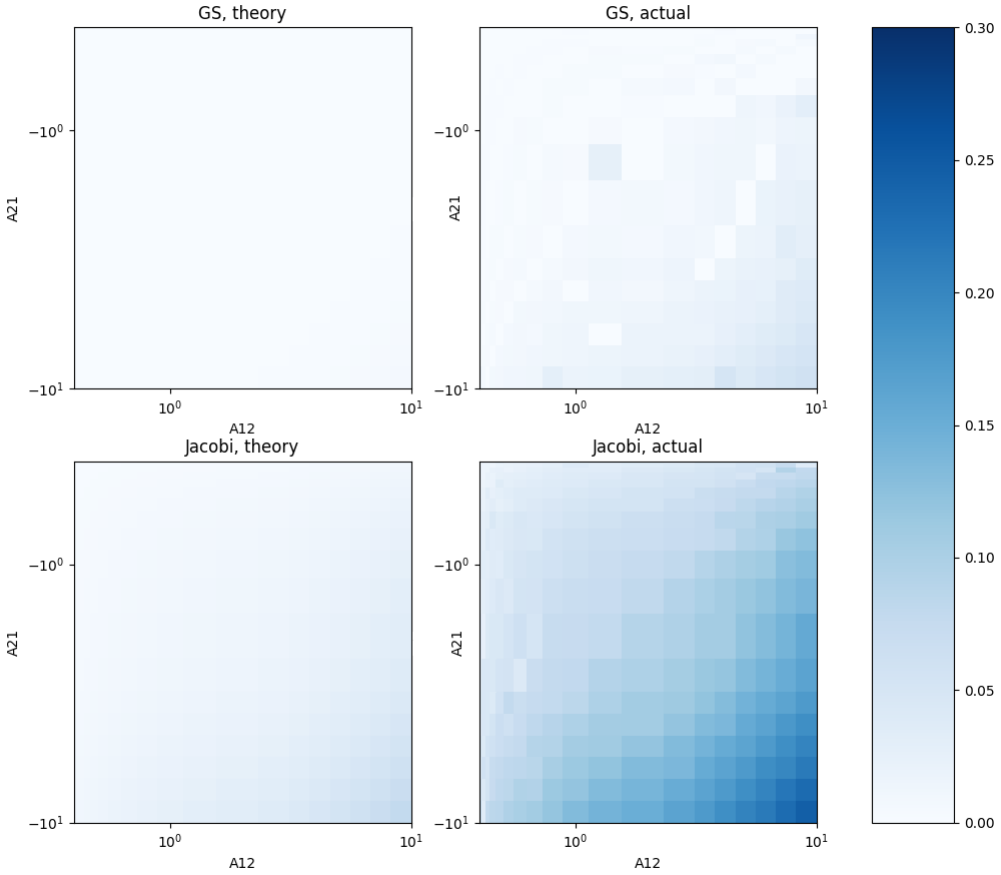

Figure 5.13: Comparison between the spectral radius given by the discrete analysis and the convergence rate obtained through numerical experiments.

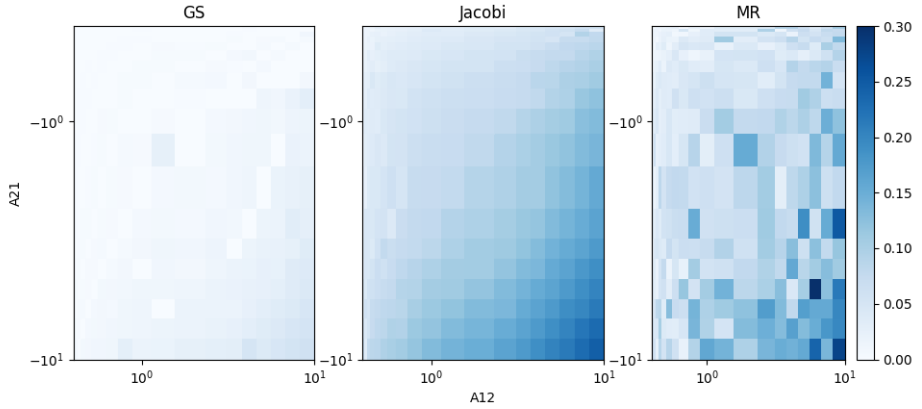

Figure 5.14: Comparison of convergence factor of the three methods.

5.3 Multirate vs Singlerate

In this section we investigate the difference in performance of the methods when applied to a multirate setting, meaning the different time steps are of different length: $\tau_x \neq \tau_y$.

For the results below we use the same settings as in chapter 5.2 except that $\tau_x = 0.005$ and $\tau_y = 0.01$.

Influence of diagonal terms

Here we are testing the system

$$A = \begin{bmatrix} -* & 1 \\ 1 & * \end{bmatrix} \tag{5.4}$$

where we vary the two diagonals with different signs.

Comparing figures 5.15 and 5.16 from the multirate setting to figures 5.7 and 5.8 from the singlerate setting we see that there is not much difference in the performance of Jacobi and Gauß-Seidel. However the new multirate method performs significantly better in this multirate setting, having similar behaviour to Gauß-Seidel.

In figures 5.17 and 5.18 we see that for Gauß-Seidel the convergence rate follows the prediction quite well. Jacobi again performs worse than predicted. The Multirate method again performs very similar to Gauß-Seidel, also when it comes to convergence rate.

Varying the two couplings

Here we are testing the system

$$A = \begin{bmatrix} 1 & -* \\ * & -1 \end{bmatrix} \tag{5.5}$$

where we vary the two coupling factors with different signs.

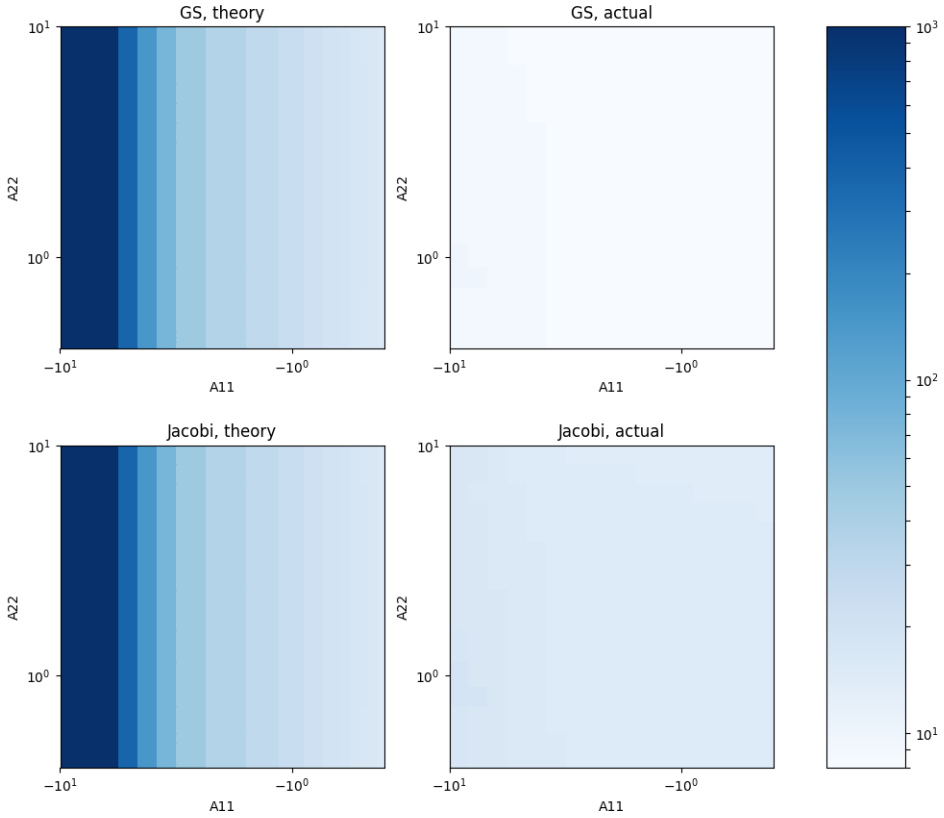

Figure 5.15: Number of iterations done by the Jacobi and Gauß-Seidel methods compared to the estimated amount, this time in a multirate setting

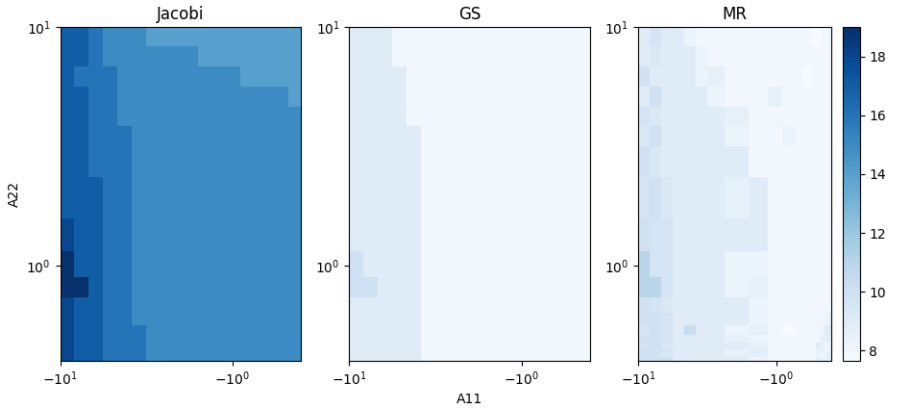

Figure 5.16: Number of iterations done by the three methods in a multirate setting.

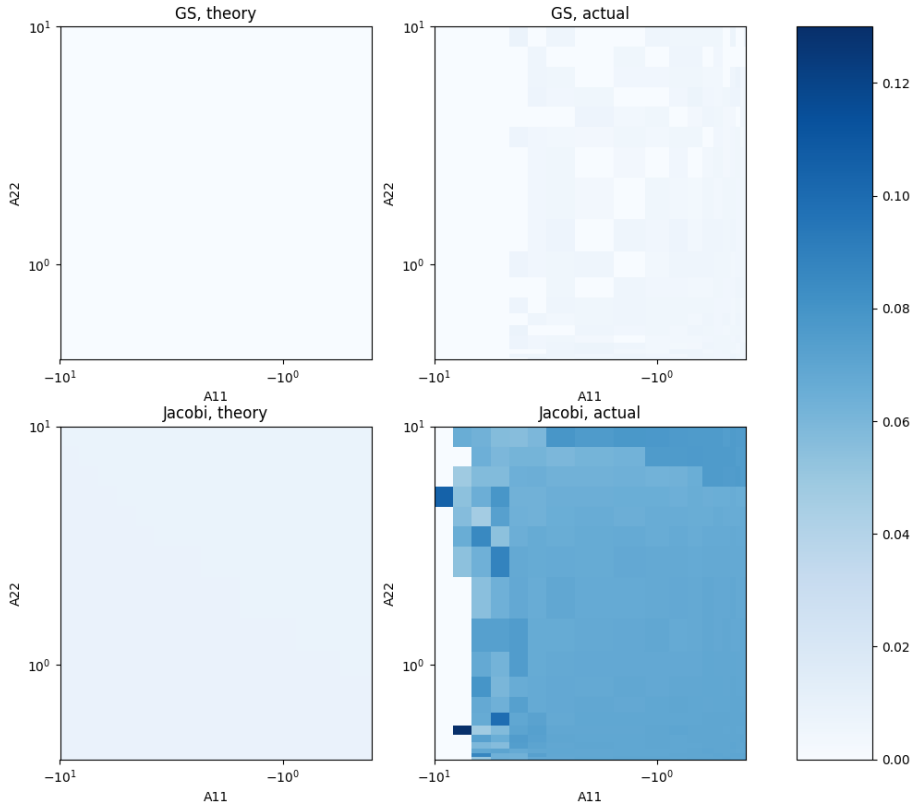

Figure 5.17: Comparison between the spectral radius given by the discrete analysis and the convergence rate obtained through numerical experiments.

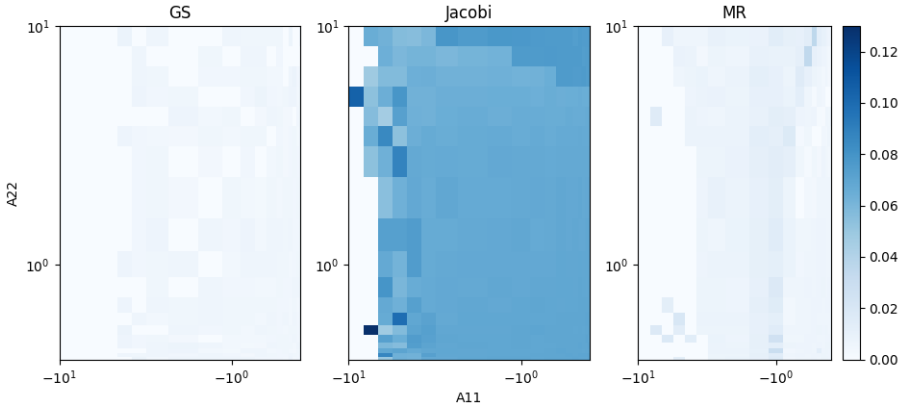

Figure 5.18: Comparison of the convergence factor of the three methods.

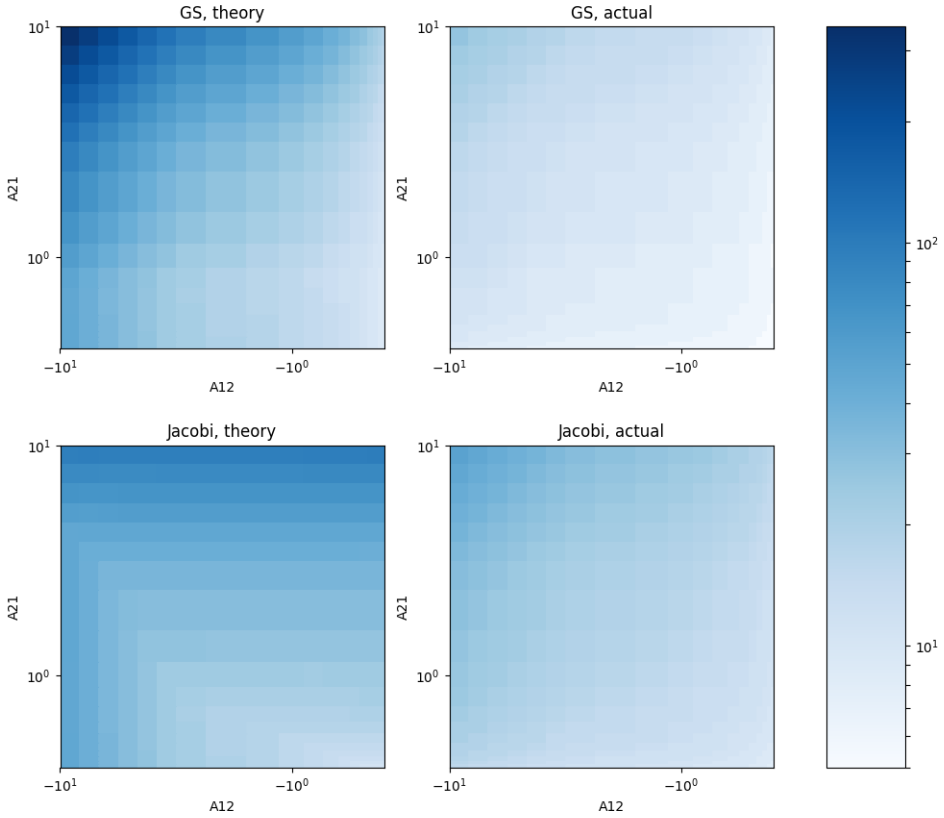

Figure 5.19: Number of iterations done by the Jacobi and Gauß-Seidel methods compared to the estimated amount, this time in a multirate setting.

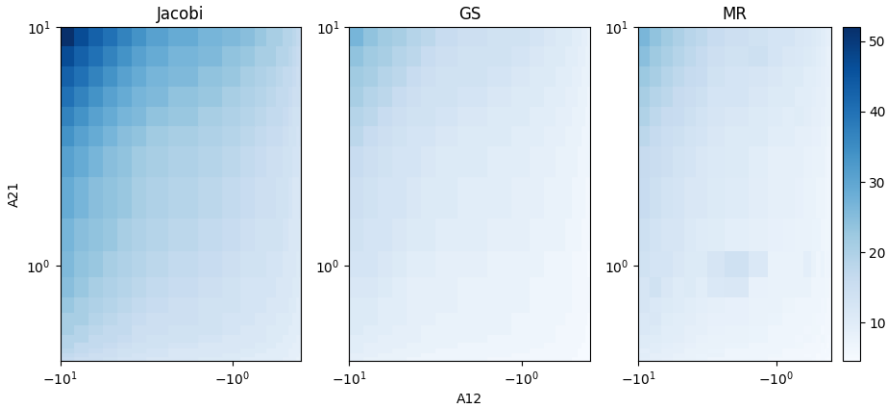

Figure 5.20: Number of iterations done by the three methods in a multirate setting.

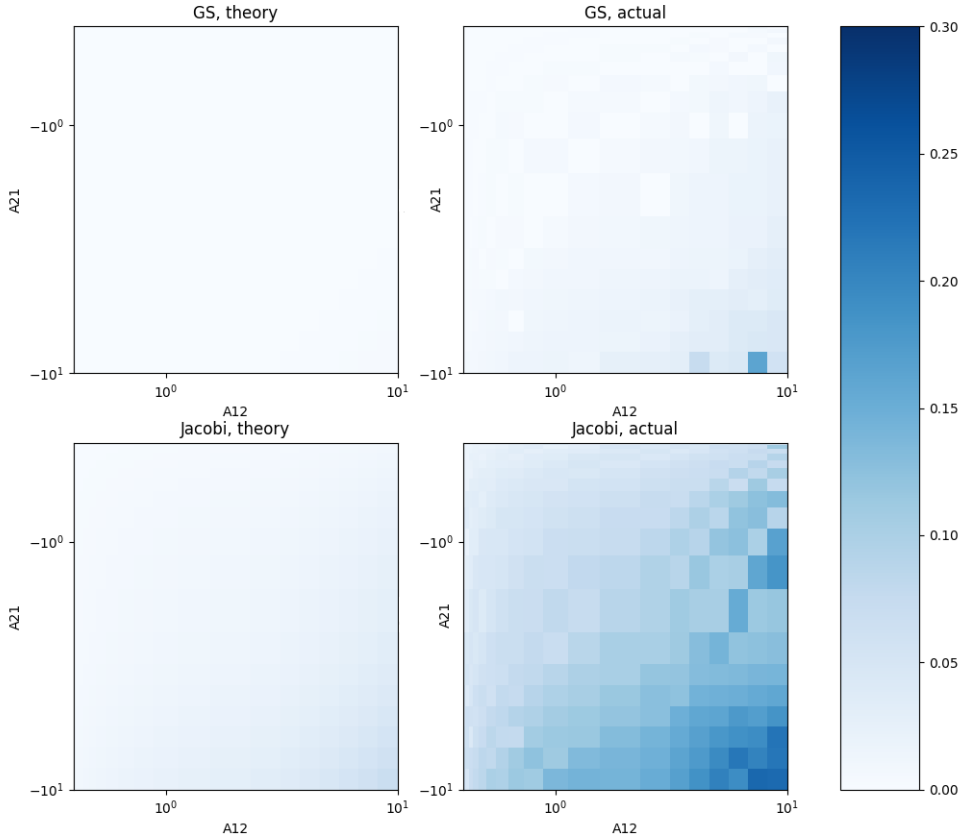

Figure 5.21: Comparison between the spectral radius given by the discrete analysis and the convergence rate obtained through numerical experiments.

Again it can be seen in figures 5.19 and 5.20 that the performance of the multirate method is very similar to Gauß-Seidel.

In figures 5.21 and 5.22 we again see that also when the coupling constants are varied, the new multirate method performs similarly to Gauß-Seidel.

General Remarks

When in a multirate setting, the new multirate method behaves much more like Gauß-Seidel than when in a single rate setting. This can be explained in the following way. Since one of the systems takes larger steps, the process concerning this system will advance in time much faster than the other system. Therefore according to equation (4.4), the slower process will always have information available from iteration $K+1$, similarly to Gauß-Seidel. In a single rate setting, both subsystems and thus both processes use the same time step. The only way for a process to advance faster in time than the other is if the operating system

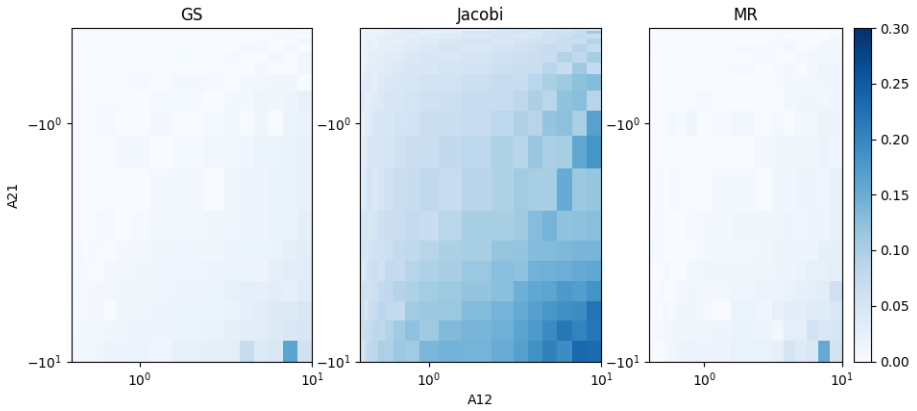

Figure 5.22: Comparison of the convergence factor of the three methods.

interferes with one of the cores associated with a process.

The noise effect in the Jacobi plots, possible induced by the large exponential component, can also be seen in the multirate case in figures 5.17 and 5.18 Also again, when applied to the other system seen in figures 5.21 and 5.22 this noise is not there.

5.4 1d Heat equation

The following results are from applying the methods to a 1 dimensional heat transfer problem. The general form of the heat equation has the following form:

$$\partial_t u + \nabla \cdot (\alpha(x)\nabla u) = f, \quad u = g_d \text{ on } \Gamma_d, \mathbf{n} \cdot \nabla u = g_n \text{ on } \Gamma_n. \tag{5.6}$$

Here f is a source term, α the heat diffusivity, g_d the prescribed Dirichlet condition on the boundary Γ_d and g_n the prescribed Neumann condition on the boundary Γ_n.

The problem we are going to study is a system of two coupled heat equations. The domain is cut in half and split between the two processes where the left domain has a heat diffusivity of α_l and the right domain α_r.

Finite difference discretisation

The discretisation is not done on equation 5.6 directly, but on a 1d variant of the equation. Also since we have a variable $\alpha(x)$ we can split the divergence term in two. We then arrive at

$$\partial_t u + \partial_x \alpha \cdot \partial_x u + \alpha \cdot \partial_{xx} u = f \tag{5.7}$$

which can easily be discretised using a finite difference scheme. On a grid with the distance Δx separating the grid points, the first order derivatives are approximated using a forward finite difference scheme

$$\partial_x u(x_i) \approx \frac{u_{i+1} - u_i}{\Delta x}$$

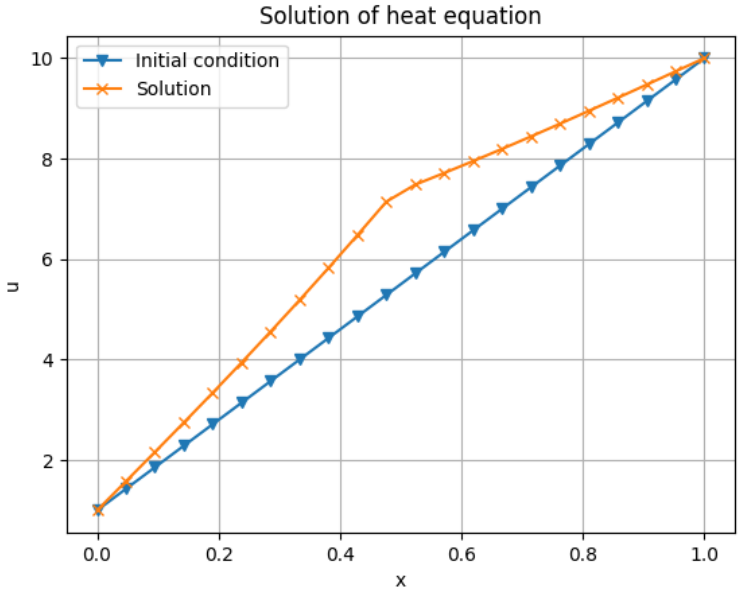

Figure 5.23: The initial conditions and the solution of the heat equation with two different heat diffusivities.

and the second order derivatives are approximated using the central scheme

$$\partial_{xx} u(x_i) \approx \frac{u_{i-1} - 2u_i + u_{i+1}}{\Delta x^2}.$$

Formally the problem description prescribes a discontinuous α. Instead between the two points where the discontinuity is situated, we describe α as the a linear function from α_l on grid point x_{19} and α_r on grid point x_{20}. This makes it possible to solve the problem, at the cost of slightly changing the problem. Since the main objective is to compare the different waveform iteration methods, not the precise computation of the heat distribution, this simplification is reasonable.

Results

The following results are from a simulation of the 1d heat equation on a domain $x \in [0, 1]$ split in two equally big parts. On the left side the heat diffusivity $\alpha = 1$, on the right side $\alpha = 2$. The equation was discretised using a finite difference discretisation with 20 internal grid points. The boundary conditions are both Dirichlet conditions with a temperature of $u_{left} = 1$ on the left side and $u_{right} = 10$ on the right side. The initial condition is a linear temperature distribution conforming to the boundary conditions, as can be seen in figure 5.23. The simulations were done for 0.1 seconds using 1 macrostep, and a termination criterion of $e^K \leq 10^{-12}$ in the max norm was used. The results presented below are done using the SDIRK2 method with different time steps for the different subsystems. For these results, no reference solution was computed. Instead the absolute error is assumed to be similar to the relative difference between

iterations, $\left\|u_e^K - u_e^{K+1}\right\|_\infty$. It is shown by Nevanlinna [3] that this is a good estimate of the absolute error.

To minimize computational effort spent on communication between the systems only relevant data is sent between the systems. In this case it is not necessary to send the full solution between the systems. Data from the boundary between the domains is sufficient.

5.4.1 Decay of iterates

Here we present results showing how the norm of the absolute difference between iterates, $\left\|u_e^K - u_e^{K+1}\right\|_\infty$, decays. In figures 5.24 and 5.25 the system was solved using a singlerate configuration with $\tau_x = \tau_y = 0.001s$ and $\tau_x = \tau_y = 0.01s$ respectively. It is observed that Jacobi uses roughly twice the amount of iterations compared to Gauß-Seidel, as expected. The new multirate method shows very different behaviour depending on the microstep length. This can be explained in the following way. When the microstep length $\tau = 0.01$ only 10 microsteps are taken for each macrostep, compared to 100 microsteps when $\tau = 0.001$. When more steps are taken, it increases the likelyhood of one process consequently being ahead of the other. When this is the case, the performance of the multirate method mimics that of Gauß-Seidel, since one process always has access to data from the most recent iteration (see section 4.2.2). When only a few steps are made, no process can get ahead of the other significantly. In that case the behaviour of the new multirate method comes closer to that of Jacobi. This could also explain the shape of the curve in figure 5.25 which shows a lot of noise compared to the curve in 5.24. To verify this explaination, this needs to be studied in more detail.

In figures 5.26 and 5.27 we present the decay when the heat equation is solved using a multirate setting, $\tau_y = 2\tau_x = 0.002$ and $\tau_y = 2\tau_x = 0.02$ respectively. Again we see that when more microsteps are taken per macrostep, the iterates of the new multirate method look similar to the iterates produced by the Gauß-Seidel method. When comparing between the singlerate and the multirate cases, we observe that the iterations from the new multirate method are more similar to the iterations produced by Gauß-Seidel in the multirate setting.

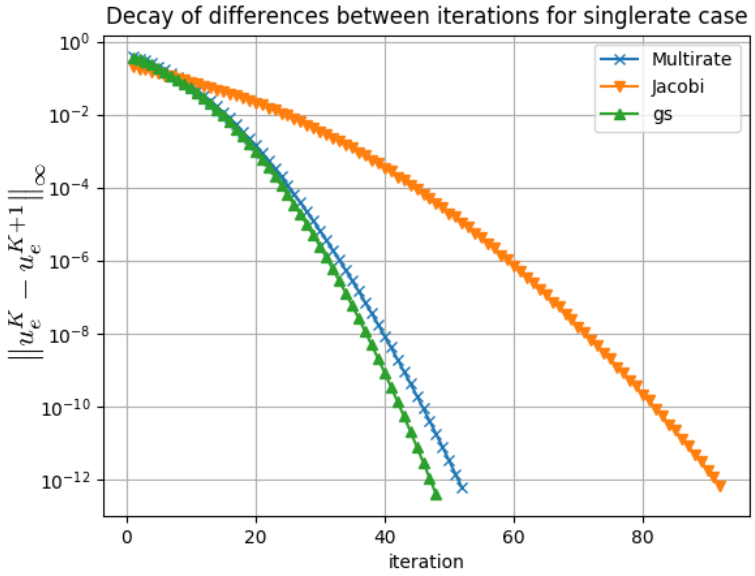

Figure 5.24: Figure showing the decay of the max norm $\left\|u_e^K - u_e^{K+1}\right\|_\infty$ when solving the heat equation using the three different methods in a singlerate setting with $\tau_x = \tau_y = 0.001s$.

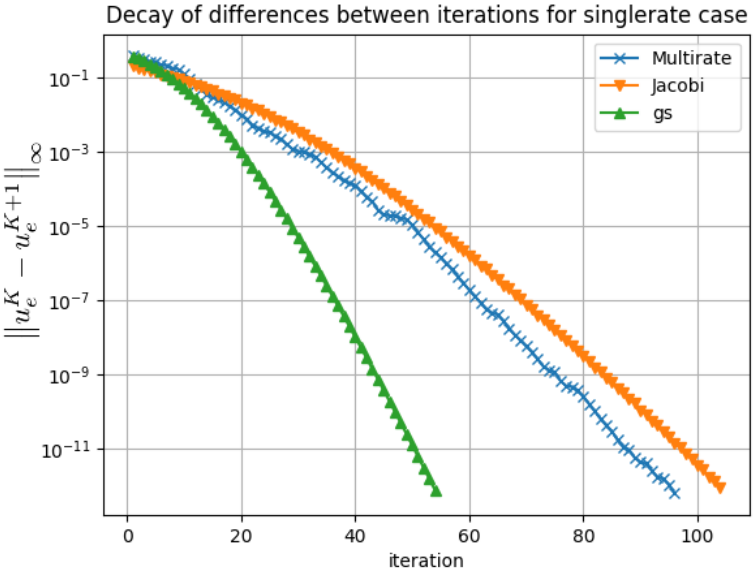

Figure 5.25: Figure showing the decay of the max norm $\left\|u_e^K - u_e^{K+1}\right\|_\infty$ when solving the heat equation using the three different methods in a singlerate setting with $\tau_x = \tau_y = 0.01s$.

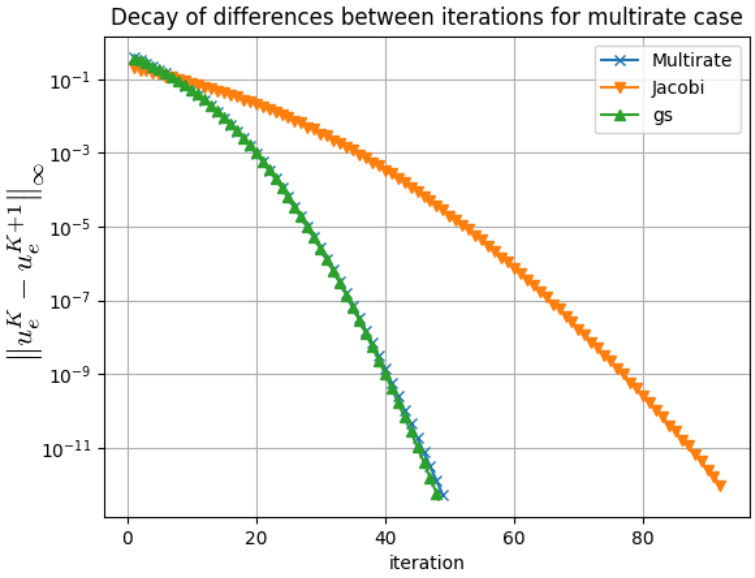

Figure 5.26: Figure showing the decay of the max norm $\|u_e^K - u_e^{K+1}\|_\infty$ when solving the heat equation using the three different methods in a multirate setting with $\tau_y = 2\tau_x = 0.002$. The multirate line is barely visible behind the Gauss -Seidel line.

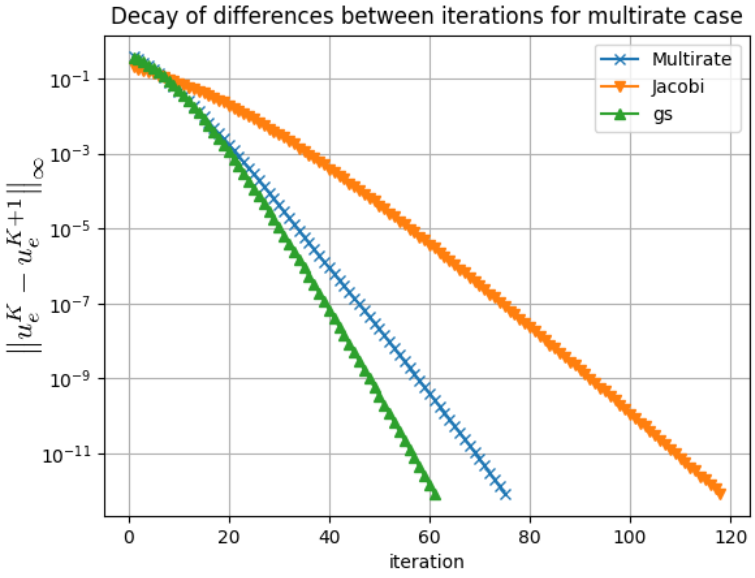

Figure 5.27: Figure showing the decay of the max norm $\|u_e^K - u_e^{K+1}\|_\infty$ when solving the heat equation using the three different methods in a multirate setting with $\tau_y = 2\tau_x = 0.02$.

5.4.2 Performance results

The time measurements were done using the python method `time.perf_counter`, measuring the wall-clock time of the simulation.

Table 5.1: Time (s) taken to simulate using the new multirate method with different internal time steps

$\tau_y \backslash \tau_x$	0.01	0.005	0.002	0.001
0.01	19.4	24.2	59.2	112.9
0.005	23.0	31.6	528.5	106.5
0.002	60.7	58.2	67.3	103.0
0.001	120.2	111.5	108.0	117.8

Table 5.2: Time (s) taken to simulate using the Jacobi method with different internal time steps

$\tau_y \backslash \tau_x$	0.01	0.005	0.002	0.001
0.01	22.4	36.9	108.7	216.1
0.005	42.4	40.1	99.5	207.8
0.002	117.1	107.7	105.2	197.6
0.001	230.8	209.9	205.3	208.9

Table 5.3: Time (s) taken to simulate using the Gauß-Seidel method with different internal time steps

$\tau_y \backslash \tau_x$	0.01	0.005	0.002	0.001
0.01	20.9	29.6	62.0	117.9
0.005	31.6	37.5	69.1	119.6
0.002	67.5	72.6	101.5	148.0
0.001	124.1	124.5	152.3	203.0

Comparing tables 5.1, 5.2 and 5.3 we see that in fact in all but one case the multirate method is faster than both the Jacobi and Gauß-Seidel methods. It is interesting to see that for both Jacobi and the multirate method, it is the shortest time step that dominantly effects the total simulation time. This can be explained by the parallel nature of these methods, where if one system takes longer time steps, it can finish before the system with shorter time steps. This also explains why this can not be seen in the Gauß-Seidel results where it is clear that the sequential nature of the method leads to both systems' time discretisations affecting the computational time, regardless of the ratio between the time steps. What also is interesting is that the relative speedup from the new multirate seems to be greater when both systems take smaller steps.

Table 5.4: Asymptotic convergence rate of the new multirate method using different time step settings.

$\tau_y \setminus \tau_x$	0.01	0.005	0.002	0.001
0.01	0.6412	0.6230	0.4893	0.4900
0.005	0.5195	0.5790	0.6763	0.4209
0.002	0.4897	0.4375	0.4729	0.4016
0.001	0.4876	0.4213	0.3822	0.4382

Table 5.5: Asymptotic convergence rate of the Jacobi method using different time step settings.

$\tau_y \setminus \tau_x$	0.01	0.005	0.002	0.001
0.01	0.7051	0.7027	0.6983	0.6996
0.005	0.7031	0.6576	0.6584	0.6492
0.002	0.6995	0.6578	0.6183	0.6162
0.001	0.6979	0.6498	0.6161	0.6058

Table 5.6: Asymptotic convergence rate of the Gauß-Seidel method using different time step settings.

$\tau_y \setminus \tau_x$	0.01	0.005	0.002	0.001
0.01	0.4965	0.4951	0.4880	0.4878
0.005	0.4931	0.4305	0.4330	0.4192
0.002	0.4882	0.4316	0.3795	0.3756
0.001	0.4880	0.4181	0.3756	0.3618

From the results presented in tables 5.4, 5.5 and 5.6 we see again that the performance of the new multirate method lies in between Jacobi and Gauß-Seidel. Gauß-Seidel also has significantly better convergence rates than Jacobi. In general for all three methods, the convergence rate is slightly better the smaller the time steps used. Looking at the results from Jacobi and Gauß-Seidel it seems that the largest time step used has the most dominating effect on the convergence rate. This is most clear by looking at the first row and first column of tables 5.5 and 5.6. These findings conform to the results from White [5], which say that the ratio of time steps does not affect the convergence rate. Rather, it is the largest time step that affects the convergence rate.

Chapter 6

Conclusions

This thesis mainly consists of two parts. First we analysed two existing waveform iteration methods, writing down their properties analytically and comparing them with numerical results. Secondly we investigated a new method and compared its numerical performance with the existing waveform iteration methods. We saw that the qualitative performance of the Gauß-Seidel and Jacobi methods can be described by the analytical formulas derived in chapter 3.2.3 and 3.2.4. However the analysis seems to overestimate the work needed for solving systems meaning it can only be used as a rough guideline and not so much for accurate predictions of the methods' performances. The Gauß-Seidel and Jacobi methods also performed in a multirate setting as predicted by the analysis.

The new waveform iteration method has shown promising results. Its performance regarding convergence rate lies in between the performance of Jacobi and Gauß-Seidel. When in a single rate setting the performance is closer to that of the Jacobi method, whilst in multirate settings the performance mimics Gauß-Seidel a lot. Because of this convergence behaviour, the method is in most cases faster than both Gauß-Seidel and Jacobi.

6.1 Outlook and future work

This work contains analysis of existing waveform iteration methods. However, no analysis other than an interpretation of the numerical results is done on the new multirate method, mainly because the non-deterministic nature of this method makes it difficult to analyse. Still an analytical description of the behaviour of the iterates produced by the method would benefit the applicability of the new method.

An other point of improvement is how the functions y^* and x^* are produced in equation (4.2). In this thesis, only the mixed Jacobi and Gauß-Seidel method, described by equation (4.4) is explored. There could be much to be gained by choosing smarter combinations of iterates, and even using extrapolation to predict a system's state when that system is behind in the time integration process.

In many of the plots in the result section a lot of noise could be seen in the Jacobi results. A possible explanation was given, where large exponential

factors might be the cause. It could be very interesting to further investigate this phenomenon.

At some points in the program the processes need to synchronize or exchange data. If one process finishes before the other, which more often than not is the case, one of the processes needs to wait for the other to finish. This means that the processes are not efficiently balanced. It would be better to make sure that both processes finish at roughly the same time. There are multiple ways to combat this problem, like changing the partitioning of the equation system or other load balancing techniques. Further exploring these techniques could improve the performance of the new method.

In this thesis, whenever interpolation was necessary, only linear interpolation was used. It would be interesting to see how different interpolation methods would affect the solution. Using a similar technique as in section 3.2.5, it is possible to introduce nearest neighbour interpolation. For higher order interpolations, other techniques need to be employed for the analysis. In [5], White discusses how different polynomial spline interpolations affect the convergence rates of waveform relaxation methods. It would be interesting to see how the new multirate method performs using different interpolations.

All work presented in this thesis revolves around a system that is split in two parts. The theory presented by Vandewalle in [2] and Nevanlinna in [3] and [4] is also valid for systems split into more than 2 parts. It would be very interesting to see how the new method behaves when partitioned into more parts. The communication between the subsystems can become very complicated very fast. To see how this new method scales is also an important topic to be analysed.

Appendix A

Methods for solving Ordinary Differential Equations

To integrate the subsystems we need numerical methods suited for ODE's. We will not go into detail about these methods and how they work. However here comes a list of methods used in this thesis as a reference.

All methods in this section are made for solving systems of the following form.

$$\dot{y} = f(t, y) \tag{A.1}$$

In the methods below y_n denotes the numerical approximation of y at time t_n. The step size is denoted as h.

A.0.1 Implicit Euler method

The implicit Euler method is a Runge-Kutta method of order 1.

$$y_{n+1} = y_n + f(t_{n+1}, y_{n+1}) \tag{A.2}$$

A.0.2 SDIRK2

The SDIRK2 method is a Runge-Kutta method of order 2. It has better dampening properties when applied to the heat equation compared to the trapezoidal rule. It is however slightly more costly due to it having two stages while the trapezoidal rule is a one stage method. SDIRK2 has the following Butcher tableau [9].

$$\begin{array}{c|cc} \alpha & \alpha & 0 \\ 1 & 1-\alpha & \alpha \\ \hline & 1-\alpha & \alpha \end{array}$$

Here $\alpha = 1 - \sqrt{\frac{1}{2}}$.

A.1 Solving systems of equations

When using implicit solvers a system of equations needs to be solved. One way of doing this is using the Newton-Raphson iteration. It works by finding roots to a function using the function's jacobian.

For a function $g(x) \in C^1$ the Newton Raphson method finds the root using the following iteration scheme [10]:

$$Dg(x_k)(x_{k+1} - x_k) = -g(x_k) \tag{A.3}$$

given an initial guess x_0. Here Dg denotes the jacobian of g.

Bibliography

[1] E. Lelarasmee, *The Waveform Relaxation Method for Time Domain Analysis of Large Scale Integrated Circuits: Theory and Applications*. PhD thesis, EECS Department, University of California, Berkeley, 1982.

[2] S. Vandewalle, *Parallel Multigrid Waveform Relaxation for Parabolic Problems*. B. G. Teubner Stuttgart, 1993.

[3] O. Nevanlinna, "Remarks on Picard-Lindelöf iteration," *BIT Numerical Mathematics*, vol. 29, pp. 328–346, Jun 1989.

[4] O. Nevanlinna, "Remarks on Picard-Lindelöf iteration," *BIT Numerical Mathematics*, vol. 29, pp. 535–562, Sep 1989.

[5] J. White, "Multirate integration properties of waveform relaxation with applications to circuit simulation and parallel computation," tech. rep., University of California, Berkeley, 1985.

[6] A. Iserles, *A First Course in the Numerical Analysis of Differential Equations*. Cambridge Texts in Applied Mathematics, Cambridge University Press, 2 ed., 2008.

[7] E. F. Van de Velde, *Concurrent scientific computing*. Springer-Verlag, 1994.

[8] E. Süli and D. F. Mayers, *An Introduction to Numerical Analysis*. Cambridge University Press, 2003.

[9] C. A. Kennedy and M. H. Carpenter, "Diagonally implicit Runge-Kutta methods for ordinary differential equations. a review," tech. rep., NASA Langley Research Center, 2016.

[10] P. Birken, "Numerical methods for the unsteady compressible Navier-Stokes equations," 2012. Habilitation thesis,University of Kassel.

Master's Theses in Mathematical Sciences 2018:E29
ISSN 1404-6342

LUTFNA-3045-2018

Numerical Analysis
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/