NORGES TEKNISK-NATURVITENSKAPELIGE
UNIVERSITET

Convergence and Stability of the Parareal algorithm: A numerical and theoretical investigation

by

Gunnar Staff

PREPRINT
NUMERICS NO. 2/2003

NORWEGIAN UNIVERSITY OF SCIENCE AND
TECHNOLOGY
TRONDHEIM, NORWAY

This report has URL http://www.math.ntnu.no/preprint/numerics/2003/N2-2003.ps
Address: Department of Mathematical Sciences, Norwegian University of Science and Technology,
N-7491 Trondheim, Norway.

Convergence and Stability of the Parareal algorithm: A numerical and theoretical investigation

Author: *Gunnar A. Staff*

Supervisor: *Einar M. Rønquist*

Autumn 2002

Norwegian University of Science and Technology
Department of mathematical science

Preface

This report is a result of the fifth year project (SIF5095P2), covering half a semester.

The idea behind this project emerged from the recent papers [17] and [16], which presented a new algorithm for parallel time-iteration, the *parareal* algorithm. Initially the idea was to implement and test the algorithm on some specially chosen test problems. An early rising question was how the convergence and the computational complexity of the algorithm would be affected by the order of the coarse propagator. During some initial testing, an instability in the algorithm was discovered for some choice of the coarse propagator, especially for the Heat equation. To be able to understand this instability, a theorem for the stability of predictor-corrector scheme was derived. In order to test this theorem, the “Theta-test” was developed. Most of the time and recources was put in to the theorem and its influence on the Heat equation. This came at the expense of the analysis of the other test problems. As a consequence, all the other test-problems needs additonal analysis and testing in order to fully understand the implications of applying the parareal algorithm.

I would like to thank my advisor Einar M. Rønquist for his support and supervision. Without him, this report would never existed. Beside my advisor, I recieved valuable guidance from the following persons: Yvon Maday for his suggestions and advise during our two meetings, especially on how to handle the fine propagator in deriving the stability-expression. Syvert P. Nørsett for his patience in helping me understand the implications of Runge-Kutta schemes. Vegard Kippe for helping me recognize the binomial coefficient in the stability expression for the parareal algorithm.

Trondheim 2002-12-20

Gunnar A. Staff

Abstract

A short introduction to the parareal algorithm is made on basis of [17]. Then, for the autonomous differential equation, a theorem for the stability property of the algorithm is derived. The theorem states that $\mathcal{G}_{\Delta T}$ must be strong A-stable with $\lim_{z\to-\infty}|R(z)| \leq \frac{1}{2}$, in order for the algorithm to be stable for all n, k. Several test problems involving periodic solutions, semidiscretized parabolic PDE's, stiffness and nonlinearity are calculated using different implicit Runge-Kutta schemes with different order and stabiliy properties. Using the theta-method, the property $\lim_{z\to-\infty}|R(z)| \leq \frac{1}{2}$ is tested, and the results verifies the theorem. Instabilities in the two different heat-equation test problems are predicted using the theorem. Different formulations of the system, for which the coarse propagator $\mathcal{G}_{\Delta T}$ operates, are tested. Substantial differences is found, and explenations are discussed.

Contents

1 Introduction

Time is by it's very nature strictly sequential. It's natural to think that we need to know our systems state tomorrow in order to find the state the day after.

So why this desire to parallelize time-integration? The main reason is the need to solve important problems more rapily then is currently possible. Examples are problems where solutions are needed in real-time, e.q flight/boat-simulators or control-problems, or when the computational time is so long that no one bothers to wait for the result.

In [12] the following reasons for this are given.

- f, from $y' = f(t, y)$, is expensive to evaluate, as might be the case if each f evaluations requires the solution of an auxiliary problem.
- The number of equations, m, in the system is large, a property characteristic of semi-discretized PDEs or large system of electrical circuits.
- The interval of integration $t \in (t_0, T)$ is long.
- The IVP must be solved repeatedly, as hapens in parameter fitting problems.

We can increase computational speed by either using a faster chip or by developing a faster algorithm. Another alternative is to buy more chips. At the time this article was written it was possible, for less then 200'000USD, to by a complete 128 nodes cluster with the following spec:

- 256 1.5GHz AMD processors (2 processors per node)
- 256GByte Main Memory (2GBytes per node)
- 15TByte Disc (IDE)
- 1GBits/sec Ethernet

This is a tremendous number-cruncher for a price that several companies can afford.

But how do we put all our chips to work?

For space there exist a variaty of parallel methods, e.g domain decomposition and parallel linear algebra. Most of these methods work for different kinds of problem, and they often scale well with the problem-size/number of prosessors.

There exist some parallel algorithms for Initial Value Problems (IVP) of ODE as well, but they have more restrictions then the space parallel-methods.

In [7], Gear classifies the means of achieving parallelism in IVP solvers into two main categories:

1. *parallelism across the system*, or equivalently *parallelism across space*
2. *parallelism across the method*, or equivalently *parallelism across time*

The class (1) will in general be methods to parallelize the right-hand-side of the system. In general the space-parallelizing methods fall into this category. Naturaly, we can expect speedup from only large or other computational-intensive systems since communication-cost will be significant for small systems. Other methods are *waveform relaxation* and *modular integration*. In class (2) there exist a variaty of methods, but they are in general limited to a maximum speedup of about ten.

Recently there has been proposed a new parallel algorithm for IVP ([17],[16],[1],[2]). This algorithm is more in the spirit of domain decomposition. As proposed in [16], we call the algorithm the *parareal* algorithm, after the desire to solve complex problems in real-time. The initial tests looks promising. So far it has been tested on molecular dynamic simulations ([1]), PDE with nonlinear right-hand-side ([16]), PDE with a control as the right-hand-side ([17]) and a non-differential PDE ([2]). Some analysis has been already been done, mostly in [2] and [16].

But this algorithm is still young, and some of it's advantages and weaknesses are probably not discovered. There has not (as far as the author knows) been done any tests with different classes of odesolvers. Nor has it been tested how the algorithm handles solutions with different time-periodicity, nor stiff equations.

This paper can be considered as a preliminiary research on this parareal algorithm. It will present numerical simulations on three main test problems, where different solvers are applied.

Section 2 gives an introduction to the algorithm, as it is defined in [17]. It also points out some obvious, and some not so obvious properties of the algorithm.

Section 3 presents the various solvers that we intend to use, and highlights some of their importent properties. It also discusses general stability properties for single-step ode-solvers. This forms a basis for Section 5.

Section 4 presents the test problems, and gives a brief analysis of how we expext the parareal algorithm to perform.

Section 5 presents a stability analysis of the parareal algorithm, and ends with the most important result in this report, the theorem formulating the stability properties of the algorithm for certain class of problems.

Section 6 presents the results of the numerical simulations, and comments on the various properties.

Section 7 concludes, and points out problems that need further research.

The focus has not been on optimal implementation of the algorithm and the different solvers. Here, a lot could have been done to increase speed. That is one of the reasons why all the implementation and simulation is done in `MATLAB`. `MATLAB` increases the implementation speed, at the expense of simulation-speed. All parallelism is simulated.

We expect the reader to have a basic knowledge in real analysis, complex analysis, linear algebra and metric spaces. Some results, that we do not expect known to all, are presented in Section A.

2 The Parareal Algorithm

For completeness we present the algorithm as presented in [17].

2.1 The algorithmic idé

We want to solve the general problem

$$\begin{cases} \frac{\partial y}{\partial t} + Ay = 0 \\ y(t_0) = y^0 \quad t \in (t_0, T), \end{cases} \tag{1}$$

where A is an operator from a Hilbert space V into V'. The strategy is to do a time decomposition in the spirit of domain decomposition. We define the decomposition as

$$t_0 = T_0 < T_1 < \cdots < T_n = n\Delta T < T_{n+1} < T_N = T.$$

We are now free to rewrite our problem (1) as

$$\begin{cases} \frac{\partial y_n}{\partial t} + Ay_n = 0 \\ y(T_n^+) = \lambda_n \quad t \in (T_n, T_{n+1}), \end{cases} \tag{2}$$

for any $n = 0, \ldots, N-1$. The collection of solutions of (2) $\{y_0, y_1, \ldots, y_N - 1\}$ is connected to the solution y of the original problem (1) if and only if, for any $n = 0, \ldots, N-1$

$$\lambda_n = y(T_n),$$

or written with the syntax of (2)

$$\lambda_n = y_{n-1}(T_n) \quad \text{with} \quad y_{-1}(T^0) = y_0$$

We complement the problem with a cost functional to be minimized

$$\mathcal{J}(\Lambda) = \sum_{n=1}^{N-1} \| \, y_{n-1}(T_n^-) - \lambda_n \, \|^2,$$

where $\Lambda = \{\lambda_0 = y_0, \lambda_1, \ldots, \lambda_{N-1}\}$.

It's obvious that the minimum of $\mathcal{J}$ is zero with the choice $\lambda_n = y(T_n)$. We now assume that A is time-independent, and introduce the propagator $\mathcal{F}_{\Delta T}$ such that for any given μ, $\mathcal{F}_{\Delta T}(\mu)$ is the solution at time ΔT of (1) with $y^0 = \mu$. We are now in a position to write (2) in a matrix form

$$\begin{pmatrix} \mathbb{I} & 0 & 0 & \ldots & 0 \\ -\mathcal{F}_{\Delta T} & \mathbb{I} & 0 & \ldots & 0 \\ 0 & -\mathcal{F}_{\Delta T} & \mathbb{I} & 0 & \ldots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & -\mathcal{F}_{\Delta T} & \mathbb{I} \end{pmatrix} \begin{pmatrix} \lambda_0 \\ \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_{N-1} \end{pmatrix} = \begin{pmatrix} y^0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \tag{3}$$

or in matrix notation

$$M\Lambda = F.$$

Normally an inversion of a triangular system involves $\mathcal{O}(N)$ resolutions. It's now we introduce our iterative scheme that allowes us to construct a sequence Λ^k that converges toward the exact solution of (3). We discretize (2) using a coarse propagator $\mathcal{G}_{\Delta T}$ and a numerical scheme, e.g. an implicit Euler scheme:

$$\frac{\mathcal{G}_{\Delta T}(\mu) - \mu}{\Delta T} + A\mathcal{G}_{\Delta T}(\mu) = 0,$$

where $\mathcal{G}_{\Delta T}$ is an approximation of $\mathcal{F}_{\Delta T}$. Our predictor-corrector scheme is then defined as

$$\lambda_{n+1}^{k+1} = \mathcal{F}_{\Delta T}(\lambda_n^k) + \mathcal{G}_{\Delta T}(\lambda_n^{k+1}) - \mathcal{G}_{\Delta T}(\lambda_n^k), \tag{4}$$

where the subscript n is time-partition in (2), and the superscript k is the iteration-number. Notice that $\mathcal{F}_{\Delta T}$ is calculated from Λ^k, which is known. This implies that $\mathcal{F}_{\Delta T}$ is implemented in parallel. $\mathcal{G}_{\Delta T}$ on the other side is calculated from the previous (in time-partition) λ from this iteration, and is therefore strictly serial. As for $\mathcal{F}_{\Delta T}$ we introduce the matrix

$$\widetilde{M} = \begin{pmatrix} \mathbb{I} & 0 & 0 & \dots & 0 \\ -\mathcal{G}_{\Delta T} & \mathbb{I} & 0 & \dots & 0 \\ 0 & -\mathcal{G}_{\Delta T} & \mathbb{I} & 0 & \dots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & -\mathcal{G}_{\Delta T} & \mathbb{I} \end{pmatrix}$$

and write the iterative procedure in the matrix form

$$\Lambda^{k+1} = \Lambda^k + \widetilde{M}^{-1} Res^k,$$

where the residual Res^k is defined by $Res^k = F - M\Lambda^k$.

The algorithm we implement is then

Algorithm 1 The parareal algorithm

```
λ_0^0 ← y_0
for i = 0 : N − 1 do
  λ_{i+1}^0 ← G_ΔT(λ_i^0)
end for
solve F_ΔT(λ_i^0) in parallel on i = 1, ..., N processors
k ← 0
while true do
  λ_0^{k+1} ← λ_0^k
  for i = 0 : N − 1 do
    solve G_ΔT(λ_i^{k+1})
    λ_{i+1}^{k+1} ← G_ΔT(λ_i^{k+1}) + F_ΔT(λ_i^k) − G_ΔT(λ_i^k)
  end for
  if convergence then
    break
  end if
  solve F_ΔT(Λ^{k+1}) in parallel on i = 1, ..., N processors
  k ← k + 1
end while
```

Convergence can be tested using $\| \lambda_n^{k+1} - \lambda_n^k \| \leq tol \ \forall n$ as test criteria.

2.2 A visual example of the algorithm

So how will this work? Lets look at a simple example and try to visualize the iterations.

We consider the simple differential equation for population growth with proper initial value

$$\begin{aligned} y' &= y(1-y), \\ y(t_0) &= 0.01 \quad t \in (t_0 = 0, T = 10) \end{aligned}$$

The whole idea behind the algorithm is to find accurate (accurate enough) starting-values (Λ) for the paralell implementation of the fine propagator $\mathcal{F}_{\Delta T}$. Therefore the quantities we want to investigate are

Legend			
—•—	y_s	+	Λ^k
–•–	y_c^{k+1}	–×–	Λ^{k+1}
——	y_f^{k+1}		

Table 1: Legend for Figure 1

- The results from $\mathcal{F}_{\Delta T}$, y_f. We name $\mathcal{F}_{\Delta T}(\Lambda^k) = y_f^k$ where k is the iteration-number. We also denote $\mathcal{F}_{\Delta T}(\lambda_n^k) = \mathcal{F}_{n\Delta T}^k$, illustrating which portion of the time domain this propagator evaluate.
- The results from $\mathcal{G}_{\Delta T}$, y_c. Again we name $\mathcal{G}_{\Delta T}(\Lambda^k) = y_c^k$ where k is the iteration-number.
- Λ^k, used to calculate y_f^k and y_c^k
- The corrected Λ^{k+1} based on (4)

We discretize using $\delta t = 0.01$ for $\mathcal{F}_{\Delta T}$, and $\Delta T = 1$ for $\mathcal{G}_{\Delta T}$. This gives us $N = 10$, which is both the number of processors, and the maximun number of iterations. Implicit midpoint-scheme from Table 3 is used for both $\mathcal{G}_{\Delta T}$ and $\mathcal{F}_{\Delta T}$, but the type of solver is of no importence for the time being.

In Figure 1 these quantities, and a serial calculation named y_s, are visualized for the initialization and the three first iterations of the algorithm. y_c and Λ^{k+1} are drawn as points ($\bullet$ versus $\times$) marking their position, with lines between. The lines may be deceiving because this gives an illution that they are continous. Remember that only the initial propagation of $\mathcal{G}_{\Delta T}$, y_c^0 uses it's previous calculated value as startingvalue for the next calculation. Otherwise, the propagation of $\mathcal{G}_{\Delta T}$ is based on the most updated Λ. Λ is the initial values of $\mathcal{F}_{\Delta T}$ (and $\mathcal{G}_{\Delta T}$). But the lines are used to make the plot more readable. The result from $\mathcal{F}_{i\Delta T}$ are plotted as separate lines, but a final result will of cource be a concatenation of all the $\mathcal{F}_{i\Delta T}$, where the last value of $\mathcal{F}_{i\Delta T}$ is substituted with the first value of $\mathcal{F}_{i+1\Delta T}$. The separation is also done to increase readability.

The first we notice in Figure 1(a) (k=0) is that the $\mathcal{F}_{0\Delta T}^0$ is exact equal the serial version. This is obvious since they have the same initial value, and they propagate using the same scheme and the same stepsize. Also Λ^0 coincide with y_c^0. This is also obvious since the initial $\mathcal{G}_{\Delta T}$ generates Λ^0. The results from $\mathcal{G}_{\Delta T}$, y_c^0, deviate from the serial solution, y_s, to some extent. This is also expected. If not the coarse solution would be accurate enough, and the fine solution would be over-sampled.

In Figure 1(b) we notice that now also $\mathcal{F}_{1\Delta T}^1$ coincide with y_s. The predictor-corrector scheme (4) is run for the first time, and forms Λ^1, which is the initial values for $\mathcal{F}_{\Delta T}^1$. Notice that Λ^1 is considerably closer to the exact solution than Λ^0.

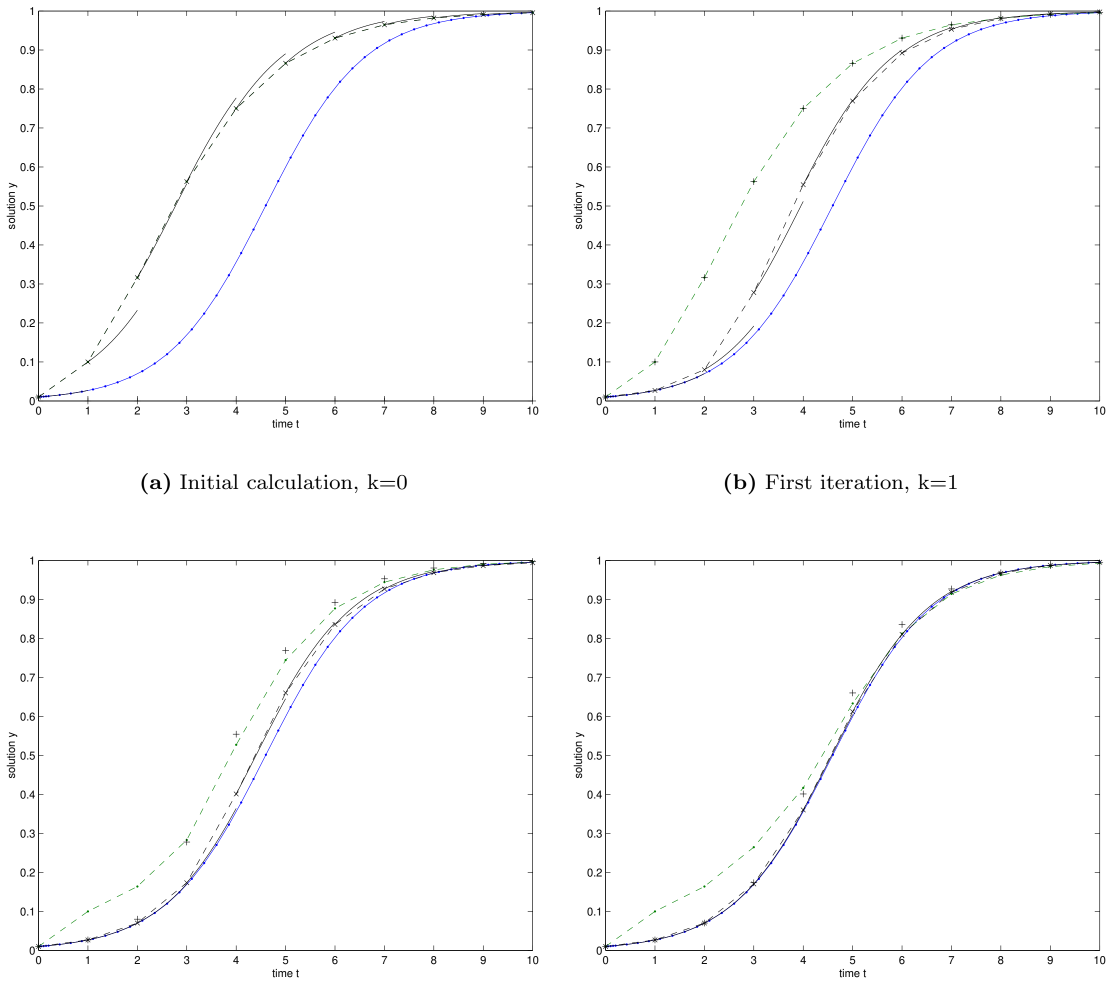
In Figure 1(c), again a new segment of $\mathcal{F}_{\Delta T}$, $\mathcal{F}_{2\Delta T}^2$, coincide with y_s. Notice that the first timesteps of the $\mathcal{G}_{\Delta T}$ doesn't seem to get more accurate. The difference we see is the difference in accuracy between $\mathcal{G}_{\Delta T}$ and $\mathcal{F}_{\Delta T}$.

In Figure 1(d) there is no visual deviation between y_f and y_s.

During this small example we have discovered a couple of interesting things about the algorithm. We will now formulate them more precise

2.3 Properties of the algorithm

First let's summarize what we know about accuracy.



(a) Initial calculation, k=0

(b) First iteration, k=1

(c) Second iteration, k=2

(d) Third iteration, k=3

Figure 1: Visualization of the parareal algortihm's approach to convergence. $t \in (0, 10)$, $\Delta T = 1$, $\delta t = 0.01$. The legend can be seen in Table 1

Proposition 1 *Given that $\mathcal{G}_{\Delta T}$ and $\mathcal{F}_{\Delta T}$ are convergent and stable for the chosen schemes and timestep dt and dT. Then, for iteration k (assuming $k = 0$ is first iteration)*

$$\| y_s - y_p \| \sim eps, \quad t \in (t_0, k\Delta T),$$

where eps is the machine accuracy. This means that

$$\| y_s - y_p \| \sim eps, \quad t \in (t_0, T),$$

at $N - 1 = \frac{T-t_0}{\Delta T} - 1$ iterations.

Like *Conjugated Gradient* the parareal algorithm is exact at maximum number of iterations.

Proposition 2 *We are free to choose what kind of ode-solver we want for $\mathcal{G}_{\Delta T}$ and $\mathcal{F}_{\Delta T}$. We have no limitations in the choice of order or number of steps. The only requirement is that, of course, the chosen methods have to be convergent and stable for the required stepsize. Nothing prevent us from having "intermediate" steps in $\mathcal{G}_{\Delta T}$.*

It would of course not be a good ide to choose a lot of intermediate steps for $\mathcal{G}_{\Delta T}$ if it makes $\mathcal{G}_{\Delta T}$ a much larger time consumer then $\mathcal{F}_{\Delta T}$.

It will, in Section 5, become clear that $\mathcal{G}_{\Delta T}$ has to be more then just A-stable (for autonomous problems), but actually strong A-stable.

Proposition 3 *$\mathcal{G}_{\Delta T}$ doesn't have to calculate on the same model as $\mathcal{F}_{\Delta T}$. We are free to simflify the model by e.g. removing highly oscillating terms that is totally undersampled by ΔT, or use a coarser space-grid if we calculate a PDE.*

The use of a coarser space-grid will of course introduce a new problem, that is how to find the missing values in Λ, used as startingvalue for $\mathcal{F}_{\Delta T}$. Interpolation is one option.

Proposition 4 *The algorithm favors single-step methods in $\mathcal{G}_{\Delta T}$.*

The startup-problems of the multi-step class is a big disadvantage in the parareal algorithm. If we don't want intermediate steps for $\mathcal{G}_{\Delta T}$, single-step methods is the only option. If we do allow intermediate steps, we still have to start the multistep method with a single step method. If the number of iterations for $\mathcal{F}_{\Delta T}$ is large, the choice of a multistep method versus single-step method would be based on other criteria (e.g adaptiv stepsize).

Proposition 5 *$\mathcal{G}_{\Delta T}$ can be implemented in parallel using a parallelism across the method.*

Nothing prevent us from parallelizing the sequential $\mathcal{G}_{\Delta T}$ using already known parallel methods. [14] states that there, for every non-stiff and stiff IVP, exists a s-stage Runge-Kutta *predictor-corrector* formula that yields a speedup of almost s. Note that predictor-corrector formula in this context is not the same as our predictor-corrector scheme.

2.4 Computational complexity in a parallel implementation

The computational complexity for the iterative algorithm can be calculated as

$$(T_{\mathcal{G}_{\Delta T}} + (T_{\mathcal{F}_{\Delta T}})_{max} + T_{com}) \cdot (k+1),$$

where $T_{\mathcal{G}_{\Delta T}}$ is the time used on the coarse iterator $\mathcal{G}_{\Delta T}$ per iteraton, $(T_{\mathcal{F}_{\Delta T}})_{max}$ is the time used on the fine iterator $\mathcal{F}_{\Delta T}$ from the "slowest" processor, T_{com} is time used on communication per iteration and k is the number of iterations required to reach a satisfactory low error. We have to add one to the iteration-cout because the initialization of the algorithm is almost equal to a full iteration. It's easy to see that if the if the number of iterations required to achieve convergence is halfed on the expense of doubling $T_{\mathcal{G}_{\Delta T}}$, we will still have speedup. This is important to bear in mind when high order schemes are considered for $\mathcal{G}_{\Delta T}$.

3 Odesolvers

This section might not seem so essential in understanding the parareal algorithm. But it turns out that properties presented here are indispensable for the analysis of the stability of the parareal algorithm. Readers that are not familiar with the concepts of A- AN- B- strong A- and L-stability, and which of these properties different implicit Runge-Kutta schemes posses, should study this section carefully in order to understand Section 5 and the results in Section 6. This section is written with support mostly from [8], [9], [22] and [10].

In order to deal with our different test problems, and to test the algorithm for different solvers, we need an arsenal of odesolvers. Since the parareal algorithm in some sence favours single-step methods (not impossible to use multi-step methods), we limit our analysis to only single-step. We use the framework of the famous Runge-Kutta methods to generate our numerical schemes. But first we will present a generel ODE.

$$\begin{aligned} y'(t) &= f(t, y(t)) \\ y(t_0) &= y_0 \end{aligned} \tag{5}$$

We define a general single-step method as

$$y_{n+1} = y_n + h\Phi(t_n, y_n, f, h), \tag{6}$$

where $\Phi(t, y, f, h)$ is a incremental function specified by the method. But how can we be sure that our method gives us the correct solution? The answer lies in the concepts of convergence and stability.

3.1 Stability and Convergence

For a method to be useful, it must be convergent. We must therefor study the convergence of the general problem (5)

Definition 1 A method is said to be *convergent* if, for every ODE with a Lipschitz function f and every $t^* > 0$ it is true that

$$\lim_{h \to 0+} \max_{n=0,\ldots,\lfloor t^*/h \rfloor} \parallel y_{n,h} - y(t_n) \parallel = 0,$$

where $\lfloor \alpha \rfloor \in \mathbb{Z}$ is the integer part of $\alpha \in \mathbb{R}$.

$y(t_n)$ is the except solution, while $y_{n,h}$ is the discretized differential equation solved by the given method, using step size h.

This means that if the timesteps are small enought, a convergent method will give us the correct answer. But the next question that naturaly arise is; how fast can we expect it to converge. A messure for this is the order of the method.

Definition 2 A singlestep method has *order p* if for sufficently smooth problems (5),

$$\parallel y_n - y(t_n) \parallel \leq C h^{p+1},$$

if the Taylor series for the exact solution $y(t_n)$ and for y_n coincide up to and including the term h^p.

But in the parareal algorithm, $\mathcal{F}_{\Delta T}$ (except $\mathcal{F}_{0\Delta T}$) has a starting value that not necessarly is accurate. The question is how this will effect the accuracy of the method.

Theorem 1 *A single-step method is used to approximate (5). If the method is of order p, and the starting value y_0 is accurate of order q, then the numerical solution $\{y_j\}$ is convergent of order* $\min(p, q)$.

As expected the initial value effects the accuracy of the method in a negative way, and the parareal algorithm can't be more accurate then the serial version with same method and stepsize as $\mathcal{F}_{\Delta T}$. This is of course not necessary true for an finite-desimal accurate computer-calculation, but the effect of chop-off error in the parareal algorithm versus serial version is not discussed in this paper.

We now rewrite (6) for a autonomous system (not timedependent) $y' = \lambda y,\ y(t = 0) = y_0$ and get

$$y_{n+1} = y_n + \Phi(y_n, h, \lambda) = R(\lambda h)y_n = R(\lambda h)^{n+1}y_0,$$

where the last step is solved by sucessivly inserting $y_n = R(\lambda h)y_{n-1}$ into y_n etc. It's obvious that if $|R(\lambda h)| > 1$, the solution when $n \to \infty$ will go through the roof. But the coarse operator $\mathcal{G}_{\Delta T}$ will have large stepsize. Let's look at two known schemes to see how the stepsize effects the property $|R(\lambda h)| > 1$. First we'll examine forward euler for our problem.

$$y_{n+1} = y_n + h\lambda y_n = (1 + \lambda h)y_n = R(\lambda h)y_n.$$

Obviously we need $|\lambda h| \leq 1$, which indicates a serious constraint in the choice of h. So what about implicit euler?

$$y_{n+1} = y_n + h\lambda y_{n+1} \quad \Rightarrow \quad y_{n+1} = \left(\frac{1}{1 - h\lambda}\right) y_n = R(\lambda h)y_n.$$

We notice that our restriction now is $|1 - \lambda h| \geq 1$. Considering $\lambda < 0$ (if not the solution is not Lipschitz), we are free to choose the stepsize we want.

Specially for $\mathcal{G}_{\Delta T}$ we are interested in methods that are stable for all stepsizes. It's therefore imperative that we investigate this futher, and defines A-, AN-, B-, L- and orbital stability.

3.1.1 Stability for autonomous systems, A-stability

We want to use a given ode-solver with a constant stepsize $h > 0$ to the scalar linear equation

$$y' = \lambda y, \quad t \geq 0, \quad y(0) = 1 \tag{7}$$

where $\lambda \in \mathbb{C}$. For our general problem (7) we define the stability function $R(z)$.

Stability function

Definition 3 The function $R(z)$ is called the *stability function* of the method. It can be interpreted as the numerical solution after one step for (7) with $z = h\lambda$. The set

$$S = \{z \in \mathbb{C}; |R(z)| \leq 1\}$$

is called the *stability domain* of the method.

We use a Runge-Kutta method (defined in Section 3.2) on (7) and get

$$R(z) = 1 + zb^T(\mathbb{I} - zA)^{-1}\mathbf{1}, \tag{8}$$

where b^T and A is defined in (17), and $\mathbf{1} = (1, \ldots, 1)^T$.

Proposition 6 *The stability function of a given Runge-Kutta scheme satisfies*

$$R(z) = \frac{\det(\mathbb{I} - zA + z\mathbf{1}b^T)}{\det(\mathbb{I} - zA)} \tag{9}$$

Proof: We apply (24) on (7) and gets the linear system

$$\begin{pmatrix} I - zA & 0 \\ -zb^T & 1 \end{pmatrix} \begin{pmatrix} \xi \\ y_{n+1} \end{pmatrix} = y_n \begin{pmatrix} \mathbf{1} \\ 1 \end{pmatrix}.$$

Applying Cramer's rule (Section A.1.1) we get that the denominator of $R(z)$ is $\det(I - zA)$, and its numinator is

$$\det \begin{pmatrix} I - zA + z\mathbf{1}b^T & 0 \\ -zb^T & 1 \end{pmatrix} = \det \begin{pmatrix} I - zA & \mathbf{1} \\ -zb^T & 1 \end{pmatrix} = \det(I - zA + z\mathbf{1}b^T).$$

Notice that the ERK schemes also can be described with (24), so this proof doesn't apply just for the IRK case. □

Definition 4 A method, whose stability domain satisfies

$$\mathbb{C}^- = \{z \in \mathbb{C} : \operatorname{Re} z < 0\} \subset \mathcal{S}$$

is called *A-stable*.

But we need a more practical test for A-stability. From (9) we see that the stability-function can be written as

$$R(z) = \frac{P(z)}{Q(z)}, \quad \deg P = k, \quad \deg Q = j$$

Using the maximum modulus theorem from Theorem 19 we see that $R(z) = \frac{P(z)}{Q(z)}$ must be analytic, which is true if all the poles from $R(z)$ lies in $\mathbb{C}^+$. If this is true, it's enough to check if $R(z) < 1$ on the boundary of $\mathbb{C}^-$, which is the imaginary axis. We now rewrite the condition $|R(z)| \leq 1$ and introduse the E-polynom.

$$\frac{|P(iy)|^2}{|Q(iy)|^2} \leq 1 \quad \Rightarrow \quad E(y) = |Q(iy)|^2 - |P(iy)|^2 = Q(iy)Q(-iy) - P(iy)P(-iy) \geq 0$$

We recapitulate this as

Proposition 7 *Given a numerical scheme with the assosiated stability function $R(z) = \frac{P(z)}{Q(z)}$. The scheme is A-stable if the following is true.*

1. $\operatorname{Re} z_i > 0$, *where z_i is the roots of $Q(z) = 0$, $i = 1, .., j$*
2. $E(y) \geq 0 \quad \forall y \in \mathbb{R}$

3.1.2 Stability for non-autonomous systems, AN-stability

But what if our differential equation is non-autonom, like our general problem

$$y' = \lambda(t)y, \quad \operatorname{Re} \lambda(t) \leq 0 \tag{10}$$

where $\lambda(t)$ is an arbitrary varying complex-valued function. For this problem our implicit Runge-Kutta method (24), with $\xi = (\xi_1, \dots, \xi_s)^T$, $\mathbf{1} = (1, \dots, 1)^T$, can be written as

$$g = \mathbf{1}y_n + AZ\xi, \quad Z = \operatorname{diag}(z_1, \dots, z_s), z_j = h\lambda(t_n + c_j h).$$

We compute ξ and insert into the solution for the next timestep which gives us

$$y_{n+1} = K(Z)y_n, \quad K(Z) = 1 + b^T Z(\mathbb{I} - AZ)^{-1}\mathbf{1}.$$

We are now ready to define AN-stability.

Definition 5 A Runge-Kutta method is called *AN-stabil* if

$$|K(Z)| \leq 1 \left\{ \begin{array}{l} \forall Z = \operatorname{diag}(z_1, \dots, z_s) \text{ satisfying } \operatorname{Re} z_j \leq 0 \\ \text{and } z_j = z_k \text{ whenever } c_j = c_k (j,k = 1, \dots, s) \end{array} \right.$$

We notice that $K(\operatorname{diag}(z, \dots, z)) = R(z)$ with $R(z)$ defined in (9).

3.1.3 Stability for non-linear systems, B-stability

We consider the nonlinear differential equation

$$y' = f(t, y) \tag{11}$$

such that for the Euclidian norm the one-side Lipschitz condition

$$\langle f(t,y) - f(t,z), y - z \rangle \leq \nu \parallel y - z \parallel^2 \tag{12}$$

holds. The number ν is the one-side Lipschitz constant of f. We use this to present the following result

Lemma 1 *Let $f(t,y)$ be continous and satisfy (12). Then, for any two solutions $y(t)$ and $z(t)$ of (11) we have*

$$\parallel y(t) - z(t) \parallel \leq \parallel y(t_0) - z(t_0) \parallel e^{\nu(t-t_0)} \qquad \textit{for } t \geq t_0.$$

Definition 6 A Runge-Kutta method is calles *B-stable* if the contractivity condition

$$\langle f(t,y) - f(t,z), y - z \rangle \leq 0$$

implies $\forall h \geq 0$

$$\parallel y_{n+1} - \hat{y}_{n+1} \parallel \leq \parallel y_n - \hat{y}_n \parallel .$$

Here, y_{n+1} and $\hat{y}_{n+1}$ are the numerical approximations after $n+1$ steps starting with initial values y_0 and $\hat{y}_0$,respectively.

Theorem 2 *If the coefficients of a Runge-Kutta method satisfy*

1. $b_i \geq 0, \quad i = 1, \dots, s$
2. $M = (m_{ij}) = (b_i a_{ij} + b_j a_{ji} - b_i b_j)_{i,j=1}^s$ *is non-negative definite,*

then the method is B-stable.

Theorem 3 *For Runge-Kutta methods it holds*

$$B\textit{-stable} \quad \Rightarrow \quad AN\textit{-stable} \quad \Rightarrow \quad A\textit{-stable}$$

We have now defined stability for several types of equations that we might run into. But we are sadly mistaken if we think that this is all we need. It turns out that for stiff problems we need to define additional properties, the L-stability and *strong* A-stability.

3.1.4 L-Stability and strong A-stability

> A-stability is not the whole answer to the problems of stiff equations.
> (R. Alexander 1977)

We have already established that the growth of a A-stable method will be be bounded. So what is exactly the problem. Again we look at an example. First we consider the implicit midtpoint-method presented in Table 3, with the the stability-function

$$R(z) = \frac{1 + \frac{z}{2}}{1 - \frac{z}{2}}$$

for the general problem (7). Undoubtedly this is an A-stable method. But remember that $z = \lambda h$. If both λ is big (stiff problem), and the stepsize h is large (which is the case for $\mathcal{G}_{\Delta T}$), z will also be large.

$$\lim_{z \to -\infty} R(z) = \lim_{z \to \infty} R(z) = -1.$$

This means that for large z, $R(z)$ will be close to -1. This results in an oscillating function, with very weak damping properties for the large eigenvalues.

So what about our trusted friend, the implicit Euler. We have already established the fact that the stability-function of EB is

$$R(z) = \frac{1}{1 - z}.$$

It's easy to verify that this scheme is A-stable. We do the same test as for implicit midtpoint-method.

$$\lim_{z \to -\infty} R(z) = \lim_{z \to \infty} R(z) = 0.$$

Obviously large eigenvalues are quickly damped out. We demonstrate this with a small example. We calculate $y' = -200(y - \cos(t)), y(t_0) = 0$ using implicit euler and implicit midtpoint. The result is shown in Figure 2. As predicted, implicit midtpoint produces oscillations, which is

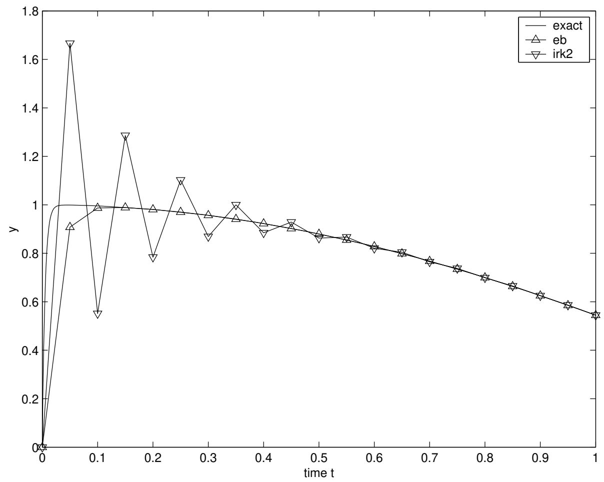


Figure 2: Solution of $y' = -200(y - \cos(t)),\ y(t_0) = 0,\ dt = 0.05$

slowly decreasing, and implict Euler is oscillation-free.

This motivates us to define another property for stability, the L-stability.

Definition 7 (Ehle 1969). A method is called $L-stable$ if it is A-stable and if in addition

$$\lim_{z\to\infty} R(z) = 0$$

We present an easier test, given in [9].

Proposition 8 *If an implicit Runge-Kutta method with nonsingular A satisfies one of the following conditions:*

$$a_s j = b_j \quad j = 1, \ldots, s, l \tag{13}$$
$$a_i 1 = b_j \quad j = 1, \ldots, s, \tag{14}$$

then $R(\infty) = 0$. This makes A-stable methods L-stable.

The proop is quite simple.

Proof: We insert $z = \infty$ into $R(z)$ given in (8) and get

$$R(\infty) = 1 - b^T A^{-1}\mathbf{1}. \tag{15}$$

Writing (13) on matrix form we get $A^T e_s = b$, where $e_s = (0, \ldots, 0, 1)^T$. This is substituted into (15) and we get $R(\infty) = 1 - e_s^T\mathbf{1} = 1 - 1 = 0$. On the other side, (14) is written on matricial form as $Ae_1 = \mathbf{1}b_1$. Now inserting into (15) we get $R(\infty) = 1 - 1 = 0$. □

But L-stability is a very strong property. Does it exist something between L-stability and "just" A-stability? We call it *strong* A-stable, and defines it like this.

Definition 8 If a method has a stability function $R(z)$ with the property

$$\lim_{z\to-\infty} R(z) = a, \quad \text{where } 0 < |a| < 1,$$

then the method is *strong* A-stable.

3.1.5 Orbit-stability

So what is orbit-stability. Again we demonstrate it with an example. We use the differential equation

$$y' = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} y, \quad y(t_0) = \begin{bmatrix} 0 \\ 1 \end{bmatrix},$$

which has the solution $y = \sin t$. We use explicit euler, implicit euler and implicit midpoint, and plot the result in the phaseplane. The result can be seen in Figure 3. We notice the interesting fact that Implicit euler loses energy, but Explicit euler increases energy. Implicit midpoint on the other hand returns to the same state as it started, which it should. This is an interesting observation which we will investigate futher.

We state the following theorem without proof. The interesting reader may refer to [8].

Theorem 4 *If the $s \times s$ matrix M with elements*

$$m_{ij} = b_i a_{ij} + b_j a_{ji} - b_i b_j, \quad i, j = 1, \ldots, s$$

satisfies $M = 0$, then the Runge-Kutta method (24) is symplectic.

We expand this to Runge-Kutta-Nyström, given in Section 3.2.2.

Theorem 5 *The s-stage Runge-Kutta-Nyström given in (23) is symplectic if $\forall i \leq i, j \leq s$*

$$\bar{b}_i = b_i(1 - c_i), \quad b_i(\bar{b}_j - aij) = b_j(\bar{b}_i - a_{ji})$$

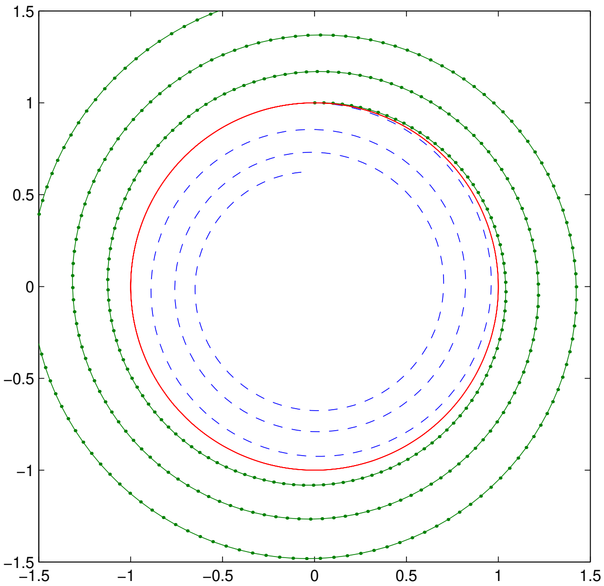


Figure 3: Solution of $y'' = -y,\ y(t_0) = 0,\ y'(t_0) = 1,\ dt = 0.05$, Implicit Euler: $--$, Explicit Euler:$-\bullet-$, Implicit Midpoint: —

3.1.6 Order-Stars

Finally, *order-stars* will be briefly mentioned. It provides us with a powerfull tool to determine order, provided that we already have an expression of the stability-function R(z). So first we need to define order-star.

Definition 9 The set

$$A = \{z \in \mathbb{C}; |R(z)| > |e^z|\} = \{z \in \mathbb{C}; |q(z)| > 1\}$$

where

$$q(z) = \frac{R(z)}{e^z}, \tag{16}$$

is called the *order star* of R.

The following Lemma will be presented without proof, and the interested reader is refered to [8].

Lemma 2 *If $R(z)$ is an approximation to e^z of order p, i.e., if*

$$e^z - R(z) = Cz^{p+1} + \mathcal{O}(z^{p+2})$$

with $C \neq 0$, then, for $z \to 0$, A behaves like a "star" with $p+1$ sectors of equal width $\pi/(p+1)$, separated by $p+1$ similar "white" sectors of the complementary set. The positive real axis is inside a black sector iff $C < 0$ and inside a white sector iff $C > 0$.

3.2 Runge-Kutta methods

We are now done with presenting the mathematical tools for testing the usefulness of our methods. It's now time to present the methods we want to use. A large portion of all the single-step methods can be described within the framework of Runge-Kutta methods. This presentation is therefore limited to various Runge-Kutta schemes. All the schemes are picked from literature, mostly [8] and [9]. For each method we will generate the stability domain, calculated from Proposition 6. We will also present the order star, calculated from (16), mostly as a tool to prove the order of the scheme.

Runge-Kutta methods are based on a powerful procedure known as quadrature. We use Butcher-tablau, defined as

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array} \tag{17}$$

to present the RK weights (b), nodes (c) and coefficients (A).

3.2.1 Explicit RK-methods

We define the famouse *explisit Runge-Kutta (ERK)* as

$$\begin{aligned} \xi_i &= y_n + h\sum_{j=1}^{i-1} a_{i,j} f(t_n + c_i h, \xi_i), \quad j = 1, \ldots, s \\ y_{n+1} &= y_n + h\sum_{i=1}^{s} b_i f(t_n + c_i h, \xi_i) \end{aligned} \tag{18}$$

3.2.2 Runge-Kutta-Nyström methods

A considerable amount of the differential equations in the world are second order.

$$y'' = f(t, y, y') \tag{19}$$

If we want to solve this using a Runge Kutta method (implicit or explicit) we transform the equation into a system of first order equations

$$\begin{pmatrix} y \\ y' \end{pmatrix} = \begin{pmatrix} y' \\ f(t, y, y') \end{pmatrix} \qquad \begin{array}{c} y(t_0) = y_0 \\ y'(t_0) = y_0' \end{array}$$

This yields

$$\begin{aligned} &\xi_i = y_n' + h\sum_{j=1}^{s} a_{ij}\xi_j' \\ &\xi_i' = f\left(t_n + c_i h,\ y_n + h\sum_{j=1}^{s} a_{ij}\xi_j,\ y_n' + h\sum_{j=1}^{s} a_{ij}\xi_j'\right) \\ &y_{n+1} = y_n + h\sum_{i=1}^{s} b_i\xi_i, \quad y_{n+1}' = y_n' + h\sum_{i=1}^{s} b_i\xi_i' \end{aligned}$$

We now insert ξ_i into ξ_i' and get

$$\begin{aligned} &\xi_i' = f\left(t_n + c_i h,\ y_n + c_i h y_n' + h^2\sum_{j=1}^{s} \bar{a}_{ij}\xi_j',\ y_n' + h\sum_{j=1}^{s} a_{ij}\xi_j'\right) \\ &y_{n+1} = y_n + h y_n' + h^2\sum_{i=1}^{s} \bar{b}_i\xi_i', \quad y_{n+1}' = y_n' + h\sum_{i=1}^{s} b_i\xi_i' \end{aligned} \tag{20}$$

where

$$\bar{a}_{ij} = \sum_{k=1}^{s} a_{ik}a_{kj}, \quad \bar{b}_i = \sum_{j=1}^{s} b_j a_{ji} \tag{21}$$

A Nyström method satisfy (20), but not necessarily (21).

Definition 10 A Nyström method (20) has *order* p if for sufficiently smooth problems (19)

$$y(t_n + h) - y_n = \mathcal{O}(h^{p+1}), \quad y'(t_n + h) - y'_n = \mathcal{O}(h^{p+1})$$

A real improvement can be achieved in the case when the right-hand side of (19) does not depend on y', i.e.,

$$y'' = f(t, y)$$

It is obvious that (20) can be shortened to

$$\xi'_i = f\left(t_n + c_i h,\ y'_n + h\sum_{j=1}^{s} a_{ij}\xi'_j\right) \tag{22}$$

$$y_{n+1} = y_n + h\sum_{i=1}^{s} b_i \xi_i, \quad y'_{n+1} = y'_n + h\sum_{i=1}^{s} b_i \xi'_i \tag{23}$$

Theorem 6 *The Nyström scheme given in Table 2 is convergent of 5. order*

Proof: This can be proven using tree-structures. But since we then have to introduce a new theory, which will give us no additional knowledge of the parareal algorithm, we leave this out. The interested reader is refeared to [8]. □

0			$\bar{a}_{ij}$	
$\frac{1}{5}$	$\frac{1}{50}$			
$\frac{2}{3}$	$-\frac{1}{27}$	$\frac{7}{27}$		
1	$\frac{3}{10}$	$-\frac{2}{35}$	$\frac{9}{35}$	
$\bar{b}_i$	$\frac{14}{336}$	$\frac{100}{336}$	$\frac{54}{336}$	0
b_i	$\frac{14}{336}$	$\frac{125}{336}$	$\frac{162}{336}$	$\frac{35}{336}$

Table 2: 5. order Nyström method methods for $y'' = f(t, y)$

3.2.3 Implicit RK-methods

The *implicit runge-Kutta (IRK)* is given by

$$\begin{aligned} \xi_i &= y_n + h\sum_{j=1}^{s} a_{i,j} f(t_n + c_j h, \xi_j), \quad i = 1, \ldots, s \qquad (24) \\ y_{n+1} &= y_n + h\sum_{i=1}^{s} b_i f(t_n + c_i h, \xi_i) \end{aligned}$$

There exists a variety of different IRK schemes. We will focus on the schemes that are suitable for solution of stiff differential equations.

For all the schemes we will present a plot of the stability-domain, and a plot of the order-star. The stability domain gives a good indication on A-stability and strong A-stability properties of the scheme. The order star is used to find the order of the scheme. All the poles of the stability function are plotted as $*$. For the stability-domain plot, the areas covered with contours are the domain for which values of z gives a stable scheme. For the order star, all the areas covered with contours are "white" sectors, while areas without concours (some with poles) are "black" sectors.

We start with collocation methods based on Gaussian quadrature formulas, i.e., $c_1, \ldots, c_s$ are the zeros of the shifted Legendre polynomial of degree s,

$$\frac{d^s}{dx^s}\left(x^s(x-1)^s\right).$$

It can be proven that all IRK schemes based on this quadrature have order $2s$. To give a clear name to each scheme we will present, we name all the IRK schemes based on Gauss quadrature GaussX, where the X is the order of the method

1 stage 2nd order IRK based on Gauss, Gauss2 We start lightly with $s = 1$

$$y_{n+1} = y_n + hf(t_n + \frac{1}{2}h, \frac{1}{2}(y_n + y_{n+1})), \quad n = 0, 1, \ldots \tag{25}$$

$$\begin{array}{c|c} \frac{1}{2} & \frac{1}{2} \\ \hline & 1 \end{array}$$

Table 3: IRK of order 2

Theorem 7 *Implicit midpoint method is convergent of order 2, BN-stabil and symplectic.*

Proof: We start with order. In Figure 4(b) we count 3 white sectors and 3 black sectors, which gives us according to Lemma 2, order 2. We use Theorem 2 and verifies that $b_1 > 0$ and $M = ba + ba - b^2 = 0$. This also satisfies Theorem 4. $\square$

We now push on and construct higher order Implicit Runge-Kutta schemes using Gauss-Legendre quadrature

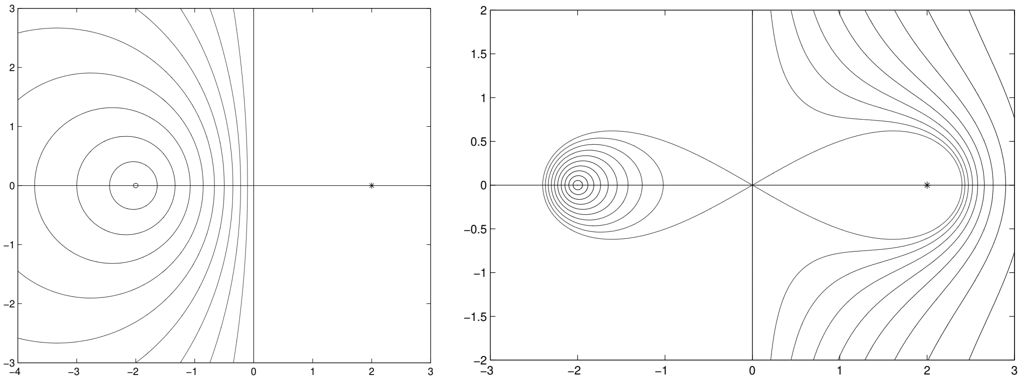
$$\begin{array}{c|cc} \frac{1}{2} - \frac{\sqrt{3}}{6} & \frac{1}{4} & \frac{1}{4} - \frac{\sqrt{3}}{6} \\ \frac{1}{2} + \frac{\sqrt{3}}{6} & \frac{1}{4} + \frac{\sqrt{3}}{6} & \frac{1}{4} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

Table 4: IRK of order 4

2 stage 4th order IRK based on Gauss, Gauss4

Theorem 8 *The 2 stage IRK scheme Table 4 is convergent of order 4, BN-stabil and symplectic.*

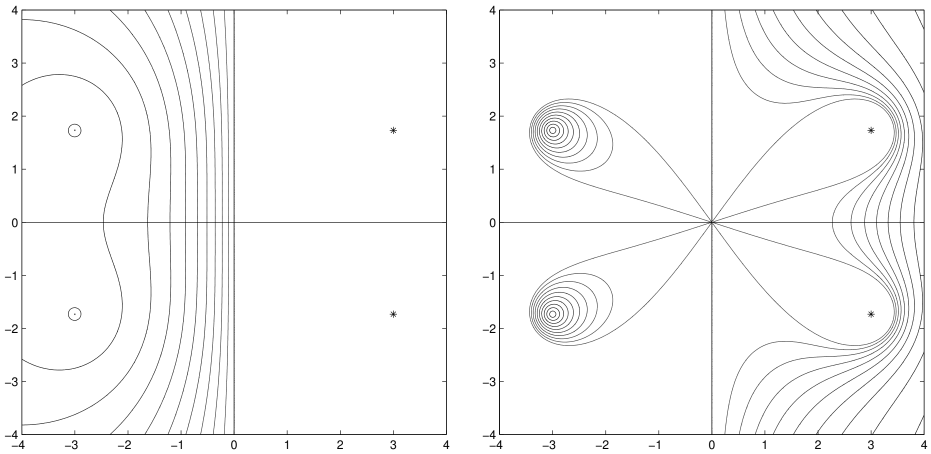
Proof: Again from Figure 5(b) we count 5 white and 5 black sectors, proving order 4. Also $b_i > 0$ and $(M)_{ij} = 0$ for $i, j = 1, 2$, proving BN-stability and symplectic behavior. $\square$



(a) Stability domain with $R(z) = \frac{1+\frac{z}{2}}{1-\frac{z}{2}}$

(b) Order Star

Figure 4: Implicit midpoint scheme (IRK2) given in (25)



(a) Stability domain with $R(z) = \frac{1+\frac{z}{2}+\frac{z^2}{12}}{1-\frac{z}{2}+\frac{z^2}{12}}$

(b) Order Star

Figure 5: 2 stage 4. order IRC given in Table 4

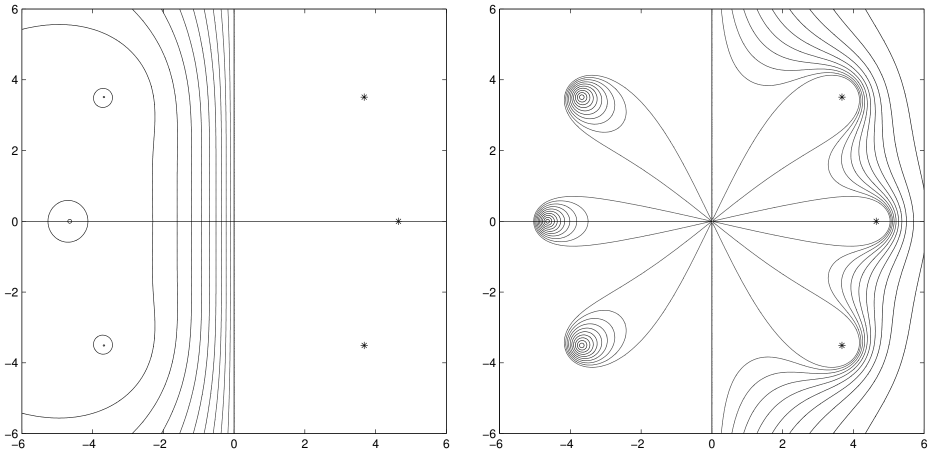
$\frac{1}{2}-\frac{\sqrt{15}}{10}$	$\frac{5}{36}$	$\frac{2}{9}-\frac{\sqrt{15}}{15}$	$\frac{5}{36}-\frac{\sqrt{15}}{30}$
$\frac{1}{2}$	$\frac{5}{36}+\frac{\sqrt{15}}{24}$	$\frac{2}{9}$	$\frac{5}{36}-\frac{\sqrt{15}}{24}$
$\frac{1}{2}+\frac{\sqrt{15}}{10}$	$\frac{5}{36}+\frac{\sqrt{15}}{30}$	$\frac{2}{9}+\frac{\sqrt{15}}{15}$	$\frac{5}{36}$
	$\frac{5}{18}$	$\frac{4}{9}$	$\frac{5}{18}$

Table 5: IRK of order 6

3 stage 6th order IRK based on Gauss, Gauss6

Theorem 9 *The 3 stage IRK scheme Table 5 is convergent of order 6, BN-stabil and symplectic.*

Proof: Again from Figure 6(b) we count 7 white and 7 black sectors, proving order 6. Also $b_i > 0$ and $(M)_{ij} = 0$ for $i, j = 1 : 3$, proving BN-stability and symplectic behavior. □



(a) Stability domain with $R(z) = \frac{1+\frac{z}{2}+\frac{z^2}{10}+\frac{z^3}{120}}{1-\frac{z}{2}+\frac{z^2}{10}-\frac{z^3}{120}}$

(b) Order Star

Figure 6: 3 stage 6. order IRC given in irk6

4 stage 8th order IRK based on Gauss, Gauss8

Theorem 10 *The 4 stage IRK scheme Table 6 is convergent or order 8, BN-stabil and symplectic.*

Proof: Again from Figure 7(b) we count 9 white and 9 black sectors, proving order 8. Also $b_i > 0$ and $(M)_{ij} = 0$ for $i, j = 1 : 4$, proving BN-stability and symplectic behavior. □

It's also possible to construct IRK schemes based on Radau quadrature formulas. We call them type I or II according to whether $c_1, \ldots, c_s$ are zeros of

$$I: \quad \frac{d^{s-1}}{dx^{s-1}}\left(x^s(x-1)^{s-1}\right), \quad \text{Radau left} \tag{26}$$

$$II: \quad \frac{d^{s-1}}{dx^{s-1}}\left(x^{s-1}(x-1)^s\right), \quad \text{Radau right.} \tag{27}$$

$$
\begin{array}{c|cccc}
\frac{1}{2}-\omega_2 & \omega_1 & \omega_1'-\omega_3+\omega_4' & \omega_1'-\omega_3-\omega_4' & \omega_1-\omega_5 \\
\frac{1}{2}-\omega_2' & \omega_1-\omega_3'+\omega_4 & \omega_1'\omega_1'--\omega_5' & \omega_1-\omega_3'-\omega_4 & \\
\frac{1}{2}+\omega_2' & \omega_1+\omega_3'+\omega_4 & \omega_1'+\omega_5' & \omega_1' & \omega_1+\omega_3'-\omega_4 \\
\frac{1}{2}+\omega_2 & \omega_1+\omega_5 & \omega_1'+\omega_3+\omega_4' & \omega_1'+\omega_3-\omega_4' & \omega_1 \\
\hline
 & 2\omega_1 & 2\omega_1' & 2\omega_1' & 2\omega_1
\end{array}
$$

$$
\begin{array}{ll}
\omega_1 = \frac{1}{8} - \frac{\sqrt{30}}{144}, & \omega_1' = \frac{1}{8} + \frac{\sqrt{30}}{144}, \\
\omega_2 = \frac{1}{8} sqrt \frac{15+2\sqrt{30}}{35}, & \omega_2' = \frac{1}{8} sqrt \frac{15-2\sqrt{30}}{35}, \\
\omega_3 = \omega_2 \left(\frac{1}{6} + \frac{\sqrt{30}}{24} \right), & \omega_3' = \omega_2' \left(\frac{1}{6} - \frac{\sqrt{30}}{24} \right), \\
\omega_4 = \omega_2 \left(\frac{1}{21} + \frac{5\sqrt{30}}{168} \right), & \omega_4' = \omega_2' \left(\frac{1}{21} - \frac{5\sqrt{30}}{168} \right), \\
\omega_5 = \omega_2 - 2\omega_3, & \omega_5' = \omega_2' - 2\omega_3'
\end{array}
$$

Table 6: IRK of order 8

We will use (27), called RadauIIA methods. As a consequence $c_s = 1$. It's possible to show that all IRK schemes based on Radau quadrature has order $2s-1$. This is lower then schemes based on Gauss-Legendre. But Radau schemes has other properties, which will be usefull later. To give a clear name to each scheme we will present, we name all the IRK schemes based on Radau quadrature RadauX, where the X is the order of the method

1 stage 1st order IRK based on Radau (Implicit Euler), EB We start with $s = 1$ and discover an old friend, the *implicit euler*. Instead of calling it Radau1, we use EB (Euler Backward) since this is the name most people are used to.

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1}), \quad n = 0, 1, \ldots \tag{28}$$

$$
\begin{array}{c|c}
1 & 1 \\
\hline
 & 1
\end{array}
$$

Table 7: IRK of order 1, Implicit Euler

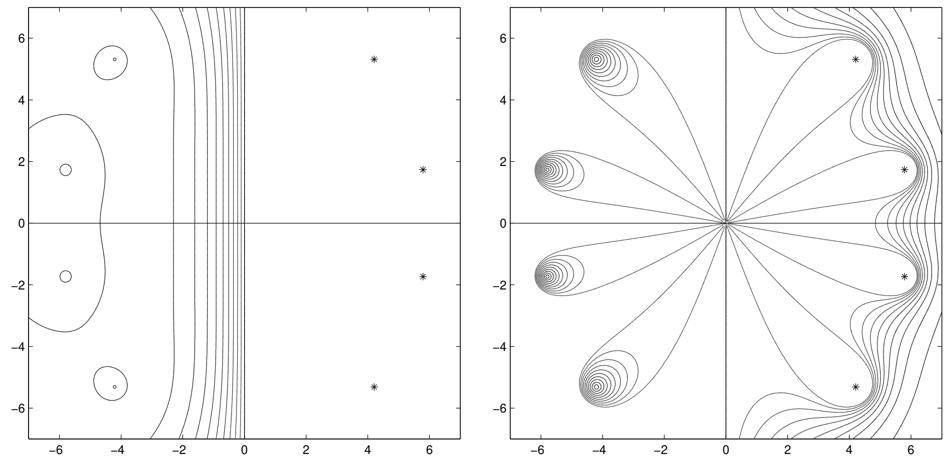
Theorem 11 *Implicit Euler is convergent of order 1, BN- and L-stable.*

Proof: Again from Figure 8(b) we count 2 white and 2 black sectors, proving order 1. Also $b > 0$ and $M > 0$, proving BN-stability. We know from Theorem 3 that a B-stable Runge-Kutta method also is A-stable. From Proposition 8 we verify that $a = b$, and we have a L-stable *stiffly-accurate* method. □

We push on for higher order

2 stage 3rd order IRK based on Radau, Radau3

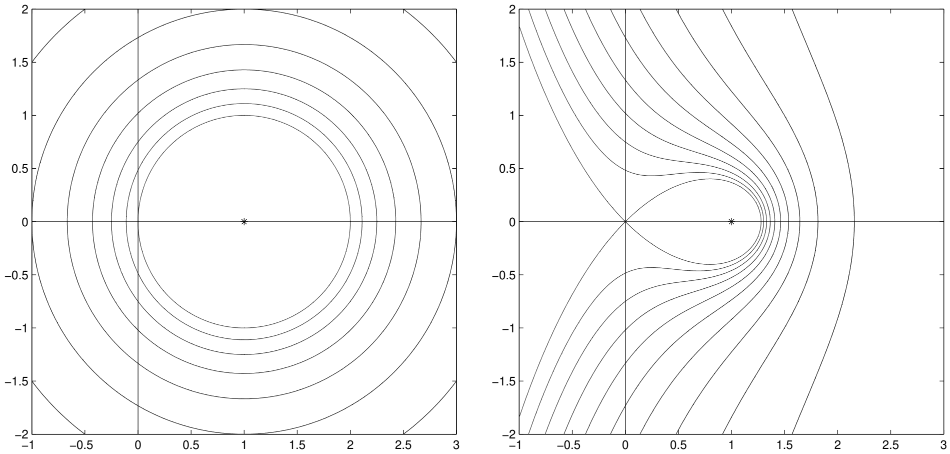
Theorem 12 *The IRK scheme based on Table 8 is convergent of order 3, B- and L-stable.*



(a) Stability domain with $R(z) = \frac{1+\frac{1}{2}z+\frac{3}{28}z^2+\frac{1}{84}z^3+\frac{1}{1680}z^4}{1-\frac{1}{2}z+\frac{3}{28}z^2-\frac{1}{84}z^3+\frac{1}{1680}z^4}$

(b) Order Star

Figure 7: 4 stage 8. order IRC given in Table 6



(a) Stability domain with $R(z) = \frac{1}{1-z}$

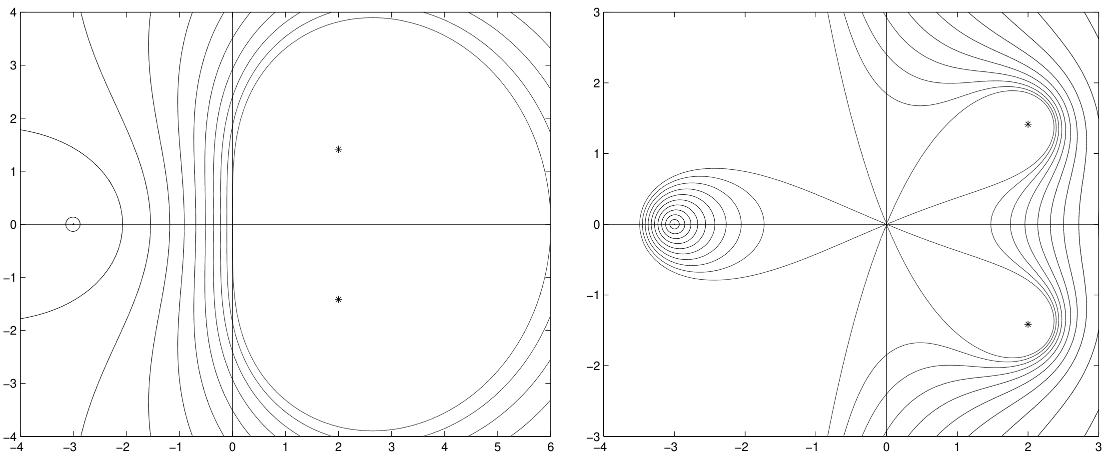
(b) Order Star

Figure 8: Implicit Euler given in (28)

$$
\begin{array}{c|cc}
\frac{1}{3} & \frac{5}{12} & -\frac{1}{12} \\
1 & \frac{3}{4} & \frac{1}{4} \\
\hline
 & \frac{3}{4} & \frac{1}{4}
\end{array}
$$

Table 8: IRK of order 3

Proof: Again from Figure 9(b) we count 4 white and 4 black sectors, proving order 3. Also $b_i > 0$ and $(M)_{ij} \geq 0$ for $i, j = 1, 2$, proving BN-stability. From Proposition 8 we verify that $a_{2j} = b_j$, for $j = 1, 2$, and we have a L-stable *stiffly-accurate* method. $\square$



(a) Stability domain with $R(z) = \frac{1+\frac{1}{3}z}{1-\frac{2}{3}z+\frac{1}{6}z^2}$ **(b)** Order Star

Figure 9: 3 order IRK given in Table 8

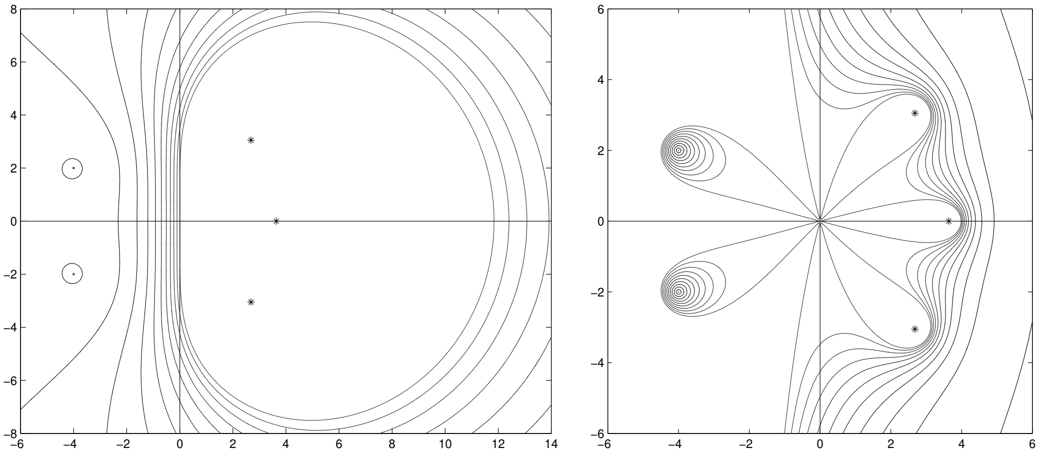
$$
\begin{array}{c|ccc}
\frac{4-\sqrt{6}}{10} & \frac{88-7\sqrt{6}}{360} & \frac{296-169\sqrt{6}}{1800} & \frac{-2+3\sqrt{6}}{225} \\
\frac{4+\sqrt{6}}{10} & \frac{296+169\sqrt{6}}{1800} & \frac{88+7\sqrt{6}}{360} & \frac{-2-3\sqrt{6}}{225} \\
1 & \frac{16-\sqrt{6}}{36} & \frac{16+\sqrt{6}}{36} & \frac{1}{9} \\
\hline
 & \frac{16-\sqrt{6}}{36} & \frac{16+\sqrt{6}}{36} & \frac{1}{9}
\end{array}
$$

Table 9: IRK of order 5

3 stage 5th order IRK based on Radau, Radau5

Theorem 13 *The IRK scheme based on Table 9 is convergent of order 5, B- and L-stable.*

Proof: Again from Figure 10(b) we count 6 white and 6 black sectors, proving order 5. Also $b_i > 0$ and $(M)_{ij} \geq 0$ for $i, j = 1 : 3$, proving BN-stability. From Proposition 8 we verify that $a_{3j} = b_j$, for $j = 1, 2, 3$, and we have a L-stable *stiffly-accurate* method. $\square$



(a) Stability domain with $R(z) = \frac{1+\frac{2}{5}z+\frac{1}{20}z^2}{1-\frac{3}{5}z+\frac{3}{20}z^2-\frac{1}{60}z^3}$ **(b)** Order Star

Figure 10: 5 order IRK given in Table 9

Another well known class or IRK which is known as good "stiff"-solvers is the *Single Diagonal Implicit RK method* (SDIRK). We will only present one scheme from this class.

$$
\begin{array}{c|cc}
\gamma & \gamma & 0 \\
1-\gamma & 1-2\gamma & \gamma \\
\hline
 & \frac{1}{2} & \frac{1}{2}
\end{array}
$$

Table 10: SDIRK method of order $s \geq 2$, which is the Crouzeix-Nørset scheme.

3rd order SDIRK, SDIRK3

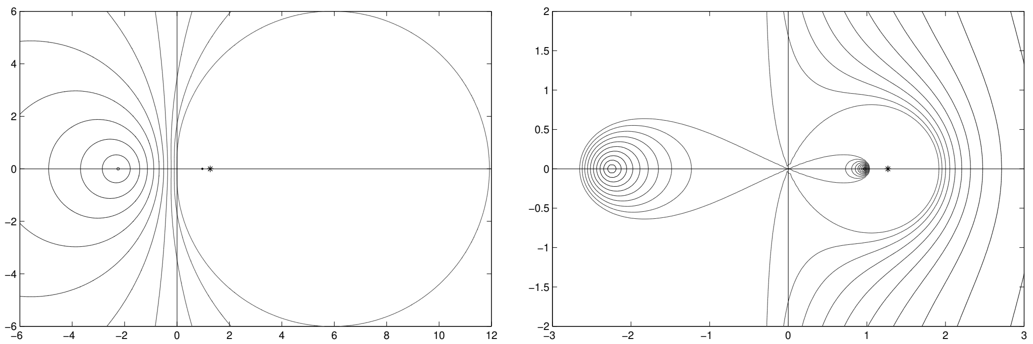
Theorem 14 *For $\gamma = \frac{3+\sqrt{3}}{6}$ the Crouzeix-Nørset scheme is convergent of order 3 and BN-stabil. It is also strong A-stable, with $\lim_{z\to-\infty} R(z) = 1-\sqrt{3}$.*

Proof: Again from Figure 11(b) we count 4 white and 4 black sectors, proving order 3. Also $b_i > 0$ and $(M)_{ij} > 0$ for $i, j = 1, 2$, proving BN-stability. By using (9) and Table 10, we find that

$$
\begin{aligned}
R(z) &= \frac{1-\frac{\sqrt{3}}{3}z-\frac{1}{6}\left(1+\sqrt{3}\right)z^2}{1-\left(1+\frac{\sqrt{3}}{3}\right)z+\frac{1}{6}\left(2+\sqrt{3}\right)z^2} \\
\lim_{z\to-\infty} R(z) &= \frac{-\frac{1}{6}\left(1+\sqrt{3}\right)}{\frac{1}{6}\left(2+\sqrt{3}\right)} = 1-\sqrt{3} \approx -0.73
\end{aligned}
$$

□

The Theta-method It will become clear in Section 5 that we will need a method where we can vary the strong A-stability propertie. We will therefore present such a method here – the



(a) Stability domain with $R(z) = \frac{1-\frac{\sqrt{3}}{3}z-\frac{1}{6}(1+\sqrt{3})z^2}{1-\left(1+\frac{\sqrt{3}}{3}\right)z+\frac{1}{6}(2+\sqrt{3})z^2}$

(b) Order Star

Figure 11: 3 order SDIRK given in Table 8

Theta method.

$$y_{n+1} = y_n + h\left(\theta f(t_n, y_n) + (1-\theta)f(t_{n+1}, y_{n+1})\right). \tag{29}$$

The stability-function is, for the autonomous differential equation (7), found to be

$$R(z) = \frac{1+\theta z}{1-(1-\theta)z}, \quad z = \lambda h. \tag{30}$$

The Theta methods properties can be systemized as

$\theta = 1$: We recive the explicit Euler (the only explicit method in the Theta method).

$1 < \theta < 1/2$: We recive an implicit, 1st order non A-stable method.

$\theta = 1/2$: We recive the Trapezoid method, which is the only 2nd order method in the Theta method.

$1/2 < \theta < 0$: We recive an implicit 1st order A-stable method with increasing strong A-stable property as θ becomes smaller.

$\theta = 0$: We recive the implicit Euler, which is the only L-stable method in the Theta method.

For a closer analysis of the Theta method, the interested reader i referred to [10].

3.3 Solving nonlinear Equations

When solving nonlinear equations using implicit methods, the ξ vector is not solvable using linear algebra. We have to apply a nonlinear solver. The literature recommends Newton iteration, and specialy modified Newton. We will here present ordinary Newton for system.

We start by writing our system of equations on the form

$$F = \begin{array}{c} f_1(x_1, \ldots, x_n) = 0 \\ \vdots \\ f_n(x_1, \ldots, x_n) = 0 \end{array}$$

A commen notation of Newton'n method is then

$$J(x^k)\Delta x^k = -F(x^k)$$
$$x^{k+1} = x^k + \Delta x^k$$

where

$$J(x^k) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \vdots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

But as for the ode-solvers, we are interested in convergence and order. We will present to importent results, without proof. The interested reader is refered to [15].

Theorem 15 *Let f'' be continous and let r be a simple zero of f. Then there is a neighborhood of r and a constant C such that if Newton's method is started in that neighborhood, the successive points become steadily closer to r and satisfy*

$$|x_{n+1} - r| \leq C(x_n - r)^2, \quad (n \geq 0).$$

In some situations Newton's iteration can be guaranteed to converge from an arbitrary starting point.

Theorem 16 *If f belongs to $C^2(\mathbb{R})$, is increasing, is convex, and has a zero, then the zero is unique, and Newton iteration will converge to it from any starting point.*

In multi-dimentional spaces this becomes more complicated. But the fact that Newton is not guarranteed to converge for all equations and all startingpoints remains. This will later become a problem in the implementation and caluclation of some nonlinear equations.

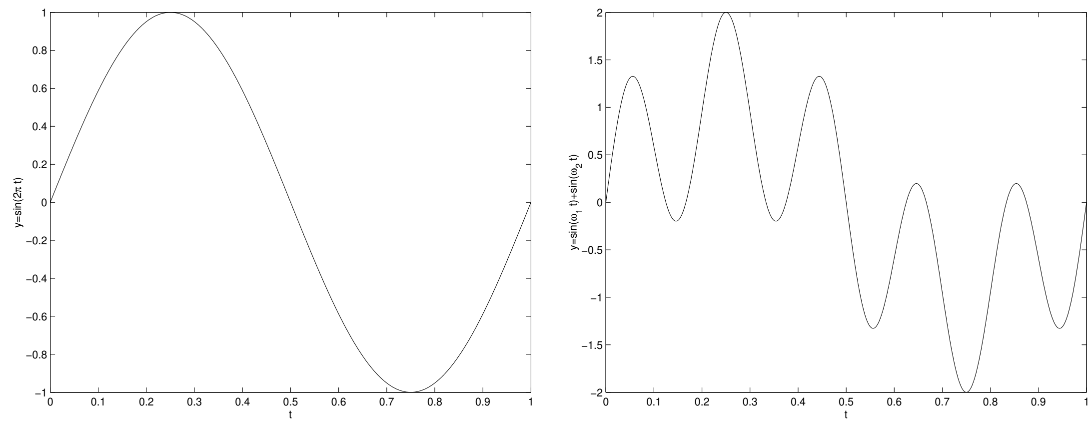
4 Test problems

4.1 Periodic equation

We want to test our algorithm on a periodic solution to see what kind of limitations this imposes on the coarse iteration. We choose the ODE for a simple sine function

$$\begin{aligned} y'' + \omega^2 y &= 0 \\ y(t_0) = 0, &\quad y'(t_0) = \omega \end{aligned} \tag{31}$$

(31) has the exact solution $y(t) = \sin(\omega t)$. A plot with $\omega = 2\pi$ can be seen in Figure 12(a). Most



(a) Plot of (31) with $\omega = 2\pi$

(b) Plot of (33) with $\omega_1 = 2\pi$ and $\omega_2 = 5\omega_1$

Figure 12:

of our solvers handles just first-order differential equations. So we have to write (31) as a system of first-order equations.

$$y' = \begin{bmatrix} 0 & \omega \\ -\omega & 0 \end{bmatrix} y, \quad \text{where } y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \tag{32}$$

By derivating y_1' once, and then substituting for the expression if y_2' we regain (31). It's easy to verify that (31) is an Hamiltonian system (see Section A.3).

For further flexibility we introduce a second sine ODE, namely

$$\begin{aligned} y'' + \omega_1^2 y + (\omega_2^2 - \omega_1^2) \sin(\omega_2 t) &= 0 \\ y(t_0) = 0, \quad y'(t_0) &= \omega_1 + \omega_2 \end{aligned} \tag{33}$$

(33) has the exact solution $y(t) = \sin(\omega_1 t) + \sin(\omega_2 t)$. A plot of (33) with $\omega_1 = 2\pi$ and $\omega_2 = 5\omega_2$ can be seen in Figure 12(b). This is also converted to a system of first order equations

$$y' = \begin{bmatrix} 0 & \omega_1 \\ -\omega_1 & 0 \end{bmatrix} y - \begin{bmatrix} 0 \\ 1 \end{bmatrix} (\omega_2^2 - \omega_1^2) \sin \omega_2 t \tag{34}$$

Notice that we now have an ODE of the form

$$\frac{\partial y}{\partial t} + Ay = F(t)$$

where $F(t)$ is a source term.

4.1.1 Analysis of the algorithm on a differential equation

We consider the parareal algorithm for the simple ODE

$$\begin{cases} \frac{\partial u}{\partial t} + \lambda u = 0 \\ u(t_0) = u_0 \end{cases} \tag{35}$$

In [2] this ODE is analysed. We will here only present the essence of that analysis. The interested reader is refered to the article.

Proposition 9 *Let $u(t)$ be the solution of (35) and U_k^n the solution of the iterative scheme parareal, where $\mathcal{G}_{\Delta T}$ is a coarse scheme of order $m \geq 1$. Then the error terms ϵ_k^n satisfy the following estimate*

$$|\epsilon_k^n| \leq C_k n^{k+1} \delta^{(m+1)(k+1)}.$$

In particular, we have $U_k^N = u(T) + \mathcal{O}(\delta^{m(k+1)})$ and $y_k^n(t) = u(t) + \mathcal{O}(n^k \delta^{(m+1)k})$, where y_k^n is from the fine solver.

4.2 Parabolic PDE

We want to test the algorithm on a PDE, and we choose the parabolic PDE

$$\begin{aligned} &\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \\ &u(0,t) = u(1,t) = 0 \ \forall t, \quad u(x,0) = \sin(\pi x), \quad x \in (0,1) \end{aligned} \tag{36}$$

(36) can easily be solved exact using separation of variables.

$$u(x,t) = \sin(\pi x)\, e^{-\pi^2 t}$$

A plot of the solution of (36) can be seen in Figure 13(a).

Also here we introduce a second PDE with a source term.

$$\begin{aligned} \frac{\partial u}{\partial t} &= \frac{\partial^2 u}{\partial x^2} + \omega_x^2 u + \omega_t \sin \omega_x x \cos \omega_t t = P(u) + F(x,t), \\ &u(0,t) = u(1,t) = 0 \ \forall t, \quad u(x,0) = 0, \quad x \in (0,1), \end{aligned} \tag{37}$$

where P is the operator $P = \frac{\partial^2}{\partial x^2} + \omega_x^2$, and $F(x,t) = \omega_t \sin \omega_x x \cos \omega_t t$. It can be shown, however not as easy as for (36), that the solution of (37) is $u(x,t) = \sin \omega_x x \sin \omega_t t$. A plot of the solution of (37) can be seen in Figure 13(b).

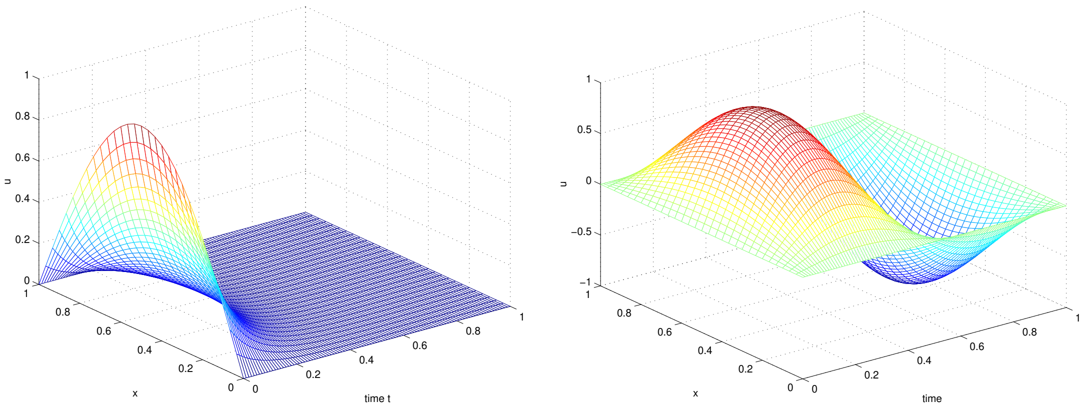
4.2.1 FEM solution of the heat equation

The semi-discrete formulation is achieved using finite element method. First we define the function spaces

$$\begin{aligned} X &= H_0^1(\Omega) = \{v \in H^1(\Omega) | v(0) = v(1) = 0\} \\ Y(X) &= \{v | \forall t \in [0,T], v(x,t) \in X, \int_0^T \| v \|_{H^1(\Omega)}^2 \, dt < \infty\}. \end{aligned}$$

The weak form can then be expressed as: Find $u \in Y(X)$ such that

$$\frac{d}{dt}(u,v) = -a(u,v) \ \forall v \in X,$$



(a) Solution of (36) **(b)** Solution of (37) with $\omega_x = \pi$ and $\omega_t = 2\pi$

Figure 13:

where

$$
\begin{aligned}
\forall w, v \in X, \quad a(w,v) &= \int_0^1 w_x v_x dx \\
\forall w, v \in X, \quad (w,v) &= \int_0^1 wv dx
\end{aligned}
$$

Here are $a(w,v)$ and (w,v) both symmetric, positive-definite bilinear forms.

We now obtain the semi-discrete formulation: Find $u_h \in Y(X_h)$ such that

$$\frac{d}{dt}(u_h, v) = -a(u_h, v) \ \forall v \in X_h,$$

with $X_h \subset X$ and $\dim(X_h) = N < \infty$. We base our discretization on linear elements, and express X_h as

$$
\begin{aligned}
X_h &= \{v \in X = H_0^1(\Omega) | v|_{T_h^k} \in \mathbb{P}_1(T_h^k), k = 1, \ldots, K\} \\
&= span\{\phi_1, \ldots, \phi_N\}.
\end{aligned}
$$

From this we find our nodal basis for X_h, that is

$$\forall v \in X_h, \quad v(x) = \sum_{i=1}^N v_i \phi_i(x), \qquad u_h(x_h, t) = \sum_{j=1}^N u_{hj} \phi_j(x). \tag{38}$$

We end up writing the semidiscrete form of (36) as

$$
\begin{aligned}
M_h \frac{du_h}{dt} &= -A_h u_h \\
(M_h)_{ij} &= \int_0^1 \phi_i \phi_j dx, \\
(A_h)_{ij} &= \int_0^1 \frac{d\phi_i}{dx} \frac{d\phi_j}{dx} dx \\
u_h &= [u_{h1}, \ldots, u_{hN}]
\end{aligned}
$$

M_h is called mass matrix, while A_h is the stiffness matix. We notice that both M_h and A_h are symmetric positive definite.

For (37), the solution is found in a similar way. The weak form can then be expressed as: Find $u \in Y(X)$ such that

$$\frac{d}{dt}(u,v) = a(u,v) + l(v) \ \forall v \in X,$$

where

$$\begin{aligned}
\forall w, v \in X, \quad a(w,v) &= -\int_0^1 w_x v_x dx + \int_0^1 wv dx \\
\forall w, v \in X, \quad (w,v) &= \int_0^1 wv dx \\
\forall v \in X, \quad l(v) &= \int_0^1 v f(x,t) dx
\end{aligned}$$

We now define the semidiscrete form of (37), using the nodal basis, as

$$\begin{aligned}
M_h \frac{du_h}{dt} &= -A_h u_h + \omega_x^2 M_h u_h + F_h \\
(F_h)_i &= \int_0^1 \phi_i f(x,t) dx, \\
(M_h)_{ij} &= \int_0^1 \phi_i \phi_j dx, \\
(A_h)_{ij} &= \int_0^1 \frac{d\phi_i}{dx} \frac{d\phi_j}{dx} dx \\
u_h &= [u_{h1}, \ldots, u_{hN}]
\end{aligned}$$

4.2.2 Estimate of error between serial and parallel calculation

We now have a PDE, and needs to calculate the error both in time and space. We do this by developing an expression for the $H^1(\Omega)$ norm for our problem (36). Since the $H^1(\Omega)$ space is defined as

$$H^1(\Omega) \equiv \left\{ v | \int_\Omega v^2 dA < \infty, \ \int_\Omega v_x^2 dA < \infty \right\}$$

the assosiated norm is defined as

$$\| v \|_{H^1(\Omega)} = \left(\int_\Omega |\nabla v|^2 + v^2 dA \right)^{\frac{1}{2}} \tag{39}$$

We use (38) and, $\forall v \in X_h$, write (39) as

$$\begin{aligned}
\| v \|^2_{H^1(\Omega)} &= \int_\Omega \left[\left(\sum_{i=1}^N (v_h)_i(t) \phi_i'(x) \right) \left(\sum_{j=1}^N (v_h)_j(t) \phi_j'(x) \right) + \left(\sum_{i=1}^N (v_h)_i(t) \phi_i(x) \right) \left(\sum_{j=1}^N (v_h)_j(t) \phi_j(x) \right) dx \right] \\
&= \sum_{i=1}^N \sum_{j=1}^N \left[(v_h)_i \left(\int_\Omega \phi_i' \phi_j' dx \right) (v_h)_j + (v_h)_i \left(\int_\Omega \phi_i \phi_j dx \right) (v_h)_j \right] \\
&= \sum_{i=1}^N \sum_{j=1}^N [(v_h)_i (A_h)_{ij} (v_h)_j + (v_h)_i (M_h)_{ij} (v_h)_j] \\
&= v_h^T A_h v_h + v_h^T M_h v_h
\end{aligned} \tag{40}$$

Since we operate in $\mathbb{R}^1$, we use exact quadrature and the norm is exact.

We have now an expression for the error in space. In time we would like to use the $L^2(\Omega)$ norm, as we will do for all the other test problems. It is defined as

$$\| v \|_{L^2(\Omega)} = \left(\int_\Omega |v|^2 dA \right)^{\frac{1}{2}} \tag{41}$$

We combine (40) and (41) for our problem (36). The error vector is defined as

$$e_n^k = y_n^s - (y_n^p)^k,$$

where n indicates the timestep, k the iteration in the algorithm and s and p stands for *serial* versus *parallel*. We then estimate the error of our algorithm in iteration k to be

$$err^k = \left(\int_0^T \| e_n^k \|_{H^1(\Omega)}^2 \, dt \right)^{\frac{1}{2}} \approx \left(\sum_{k=0}^{N_{\delta t}} \left((e_n^k)^T A_h e_n^k + (e_n^k)^T M_h e_n^k \right) \delta t \right)^{\frac{1}{2}}$$

Notice that we no longer have an exact expression for the error since the time discretization isn't exact.

4.2.3 Analysis of the Heat-equation

We write (36) on the form

$$\begin{cases} \frac{\partial u}{\partial t} + P(u) = 0 \\ u(t_0, x) = u_0(x), \end{cases} \tag{42}$$

where $P(u)$ is a pseudo-differential operator

$$P(u) = \widehat{P(\xi)\hat{u}(\xi)},$$

with symbol $P(\xi) \geq 0$. In our case $P(u) = -\frac{\partial^2 u}{\partial x^2}$ and $P(\xi) = \xi^2$.

In [2] this PDE is analysed. We will here only present the essence of that analysis.

Proposition 10 *Let $u(t,x)$ be the solution of (42) and $U_k^n(x)$ the solution of the parareal scheme with implicit Euler coarse discretization. Then we have the following estimate*

$$\| u(T,x) - U_k^n(x) \|_{L^2(\mathbb{R})} \leq \left(\frac{C(k+1)}{n} \right)^{k+1} \| u_0 \|_{L^2(\mathbb{R})} .$$

This proved that the iterative scheme transforms the implicit Euler time discretization of order 1 into a time discretization of order $k+1$ for any linear parabolic problem, especially for our problem (36).

In the actual implementation of the method, a scheme with fine time step δt is used in parallel over each (T^n, T^{n+1}). The gain in computation time obtained by using the iterative scheme is estimated as follows. The cost of the direct scheme, based on the sole use of the fine solver, is proportional to $T/\delta t$. The cost of the iterative scheme is proportional (with the same constant of proportionality) to $(k+1)(T/\Delta T + \Delta T/\delta t)$, with $n = (\Delta T)^{-1}$. For $k+1$ fixed, the iterative scheme cost is optimized provided $T/\Delta T = \Delta T/\delta t$ since the product $T/\delta t$ is fixed. This implies $\Delta T = \sqrt{T\delta t}$. The accuracy of the iterative scheme will therefore be comparable to the direct simulation provided $k+1=2$ since $\delta t = T^{-1}(\Delta T)^2$. The maximal gain in time is then found to be

$$G = \frac{1}{4}\left(\frac{T}{\delta t}\right)^{\frac{1}{2}},$$

provided we have at our disposal $(T/\delta t)^{1/2}$ processors. This theoretical calculation doesn't take into account the communication cost in a real parallel implementation.

4.3 Stiff nonlinear equations

> I have a theory that whenever you want to get in trouble with a method, look for the Van der Pol equation.
> (P.E.Zadunaisky 1982)

One of the simplest nonlinear equations describing a vacuum-tube circuit is van der Pol's equation

$$\begin{aligned} &y'' + \mu(y^2-1)y' + y = 0, \\ &y(t_0) = 2, \quad y'(t_0) = 0. \end{aligned} \tag{43}$$

A plot of (43) with $\mu = 10$ can be seen in Figure 14.

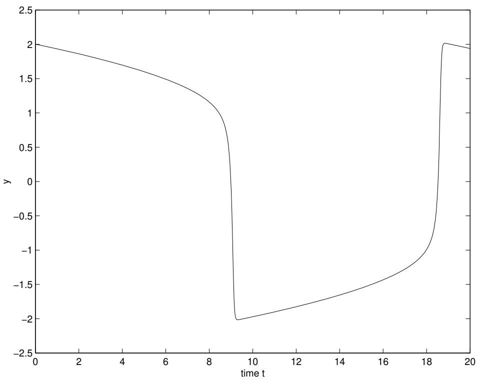


Figure 14: Solution of van der Pol with $\mu = 10$

Again we write it as a system of first order differential equations

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}' = \begin{bmatrix} y_2 \\ \mu(1-y_1^2)y_2 - y_1 \end{bmatrix} \tag{44}$$

Van der Pol has several attractive properties.

- nonlinear
- stiff, with a controllable "stiffness-factor"

4.3.1 Analysis of van der Pol

We can write (43) as a Lienard system

$$\ddot{x} + f(x)\dot{x} + g(x) = 0, \quad F(x) = \int_0^x f(s)ds, \quad G(x) = \int_0^x g(s)ds$$

with the energy function

$$u(x,y) = \frac{y^2}{2} + G(x),$$

where y comes from the system of (45)

$$\begin{aligned} \dot{x} &= y - F(x) \\ \dot{y} &= -g(x) \end{aligned}$$

To analyse (43) we need Lienard's theorem, which we present without proof. The interested reader may refere to [19].

Theorem 17 *Under the assumption that $F, g \in C^1(\mathbb{R})$, F and g are odd functions of x, $xg(x) > 0$ for $x \neq 0$, $F(0) = 0$, $F'(0) < 0$, F has single positive zero at $x = a$, and F increases monotonically to infinity for $x \geq a$ as $x \to \infty$, it follows that the Lienard system (45) has exactly one limit cycle and it is stable.*

We use this theorem and state the following corrolary

Corrolar 1 *For $\mu > 0$, van der Pol's equation (43) has a unique limit cycle and it is stable.*

Proof: $g(x) = x$ and we calculating $F(x)$.

$$F(x) = \mu x \left(\frac{x^2}{3} - 1 \right)$$

It's obvious that $g, F \in C^1(\mathbb{R})$, and $g(-x) = -x = g(x)$ and $F(-x) = -\mu x(\frac{(-x)^2}{3} - 1) = -F(x)$ both are odd functions of x. Further $F(0) = 0$ and $F'(0) = \mu(x^2 - 1)|_{x=0} = -\mu < 0$ for $\mu > 0$. $F(x)$ has roots at $x = 0, \pm\sqrt{3}$, and is monotonically increasing for $x = \sqrt{3}$. $\lim_{x\to\infty} F(x) = \infty$. $\square$

This shows that our non-linear equation (43) is bounded, and therefore a stable numerical scheme will also be stable on the problem.

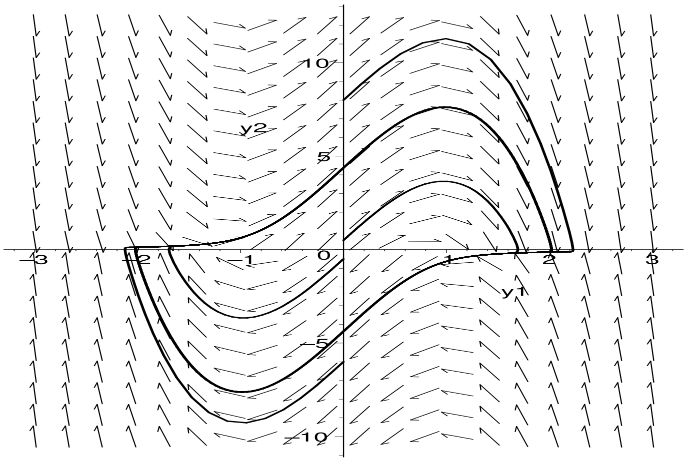


Figure 15: Phaseplot for van der Pol with $\mu = 10$

A phaseplot for $\mu = 10$ can be seen in Figure 15. Notice that all the initial-values end up in the stable cycle, just as we stated in Corrolary 1. We also notice that the stable cycle lies on the interval $x \in [-2, 2]$. From this we draw the conclusion that van der Pol has a periodic nature, for $\mu > 0$.

4.3.2 What kind of ODE-solvers are normaly used on stiff problems?

> ... Around 1960, things became completly different and everyone became aware that the world was full of stiff problems
> (G. Dahlquist in Aiken 1985)

To be able to discuss this, we have to know what stiff is. It turns out that stiffness is not very well defined.

A very "engineer-like" definition is the following. A ODE system is *stiff* if some methods requires (not necessary on the entire time-domain) a significant deprication of the step-size to avoid instability. From Section 3.1 we remember that non A-stability needed a decrease in stepsize if the eigenvalues increased (A-, AN- or BN-stabil dependent of the problem). It's therefore intuitive to define a *stiffnes ratio* as *stiffness-ratio*$=\frac{\lambda_{max}}{\lambda_{min}}$.

Shampine points out in [22] that even if a ODE-system is stiff, it doesn't meen a non-stiff solver will get into problems because our time-domain not necessarly include the stiff portions of the ODE-system.

Not to make it more confusing we will only point out that there is no optimal general procedure for stiff problems, and every stiff problem (and engineer/scientist) has a "favorite" method.

But we will mention some numerical schemes that work well for general stiff problems.

- The Radau class of implicit Runge-Kutta methods (e.g Table 7,Table 8,Table 9)
- Some DIRK and SDIRK schemes (e.g Table 10, but this is not L-stable, and therefore not very good for all kind of stiff problems)
- A class called Rosenbrock-Type methods (not handled here). Generally it's IRK method that avoids nonlinear systems by replacing them with a sequence of linear systems.
- Different types of multi-step method. This is the most used method for stiff problems, probably because it was the first numerical method that was proposed for this kind of problems [9].

5 Stability analysis of the algorithm

We intend to analyse the stability of the predictor-corrector scheme

$$\lambda_n^k = \mathcal{G}_{\Delta T}(\lambda_{n-1}^k) + \mathcal{F}_{\Delta T}(\lambda_{n-1}^{k-1}) - \mathcal{G}_{\Delta T}(\lambda_{n-1}^{k-1}), \quad n, k = 1, \ldots, N. \tag{45}$$

First a couple of facts has to be established.

$$y_0 = \lambda_0^0 = \lambda_0^1 = \ldots = \lambda_0^N \tag{46}$$

$$\lambda_{i+1}^0 = \mathcal{G}_{\Delta T}(\lambda_i^0), \quad i = 0, \ldots, N-1, \tag{47}$$

which is just initialization of the algorithm. We now analyse (45) on the autonomous differential equation

$$y' = \mu y, \quad y(t_0) = y_0.$$

The first important assumption is now made. $\mathcal{F}_{\Delta T}(\lambda_n^k)$ is assumed to be exact (exact in our context), and we may write it as

$$\mathcal{F}_{\Delta T}(\lambda_n^k) = e^{\mu \Delta T} \lambda_n^k$$

$\mathcal{G}_{\Delta T}(\lambda_n^k)$ har, for the chosen scheme, the stability-function $R(z)$, where $z = \mu \Delta T$. We then rewrite the expression of $\mathcal{F}_{\Delta T}(\lambda_n^k)$, using z.

$$\mathcal{F}_{\Delta T}(\lambda_n^k) = e^z \lambda_n^k$$

(45) is now rewritten as

$$\lambda_n^k = R(z)\lambda_{n-1}^k + e^z \lambda_{n-1}^{k-1} - R(z)\lambda_{n-1}^{k-1}, \quad n, k = 0, \ldots, N$$

We now make our second assumption. We assume μ is large and negative, and that ΔT also is large (this is not an unlikely assumption in the parareal algorithm). This makes $|z| \gg 1$, with negative sign. As a consequence of this we assume that

$$e^{\mu \Delta T} \approx 0$$

We now have our correction-scheme on a form that simplifies the analysis.

$$\lambda_n^k = R(z)\lambda_{n-1}^k - R(z)\lambda_{n-1}^{k-1} + \mathcal{O}(e^z) \tag{48}$$

We now intend to write the correction scheme (48) on the form $\lambda_n^k = H(n, k, z) y_0$, where $H(n, k, z)$ will be en expression of the stability function. To do this we will recursively substitute λ_i^j in (48) until $i = 0$ or $j = 0$. When $i = 0$ we use (46), and when $j = 0$ we use (47). R will be written as a short for $R(z)$. To make the pattern more clear, we write out a few recursions.

$$\begin{aligned}
\lambda_n^k &= R\lambda_{n-1}^k - R\lambda_{n-1}^{k-1} = R(\lambda_{n-1}^k - \lambda_{n-1}^{k-1}) \\
&= R(R\lambda_{n-2}^k - R\lambda_{n-2}^{k-1} - R\lambda_{n-2}^{k-1} + R\lambda_{n-2}^{k-2}) = R^2(\lambda_{n-2}^k - 2\lambda_{n-2}^{k-1} + \lambda_{n-2}^{k-2}) \\
&= R^3(\lambda_{n-3}^k - 3\lambda_{n-3}^{k-1} + 3\lambda_{n-3}^{k-2} - \lambda_{n-3}^{k-3}) \\
&= R^4(\lambda_{n-4}^k - 4\lambda_{n-4}^{k-1} + 6\lambda_{n-4}^{k-2} - 4\lambda_{n-4}^{k-3} + \lambda_{n-4}^{k-4})
\end{aligned}$$

This recursive development of λ_i^j can easily be expressed as a directed graph

$$
\begin{array}{c|ccccccccc}
 & 0 & & 1 & & 2 & & 3 & & 4 \\
\hline
0 & \lambda_n^k & & & & & & & & \\
 & \downarrow & \searrow & & & & & & & \\
1 & \lambda_{n-1}^k & & \lambda_{n-1}^{k-1} & & & & & & \\
 & \downarrow & \searrow & \downarrow & \searrow & & & & & \\
2 & \lambda_{n-2}^k & & \lambda_{n-2}^{k-1} & & \lambda_{n-2}^{k-2} & & & & \\
 & \downarrow & \searrow & \downarrow & \searrow & \downarrow & \searrow & & & \\
3 & \lambda_{n-3}^k & & \lambda_{n-3}^{k-1} & & \lambda_{n-3}^{k-2} & & \lambda_{n-3}^{k-3} & & \\
 & \downarrow & \searrow & \downarrow & \searrow & \downarrow & \searrow & \downarrow & \searrow & \\
4 & \lambda_{n-4}^k & & \lambda_{n-4}^{k-1} & & \lambda_{n-4}^{k-2} & & \lambda_{n-4}^{k-3} & & \lambda_{n-4}^{k-4}
\end{array}
$$

where we se that alle the elements from one level is devided into 2 levels on the next step. This pattern can also be seen directly from (48).

But of course it is the constants in front of each λ_i^j that interest us most. We systemize the contributions from the recursions in the graph C, where C_i^j are the constants for the terms $C_i^j R^i \lambda_{n-i}^j$.

$$
C = \begin{array}{r|ccccccccc}
i\backslash j & 0 & & 1 & & 2 & & 3 & & 4 \\
\hline
1 & 1 & & & & & & & & \\
 & \downarrow & \searrow & & & & & \downarrow & \searrow & \\
2 & 1 & & -1 & & & & + & & - \\
 & \downarrow & \searrow & \downarrow & \searrow & & & & & \\
3 & 1 & & -2 & & 1 & & & & \\
 & \downarrow & \searrow & \downarrow & \searrow & \downarrow & \searrow & & & \\
4 & 1 & & -3 & & 3 & & -1 & & \\
 & \downarrow & \searrow & \downarrow & \searrow & \downarrow & \searrow & \downarrow & \searrow & \\
5 & 1 & & -4 & & 6 & & -4 & & 1
\end{array}
$$

The rules are derived from (48):

- every node is added down
- every node is subtracted down-right

No other paths exist.

It's obvious from (48) that

$$\lambda_n^k = R^i \sum_{j=0}^{i} C_i^j \lambda_{n-i}^{k-j} + \mathcal{O}(e^z), \quad i \leq k$$

for our i^{th} recursion.

We now apply (47). Assume $j = k$ for some $i \leq n$. Then

$$
\begin{aligned}
& \lambda_{n-i}^0 = R^{n-i} \lambda_0^0 \\
\Rightarrow \qquad & \lambda_n^k = \ldots + C_i^k R^i R^{n-i} \lambda_0^0 \ldots = \ldots + C_i^k R^n \lambda_0^0
\end{aligned}
$$

This means that the propagator $\mathcal{G}_{\Delta T}$ is applied n times to reach timestep n, which really is a confirmation of the obvious. Then we apply (46). Assume $i = n-1$ for some $j > 0$. Then

$$\lambda_1^j = R(\lambda_0^j - \lambda_0^{j-1}) = 0.$$

This implies that only terms that reaches $k = 0$ for $n \geq 1$ are contributing to the constant. But this property is handled by our down-right graph. This is easier seen with an example, where $k = 2$ and $n = 5$.

$$C = \begin{array}{c|ccccc} i\backslash j & 0 & & 1 & & 2 \\ \hline 1 & 1 & & & & \\ & \downarrow & \searrow & & & \\ 2 & 1 & & -1 & & \\ & \downarrow & \searrow & \downarrow & \searrow & \\ 3 & 1 & & -2 & & 1 \\ & \downarrow & \searrow & \downarrow & \searrow & \downarrow \\ 4 & 1 & & -3 & & 3 \\ & \downarrow & \searrow & \downarrow & \searrow & \downarrow \\ 5 & 1 & & -4 & & 6 \end{array} \quad \begin{array}{c|ccccc} i\backslash j & 0 & & 1 & & 2 \\ \hline 1 & \lambda_5^2 & & & & \\ & \downarrow & \searrow & & & \\ 2 & \lambda_4^2 & & \lambda_4^1 & & \\ & \downarrow & \searrow & \downarrow & \searrow & \\ 3 & \lambda_3^2 & & \lambda_3^1 & & \lambda_3^0 \\ & \downarrow & \searrow & \downarrow & \searrow & \downarrow \\ 4 & \lambda_2^2 & & \lambda_2^1 & & \lambda_2^0 \\ & \downarrow & \searrow & \downarrow & \searrow & \downarrow \\ 5 & \lambda_1^2 & & \lambda_1^1 & & \lambda_1^0 \end{array}$$

The lower-left value is the summation of all the terms that reached $j = k$ for $i < n$.

We now need an expression for C. $|C|$ is recognized as the binomial coefficient, which is adjusted with -1 for n since n starts at 1 in C.

$$\binom{n-1}{k} = \frac{(n-1)!}{(n-k-1)!k!}$$

The alternating sign is easily handled.

We are now ready to write an expression for the stability-function of our predictor-corrector scheme.

$$\lambda_n^k = (-1)^k \binom{n-1}{k} R(z)^n y_0 + \mathcal{O}(e^z) \tag{49}$$

But is our assumption that $e^z \approx 0$ correct? If we do not omit e^z, (45) can be written as

$$\lambda_n^k = (-1)^k \binom{n-1}{k} R(z)^n y_0 + \sum_{i=1}^{k} (-1)^{k+i} c_i \left(e^z\right)^i R(z)^{n-i} y_0, \tag{50}$$

where c_i are positive constants. Notice that c_i are not necessary equal for different values of n and k. In order to find c_i, we need to develop a 3-dimentional graph. This problem is not handled here. For the time beeing we conclude that this is bounded for sufficient large z, and our assumption holds.

Immediately we recognize one of the properties of the algorithm.

- For $j \geq i$, $\binom{n-1}{k} = 0$, and λ_i^j only depend of $\mathcal{F}_{\Delta T}$, meaning the seriel and parallel verison are equal to this point.

We notice that for n large, and $k = \lfloor n/2 \rfloor$ (worst case), the binomial coefficient grows exponential.

Have we proven the failure of the parareal algorithm? Probably not. The answer lies in $R(z)$. It should now be clear to everyone why we need strong A-stable schemes, perhaps as strong as L-stable schemes to bound the binomial coefficient.

We proceed with an analysis of the growth of the binomial coefficient. It is a fact that for a given n, $\binom{n-1}{k}$ is largest when $k = \lfloor (n-1)/2 \rfloor$. n is now increased, such that the coefficient is $\binom{n+1}{k}$, where $k = \lfloor (n+1)/2 \rfloor$. We further calculate the ratio between them as $a^2 = \binom{n+1}{k} / \binom{n-1}{k}$. The power of two for a comes from the fact that we increased n by two. Since $\binom{n+1}{k}$ now is the largest number for this value of n, the argument is that for any k and n, $\binom{n-1}{k} \leq a^n$. So what is the value of a?

First we find , for $n+1$ and $k = \lfloor n/2 \rfloor$, where we assume (for simplicity) that $k = n/2$ is an integer.

$$\binom{n}{n/2} = \frac{n!}{(n-\frac{n}{2})!\frac{n}{2}!} = \frac{n!}{\left(\frac{n}{2}!\right)^2}.$$

Now, for $n+3$ and $k = \lfloor (n+2)/2 \rfloor$, we have that

$$\begin{aligned}
\binom{n+2}{n/2+1} &= \frac{(n+2)!}{(n+2-\frac{n}{2}-1)!\,(\frac{n}{2}+1)!} \\
&= \frac{(n)!}{\left(\frac{n}{2}!\right)^2} \cdot \frac{(n+1)(n+2)}{\left((\frac{n}{2}+1)!\right)^2} \\
&= \frac{(n)!}{\left(\frac{n}{2}!\right)^2} \cdot \frac{4(n+1)}{(n+2)}
\end{aligned}$$

This means that

$$a^2 = \frac{4(n+1)}{(n+2)},$$

and we find

$$\lim_{n\to\infty} a = 2.$$

From this we can say that

$$\binom{n-1}{k} < 2^n, \quad \forall n, k$$

This proves that, under the assumptions made, it is sufficient to use a strong A-stable coarse propagator with $|R(z)| = \frac{1}{2}$, where $z = \mu\Delta T$, to make the predictor-corrector scheme stable!

We summerize all the discovered results in a theorem

Theorem 18 *Assume $y' = \mu y,\ y(t_0) = y_0,\ t \in (t_0, T)$ is solved using the predictor-corrector scheme*

$$\lambda_n^k = \mathcal{G}_{\Delta T}(\lambda_{n-1}^k) + \mathcal{F}_{\Delta T}(\lambda_{n-1}^{k-1}) - \mathcal{G}_{\Delta T}(\lambda_{n-1}^{k-1}), \tag{51}$$

where $\mathcal{G}_{\Delta T}$ is the coarse propagator and $\mathcal{F}_{\Delta T}$ is the fine propagator defined in Section 2. Assume also that $|z| = |\mu\Delta T| \gg 1$, where $z < 0$. Then the stability function for (51), for any values of $n, k \leq N$ where $N = \frac{T-t_0}{\Delta T}$ is the number of partitions in time, is given by

$$\lambda_n^k = (-1)^k \binom{n-1}{k} R(z)^n y_0, \quad z = \mu\Delta T,$$

where $R(z)$ is the stability-function of the coarse propagator $\mathcal{G}_{\Delta T}$. For the choice of coarse propagator $\mathcal{G}_{\Delta T}$ with the property

$$|R(z)| \leq \frac{1}{2},$$

the predictor-corrector scheme (51) is stable for all values of n, k.

The theorem also applies for $y' = Ay$, where y is a vector and A is a matrix. The only difference is that μ will now be the largest eigenvalue in A.

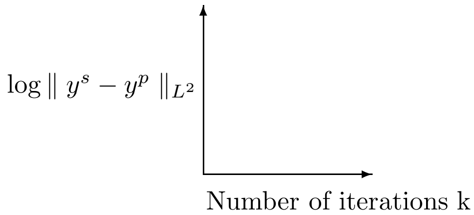
So what about non-autonomous and non-linear problems? Theorem 18 does not apply for this kinds of problem. The difference lies in the stability-function $R(z)$, which is not the same for non-autonomous and non-linear problems. An extention of the theory for those kind of problems is left for future work.

6 Numerical Results

To be able to compare the different results we are about to present, some important quantities will be defined, which will be explicitly given for each result.

- Timestep δt and ΔT for $\mathcal{F}_{\Delta T}$ and $\mathcal{G}_{\Delta T}$.
- The systems natural time constant τ_s. As an example, for a solution $u = \sin 2\pi t$, $\tau_s = 1$.
- The ratio $\Delta\tau_s = \frac{\Delta T}{\tau_s}$.
- The solvers for $\mathcal{G}_{\Delta T}$ and $\mathcal{F}_{\Delta T}$.

As long as nothing else is specified, all the plots are on the form



The x-axis is the iteration-counter k (called *outer iter k* to seperate it from any internal iteration, e.g. Newton), and the y-axis is the difference between the serial and the parareal version measured in L^2-norm.

The same numerical schemes will be applied to almost all the tests. So again, as long as nothing else is spesified, the legend will be as in Table 11.

Legend							
——	EB	·—·	Radau3	—+—	Radau5	—▽—	SDIRK3
— —	Gauss2	—×—	Gauss4	—◇—	Gauss6	—△—	Gauss8

Table 11: Legend for the plots

Since the value of the stability function $R(z)$ will be an important issue for all the schemes, for different values of z, we first present their behaviour in Figure 16.

We will not discusse speedup. This is because it is generally difficult to compare computational expence when we are using numerical schemes that are not optimal implemented (implemented in `Matlab`). The implementation may change the computational expence radically. But as a rule of thumb we state that to achieve any speedup at all, we need convergens to a satisfactory level of error on less then half of the maximum number of iterations. This is based on the assumption that the time used by $\mathcal{G}_{\Delta T}$ and $\mathcal{F}_{\Delta T}$ per iteration is approximate the same. This should be kept in mind when considering the convergence-plots.

We start by presenting the results from the simulation of the simple heat equation (36).

6.1 Heat-equation

We start with (36) since it is, in many ways, an easy equation to solve. It is also a nice test-equation for the stability-properties we discovered in Section 5.

Space is discretized using finite element method (FEM) as spesified in Section 4.2.1. In the following plots we have discretized in space using 20 equal-size finite elements (FE). $t \in (0, 1)$,

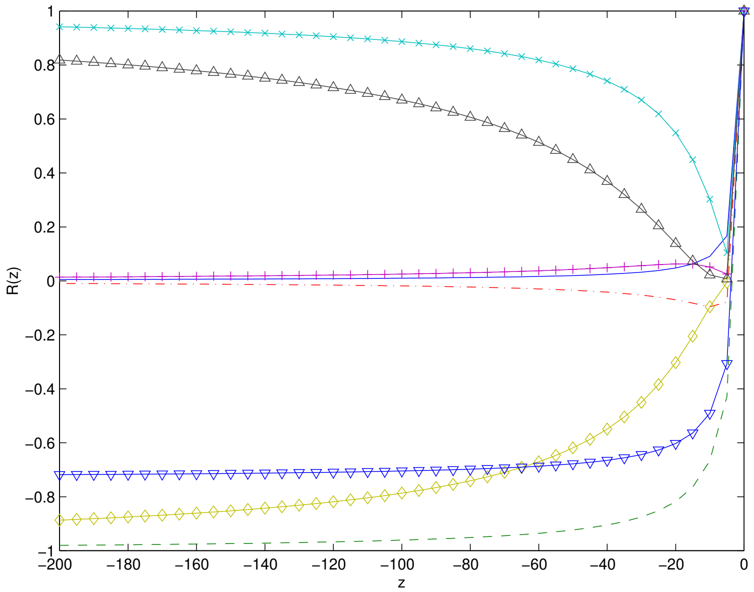


Figure 16: $R(z)$ for the schemes presented in Section 3. Legend found in Table 11

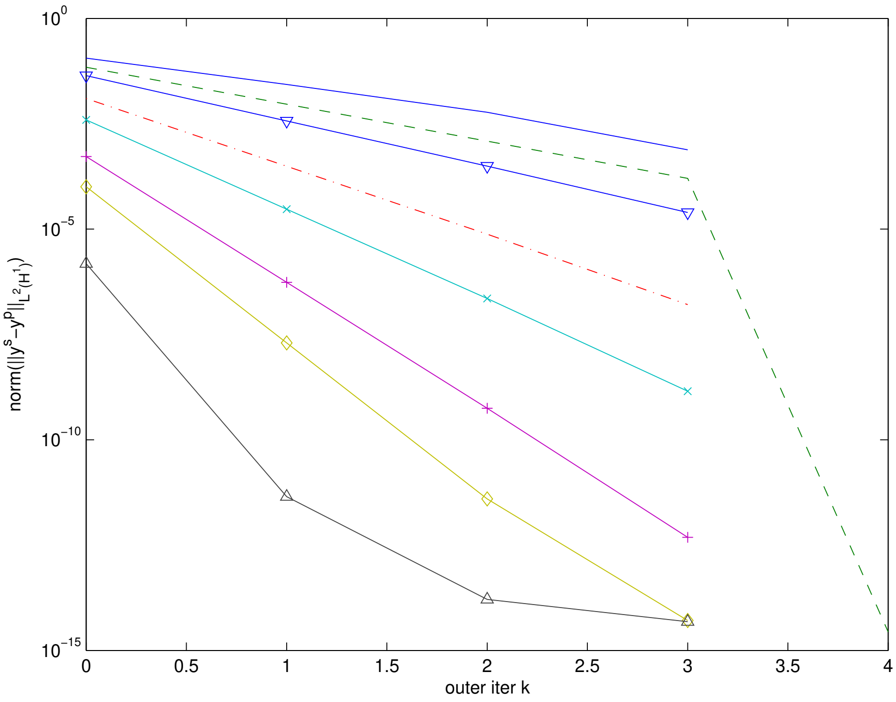
$\tau_s = 1$, $\delta t = 1 \cdot 10^{-3}$, and we use the same solver for both $\mathcal{G}_{\Delta T}$ and $\mathcal{F}_{\Delta T}$. This gives us the largest eigenvalue $\lambda_{max} = -4.7124 \cdot 10^3$ from the system $\frac{\partial u}{\partial t} = -(Mh^{-1}Ah)u$. From Section 5 we know that if $|z| \gg 1$ with $z = \lambda_{max}\Delta T$, we will experience instability for schemes with $R|(z)| > \frac{1}{2}$. From the analysis of the schemes, we know that all methods except the Radau methods have $\lim_{z\to-\infty} |R(z)| > \frac{1}{2}$. In Figure 17 and Figure 18 the same scheme is used for both $\mathcal{G}_{\Delta T}$ and $\mathcal{F}_{\Delta T}$, with different values for ΔT.

In Figure 17(a), all the schemes converge without any sign of divergence. Notice also that the convergence is exponential (except Gauss8 which is almost fully converged after one iteration), and it increases with order, as expected. The reason why some of the plots have no point for the last value of *outer iter* k is that the error is zero in the machine-precision. $\log(0) = 10^{-\infty}$, and is therefore not plotted. Already in Figure 17(b), a slight divergence can be seen for some value of k in Gauss4, Gauss6 and Gauss8. In Figure 18(a), all the schemes except the Radau schemes diverge for some values of the outer iteration k. This divergence increases for smaller value of ΔT, but the Radau-schemes are never effected.

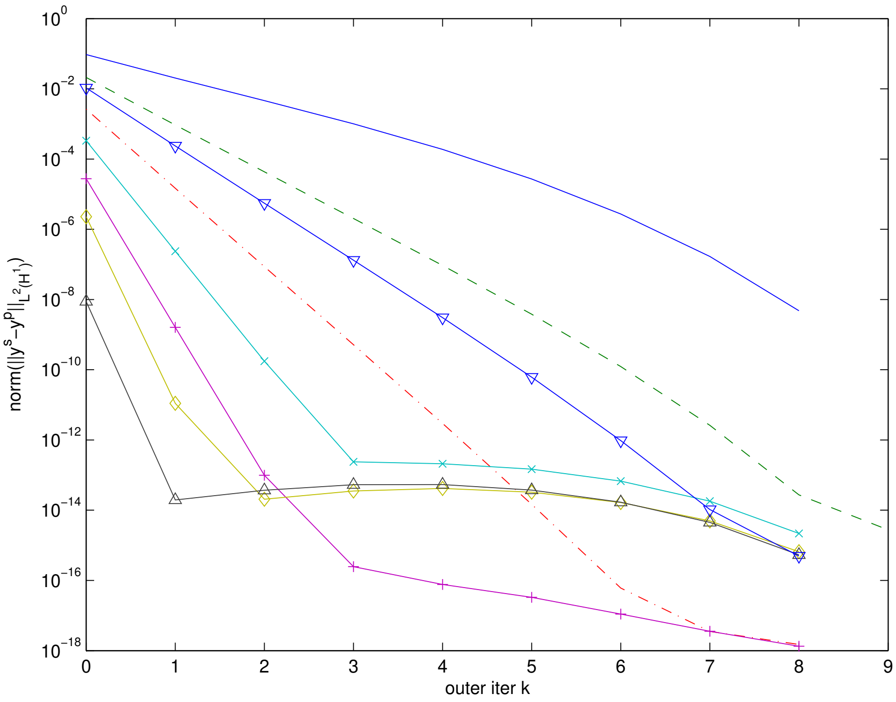
But before we establish if this could be predicted, we want to verify that only the coarse propagator $\mathcal{G}_{\Delta T}$ effects the stability of the predictor-corrector scheme. In Figure 19 and Figure 20, we have frozen either $\mathcal{F}_{\Delta T}$ or $\mathcal{G}_{\Delta T}$ as either Gauss6 or Radau5. In Figure 19(a) and Figure 20(a), $\mathcal{F}_{\Delta T}$ are Gauss6 versus Radau5 for alle the simulations. $\mathcal{G}_{\Delta T}$ is altered according to Table 11. It's obvious that that the choice of $\mathcal{F}_{\Delta T}$ has no effect on the stability of the predictor-corrector scheme, as predicted in Theorem 18. In Figure 19(b) and Figure 20(b), $\mathcal{G}_{\Delta T}$ are Gauss6 versus Radau5 for alle the simulations. $\mathcal{F}_{\Delta T}$ is altered according to Table 11. Clearly Radau5 prevents instability, but Gauss6 fails to do so. This also predicted using Theorem 18 and Figure 16, with the values $z = \lambda_{max}\Delta T \approx 120$.

But what happens that generates this instability? To try to answar this we look, in Figure 21 and Figure 22, closer at a special case with the following variables:

- $\mathcal{G}_{\Delta T}$ is Gauss2
- $\mathcal{F}_{\Delta T}$ is Gauss2 in Figure 21, and Radau3 in Figure 22

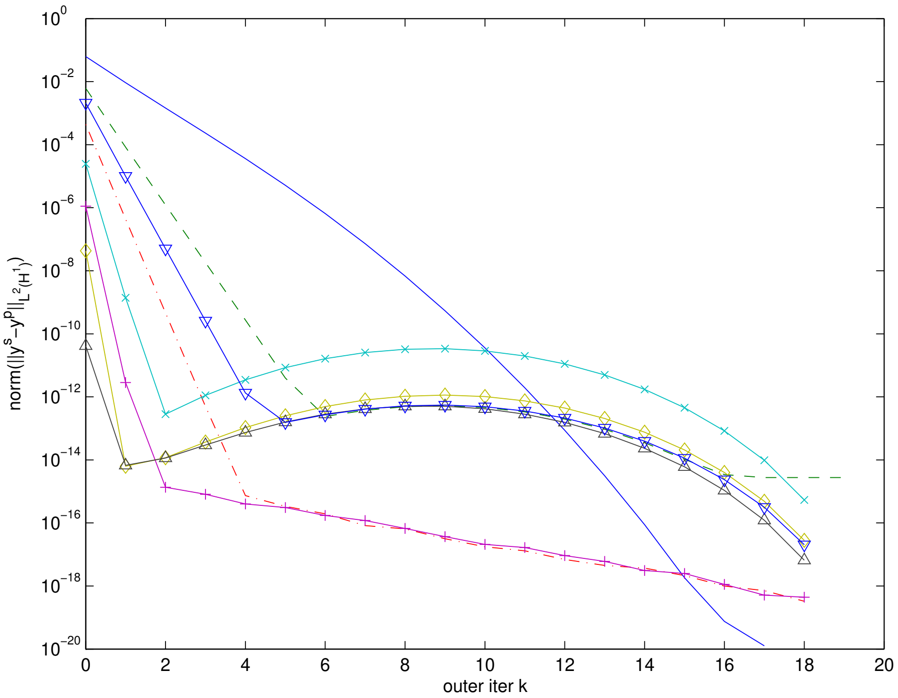


(a) $\Delta T = 0.2$, $\Delta\tau_s = 0.2$

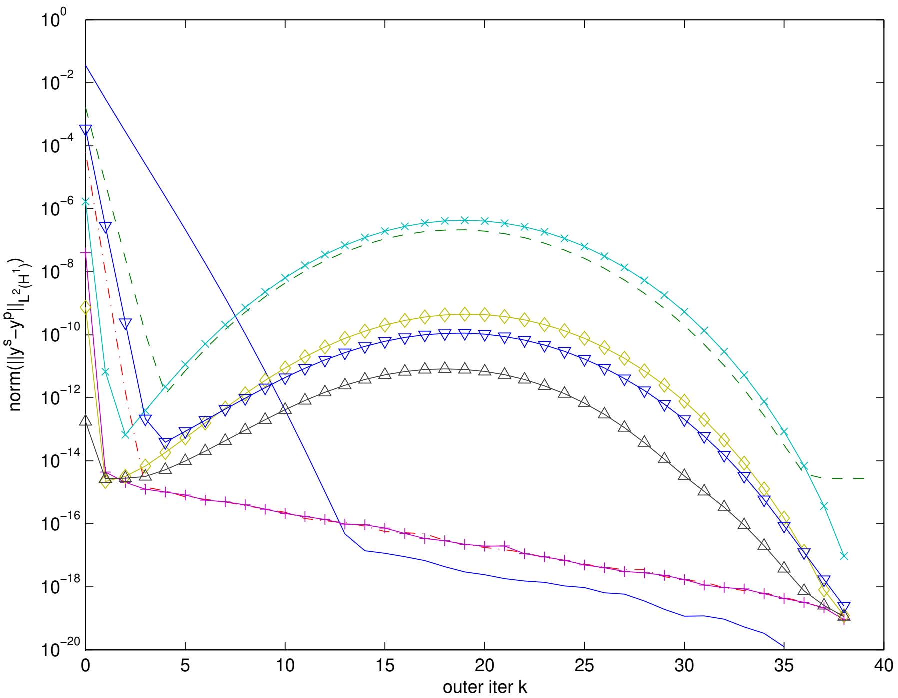


(b) $\Delta T = 0.1$, $\Delta\tau_s = 0.1$

Figure 17: Solution of (36) with $\delta t = 1 \cdot 10^{-3}$, $\tau_s = 1$. Legend found in Table 11

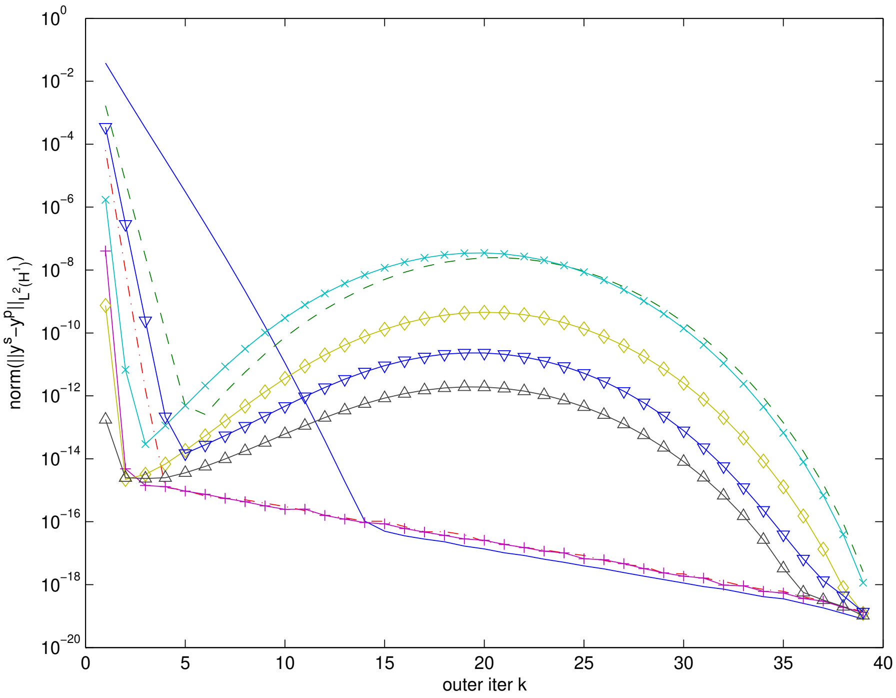


(a) $\Delta T = 0.05$, $\Delta\tau_s = 0.05$

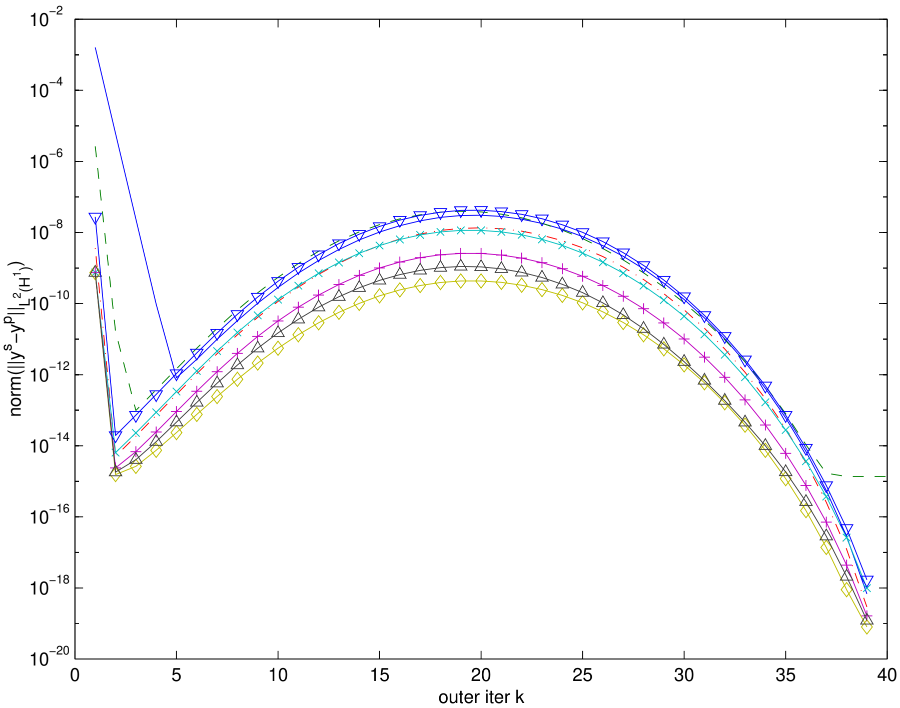


(b) $\Delta T = 0.025$, $\Delta\tau_s = 0.025$

Figure 18: Solution of (36) with $\delta t = 1 \cdot 10^{-3}$, $\tau_s = 1$. Legend found in Table 11

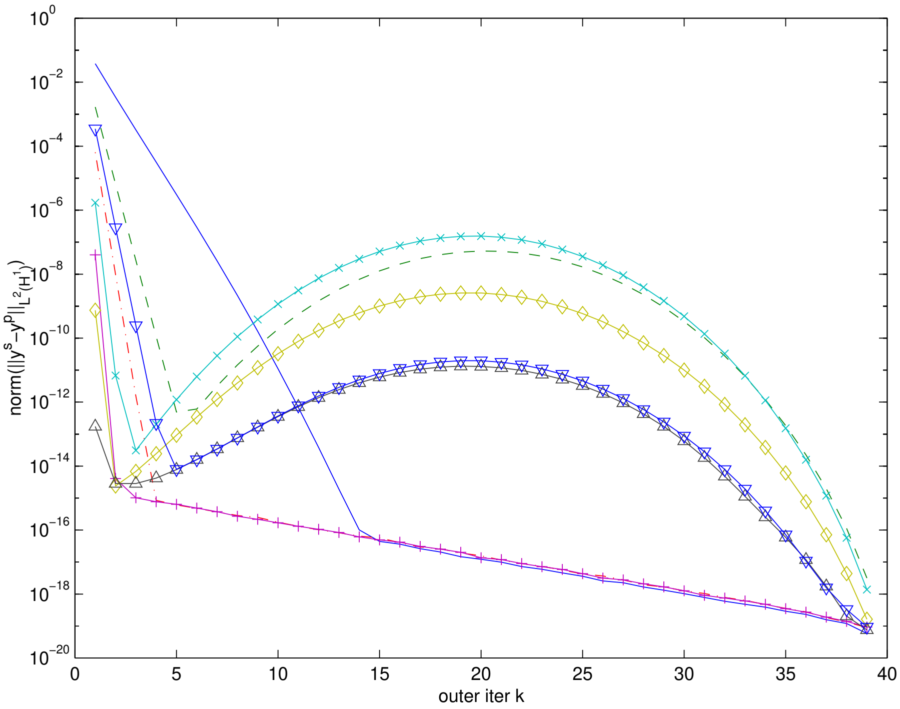


(a) Gauss6 as $\mathcal{F}_{\Delta T}$. $\mathcal{G}_{\Delta T}$ found in Table 11.

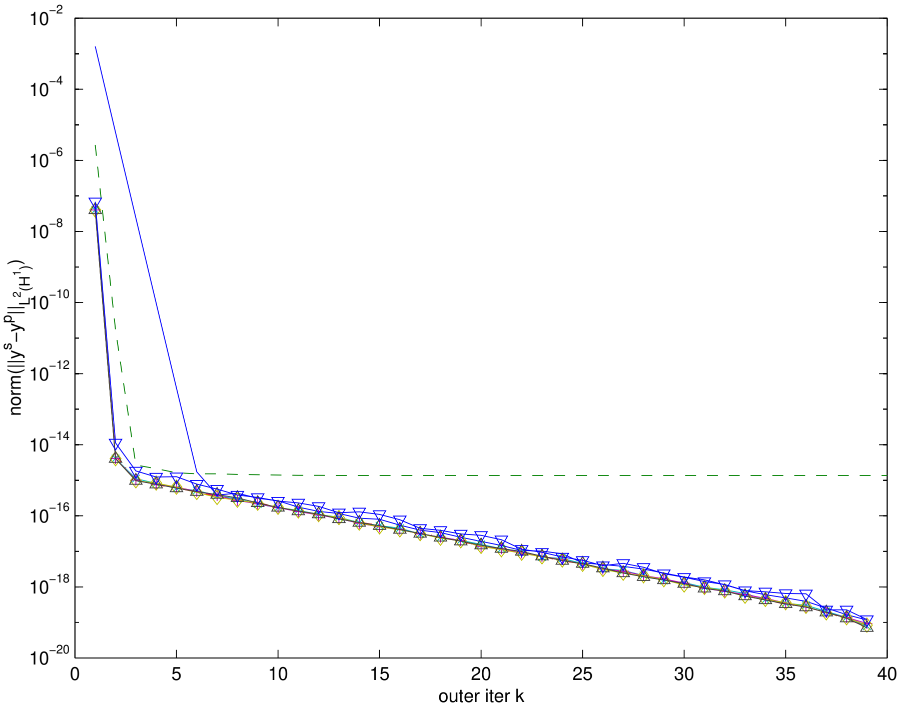


(b) Gauss6 as $\mathcal{G}_{\Delta T}$. $\mathcal{F}_{\Delta T}$ found in Table 11.

Figure 19: Solution of (36) with $\delta t = 1 \cdot 10^{-3}$, $\tau_s = 1$, $\Delta T = 2.5 \cdot 10^{-2}$, $\Delta\tau_s = 2.5 \cdot 10^{-2}$. Legend found in Table 11



(a) Radau5 as $\mathcal{F}_{\Delta T}$. $\mathcal{G}_{\Delta T}$ found in Table 11.



(b) Radau5 as $\mathcal{G}_{\Delta T}$. $\mathcal{F}_{\Delta T}$ found in Table 11.

Figure 20: Solution of (36) with $\delta t = 1 \cdot 10^{-3}$, $\tau_s = 1$, $\Delta T = 2.5 \cdot 10^{-2}$, $\Delta \tau_s = 2.5 \cdot 10^{-2}$. Legend found in Table 11

- The iteration k= 20
- $\Delta T = 1 \cdot 10^{-2}$
- Result plotted for $x = 0.8$

The rest is the same as the previous cases, namely $\delta t = 1 \cdot 10^{-3}$ and $\tau_s = 1$. The reason why ΔT is chosen so small is to have really large oscillations to study. The x-value is frozen, so we plot the time-variations of the specified x-value. We then plot the following quantaties:

- $\mathcal{F}_{\Delta T}(\lambda^{k+1})$
- $\mathcal{G}_{\Delta T}(\lambda^{k+1})$ and $\mathcal{G}_{\Delta T}(\lambda^{k})$
- λ^{k+1} and λ^{k}

We gather the results from all the parallel computed $\mathcal{F}_{\Delta T}$ such that the first value from processor $i+1$ overwrites the last value from processor i.

In Figure 21(a) and Figure 22(a), the oscillations are significant. In Figure 21(b) and Figure 22(b) only the last section of the time-domain is plotted. The first thing we notice is the difference in damping-factor between Gauss2 and Radau3. This is because Radau3 is L-stable, and Gauss2 is not even strong A-stable (discussed in Section 3.1.4).

But our attention should be focused on $\mathcal{G}_{\Delta T}$ and λ. Remember that $\mathcal{G}_{\Delta T}(\lambda^{k+1})$ uses one timestep, with λ^{k+1} as initial-value. Notice that $\mathcal{G}_{\Delta T}(\lambda^{k+1})$ is a weakly damped oscillation from the initial value λ^{k+1}. So when (4) is applied, $\mathcal{G}_{\Delta T}(\lambda^{k+1})$ and $\mathcal{G}_{\Delta T}(\lambda^{k})$ have opposite signs and is therefor amplifying each other instead of equalizing each other. What Theorem 18 is saying, is really how strong should the damping factor of $\mathcal{G}_{\Delta T}$ be to prevent this amplifying.

In Figure 17(a), all the schemes look stable. But are they really stable? We investigate this by extracting λ_3^4 from the simulation with a guaranteed stable scheme (Radau5) as $\mathcal{G}_{\Delta T}$, and the scheme we want to investigate (Gauss6) as $\mathcal{G}_{\Delta T}$. The difference is plotted in Figure 23. What is plottet is really the difference of Radau5 and Gauss6 in the start-vector for the last time-section. A stable scheme will preserve the smoth solution in space, given by the initial value from (36). Because of difference in order it is expected to be a difference between Radau5 and Gauss6. But the difference should also bo a smooth function. Figure 23 shows sign of oscillations. Remember that the largest value of the exact solution for this time is $3.7 \cdot 10^{-4}$, meaning that the oscillation will appeare in the ninth digit. But the oscillations are so small that they do not have a significant slowdown on the convergence. This is not a proof, but a good indication that there may also excist small instability, even though convergence is achieved.

If we use Theorem 18, the results are that we should have instability for all the Gauss schemes and the SDIRK3 scheme, while all the Radau schemes would be stable.

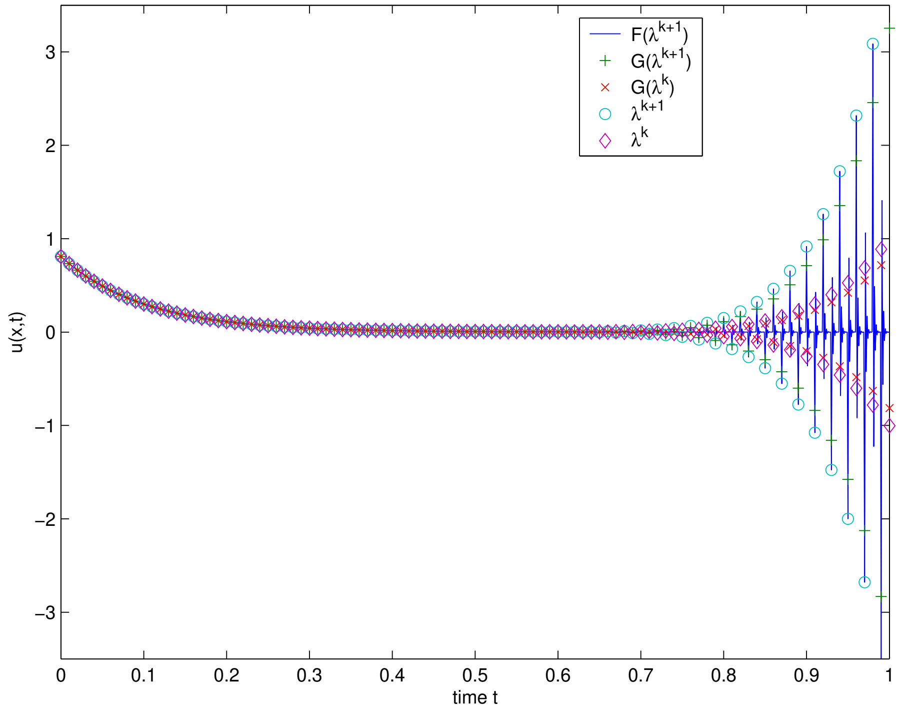
But we have to remember that Theorem 18 is a conservative estimate. The question is how sharp is the property that $\lim_{z\to-\infty} |R(z)| \leq 1/2$. This will be tested numerically.

6.1.1 Verification of Theorem 18

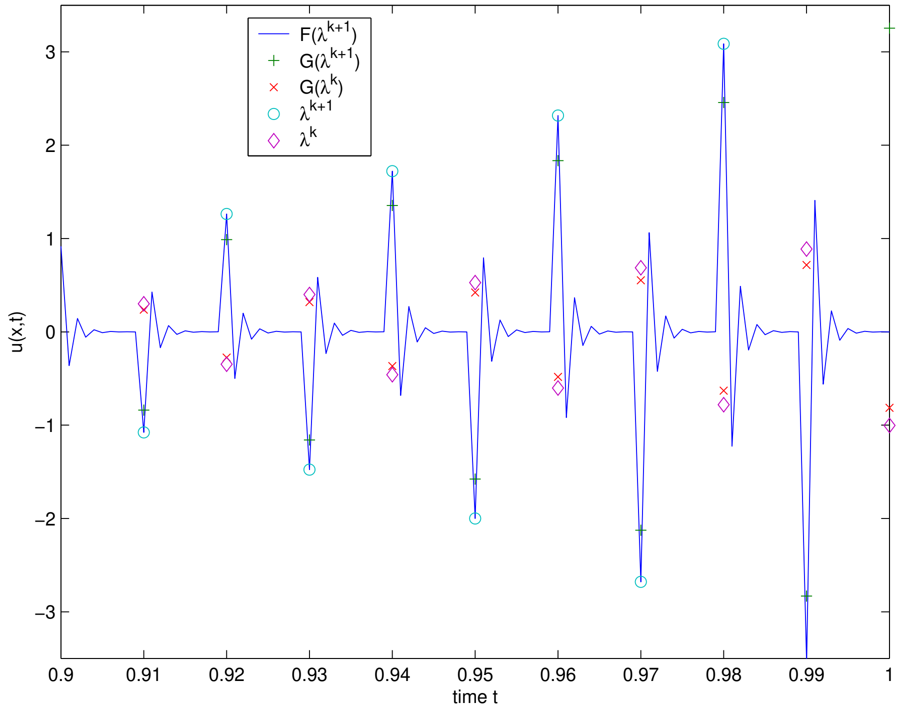
Theorem 18 claims that

$$\lim_{z\to-\infty} |R(z)| \leq \frac{1}{2},$$

where $R(z)$ is the stability-function of $\mathcal{G}_{\Delta T}$, in order for the predictor-corrector scheme (4) to be stable. The reader is refered to Theorem 18 if the assumptions for this is not clear.

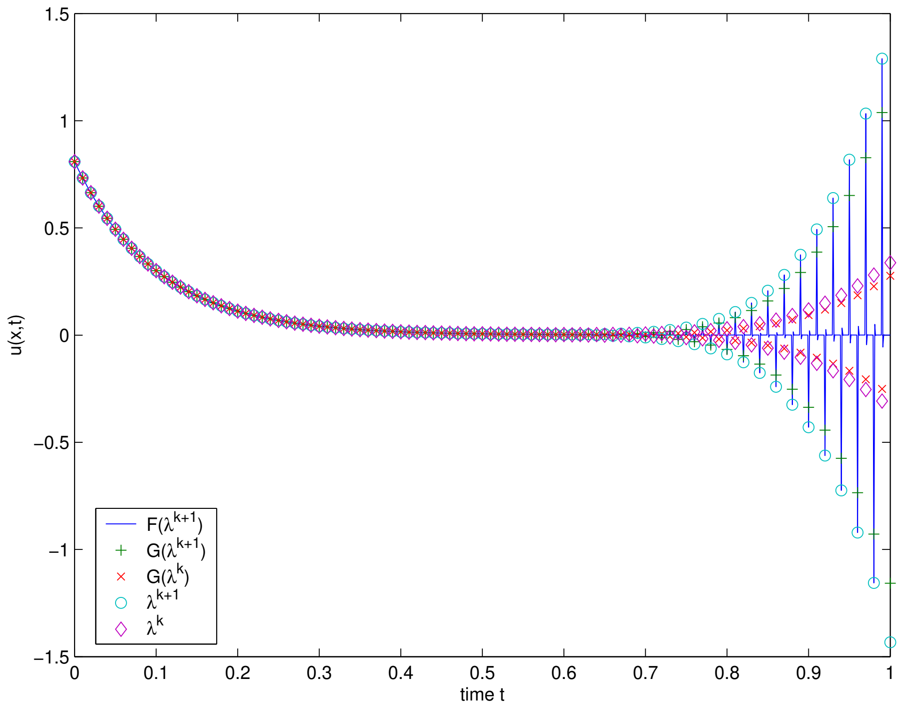


(a) Plot of $x = 0.8$ for the entire timedomain $t \in (0, 1)$

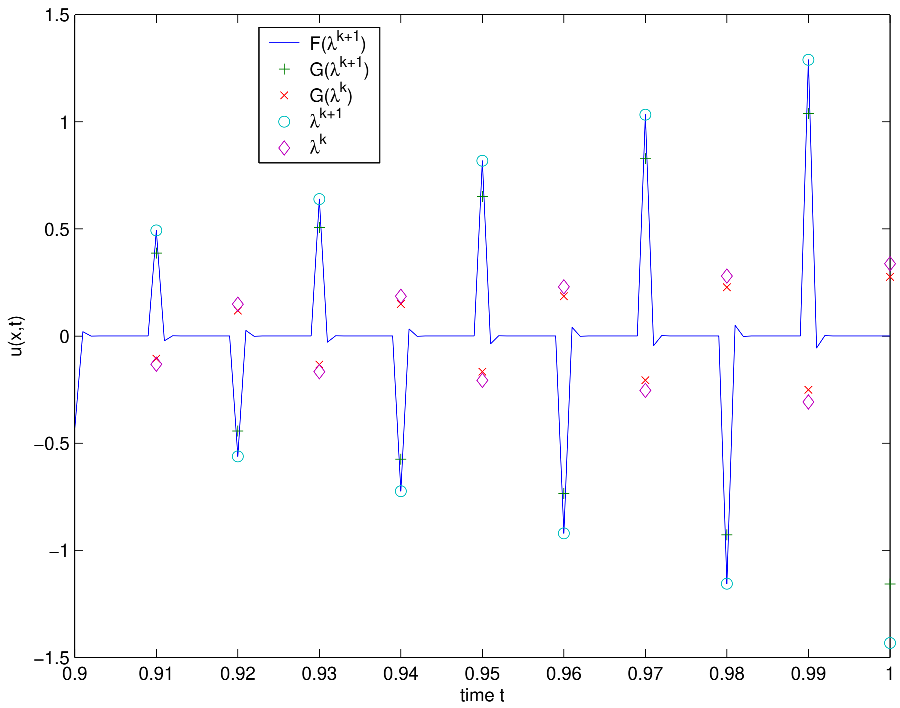


(b) Plot of $x = 0.8$ for a section of the timedomain $t \in (0.9, 1)$

Figure 21: Solution of (36) with $\delta t = 1 \cdot 10^{-3}$, $\tau_s = 1$, $\Delta T = 1.0 \cdot 10^{-2}$, $\Delta\tau_s = 1.0 \cdot 10^{-2}$, $k = 20$ and Gauss2 as $\mathcal{F}_{\Delta T}$ and $\mathcal{G}_{\Delta T}$.



(a) Plot of $x = 0.8$ for the entire timedomain $t \in (0, 1)$



(b) Plot of $x = 0.8$ for a section of the timedomain $t \in (0.9, 1)$

Figure 22: Solution of (36) with $\delta t = 1 \cdot 10^{-3}$, $\tau_s = 1$, $\Delta T = 1.0 \cdot 10^{-2}$, $\Delta\tau_s = 1.0 \cdot 10^{-2}$, $k = 20$, Gauss2 as $\mathcal{F}_{\Delta T}$ and Radau3 as $\mathcal{G}_{\Delta T}$.

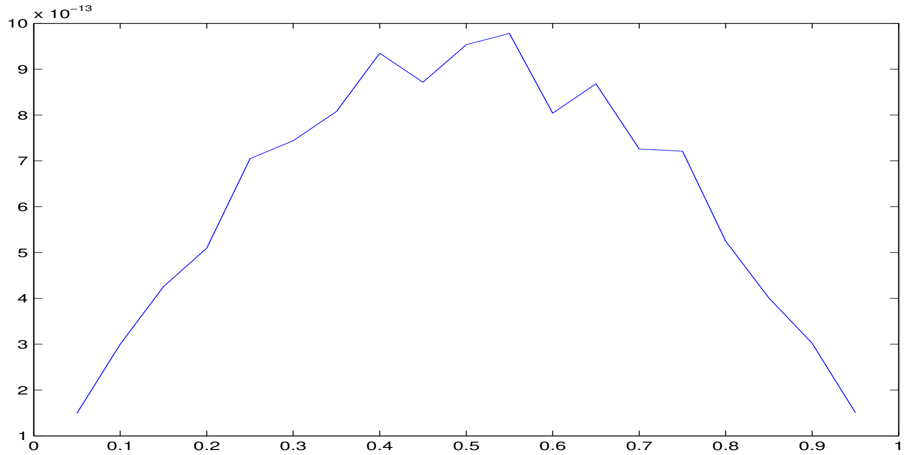


Figure 23: $(\lambda_n^k)_{u_{Gauss6}} - (\lambda_n^k)_{u_{Radau5}}$ where u_{Gauss6} and u_{Radau5} are solution of (36) with $\delta t = 1 \cdot 10^{-3}$, $\tau_s = 1$, $\Delta T = 2.0 \cdot 10^{-1}$, $\Delta\tau_s = 2.0 \cdot 10^{-1}$, $k = 3$, $n = 4$

In order to adjust $\lim_{z\to-\infty} R(z)$, we use the Theta-method, presented in (29). From (30) we easily find that $\theta = 1/3$ gives $\lim_{z\to-\infty} |R(z)| = 1/2$, and $\theta > 1/3$ gives $\lim_{z\to-\infty} |R(z)| > 1/2$

A simulation with $\delta t = 1 \cdot 10^{-3}$, $\Delta T = 2.5 \cdot 10^{-2}$ and Gauss6 and $\mathcal{F}_{\Delta T}$ gave the results found in Figure 24 Already for $\theta = 0.345$, a small divergence is visible. This test provides numerical support for Theorem 18.

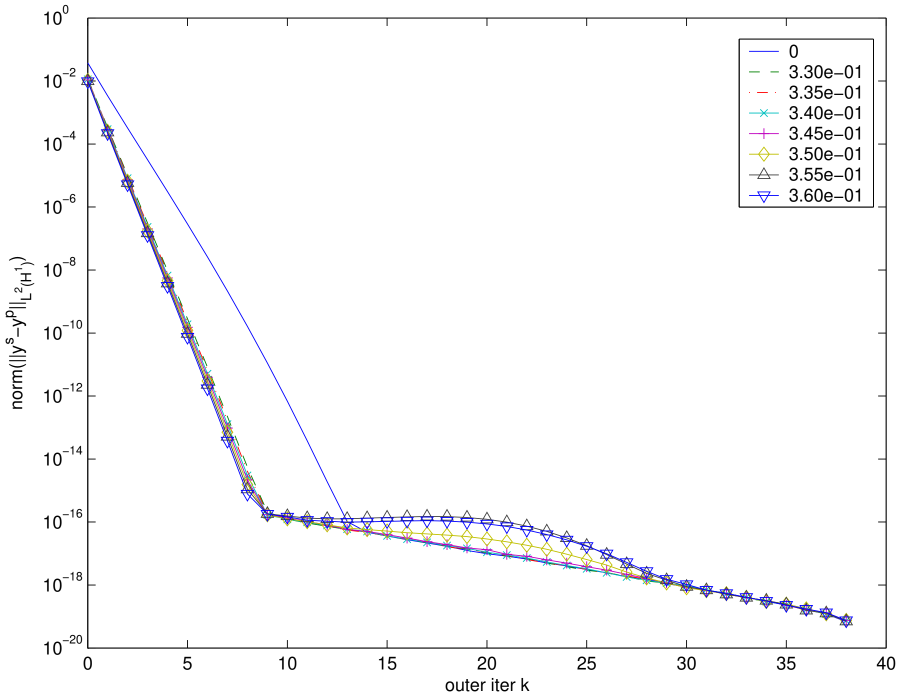


Figure 24: Simulation of (36) using theta method (29). $\delta t = 1 \cdot 10^{-3}$, $\tau_s = 1$, $\Delta T = 2.5 \cdot 10^{-2}$, $\Delta\tau_s = 2.5 \cdot 10^{-2}$. Legend shows the different θ-values

6.2 Periodic solutions

We want to solve (31), using the iterative time-scheme. (31) is an example of an autonomous 2nd order differential equation.

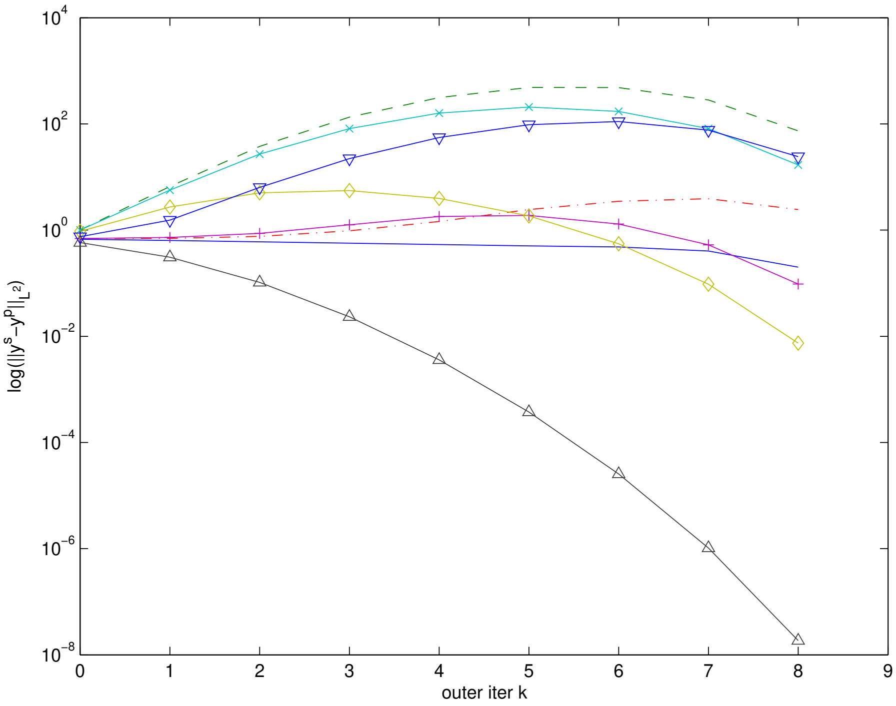
High oscillations, like we experience here, is also an example of a stiff system in the way that an explicit solver requires an enormous stepsize-reduction in order to be stable. But an interesting fact is that even though our implicit solvers are stable, they need approximately the same number of steps in order to be *accurate* (with same order). A certain minimum stepsize is required in order to resolve the systems "physics". With "physics" it should be understood the physical behaviour the solution tries to model. We should therefore expect $\mathcal{G}_{\Delta T}$ to be stable, but inaccurate. How will this effect the algorithm? Another aspect for this test-equation is whether $\mathcal{G}_{\Delta T}$ is symplectic or not has an impact on the convergence.

But what about Theorem 18. We remember that one of the assumptions was that $|z| \gg 1$. But $z = \mu\Delta T$ where μ is pure imaginary for problem (32). It should be obvious that Theorem 18 do not apply for this kind of problems.

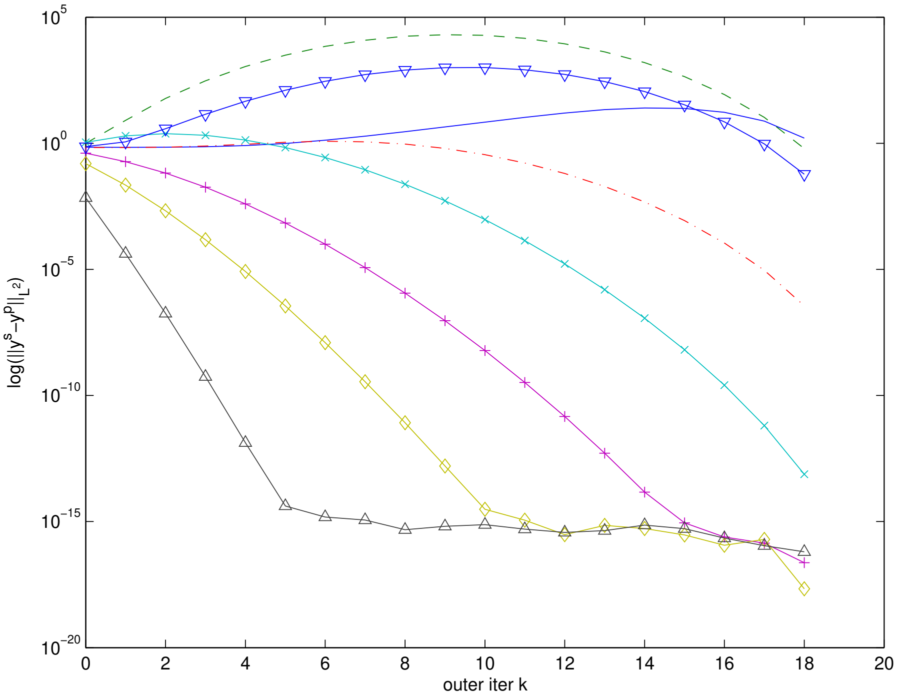
We will here use an optimal solver for $\mathcal{F}_{\Delta T}$, the 5th order Nyström scheme in Table 2. It is optimal in the sense that it only needs four explicit function evaluations to achieve fifth order. It is not symplectic (is should be for large time-domains), but tests indicates that the error because of non-symplectic behaviour is neglectiable for our test problem with the chosen values for δt. We set our domain to $t \in [0, 1]$. $\mathcal{G}_{\Delta T}$ is altered according to Table 11.

The interesting aspect with this test-equation is to see how well the algortihm calculates the oscillations. We therefore choose $\omega = 10 \cdot 2\pi$, which gives us 10 complete periodes on the domain, and therefore $\tau_s = 0.1$. $\delta t = 2.5 \cdot 10^{-4}$.

In Figure 25(a), $\tau_s = \Delta T$. The only scheme that shows any sign of convergence is Gauss8. But

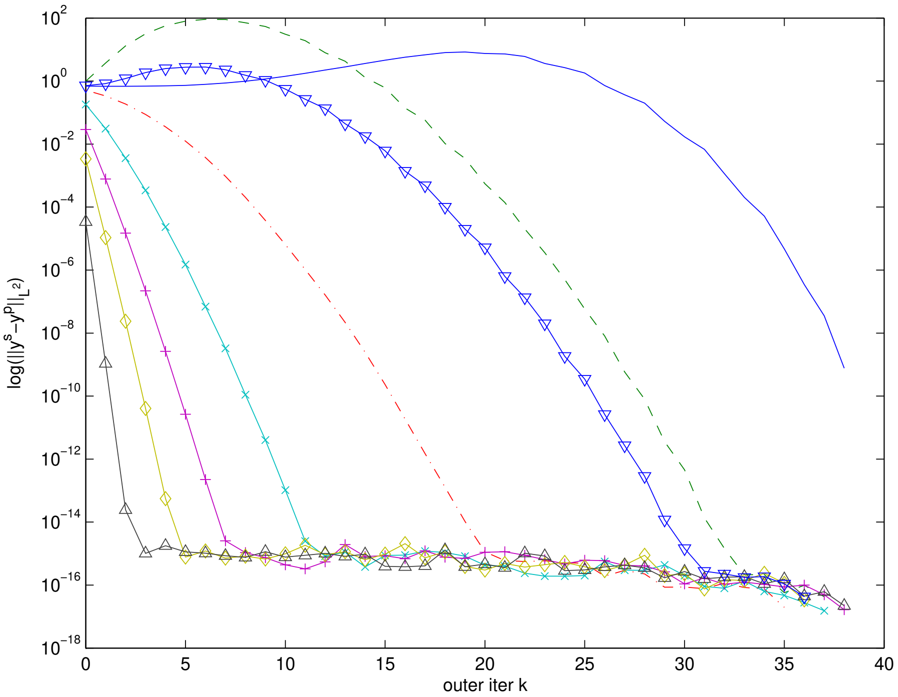


(a) $\Delta T = 0.1 = \tau_s$. One sample-point per period ($\Delta\tau_s = 1$)

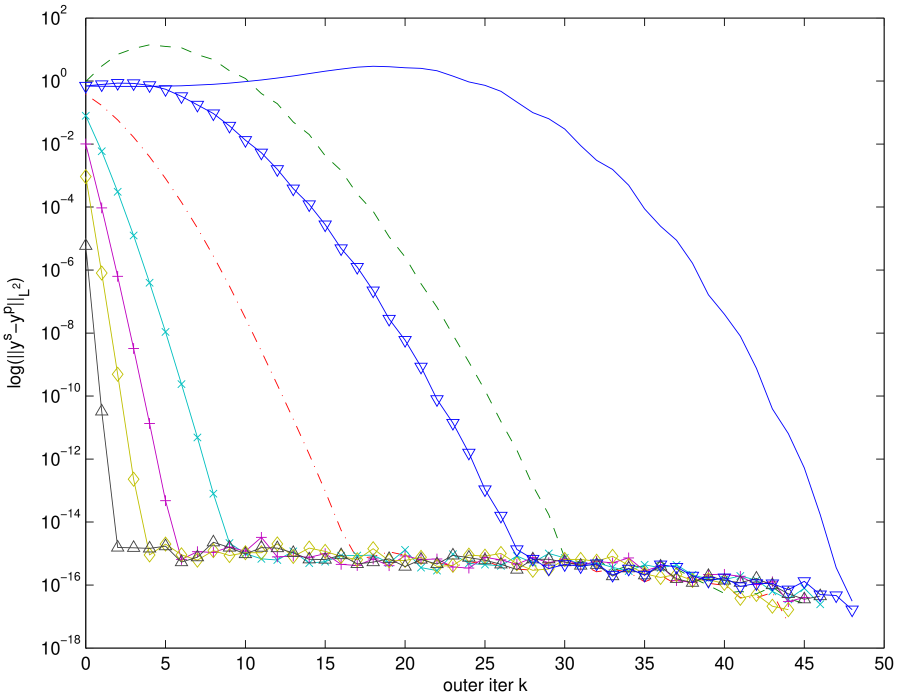


(b) $\Delta T = 0.05 = \tau_s/2$. Two sample-points per period ($\Delta\tau_s = 0.5$)

Figure 25: Solution of (32). $\delta t = 2.5 \cdot 10^{-4}$, $\tau_s = 0.1$, Legend found in Table 11



(a) $\Delta T = 0.025 = \tau_s/4$. Four sample-point per period ($\Delta\tau_s = 0.25$)



(b) $\Delta T = 0.02 = \tau_s/5$. Five sample-points per period ($\Delta\tau_s = 0.2$)

Figure 26: Solution of (32). $\delta t = 2.5 \cdot 10^{-4}$, $\tau_s = 0.1$, Legend found in Table 11

we have to remember that this is a 4-stage scheme. so in a way, Gauss8 has 4 sample-points per periode. None of the other schemes shows any particular sign of convergence. But notice that they are still correct in the maximum number of iterations (the last point is not plotted since it is equal to zeros in the machine precision.

For smaller ΔT the convergence improves, also for the lower order schemes (Figure 25(b) and Figure 26). This test indicates that the system limits the choice of ΔT, in order to achieve fast convergence. The Gauss schemes doesn't show any sign of being better then the Radau and SDIRK schemes (if we consider the difference in order), so it seem that symplectic behaviour for $\mathcal{G}_{\Delta T}$ isn't an inportent propertie.

6.3 Periodic solutions with two different frequences

We now want to solve (33), using the iterative time-scheme. In general, (33) differ from (31) by a force-term. From Section 4.1 it is known that the solution of (33) is a sum of two sine terms with different frequencies. An interesting observation is that if we switch values between ω_1 and ω_2, (34) changes while the exact solution is the same. It would be interesting to see how this effects the parareal algorithm.

As for the previous problem, we use the Nyström scheme from Table 2 as $\mathcal{F}_{\Delta T}$, with $\delta t = 1 \cdot 10^{-4}$. $\mathcal{G}_{\Delta T}$ is altered according to Table 11. In Figure 27-Figure 30, all the (a) figures will be with $\omega_1 = 10 \cdot 2\pi$,and $\omega_2 = 2.1\pi \cdot \omega_1$, which gives $\omega_1 < \omega_2$. All the (b) figures will have the same values for ω, but opposite: $\omega_2 = 10 \cdot 2\pi$,and $\omega_1 = 2.1\pi \cdot \omega_2$, which gives $\omega_1 > \omega_2$. The systems time constants are $\tau_{ss} = 0.1$ and $\tau_{sf} \approx 0.015$, where τ_{ss} is the slowly frequence and τ_{sf} is the fast frequence. $\Delta\tau_s$ is here given relative to the ω_1.

The reason why the ratio between ω_1 and ω_2 is $2.1 \cdot \pi$ is to be sure that we do not experience som mysterious effects because of integer ratio.

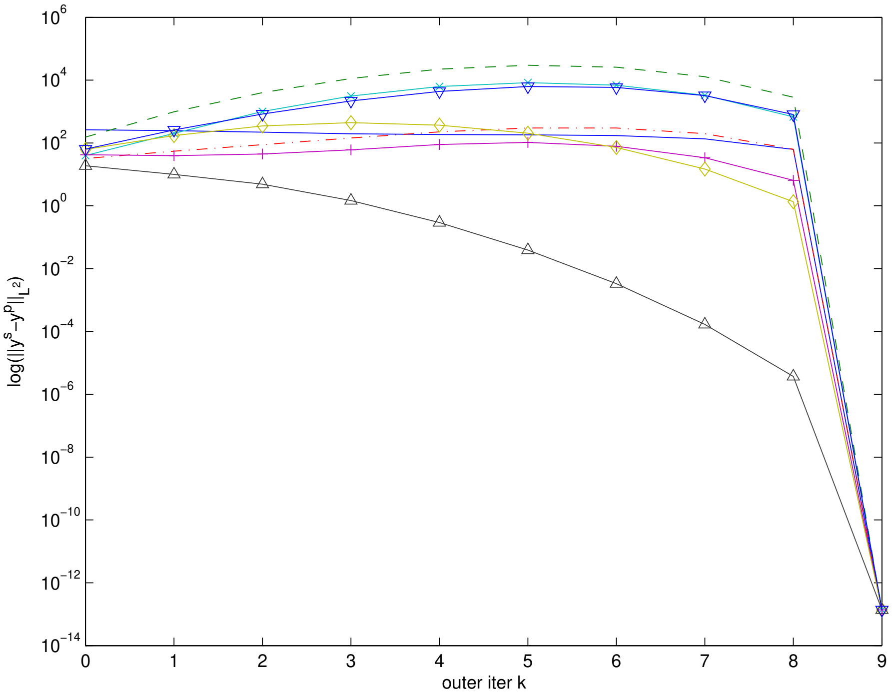
Already in Figure 27 we notice a difference in convergence between the two systems. In Figure 28 it is clear that the system with the slow frequence in the system matrix and the fast frequence in the force term is considerable better the the opposite choice. Remember that the solution of this two systems are the same!

Notice that Gauss6 (as the only scheme) converge in Figure 28, but diverge in Figure 29. Actually, in Figure 27(b)-Figure 29(b) is seems that order has nothing to do with the convergence. This is due to aliasing in the solution of $\mathcal{G}_{\Delta T}$. In Figure 31 we plot $\mathcal{G}_{\Delta T}(\lambda^0)$ for Radau5, Gauss6 and Gauss8. We see that Gauss6 is "lucky", due to the chosen parameters.

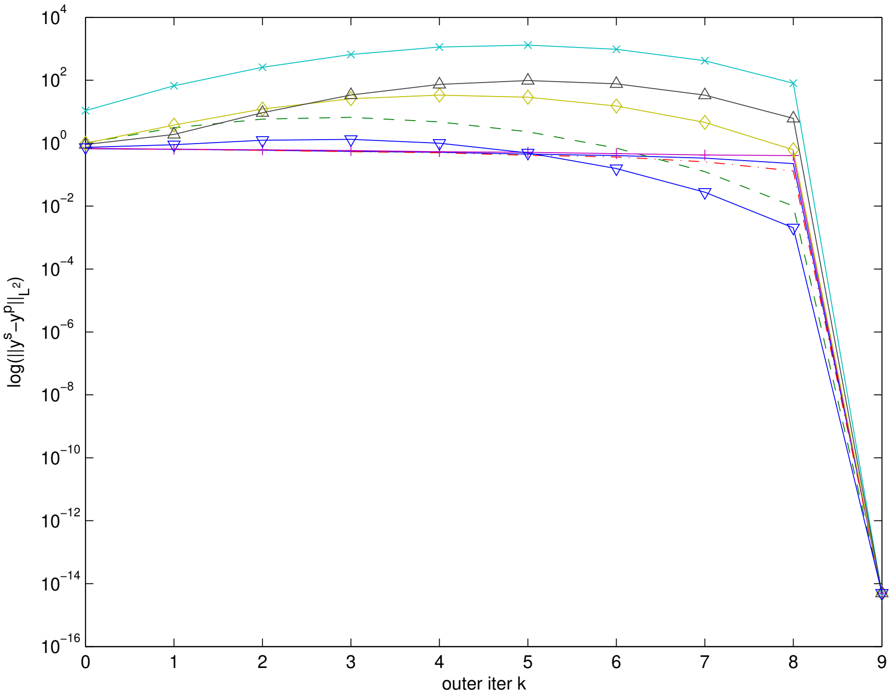
(34) with $\omega_1 = \omega$ and $\omega_2 = 2.1\pi\omega_1$ has just a litle slower convergence then (32) with $\omega = \omega_1$. But (34) with $\omega_2 = \omega$ and $\omega_1 = 2.1\pi\omega_2$ doesn't converge, except for some cases with aliasing, before the eigenvalues in the system matrix is sampled with at least one sample-point per periode (given by the system matrix). This is seen in Figure 30(b). For smaller values of ΔT, all the schemes will eventually converge. Beware that Figure 30 is not iterated to maximun number of iterations.

The different convergence is due to what kind of physics we try to approximate. In the case where $\omega_1 < \omega_2$, we try to approximate the slowest oscillations, while the force-term with the high frequences is undersampled. But the force-term doesn't destroy the solution, even though it is poorly estimated. This is because teh force term is independent of y, and consequensly approximated using ordinary quadrature in Gauss-points. It is well known that smooth functions are approximated wery vell using various Gauss quadratures. In the case where $\omega_1 > \omega_2$ we try to estimate the fast oscillation. But since this is heavily undersampled, we experience grave inaccuracies. The force-term is estimated with fair accuracy. But it's contributions in the terms of error is neglectiable.

It is quite clear that we have to bear in mind what we are capable of estimating with large time-steps, when we construct our system. A natural expantion of this is to say that we could

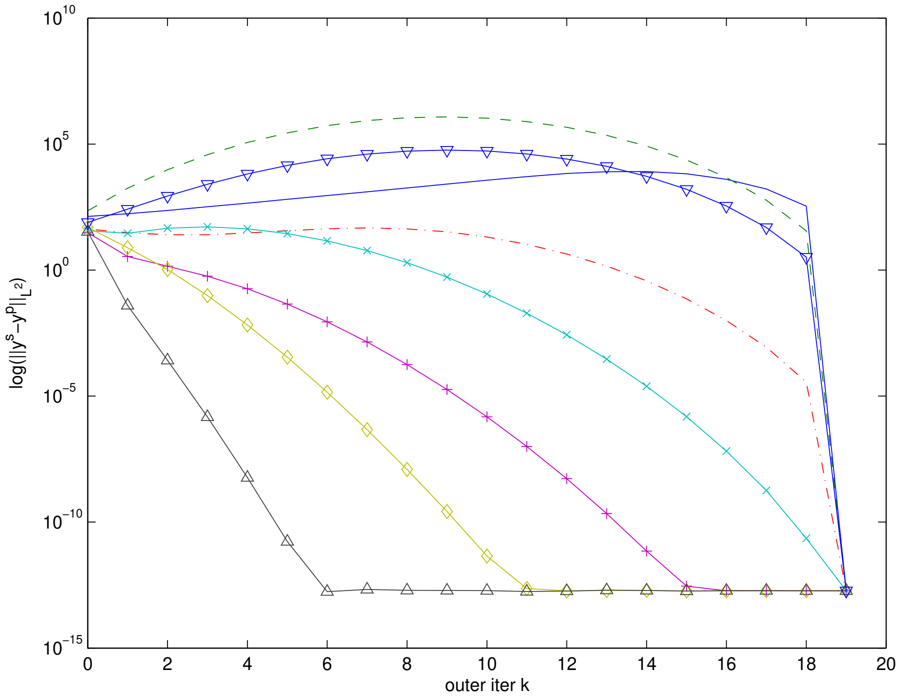


(a) $\Delta T = 0.1$, $\omega_1 = 10 \cdot 2\pi$, $\omega_2 = 2.1\pi \cdot 10 \cdot 2\pi$, $\Delta\tau_s = 1$

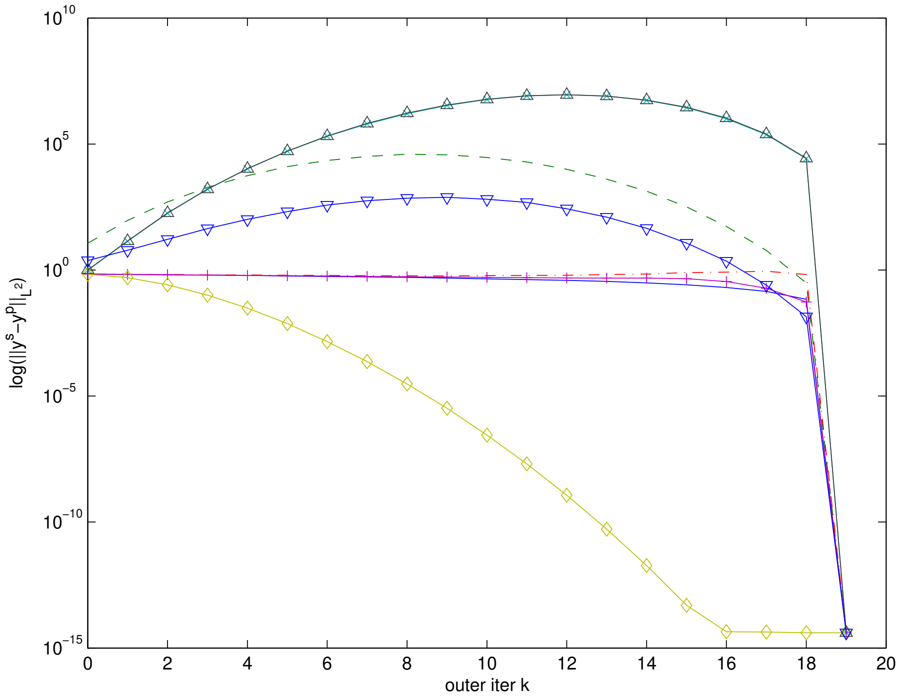


(b) $\Delta T = 0.1$, $\omega_2 = 2.1\pi \cdot 10 \cdot 2\pi$, $\omega_2 = 10 \cdot 2\pi$, $\Delta\tau_s = 6.6$

Figure 27: Solution of (34) with $\delta t = 1 \cdot 10^{-4}$, $\tau_{ss} = 0.1$ and $\tau_{sf} \approx 0.015$. Legend found in Table 11

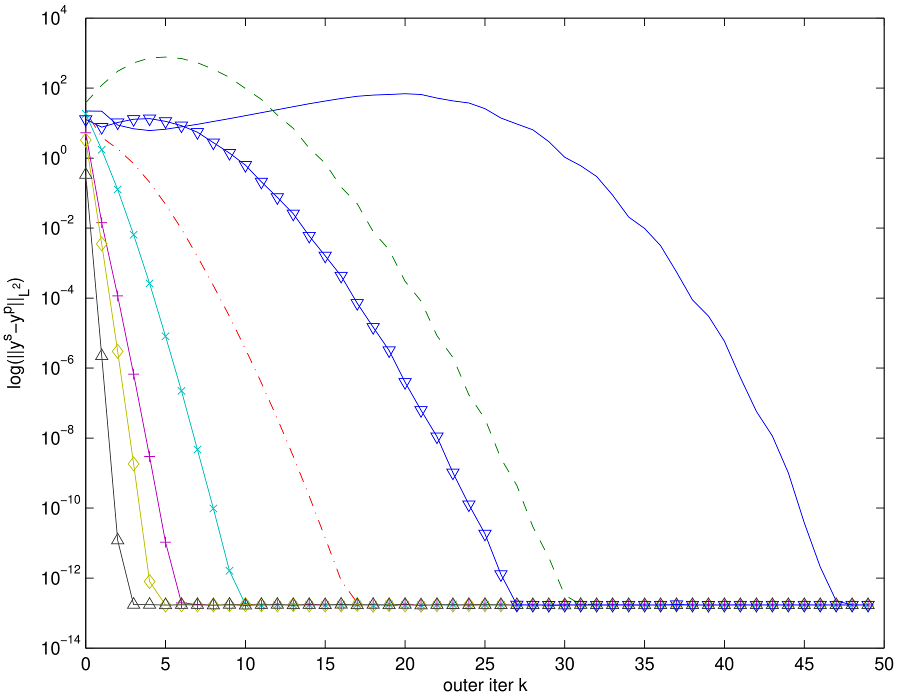


(a) $\Delta T = 0.05$, $\omega_1 = 10 \cdot 2\pi$, $\omega_2 = 2.1\pi \cdot 10 \cdot 2\pi$, $\Delta\tau_s = 0.5$

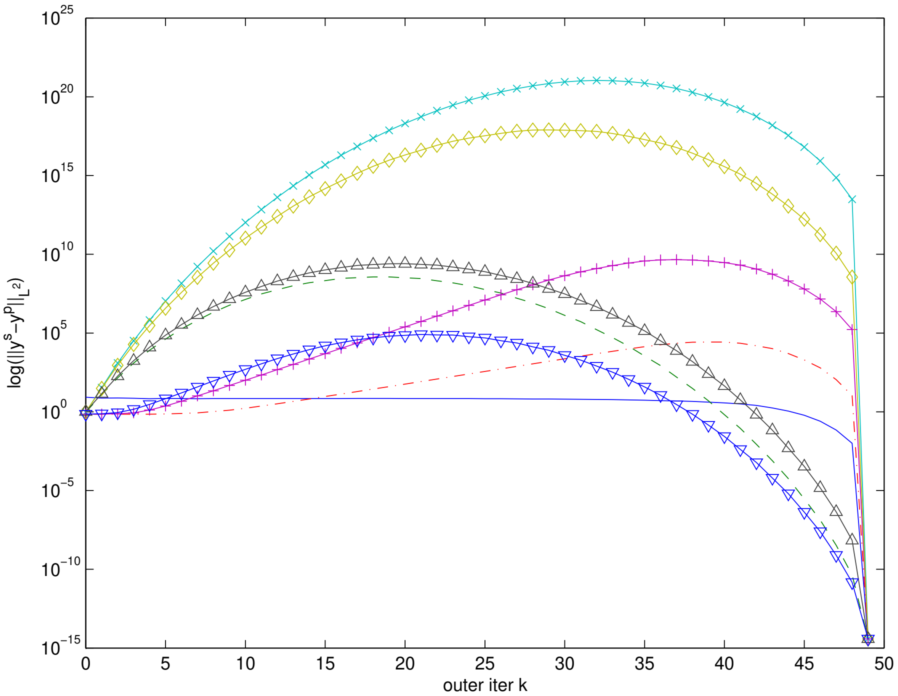


(b) $\Delta T = 0.05$, $\omega_2 = 2.1\pi \cdot 10 \cdot 2\pi$, $\omega_2 = 10 \cdot 2\pi$, $\Delta\tau_s = 3.3$

Figure 28: Solution of (34) with $\delta t = 1 \cdot 10^{-4}$, $\tau_{ss} = 0.1$ and $\tau_{sf} \approx 0.015$. Legend found in Table 11

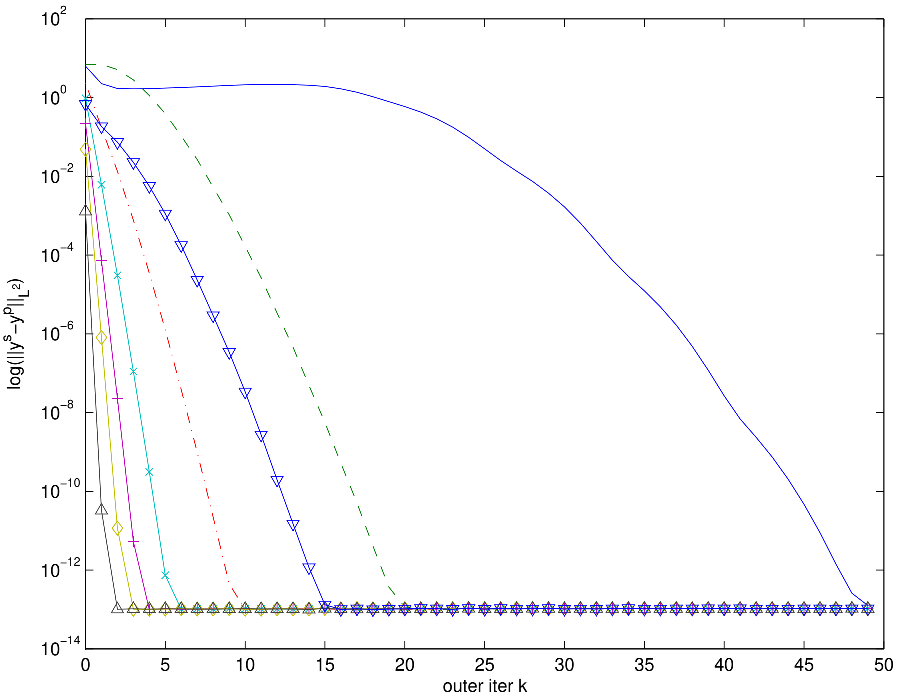


(a) $\Delta T = 0.02$, $\omega_1 = 10 \cdot 2\pi$, $\omega_2 = 2.1\pi \cdot 10 \cdot 2\pi$, $\Delta\tau_s = 0.2$

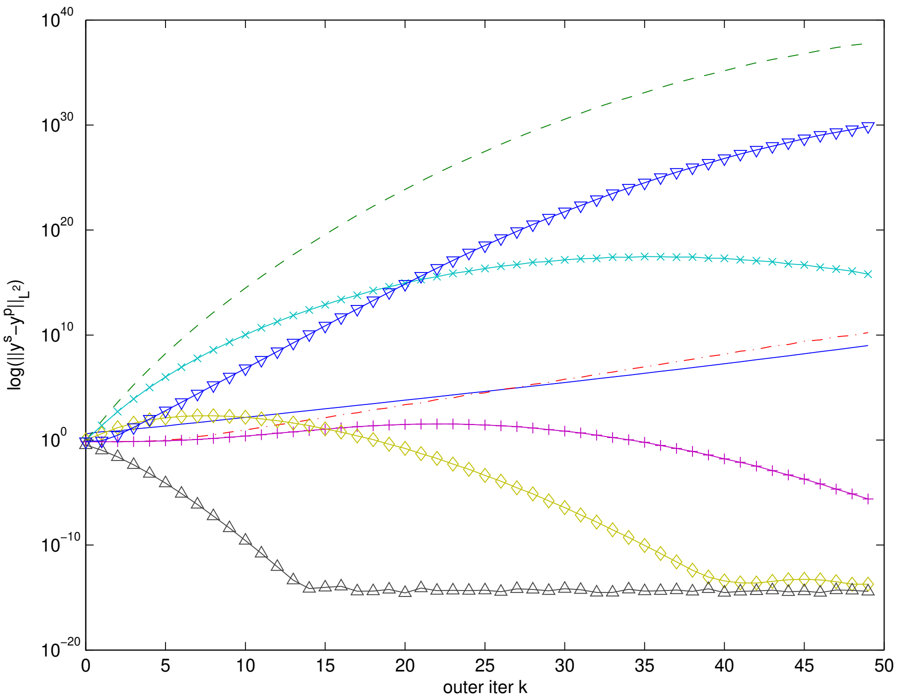


(b) $\Delta T = 0.02$, $\omega_2 = 2.1\pi \cdot 10 \cdot 2\pi$, $\omega_2 = 10 \cdot 2\pi$, $\Delta\tau_s = 1.3$

Figure 29: Solution of (34) with $\delta t = 1 \cdot 10^{-4}$, $\tau_{ss} = 0.1$ and $\tau_{sf} \approx 0.015$. Legend found in Table 11



(a) $\Delta T = 0.01$, $\omega_1 = 10 \cdot 2\pi$, $\omega_2 = 2.1\pi \cdot 10 \cdot 2\pi$, $\Delta\tau_s = 0.1$



(b) $\Delta T = 0.01$, $\omega_2 = 2.1\pi \cdot 10 \cdot 2\pi$, $\omega_2 = 10 \cdot 2\pi$, $\Delta\tau_s = 0.66$

Figure 30: Solution of (34) with $\delta t = 1 \cdot 10^{-4}$, $\tau_{ss} = 0.1$ and $\tau_{sf} \approx 0.015$. Legend found in Table 11

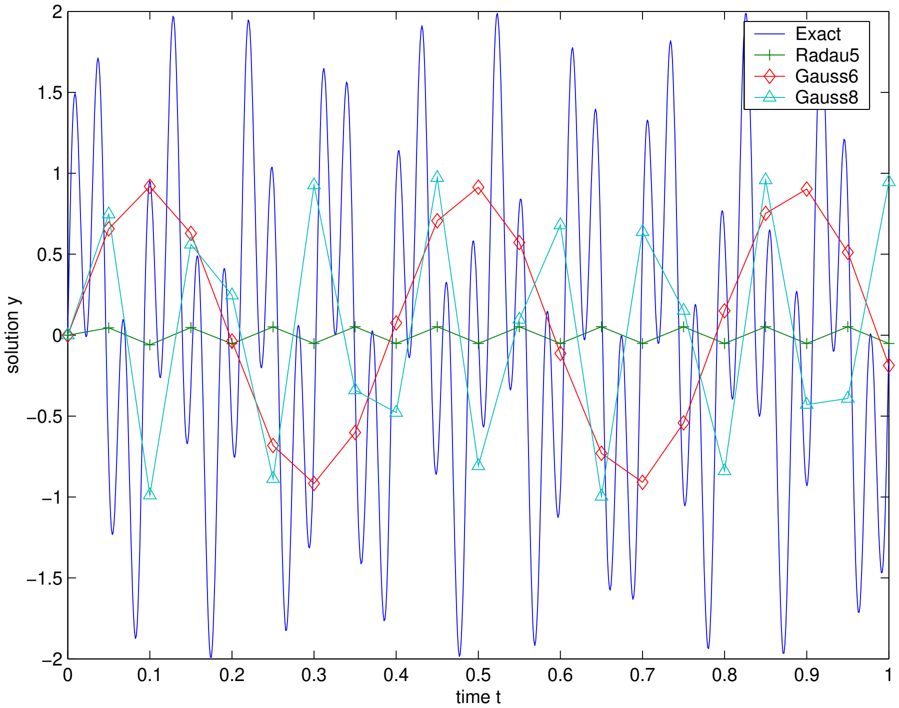


Figure 31: Solution of (34). $\delta t = 0.05$ for Radau5, Gauss6 and Gauss8 $\tau_{ss} = 0.1$ and $\tau_{sf} \approx 0.015$.

have used $\mathcal{G}_{\Delta T}$ on a "coarce physic", meaning we could have omitted the fource-term. If the amplitude of the fast oscillations had been smaller then the amplitude of the slow oscillations, this would have been an even safer simplification of the system.

6.4 Heat-equation with periodic behaviour in time

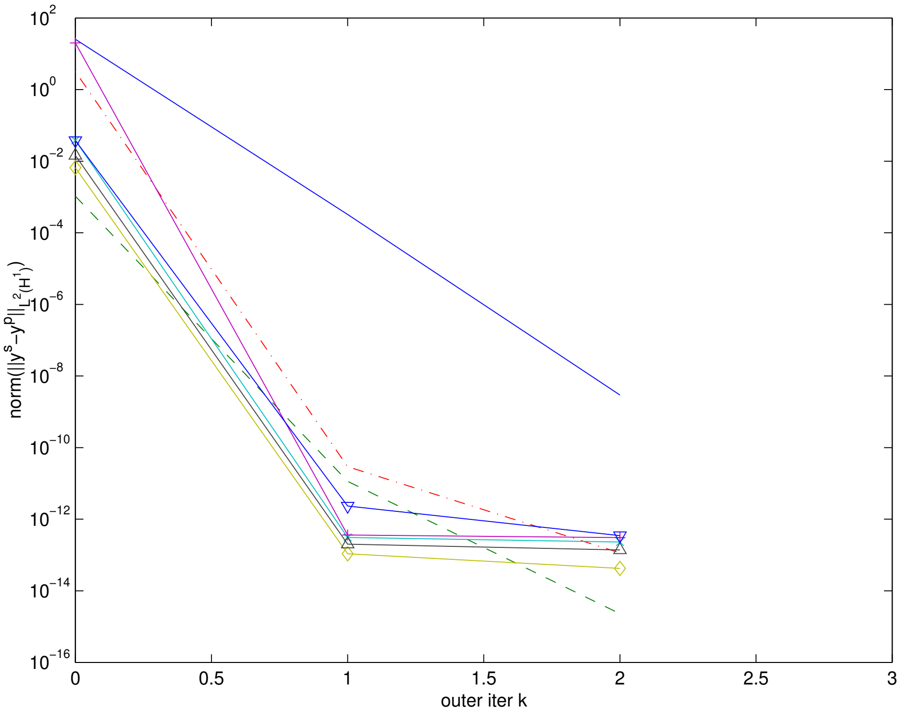
An interesting extension of the Heat-equation is (37), which has periodic behaviour in time. The question is whether we can apply our knowledge from Section 6.3 or not.

Theorem 18 do apply here since the eigenvalues are real, and large ($\lambda_{max} = -4.102 \cdot 10^3$). We can then expect all the Gauss schemes, and the SDIRK3 scheme to diverge for large n and k. From Section 6.3 we learned that the force-term doesn't effect the convergence to any extent. Actually, since the force-term is a product of sine in time and space, we expect an extreamly fast convergence since the calculation of the force-term is simply an interpolation using different Gauss-quadrature. This, we know, is very accurate on a smooth function like sine.

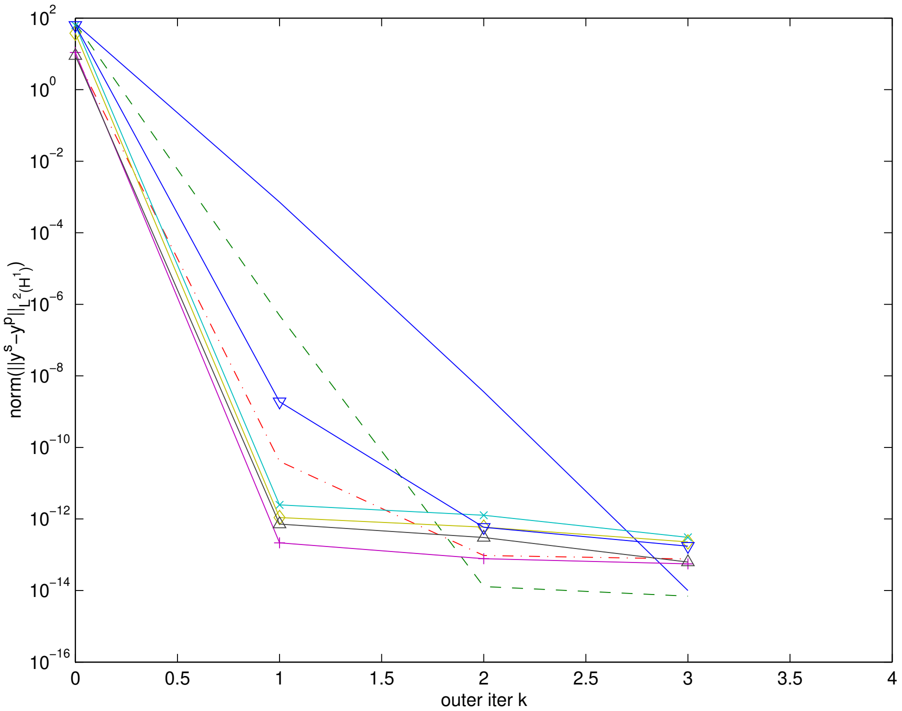
The simulaton is done using 20 equal size FE. $\delta t = 2.4 \cdot 10^{-4}$, $\tau_s = 0.1$. $\mathcal{G}_{\Delta T}$ and $\mathcal{F}_{\Delta T}$ are the same during all the tests. The results can be seen in Figure 32 and Figure 33.

From Figure 32 and Figure 33 we notice that for all schemes (maybe except EB), we have convergence after only one iteration of the predictor-corrector scheme. Notice in Figure 33(b) that we, for Gauss8, have an error of about 10^{-7} after only the initial run. This is with only two sample-points per periode ($\Delta T = 0.05$) in time. But we have to remeber that Gauss8 uses four quadrature-points per time-step, leaving it with eight quadrature-points per periode. So the results are not that unexpected.

In Figure 33(a) ($\Delta T = 0.1$) the algorithm starts to diverge for all schemes but the Radau schemes. This is consistent with Theorem 18, and with the result from Section 6.1.

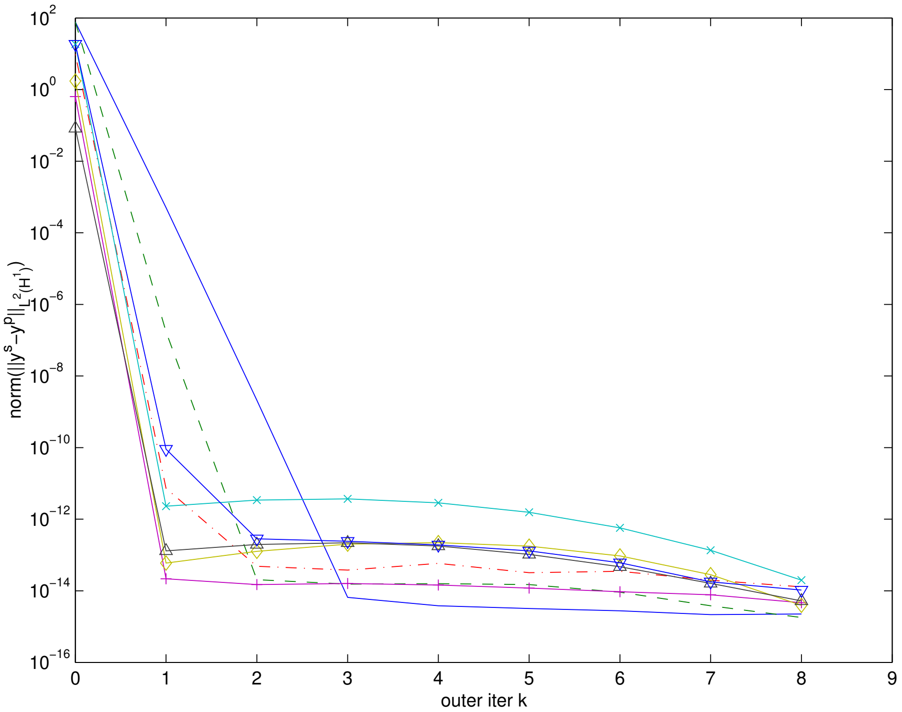


(a) $\Delta T = 0.25$, $\Delta\tau_s = 2.5$

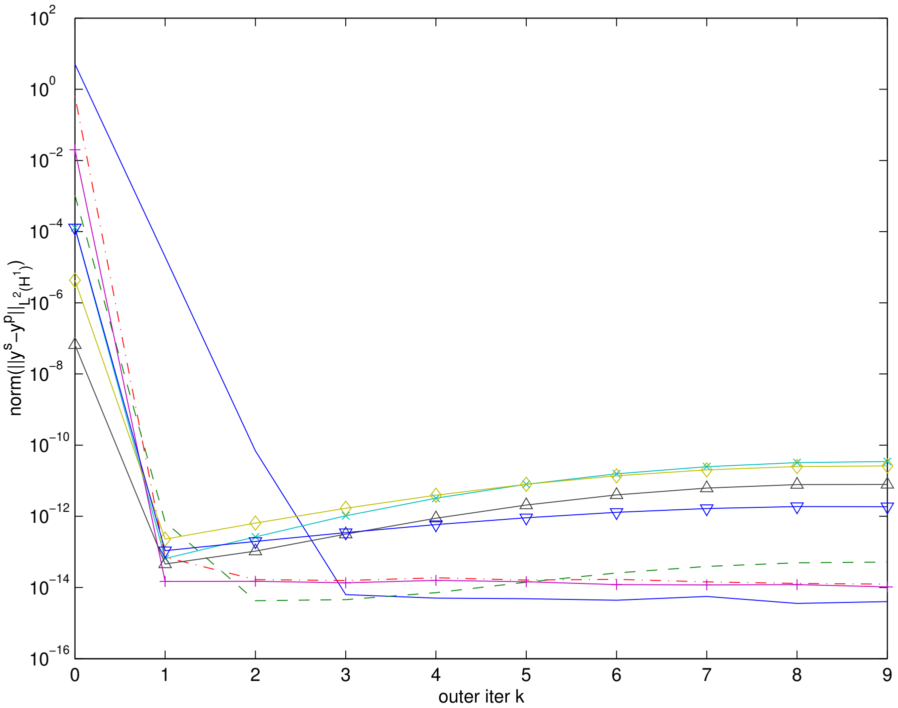


(b) $\Delta T = 0.2$, $\Delta\tau_s = 2$

Figure 32: Solution of (37) with $\delta t = 2.5 \cdot 10^{-4}$ and $\tau_{ss} = 0.1$. Legend found in Table 11



(a) $\Delta T = 0.1$, $\Delta\tau_s = 1$



(b) $\Delta T = 0.05$, $\Delta\tau_s = 0.5$

Figure 33: Solution of (37) with $\delta t = 2.5 \cdot 10^{-4}$ and $\tau_{ss} = 0.1$. Legend found in Table 11

6.5 van der Pol

So how will the parareal algorithm handle a stiff nonlinear differential equation. It is obvious that we can not use Theorem 18, since van der Pol (vdP) is a nonlinear. But vdP har periodic nature. We may expect some similarity to the periodic solutions, that is $\mathcal{G}_{\Delta T}$ have to approximate the general physics in the system to a certain extence to achieve convergence.

As described in Section 3.3, we apply Newton's method to solve the nonlinear system we recive using our implicit ode-solvers. But newton is not guarantied fast convergence unless the initial guess is fairly close to the solution. Since our nonlinear system is of order higher then one, it may also converge to the wrong solution. It is therefore necessary to allow the solver to decrease stepsize in order to assure convergence. The consequence is that we loose the control of the time-use for $\mathcal{G}_{\Delta T}$ (also for $\mathcal{F}_{\Delta T}$, but it is not so determining since the step-size is small already, and the convergence is usually fast). Tests have also shown that in some cases, the iteration converges to the wrong solution.

As a result, Table 12 displays the number of iterations used by $\mathcal{G}_{\Delta T}$ for Figure 34 and Figure 35. The number displayed is the number used to calculate $\mathcal{G}_{\Delta T}(\Lambda^0)$. Since Λ^k vary for different k, the number of iterations will also vary. But The difference is not significant.

Van der Pol's equation (43) is solved with $\mu = 10$ and $t \in (0, 100)$. This includes about five periodes (remember that vdp has periodic nature). $\delta t = 1 \cdot 10^{-3}$ and $\tau_s \approx 20$.

ΔT	Gauss2	Gauss4	Gauss6	Gauss8	EB	Radau3	Radau5	SDIRK3	Suggested
10	124	39	468	324	45	59	631	146	10
5	453	47	715	321	74	69	688	125	20
2	710	65	451	320	210	89	668	165	50
1	694	122	442	317	243	134	692	195	100

Table 12: Number of timesteps used by $\mathcal{G}_{\Delta T}(\Lambda^0)$ for different values of ΔT. Plot seen in Figure 34 and Figure 35

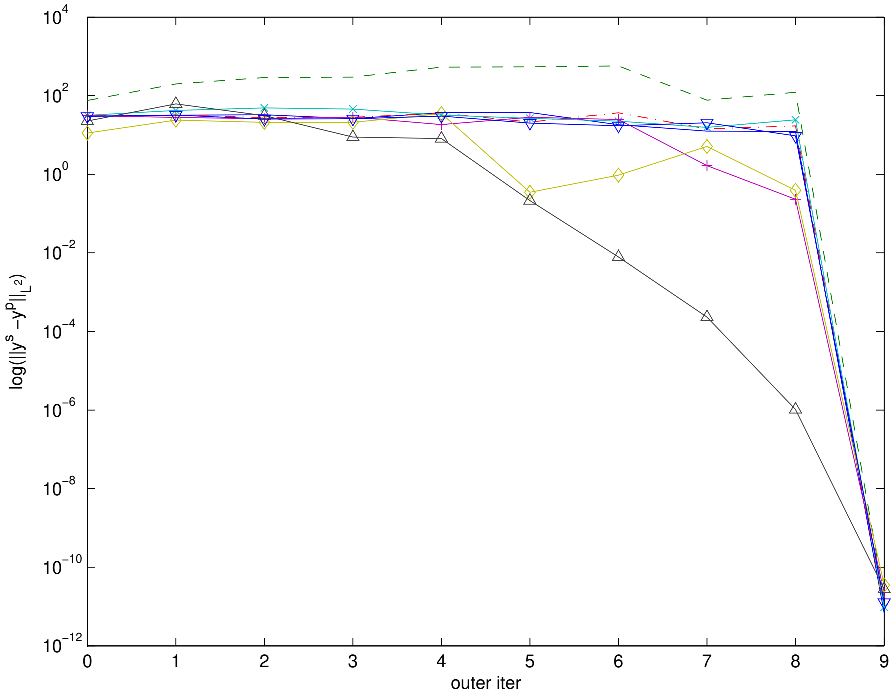
The first we notice in Table 12 is that most of the solvers use considerably more steps then suggested (suggested equals n, the number of partitions of the time-domain). Notice also that some actually use fever timesteps for smaller ΔT. This indicates that the nonlinear solver is not very good. This prevents us from drawing any final conclusions, especially on comparison between different solvers and convergence-rate depending on ΔT. But the results will still indicate whether the algorithm seems to work or not.

In Figure 34 and Figure 35 we notice that the high order schemes Radau5, Gauss6 and Gauss8 shows good sign of convergence. But they do also use a large number of steps, especially Radau5. Some of the solvers perform really bad, e.g Gauss4 and SDIRK3. But we also notice that they use few timesteps relative to e.g Gauss6. Tests have shown that both Gauss4 and SDIRK3 have a tendency of converging to the wrong solution in the region where vdp changes rapidly.

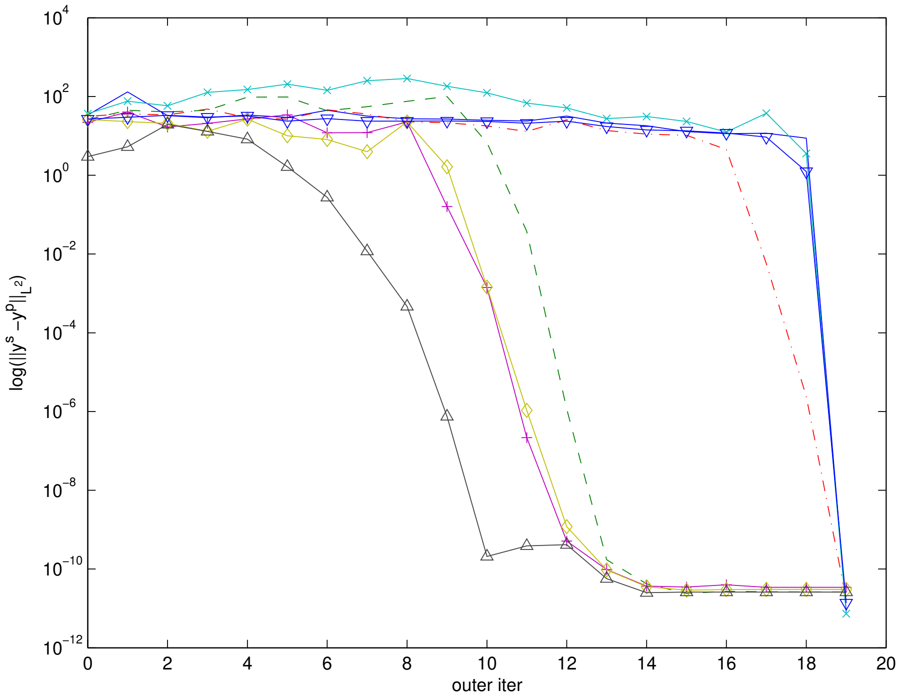
Another important propertie is the loss of order due to newton-iteration. According to [13], this order-reduction is quite severe.

Idealy $\mathcal{F}_{\Delta T}$ is a adaptiv step-size solver. But this will result in a grave difference in computational expence for the partitions of $\mathcal{F}_{\Delta T}$ since some may compute on a stiff intevall, while others may compute on a non-stiff intervall.

For comparison it can be tested that `Matlab`'s own stiff-solver, `ode15s`, uses less then $2.5 \cdot 10^5$ steps for the same problem with approximate half the error as Radau5 using $1 \cdot 10^6$ timesteps. Clearly variable timestep is advantageous for this problem, something that will decrease the speedup when computational expence is compared to an optimal seriel method with adaptive

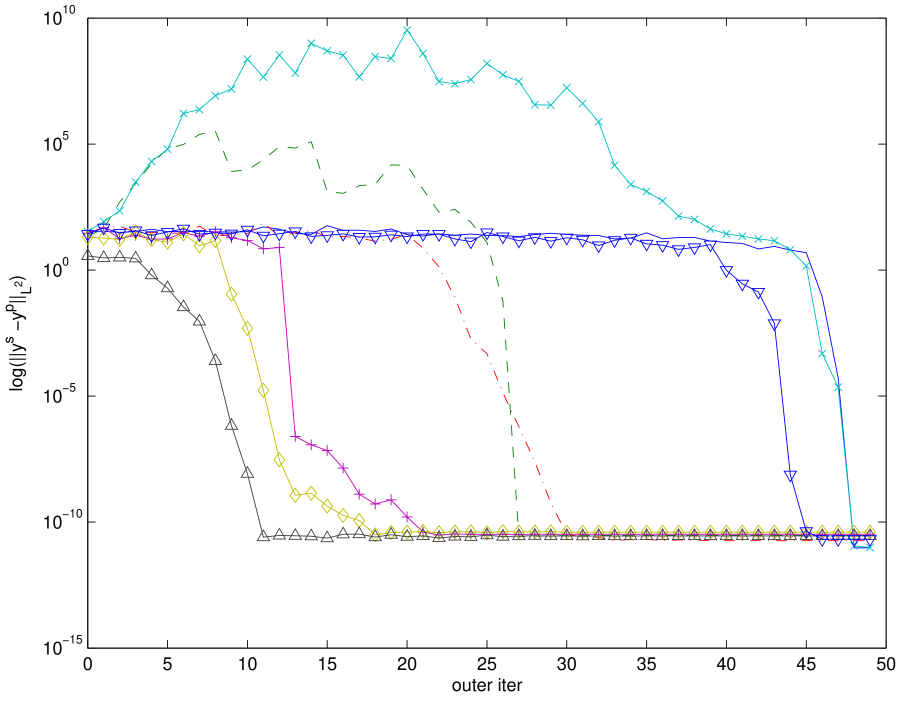


(a) $\Delta T = 10$, $\Delta\tau_s \approx 0.5$. Iteration count found in Table 12.

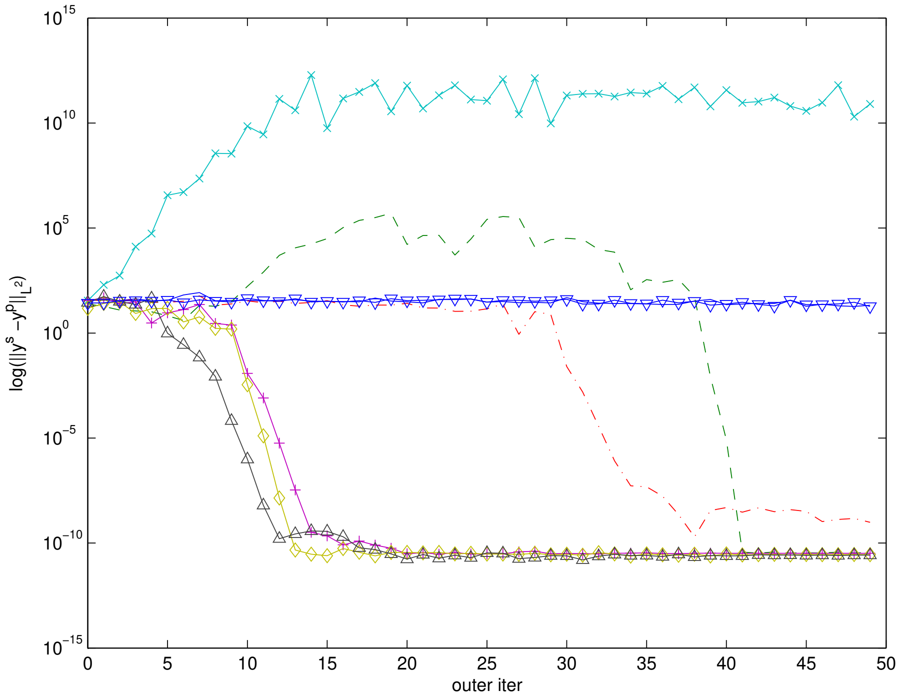


(b) $\Delta T = 5$, $\Delta\tau_s \approx 0.25$. Iteration count found in Table 12.

Figure 34: Solution of (44) with $\delta t = 1 \cdot 10^{-3}$ and $\tau_s = 20$. Legend found in Table 11



(a) $\Delta T = 2$, $\Delta\tau_s \approx 0.1$. Iteration count found in Table 12.



(b) $\Delta T = 1$, $\Delta\tau_s \approx 0.05$. Iteration count found in Table 12.

Figure 35: Solution of (44) with $\delta t = 1 \cdot 10^{-3}$ and $\tau_s = 20$. Legend found in Table 11

step-size.

6.6 Implementation

About 7000 code-lines are produced in order to calculate the various test-equations with the different schemes, and produce all the plots in this report. The source-code will not be included; both because its volume and since it is of no interest in itself. The implementation is done on basis of Section 3 and Section 4, so it should be clear how everything is done (except small programming details).

The only procedure that need special explanation is the Newton-iteration. From Section 3.3 it was clear that the iteration not necessarly converge for all starting-values. A normal solution to this is to decrease (half) the step-size when slow convergence are detected. Assume we want to solve the differential equation from t_n ot t_{n+1}. The algorithm for the newton-iteration is in general implemented like Algorithm 2.

Algorithm 2 The newton iteration

```
t_start ← t_n
t_end ← t_{n+1}
iter ← 0
while true do
  iter ← iter + 1
  Perform one newton-iteration
  if convergence then
    t_start ← t_end
    if t_start == t_{n+1} then
      store solution
      break
    end if
    store temporary solution
    t_end ← t_{n+1}
  end if
  if iter==10 then
    t_end = (t_start + t_end)/2
    break
  end if
end while
```

The algorithm allways tries to take maximun step-size.

All the code-lines have been produced by the author, execpt some procedures in the element-solution, providing rutines for gridgeneration, local-to-global mapping, assembling stiffness- and mass-matrixes. These have been taken from the course SIF5050, *Numerical Solution of Partial Differential Equations using Element Methods*, held by the advisor of this project. `Matlab`-rutines have of course been used in a great extent.

7 Conclutions

A theorem stating a restriction in the strong A-stability propertie of $\mathcal{G}_{\Delta T}$ in order for the predictor-corrector scheme to be stable for aotunomous differential equations, has been proposed. During several tests, Theorem 18 has been substantiated. A special test, the Theta-test, has also been applied for the ordinary one-dimentional heat equation. This test also confirms Theorem 18, and indicates that the property $\lim_{z\to -z} R(z)$ for $\mathcal{G}_{\Delta T}$ is quite accurate.

It seems clear that the systems physic sets a limitation in the minimun choice of ΔT in order to achieve convergence. The testproblems, especially (34), indicates that the choice of system for the $\mathcal{G}_{\Delta T}$ is importent in order to estimate the physics in an optimal way. The two different formulation of the same system gave a dramatic difference in convergence.

Tests on the nonlinear stiff problem van der Pol pointed out new problems. How should the nonlinear system be solved when unnatural large stepsize is applied? We were unable to solve this properly in this report. But nothing in the tests suggest that this problem is nonsolvable using the parareal algorithm.

For all problems, we have experienced increase in convergence by the use of high order schemes for $\mathcal{G}_{\Delta T}$. This is particular interesting when we know that it is possible to parallelize $\mathcal{G}_{\Delta T}$ using other parallel schemes. After all, there is one processor that has to calculate the predictor-corrector scheme. Why not put the other processors to work. Parallelizing $\mathcal{G}_{\Delta T}$ may prevent increase in computational cost for high order schemes, while the number of iterations required for convergence will decrease.

We have also demostrated that the choice of $\mathcal{F}_{\Delta T}$ can be a highly optimal solver (not the same as $\mathcal{G}_{\Delta T}$) – your favorite solver.

7.1 Future work

During the work of this projekt, several other interesting aspects of the parareal algorithm was discovered. We will here recommend that the following is investigated futher:

In the stability expression for the predictor-corrector scheme applied on an autonomous differential equation, an expression for the constants attached to e^z should be evolved. A possible approach is to expand the graph structure from Section 5 to a graph dimentional tree. This should be done in order to find how small e^z should be in order for the assumptions of Theorem 18 to hold.

The theorem should be extended to include both non-autonomous and nonlinear differential equations. For the non-autonomous differential equation, a possible approach is to exchange $R(z)$ with $K(Z)$, where $K(Z)$ is defined in Section 3.1.2. From this it should be possible to find a test property involving the Runge-Kutta weights (b) and coefficients (A). It is possible to show that for non-confluent Runge-Kutta methods (i.e methods with all nodes c_i distinct), the concepts of B-, AN- and algebraicly-stability are equivalent. It should therefore be investigated to verify if this also applies under the new test propertie for non-autonomous stability of the predictor-corrector scheme. If so, we also have nonlinear stability when non-autonomous stability is established.

The consequences of applying G on another system (i.e. coarse in space, and highly oscillating terms removed) should be fully explored. If a coarse space discretization is used, a way of transfering Λ over to the space $\mathcal{F}_{\Delta T}$ works on, should be developed.

Solving stiff nonlinear differential equations should be futher investigated. The problem with nonlinear solver is especially intereting. Simplified Newton should be tested. Also schemes that avoide nonlinear solvers (e.g Rosenbrock-type schemes) should be explored in this context.

The effect of parallelizing $\mathcal{G}_{\Delta T}$, using e.g. parallel implicit Runge-Kutta schemes, should be explored to identify possible additional speedup.

References

[1] L. Baffico, S. Bernard, Y. Maday, G. Turinici, and G. Zérah. Paralell in time molecular dynamics simulations. 2002.

[2] Guillaume Bal and Yvon Maday. A ”parareal” time discretization for non-linear pde's with application to the pricing of an american put. 2002.

[3] Dietrich Braess. *Finite Elements*. Cambridge University Press, 72. edition, 2001.

[4] C.J. Budd and M.D. Piggott. Geometric integration and its applications.

[5] P. Chartier, St Quentin Yvelines, and Rennes B. Phillips. A parallel shooting technique for solving dissipative ode's. *Computing – Springer*, 51:209–236, 1993.

[6] C.W. Gear. The potential for parallelism in ordinary differential equations. Technical Report UIUCDCD-R-87-13246, University of Illinois, 1986.

[7] C.W. Gear. Parallel methods for ordinary differential equations. Technical Report UIUCDCD-R-87-1369, University of Illinois, 1987.

[8] E. Harier, S.P Nørsett, and G. Wanner. *Solving Ordinary Equations I*. Springer Series in Computational Mathematics, 8. Springer, 2. edition, 2000.

[9] E. Harier and G. Wanner. *Solving Ordinary Equations II*. Springer Series in Computational Mathematics, 14. Springer, 2. edition, 2002.

[10] A. Iserles. *A first course in the Numerical Analysis of Differential Equations*. Cambridge Texts in Applied Mathematics. Cambridge, 1996.

[11] A. Iserles and S.P. Nørsett. *Order Stars*. Chapman and Hall, 1991.

[12] K. R. Jackson. A survey of parallel numerical methods for initial value problems for ordinary differential equations. *IEEE Transactions on Magnetics*, pages 3792–3797, 1991.

[13] K. R. Jackson, A. Kværnø, and S. P. Nørsett. An analysis of the order of runge-kutta methods that use an iterative scheme to compute their internal stage values. *BIT*, pages 713–765, 1996.

[14] K. R. Jackson and S. P. Norsett. The potential for parallelism in runge-kutta methods. part 1: Rk formulas in standard form. *SIAM Journal of Numerical Analysis*, Feb 1995.

[15] David Kincaid and Ward Cheney. *Numerical Analysis*. Brooks/Cole Publishing Company, 2. edition, 1996.

[16] Jacques-Louis Lion, Yvon Maday, and Gabriel Turinici. Résolution d'edp par un schéma en temps pararéel. 2000.

[17] Yvon Maday and Gabriel Turinici. A parareal in time procedure for the control of partial differential equations. 2002.

[18] J. M. Ortega and W. C. Rheinboldt. *Iterative Solutions of Nonlinear Equations in Several Variables*. Academic Press, INC, 1970.

[19] Lawrence Perco. *Differential Equations and Dynamical Systems*. Texts in Applied Mathematics, 7. Springer, 3. edition, 2001.

[20] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Numerical Mathematics*. Texts in Applied Mathematics, 37. Springer, 2000.

[21] Einar M. Rønquist. Lecture notes in course ”numerical solution of partial differential equations using element methods”. 2001.

[22] Lawrence F. Shampine. *Numerical Solution of Ordinary Differential Equations.* Chapman and Hall, 1994.

[23] Nicholas Young. *An Introduction to Hilbert Space.* Cambridge University Press, 1988.

A Mathematics

A.1 Linear Algebra

A.1.1 Cramer's rule

The jth component of $x = A^{-1}b$ is

$$x_j = \frac{\det B_j}{\det A}, \quad \text{where} \quad B_j = \begin{pmatrix} a_{11} & \dots & a_{1(j-1)} & b_1 & \dots & a_{1n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n1} & \dots & a_{n(j-1)} & b_n & \dots & a_{nn} \end{pmatrix}.$$

In B_j, the vector b replaces the jth coloumn of A.

A.2 Complex Analysis

Theorem 19 (Maximum modulus theorem for analytic functions) *Let $F(z)$ be analytic and nonconstant in a domain containing a bounded region D and its boundary. Then the absolute value $|F(z)|$ cannot have a maximum at an interior point of D. Consequently, the maximum of $|F(z)|$ is taken on the boundary of D. If $F(z) \neq 0$ in D, the same is true with respect to the minimum of $|F(z)|$.*

A.3 Hamiltonian Systems

We have a system on the general form

$$\dot{x} = X(x,y), \quad \dot{y} = Y(x,y). \tag{52}$$

This system is called a Hamiltonian system if there exist a function $H(x,y)$ such that

$$X = \frac{\partial H}{\partial y}, \quad Y = -\frac{\partial H}{\partial x},$$

and the condition

$$\frac{\partial X}{\partial x} + \frac{\partial Y}{\partial y} = 0$$

is fulfilled. The function $H(x,y)$ is called the Hamiltonian fynction for the system. It can easily be showed that $H(x,y) = \textit{constant}$ along any phase path. It's also possible to prove that the corresponding flow is symplectic, i.e., preserves the differential 2-form

$$\omega^2 = \sum_{i=1}^{n} dx_i \wedge dy_i,$$

where $dx_i \wedge dy_i$ (called the *exterior product*) is a bilinear map acting on a pair of vectors

$$\begin{aligned} (dx_i \wedge dy_i)(\xi_1, \xi_2) &= \det \begin{pmatrix} dx_i(\xi_1) & dx_i(\xi_2) \\ dy_i(\xi_1) & dy_i(\xi_2) \end{pmatrix} \\ &= dx_i(\xi_1)dy_i(\xi_2) - dx_i(\xi_2)dy_i(\xi_1) \end{aligned}$$

satisfying Grassmann's rules for exterior multiplication

$$dx_i \wedge dx_j = -dx_j \wedge dx_i, \quad dx_i \wedge dx_i = 0.$$

Symplectic is defines as

Theorem 20 *The flow of a canonical system (52) is symplectic, i.e.,*

$$(\varphi_t)^*\omega^2 = \omega^2 \qquad \forall t.$$