# The Parareal Algorithm

A survey of present work

Author: *Gunnar A. Staff*

Summer 2003

NOTUR
Emerging Technologies
Cluster Technologies

NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF MATHEMATICAL SCIENCES

# Preface

This project was a 6-week long project supported by NOTUR. Specifically, the funding came from the project: **Emerging Technologies, Cluster Technologies, Impact of future numerical algorithms and methods**.

In order to narrow the scope of this limited effort, the Parareal algorithm was considered an emerging technology in the algorithmic sense, and was chosen as the research topic.

The survey reported here is based on the author's work in [13] and [12], and on the work by several other authors. This survey represents, to the knowledge of the author, all the published and some unpublished work on this algorithm. Included are presentations given at the 15th International Conference on Domain Decomposition Methods held in Berlin 21-25 July 2003.

**Abstract**

The Parareal algorithm is presented as in [3]. Important properties like convergence, stability is discussed. Parallel properties like speed-up and efficiency is also presented. A short resume of different problems on which the Parareal algorithm has been tested, is given. Status quo of the algorithm is given, both based on the work presented so far, and also on preliminary results given in 15th international conference in Domain Decomposition in Berlin 2003.

# Contents

# Notation

In order to avoid confusion and ambiguities, a short list of the most important notation is presented here.

| | |
|---|---|
| $\mathbb{R}$ | all real numbers |
| $\mathbb{C}$ | all complex numbers |
| $\mathbb{C}^-$ | all complex numbers where the real value are less or equal to zero |
| $\mathbb{Z}$ | all integers |
| $\mathbb{Z}^+$ | all positive integers |
| $\mathcal{G}_{\Delta T}$ | coarse propagator of the Parareal algorithm |
| $\Delta T$ | timestep of the coarse propagator $\mathcal{G}_{\Delta T}$ |
| $\mathcal{F}_{\Delta T}$ | fine propagator of the Parareal algorithm |
| $\delta t$ | timestep of the fine propagator $\mathcal{F}_{\Delta T}$ |
| $\Omega$ | domain in space |
| $t_0$ | initial time where we have an initial value for the ordinary differential equation |
| $T$ | the end of the domain in time |
| $N$ | number of time decompositions – subdomains |
| $n$ | index spesifying a particular subdomain in time |
| $k$ | iteration counter for the predictor-corrector scheme (Parareal algorithm) |
| $s$ | number of time-steps for the fine propagator $\mathcal{F}_{\Delta T}$ over one subdomain ($s = \frac{\Delta T}{\delta t}$) |
| $\lambda_n^k$ | initial value for subdomain $n$ at iteration number $k$ |
| $\mathbb{I}$ | identy matrix |
| $R(z)$ | stability function in general. When used in the Parareal context is means the stability function of $\mathcal{G}_{\Delta T}$. |
| $r(z)$ | stability function of $\mathcal{F}_{\Delta T}$ in the Parareal context |

# 1 Introduction

Many of the governing equations describing mechanisms in nature are time-dependent. Significant work have been spent on deriving computational methods for integration of these in time. Most of the methods developed have one thing in common: they are sequential. They need to compute the solution to the equations at one time-step in order to find the answer at a later time-step. Time itself has been considered purely sequential.

Some work has been done on parallel solvers in time. But up to this point they have either been highly specialized, or they have had an upper limit of the speed-up. By speed-up we mean the wall-clock time for a parallel computation versus a serial computation.

In the context of solving time-dependent partial differential equations, a lot of parallel algorithms have been developed for the spatial discretization. One successful class of methods is called domain decomposition methods. These methods are attractive both in a serial and a parallel context.

The point of departure for this work is the Parareal algorithm, first presented by Lions, Maday and Turinici in 2001 (see [9]). We will present it as it is presented in the later work of Maday and Turinici from 2002 (see [3]). It is an attempt to use Domain Decomposition in time, creating a Time Decomposition. As for Domain Decomposition, the Parareal algorithm uses a coarse and a fine grid in time. These two grids are combined in a predictor-corrector scheme, creating an update for the entire time domain. The coarse grid and the predictor-corrector update are strictly sequential. The fine grid on the other hand are sequential only inside a sub-domain, and this allows for a parallel implementation of the different sub-domains. The predictor-corrector scheme is iterated until convergence. The Parareal algorithm is in principle problem independent. However, we will later discover that there are indications that the algorithm is less favorable for some classes of problems, namely problems with pure imaginary eigenvalues from the system matrix.

So why do we need this algorithm? The main reason is the need to solve important problems faster then currently possible. Examples are problems where the solutions are needed in real-time, e.g., flight/boat-simulators or control-problems. From this aspect, the algorithm has gotten its name. Other examples are large time-dependent fluid simulations, long-time weather-forecast, long-time planet orbit calculation etc. Another reason is the desire to effectively use the new supercomputer/clusters scheduled for the next decade. As an example the *Red Storm project* can be mentioned, which is assumed to use approximately 16'000 processors.

This work will focus on the following subjects

- Expected computational gain in a parallel implementation
- Known properties of the algorithm – possibilities/limitations
- Actual problems, on which the algorithm has been tested.

Section 2 presents the algorithm as it is presented by Bal an Maday in [3]. There exists other formulations, e.g. the one presented by Farhat and Chandesris in [4]. But these different formulations are believed to be equivalent, so we will pay Farhat's formulation no more attention.

Section 3 presents important numerical properties like convergence, stability and parallel properties like speed-up and efficiency.

Section 4 presents some of the different problems of which the Parareal algorithm has been tested. Speed-up results are presented in the cases where it is estimated.

Section 5 presents impressions from the 15th international conference in Domain Decomposition in Berlin where the Parareal algorithm was presented in large scale.

Section A presents a proof of equivalence between the formulation in [4] and [3] for the autonomous differential equation.

# 2 The Parareal Algorithm

In order to understand the implications of the Parareal algorithm (PA), it is imperative that the theory on which the algorithm is based is understood. First, the algorithm will be presented as in [3]. Then some properties of the algorithm will be pointed out.

## 2.1 Basic ideas

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) as

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

We now assume that $A$ is time-independent, and introduce the propagator $\mathcal{F}_{\Delta T}$ such that for any given $\mu$, $\mathcal{F}_{\Delta T}(\mu)$ is the solution at time $\Delta T$ 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 $\mathcal{O}(N)$ solutions. It's now we introduce our iterative scheme that allows us to construct a sequence $\Lambda^k$ 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

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

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

$$\widetilde{M} = \begin{pmatrix} \mathbb{I} & 0 & 0 & \dots & 0 \\ -\mathcal{G}_{\Delta T} & \mathbb{I} & 0 & \dots & 0 \\ 0 & -\mathcal{G}_{\Delta T} & \mathbb{I} & 0 & \dots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & -\mathcal{G}_{\Delta T} & \mathbb{I} \end{pmatrix} \tag{5}$$

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 $Res^k = F - M\Lambda^k$.

Convergence of Algorithm 1 can be tested using $\| \lambda_n^{k+1} - \lambda_n^k \| \leq tol \; \forall n$ as a test criterion.

## 2.2 Properties of the algorithm

We will now try to outline some of the most important properties of this algorithm. Some are really obvoius, and some are perhaps not so obvious.

**Remark 1** *The Parareal algorithm is a pure parallel algorithm, and has no value in a serial computation.*

This is in contrast to standard domain decomposition methods which are used in serial computation as well.

**Remark 2** *The Parareal algorithm converges towards a serial solution using the same solver and same discretization in time (and space for PDE's) as* $\mathcal{F}_{\Delta T}$.

**Algorithm 1** The Parareal algorithm

```
$\lambda_0^0 \leftarrow y_0$
for $i = 0 : N-1$ do
  $\lambda_{i+1}^0 \leftarrow \mathcal{G}_{\Delta T}(\lambda_i^0)$
end for
solve $\mathcal{F}_{\Delta T}(\lambda_i^0)$ in parallel on $i = 0, \dots, N-1$ processors with one fine subproblem per processor
$k \leftarrow 0$
while true do
  $\lambda_0^{k+1} \leftarrow \lambda_0^k$
  for $i = 0 : N-1$ do
    solve $\mathcal{G}_{\Delta T}(\lambda_i^{k+1})$
    $\lambda_{i+1}^{k+1} \leftarrow \mathcal{G}_{\Delta T}(\lambda_i^{k+1}) + \mathcal{F}_{\Delta T}(\lambda_i^k) - \mathcal{G}_{\Delta T}(\lambda_i^k)$
  end for
  if convergence then
    break
  end if
  solve $\mathcal{F}_{\Delta T}(\Lambda^{k+1})$ in parallel on $i = 0, \dots, N-1$ processors with one fine subproblem per processor
  $k \leftarrow k+1$
end while
```

It is important to bear in mind that even if the Parareal algorithm converges towards machine accuracy, the result is never more accurate then a serial computation. This means that when the same order of error is reached, futher iterations is superfluous.

**Remark 3** *Given that $\mathcal{G}_{\Delta T}$ and $\mathcal{F}_{\Delta T}$ are convergent and stable for the chosen schemes and timestep $\delta t$ and $\Delta T$. 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 a maximum number of iterations.

**Remark 4** *Assume the computational cost of computing $\mathcal{G}_{\Delta T}$ for $n = 0 \dots N-1$ on one processor is roughly the same as computing $\mathcal{F}_{\Delta T}$ for one subdomain. Then convergence has to be reached for $k < N/2$, in order to achieve speedup.*

Assuming 2 iterations before convergence, we need close to 10 processors to achieve a good speedup. Obviously, the Parareal algorithm is not useful for a two-processor implementation. By accelerating $\mathcal{G}_{\Delta T}$, by e.g. using a coarse spatial discretization for PDE's, the speed-up will increase for the same number of processors $N$ and iterations $k$. This allows for a speed-up for as few as 4 processors.

**Remark 5** *$\mathcal{G}_{\Delta T}$ doesn't have to be computed using the same equations as $\mathcal{F}_{\Delta T}$. We are free to modifie the model by e.g. removing highly oscillating terms that is totally undersampled by $\Delta T$, or use a coarser spatial discretization if we are evaluation a PDE. This will not affect the overall accuracy of the algorithm, only the convergence-rate.*

$\mathcal{G}_{\Delta T}$ has in a sense several degrees of freedom, meaning we have some freedom in the choice of equation and spatial discretization of $\mathcal{G}_{\Delta T}$.

**Remark 6** *We are free to choose what kind of ode-solver we want for $\mathcal{G}_{\Delta T}$ and $\mathcal{F}_{\Delta T}$. The only requirement is that, of course, the chosen methods have to be convergent and stable for the spesified stepsizes $\Delta T$ and $\delta t$.*

This means we e.g may use an implicit solver for $\mathcal{G}_{\Delta T}$, and a fast explicit solver for $\mathcal{F}_{\Delta T}$. Later we will present results restricting the choice of $G$ in order to achieve stability for the Parareal algorithm.

**Remark 7** *The algorithm favors single-step methods.*

The Parareal algorithm consists of $N$ individual problems with its own initial values. The startup problems of the multi-step schemes disfavours them in the Parareal algorithm context. Singel-step schemes, which suffers from no start-up problems, are therefore preferable.

# 3 Convergence, Stability and Speed-up properties of PA

We will in this section present some of the most important convergence, stability and theoretical speed-up results.

## 3.1 Convergence

If a numerical scheme is not convergent, it is of no use. It is therefore imperative that convergence is established. The work presented in this subsection is not done by the author. Proofs will not be presented, so the interested reader is referred to the references.

In [3] the convergence is analyzed for the autonomous differential equation

$$y' = \mu y, \quad y(0) = y_0, \quad \mu < 0 \tag{6}$$

**Proposition 1** *Let $u(t)$ be the solution of (6) and $\lambda_n^k$ be the solution from the Parareal algorithm (4), and that the coarse propagator $\mathcal{G}_{\Delta T}$ uses a scheme of order $m \geq 1$. Then the error terms $\varepsilon_n^k$ satisfy the following estimate*

$$|\varepsilon_n^k| \leq C_k n^{k+1} z^{(m+1)(k+1)}$$

*where $z = \mu \Delta T$. In particular, we have $\lambda_N^k = u(T) + \mathcal{O}(z^{m(k+1)})$ and $y_n^k(t) = u(t) + \mathcal{O}(n^k z^{(m+1)k})$, where $y_n^k(t)$ is the results from the Parareal algorithm at sub-domain $n$ and iteration $k$.*

This means that the Parareal algorithm turns the coarse propagator $\mathcal{G}_{\Delta T}$ of order $m$ into a scheme of order $(m+1)(k+1)$. Notice that this is the convergence to the serial version, and not to the exact solution. The Parareal algorithm is never more accurate then the serial version.

## 3.2 Stability

Stability is beside convergence the most important property of an initial value problem solver. If the algorithm is not stable, we can not trust the results. It is therefore imperative that the stability properties is investigated. This is done by the author in [13] and [12]. Farhat and Chandesris do, in [4], also present an investigation of the stability for an autonomous problem, using a different approach.

The core of the stability analysis is the predictor-corrector scheme

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

A stability analysis is performed on the autonomous differential equation

$$y' = \mu y, \quad y(0) = y_0, \quad \mu < 0 \tag{6}$$

The exact solution to this problem is $y = e^{\mu t} y_0$. Since $\mu < 0$, this is a decaying function for increasing $t$. The numerical solution of (6) is an approximation to the exact solution. It is well known that a numerical scheme is only exact for small time-steps. By choosing to large time-steps, some numerical schemes approximates this as an increasing exponential function instead of an decreasing. Obviously this is something we want to prevent. A numerical scheme that approximates an non-increasing exponential function for the chosen time-step is called stable.

To better understand the derivation of the stability properties of PA, we start by deriving the stability properties for to well known numerical schemes, namely explicit and implicit Euler. Applied to our differential equation, the two schemes can be written as

$$\begin{aligned} y_n &= y_{n-1} + \Delta T \mu y_{n-1} && \text{explicit Euler} \\ y_n &= y_{n-1} + \Delta T \mu y_n && \text{implicit Euler} \end{aligned}$$

where $\Delta T$ is the time-step By backward substituting for $y_{n-1}$, $y_{n-2}$ etc. we may write it as

$$\begin{aligned} y_n &= (1 + \Delta T \mu)^n y_0 = R(z)^n y_0 && \text{explicit Euler} \\ y_n &= \left(\tfrac{1}{1-\Delta T \mu}\right)^n y_0 = R(z)^n y_0 && \text{implicit Euler} \end{aligned}$$

where $z = \Delta T \mu$ and $R(z)$ is called the *stability function* of the chosen scheme. Obviously $|R(z)| \leq 1$ will prevent the numerical schemes from blowing up for increasing $t$.

An interesting observation is the difference in time-step restriction for explicit and implicit Euler, which can be seen in Figure 1. Obviously explicit Euler suffers from severe time-step restrictions, while implicit Euler is stable for all possible choices of the time-step $\Delta T$. The coarse propagator $\mathcal{G}_{\Delta T}$ is forced to take unnatural large time-steps, which clearly indicates that implicit Euler is a better choice then explicit Euler.

But what about PA? We need to write the predictor-corrector scheme (4) on the form

$$\lambda_n^k = H(n, k, r, R)\lambda_0,$$

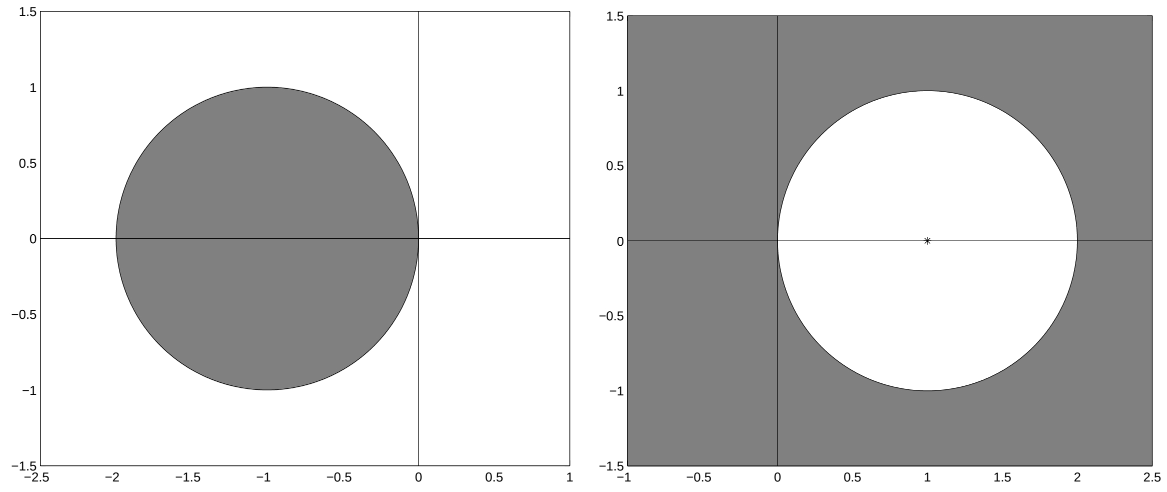
where $n$ is the specified sub-domain in time, $k$ is the iteration number, $r$ is the stability function for the fine propagator $\mathcal{F}_{\Delta T}$, and $R$ is the stability function for the coarse propagator $\mathcal{G}_{\Delta T}$.

To do this we start with the predictor corrector scheme given in (4).

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

We discretize (4) using our choice for numerical scheme for $\mathcal{F}_{\Delta T}$ and $\mathcal{G}_{\Delta T}$ and get

$$\lambda_n^k = \bar{r}(\mu \delta t)\lambda_{n-1}^{k-1} + R(\mu \Delta T)\lambda_{n-1}^k - R(\mu \Delta T)\lambda_{n-1}^{k-1}, \tag{7}$$



**(a)** explicit Euler **(b)** implicit Euler

**Figure 1:** Stability domain for explicit and implicit Euler. The dark region is the stability domain. The $x$-axis is the real axis, while the $y$-axis is the imaginary axis

where $\bar{r}(\mu\delta t) = r(\mu\delta t)^s$ is the stability function for the fine operator after $s = \frac{\Delta T}{\delta t}$ fine time-steps $\delta t$. $R(\mu\Delta T)$ is the stability function for the coarse operator $\mathcal{G}_{\Delta T}$ where $\Delta T$ is the coarse time-step. For simplicity we will write $\bar{r} = \bar{r}(\mu\delta t)$ and $R = R(\mu\Delta T)$.

We rearrange (7) and write

$$\lambda_n^k = R\lambda_{n-1}^k + (\bar{r} - R)\lambda_{n-1}^{k-1} = R\lambda_{n-1}^k + S\lambda_{n-1}^{k-1}. \tag{8}$$

Obviously the recursion is solved like this:

$$\begin{tikzcd}
\lambda_n^k \arrow[d] \arrow[dr] & & & \lambda_n^k \arrow[d, "R"] \arrow[dr, "S"] & \\
\lambda_{n-1}^k \arrow[d] \arrow[dr] & \lambda_{n-1}^{k-1} \arrow[d] \arrow[dr] & & \lambda_{n-1}^k & \lambda_{n-1}^{k-1} \\
\lambda_{n-2}^k \arrow[d] \arrow[dr] & \lambda_{n-2}^{k-1} \arrow[d] \arrow[dr] & \lambda_{n-2}^{k-2} \arrow[d] \arrow[dr] & & \\
\lambda_{n-3}^k \arrow[d] \arrow[dr] & \lambda_{n-3}^{k-1} \arrow[d] \arrow[dr] & \lambda_{n-3}^{k-2} \arrow[d] \arrow[dr] & \lambda_{n-3}^{k-3} \arrow[d] \arrow[dr] & \\
\lambda_{n-4}^k & \lambda_{n-4}^{k-1} & \lambda_{n-4}^{k-2} & \lambda_{n-4}^{k-3} & \lambda_{n-4}^{k-4}
\end{tikzcd}$$

The Pascal tree is recognized, and we may write (8) as

$$\lambda_n^k = \sum_{i=0}^{k} \binom{n}{i} (\bar{r} - R)^i R^{n-1} \lambda_0,$$

where we identify the stability function as

$$H(n,k,r,R) = \sum_{i=0}^{k} \binom{n}{i} (\bar{r} - R)^i R^{n-1}.$$

Stability is achieved if

$$\sup_{1\leq n\leq N} \sup_{1\leq k\leq N} |H(n,k,r,R)| \leq 1. \tag{9}$$

The stability requirements needs to be derived.

Assume $|(\bar{r} - R) + R| \leq 1$ and $|(\bar{r} - R) - R| \leq 1$.

$$\begin{aligned} \left|\sum_{i=0}^{k} \binom{n}{i} (\bar{r} - R)^i R^{n-i}\right| &\leq \sum_{i=0}^{k} \binom{n}{i} |(\bar{r} - R)|^i |R|^{n-i} \\ &\leq \sum_{i=0}^{n} \binom{n}{i} |(\bar{r} - R)|^i |R|^{n-i} \\ &= (|\bar{r} - R| + |R|)^n \leq 1 \end{aligned}$$

since $|\bar{r} - R| + |R|$ is either $|(\bar{r} - R) + R|$ or $|(\bar{r} - R) - R|$ which are our assumptions. $|(\bar{r} - R) + R| = |\bar{r}| \leq 1$ is the stability restriction to the fine operator, and it should be true independent of the use of the Parareal algorithm.

$|(\bar{r} - R) - R| = |2R - \bar{r}| \leq 1$ can be rewritten as

$$\frac{\bar{r} - 1}{2} \leq R \leq \frac{\bar{r} + 1}{2}$$

**Theorem 1 (Stability)** *Assume we want to solve the autonomous differential equation*

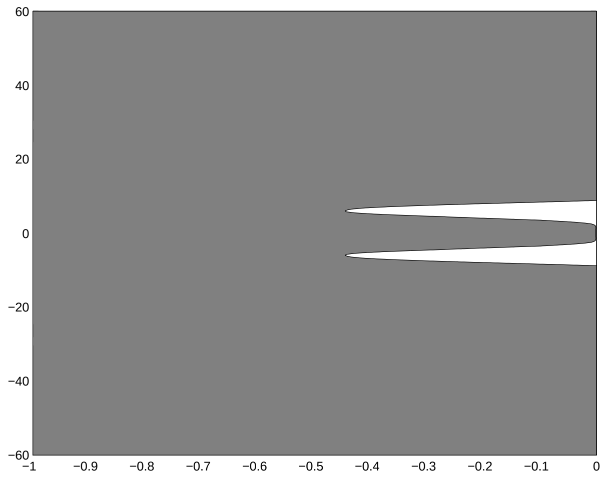
$$y' = \mu y \quad y(0) = y_0 \quad 0 > \mu \in \mathbb{R}$$

*and that $-1 \leq r, R \leq 1$ where $r = r(\mu\delta t)$ is the stability function for the fine propagator $\mathcal{F}_{\Delta T}$ using time-step $\delta t$ and $R = R(\mu\Delta T)$ is the stability function for the coarse propagator $\mathcal{G}_{\Delta T}$ using time-step $\Delta T$ . Then the Parareal algorithm is stable for all possible values of sub-domains $N$ and all number of iterations $k \leq N$ as long as*

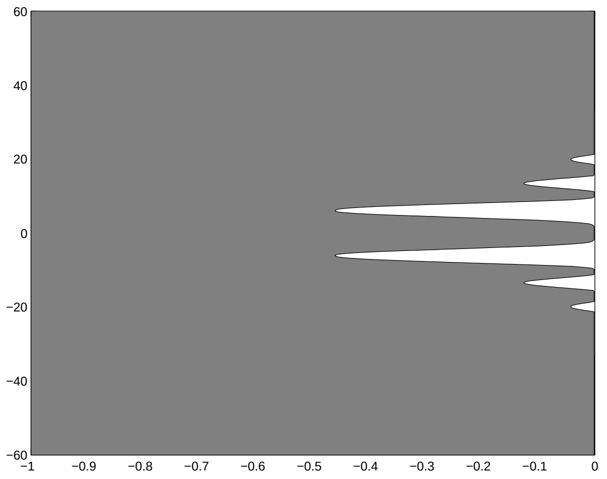
$$\frac{\bar{r} - 1}{2} \leq R \leq \frac{\bar{r} + 1}{2}$$

*where $\bar{r} = r(\mu\delta t)^s$ and $s = \frac{\Delta T}{\delta t}$.*

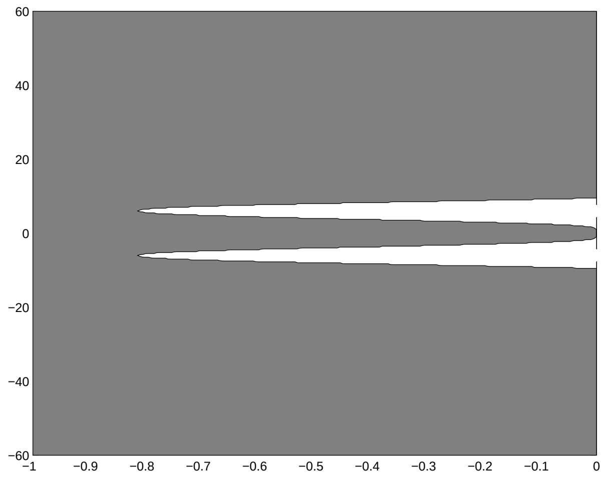
Notice that Theorem 1 is true for ode's and system of ode's where the eigenvalues of the system matrix have pure real eigenvalues. For complex eigenvalues, (9) needs to be



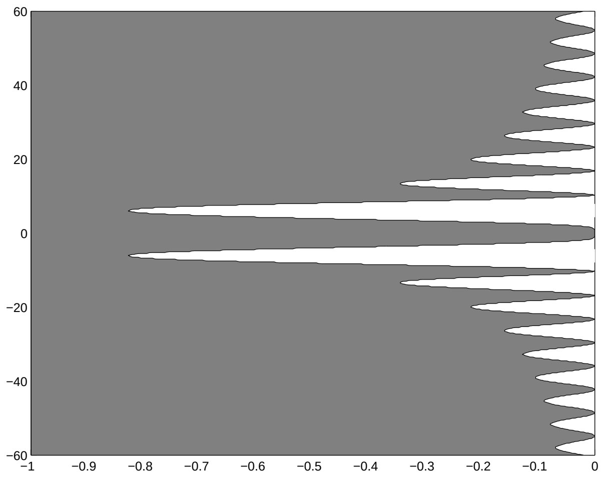
**(a)** $N = 10$, $s = 10$



**(b)** $N = 10$, $s = 1000$



**(c)** $N = 1000$, $s = 10$



**(d)** $N = 1000$, $s = 1000$

**Figure 2:** Stability plot for complex eigenvalues for different choices of $N$ and $s$, using Radau3 for both $\mathcal{G}_{\Delta T}$ and $\mathcal{F}_{\Delta T}$. The dark regions are the stability domain. The x-axis is $\mathrm{Re}(\mu\Delta T)$ and the y-axis is $\mathrm{Im}(\mu\Delta T)$.

fulfilled. This is done numerically in Figure 2. From Figure 2 we notice that the Parareal algorithm is unstable for pure imaginary eigenvalues, and some complex eigenvalues where the imaginary part is much larger then the real. No multistage scheme has yet been found that makes the present formulation of the Parareal algorithm stable for all kind of eigenvalues. This means that computation of some hyperbolic problems, and convection-diffusion problem with highly dominant convection (e.g Navier-Stokes with high Reynolds numbers) is probably unstable using the Parareal algorithm

It is believed that by introducing an rotation instead of an translation, the Parareal algorithm will be stable for pure imaginary eigenvalues as well. This is ongoing research.

## 3.3 Speed-up, Efficiency and Restarted algorithm

For all parallel algorithms, the speed-up is the most important. How much faster is it possible to calculate our problem assuming we have $N$ processors? But Efficiency is also of great importance since if the efficiency drops, it may be more efficient to use the rest of the processors in the spatial domain, instead of the time-domain.

Both [2] and [4] presents an analysis of the possible speed-up and efficiency. But we will only present the analyze from [2] since it is the most thorough.

For the reader who is not familiar with the term speed-up and efficiency in the context of parallel computations, it can be defined as this: The speed-up is the ratio between the wall-clock time of the serial and the parallel computation. The efficiency is the ratio of this speed-up with the number of processors used.

The following results are derived in [2] assuming that the order of both $\mathcal{G}_{\Delta T}$ and $\mathcal{F}_{\Delta T}$ are 1. The results can easily be extended to higher order schemes. The speed-up is defined as

$$S = \frac{\frac{T}{\delta t}}{k\frac{T}{\Delta T} + (k-1)\frac{\Delta T}{\delta t}} = \frac{1}{k\frac{\delta t}{\Delta T} + (k-1)\frac{\Delta T}{T}}$$

The number of processors is given by $P = \frac{T}{\Delta T}$ so that the efficiency is given by

$$E = \frac{1}{(k-1) + k\frac{T\delta t}{(\Delta T)^2}}$$

The speed-up assuming $k \geq 1$ is maximized for $k = 2$ and $\Delta T = \sqrt{2T\delta t}$. From this it can be deduced that

$$S = \frac{1}{2}\sqrt{T}2\delta t, \qquad P = \sqrt{T}2\delta t, \qquad E = \frac{1}{2}.$$

This seems in a way that the efficiency is locked by the factor of $\frac{1}{2}$. But Bal shows in [2] that this can be overcome by introducing a multi-step system. We assume that we have a scale of time steps such that

$$\Delta_m T \ll \Delta_{m-1} T \ll \ldots \ll \Delta_1 T \ll \Delta_0 T \ll T$$

The speed-up of the multi-step algorithm is then given by

$$S = \frac{\frac{T}{\Delta_m T}}{N\left(2\left(\frac{\Delta_0 T}{\Delta_1 T} + \ldots + \frac{\Delta_{m-2} T}{\Delta_{m-1} T}\right) + \frac{\Delta_{m-1} T}{\Delta_m T}\right)}$$

By using this multi-step method, which is implemented as a restarted algorithm (see [2] for details), the efficiency can be close to 1.

# 4 Problems on which PA have been tested

In this section we will present some problems on which the Parareal algorithm has been tested, and point out promising and not so promising aspects. Notice that most of the tests are done using simulated parallel implementation. Therefore no wall-clock time is available, but a theoretical speed-up is usually presented based on the assumption that the communication cost is neglectable compared to the computational cost.

## 4.1 Linear and nonlinear Parabolic problems

Parabolic problems is highly interesting since it covers a large class of problems in science and engineering.

Linear parabolic problems have been tested in [13, 12].

Nonlinear parabolic problems is tested in [12] and [3].

The Parareal algorithm looks particularly promising for this kind of problems. As long as the stability requirement in Theorem 1 is fulfilled, the convergence is fast. For parabolic problems with eigenvalues where the imaginary part is highly dominant, it seems difficult to fulfill the stability-requirement. This will be discussed in Section 4.8

## 4.2 Non-differential equation: The American Put

In [3] the pricing of an American option is solved using the Parareal algorithm. This equation is of interest since it is frequently calculated in the financial world.

The actual equation calculated was the following.

$$\min\left(\partial_t u - \partial_{xx}^2 u, u - g(x)\right) = 0, \tag{10}$$

with $u(t=0) = g(x) = \max\left(e^x - 1, 0\right)$.

Notice that the results in [3] is based on a simulated parallel implementation, and therefore no actual wall-clock time is presented. The speedup is based on an assumption that the communication cost is neglectable compared to the computational cost. 3 iterations was needed to reach same level of error as for the serial version, and therefore the speed-up found was 6.25 based on an implementation on 50 processors.

## 4.3 Molecular-dynamics simulations

In [1] the Parareal algorithm is applied on a small molecular dynamic simulation example. A asymmetric molecule $A-A-B$ composed of three atoms of mass $m_A = 1$ and $m_B = 2$ is considered. The bound lengths between atoms are denoted by $r_{AA}$ and $r_{AB}$, and the angle

$A - \hat{A} - B$ of the bounds is denoted $\theta$. This molecule evolves on the potential surface $U$ given by

$$U(r_{AA}, r_{AB}, \theta) = \mathcal{V}(r_{AA}) + \mathcal{V}(r_{AB}) + f(\theta)$$

where $\mathcal{V}(r) = 4\varepsilon[(\sigma_1/r)^{12} - (\sigma_1/r)^6]$ is the Lennard-Jones potential and
$f(\theta) = (\lambda/\sqrt{2\pi}\sigma_2)e^{-(\theta-\pi)^2/2\sigma_2} + \mu/\sin\theta/2$.

The coarse operator $\mathcal{G}_{\Delta T}$ is implemented both using only a coarse time-step, and also using a coarse physical model. The results look promising. But some properties, like the symplectic behavior of the algorithm is not yet fully investigated. All the simulations here is a simulated parallel implementation.

## 4.4 Optimal control for Partial differential equations

In [10] the scheme is reinterpreted as a preconditioning procedure on an algebraic setting if the time discretization. This is done to extend the parallel methodology of the Parareal algorithm to the problem of optimal control for partial differential equations. The chosen test-equation is: Find $y$ such that

$$\begin{aligned} \frac{\partial y}{\partial t} - \frac{\partial^2 y}{\partial x^2} &= v\chi, \quad x \in (0,1) \\ y(0) = 10x(1-x)&, \quad t \in (0,100) \end{aligned}$$

where $v$ is the control and $\chi$ is the indicator $]1/2, 2/3[$. This is simulated so as to drive it to the target $y^T = \sin 2\pi x$.

For reasons not discussed here (see [10] for details) the adjoint state is introduced. It is written on the same form as (2). Let $p_{N-1}$ be the solution over $(T_{N-1}, T_N)$ of

$$\begin{cases} -\frac{\partial p_{N-1}}{\partial t} + A^* p_{N-1} = 0 \quad \text{over } (T_{N-1}, T_N) \\ p_{N-1}(T) = \alpha(y_{N-1}(T) - y^T), \end{cases}$$

and the collection $p_n$, $n = N-2, N-1, \dots, 0$ of solutions of

$$\begin{cases} -\frac{\partial p_n}{\partial t} + A^* p_n = 0 \quad \text{over } (T_n, T_{n+1}) \\ p_n(T_{n+1}^-) = \frac{1}{\varepsilon\Delta T}(y_n(T_{n+1}^-) - \lambda_{n+1}), \end{cases}$$

The following preconditioned gradient method is proposed in [10]:

$$\begin{aligned} v_n^{k+1} &= v_n^k - \rho(v_n^k + B^* p_n) \quad \text{in } (T_n, T_{n+1}), \quad n = 0, \dots, N-1 \\ \lambda_n^{k+1} &= \lambda_n^k - \rho[\widetilde{M}^{-1}(\widetilde{M}^{-1})^*](p_n^k(T_n^+) - p_{n-1}^k(T_n^-)), \quad n = 1, \dots, N-1 \end{aligned}$$

where $\widetilde{M}$ is given in (5). $\widetilde{M}^{-1}(\widetilde{M}^{-1})^*$ is the preconditioner for the system.

The results are surprisingly promising. For only 25 iterations of the Parareal algorithm, the cost-function used to measure the error is of the same order as after 100 iterations of a serial version. This factor of 4 is multiplied to the parallel effect of using 100 processors. The reason for this is not understood, and it is not expected that this reduction in iterations will apply for more complex problems.

## 4.5 Stochastic ordinary differential equations and filtering problems

In [2] stochastic ordinary differential equations and filtering problems are computed. One test-equations is the Geometric Brownian motion, defined as the solution of

$$\begin{aligned} &dX(t) = rX(t)dt + \sigma X(t)dB(t), \quad t \in (0,T), \\ &X(0) = 1, \end{aligned}$$

where $B(t)$ is an $m$-dimensional Brownian motion and $\sigma$ so a mapping from $[0,T] \times \mathbb{R}^d$ to $\mathcal{M}_{d\times m}$, the set of $d\times m$ matrices. The results look promising. But as it is commented in [2]: *Let us mention however that when only statistical averages of the solution are required, such as* $\mathbb{E}[X(T)]$, *it might be better to use the available number of processors to run independent realizations of the random process using the fine time step* $\delta t$.

But an interesting application where the solution of a stochastic equation for one realization of the random process matters, is the filtering problem. Here [2] shows promising results.

## 4.6 Reservoir simulations

In [6], the Parareal algorithm is applied to a reservoir simulation. The simulations compute pressure, temperature and molar mass. Molar mass needs a finer discretization in time because of different time-scales in the system. Therefore the Parareal algorithm is applied to this fine time-steps inside a coarse time-step for temperature and pressure.

By using $T/\delta t = 16$, and $T/\Delta T = 4$, the acquired speed-up is 2, which is predicted by the speed-up theory developed in [2]. It is worth noticing that [6] have an actual parallel implementation where computational and communicational cost and speed-up is measured.

## 4.7 Fluid, structure and fluid-structure computations

In [4] fluid, structure and coupled fluid-structure models are computed. The results in this article are based on real parallel implementations. The scheme presented is an alternative algorithm, but as shown in Section A, the algorithm is equivalent to the Parareal algorithm as it is presented in [4], at least for the autonomous differential equation. There exist no proof of equivalence for other type of equations, but it is believed that they are indeed equivalent for other types as well.

For unsteady flow, [4] presents a speedup of 8.2 by using 20 processors. This is an efficiency of 41%, which is not competiable with parallel methods in space, but assuming sufficient number of processors this yields an additional speed-up.

For a structural dynamics model problem there were however some difficulties.

The problem which is solved is

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix} \frac{\mathrm{d}q}{\mathrm{d}t} \\ q \end{pmatrix} + \begin{pmatrix} M^{-1}D & M^{-1}K \\ -I & 0 \end{pmatrix}\begin{pmatrix} \frac{\mathrm{d}q}{\mathrm{d}t} \\ q \end{pmatrix} = 0$$

where $q$ denotes the vector of displacement degrees of freedoms, $M$, $D$ and $K$ denote the finite element mass, Rayleigh damping, and stiffness matrices respectively, and $I$ is the identity matrix. The scheme is in general not stable for this type of problems. In [4] a nice argument is presented where it is showed that the predictor-corrector scheme generates a resonance for the undamped problem. It is believed by the author of this report that this resonance is closely connected to the instability for general problems with pure imaginary eigenvalues, and eigenvalues with large relative imaginary part.

Fluid-structure problems is also considered. Not surprisingly, the result is the same as for the structure problem. If instabilities occur in the structure part of the computation, it will pollute the solution of the fluid problem.

## 4.8 Navier-Stokes and related equations

In [5] Navier-Stokes simulations is performed using the Parareal algorithm. The results are exactly what the theory in Section 3.2 predicts. For small Reynolds numbers it works very well, but for large Reynolds numbers it fails because of violation of the stability properties.

# 5 Conclusions and general impression from DD15

## 5.1 General impression from DD15

The 15th conference in Domain Decomposition in Berlin was the first time the Parareal algorithm was presented in wide scale. In total there were 1 invited talk and 5 minisymposium talks which had the Parareal algorithm as the main subject. In addition, one other main speaker covered the Parareal algorithm in part.

The impression from the author is that it is an algorithm that will receive a lot attention from several new groups in the time to come, both from theoretical and applied groups. The need to be able to use all the processors available from the scheduled supercomputers (e.g the Blue Gene/L machine delivered 2004 which is supposed to have 65536 processors) will make parallelization in time more attractive. This huge amount of processors makes time-parallelization an addition to spatial parallelization instead of a competitor.

## 5.2 General conclusions of the applicability of the Parareal algorithm

Except from a few problems like e.g. earth-quake simulations, turbulence simulations and transient-flow simulations, the space is in most cases well parallelized using the available number of processors. But the number is constantly increasing, and the desire to put them to work turns the focus towards time-parallelization. Until now not much work has been done on time-parallelization. Mainly because of spatial parallelization has been more effective. But in this new situation with $10^4$ to $10^5$ processors on a single cluster/supercomputer, there is no doubt that the Parareal algorithm, as an time-parallel algorithm, tries to fill the need for putting the increasing number of available processors to work.

For parabolic problems with eigenvalues having not a too large relative imaginary part, the Parareal algorithm works efficiently. In [2], it is showed that a parallel efficiency of up to 1 is possible, which makes the algorithm competitive with the best spatial parallel algorithms. More research will be put into lowering the computational cost of the sequential part of the algorithm, by e.g. introducing a spatial discretization for the coarse operator $\mathcal{G}_{\Delta T}$. This has already been investigated in [12] and [6], but more analysis is needed in order to understand the implications for convergence and stability.

It has also been shown that it works for other classes of problems, like the non-differential equation the American put, and optimal control problems. More analysis is needed before all the implications is understood, but the numerical tests is promising.

But the Parareal algorithm is not flawless at this moment. The instability occurring for complex eigenvalues with relative large imaginary part is a problem since many interesting problems do possess this property. As examples, the following can be mentioned:

- The Navier-Stokes equations with high Reynolds numbers
- Most hyperbolic equations and conservation laws
- Structural dynamic vibration problems

But a lot of work will be put into solving this difficulties. And if this is solved without introducing additional flaws, the Parareal algorithm will pay a vital contribution to the tool box of parallel algorithms.

## References

[1] L. Baffico, S. Bernard, Y. Maday, G. Turinici, and G. Zérah. Paralell in time molecular dynamics simulations. *Physical review. E*, 66, 2002.

[2] Guillaume Bal. Parallelization in time of (stochastic) ordinary differential equations. submitted, 2002.

[3] Guillaume Bal and Yvon Maday. A "parareal" time discretization for non-linear pde's with application to the pricing of an american put. In Luca F. Pavarino and Andrea Toselli, editors, *Recent Developments in domain Decomposition Methods*, volume 23 of *Lecture Notes in Computational Science and Engineering*, pages 189–202. Springer, 2002.

[4] Charbel Farhat and Marion Chandesris. Time-decomposed parallel time-integrators: theory and feasibility studies for fluid, structure, and fluid-structure applications. to appear in International Journal for Numerical Methods in Engineering, 2003.

[5] Paul F. Fischer. Investigation of the parareal algorithm for semi-implicit incompressible navier-stokes simulations. Talk at 15th domain decomposition conferance in Berlin, 2003.

[6] I. Garrido, G. Fladmark, and M. Espedal. Parallelization in time for reservoir simulation. submitted, 2003.

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

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

[9] Jacques-Louis Lions, Yvon Maday, and Gabriel Turinici. Résolution d'edp par un schéma en temps pararéel. *C.R.Acad Sci. Paris Sér. I Math*, 332:661–668, 2001.

[10] Yvon Maday and Gabriel Turinici. A parareal in time procedure for the control of partial differential equations. *C.R.Acad Sci. Paris Sér. I Math*, 335:387–392, 2002.

[11] Yvon Maday and Gabriel Turinici. A parallel in time approach for quantum control: the parareal algorithm. 2003.

[12] Gunnar A. Staff. Convergence and stability of the parareal algorithm. Master's thesis, Norwegian University of Science and Technology, 2003.

[13] Gunnar A. Staff. Convergence and stability of the parareal algorithm: A numerical and theoretical investigation. Technical Report 2, Norwegian University of Science and Technology, 2003. ISSN: 0804-9181.

## A Equivalence between Farhat's formulation and standard Parareal algorithm for the autonomous differential equations

The purpose of this section is to prove the equivalence between Farhat's formulation of a parallel scheme in time, [4] and Bal and Maday's formulation given in [3] and especially [1], for the autonomous differential equation.

From [4] we extract the essence of the algorithm:

$$\Delta_k^i = y_k^{i-1}(T^i) - Y_k^i, \quad 1 \le i \le N_{ts} - 1 \tag{11}$$

$$Y_{k+1}^i = y_k^{i-1}(T^i) + C_k^i \quad C_k^i = c_k(T^{i-}) \tag{12}$$

$$c_k(T^{i+}) = c_k(T^{i-}) + \Delta_k^i, \quad c_k(T^0) = 0, \quad 1 \le i \le N_{ts} - 1 \tag{13}$$

Subscript $k$ is the iteration counter, while superscript $i$ is a specified sub-domain in time. $N_{ts}$ is the number of sub-domains in time. $f$ is the number of fine time-steps over one sub-domain. The algorithm is then presented in Algorithm 2 as it is presented in [4].

**Algorithm 2** Farhat's algorithm

```
- Provide initial seed values $Y_0^i$, $0 \le i \le N_{ts}$.
  for $k = 0, 1, \dots$ do
-     Using the updated seed values $Y_k^i$ as initial conditions, apply the chosen fine solver to
      the differential equation on the fine time-grid, which generates the numerical values
      $y_k^i$, $0 \le i \le f$
-     Evaluate the jumps $\Delta_k^i$, $1 \le i \le N_{ts} - 1$ on the coarse time-grid equation (11). Stop
      if all the jumps are sufficiently small
-     Apply the chosen coarse propagator to problem (13) on the coarse time-grid in order
      to compute the correction coefficients $C_k^i$, $1 \le i \le N_{ts} - 1$.
-     Update the seed values $Y_{k+1}^i$, $0 \le i \le N_{ts} - 1$ using (12).
  end for
```

We will here assume that the same numerical scheme used to calculate $c_k(T^{i+})$ is used to find the initial seed values $Y_0^i$, $0 \le i \le N_{ts}$.

We will prove that this is equivalent to the predictor-corrector scheme given in [1], which is

$$\lambda_n^k = G\lambda_{n-1}^k + F\lambda_{n-1}^{k-1} - G\lambda_{n-1}^{k-1}, \tag{4}$$

where the superscript $k$ is the iteration counter, and the subscript $n$ is the specified sub-domain in time. First we will rewrite (11), (12) and (13) using the same notation as used in [1].

$$\Delta_n^k = FY_{n-1}^k - Y_n^k, \quad 1 \le n \le N_{ts} - 1 \tag{14}$$
$$Y_n^{k+1} = FY_{n-1}^k + G\mathcal{Y}_{n-1}^k \tag{15}$$
$$\mathcal{Y}_{n+1}^k = G\mathcal{Y}_n^k + \Delta_n^k, \quad c_k(T^0) = 0, \quad 1 \le n \le N_{ts} - 1 \tag{16}$$

$i$ is changed with $n$ and written as subscript instead of superscript, $k$ is superscript instead of subscript. $c_k^i(Ti+)$ is rewritten at the coarse propagator $G\mathcal{Y}_n^k$, where $\mathcal{Y}_n^k$ is the initial value for the coarse propagator at sub-domain $n$ for iteration $k$. The same is done for the calculation on the fine tine-grid where $y_k^i(T^{i+1})$ is written as the fine propagator $FY_n^k$ where $Y_n^k$ is the seed value (initial value) for sub-domain $n$ at iteration $k$. For simplicity we combine (14) and (15) and get the following two coupled recursion equations

$$Y_n^k = FY_{n-1}^{k-1} + G\mathcal{Y}_{n-1}^{k-1} \tag{17}$$
$$\mathcal{Y}_n^k = G\mathcal{Y}_{n-1}^k + FY_{n-1}^k - Y_n^k, \quad c_k(T^0) = 0, \quad 1 \le n \le N_{ts} - 1 \tag{18}$$

It can be shown that (4) can be written as

$$\lambda_n^k = \sum_{i=0}^{k} \binom{n}{i} (F-G)^i G^{n-i} \lambda_0. \tag{19}$$

We want to show, by using induction, that $Y_n^k$ may be written on the same form

$$Y_n^k = \sum_{i=0}^{k} \binom{n}{i} (F-G)^i G^{n-i} Y_0.$$

For $n = k = 1$ it is quite easy:

$$Y_1^1 = FY_0^0 + G\mathcal{Y}_0^0 = FY_0^0, \quad \text{since } \mathcal{Y}_0^k = 0,\ 0 \le k \le N_{ts} - 1$$

This is the same as we get from (4), namely

$$\lambda_1^1 = G\lambda_0^1 + F\lambda_0^0 - G\lambda_0^0 = F\lambda_0^0, \quad \text{since } \lambda_0^k = y_0,\ 0 \le k \le N_{ts} - 1$$

We assume that

$$Y_{n-1}^{k-1} = GY_{n-2}^{k-1} + FY_{n-2}^{k-2} - GY_{n-2}^{k-2}$$

or the equivalent formulation based on (19),

$$Y_{n-1}^{k-1} = \sum_{i=0}^{k-1} \binom{n-1}{i} (G-F)^i G^{n-i-1} Y_0. \tag{20}$$

We are then left with proving that this is true for $Y_n^k$.

$$\begin{aligned}
Y_n^k &= FY_{n-1}^{k-1} + G\mathcal{Y}_{n-1}^{k-1} \\
&= FY_{n-1}^{k-1} + G^2\mathcal{Y}_{n-2}^{k-1} + GFY_{n-2}^{k-1} - GY_{n-1}^{k-1} \quad \text{using (18)} \\
&= (F-G)Y_{n-1}^{k-1} + GFY_{n-2}^{k-1} + G^2\mathcal{Y}_{n-2}^{k-1} \\
&= (F-G)Y_{n-1}^{k-1} + GFY_{n-2}^{k-1} + G^3\mathcal{Y}_{n-3}^{k-1} + G^2FY_{n-3}^{k-1} - G^2Y_{n-2}^{k-1} \quad \text{using (18)} \\
&= (F-G)Y_{n-1}^{k-1} + G(F-G)Y_{n-2}^{k-1} + G^2FY_{n-3}^{k-1} + G^3\mathcal{Y}_{n-3}^{k-1}
\end{aligned}$$

This is done until we reach $Y_0^{k-1}$ and $\mathcal{Y}_0^{k-1}$. We may then write

$$\begin{aligned}
Y_n^k &= \sum_{i=0}^{n-2} G^i(F-G)Y_{n-1-i}^{k-1} + G^{n-1}FY_0^{k-1} + R^n \underbrace{\mathcal{Y}_0^{k-1}}_{=0} \underbrace{-(R^nY_0^{k-1} - R^nY_0^{k-1})}_{=0} \\
&= \sum_{i=0}^{n-1} G^i(F-G)Y_{n-1-i}^{k-1} + G^nY_0^{k-1}
\end{aligned}$$

We now use what we assumed correct in (20) and write

$$\begin{aligned}
Y_n^k &= G^nY_0^{k-1} + \sum_{i=0}^{n-1} G^i(F-G)Y_{n-1-i}^{k-1} \\
&= G^nY_0^{k-1} + \sum_{i=0}^{n-1} G^i(F-G)\sum_{j=0}^{k-1}\binom{n-1-i}{j}(F-G)^jG^{n-1-j-i}Y_0^{k-1} \\
&= G^nY_0^{k-1} + \sum_{i=0}^{n-1}\sum_{j=0}^{k-1}\binom{n-1-i}{j}(F-G)^{j+1}G^{n-1-j}Y_0^{k-1} \\
&= G^nY_0^{k-1} + \sum_{j=0}^{k-1}(F-G)^{j+1}G^{n-1-j}\sum_{i=0}^{n-1}\binom{n-1-i}{j}Y_0^{k-1} \\
&= G^nY_0^{k-1} + \sum_{j=0}^{k-1}(F-G)^{j+1}G^{n-1-j}\binom{n}{j+1}Y_0^{k-1}
\end{aligned}$$

We do a change of variable, writing $i = j + 1$ and get

$$\begin{aligned}
Y_n^k &= G^nY_0^{k-1} + \sum_{i=1}^{k}\binom{n}{i}(F-G)^iG^{n-i}Y_0^{k-1} \\
&= \sum_{i=0}^{k}\binom{n}{i}(F-G)^iG^{n-i}Y_0
\end{aligned}$$

This completes the proof $\square$.

This proves that the algorithm presented in [4] is equivalent to the algorithm presented in [3] for the autonomous differential equation. It is believed that they are equivalent for other types of equations as well.