Parallel-in-Time integration with application to eddy current simulations

Dissertation

Bergische Universität Wuppertal
Fakultät für Mathematik und Naturwissenschaften

eingereicht von
Jens Hahne, M. Sc.
zur Erlangung des Grades eines Doktors der Naturwissenschaften

Betreut durch Dr. Stephanie Friedhoff und Prof. Dr. Matthias Bolten

Wuppertal, 25.04.2023

Acknowledgments

First, I would like to thank my supervisors Stephanie Friedhoff and Matthias Bolten for raising my interest in parallel-in-time methods and giving me the opportunity to work on interesting projects in this research area; as well as for the guidance and all the fruitful discussions during the last years. Thank you for the always pleasant working atmosphere and the encouragement and support to present my work at many different conferences.

Many thanks to everyone I have had the pleasure of working and discussing with over the years, especially my colleagues at TU Darmstadt, Los Alamos National Lab, TU Hamburg, University of New Mexico, and Sandia National Labs. Many thanks to Jacob Schroder for making my stay at UNM possible and for the great support during my time in New Mexico. It was a great experience and I learned a lot. Thanks also to Eric Cyr for his support and for giving me the opportunity to work on an exciting project.

I also wish to thank all my current and former colleagues in the Scientific Computing and High Performance Computing group, many of whom have become friends over the years. I have always enjoyed the discussions, lunches, cakes, and activities during and after work. A special thanks goes to Marcel for proofreading this thesis.

A huge thank you to my family and friends for their love and constant support. And finally, a special thanks to Kim for her patience, support and entertainment over the past years.

Foreword

The work presented in this thesis is partly based on the following publications:

- M. Bolten, S. Friedhoff, J. Hahne, and S. Schöps, *Parallel-in-time simulation of an electrical machine using MGRIT*, Computing and Visualization in Science, 23 (2020), pp. Paper No. 14, 14.

  Parts of this publication are incorporated in Chapters 4 and 7.

- J. Hahne, S. Friedhoff, and M. Bolten, *Algorithm 1016: PyMGRIT: A Python Package for the Parallel-in-Time Method MGRIT*, ACM Transactions on Mathematical Software, 47 (2021).

  Parts of this publication are incorporated in Chapter 5.

- J. Hahne, B. S. Southworth, and S. Friedhoff, *Asynchronous truncated multigrid-reduction-in-time*, SIAM Journal on Scientific Computing, (2022), pp. S281–S306.

  Parts of this publication incorporated in Chapters 4, 6, and 7.

- J. Hahne, B. Polenz, I. Kulchytska-Ruchka, S. Friedhoff, S. Ulbrich, and S. Schöps, *Parallel-in-time optimization of induction motors*, 2022 (Submitted).

  Parts of this publication are incorporated in Chapters 3 and 7.

- M. Bolten, S. Friedhoff, and J. Hahne, *Task Graph-Based Performance Analysis of Parallel-in-Time Methods.* Available at SSRN: `https://ssrn.com/abstract=4201056`, 2022 (Submitted).

  Parts of this publication are incorporated in Chapters 4 and 8.

Contents

Chapter 1

Introduction

Computational Science and Engineering has established itself in many natural science and engineering disciplines as a simulation science that encompasses and closely links the development of models, algorithms and software. In many applications, numerical simulation complements the two principles of theory and experiment and, in particular, allows for a reduction of costly experiments. In order to draw conclusions about real-world problems through computer-aided simulations, increasingly complex mathematical models are required, leading to an exponentially growing computational effort. The effective use of modern supercomputers therefore plays a crucial role in performing these elaborate simulations. To effectively use such modern supercomputers, the development of parallel algorithms and methods is essential. In the field of numerical simulation of time-dependent processes, the classical approach is to parallelize the spatial dimension and consider the temporal dimension completely sequentially. However, available spatial parallelization can eventually become exhausted on large scale machines, even though additional resources are available. Moreover, the sequential nature of the temporal dimension often becomes a bottleneck in such simulations. A promising approach to further reduce simulation times for time-dependent problems with these resources is using parallel-in-time (PinT) methods.

The development of PinT methods goes back at least 50 years [82], with the majority of PinT methods being multilevel and iterative. They use a type of multilevel hierarchy, with the finest level representing the mathematical model of interest discretized to the desired accuracy. The coarser levels represent simplified models, e. g., by choosing coarser (space-)time grids and/or reduced mathematical models. PinT methods then parallelize all but the coarsest level in time by considering multiple time slices simultaneously. Since the initial conditions for each of these time slices, except the first, are not available at the beginning of the method, a correction procedure and the multilevel hierarchy are used to iteratively correct the solution at the finest level. Examples of such PinT methods include Parareal [71], the parallel full approximation scheme in space and time (PFASST) [29], and multigrid-reduction-in-time (MGRIT) [30]. An example of an alternative strategy is exponential integrators, which allow one to parallelize the work within each time step. An excellent overview of many different PinT methods is provided in the review articles [39, 83].

Electromechanical energy converters such as electrical machines can be found in a wide range of devices, e. g., in industrial automation, household devices or as drives or components in the automotive industry. The simulation of electrical machines, e. g., synchronous or induction machines, transformers and cables, is therefore an established method in industry. In the design process, computer-aided simulations can be used to investigate many different geometries and optimize them with respect to various criteria such as efficiency, performance, and also reduced resource consumption (e. g., rare earths). For low-frequency devices, the magnetoquasistatic equation, also called the eddy current problem, is usually used for the numerical simulation of electromagnetic fields. The eddy current problem is a simplification of Maxwell's equations in which the displacement current is neglected with respect to the source currents. When simulating the start-up phase of an electrical machine, i. e., the phase needed until the machine reaches its steady state, the system behavior has to be simulated for a large time interval. In addition, machines are usually driven by pulse-width modulated (PWM) excitations. PWM signals are fast switching pulses whose width is controlled so that the time average corresponds to a specific waveform, usually a sine wave. To resolve these discontinuous and highly oscillatory signals, a very small time step is required, which, in combination with large time intervals, leads to high simulation costs.

In this thesis, we apply the MGRIT framework to a numerical model of an induction machine to reduce the simulation time by parallelizing the time dimension. We consider different parameters for the numerical model and investigate different effects, such as the application of the MGRIT method for a discontinuous problem and the effect of spatial coarsening. Motivated by observed limitations of the temporal grid hierarchy when applying the MGRIT algorithm to the induction machine model and processor-local multigrid hierarchies used in geometric and algebraic multigrid for elliptic problems [7, 77, 78], we introduce a new variant of the MGRIT algorithm, called *asynchronous truncated MGRIT* (AT-MGRIT), study it theoretically, and embed the method in an optimization workflow for geometry optimization of an induction machine. The appropriate choice of the multilevel hierarchy and other parameters within MGRIT (and other PinT methods) plays an important role for the efficiency of the method, and to this end we present a performance model for PinT methods that can help in choosing these parameters.

The structure of this thesis is as follows: In Chapter 2 we introduce various fundamentals on which the further chapters build, with the aim of making this thesis as self-contained as possible. To this end, we first introduce the concept of differential equations and then consider three particular classes, more specifically ordinary differential equations (ODEs), differential-algebraic equations (DAEs), and partial differential equations (PDEs). Afterwards, we examine discretization methods, first considering discretization of the space dimension using finite difference and finite element methods, and then methods for discretizing the time domain. The discretization of a differential equation leads to a system of algebraic equations, and we present methods for solving these potentially nonlinear equations. In Chapter 3 we describe the modeling of an induction motor. Therefore, we first introduce Maxwell's equations and the magnetoquasistatic approximation of the equations. Then we give a brief introduction to electrical network modulation and PWM signals. Subsequently, the modeling of an induction motor is illustrated using the individual components. Chapter 4 introduces some iterative PinT methods and theoretically examines the MGRIT algorithm and the AT-MGRIT algorithm. The framework `PyMGRIT`, which implements the MGRIT and AT-MGRIT algorithms and is used in the numerical simulations that follow, is presented in Chapter 5. Both Chapter 6 and Chapter 7 present numerical experiments, with Chapter 6 exploring properties of the AT-MGRIT algorithm for a model problem and a chemical reaction. Chapter 7 contains several experiments on time-parallel simulation of an induction motor. In particular, we study here the effect of a discontinuous source on the MGRIT algorithm, the effect of spatial coarsening within the MGRIT algorithm for this model, and embed the AT-MGRIT algorithm in an optimization procedure for the geometry optimization of an induction motor model. In the subsequent Chapter 8, we present a new general performance model for iterative PinT methods and compare the predictions of the model with parallel simulations using four PinT libraries. Finally, we discuss the results of this work in Chapter 9 and provide an outlook for future research.

Chapter 2

Review of basic material

This chapter provides an overview of basic material used in the remainder of this thesis. First, three different classes of differential equations are introduced, namely ordinary differential equations, differential-algebraic equations, and partial differential equations. Second, we give a brief insight into the discretization, i. e., the replacement of the continuous problem by a discrete representation, of differential equations, yielding a system of algebraic equations. We then give a brief overview of how such linear and nonlinear systems of algebraic equations can be solved.

The individual sections of this overview chapter do not claim to be a full introduction to the various fields, but serve to introduce notations and concepts that will be used throughout this thesis.

2.1 Differential equations

A differential equation is an equation that establishes a relationship between one or more unknown functions and their derivatives. They play an important role in many disciplines such as engineering, physics, and biology, where the functions usually represent physical quantities, the derivatives indicate the rate of change of these quantities, and the differential equation defines a relationship between these two components. Differential equations are usually divided into classes based on their properties, such as ODEs, DAEs, or PDEs. An ODE contains one or more functions of an unknown variable, while a PDE contains partial derivatives. A DAE is a system of equations that contains both differential and algebraic equations.

2.1.1 Ordinary differential equations

An ODE is an equation of the form

$$F\left(t, u(t), u'(t), \dots, u^{(n-1)}(t), u^{(n)}(t)\right) = 0, \qquad t \in (0, T], \tag{2.1}$$

where $t \in \mathbb{R}$ is an independent variable, $u(t)$ is an unknown function, F is a given function and $T > 0$. The variable $n \geq 1$, given by the highest order of the derivative in (2.1), determines the order of the ODE. If (2.1) satisfies n initial conditions

$$u^{(i)}(0) = u_{i,0}, \quad i = 0, ..., n-1, \tag{2.2}$$

then (2.1) together with (2.2) is called *initial value problem.* The form (2.1) of an ODE is called *implicit*, while an ODE solved explicitly for the highest-order derivative, i. e.,

$$u^{(n)}(t) = f\left(t, u(t), u'(t), \dots, u^{(n-1)}(t)\right),$$

is called *explicit.*

Any ODE of order n can be transformed into a system of first-order ODEs [103]. Considering a system of d explicit first-order ODEs, the general form of an initial value problem can be written as

$$\mathbf{u}'(t) = \mathbf{f}\left(t, \mathbf{u}(t)\right), \qquad t \in (0, T], \tag{2.3}$$

$$\mathbf{u}(0) = \mathbf{u}_0, \tag{2.4}$$

where $\mathbf{u}(t) = [u_1(t), ..., u_d(t)]^\top \in \mathbb{R}^d$ denotes a d-dimensional vector, $\mathbf{u}_0 \in \mathbb{R}^d$ is the initial condition and $\mathbf{f} : [0, T] \times \mathbb{R}^d \to \mathbb{R}^d$ is a d-dimensional function.

The Picard-Lindelöf theorem [103, Theorem 7.1.1] assures the existence of a unique solution of the initial value problem (2.3)-(2.4) if the function $\mathbf{f}$ is Lipschitz continuous with Lipschitz constant L, i. e., if $\mathbf{f}$ satisfies

$$||\mathbf{f}(t, \mathbf{x}) - \mathbf{f}(t, \mathbf{y})|| \leq L||\mathbf{x} - \mathbf{y}||, \quad \text{for all } t \in [0, T], \quad \mathbf{x}, \mathbf{y} \in \mathbb{R}^d,$$

where $|| \cdot ||$ denotes some suitable norm.

2.1.2 Differential-algebraic equations

Consider a system

$$\mathbf{F}\left(t, \mathbf{u}(t), \mathbf{u}'(t)\right) = \mathbf{0}, \qquad t \in (0, T], \tag{2.5}$$

with function $\mathbf{F} : [0, T] \times \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ and unknown $\mathbf{u} : [0, T] \to \mathbb{R}^d$, $d \geq 2$. If $\frac{\partial \mathbf{F}}{\partial \mathbf{u}'}$ is nonsingular, (2.5) is a system of ODEs. If, though, $\frac{\partial \mathbf{F}}{\partial \mathbf{u}'}$ is singular, then (2.5) is a system of DAEs. DAEs can thus be viewed as a generalization of ODEs, although DAEs have different properties and are usually more difficult to solve [4]. An important class of DAEs are *semi-explicit* DAEs or *ODEs with constraints*, given by

$$\mathbf{u}_d'(t) = \mathbf{f}\left(t, \mathbf{u}_d(t), \mathbf{u}_a(t)\right), \tag{2.6}$$

$$\mathbf{0} = \mathbf{g}\left(t, \mathbf{u}_d(t), \mathbf{u}_a(t)\right), \tag{2.7}$$

where $\mathbf{u}_d(t) \in \mathbb{R}^{d_d}$, $\mathbf{u}_a(t) \in \mathbb{R}^{d_a}$, $\mathbf{f} : \mathbb{R}^{d_d+d_a+1} \to \mathbb{R}^{d_d}$ and $\mathbf{g} : \mathbb{R}^{d_d+d_a+1} \to \mathbb{R}^{d_a}$, i. e., the unknowns are divided into differential variables $\mathbf{u}_d$ and algebraic variables $\mathbf{u}_a$.

Various measures have been introduced to classify DAEs, such as the differentiation index or the pertubation index; for further details and index concepts, we refer to [91]. In this work, we only consider the differentiation index, hereafter simply called *index*, which indicates how many transformations must be performed to convert a DAE into an ODE. This index can be defined as follows.

Definition 2.1 ([57, Definition VII.1.2]). *The system* (2.5) *has index m if m is the minimal number of analytical differentiations*

$$\mathbf{F}\left(t, \mathbf{u}(t), \mathbf{u}'(t)\right) = \mathbf{0}, \ \frac{\mathrm{d}\mathbf{F}\left(t, \mathbf{u}(t), \mathbf{u}'(t)\right)}{\mathrm{d}t} = \mathbf{0}, \ \dots, \ \frac{\mathrm{d}^m\mathbf{F}\left(t, \mathbf{u}(t), \mathbf{u}'(t)\right)}{\mathrm{d}t^m} = \mathbf{0}, \tag{2.8}$$

such that equations (2.8) *allow us to extract an explicit ODE system $\mathbf{u}(t)' = \varphi\left(t, \mathbf{u}(t)\right)$ by using only algebraic manipulations.*

It follows that a system of ODEs can be classified as a system of DAEs with index $m = 0$. An example of an index-1 DAE is given by (2.6)-(2.7) if $\frac{\partial \mathbf{g}}{\partial \mathbf{u}_a}$ is nonsingular.

An important component to consider when solving a DAE is the correct treatment of the initial conditions. Considering the previous example of an index-1 DAE, only the initial condition for the differential variables $\mathbf{u}_d(0) = \mathbf{u}_{d,0}$ can be freely chosen, but not the initial condition for the algebraic variables $\mathbf{u}_a(0) = \mathbf{u}_{a,0}$. This initial condition must be calculated based on the algebraic constraint

$$\mathbf{0} = \mathbf{g}(0, \mathbf{u}_{d,0}, \mathbf{u}_{a,0}).$$

When considering higher indices, these constraints increase further [57]. In the following thesis, we consider only index-1 DAEs, which behave essentially like ODEs.

2.1.3 Partial differential equations

So far we have only considered differential equations with derivatives with respect to one variable. In this section, we will briefly introduce PDEs which contain partial derivatives. In the following we call an open and connected subset of $\mathbb{R}^d, d \in \mathbb{N}$, a *domain*, denoted by Ω and its boundaries by $\partial\Omega$. A general form of a PDE is given by

$$F\left(\mathbf{x}, u(\mathbf{x}), \frac{\partial u(\mathbf{x})}{\partial x_1}, \ldots, \frac{\partial u(\mathbf{x})}{\partial x_d}, \frac{\partial^2 u(\mathbf{x})}{\partial x_1 \partial x_1}, \ldots, \frac{\partial^2 u(\mathbf{x})}{\partial x_1 \partial x_d}, \ldots\right) = 0, \tag{2.9}$$

with $\mathbf{x} \in \mathbb{R}^d$, the unknown function $u(\mathbf{x})$, and where F depends on $\mathbf{x}$, the value of $u(\mathbf{x})$, and the partial derivatives of $u(\mathbf{x})$ at $\mathbf{x}$.

Since there is no general theory for PDEs of the form (2.9), it is common to classify PDEs based on their properties and characteristics. Similar to an ODE (cf. Section 2.1.1), the order n of a PDE is given by the highest order derivative in the equation. Furthermore, the equation can be classified according to its linearity, typically distinguishing between *linear*, *semilinear*, *quasilinear*, and *nonlinear* PDEs. A PDE is called linear if F depends only linearly on u and all partial derivatives, i. e., the coefficient functions depend only on variables $\mathbf{x}$. Semilinear PDEs, on the other hand, depend only linearly on the highest order partial derivatives, and a PDE is called quasilinear if the coefficient functions of the highest order partial derivatives depend only on $\mathbf{x}$, u, or lower order derivatives. If a PDE cannot be classified into one of these groups, it is called nonlinear.

Many physical problems can be classified as second-order linear PDEs. This class can be written as

$$\mathcal{A}u(\mathbf{x}) = -\sum_{i,j=1}^{d} a_{i,j}(\mathbf{x}) \frac{\partial^2}{\partial x_i \partial x_j} u(\mathbf{x}) + \sum_{j=1}^{d} b_j(\mathbf{x}) \frac{\partial}{\partial x_j} u(\mathbf{x}) + c(\mathbf{x}) u(\mathbf{x}) = f(\mathbf{x}). \tag{2.10}$$

For this particular and important class, there are further classifications based on the eigenvalues of the coefficient matrix $A = (a_{i,j})_{i,j=1}^{d}$ in (2.10). Within this class, PDEs are called elliptic if all eigenvalues of A have the same sign. If all eigenvalues of A have the same sign, except for one vanishing eigenvalue, the PDEs are called parabolic, and hyperbolic if all eigenvalues of A have the same sign, except for one eigenvalue which has the opposite sign.

To obtain a unique solution of a PDE, we need predescribed boundary conditions at the boundaries $\partial\Omega$, i. e., given values at the boundaries of the domain. Many different boundary conditions can be found in the literature, e. g., Dirichlet, Neuman, or periodic. More precisely, we speak of Dirichlet boundary conditions if the values of u at the boundaries are prescribed, i. e.,

$$u(\mathbf{x}) = d_1(\mathbf{x}), \qquad \mathbf{x} \in \partial\Omega,$$

where $d_1 : \partial\Omega \to \mathbb{R}$ is a given function. Neumann boundary conditions are given when values of the derivative (instead of values of the function) are specified at the boundaries, i. e.,

$$\frac{\partial u}{\partial \mathbf{n}}(\mathbf{x}) = d_2(\mathbf{x}), \qquad \mathbf{x} \in \partial\Omega,$$

where $d_2 : \partial\Omega \to \mathbb{R}$ is a given function, and $\frac{\partial u}{\partial \mathbf{n}}$ is the partial derivate of u with respect to the unit outward normal $\mathbf{n}$ of $\partial\Omega$. If the PDE is defined on a periodic domain, e. g., a torus, one does not need boundary conditions, in which case we speak of periodic boundaries.

A formally correct treatment of a PDE includes the proof of the existence and uniqueness of the solution, for which some basics of functional analysis are required. For further literature see [15, 17]. For more complex systems, the proof is not always straightforward, especially when nonlinear systems are considered. Since only well-known problems are considered in this thesis and this part is not the focus of this dissertation, we assume the existence of a unique solution of the PDEs considered in this thesis.

2.2 Discretization

The first step to numerically solve a differential equation is to find a discrete representation of the continuous problem. A typical approach for the discretization of time-dependent PDEs is the method of lines [56]. In the first step of the method, all dimensions except the time dimension, i. e., the spatial dimensions, are discretized. As a result, we obtain a system of ODEs or DAEs that depends only on the time dimension and is called a *semi-discrete* system. In a second step, we discretize the semi-discrete system in the time dimension using a time integration method and obtain a fully discrete system of algebraic equations. In the following, we first present methods for discretizing the spatial dimensions and then time integration methods.

2.2.1 Spatial discretization

There are several discretization methods for discretizing the space dimensions, e. g., finite differences, finite volumes, or finite elements, all of which have different approaches. In the following, we focus on the finite difference method and the finite element method, which will be used later in this thesis.

2.2.1.1 Finite differences

The basic idea of the finite difference scheme is to replace the derivatives in the differential operator equation, i. e.,

$$u'(x) := \lim_{h \to 0} \frac{u(x+h) - u(x)}{h},$$

for a function u at a point $x \in \mathbb{R}$, by difference quotients. Obviously, the smaller h is, the better the approximation becomes. It is possible to measure the discretization error of the approximation using the Taylor expansion

$$u(x+h) = u(x) + hu'(x) + \mathcal{O}(h^2), \tag{2.11}$$

where the $\mathcal{O}$-notation denotes the difference between the original function and the Taylor polynomial. From (2.11) we obtain the so-called *forward difference* approximation of u', given by

$$u'(x) = \frac{u(x+h) - u(x)}{h} + \mathcal{O}(h). \tag{2.12}$$

Equation (2.12) states that the error of the approximation of the derivative is of order h. In the same way, we can define other approximations, e. g., we can consider a point $x - h$ and obtain the *backward difference* approximation given by

$$u'(x) = \frac{u(x) - u(x-h)}{h} + \mathcal{O}(h).$$

To improve the order of accuracy, we can consider, for example, both $x + h$ and $x - h$, which gives us

$$u(x+h) = u(x) + hu'(x) + h^2 \frac{u''(x)}{2!} + h^3 \frac{u'''(x)}{3!} + \mathcal{O}(h^4),$$

and

$$u(x-h) = u(x) - hu'(x) + h^2 \frac{u''(x)}{2!} - h^3 \frac{u'''(x)}{3!} + \mathcal{O}(h^4).$$

By simple calculations we obtain the approximations for the first and second derivatives, given by

$$u'(x) = \frac{u(x+h) - u(x-h)}{2h} + \mathcal{O}(h^2),$$

and

$$u''(x) = \frac{u(x+h) + u(x-h) - 2u(x)}{h^2} + \mathcal{O}(h^2),$$

respectively. These approximations are called second-order *central differences* approximation of the first and second derivate. Based on the same strategy, but considering additional points, even higher-order approximations can be created. Furthermore, the method can be extended to higher dimensions following the same approach.

2.2.1.2 Finite element method

The finite element method is a flexible strategy for discretizing arbitrary domains based on basis functions and the *variational* (or *weak*) formulation of a PDE. The idea is to divide the entire domain Ω into small elements, e. g., triangles for a two-dimensional (2D) domain, and approximate the PDE locally on these elements. These approximations can then be combined to form a global system of equations that is the discrete representation of the problem. We begin by defining some standard function spaces and then demonstrate the methods with an example.

$C^k(\Omega)$: the set of k times continuously differentiable functions on Ω,

$C_0^k(\Omega)$: the set of functions $\phi \in C^k(\Omega)$ having compact support in Ω,

$C_0^k(\Omega)'$: the dual space of $C_0^k(\Omega)$,

$L^p(\Omega), 1 \leq p < \infty$: the set of functions ϕ on Ω for which $|\phi|^p$ is Lebesgue integrable.

The fundamental Sobolev spaces are given by

$$W^{s,p}(\Omega) := \{\phi \in L^p(\Omega) \ : \ \partial^\alpha \phi \in L^p(\Omega) \text{ for all } |\alpha| \leq s\},$$

with $s \in \mathbb{Z}_+$ and $1 \leq p < \infty$. Fixing $p = 2$ and using the L^2-inner product

$$\langle u, v \rangle := \int_\Omega uv \, \mathrm{d}x,$$

we obtain Hilbert spaces given by

$$H^s(\Omega) := \Big\{ u \in C_0^\infty(\Omega)' \ : \ u = U|_\Omega \text{ for some } U \in W^{s,2}\left(\mathbb{R}^d\right) \Big\},$$

where $s \in \mathbb{Z}_+$.

We illustrate the finite element method based on a simple example PDE, the Poisson equation

$$\begin{aligned} -\nabla^2 u(x) &= f(x), \quad x \in \Omega, \\ u(x) &= 0, \qquad\ \ x \text{ on } \partial\Omega, \end{aligned} \tag{2.13}$$

with unknown function $u = u(x)$, given function $f = f(x)$, and ∇^2 the Laplace operator. Further, we assume the solution u is in $V := H^1(\Omega)$. We define a set of functions $v \in V$, called *test functions*, which vanish at the boundaries where Dirichlet conditions are imposed, i. e., $v = 0$ on $\partial\Omega$. Multiplying (2.13) with v and integrating throughout Ω results in

$$-\int_\Omega (\nabla^2 u) v \, \mathrm{d}x = \int_\Omega f v \, \mathrm{d}x.$$

Using Green's first identity

$$-\int_{\Omega}(\nabla^2 u)v\,\mathrm{d}x = \int_{\Omega}\nabla u\cdot\nabla v\,\mathrm{d}x - \int_{\partial\Omega}\frac{\partial u}{\partial n}v\,\mathrm{d}s, \tag{2.14}$$

where $\frac{\partial u}{\partial n} = \nabla u \cdot n$ is the derivate of u in the outward normal direction n on the boundary, we obtain the weak formulation of the Poisson equation

$$\int_{\Omega}\nabla u\cdot\nabla v\,\mathrm{d}x = \int_{\Omega} fv\,\mathrm{d}x \quad \text{for all } v \in V. \tag{2.15}$$

Note that the second term on the right-hand side of (2.14) vanishes on the boundaries due to our choice of the test functions. We can rewrite (2.15) as

$$a(u,v) = l(v), \quad u, v \in V, \tag{2.16}$$

where a and l are the bilinear form and linear form given by

$$a(u,v) := \int_{\Omega}\nabla u\cdot\nabla v\,\mathrm{d}x, \quad \text{and} \quad l(v) := \int_{\Omega} fv\,\mathrm{d}x,$$

respectively. The finite element method discretizes the weak formulation by introducting a finite-dimensional space $V_h \subset V$ with basis $\phi_i, i = 1, ..., N$, where $\dim(V_h) = N$. Thereby, the idea is to use basis functions with a small support in the computational domain so that the final result is a sparse system matrix, i. e., a matrix with many zero entries. For this purpose, the domain is decomposed into elements and the basis functions are chosen as relatively low order polynomials on each of these elements, e. g., piecewise linear functions that are one at one node of an element and zero elsewhere. Similar to the finite difference method, better approximations can then be achieved by reducing the size of these elements. We can express the approximation $\tilde{u}$ of u and the test function v as a finite sum using the chosen basis of V_h, i. e.,

$$\tilde{u}(x) = \sum_{i=1}^{N}\alpha_i\phi_i(x), \quad \text{and} \quad v(x) = \sum_{j=1}^{N} v_j\phi_j(x), \tag{2.17}$$

respectively. Plugging (2.17) into (2.16) yields

$$a\left(\sum_{i=1}^{N}\alpha_i\phi_i(x), \sum_{j=1}^{N} v_j\phi_j(x)\right) = l\left(\sum_{j=1}^{N} v_j\phi_j(x)\right),$$

which can be transformed to

$$\sum_{j=1}^{N} v_j \sum_{i=1}^{N} a(\phi_i(x), \phi_j(x))\alpha_i = \sum_{j=1}^{N} v_j l\left(\phi_j(x)\right). \tag{2.18}$$

Finally, equation (2.18) can be rewritten in matrix form $\mathbf{v}^\top A\mathbf{u} = \mathbf{v}^\top \mathbf{f}$, where

$$\mathbf{v}^\top = (v_1, ..., v_N), \quad \mathbf{u}^\top = (\alpha_1, ..., \alpha_N), \quad \mathbf{f}^\top = \left(l(\phi_1(x)), ..., l\left(\phi_N(x)\right)\right),$$

and

$$A = \begin{pmatrix} a(\phi_1(x), \phi_1(x)) & \dots & a(\phi_1(x), \phi_N(x)) \\ \vdots & & \vdots \\ a(\phi_N(x), \phi_1(x)) & \dots & a(\phi_N(x), \phi_N(x)) \end{pmatrix}.$$

Since we force this for all $v \in V$ (see (2.15)), we can rewrite $\mathbf{v}^\top A\mathbf{u} = \mathbf{v}^\top \mathbf{f}$ as $A\mathbf{u} = \mathbf{f}$.

2.2.2 Temporal discretization

Consider an initial value problem of the form (2.3)-(2.4), which occurs, for example, after the spatial discretization of a space-time PDE. For simplicity, consider a uniformly distributed grid

$$\mathcal{T} := \{n\Delta t : n = 0, 1, \dots, N_t\} \tag{2.19}$$

with constant time step size Δt and N_t time steps, and let $\mathbf{u}_n$ be the approximation of the solution at time $t_n = n\Delta t$. The *local truncation error* [18], given by

$$l_n = \mathbf{u}(t_n) - \mathbf{u}_n,$$

defines the difference between the exact solution $\mathbf{u}(t_n)$ and the approximated solution $\mathbf{u}_n$. The order of a time integration method is a number $q \geq 1$ for which $l_n = \mathcal{O}(\Delta t^{q+1})$ holds. For the uniformly distributed grid (2.19), the θ-method [56, Section II.7] is given by

$$\mathbf{u}_{n+1} = \mathbf{u}_n + \theta\Delta t\mathbf{f}_n + (1-\theta)\Delta t\mathbf{f}_{n+1}, \quad n = 0, ..., N_t - 1, \tag{2.20}$$

where $0 \leq \theta \leq 1$ and $\mathbf{f}_n = \mathbf{f}(t_n, u_n)$. Different choices for θ lead to different one-step time integration schemes, of which the following three are the most common

$$\theta = \begin{cases} 0, & \text{for explicit Euler,} \\ 0.5, & \text{for trapezoidal rule,} \\ 1, & \text{for implicit Euler.} \end{cases}$$

The explicit and implicit Euler methods are of order one and the trapezoidal rule is of order two. Note that the methods are finite difference methods (Section 2.2.1.1), and that they belong to the family of Runge-Kutta methods, which also include higher-order time integration schemes. For more information on Runge-Kutta methods, we refer to [57].

In the following, we investigate stability properties of time integration methods. For this we consider Dahlquist's test equation

$$u'(t) = \lambda u, \quad \lambda \in \mathbb{C}. \tag{2.21}$$

A general one-step method for the problem (2.21) on a temporal grid (2.19) is given by

$$u_n = R(z)u_{n-1}, \tag{2.22}$$

where $z = \lambda \Delta t$.

Definition 2.2 ([57], Section IV.2)**.** *The function $R(z)$ in* (2.22) *is called stability function. The set*

$$S_d \coloneqq \{z \in \mathbb{C} : |R(z)| \leq 1\}$$

is called the stability domain of the method.

The stability function for the θ-method (2.20) is given by

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

thus, in particular

$$R_1(z) = 1 + z, \qquad R_2(z) = \frac{1 + 0.5z}{1 - 0.5z}, \qquad R_3(z) = \frac{1}{1-z},$$

for the explicit Euler method, the trapezoidal rule and the implicit Euler method, respectively. Time integration methods can be classified based on their stability properties; for further literature we refer to [25]. In the following, we present two classifications.

Definition 2.3 ([57], Section IV.3)**.** *A time-integration method, whose stability domain satisfies*

$$\mathbb{C}^- \coloneqq \{z \in \mathbb{C} : Re(z) \leq 0\} \subset S_d,$$

is called A-stable.

Definition 2.4 ([57], Section IV.3)**.** *A method is called L-stable if it is A-stable and*

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

It follows from the definitions that the explicit Euler method is not A-stable and therefore not L-stable. Both the implicit Euler method and the trapezoidal rule are A-stable, while the implicit Euler method is also L-stable, but the trapezoidal rule is not.

Another important concept when considering ODEs is the notion of stiffness, although there is no strict mathematical definition of this term. Vaguely, the term can be described as an ODE being stiff if explicit numerical methods for solving an equation are unstable, as long as the time step size is not chosen to be extremely small. Implicit methods, on the other hand, provide a stable solution to stiff problems regardless of the time step size. An example of a stiff problem including the study of numerical methods can be found in [57]. In the following thesis, only implicit methods suitable for solving stiff problems are used.

2.3 Solvers

The discretization of differential equations leads to a system of algebraic equations that must be solved to obtain the numerical solution of the problem. We first consider linear systems of the form

$$A\mathbf{u} = \mathbf{f}.$$

Depending on the properties of the system matrix A, the choice of the optimal solution strategy may vary. We typically distinguish between direct and iterative methods for solving the equation. Direct methods, such as Gaussian elimination, solve the system of equations exactly (up to rounding errors), but for large systems they have the disadvantage that they are very computationally intensive and require a lot of memory [107]. Iterative methods, on the other hand, improve a given initial guess $\mathbf{u}^0$ stepwise by computing a new approximation $\mathbf{u}^k$ in each iteration k until a certain solution quality is reached. In the following, we first introduce iterative methods and their properties. Subsequently, we consider Newton's method for solving nonlinear systems of equations.

2.3.1 Iterative methods

Consider a linear system

$$A\mathbf{u} = \mathbf{f},$$

with system matrix $A \in \mathbb{R}^{d\times d}$, solution vector $\mathbf{u} \in \mathbb{R}^d$ and vector $\mathbf{f} \in \mathbb{R}^d$. Now let $\hat{\mathbf{u}}$ be an approximation of the solution, then we define the *error*, $\mathbf{e}$, to the exact solution $\mathbf{u} = A^{-1}\mathbf{f}$ as

$$\mathbf{e} \coloneqq \mathbf{u} - \hat{\mathbf{u}}. \tag{2.23}$$

However, the error is usually not computable because the exact solution $\mathbf{u}$ is unknown. Therefore, we introduce another measure to determine the quality of the approximation compared to the exact solution, the so-called *residual*, given by

$$\mathbf{r} := \mathbf{f} - A\hat{\mathbf{u}}. \tag{2.24}$$

Using the definitions (2.23) and (2.24), we can derive the following relationship between the two measures,

$$A\mathbf{e} = \mathbf{r}. \tag{2.25}$$

Equation (2.25) is called *residual equation* and allows to calculate the solution $\mathbf{u}$ based on the approximation $\hat{\mathbf{u}}$ by the update

$$\mathbf{u} = \hat{\mathbf{u}} + \mathbf{e} = \hat{\mathbf{u}} + A^{-1}\mathbf{r}. \tag{2.26}$$

To solve equation (2.26), A must be inverted, which is essentially the same problem as solving $\mathbf{u} = A^{-1}\mathbf{f}$. Instead of inverting A directly, iterative methods are based on an approximation of the inverse of A, i. e., $\widetilde{A} \approx A^{-1}$. Based on this, a new and improved approximation $\hat{\mathbf{u}}^{k+1}$ can be obtained from an approximation $\hat{\mathbf{u}}^{k}$ by the update

$$\hat{\mathbf{u}}^{k+1} = \hat{\mathbf{u}}^{k} + \widetilde{A}\mathbf{r} = \hat{\mathbf{u}}^{k} + \widetilde{A}(\mathbf{f} - A\hat{\mathbf{u}}^{k}) = (I - \widetilde{A}A)\hat{\mathbf{u}}^{k} + \widetilde{A}\mathbf{f}.$$

If we define $S = I - \widetilde{A}A$ as iteration matrix, the general form of a stationary iterative procedure can be written as

$$\hat{\mathbf{u}}^{k+1} = S\hat{\mathbf{u}}^{k} + C(\mathbf{f}), \tag{2.27}$$

where $C(\mathbf{f})$ represents a set of operations on $\mathbf{f}$. The iterative procedure is called stationary because neither S nor $C(\mathbf{f})$ depend on the iteration. Since the solution $\mathbf{u}$ is a fixed point of the iteration, there is a relation between the errors of two successive iterations given by

$$\mathbf{e}^{k+1} = S\mathbf{e}^{k}.$$

Furthermore, it can be shown by a simple induction that

$$\mathbf{e}^{k} = S^{k}\mathbf{e}^{0}, \qquad k \geq 0, \tag{2.28}$$

where $\mathbf{e}^0$ is the error of the initial guess $\hat{\mathbf{u}}^0$. The choice of the iteration matrix S is crucial for the iterative method, and not all choices of S lead to a convergent method. From (2.28) we know that the method converges if $\lim_{k\to\infty} S^k = 0$. Therefore, we can determine a necessary and sufficient condition for the convergence of a method based on the *spectral radius* of S, defined in the following.

Theorem 2.5 ([103, Section 8.2])**.** *The stationary iterative method* (2.27) *is convergent, i. e.,* $\lim_{k\to\infty} S^k = 0$*, iff the condition*

$$\rho(S) < 1$$

holds, where

$$\rho(S) := \max\{|\lambda| : \lambda \textit{ eigenvalue of } S\}$$

denotes the spectral radius of the iteration matrix S.

Many iteration matrices are based on a splitting of the matrix A of the form

$$A = D - L - U, \tag{2.29}$$

where D is the diagonal of A, and $-L$ and $-U$ are the strictly lower and upper triangular parts of A, respectively. A common example of this splitting is the *Jacobi method*, which approximates the inverse of the matrix A by the inverse of the matrix D, i. e., $\widetilde{A} = D^{-1} \approx A^{-1}$. Thus, a new approximation can be calculated by

$$\hat{\mathbf{u}}^{k+1} = \hat{\mathbf{u}}^k + D^{-1}\mathbf{r}^k = D^{-1}(L+U)\hat{\mathbf{u}}^k + D^{-1}\mathbf{f}, \tag{2.30}$$

which yields the iteration matrix $S = D^{-1}(L+U)$ for the Jacobi method. By introducing a weighting factor $\omega \in \mathbb{R}$, $\omega > 0$, we obtain the *ω-Jacobi method*

$$\hat{\mathbf{u}}^{k+1} = \hat{\mathbf{u}}^k + \omega D^{-1}\mathbf{r}^k = \big((1-\omega)I + \omega D^{-1}(L+U)\big)\,\hat{\mathbf{u}}^k + \omega D^{-1}\mathbf{f}.$$

In contrast to the Jacobi method, the *Gauss-Seidel method* uses the components of the new approximation as soon as they have been calculated. We thus obtain

$$\hat{\mathbf{u}}^{k+1} = \hat{\mathbf{u}}^k + (D-L)^{-1}\mathbf{r}^k = (I - (D-L)^{-1}A)\hat{\mathbf{u}}^k + (D-L)^{-1}\mathbf{f}.$$

Again, weighting can be introduced so that we obtain the *ω-Gauss-Seidel method*, usually referred to as *successive over-relaxation* (SOR), which is given by

$$\hat{\mathbf{u}}^{k+1} = \hat{\mathbf{u}}^k \left(\frac{1}{\omega}D - L\right)^{-1}\mathbf{r}^k = \left(I - \left(\frac{1}{\omega}D - L\right)^{-1}A\right)\hat{\mathbf{u}}^k + \left(\frac{1}{\omega}D - L\right)^{-1}\mathbf{f}.$$

Block iterative methods

The discretization of differential equations often leads to a natural block structure of the system matrix A, i. e.,

$$A = \begin{bmatrix} A_{11} & \dots & A_{1N} \\ \vdots & & \vdots \\ A_{N1} & \dots & A_{NN} \end{bmatrix},$$

where $A_{ii}, i = 1, ..., N$ are square matrices. If all diagonal block matrices A_{ii} are nonsingular, the iterative methods previously considered for individual points can be extended to block versions. Let π be a given partition of A into blocks, we obtain a splitting relative to the partition π analogous to (2.29),

$$A = D_\pi - L_\pi - U_\pi,$$

where D_π contains the block diagonal of A, and $-L_\pi$ and $-D_\pi$ are the strictly lower and upper block triangular parts of A, respectively.

Then, analogously to (2.30), the *block Jacobi method* for solving a linear system of the form $A\mathbf{u} = \mathbf{f}$ can be defined as

$$\hat{\mathbf{u}}^{k+1} = D_\pi^{-1}(L_\pi + U_\pi)\hat{\mathbf{u}}^k + D_\pi^{-1}\mathbf{f}.$$

In the same way, the *block Gauss-Seidel method* and the *block SOR method* can be constructed.

2.3.2 Newton method

Consider a scalar nonlinear equation of the form

$$F(u) = 0,$$

where F is a continuously differentiable function. Furthermore, let $\hat{u}^0$ be an initial approximation of the unknown solution u. Expanding F in a Taylor series at the initial approximation $\hat{u}^0$, we obtain

$$0 = F(u) = F(\hat{u}^0) + (u - \hat{u}^0)F'(\hat{u}^0) + \mathcal{O}(|u - \hat{u}^0|^2).$$

Neglecting the higher-order terms and assuming $F'(\hat{u}^0) \neq 0$, we obtain the update

$$\hat{u}^1 = \hat{u}^0 - \frac{F(\hat{u}^0)}{F'(\hat{u}^0)} \tag{2.31}$$

for the first correction of the initial approximation. The Newton method is obtained by iterating (2.31) and is given by

$$\hat{u}^{k+1} = \hat{u}^k - \frac{F(\hat{u}^k)}{F'(\hat{u}^k)}, \qquad k = 0, 1, \dots .$$

Further generalization of Newton's method for a system of d nonlinear equations is straightforward, and we derive

$$\hat{\mathbf{u}}^{k+1} = \hat{\mathbf{u}}^k - J(\hat{\mathbf{u}}^k)^{-1}\mathbf{F}(\hat{\mathbf{u}}^k), \qquad k = 0, 1, \dots ,$$

where $\hat{\mathbf{u}} \in \mathbb{R}^d$, $\mathbf{F}$ is a vector-valued function, and $J(\hat{\mathbf{u}})^{-1}$ is the inverse of the Jacobian matrix.

The method can be extended by a damping factor $0 < \omega < 1$, resulting in Newton's method with damping given by

$$\hat{\mathbf{u}}^{k+1} = \hat{\mathbf{u}}^k - \omega J(\hat{\mathbf{u}}^k)^{-1}\mathbf{F}(\hat{\mathbf{u}}^k), \qquad k = 0, 1, \dots .$$

Newton's method converges quadratically if the initial guess $\hat{\mathbf{u}}^0$ is sufficiently close to the solution [24]. However, if the initial approximation is too far from the solution, Newton's method as well as Newton's method with damping may not converge.

Chapter 3

Governing application

In this chapter, we present the modeling of the governing application used for most of the numerical results in this thesis. More specifically, we present the modeling of a three-phase induction motor (also called an asynchronous motor), which is a common type of electrical machine used in many devices. Before presenting the model of an induction motor in Section 3.4, we first discuss the individual components used in this model. In Section 3.1 we introduce the fundamental laws of electromagnetism described by Maxwell's equations and the coupling of the equation by constitutive relations. We also introduce a simplification of Maxwell's equations called the magnetoquasistatic approximation, which is often used in the simulation of low-frequency devices. The fundamentals of electrical networks, which are an important part of many electromagnetic field models, are presented in Section 3.2. A special class of voltage sources within an electrical network are PWM sources, and their properties are briefly introduced in Section 3.3. This chapter is mainly based on the exposition in [66].

3.1 The Maxwell equations

Maxwell's equation, presented in its original form by J. C. Maxwell in 1864 [74], is a set of coupled differential equations describing the phenomena of electromagnetism. Maxwell's work combines and extends the work of many other well-known scientists such as Coulomb, Faraday, and others. A historical overview of the development of Maxwell's equations is given in [89].

For media at rest, the equations are given in integral form by

$$\int\limits_{\partial S} \mathbf{E} \cdot \mathrm{d}\mathbf{s} = -\int\limits_{S} \frac{\partial \mathbf{B}}{\partial t} \cdot \mathrm{d}\mathbf{S}, \tag{3.1}$$

$$\int\limits_{\partial S} \mathbf{H} \cdot \mathrm{d}\mathbf{s} = \int\limits_{S} \left(\frac{\partial \mathbf{D}}{\partial t} + \mathbf{J} \right) \cdot \mathrm{d}\mathbf{S}, \tag{3.2}$$

$$\int\limits_{\partial V} \mathbf{D} \cdot \mathrm{d}\mathbf{S} = \int\limits_{V} \varrho \, \mathrm{d}V, \tag{3.3}$$

$$\int\limits_{\partial V} \mathbf{B} \cdot \mathrm{d}\mathbf{S} = 0, \tag{3.4}$$

where $\mathbf{E}$ is the electric field strength, $\mathbf{B}$ is the magnetic flux density, $\mathbf{H}$ is the magnetic field strength, $\mathbf{D}$ is the electric flux density, $\mathbf{J}$ is the electric current density, and ϱ is the electric charge density. All fields are functions of space $\mathbf{x} \in \mathbb{R}^3$ and time $t \in \mathbb{R}$. Moreover, $V \subset \mathbb{R}^3$ is a connected volume with closed boundary surface ∂V and $S \subset \mathbb{R}^2$ is a connected surface with closed boundary curve ∂S. The curve element $\mathrm{d}\mathbf{s}$ can be written as $\mathrm{d}\mathbf{s} = \tau \, \mathrm{d}s$, oriented in the direction of the unit vector τ and tangent to ∂S. Similarly, the surface element $\mathrm{d}\mathbf{S}$ can be written as $\mathrm{d}\mathbf{S} = \mathbf{n} \, \mathrm{d}S$ oriented in the direction of the outward unit vector $\mathbf{n}$ and normal to the surface S. If $S = \partial V$ is the enclosed surface bounding a volume V, then $\mathbf{n}$ points outward from the enclosed volume V.

Using Stokes' theorem and Gauss' theorem given by

$$\int\limits_{\partial S} \mathbf{F} \cdot \mathrm{d}\mathbf{s} = \int\limits_{S} \nabla \times \mathbf{F} \cdot \mathrm{d}\mathbf{S},$$

and

$$\int\limits_{\partial V} \mathbf{F} \cdot \mathrm{d}\mathbf{S} = \int\limits_{V} \nabla \cdot \mathbf{F} \, \mathrm{d}V,$$

respectively, the integral form of Maxwell's equations (3.1)-(3.4) can be converted to their differential form given by

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t}, \tag{3.5}$$

$$\nabla \times \mathbf{H} = \frac{\partial \mathbf{D}}{\partial t} + \mathbf{J}, \tag{3.6}$$

$$\nabla \cdot \mathbf{D} = \varrho, \tag{3.7}$$

$$\nabla \cdot \mathbf{B} = 0. \tag{3.8}$$

The equations (3.1) and (3.5) are called *Faraday's law* or *law of electromagnetic induction* and state that electric fields are generated by alternating magnetic fields. The second equation, *Maxwell-Ampère's law* given by (3.2) and (3.6), states that circulating magnetic fields are generated by alternating electric fields and by electric currents. The *electric Gauss law*, given by (3.3) and (3.7), describes that electric fields diverge from electric charge, and the *magnetic Gauss law*, given by (3.4) and (3.8), that there are no magnetic monopoles. Although the four equations hold for the whole space $(\mathbf{x}, t) \in \mathbb{R}^3 \times \mathbb{R}$, in the following we restrict ourselves to a finite space-time domain $\Omega \times [0, T]$ with $T > 0$, with an open, bounded and connected domain $\Omega \subset \mathbb{R}^3$.

Maxwell's equations are completed by the following constitutive relations

$$\mathbf{D} = \epsilon \mathbf{E}, \tag{3.9}$$

$$\mathbf{J} = \sigma \mathbf{E} + \mathbf{J}_s, \tag{3.10}$$

$$\mathbf{H} = \nu \mathbf{B}, \tag{3.11}$$

where $\epsilon > 0$ is the electric permittivity, $\sigma \geq 0$ is the electric conductivity, $\nu > 0$ is the magnetic reluctivity, and $\mathbf{J}_s$ is the source current density. The source current density can be written as

$$\mathbf{J}_s(\mathbf{x}, t) = \sum_{j=1}^{n_s} \chi_j(\mathbf{x}) i_j(t), \tag{3.12}$$

using winding functions $\chi_j(\mathbf{x}) \in \mathbb{R}^3$ [96] that spatially distribute the currents $i_j(t) \in \mathbb{R}$ flowing through n_s stranded conductors. For the electrical permittivity and electrical conductivity, we assume linear dependencies, i. e., $\epsilon = \epsilon(\mathbf{x})$ and $\sigma = \sigma(\mathbf{x})$, while due to magnetic saturation in ferromagnetic materials, we assume nonlinear behavior of the magnetic reluctivity, i. e., $\nu = \nu(\mathbf{x}, |\mathbf{B}|)$. The properties of the reluctivity function are usually described by the relation of the magnetic flux density $\mathbf{B}$ on the magnetic field strength $\mathbf{H}$, also called *BH-curve*. Figure 3.1 shows an example of such a BH-curve for iron [35]. For further details and theory on BH-curves we refer to [85, 86].

Note that in practice the domain Ω usually consists of different materials and that the material properties depend on the domain. In Figure 3.2, the domain Ω consists of three subdomains Ω_{Fe}, Ω_{Cu} and Ω_{Air}, which represents areas of iron, copper and air, respectively, and $\Omega = \Omega_{\text{Fe}} \cup \Omega_{\text{Cu}} \cup \Omega_{\text{Air}}$.

3.1.1 Magnetoquasistatic approximation

In low-frequency applications, such as electric motors, it is often sufficient to consider an approximation of Maxwell's equations, the so-called magnetoquasistatic

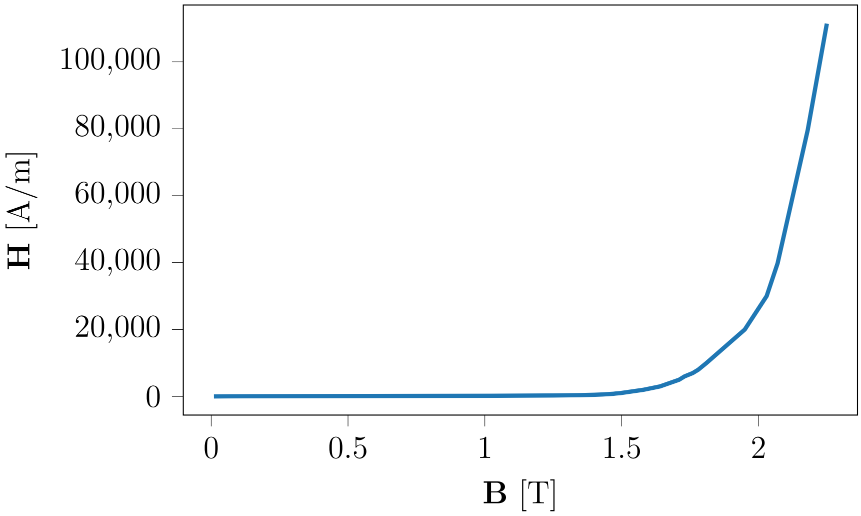


Figure 3.1: BH-curve of iron [35].

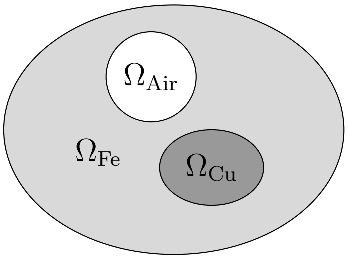


Figure 3.2: Multi material domain.

approximation, since in this regime the displacement currents are negligible compared to the source currents [63], i. e.,

$$\left|\frac{\partial \mathbf{D}}{\partial t}\right| \ll |\mathbf{J}|\,. \tag{3.13}$$

Using (3.13) to neglect the displacement current in Maxwell-Ampère's law (3.6), and then applying the constitutive relations (3.10) and (3.11), we obtain

$$\nabla \times (\nu \mathbf{B}) = \sigma \mathbf{E} + \mathbf{J}_s. \tag{3.14}$$

For the unique definition of electromagnetic fields in the domain Ω we impose conditions at the boundary $\Gamma = \partial\Omega$. Two typical choices for the boundary conditions are the electric and the magnetic boundary conditions [1], given by

$$\mathbf{n} \times \mathbf{E} = 0 \quad \text{on } \Gamma_E, \qquad \text{and} \qquad \mathbf{n} \times \mathbf{H} = 0 \quad \text{on } \Gamma_M, \tag{3.15}$$

respectively, where the boundary consists of two parts Γ_E and Γ_M with $\Gamma_E \cup \Gamma_M = \Gamma$ and $\Gamma_E \cap \Gamma_M = \emptyset$ and $\mathbf{n}$ denotes the outward normal vector to Γ.

Since the divergence of the curl operator in the magnetic Gauss law (3.8) is zero, we conclude the existence of the *magnetic vector potential* $\mathbf{A} = \mathbf{A}(\mathbf{x}, t)$ [47], such that

$$\mathbf{B} = \nabla \times \mathbf{A}. \tag{3.16}$$

Using that the curl of any gradient field is zero and substituting (3.16) into Faraday's law (3.5), we obtain

$$\mathbf{E} = -\frac{\partial A}{\partial t} - \nabla\phi, \tag{3.17}$$

where $\phi = \phi(\mathbf{x}, t)$ is the *electric scalar* potential. Note that (3.16) does not define the magnetic vector potential uniquely, since arbitrary curl-free components can be added to the magnetic potential without changing the observed magnetic field [1]. Therefore, in order to ensure the uniqueness of the solution, the concept of gauge is introduced. One possibility is to use the Coulomb gauge [1], given by the gauge condition $\nabla \cdot \mathbf{A} = 0$. For further literature on gauging, see for example [1]. Plugging (3.16) and (3.17) into (3.14), we get the $\mathbf{A}$-*ϕ-formulation*

$$\nabla \times (\nu\nabla \times \mathbf{A}) = -\sigma\frac{\partial \mathbf{A}}{\partial t} - \sigma\nabla\phi + \mathbf{J}_s.$$

Introducing

$$\mathbf{A}^* = \mathbf{A} + \int\limits_0^t \nabla\phi \,\mathrm{d}s, \qquad t \in [0, T],$$

we obtain the $\mathbf{A}^*$-formulation, also called *eddy current equation*, given by

$$\nabla \times (\nu\nabla \times \mathbf{A}^*) = -\sigma\frac{\partial \mathbf{A}^*}{\partial t} + \mathbf{J}_s. \tag{3.18}$$

The boundary conditions (3.15) for each time $t \in [0, T], T > 0$ are now given in terms of the magnetic vector potential by

$$\mathbf{n} \times \mathbf{A}^* = 0 \text{ on } \Gamma_E, \quad \text{and} \quad \mathbf{n}(\nu\nabla \times \mathbf{A}^*) = 0 \text{ on } \Gamma_M,$$

which corresponds to homogeneous Dirichlet and homogeneous Neumann boundary conditions, respectively. Further, an initial condition

$$\mathbf{A}^*(\mathbf{x}, 0) = \mathbf{A}^*_0, \quad \mathbf{x} \in \Omega,$$

is required for the unique solvability. In the following, we abuse notation and denote the magnetic vector potential in the $\mathbf{A}^*$-formulation by $\mathbf{A}$.

In many applications the three-dimensional (3D) problem is reduced to a 2D problem given by a cross section, assuming that the model is invariant in the axial x_3-direction. We get $\mathbf{B}(\mathbf{x}, t) = [\mathbf{B}_1(x_1, x_2, t), \mathbf{B}_2(x_1, x_2, t), 0]^\top$ and thus $\mathbf{J}_s(\mathbf{x}, t) =$

$[0, 0, \mathbf{J}_{s,3}(x_1, x_2, t)]^\top$ and $\mathbf{A}(\mathbf{x}, t) = [0, 0, \mathbf{A}_3(x_1, x_2, t)]^\top$. Based on this assumption, we can reformulate equation (3.18) to

$$\sigma \frac{\partial \mathbf{A}_3}{\partial t} - \nabla \cdot (\nu \nabla \mathbf{A}_3) = \mathbf{J}_{s,3}, \qquad (\mathbf{x}, t) \in \Omega \times (0, T], \tag{3.19}$$

where $\Omega \in \mathbb{R}^2$ is an open, bounded and connected 2D domain, and $T > 0$. Note that in 2D the gauge $\nabla \cdot \mathbf{A} = 0$ is automatically satisfied. Abusing notations for the case that (3.19) is nonlinear, the equation is of parabolic type for $\sigma > 0$ and of elliptic type if $\sigma = 0$ [66].

Considering homogeneous Dirichlet boundary conditions

$$\mathbf{A}_3(\mathbf{x}, t) = 0, \qquad (\mathbf{x}, t) \in \partial\Omega \times [0, T], \tag{3.20}$$

at the boundaries $\partial\Omega$ of Ω and an initial condition

$$\mathbf{A}_3(\mathbf{x}, 0) = \mathbf{A}_{3,0}(\mathbf{x}), \qquad \mathbf{x} \in \Omega, \tag{3.21}$$

at initial time $t = 0$, we obtain the 2D eddy current initial value problem (3.19)-(3.21).

3.2 Electric circuit theory

Electrical networks are an important component of many applications, such as power converters or microprocessors. Electrical networks consist of circuit elements, branches given by two-pole circuit elements, and nodes connecting two or more circuit elements (branches) (see Figure 3.3(b)). The connection conditions of electrical networks are given by Kirchhoff's laws, more specifically *Kirchoff's current law* and *Kirchoff's voltage law*. Kirchhoff's current law is given by

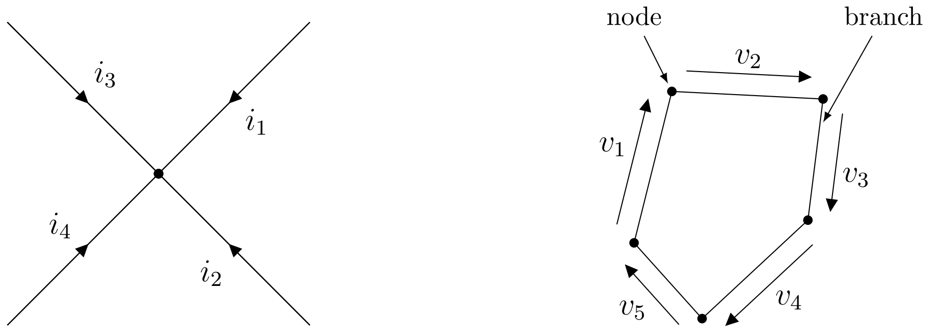
$$\sum_{j=1}^{N_b} \pm i_j = 0, \tag{3.22}$$

and states that the sum of the currents flowing into any node must be zero [88]. In (3.22), N_b defines the number of branches entering the node, where the sign depends on whether the current enters or leaves the node. Figure 3.3(a) shows an example with $N_b = 4$. Note that (3.22) can be derived from Maxwell-Ampère's law (3.2) [90].

Kirchhof's voltage law states that the sum of the voltage drops across each element along a closed loop must be zero [88] and is given by

$$\sum_{j=1}^{N_l} \pm v_j = 0, \tag{3.23}$$

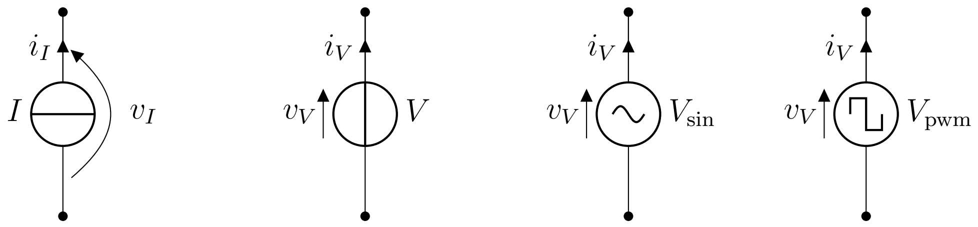
where N_l denotes the number of branches forming a loop. Again, the signs are determined by the orientation of the branches. Figure 3.3(b) shows an illustration of Kirchhoff's voltage law for $N_l = 5$ branches. Note that (3.23) can be derived by Faraday's law (3.1) [90].



(a) Kirchhoff's current law.

(b) Kirchhoff's voltage law.

Figure 3.3: Visualization of Kirchhoff's current and voltage laws.

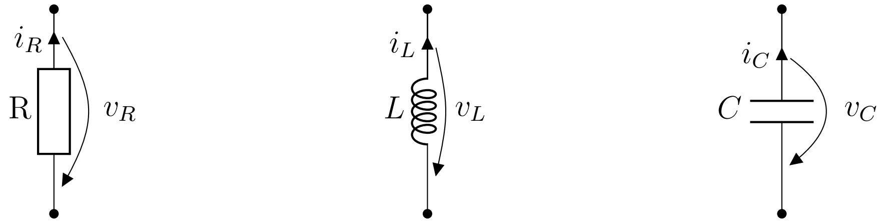


(a) Direct current source.

(b) Direct voltage source.

(c) Sinusoidal voltage source.

(d) PWM voltage source.



(e) Resistor.

(f) Inductor.

(g) Capacitor.

Figure 3.4: Graphical representation of the various circuit components. The upper four are active elements, while the lower three represent passive elements.

Electrical networks consist of elements, such as current or voltage sources, resistors or inductors, which are represented in Figure 3.4. We distinguish between *active* and *passive* elements. Active circuit elements supply the circuit with energy and are usually provided by voltage or current sources. Four different active elements are illustrated in Figure 3.4(a) - Figure 3.4(d). Figure 3.4(a) and Figure 3.4(b) show constant current or direct current (DC) and constant voltage sources. In contrast, Figure 3.4(b) and Figure 3.4(c) show elements with time-dependent input power provided by sinusoidal and PWM voltage sources, respectively. Time-dependent sinusoidal current or alternating current (AC) and PWM elements look analogously.

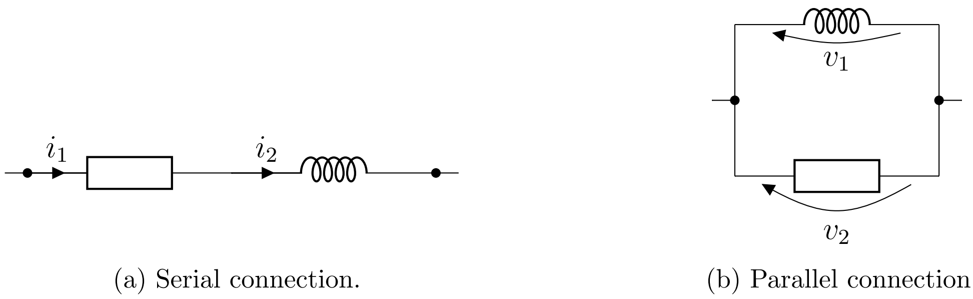
Circuit elements that do not contain an active source, such as resistors (Figure 3.4(e)), inductors (Figure 3.4(f)), or capacitors (Figure 3.4(g)), are called passive elements. These elements define time-dependent constitutive relations of voltages and currents, given by

$$v_R(t) = R i_R(t), \tag{3.24}$$

$$v_L(t) = L \frac{\mathrm{d}}{\mathrm{d}t} i_L(t), \tag{3.25}$$

$$i_C(t) = C \frac{\mathrm{d}}{\mathrm{d}t} v_C(t), \tag{3.26}$$

for a resistor, an inductor and a capacitor, respectively. In the equations (3.24)-(3.26), R is the resistance, L is the inductance, and C is the capacitance. These relations, which may also be nonlinear, can be derived from Maxwell's equations (3.5)-(3.8) and the constitutive relations (3.9)-(3.11) [90].



(a) Serial connection.

(b) Parallel connection.

Figure 3.5: Two connections types of two elements in a circuit.

Circuit elements can be connected either in series, in parallel, or in any combination of these two ways, see Figure 3.5. In Figure 3.5(a) a resistor and an inductor are connected in series, and it follows from Kirchhoff's rules that the current is the same for all elements. In Figure 3.5(b) the same components are connected in parallel, and it follows from Kirchhoff's law that the voltage is the same for all elements.

Using the modified node analysis [60] or the modified loop analysis [90] and the previously described laws and relations, a mathematical model can be created for any given circuit. This mathematical model usually consists of (a system of) ODEs or DAEs.

3.3 PWM signals

Pulse-width modulation is a technique that produces a square wave signal that oscillates between different states. In electronic devices, such as power converters, this discontinuous signal has technical advantages and can be realized by switching semiconductors on and off. While there are various forms of PWM signals [11], we consider PWM signals generated by comparing a reference signal with a carrier signal, i. e., we consider signals with constant switching frequency f_s of the form

$$b(t) = \operatorname{sgn}\left[r(t) - c(t)\right], \quad t \in \mathbb{R},$$

where $r(t)$ denotes a reference signal and $c(t)$ denotes a carrier signal. Note that the constant switching frequency f_s is part of the carrier signal.

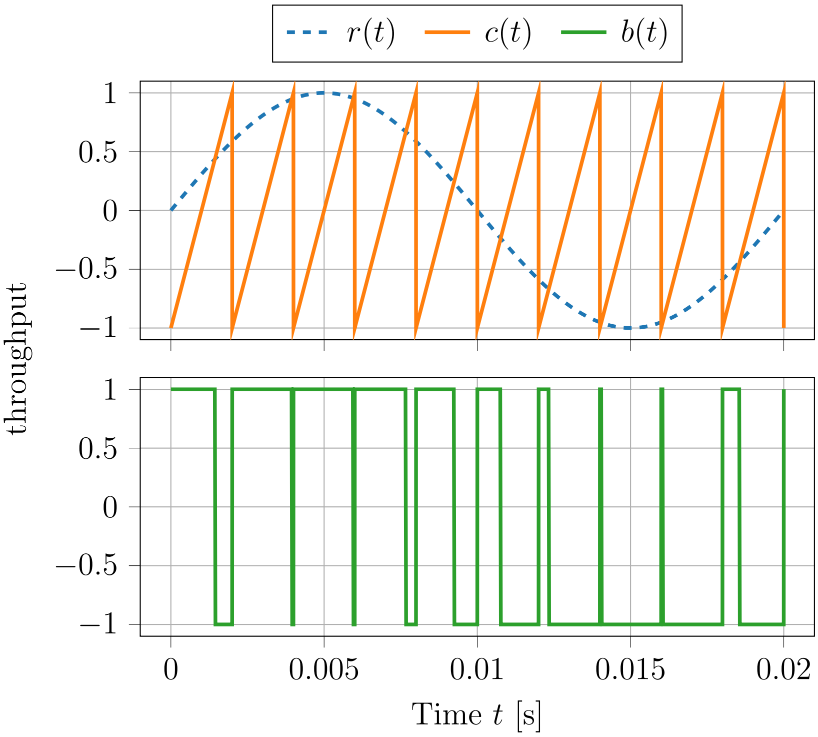


Figure 3.6: Illustration of a PWM signal (green, lower plot), generated by a sine wave reference signal (blue, upper plot) and a sawtooth carrier signal (orange, upper plot).

The exact choice of switching frequency, reference signal, and carrier signal depends on the application, with, for example, constant or sinusoidal signals being a typical choice for the reference signal [109], and the sawtooth carrier, inverted sawtooth carrier, and triangle carrier being typical variants for the carrier signal [109]. Figure 3.6 shows an example of a PWM signal for $t \in [0, 0.02]$, with the reference signal (blue dashed line) and carrier signal (orange solid line) in the top subplot and the resulting PWM signal (green solid line) in the bottom subplot. The reference signal is given by a sinusoidal reference signal of 50 Hz, and the carrier signal is modeled by a sawtooth carrier of 500 Hz. The resulting PWM signal of $f_s = 500\,\text{Hz}$ alternates between 1 and -1, corresponding to the on and off states of an electrical device. It can be seen that the switching between the two states of the PWM signal occurs exactly at the intersection points of the reference and carrier signals.

Note that when a discontinuous PWM signal is used as the right-hand side of an ODE, the standard Picard-Lindelöf theory for existence and uniqueness of the solution cannot be applied. However, the unique solvability of such a system can be shown using the theory of Carathéodory's differential equations; we refer for more details to [3, 66].

3.4 Induction motors

Electric machines are electrical devices that convert electrical energy into mechanical energy. Here, we introduce a specific type of electric motors, namely three-phase induction motors. This class of motors is the most widespread type of electric motors in the power range below 500 kW.

Three-phase induction motors, as the name implies, are powered by a three-phase current or voltage source. A three-phase voltage source v_j is given by

$$v_j(t) = \hat{V} \sin\bigl(2\pi f t - (j-1) \cdot 2\pi/3\bigr), \quad j = 1, 2, 3,$$

where f is the frequency and $\hat{V}$ is the amplitude of the voltage source. Note that the three phases have the same amplitude and frequency, but differ in time phase. An example of such a three-phase voltage source with $f = 50\,\text{Hz}$ and amplitude $\hat{V} = 311.1\,\text{V}$ in the time interval $[0, 0.02]$ is shown in Figure 3.7. In the same way, a three-phase current source $i_j(t)$ can be defined.

Figure 3.8 shows the model of a 2D induction motor. The motor consists of a rotor (inner part), a stator (outer part) and an air gap between the stator and the rotor. As rotor, we consider a squirrel cage rotor consisting of solid conductor bars inserted into the rotor slots and connected at both ends by conductive end rings. The stator contains stator slots that contain the windings that carry the three-phase voltage. In Figure 3.8, each phase is arranged on three consecutive slots, so that each pair of three of the same phase occurs four times in the stator. This arrangement results in the magnetic flux distributed on four poles, i. e., $p = 2$ north-south pole pairs. Typically, the windings carrying the three-phase current are star or delta connected (see Figure 3.9). When a three-phase current is applied to these specially connected windings, a uniformly rotating magnetic

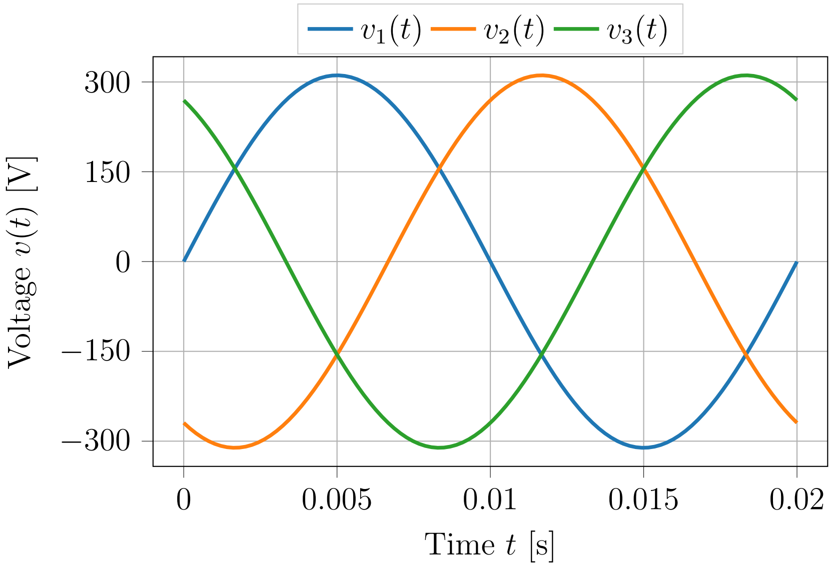


Figure 3.7: Example of a three-phase voltage source for a frequency of $f = 50\,\mathrm{Hz}$ and an amplitude of $\hat{V} = 311.1\,\mathrm{V}$

field with synchronous speed ω_{sync} is generated. The air gap between the rotor and stator is traversed by the magnetic flux, allowing current to flow in the rotor bars. This results in the Lorentz force acting on the cage, causing the rotor to rotate at mechanical speed ω_{mech}. Finally, the generated electromagnetic torque T_{EM} is transmitted to the mechanical load via a shaft inside the motor.

Characteristic for induction motors is that the synchronous speed ω_{sync} and the mechanical speed ω_{mech} are slightly different, with the mechanical speed being smaller than the synchronous speed. Note that induction motors are also called asynchronous motors because of this asynchrony. The relative difference between the two speeds is called *slip* and is given by

$$s = \frac{\omega_{\text{sync}} - \omega_{\text{mech}}}{\omega_{\text{sync}}} = 1 - \frac{\omega_{\text{mech}}}{\omega_{\text{sync}}}.$$

From this, in addition to the asynchronous case, i. e., $0 < s < 1$, two special cases can be derived. In the case of $s = 0$ the rotor rotates with synchronous speed, i. e., under no-load conditions. In the case of $s = 1$ we speak of a stationary rotor, i. e., under locked rotor operation. For a more detailed description of three-phase induction motors, we refer to [66].

The electromagnetic torque T_{EM} can be calculated using the formula [2]

$$T_{\text{EM}} = \int_S \mathbf{r} \times \boldsymbol{\sigma} \cdot \mathrm{d}\mathbf{S} = \int_S \mathbf{r} \times (\boldsymbol{\sigma} \cdot \mathbf{n})\mathrm{d}S, \tag{3.27}$$

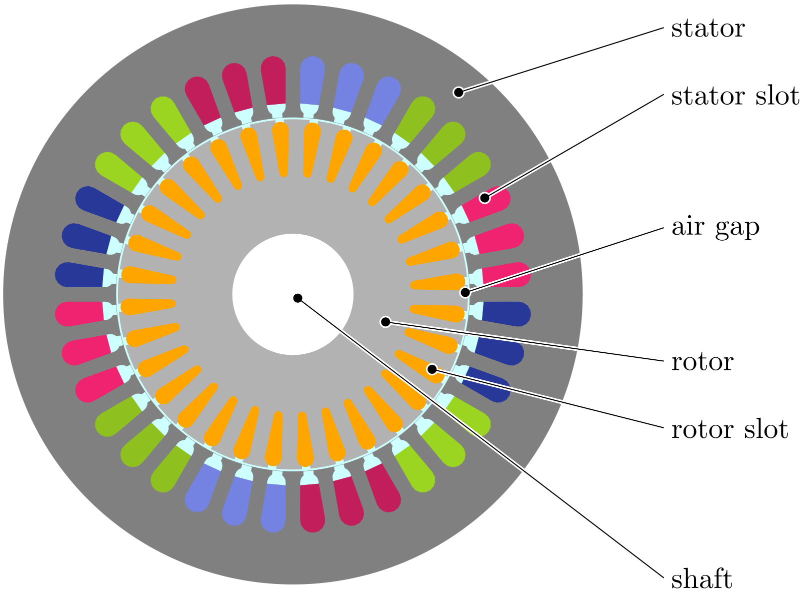
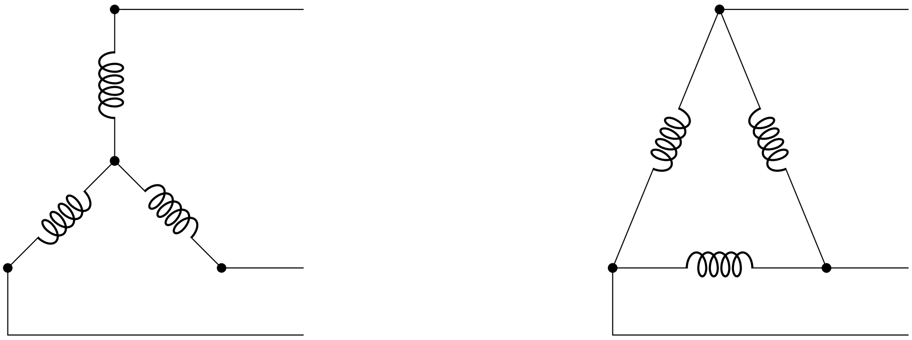


Figure 3.8: Cross section of the four-pole squirrel cage induction motor “im_3_kw” of [48].



(a) Star connection.

(b) Delta connection.

Figure 3.9: Connections in three-phase windings.

where S is the surface enclosing the rotor, $\mathbf{r}$ is the position vector connecting the rotor origin to S, $\mathbf{n}$ is the unit normal vector to S, and $\boldsymbol{\sigma}$ is the *Maxwell stress tensor* [95, Section 6.3], given by

$$\boldsymbol{\sigma}_{ij} = \nu_0(\mathbf{B}_i\mathbf{B}_j - 0.5|\mathbf{B}|^2\delta_{ij}), \quad i,j = 1,2,3,$$

with the reluctivity in vacuum ν_0, the magnetic flux density $\mathbf{B}$, and the Kronecker delta δ_{ij}. Note that the product of the generated torque and the speed defines the *mechanical power* P_{mech}, i. e.,

$$P_{\text{mech}} = T_{\text{EM}}\omega_{\text{mech}}. \tag{3.28}$$

Part of the power supplied is lost as heat, which can be calculated using Joule losses. For the 2D-setting from Section 3.1.1 and the length ℓ_3 of the motor in axial x_3-direction these losses are given by

$$P_{\mathrm{loss}} = \int\limits_{\Omega_{2\mathrm{D}}} \sigma \left(\frac{\partial \mathbf{A}_3}{\partial t} \right)^2 \ell_3 \,\mathrm{d}\Omega_{2\mathrm{D}}, \tag{3.29}$$

where $\mathbf{A}_3$ denotes the x_3-component.

Finally, we obtain a mathematical model for the simulation of induction motors. The electromagnetic phenomena inside the machine are described by the eddy current problem (3.19)-(3.21), which for $(\mathbf{x}, t) \in \Omega \times (0, T]$, $\Omega \in \mathbb{R}^3$ and $T > 0$ is given by

$$\sigma(\mathbf{x}) \frac{\partial \mathbf{A}}{\partial t} + \nabla \times (\nu(|\mathbf{x}, \nabla \times \mathbf{A}|) \nabla \times \mathbf{A}) = \sum_{j=1}^{3} \chi_j(\mathbf{x}) i_j(t), \tag{3.30}$$

$$\mathbf{n} \times \mathbf{A} = 0, \qquad \text{on}\, \partial\Omega \times (0, T], \tag{3.31}$$

$$\mathbf{A}(\mathbf{x}, 0) = \mathbf{A}_0(\mathbf{x}), \qquad \mathbf{x} \in \Omega, \tag{3.32}$$

where the definition of the winding functions (3.12) is used. For the connected electrical network we obtain

$$v_j(t) = R_j i_j(t) + \int\limits_{\Omega} \chi_j(\mathbf{x}) \cdot \frac{\partial \mathbf{A}(\mathbf{x}, t)}{\partial t} \mathrm{d}\Omega, \qquad j = 1, 2, 3\,, \tag{3.33}$$

where R_j denotes the resistance of the stator stranded conductors. The motion of the rotor can be described by the mechanical equation

$$J\omega'_{\mathrm{mech}}(t) + C\omega_{\mathrm{mech}}(t) = T_{\mathrm{EM}}(t, \theta(t), \mathbf{A}) - T_{\mathrm{load}}, \tag{3.34}$$

$$\theta'(t) = \omega_{\mathrm{mech}}(t), \tag{3.35}$$

where J is the moment of inertia, C is the coefficient of friction, θ is the rotor angle, and T_{load} is the shaft torque.

Reducing the 3D problem to a 2D problem as described in Section 3.1.1, and discretizing (3.30)-(3.32) using finite elements with d_a degrees of freedom together with (3.33)-(3.35), we obtain a semi-discrete system of equations of the form

$$M\mathbf{u}'(t) + K(\mathbf{u}(t))\mathbf{u}(t) = \mathbf{f}(t), \quad t \in [0, T], \tag{3.36}$$

$$\mathbf{u}(t_0) = \mathbf{u}_0, \tag{3.37}$$

with unknowns $\mathbf{u}^\top = [\mathbf{a}^\top, \mathbf{i}^\top, \theta, \omega_{\mathrm{mech}}]^\top : [0, T] \to \mathbb{R}^d$ and initial condition $\mathbf{u}_0 \in \mathbb{R}^d$. The solution $\mathbf{u}(t) \in \mathbb{R}^d$ consists of the magnetic vector potential $\mathbf{a}(t) \in \mathbb{R}^{d_a}$ at time $t \in (0, T]$, the currents of the three phases $\mathbf{i}(t)$, the rotor angle θ, and $\omega_{\mathrm{mech}}(t) \in \mathbb{R}$ ($d = d_a + 5$). The right-hand side $\mathbf{f}(t)$ contains the given voltages $\mathbf{v}(t) \in \mathbb{R}^3$ and the mechanical excitation. Note that the problem (3.36) is in general a system of index-1 DAEs, since the matrix M is singular when considering regions with non-conducting materials. Integrating the semi-discrete system (3.36)-(3.37) using implicit Euler yields a system of the form

$$\begin{aligned} \left(\frac{1}{\Delta t} M + K\left(\mathbf{u_i}\right)\right) \mathbf{u_i} &= \mathbf{f_i} + \frac{1}{\Delta t} M \mathbf{u}_{i-1}, \\ \mathbf{u}_0 &= \mathbf{u}(0). \end{aligned}$$

Chapter 4

Parallel-in-time integration

In the following, we present methods for solving the initial value problem

$$\mathbf{u}'(t) = \mathbf{f}(t, \mathbf{u}(t)), \quad \mathbf{u}(0) = \mathbf{u}_0, \quad t \in (0, T]. \tag{4.1}$$

The classical approach for solving a problem of the form (4.1) is based on discretizing the equation on a temporal grid with N_t time steps, followed by time stepping, the sequential application of a time integrator. Time stepping yields the discrete solution after exactly N_t application of the time integrator, but is completely sequential in the time domain. In contrast to this sequential approach, PinT methods allow parallelization in the time domain. Based on the review article [39], PinT methods can be divided into the following classes:

- Shooting-type time parallel methods,
- domain decomposition methods in space-time,
- multigrid methods in space-time,
- direct solvers in space-time.

Each category consists of many different methods, which often allow various variations and modifications. Probably the best known and most studied method is Parareal [71], which can be interpreted in a variety of frameworks of numerical schemes, e. g., as a multiple-shooting method or as a multigrid method in time. PFASST [29] is based on spectral deferred correction (SDC) [28] and allows parallelization in space-time using SDC on a space-time hierarchy. The MGRIT algorithm [30] applies the principles of multigrid reduction in the time domain. Other examples of PinT methods include waveform relaxation [21, 34], space-time multigrid [61], or revisionist integral deferred correction [20]. For a comprehensive overview, we refer to [39, 83].

In more detail, the idea of the MGRIT algorithm is based on a multilevel hierarchy, with the finest level representing the mathematical model of interest discretized to the desired accuracy. At the coarser levels, simplified models are considered, with the MGRIT algorithm typically considering coarser temporal grids to reduce the complexity of the problem, but (additional) coarsening in space and/or the use of simplified mathematical models is also possible. Thereby, each level is characterized by a time integrator defining a time step on the respective grid. At the coarsest level of the multilevel hierarchy, the entire temporal domain is considered, while at all other levels temporal subdomains are considered in parallel. A key property of the MGRIT algorithm is its non-intrusiveness, i. e., existing time integrators can be integrated into the MGRIT framework without making too many changes to the existing code. The algorithm has been theoretically studied in [26, 59, 99] and successfully applied to various problems, e. g., linear and nonlinear parabolic problems [30, 33], compressible fluid dynamics [31], power systems [69, 98], linear advection [23, 62], and machine learning [49, 79, 97]. The use of spatial coarsening in MGRIT was investigated for the p-Laplacian [33] and for the Burgers equation [62].

The structure of the multilevel hierarchy, i. e., the choice of the number of levels and the choice of the coarsest grid, within the MGRIT algorithm is both critical and challenging. The typical choice of the coarse grid in the two-level setting is based on the number of processes, choosing as many points on the coarse grid as there are processes available [71]. With this strategy, the fine level can be perfectly parallelized, but for a large number of processes, the serial work on the coarsest level dominates the runtime. Using more than two grid levels can significantly reduce the serial work by using a coarsest grid with only a few time points, but the resulting very large time steps can be very expensive, if not infeasible, to compute for some applications [13] and/or may affect the convergence of the algorithm [22]. In [55] we have introduced a new way to define the coarsest level in MGRIT, emphasizing reducing the serial work while avoiding large time steps. Instead of solving the entire time interval serially on the coarsest grid, we define multiple independent local coarse grids each consisting of w coarse-grid time points that can be propagated independently and simultaneously. Due to the asynchronous nature of computing the truncated coarsest grids, we refer to the new algorithm as “asynchronous truncated MGRIT” (AT-MGRIT).

In the following, the established PinT methods Parareal, PFASST, MGRIT are first presented based on the review of published literature. Thereupon, the new method AT-MGRIT, which was developed in the context of this thesis, is presented. The focus of this work is on the last two methods, which are therefore described in more detail. Finally, we briefly present various convergence criteria used in the different methods to determine the quality of the solution and terminate the iteration process when the quality of the solution is sufficient.

4.1 Parareal

Let

$$0 = t_0 < t_1 < ... < t_{N_t} = T \tag{4.2}$$

be a decomposition of the time domain $(0, T]$ of the initial value problem (4.1) into N_t non-overlapping time intervals $(t_{n-1}, t_n]$, $n = 1, ..., N_t$. The Parareal algorithm is based on two propagators, a fine but expensive operator $\mathcal{F}(t_n, t_{n-1}, \mathbf{u}_{n-1})$ and a coarse, but cheap operator $\mathcal{G}(t_n, t_{n-1}, \mathbf{u}_{n-1})$. Here, a propagator is an operator which, based on an initial value $\mathbf{u}_{n-1}$ at time t_{n-1}, provides an approximate solution to the initial value problem at time t_n. The idea of the iterative Parareal algorithm is to alternately apply the fine and the coarse propagator and improve the solution step by step. Let $\mathbf{u}_n^k$ be the approximation for the time points t_n, $n = 0, ..., N_t$ in the k-th Parareal iteration, and $\mathbf{u}_0^0, ..., \mathbf{u}_{N_t}^0$ be a given initial guess for the algorithm. Typically, this initial guess is obtained by applying the coarse operator as predictor

$$\mathbf{u}_0^0 = \mathbf{u}_0, \tag{4.3}$$

$$\mathbf{u}_n^0 = \mathcal{G}(t_n, t_{n-1}, \mathbf{u}_{n-1}^0) \text{ for } n = 1, ..., N_t, \tag{4.4}$$

but other choices are also possible.

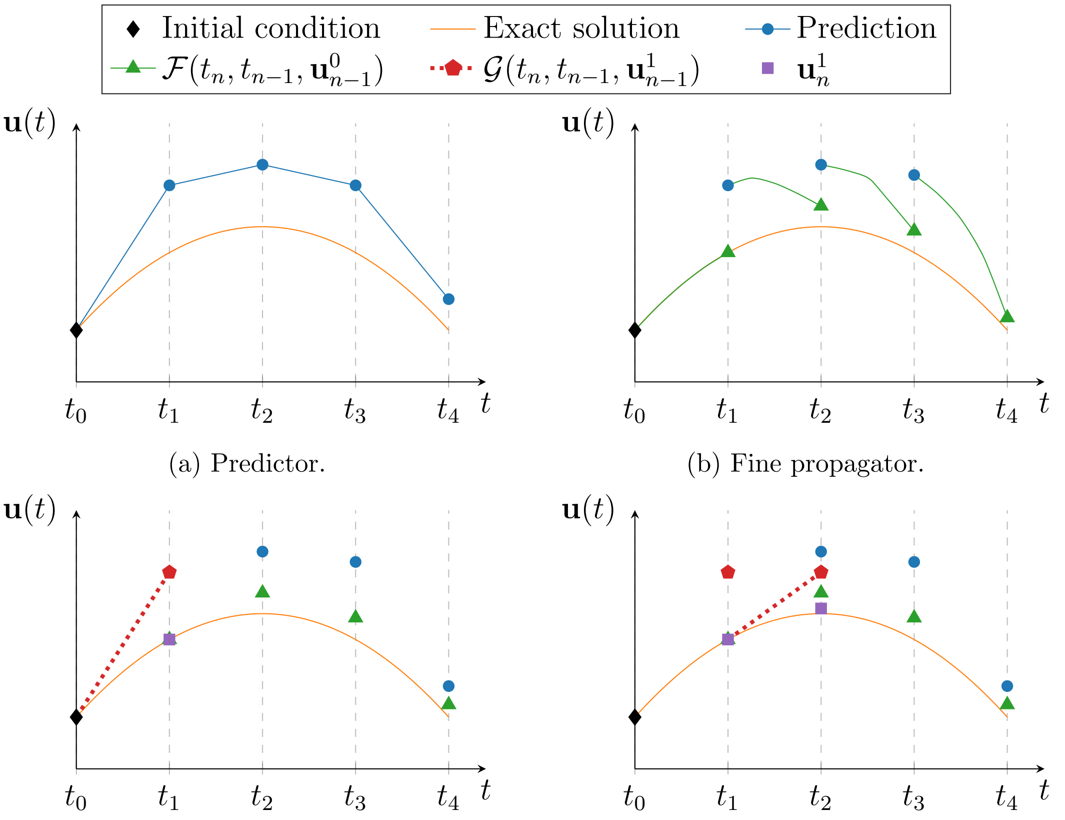
Then, a Parareal iteration is given for $n = 1, ..., N_t$ by the following rule

$$\mathbf{u}_0^{k+1} = \mathbf{u}_0, \tag{4.5}$$

$$\mathbf{u}_n^{k+1} = \mathcal{F}(t_n, t_{n-1}, \mathbf{u}_{n-1}^k) + \mathcal{G}(t_n, t_{n-1}, \mathbf{u}_{n-1}^{k+1}) - \mathcal{G}(t_n, t_{n-1}, \mathbf{u}_{n-1}^k). \tag{4.6}$$

Figure 4.1 sketches the functioning of Parareal for $N_t = 4$. First, the predictor (4.3)-(4.4) is used to obtain the initial values $\mathbf{u}_n^0, n = 1, ...N_t$, see Figure 4.1(a). Starting from these initial values, the fine operator $\mathcal{F}$ is applied (Figure 4.1(b), green triangles) to compute intermediate approximations of the solution. The next step is to serially apply the coarse operator to provide another intermediate approximation, which can be seen for the first two time points t_1 and t_2 by the pentagons in Figure 4.1(c) and Figure 4.1(d). The intermediate approximations are then used to correct the approximation using (4.6), resulting in a new approximation of the solution after the first iteration given by the squares in Figure 4.1(c) and Figure 4.1(d) for the first two time points. Note that after the first iteration, the fine solution of the propagator $\mathcal{F}$ is obtained for the first subinterval $[t_0, t_1]$. This is a special property of Parareal, which is mathematically reflected in the following convergence result [40].

Theorem 4.1 ([40, Theorem 1]). *Let the partition* (4.2) *be uniformly distributed, i. e., all temporal subdomains have exactly the same size* $\Delta t = T/N_t$, *and let*



(a) Predictor.

(b) Fine propagator.

(c) Coarse propagator and correction at t_1.

(d) Coarse propagator and correction at t_2.

Figure 4.1: Visualization of the Parareal method. Starting from an initial guess (top left), the fine operator can be applied to all time intervals simultaneously (top right), leading to an intermediate approximation (green triangles). Then, each time point is successively corrected based on another intermediate solution (red dot), leading to a new approximation (purple square) (bottom left and right for the first two time points).

the right-hand side $\mathbf{f}$ *in* (4.1) *be sufficiently smooth. Furthermore, assume that* $\mathcal{F}(t_n, t_{n-1}, \mathbf{u}_{n-1})$ *and* $\mathcal{G}(t_n, t_{n-1}, \mathbf{u}_{n-1})$ *are an exact solution and an approximate solution for a subinterval* $[t_{n-1}, t_n]$*, where the local truncation error is bounded by* $C_3\Delta t^{q+1}$ *with* $q \geq 1$ *and can be expanded for small* Δt *and an initial value* $\mathbf{u}$ *as*

$$\mathcal{F}(t_n, t_{n-1}, \mathbf{u}) - \mathcal{G}(t_n, t_{n-1}, \mathbf{u}) = c_{q+1}(\mathbf{u})\Delta t^{q+1} + c_{q+2}(\mathbf{u})\Delta t^{q+2} + ...,$$

where the functions $c_j, j = q+1, q+2, ...$ *are continuously differentiable. Furthermore, we assume that* $\mathcal{G}$ *satisfies the Lipschitz condition*

$$||\mathcal{G}(t + \Delta t, t, \mathbf{u}) - \mathcal{G}(t + \Delta t, t, \mathbf{v})|| \leq (1 + C_2\Delta t)||\mathbf{u} - \mathbf{v}||,$$

for $t \in [0, T]$*, for all* $\mathbf{u}, \mathbf{v}$*, and a constant* C_2*. Then, at iteration* k *of Parareal*

(4.5)-(4.6) *for a constant* $C_1 > 0$*, we have the bound*

$$||\mathbf{u}(t_n) - \mathbf{u}_n^k|| \leq \frac{C_3}{C_1} \frac{(C_1 \Delta t^{q+1})^{k+1}}{(k+1)!} (1 + C_2 \Delta t)^{n-k-1} \prod_{j=0}^{k} (n-j). \tag{4.7}$$

In Equation (4.7) the previously described property can be observed, more precisely one can see that the error of Parareal vanishes at a time t_n in iteration $k = n$ for $n = 1, ..., N_t$. Thus, in each iteration, the exact solutions given by the fine operator are propagated exactly one time subdomain further, so that after $k = N_t$ iterations one obtains the same solution as in the sequential application of the fine propagator. In practice, we assume that only $K \ll N_t$ iterations are necessary to obtain a certain quality of the solution. Note that this assumption is crucial for reducing simulation times compared to sequential time stepping. Various convergence criteria are briefly discussed in Section 4.5.

The Parareal algorithm (4.5)-(4.6), including the computation of an initial guess (4.3)-(4.4), can be written in a data-driven formulation as in Algorithm 4.1. Note that the loops in lines 6 and 8 depend on the current iteration of the algorithm. This ensures that the propagators are only applied to time subintervals where the solution does not match the solution of the sequential application of the fine propagator.

Algorithm 4.1: Parareal

```
1  $\mathbf{u}_0^0 \leftarrow \mathbf{u}_0$
2  for $n \leftarrow 1$ to $N_t$ do
3      $\tilde{\mathbf{u}}_n^0 \leftarrow \mathcal{G}(t_n, t_{n-1}, \mathbf{u}_{n-1}^0)$
4      $\mathbf{u}_n^0 \leftarrow \tilde{\mathbf{u}}_n^0$
5  for $k \leftarrow 1$ to $N_t$ do
6      foreach $n \in \{k, k+1, \dots, N_t\}$ do
7          $\hat{\mathbf{u}}_n^{k-1} \leftarrow \mathcal{F}(t_n, t_{n-1}, \mathbf{u}_{n-1}^{k-1})$
8      for $n \leftarrow k$ to $N_t$ do
9          $\tilde{\mathbf{u}}_n^k \leftarrow \mathcal{G}(t_n, t_{n-1}, \mathbf{u}_{n-1}^{\min(k,n-1)})$
10         $\mathbf{u}_n^k \leftarrow \tilde{\mathbf{u}}_n^k + \hat{\mathbf{u}}_n^{k-1} - \tilde{\mathbf{u}}_n^{k-1}$
11     if convergence criterion is reached then
12         break
```

4.2 PFASST

Let (4.2) be again a decomposition of the time domain $(0, T]$ of the initial value problem (4.1). The idea of the PFASST algorithm consists of a combination of the

SDC time integration and multigrid algorithms. Here, we briefly describe SDC for a single time interval and then introduce the time-parallel PFASST method. For a single time step from t_{n-1} to t_n, the Picard formulation for (4.1) is given by

$$\mathbf{u}(t) = \mathbf{u}_0 + \int_{t_{N_t-1}}^{t} \mathbf{f}(\mathbf{u}(s))\,\mathrm{d}s, \quad t \in [t_{n-1}, t_n].$$

Considering $M+1$ quadrature nodes $\tau_{n,(0)}, \tau_{n,(1)}, ..., \tau_{n,(M)}$ with $t_{n-1} = \tau_{n,(0)}$ and $t_n = \tau_{n,(M)}$, we can formulate the linear or nonlinear collocation problem

$$(I - \Delta t Q \hat{\mathcal{F}})(\overline{\mathbf{u}}_n) = \overline{\mathbf{u}}_{n,0},$$

where $\Delta t = t_n - t_{n-1}$, $\overline{\mathbf{u}}_n = (\mathbf{u}_{n,(0)}, ..., \mathbf{u}_{n,(M)})^T \approx (\mathbf{u}(\tau_{n,(0)}), ..., \mathbf{u}(\tau_{n,(M)}))^T$, $\overline{\mathbf{u}}_{n,0} = (\mathbf{u}_{n,(0)}, ..., \mathbf{u}_{n,(0)})^T$, the function $\hat{\mathcal{F}}(\overline{\mathbf{u}}_n) = ((\mathbf{f}(\mathbf{u}_{n,(0)}, \tau_{n,(0)}), ..., \mathbf{f}(\mathbf{u}_{n,(M)}, \tau_{n,(M)}))^T$ represents the right-hand side of the problem evaluated at each collocation node, and Q is an integration matrix. The SDC method solves this dense and potentially nonlinear system of equations with a fixed-point iteration, where the matrix Q is preconditioned by applying a low-order time integration scheme, such as backward or forward Euler. One iteration of the SDC method is called "sweep" and can be written as

$$\overline{\mathbf{u}}_n^{k+1}, \overline{\mathbf{f}}_n^{k+1} = \texttt{SDCSweep}(\overline{\mathbf{u}}_n^k, \overline{\mathbf{f}}_n^k, t_n, t_{n-1}),$$

where $\overline{\mathbf{f}}_n = \hat{\mathcal{F}}(\overline{\mathbf{u}}_n)$ and the first entry $\mathbf{u}_{n,(0)}$ of $\overline{\mathbf{u}}_n^k$ is used to form $\overline{\mathbf{u}}_{n,0}$.

Considering all N_t time steps simultaneously, we can formulate the so-called composite collocation problem, given as

$$\begin{bmatrix} I - \Delta t Q \hat{\mathcal{F}} & & & \\ -H & I - \Delta t Q \hat{\mathcal{F}} & & \\ & \ddots & \ddots & \\ & & -H & I - \Delta t Q \hat{\mathcal{F}} \end{bmatrix} \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \vdots \\ \mathbf{u}_{N_t} \end{bmatrix} = \begin{bmatrix} \mathbf{u}_{0,0} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix}, \tag{4.8}$$

where the matrix H copies the information from one time step to the next, see [14] for more details.

PFASST solves problem (4.8) using multigrid techniques and SDC. Thereby, PFASST typically uses a two-level strategy, which we also use for the following description, but note that the method is not limited to two levels. The coarse level contains a simplified version of the problem, which can be given, for example, by a reduction of the spatial problem, a reduction of the quadrature nodes, or a simplified representation of the problem. The two-level PFASST method can be described as follows: First, PFASST smoothes the problem at the fine level using a block Jacobi preconditioner, which is equivalent to applying one sweep per time point based on known values of the last iteration. Then, the fine approximation is transported to the coarse level using an operator R_p. Note that R_p depends on the coarsening strategy. Using the full approximation storage (FAS) framework [16], the $\boldsymbol{\tau}$-correction is computed. The coarse system is relaxed using a block Gauss-Seidel preconditioner, where the $\boldsymbol{\tau}$-correction is added to the problem. The error correction is computed based on the FAS formulation, transported to the fine level using an interpolation operator P_p, and then added to the fine approximation. The multilevel PFASST method for L levels and K iterations can be written in a data-driven formulation as in Algorithm 4.2. The presented PFASST algorithm corresponds to the newer multilevel view of the algorithm. The classical view of the PFASST method consists of individual multilevel SDC sweeps that exchange information at specific points. This classical view can be achieved by merging and moving the loops in lines 10, 18, and 24 outside of the loops for the levels (after line 8). Also, the classical view has a slightly different data flow; a good comparison between the two views can be found in [102]. The PFASST algorithm provides many different options for computing an initial guess, commonly referred to as prediction phase, such as a fine-level sweep for all time points or a burn-in at the coarsest level [29]. In Algorithm 4.2, we omit the prediction phase for simplicity. Other variations of the method include a variable number of sweeps at each level and/or skipping the first or last fine-level sweep within each iteration.

4.3 MGRIT

We discretize (4.1) on a uniformly-spaced temporal grid $\mathcal{T} = \{n\Delta t : n = 0, 1, \ldots, N_t\}$, with time points $t_n = n\Delta t$ and constant step size $\Delta t = T/N_t$, and let $\mathbf{u}_n \approx \mathbf{u}(t_n)$ for $i = 0, \ldots, N_t$ with $\mathbf{u}_0 = \mathbf{u}(0)$. A general form of a single step time integration method for the time-discrete initial value problem is

$$\mathbf{u}_n = \boldsymbol{\Phi}_n(\mathbf{u}_{n-1}) + \mathbf{g}_n, \quad n = 1, 2, \ldots, N_t, \tag{4.9}$$

where $\boldsymbol{\Phi}_n$ is a one-step time integrator, propagating a solution $\mathbf{u}_{n-1}$ from a time point t_{n-1} to time point t_n, and $\mathbf{g}_n$ contains forcing terms. Equation (4.9) can be written as a semi-linear matrix equation

$$A(\mathbf{u}) \equiv \begin{bmatrix} I & & & \\ -\boldsymbol{\Phi}_1(\cdot) & I & & \\ & \ddots & \ddots & \\ & & -\boldsymbol{\Phi}_{N_t}(\cdot) & I \end{bmatrix} \begin{bmatrix} \mathbf{u}_0 \\ \mathbf{u}_1 \\ \vdots \\ \mathbf{u}_{N_t} \end{bmatrix} = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_{N_t} \end{bmatrix} \equiv \mathbf{g}, \tag{4.10}$$

where $\boldsymbol{\Phi}_n(\cdot)$ indicates that $\boldsymbol{\Phi}_n$ is nonlinearly evaluated at the corresponding (block) vector entry. This system can be solved by a (linear) sequential block

Algorithm 4.2: PFASST-multigrid

```
foreach $i \in \{1, 2, \dots, N_t\}$ do                                   ▷ Initialize points
    $\mathbf{u}_{i,0}^{0,0} \leftarrow \mathbf{u}_0$
    for $m \leftarrow 1$ to $M^0$ do
        $\overline{\mathbf{u}}_{i,m}^{0,0} \leftarrow \mathbf{0}$
    $\overline{\mathbf{f}}_i^{0,0} \leftarrow$ FEvalAll$(\overline{\mathbf{u}}_i^{0,0})$
for $k \leftarrow 1$ to $K+1$ do                                          ▷ PFASST iterations
    foreach $i \in \{1, 2, \dots, N_t\}$ do
        $\boldsymbol{\tau}_i^{k,0} \leftarrow \mathbf{0}$
    for $\ell \leftarrow 0$ to $L-2$ do                                   ▷ Down cycle
        foreach $i \in \{1, 2, \dots, N_t\}$ do
            if $i > 1$ then
                $\overline{\mathbf{u}}_{i,0}^{k-1,\ell} \leftarrow \overline{\mathbf{u}}_{i-1,M^\ell}^{k-1,\ell}$
                $\overline{\mathbf{f}}_{i,0}^{k-1,\ell} \leftarrow$ FEvalSingle$(\overline{\mathbf{u}}_{i,0}^{k-1,\ell})$
            $\overline{\mathbf{u}}_i^{k,\ell}, \overline{\mathbf{f}}_i^{k,\ell} \leftarrow$ SDCSweep$(\overline{\mathbf{u}}_i^{k-1,\ell}, \overline{\mathbf{f}}_i^{k-1,\ell}, \boldsymbol{\tau}_i^{k,\ell}, t_i, t_{i-1})$
            $\overline{\mathbf{u}}_i^{k-1,\ell+1} \leftarrow$ RestrictAll$(\overline{\mathbf{u}}_i^{k,\ell})$
            $\overline{\mathbf{f}}_i^{k-1,\ell+1} \leftarrow$ FEvalAll$(\overline{\mathbf{u}}_i^{k-1,\ell+1})$
            $\boldsymbol{\tau}_i^{k,\ell+1} \leftarrow$ FAS$(\overline{\mathbf{f}}_i^{k,\ell}, \overline{\mathbf{f}}_i^{k-1,\ell+1}, \boldsymbol{\tau}_i^{k,\ell})$
    for $i \leftarrow 1$ to $N_t$ do                                      ▷ Coarsest level
        if $i > 1$ then
            $\overline{\mathbf{u}}_{i,0}^{k-1,L-1} \leftarrow \overline{\mathbf{u}}_{i-1,M^{L-1}}^{k,L-1}$
            $\overline{\mathbf{f}}_{i,0}^{k-1,L-1} \leftarrow$ FEvalSingle$(\overline{\mathbf{u}}_{i,0}^{k-1,L-1})$
        $\overline{\mathbf{u}}_i^{k,L-1}, \overline{\mathbf{f}}_i^{k,L-1} \leftarrow$ SDCSweep$(\overline{\mathbf{u}}_i^{k-1,L-1}, \overline{\mathbf{f}}_i^{k-1,L-1}, \boldsymbol{\tau}_i^{k,L-1}, t_i, t_{i-1})$
    for $\ell \leftarrow L-2$ to $0$ do                                   ▷ Up cycle
        foreach $i \in \{1, 2, \dots, N_t\}$ do
            $\overline{\mathbf{v}}_i^{k,\ell} \leftarrow \overline{\mathbf{u}}_i^{k,\ell} +$ InterpolateAll$(\overline{\mathbf{u}}_i^{k,\ell+1} - \overline{\mathbf{u}}_i^{k-1,\ell+1})$
            $\overline{\mathbf{f}}_i^{k,\ell} \leftarrow$ FEvalAll$(\overline{\mathbf{v}}_i^{k,\ell})$
            if $i > 1$ then
                $\overline{\mathbf{v}}_{i,0}^{k,\ell} \leftarrow \overline{\mathbf{v}}_{i-1,M^\ell}^{k,\ell}$
                $\overline{\mathbf{f}}_{i,0}^{k,\ell} \leftarrow$ FEvalSingle$(\overline{\mathbf{v}}_{i,0}^{k,\ell})$
            $\overline{\mathbf{u}}_i^{k,\ell}, \overline{\mathbf{f}}_i^{k,\ell} \leftarrow$ SDCSweep$(\overline{\mathbf{v}}_i^{k,\ell}, \overline{\mathbf{f}}_i^{k,\ell}, \boldsymbol{\tau}_i^{k,\ell}, t_i, t_{i-1})$
    if *convergence criterion is reached* then
        break
```

forward solve. Note that the notation here has changed slightly from the previous two sections, although the underlying problem and idea is the same. We follow here the typical notation for the MGRIT algorithm, which offers some advantages for the theoretical study of the algorithm.

In the following, we first present the linear two-level MGRIT method for solving linear problems of form (4.10). Thereupon, we extend the two-level method to the multilevel case using the FAS scheme to solve both nonlinear and linear problems. We also present some extensions of the method. We then present a convergence analysis of the linear two-level MGRIT method based on [26, 99].

4.3.1 Two-level MGRIT

For a given fine temporal grid $\mathcal{T}^{(0)} = \{n\Delta t^{(0)} : n = 0, 1, \ldots, N_t^{(0)}\}$, and a given integer coarsening factor $m > 1$, we define a splitting of the fine grid points into F- and C-points, where every m-th point is a C-point and all other points are F-points. Note that non-uniform coarsening is also possible; uniform coarsening is used here to simplify the presentation. Looking only at the C-points, we obtain a global coarse grid $\mathcal{T}^{(1)} = \{n\Delta t^{(1)} : n = 0, 1, \ldots, N_t^{(1)}\}$, with $N_t^{(1)} = N_t^{(0)}/m$ and time step $\Delta t^{(1)} = m\Delta t^{(0)}$, as shown in Figure 4.2.

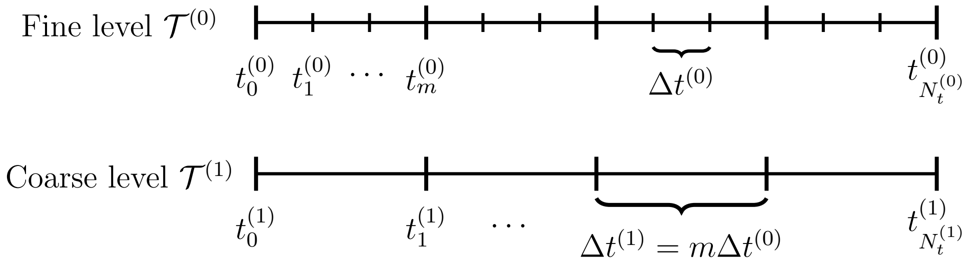
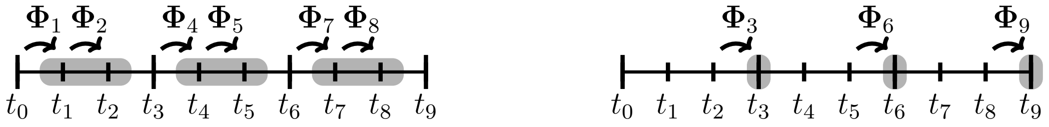


Figure 4.2: Uniform coarsening strategy of the MGRIT algorithm. The fine grid $\mathcal{T}^{(0)}$ is based on a coarsening factor m divided into C-points (long markers) and F-points (short markers), where the C-points form the coarse grid $\mathcal{T}^{(1)}$.

The linear two-level MGRIT algorithm uses this time-grid hierarchy to solve time-dependent problems of the form (4.10), where we first consider the linear case, i. e., a space-time system of equations of the form $A\mathbf{u} = \mathbf{g}$. The algorithm can be described as follows: Given an initial fine approximation $\mathbf{u}^{(0)}$ and the right-hand side $\mathbf{g}^{(0)}$, the first step of the algorithm is to apply a block relaxation to the fine space-time equation system $A^{(0)}\mathbf{u}^{(0)} = \mathbf{g}^{(0)}$. The exact block relaxation is based on two relaxation schemes and combinations of the two. The so-called F-relaxation performs a relaxation of all F-points by propagating the solution from one C-point to all following F-points until the next C-point. The relaxation of each interval of F-points can be performed in parallel and consists of $m-1$ sequential applications of the time integrator. The second scheme, the C-relaxation, analogously performs a relaxation of all C-points by propagating the solution from the preceding F-point to a C-point. Again, all intervals of C-points can be updated simultaneously. Both relaxations are depicted in Figure 4.3. The FCF-relaxation, i. e., an F-relaxation followed by a C- and another F-relaxation, is the typical choice for the MGRIT algorithm, but other choices are also possible. Note that the use of FCF-relaxation is required for multilevel MGRIT to be optimal in the sense that it can obtain convergence for a discrete problem regardless of the spatial or temporal grid spacing and problem size [30]. In the two-level case, F-relaxation is sufficient to obtain this optimality. In the next step, the global residual vector $\mathbf{r}^{(0)}$ is computed and restricted to the coarse grid by injection (R_I). The global coarse grid system $A^{(1)}\mathbf{u}^{(1)} = \mathbf{r}^{(1)}$, where $\mathbf{r}^{(1)}$ is the restricted residual vector, is then solved by the sequential application of a coarse integrator $\Phi^{(1)}$. For the coarse operator, we choose a rediscretization of the problem with step size $\Delta t^{(1)}$, but other choices such as coarsening in space [62, 73, 93] or order of discretization [32, 81] can also be used. Subsequently, the fine approximation is corrected using the "ideal" prolongation P, which is defined as a transpose of an injection followed by an F-relaxation. The attribute "ideal" originates from the theory of algebraic multigrid as presented in [104], since for this choice the Galerkin coarse grid operator is the Schur complement after reordering the system matrix according to the C/F-splitting (see Section 4.3.3). The steps are applied iteratively until a desired solution quality is achieved. Two-level MGRIT is summarized in Algorithm 4.3.


Φ_1 Φ_2 Φ_4 Φ_5 Φ_7 Φ_8
t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9
Φ_3 Φ_6 Φ_9
t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9

F-relaxation.

C-relaxation.

Figure 4.3: Schematic representation of the F- and C-relaxation for a temporal grid with 10 time points and a coarsening factor of three. In both relaxations, the independent intervals (gray) can be updated simultaneously

Algorithm 4.3: Two-level MGRIT

```
repeat
    Apply $F$-relaxation to $A^{(0)}\mathbf{u}^{(0)} = \mathbf{g}^{(0)}$
    Apply $\nu$ times $CF$-relaxation to $A^{(0)}\mathbf{u}^{(0)} = \mathbf{g}^{(0)}$
    Compute residual $\mathbf{r}^{(0)} \leftarrow \mathbf{g}^{(0)} - A\mathbf{u}^{(0)}$
    Restrict residual $\mathbf{r}^{(1)} \leftarrow R_I\mathbf{r}^{(0)}$
    Solve the coarse system $A^{(1)}\mathbf{u}^{(1)} = \mathbf{r}^{(1)}$
    Correct using $\mathbf{u}^{(0)} \leftarrow \mathbf{u}^{(0)} + P\mathbf{u}^{(1)}$
until convergence criterion is reached
```

The relaxation scheme can be controlled by the parameter ν, where at least one F-relaxation is performed per iteration and then ν subsequent CF-relaxations are performed. We follow the typical MGRIT notation here and therefore specify the F-relaxation in line 2. Starting with the second iteration, this F-relaxation can be skipped since the updates are already performed as part of the ideal interpolation of the previous iteration. Note that the two-level variant with F-relaxation is equivalent to the Parareal method [30]. MGRIT yields the same solution as the sequential application of the fine propagator after $N_t^{(0)}/m$ and $N_t^{(0)}/(2m)$ iterations for F-relaxation and FCF-relaxation [30], respectively.

4.3.2 Multilevel FAS MGRIT

In the following, we generalize the linear two-level MGRIT for the multilevel case and use the FAS scheme to solve both linear and nonlinear problems. Analogous to the two-level case, we construct a hierarchy of time grids for L levels based on the coarsening factors $m^{(\ell)}, \ell = 0, ..., L-2$. For uniform coarsening factors, i. e., if the coarsening factor is the same for each level, we use the simplified notation $m = m^{(\ell)}$. Figure 4.4 shows a non-uniform coarsened hierarchy of time grids for $L = 3$ and coarsening factors $m^{(0)} = 3, m^{(1)} = 2$.

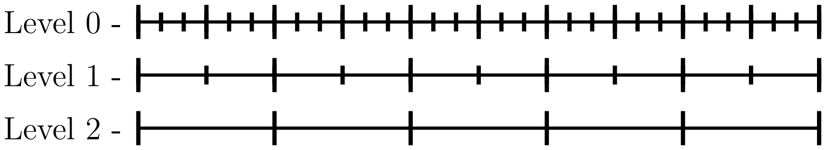
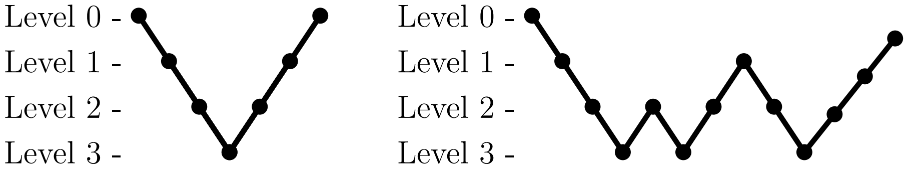


Figure 4.4: Example of a three-level time grid hierarchy with a non-uniform coarsening strategy with coarsening factors $m^{(0)} = 3$ and $m^{(1)} = 2$.

In the following, we assume that all problem-dependent forcing terms are included in the time integrator. Then, the multilevel FAS MGRIT V-cycle algorithm is given in Algorithm 4.4, where $\mathcal{A}^{(\ell)}\mathbf{u}^{(\ell)} = \mathbf{g}^{(\ell)}$ specifies the space-time equation system at levels $\ell = 0, 1, \ldots, L-1$. At all levels except the coarsest, $\ell = 0, ..., L-2$, we use restriction by injection ($R_I^{(\ell)}$), "ideal" interpolation ($P^{(\ell)}$), and $F(CF)^{\nu^{(\ell)}}$-relaxation. The MGRIT variant in Algorithm 4.4 defines a V-cycle, but with minor modifications other multigrid cycles, such as F-cycles (Figure 4.5), can be implemented. The MGRIT-FAS algorithm is based on an initial guess. This initial guess can be chosen arbitrarily; however, a good initial guess provides natural advantages for the convergence of the algorithm. If the solution is already known, it should be chosen as the initial guess. If nothing is known, an improved initial guess can be computed using the nested iteration strategy [64, 65]. The idea of nested iteration is to compute an initial approximation at the coarsest level and interpolate the approximation to the finer levels performing one V-cycle per level, see Figure 4.6.

Algorithm 4.4: FAS MGRIT(ℓ)

```
repeat
    if $\ell$ is the coarsest level then
        Solve coarse-grid system $\mathcal{A}^{(\ell)}(\mathbf{u}^{(\ell)}) = \mathbf{g}^{(\ell)}$
    else
        Apply $F$-relaxation to $\mathcal{A}^{(\ell)}(\mathbf{u}^{(\ell)}) = \mathbf{g}^{(\ell)}$
        Apply $\nu^{(\ell)}$ times $CF$-relaxation to $\mathcal{A}^{(\ell)}(\mathbf{u}^{(\ell)}) = \mathbf{g}^{(\ell)}$
        Inject the approximation and its residual to the coarse grid
            $\mathbf{u}^{(\ell+1)} \leftarrow R_I^{(\ell)}(\mathbf{u}^{(\ell)})$
            $\mathbf{v}^{(\ell+1)} \leftarrow \mathbf{u}^{(\ell+1)}$
            $\mathbf{g}^{(\ell+1)} \leftarrow R_I^{(\ell)}(\mathbf{g}^{(\ell)} - \mathcal{A}^{(\ell)}\mathbf{u}^{(\ell)})$
        if spatial coarsening on level $\ell$ then
            $\mathbf{u}^{(\ell+1)} \leftarrow R_S^{(\ell)}(\mathbf{u}^{(\ell+1)})$
            $\mathbf{v}^{(\ell+1)} \leftarrow \mathbf{u}^{(\ell+1)}$
            $\mathbf{g}^{(\ell+1)} \leftarrow R_S^{(\ell)}(\mathbf{g}^{(\ell+1)})$
        Compute right-hand side $\mathbf{g}^{(\ell+1)} \leftarrow \mathcal{A}^{(\ell+1)}(\mathbf{v}^{(\ell+1)}) + \mathbf{g}^{(\ell+1)}$
        Solve on next level: MGRIT($\ell + 1$)
        Compute the error approximation: $\mathbf{e} \leftarrow \mathbf{u}^{(\ell+1)} - \mathbf{v}^{(\ell+1)}$
        if spatial coarsening on level $\ell$ then
            $\mathbf{e} \leftarrow P_S^{(\ell)}(\mathbf{e})$
        Correct using ideal interpolation: $\mathbf{u}^{(\ell)} \leftarrow \mathbf{u}^{(\ell)} + P^{(\ell)}(\mathbf{e})$
until convergence criterion is reached
```



V-cycle. F-cycle.

Figure 4.5: Structure of V- and F-cycles for four grid levels.

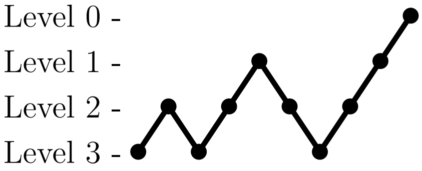


Figure 4.6: Visualization of the nested iteration strategy for a four-level MGRIT algorithm.

In addition, the MGRIT variant in Algorithm 4.4 allows for spatial coarsening between individual levels, where spatial coarsening and spatial interpolation are given by $R_S^{(\ell)}$ and $P_S^{(\ell)}$, respectively, for $\ell = 0, ..., L-2$. Note that spatial coarsening is optional and can be applied after temporal semi coarsening and therefore does not affect the non-intrusive nature of the algorithm. Thus, both operators must be specified manually, but are independent of the time integration scheme. Clearly, the spatial grid transfer operators have a direct impact on the convergence behavior of the algorithm and should therefore be chosen wisely.

In a multilevel setting, multiple coarsening strategies can be chosen depending on the number of available grids in time and space. Both coarsening dimensions are independent of each other and, therefore, can be applied in different ways. Assuming that more temporal grids than spatial grids are available, the most cost efficient approach is to use a *direct* spatial coarsening strategy, starting with spatial coarsening as early as possible. This strategy would reduce the computational costs per iteration for all spatial coarse level problems, but studies [33] have demonstrated that this direct strategy can degrade the convergence behavior of the MGRIT algorithm for linear and nonlinear problems. To overcome the degradation in MGRIT convergence, in [33] a *delayed* spatial coarsening strategy was proposed. This strategy applies spatial coarsening as late as possible, i. e., only on the coarsest time grids. Figure 4.7 illustrates the different coarsening strategies for a five-level V-cycle for the case of three spatial grids, characterized by spatial grid spacings of Δx, $2\Delta x$, and $4\Delta x$, i. e., assuming factor-two coarsening in space. The temporal coarsening factor is chosen to be $m = 4$, i. e., the time step on each temporal grid is given by $4^{\ell-1}\Delta t,\ \ell > 0$. The left V-cycle represents the MGRIT algorithm with coarsening only in the time dimension. In the middle, the direct spatial coarsening strategy is shown, where spatial coarsening is applied on the first and second time levels. Finally, the right V-cycle illustrates the delayed strategy, with spatial coarsening starting on the third level to make use of the coarsest space grid on the coarsest level.

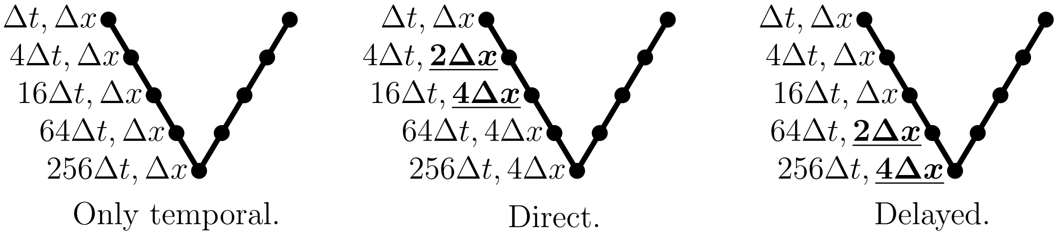


Figure 4.7: Five-level MGRIT V-cycle algorithm with three spatial grids and different space-time coarsening strategies. The left figure shows the MGRIT algorithm with only temporal coarsening, the middle illustrates direct spatial coarsening, and the right uses the delayed spatial coarsening strategy.

4.3.3 Error propagation

In the following, we consider the convergence of the linear two-level MGRIT algorithm (Section 4.3.1), first deriving the error propagator based on [30] and then giving a convergence bound for two-level MGRIT based on [26, 99]. In the analysis we restrict ourselves to the linear two-level MGRIT algorithm with F-relaxation, for an investigation of FCF-relaxation we refer to [26, 99]. An analysis of multilevel MGRIT can be found in [59]. Furthermore, we assume without loss of generality that $N_t^{(0)}$ is evenly divisible by m.

We consider a reordering of the system matrix $A^{(0)}$ so that the entries are arranged that the F-points come first, followed by the C-points. Furthermore, we use the indices c and f to denote the two sets of points. We get

$$\begin{bmatrix} A_{ff}^{(0)} & A_{fc}^{(0)} \\ A_{cf}^{(0)} & A_{cc}^{(0)} \end{bmatrix} \begin{bmatrix} \mathbf{u}_f^{(0)} \\ \mathbf{u}_c^{(0)} \end{bmatrix} = \begin{bmatrix} \mathbf{g}_f^{(0)} \\ \mathbf{g}_c^{(0)} \end{bmatrix}.$$

Based on this structure, the Schur complement decomposition can be formed

$$A^{(0)} = \begin{bmatrix} I_f & \mathbf{0} \\ A_{cf}^{(0)}(A_{ff}^{(0)})^{-1} & I_c \end{bmatrix} \begin{bmatrix} A_{ff}^{(0)} & \mathbf{0} \\ \mathbf{0} & A_{cc}^{(0)} - A_{cf}^{(0)}(A_{ff}^{(0)})^{-1}A_{fc}^{(0)} \end{bmatrix} \begin{bmatrix} I_f & (A_{ff}^{(0)})^{-1}A_{fc}^{(0)} \\ \mathbf{0} & I_c \end{bmatrix},$$

where I_c and I_f represent identity matrices. This representation directly implies the restriction R and the interpolation P, known as the ideal restriction and interpolation, respectively, and of S, given by

$$R = \begin{bmatrix} -A_{cf}^{(0)}(A_{ff}^{(0)})^{-1} & I_c \end{bmatrix}, \quad P = \begin{bmatrix} (-A_{ff}^{(0)})^{-1}A_{fc} \\ I_c \end{bmatrix}, \quad S = \begin{bmatrix} I_f \\ \mathbf{0} \end{bmatrix}.$$

Since $S^T A^{(0)} S = A_{ff}^{(0)}$ and $RA^{(0)}P = A_{cc}^{(0)} - A_{cf}^{(0)}(A_{ff}^{(0)})^{-1}A_{fc}^{(0)}$, we obtain

$$(A^{(0)})^{-1} = P(RA^{(0)}P)^{-1}R + S(S^T A^{(0)} S)^{-1} S^T,$$

and, thus, the error propagator for the exact two-level method is given by

$$0 = I - (A^{(0)})^{-1}A^{(0)} = (I - P(RA^{(0)}P)^{-1}RA^{(0)})(I - S(S^T A^{(0)} S)^{-1} S^T A^{(0)}),$$

where equality holds since $RA^{(0)}S = 0$. The two terms represent the error propagators of the different components, where the first term is the error propagator of the coarse-grid correction and the second part is the error propagator of the F-relaxation. Note that $RA^{(0)}P = R_I A^{(0)} P$, where $R_I = \begin{bmatrix} \mathbf{0} & I_c \end{bmatrix}$ represents the restriction by injection. We use the computationally cheaper R_I in Algorithm 4.4 and in the rest of the convergence study. Note that the second term $(I - S(S^T A^{(0)} S)^{-1} S^T A^{(0)})$ is equivalent to PR_I [30], which we will use in the following. Defining $A^{(1)} = R_I A^{(0)} P$, we can write the error propagator for the exact two-level method as

$$\mathcal{J}_e = (I - P(A^{(1)})^{-1} R_I A^{(0)}) P R_I. \tag{4.11}$$

However, inverting the coarse-grid operator $A^{(1)}$ is roughly as expensive as inverting the original fine-grid problem $A^{(0)}$. Multigrid reduction methods are based on approximating individual components of the exact two-level method (4.11). Here, we replace the coarse-grid operator by an approximation, i. e., $\widetilde{A}^{(1)} \approx A^{(1)}$. Thus, the error propagator for two-level MGRIT with F-relaxation using an approximated coarse-grid problem is given by

$$\mathcal{J}_a = (I - P(\widetilde{A}^{(1)})^{-1} R_I A^{(0)}) P R_I. \tag{4.12}$$

In the next step, we consider the actual matrices instead of the previous abstract construct. For this, we consider again the original order of F- and C-points. The exact coarse-grid problem $A^{(1)}$ and the approximated coarse-grid problem $\widetilde{A}^{(1)}$ are given by

$$A^{(1)} = \begin{bmatrix} I & & & \\ -\Phi^m & I & & \\ & \ddots & \ddots & \\ & & -\Phi^m & I \end{bmatrix}, \ \widetilde{A}^{(1)} = \begin{bmatrix} I & & & \\ -\Psi & I & & \\ & \ddots & \ddots & \\ & & -\Psi & I \end{bmatrix}, \tag{4.13}$$

the restriction by injection R_I is defined as

$$R_I = \begin{bmatrix} I & & & & & & & & \\ & 0 & \cdots & 0 & I & & & & \\ & & & & & \ddots & & & \\ & & & & & & 0 & \cdots & 0 & I \end{bmatrix}, \tag{4.14}$$

and the ideal interpolation as

$$P = \begin{bmatrix} I & & \\ \Phi & & \\ \vdots & & \\ \Phi^{m-1} & & \\ & I & \\ & \Phi & \\ & \vdots & \\ & \Phi^{m-1} & \\ & & \ddots \\ & & & I \end{bmatrix}. \tag{4.15}$$

When comparing the exact and approximated coarse systems (4.13), the approximated system replaces the powers Φ^m corresponding to m applications of the fine operator Φ with a coarse operator Ψ. The operator R_I (4.14) injects the C-points and the operator P defined in (4.15) injects from the coarse to the fine grid and then applies an F-relaxation.

In the following, we substitute the matrices stepwise in the error propagation (4.12). First, we obtain

$$P(\widetilde{A}^{(1)})^{-1}R_I A^{(0)} = \begin{bmatrix} \mathcal{S} & & & & & \\ \widetilde{\mathcal{V}}_0 & \mathcal{S} & & & & \\ \widetilde{\mathcal{V}}_1 & \widetilde{\mathcal{V}}_0 & \mathcal{S} & & & \\ \vdots & \ddots & \ddots & \ddots & & \\ \widetilde{\mathcal{V}}_{N_t^{(1)}-1} & \dots & \widetilde{\mathcal{V}}_1 & \widetilde{\mathcal{V}}_0 & \mathcal{S} & \\ \widetilde{\bar{\mathcal{V}}}_{N_t^{(1)}} & \widetilde{\bar{\mathcal{V}}}_{N_t^{(1)}-1} & \dots & \widetilde{\bar{\mathcal{V}}}_1 & \widetilde{\bar{\mathcal{V}}}_0 & \bar{\mathcal{S}} \end{bmatrix}.$$

with $m \times m$ submatrices

$$\widetilde{\mathcal{V}}_j = \begin{bmatrix} \Phi^0\Psi^{(j+1)} & \mathbf{0} & \dots & \mathbf{0} & -\Phi^0\Psi^j\Phi \\ \Phi^1\Psi^{(j+1)} & \mathbf{0} & \dots & \mathbf{0} & -\Phi^1\Psi^j\Phi \\ \vdots & \vdots & & \vdots & \vdots \\ \Phi^{m-1}\Psi^{(j+1)} & \mathbf{0} & \dots & \mathbf{0} & -\Phi^{m-1}\Psi^j\Phi \end{bmatrix}, \ \mathcal{S} = \begin{bmatrix} I & \mathbf{0} & \dots & \mathbf{0} \\ \Phi & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ \Phi^{m-1} & \mathbf{0} & \dots & \mathbf{0} \end{bmatrix}, \tag{4.16}$$

and $1 \times m$ submatrices

$$\widetilde{\bar{\mathcal{V}}}_j = \begin{bmatrix} \Phi^0\Psi^{(j+1)} & \mathbf{0} & \dots & \mathbf{0} & -\Phi^0\Psi^j\Phi \end{bmatrix}, \qquad \bar{\mathcal{S}} = \begin{bmatrix} I & \mathbf{0} & \dots & \mathbf{0} \end{bmatrix}. \tag{4.17}$$

The $m \times m$ matrices represent an interval of one C-point and $m-1$ F-points, and the $1 \times m$ matrices represent the last C-point. Note that the structure of the matrix representing the single C-point is the same as the row in the $m \times m$ matrices representing a C-point. For the error propagation PR_I of the F-relaxation we obtain

$$PR_I = \begin{bmatrix} \mathcal{S} & & & \\ & \ddots & & \\ & & \mathcal{S} & \\ & & & \bar{\mathcal{S}} \end{bmatrix}, \tag{4.18}$$

with $\mathcal{S}$ and $\bar{\mathcal{S}}$ from (4.16) and (4.17), respectively. Finally, the error propagator (4.12) is given by

$$\mathcal{J}_a = \begin{bmatrix} \mathbf{0} & \dots & & & & \mathbf{0} \\ \mathcal{Z}_0 & \mathbf{0} & \dots & & & \mathbf{0} \\ \mathcal{Z}_1 & \mathcal{Z}_0 & \mathbf{0} & \dots & & \mathbf{0} \\ \vdots & \ddots & \ddots & \ddots & & \vdots \\ \mathcal{Z}_{N_t^{(1)}-2} & \dots & \mathcal{Z}_1 & \mathcal{Z}_0 & \mathbf{0} & \mathbf{0} \\ \bar{\mathcal{Z}}_{N_t^{(1)}-1} & \bar{\mathcal{Z}}_{N_t^{(1)}-2} & \dots & \bar{\mathcal{Z}}_1 & \bar{\mathcal{Z}}_0 & \mathbf{0} \end{bmatrix}, \tag{4.19}$$

with

$$\mathcal{Z}_j = \begin{bmatrix} \Phi^0\Psi^j(\Phi^m - \Psi) & \mathbf{0} & \dots & \mathbf{0} \\ \Phi^1\Psi^j(\Phi^m - \Psi) & \mathbf{0} & \dots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots \\ \Phi^{m-1}\Psi^j(\Phi^m - \Psi) & \mathbf{0} & \dots & \mathbf{0} \end{bmatrix}, \tag{4.20}$$

and

$$\bar{\mathcal{Z}}_j = \begin{bmatrix} \Phi^0\Psi^j(\Phi^m - \Psi) & \mathbf{0} & \dots & \mathbf{0} \end{bmatrix}. \tag{4.21}$$

4.3.4 Convergence bounds

To avoid multiple subscripts, we omit the subscript a of $\mathcal{J}_a$ from (4.19) in the sequel. Using f and c subscripts to denote F- and C-points, respectively, $\mathcal{J}$ from (4.19) can be reordered and partitioned into 2×2 block form. Moreover, notice that F-points columns of $\mathcal{Z}_j$ from (4.20) and $\bar{\mathcal{Z}}_j$ from (4.21) (and therefore also of $\mathcal{J}_a$ from (4.19)) are all zero. If we then consider powers of the matrix, which correspond to several iterations, we get

$$\mathcal{J}^\ell := \begin{bmatrix} \mathcal{J}_{ff} & \mathcal{J}_{fc} \\ \mathcal{J}_{cf} & \mathcal{J}_{cc} \end{bmatrix}^\ell = \begin{bmatrix} \mathbf{0} & \mathcal{J}_{fc} \\ \mathbf{0} & \mathcal{J}_{cc} \end{bmatrix}^\ell = \begin{bmatrix} \mathbf{0} & \mathcal{J}_{fc}\mathcal{J}_{cc}^{\ell-1} \\ \mathbf{0} & \mathcal{J}_{cc}^\ell \end{bmatrix}.$$

It follows from above that for multiple iterations, convergence is fully determined by $\mathcal{J}_{cc} \in \mathbb{R}^{N_t^{(1)}+1 \times N_t^{(1)}+1}$, that is, $\mathcal{J}^\ell$ will be convergent in some norm, that is, $\|\mathcal{J}^\ell\| < 1$, if and only if $\mathcal{J}_{cc}^\ell$ is as well. To that end, we consider analyzing the C-C principle submatrix of (4.19),

$$\mathcal{J}_{cc} = (\Phi^m - \Psi) \begin{bmatrix} \mathbf{0} & \dots & & & \mathbf{0} \\ 1 & \mathbf{0} & \dots & & \mathbf{0} \\ \Psi^1 & 1 & \mathbf{0} & \dots & \mathbf{0} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \Psi^{N_t^{(1)}-1} & \dots & \Psi^1 & 1 & \mathbf{0} \end{bmatrix}.$$

Now consider the case of Φ and Ψ being simultaneously diagonalizable, as would occur if the same (diagonalizable) spatial matrix is used on the fine and coarse grid. Note that this property is equivalent to assuming that Φ and Ψ commute and are both diagonalizable. Commutativity of both operators holds for almost all standard time integration methods, including all one-step Runge-Kutta methods, when the same method is used at the fine and coarse levels [58, Section 4.3]. Diagonalizability holds for most parabolic PDEs [36, 99]. Let U denote the shared eigenvector matrix of Φ and Ψ, with eigenvalues $\mu \in \sigma(\Psi)$ and $\lambda \in \sigma(\Phi)$, where $\sigma(\Psi)$ and $\sigma(\Phi)$ denote the spectrum of Ψ and Φ, respectively. Following the frameworks developed in [26, 99], let $\widetilde{U}$ denote a block-diagonal matrix, with diagonal blocks given by eigenvectors U. Then,

$$\|\mathcal{J}_{cc}\|_{(\widetilde{U}\widetilde{U}^*)^{-1}} = \max_{\{\mu,\lambda\}} \|\widetilde{\mathcal{J}}_{cc}\|,$$

where $\|\cdot\|$ corresponds to the ℓ^2-norm, and $\widetilde{\mathcal{J}}_{cc}$ is defined as follows for a fixed pair of eigenvalues $\{\mu, \lambda\}$:

$$\widetilde{\mathcal{J}}_{cc} := (\lambda^m - \mu) \begin{bmatrix} \mathbf{0} & \dots & & & \mathbf{0} \\ 1 & \mathbf{0} & \dots & & \mathbf{0} \\ \mu^1 & 1 & \mathbf{0} & \dots & \mathbf{0} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \mu^{N_t^{(1)}-1} & \dots & \mu^1 & 1 & \mathbf{0} \end{bmatrix}.$$

If the spatial matrix is normal, then $(\widetilde{U}\widetilde{U}^*)^{-1} = I$. In general, bounding $\widetilde{\mathcal{J}}_{cc}$ in the ℓ^2-norm for each eigenvalue pair guarantees convergence of $\mathcal{J}_{cc}$ in a certain eigenvector induced norm, where “convergence” corresponds to a guaranteed reduction in error every iteration (in contrast to, e. g., nilpotency, where convergence is eventually guaranteed, but error could in principle diverge significantly for many iterations before sudden convergence to the exact solution).

Recall the inequality $\|\widetilde{\mathcal{J}}_{cc}\|^2 \leq \|\widetilde{\mathcal{J}}_{cc}\|_1 \|\widetilde{\mathcal{J}}_{cc}\|_\infty$. Given that $\widetilde{\mathcal{J}}_{cc}$ is Toeplitz, the maximum row and column sums are equal, yielding the bound for two-level MGRIT with F-relaxation

$$\begin{aligned} \|\widetilde{\mathcal{J}}_{cc}\| \leq \|\widetilde{\mathcal{J}}_{cc}\|_1 &= |\lambda^m - \mu| \sum_{\ell=0}^{N_t^{(1)}-1} |\mu^\ell| \\ &= \frac{|\lambda^m - \mu|(1 - |\mu|^{N_t^{(1)}})}{1 - |\mu|}. \end{aligned} \tag{4.22}$$

4.4 AT-MGRIT

The AT-MGRIT algorithm is a variant of the MGRIT algorithm in which the global time grid at the coarsest level is replaced by several local grids, each of which contains only temporal subdomains. This idea was originally motivated by similar processor-local multigrid hierarchies used in geometric and algebraic multigrid for elliptic problems [7, 77, 78]. The structure of the section follows that of the previous section.

4.4.1 Two-level AT-MGRIT

For the two-level case, we again assume that the forcing term is no longer in the time integrator. Equivalent to the two-level MGRIT algorithm (see Section 4.3.1), we form a global coarse grid $\mathcal{T}^{(1)} = \{n\Delta t^{(1)} : n = 0, 1, \dots, N_t^{(1)}\}$ based on a fine temporal grid $\mathcal{T}^{(0)} = \{n\Delta t^{(0)} : n = 0, 1, \dots, N_t^{(0)}\}$, a coarsening factor m, and a partition into F- and C-points. Based on this global coarse grid, we define $N_t^{(1)} + 1$ overlapping *local* coarse grids. For a given local grid size w, the pth local coarse grid, $\mathcal{T}^{(1,p)}$ for $p = 0, ..., N_t^{(1)}$, is given by

$$\mathcal{T}^{(1,p)} = \left\{n\Delta t^{(0)} : n \in [\max(0, p - w + 1), p]\right\},$$

with time step size $\Delta t^{(1)} = m\Delta t^{(0)}$, as depicted in Figure 4.8.

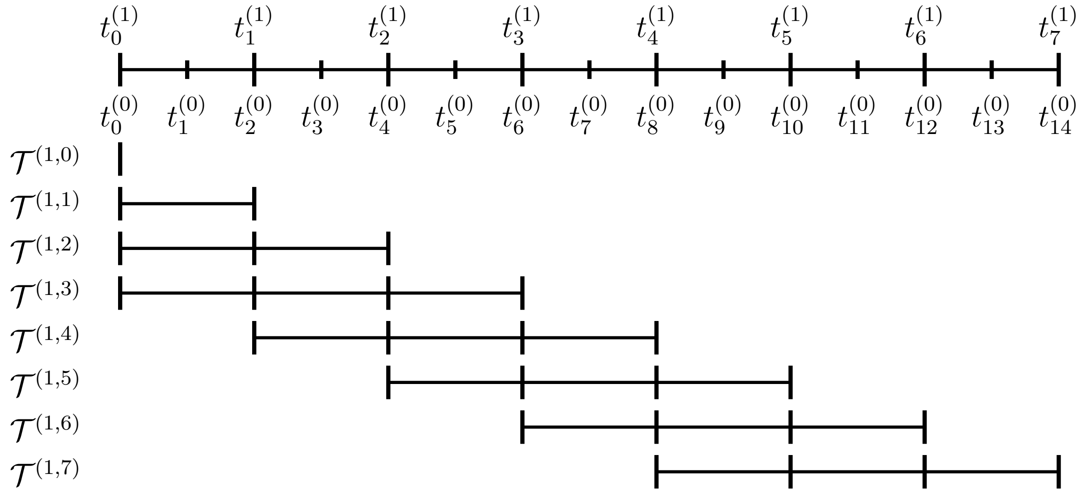

$t_0^{(1)}$ $t_1^{(1)}$ $t_2^{(1)}$ $t_3^{(1)}$ $t_4^{(1)}$ $t_5^{(1)}$ $t_6^{(1)}$ $t_7^{(1)}$
$t_0^{(0)}$ $t_1^{(0)}$ $t_2^{(0)}$ $t_3^{(0)}$ $t_4^{(0)}$ $t_5^{(0)}$ $t_6^{(0)}$ $t_7^{(0)}$ $t_8^{(0)}$ $t_9^{(0)}$ $t_{10}^{(0)}$ $t_{11}^{(0)}$ $t_{12}^{(0)}$ $t_{13}^{(0)}$ $t_{14}^{(0)}$
$\mathcal{T}^{(1,0)}$
$\mathcal{T}^{(1,1)}$
$\mathcal{T}^{(1,2)}$
$\mathcal{T}^{(1,3)}$
$\mathcal{T}^{(1,4)}$
$\mathcal{T}^{(1,5)}$
$\mathcal{T}^{(1,6)}$
$\mathcal{T}^{(1,7)}$

Figure 4.8: Two-level temporal grid-hierarchy example for the AT-MGRIT algorithm with $N_t^{(0)} = 14$, $m = 2$ and $w = 4$. The C-points (long markers) define the global coarse grid. For each point $p = 0, \dots, 7$ on the global coarse grid, a local coarse grid $\mathcal{T}^{(1,p)}$ is created.

Then, the AT-MGRIT algorithm works as follow: Given an initial approximation $\mathbf{u}^{(0)}$ and the right-hand side $\mathbf{g}^{(0)}$, a block relaxation consisting of a combination of F- and C-relaxation is applied to the fine problem. In the next step, the global residual vector $\mathbf{r}^{(0)}$ is computed and restricted to all local coarse grids by injecton using operators $R_I^{(p)}$. For each local coarse grid, the coarse system $A^{(1,p)}\mathbf{u}^{(1,p)} = \mathbf{r}^{(1,p)}$ is solved, which consists of $w - 1$ sequential applications of the coarse time integrator. Since the coarse grid problems are independent of each other, they can be solved simultaneously. Then, the global solution vector is corrected by *selective ideal* interpolation $P_S^{(p)}$. Selective ideal interpolation is the transpose of an injection followed by an F-relaxation starting from exactly one time point. More precisely, the approximation of the solution at the last time point of each local coarse grid is interpolated to the fine grid, and then an F-relaxation is performed using these interpolated points. Again, these steps are performed iteratively until the desired quality of the solution is achieved. The two-level AT-MGRIT algorithm is summarized in Algorithm 4.5.

Algorithm 4.5: Two-level AT-MGRIT

```
repeat
    Apply $F$-relaxation to $A^{(0)}\mathbf{u}^{(0)} = \mathbf{g}^{(0)}$
    Apply $\nu$ times $CF$-relaxation to $A^{(0)}\mathbf{u}^{(0)} = \mathbf{g}^{(0)}$
    Compute residual $\mathbf{r}^{(0)} \leftarrow \mathbf{g}^{(0)} - A\mathbf{u}^{(0)}$
    for $p = 0$ to $N_t^{(1)}$ do
        Restrict residual, $\mathbf{r}^{(1,p)} \leftarrow R_I^{(p)}\mathbf{r}^{(0)}$
        Solve local system $A^{(1,p)}\mathbf{u}^{(1,p)} = \mathbf{r}^{(1,p)}$
        Correct using $\mathbf{u}^{(0)} \leftarrow \mathbf{u}^{(0)} + P_S^{(p)}\mathbf{u}^{(1,p)}$
until convergence criterion is reached
```

Note that the AT-MGRIT algorithm solves for the exact solution in $N_t^{(1)}$ iterations if $w > 1$. Furthermore, the algorithm is equivalent to the two-level MGRIT method if $w = N_t^{(0)} + 1$, i. e., if all local coarse grids contain all C-points before in time. All components of the AT-MGRIT algorithm are highly parallel. The only communication needed is for the residual computation and the distribution of the residual (performed by the matrix-vector product $\mathbf{r}^{(1,p)} = R_I^{(p)}\mathbf{r}^{(0)}$ in Algorithm 4.5). Moreover, the coarse-level solve is communication-free (except for any communication that arises in spatial parallelism). This is particularly relevant for emerging heterogeneous computing architectures, where communication to and from GPU nodes can be quite expensive, and high efficiency is obtained with a low communication to computation ratio.

4.4.2 Multilevel FAS AT-MGRIT

Analogously to MGRIT, we can easily generalize the two-level algorithm to the multilevel case and use the FAS scheme for solving nonlinear problems. We first construct a multilevel hierarchy of temporal grids recursively using a uniform or non-uniform coarsening strategy. AT-MGRIT uses the same levels, coarsening, relaxation, and transfer operators as MGRIT on all finer levels in the hierarchy, but on the coarsest level the global grid is replaced by multiple local grids. Figure 4.9 shows an example grid hierarchy for three-level AT-MGRIT with $N_t^{(0)} = 20$, $m = 2$, and $w = 4$. While MGRIT utilizes the global coarse grid on level 2, AT-MGRIT uses local grids $\mathcal{T}^{(2,p)}, p = 0, \ldots, 5$.

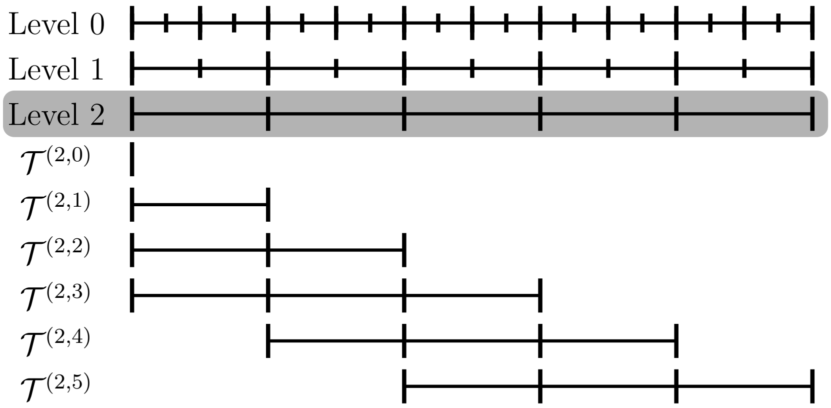


Figure 4.9: Example of a three-level time grid hierarchy for the AT-MGRIT algorithm for a fine grid with 21 time points, $m = 2$ and $w = 4$. At the coarsest level, a local coarse grid is generated for each C-point of the global coarse grid (gray box). These local grids ($\mathcal{T}^{(2,p)}, p = 0, \dots, 5$) replace the global coarse grid used in the classical MGRIT algorithm.

In the following, we assume again that all problem-dependent forcing terms are included in the time integrator. Then, the multilevel FAS AT-MGRIT V-cycle algorithm is given in Algorithm 4.6, where $N_t^{(\ell)}$ denotes the number of time points, and $\mathcal{A}^{(\ell)}\mathbf{u}^{(\ell)} = \mathbf{g}^{(\ell)}$ and $\mathcal{A}^{(\ell,p)}\mathbf{u}^{(\ell,p)} = \mathbf{g}^{(\ell,p)}$ specifies the space-time system of equations on levels $\ell = 0, 1, \dots, L-1$ and on the local coarse grids $p = 0, 1, \dots, N_t^{(\ell)}$, respectively. Further, we use restriction by injection ($R_I^{(\ell)}$), "ideal" interpolation ($P^{(\ell)}$), and $F(CF)^{\nu^{(\ell)}}$-relaxation on levels $\ell = 0, ..., L-2$. At the coarsest level, restriction and interpolation to and from the local coarse grids is done by injection, denoted by $R_I^{(\ell,p)}$ and $P_I^{(\ell,p)}$, respectively. Note that the residual is first transferred to the global coarse grid of the coarsest level and then to the local coarse grids, which allows for a simpler notation of the algorithm. AT-MGRIT can also be used with other common multigrid cycle types, such as F-cycles or nested iterations. For all cycle types, the standard MGRIT coarsest level can be replaced by local coarse grids, but hybrid versions are also possible, where either the global problem or the local problems are solved at the coarsest level. Independent of the initial guess, the AT-MGRIT algorithm solves for the exact discrete solution after $N_t^{(0)}/(2m)$ iterations for FCF-relaxation if $w > 1$. AT-MGRIT is equivalent to MGRIT if $w = N_t^{(L-1)} + 1$.

4.4.3 Error propagation

This section presents the convergence theory for AT-MGRIT in the linear two-level setting, which we developed in [55]. The analysis builds on the two-level

Algorithm 4.6: FAS AT-MGRIT(ℓ)

```
repeat
    if $\ell$ is the coarsest level then
        for $i = 0$ to $N_t^{(\ell)}$ do
            Restrict to local grids
                $\mathbf{u}^{(\ell,p)} \leftarrow R_I^{(\ell,p)}(\mathbf{u}^{(\ell)})$
                $\mathbf{v}^{(\ell,p)} \leftarrow \mathbf{u}^{(\ell,p)}$
                $\mathbf{g}^{(\ell,p)} \leftarrow R_I^{(\ell,p)}(\mathbf{g}^{(\ell)})$
            Solve local problem $\mathcal{A}^{(l,p)}(\mathbf{u}^{(\ell,p)}) = \mathcal{A}^{(\ell,p)}(\mathbf{v}^{(\ell,p)}) + \mathbf{g}^{(\ell,p)}$
            Update $\mathbf{u}^{(\ell)} \leftarrow P_I^{(\ell,i)}\mathbf{u}^{(\ell,i)}$
    else
        Apply $F$-relaxation to $\mathcal{A}^{(\ell)}(\mathbf{u}^{(\ell)}) = \mathbf{g}^{(\ell)}$
        Apply $\nu^{(\ell)}$ times $CF$-relaxation to $\mathcal{A}^{(\ell)}(\mathbf{u}^{(\ell)}) = \mathbf{g}^{(\ell)}$
        Inject the approximation and its residual to the coarse grid
            $\mathbf{u}^{(\ell+1)} \leftarrow R_I^{(\ell)}(\mathbf{u}^{(\ell)})$
            $\mathbf{v}^{(\ell+1)} \leftarrow \mathbf{u}^{(\ell+1)}$
            $\mathbf{g}^{(\ell+1)} \leftarrow R_I^{(\ell)}(\mathbf{g}^{(\ell)} - \mathcal{A}^{(\ell)}\mathbf{u}^{(\ell)})$
        Compute right-hand side $\mathbf{g}^{(\ell+1)} \leftarrow \mathcal{A}^{(\ell+1)}(\mathbf{v}^{(\ell+1)}) + \mathbf{g}^{(\ell+1)}$
        Solve on next level: MGRIT($\ell + 1$)
        Compute the error approximation: $\mathbf{e} \leftarrow \mathbf{u}^{(\ell+1)} - \mathbf{v}^{(\ell+1)}$
        Correct using ideal interpolation: $\mathbf{u}^{(\ell)} \leftarrow \mathbf{u}^{(\ell)} + P^{(\ell)}(\mathbf{e})$
until convergence criterion is reached
```

theory from Section 4.3.3, and provides insight into the effects of truncating the coarse time grid. We begin by introducing the error propagation operator for exact solves on the truncated coarse grids and then show results for inexact coarse grid solves. Afterwards, formal two-level convergence bounds are provided.

Following (4.11), the two-level error propagation operator for linear AT-MGRIT with an exact coarse-grid solve is given by

$$\mathcal{E} := \left(I - \sum_{p=0}^{N_t^{(1)}} P_S^{(p)} (A^{(1,p)})^{-1} R_I^{(p)} A^{(0)} \right) P R_I, \tag{4.23}$$

where $A^{(1,p)}$ represents the local coarse grid systems, $R_I^{(p)}$ is the restriction operator to the local coarse grids, and $P_S^{(p)}$ defines the interpolation from the local coarse grids that updates the fine grid using *selective ideal interpolation*, i. e., for one specific C-point, this C-point and the following interval of F-points are updated. We see that (4.23) is analogous to (4.11), but here we must sum over all C-points, as each C-point is updated by a unique local coarse-grid.

The operators P and R_I, corresponding to ideal interpolation and restriction by injection, are the same operators as described in (4.14) and (4.15), respectively. The selective ideal interpolation from the local coarse grid to the fine grid is given by

$$P_S^{(p)} := \left[\begin{array}{cc} \overbrace{\qquad\qquad}^{\min(p,w-1)} & \\ & \\ & \\ & I \\ & \Phi \\ & \vdots \\ & \Phi^{m-1} \\ & \\ & \end{array}\right] \begin{array}{l} \left.\vphantom{\begin{array}{c}a\\a\\a\end{array}}\right\}{\scriptstyle pm} \\ \left.\vphantom{\begin{array}{c}I\\\Phi\\\vdots\\\Phi^{m-1}\end{array}}\right\}{\scriptstyle m} \quad . \\ \left.\vphantom{\begin{array}{c}a\\a\end{array}}\right\}{\scriptstyle N_t^{(0)}+1-(p+1)m} \end{array}$$

Note that the exact dimension and structure of the operator depends on the considered local coarse grid. Recall that the fine-grid operator has block dimension $(N_t^{(0)}+1) \times (N_t^{(0)}+1)$, with each block being a square operator the size of Φ. Letting $N_t^{(0)} = mN_t^{(1)}$ for coarse-grid points $0, ..., N_t^{(1)}$, the fine-grid size can be written as $(mN_t^{(1)}+1) \times (mN_t^{(1)}+1)$, which we will use to express error propagation largely in terms of $m \times m$ coarse blocks. Each of these blocks represents a block of one C-point and $m-1$ following F-points. At the end, there is a single block containing only one C-point. Note that the structure for this block is always a submatrix of the $m \times m$ blocks, containing only the part corresponding to the C-point.

Exact local coarse grid solve

First, we consider the effect of the local coarse grids using exact solves on the coarse time steps. For this purpose, we define the local coarse-grid problem as

$$A^{(1,p)} := R_I^{(p)} A^{(0)} P^{(p)},$$

where $P^{(p)}$ and $R_I^{(p)}$ define the transfer between the fine grid and the local coarse grids and are submatrices of P and R_I. For $P^{(p)}$, only columns of P associated to points lying on this local coarse grid are considered. Equivalently, only the associated rows are considered for the restriction. Then, the coarse-grid problems are given by

$$A^{(1,p)} = \left.\overbrace{\begin{bmatrix} I & & & & \\ -\Phi^m & I & & & \\ & -\Phi^m & I & & \\ & & \ddots & \ddots & \\ & & & -\Phi^m & I \end{bmatrix}}^{\min(p+1,w)}\right\}{\scriptstyle \min(p+1,w)} . \tag{4.24}$$

Here, it is important to note that all local coarse-grid systems $A^{(1,p)}$ have the same structure, but consider different time intervals. In fact, the exact local coarse-grid systems are principal submatrices of the Schur complement corresponding to a standard MGRIT coarse-grid with exact solves (see (4.13)).

We consider the error propagation $\mathcal{E}_e$ for one C-point $p = 0, \ldots, N_t^{(1)}$ using the ideal local coarse-grid problem (i. e., exact coarse grid and inverses). The structure of the matrices for the first w C-points differs from all other C-points, since the local coarse grids corresponding to the first w C-points contain all C-points prior in time. Here, we want to study the effect of local coarse grids that do not extend back to $t = 0$. Therefore, we start by considering all local coarse grids $N_t^{(1)} > p \geq w$ and subsequently discuss the structure for $p < w$. Note that the structure of the matrices of $p = N_t^{(1)}$ is always a submatrix of $N_t^{(1)} > p \geq w$ and therefore is not explicitly stated. For $N_t^{(1)} > p \geq w$ the matrix $R_I^{(p)} A^{(0)}$ is given by

$$\left.\left[\begin{array}{c:c:c:c:c:c} \multicolumn{1}{c}{\overbrace{\qquad}^{(p-w)m}} & \multicolumn{1}{c}{\overbrace{\qquad\qquad}^{m}} & \multicolumn{3}{c}{\overbrace{\qquad\qquad\qquad\qquad\qquad\qquad}^{wm}} & \multicolumn{1}{c}{\overbrace{\qquad}^{N_t^{(0)}+1-(p+1)m}} \\ & \mathbf{0}_{1\times(m-1)} \ -\Phi & I \ \ \mathbf{0}_{1\times(m-1)} & & & \\ \hdashline & & \ddots & \ddots & & \\ \hdashline & & & \mathbf{0}_{1\times(m-1)} \ -\Phi & I \ \ \mathbf{0}_{1\times(m-1)} & \end{array}\right]\right\}{\scriptstyle w} , \tag{4.25}$$

which initially contains $(p - w)m + m$ columns corresponding to the omitted points on the local coarse grid. The following wm columns correspond to the C-points present on the local coarse grid and their corresponding following interval of F-points. Next, we consider

$$P_S^{(p)}(A^{(1,p)})^{-1} = \left[\begin{array}{cccc} \multicolumn{4}{c}{\overbrace{\qquad\qquad\qquad\qquad\qquad\qquad}^{w}} \\ & & & \\ \hdashline \Phi^{(w-1)m} \ \cdots & \Phi^{2m} & \Phi^{m} & I \\ \Phi^{(w-1)m+1} \cdots & \Phi^{2m+1} & \Phi^{m+1} & \Phi \\ \vdots & \vdots & \vdots & \vdots \\ \Phi^{wm-1} \ \cdots & \Phi^{3m-1} & \Phi^{2m-1} & \Phi^{m-1} \\ \hdashline & & & \end{array}\right] \begin{array}{l} \}\ pm \\ \}\ m\ , \\ \}\ N_t^{(0)}+1-(p+1)m \end{array}$$

with $A^{(1,p)}$ as in (4.24), which defines the effect of selective ideal interpolation multiplied by the inverse of the coarse-grid problem. Due to the selective ideal interpolation operator, exactly m points are considered, namely the C-point to be updated and the following F-interval consisting of $m-1$ points. All other points are not changed by the update of one p and the corresponding rows are therefore zero. As a consequence, the product $P_S^{(p)}(A^{(1,p)})^{-1}R_I^{(p)}A^{(0)}$ also has only m nonzero rows. Furthermore, we have exactly $w+1$ blocks of $m \times m$ matrices which are not equal to zero. The matrix $P_S^{(p)}(A^{(1,p)})^{-1}R_I^{(p)}A^{(0)}$ in block form is given by

$$\begin{array}{c} \begin{array}{ccccc} \overbrace{\qquad\quad}^{(p-w)m} & \overbrace{\qquad\quad}^{m} & \overbrace{\qquad\qquad\qquad\qquad\qquad}^{(w-1)m} & \overbrace{\qquad\quad}^{m} & \overbrace{\qquad\quad}^{N_t^{(0)}+1-(p+1)m} \end{array} \\ \left[\begin{array}{c:c:c:c:c:c:c} & & & & & & \\ \hdashline & \mathcal{D} & \mathcal{V}_{w-2} & \cdots & \mathcal{V}_0 & \mathcal{S} & \\ \hdashline & & & & & & \end{array}\right] \begin{array}{l} \}\ pm \\ \}\ m\ , \\ \}\ N_t^{(0)}+1-(p+1)m \end{array} \end{array} \tag{4.26}$$

with block $m \times m$ inner matrices

$$\mathcal{V}_j = \begin{bmatrix} \Phi^{(j+1)m} & \mathbf{0} & \cdots & \mathbf{0} & -\Phi^{jm+1} \\ \Phi^{(j+1)m+1} & \mathbf{0} & \cdots & \mathbf{0} & -\Phi^{jm+2} \\ \vdots & \vdots & & \vdots & \vdots \\ \Phi^{(j+2)m-1} & \mathbf{0} & \cdots & \mathbf{0} & -\Phi^{(j+1)m} \end{bmatrix}, \quad \mathcal{D} = \begin{bmatrix} \mathbf{0} & \cdots & \mathbf{0} & -\Phi^{(w-1)m+1} \\ \mathbf{0} & \cdots & \mathbf{0} & -\Phi^{(w-1)m+2} \\ \vdots & & \vdots & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & -\Phi^{wm} \end{bmatrix},$$

and $\mathcal{S}$ as given in (4.16). Here, $\mathcal{D}$ comes from the truncated coarse-grid points, $\mathcal{V}_{w-2}, \ldots, \mathcal{V}_0$ represent the first $w-1$ local coarse-grid points, and $\mathcal{S}$ corresponds to the last point of the local coarse grid. Note that the $\mathcal{S}$-block is a diagonal block of the matrix. The operator PR_I, which is equivalent to an F-relaxation, is given by (4.18).

We can now calculate the error propagation $(I-\sum_{p=0}^{N_t^{(1)}} P_S^{(p)}(A^{(1,p)})^{-1}R_I^{(p)}A^{(0)})PR_I$ by exploiting the structure of the matrices $P_S^{(p)}(A^{(1,p)})^{-1}R_I^{(p)}A$ and PR_I. Instead of computing the complete matrix, we can compute the blocks $-\mathcal{D}\mathcal{S}, -\mathcal{V}_j\mathcal{S}$ for $j = w-2,\dots,0$, and $(I-\mathcal{S})\mathcal{S}$. Note that the identity term is added to $-\mathcal{S}$ because $\mathcal{S}$ is the diagonal block of $P_S^{(p)}(A^{(1,p)})^{-1}R_I^{(p)}A^{(0)}$. Working through the algebra yields $-\mathcal{V}_j\mathcal{S} = \mathbf{0}$ for $j = w-2,\dots,0$, $(I-\mathcal{S})\mathcal{S} = \mathcal{S}^2 - \mathcal{S} = \mathbf{0}$, and the block $m \times m$ matrix

$$-\mathcal{D}\mathcal{S} = \begin{bmatrix} \Phi^{wm} & \mathbf{0} & \cdots & \mathbf{0} \\ \Phi^{wm+1} & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots \\ \Phi^{wm+m-1} & \mathbf{0} & \cdots & \mathbf{0} \end{bmatrix}.$$

Note that for the case $p < w$ in matrix (4.26) the operator $\mathcal{D}$ is omitted, since for these C-points all previous C-points are contained in the local coarse grid.

We can now examine the error propagator $\mathcal{E}_e$ using exact solves on the local coarse grids. In forming $\mathcal{E}_e$ by summing over $p = 0,\dots,N_t^{(1)}$, we obtain a block lower triangular matrix, whereby each p updates m rows of $\mathcal{E}_e$, and the error propagator using ideal local coarse grids can be written in block form as

$$\mathcal{E}_e = \begin{array}{l}
\quad\overbrace{\qquad}^{m}\overbrace{\qquad\qquad\qquad\qquad}^{m(N_t^{(1)}-2)}\overbrace{\qquad}^{m+1} \\
\left[\begin{array}{ccccccc}
\mathbf{0} & \cdots & & & & & \mathbf{0}\\
\vdots & \cdots & & & & & \vdots\\
\mathbf{0} & \cdots & & & & & \mathbf{0}\\
\mathcal{C} & \mathbf{0} & \cdots & & & & \mathbf{0}\\
\mathbf{0} & \ddots & \ddots & & & & \vdots\\
\mathbf{0} & \mathbf{0} & \mathcal{C} & \mathbf{0} & \cdots & & \mathbf{0}\\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \bar{\mathcal{C}} & \mathbf{0} & \cdots & \mathbf{0}
\end{array}\right]
\begin{array}{l}
\}\; m \\
\left.\begin{array}{l} \\ \\ \end{array}\right\} (k-1)m \\
\left.\begin{array}{l} \\ \\ \\ \end{array}\right\} (N_t^{(1)}-k)m \\
\}\; 1
\end{array}
\end{array},$$

where $\mathcal{C} = -\mathcal{D}\mathcal{S}$ and $\bar{\mathcal{C}}$ is identical to the first row of $\mathcal{C}$ and represents the additional block consisting of one C-point. Note that the error propagator $\mathcal{E}_e$ is nonzero, so unlike two-level MGRIT, AT-MGRIT using exact local coarse-grid inverses is not a direct method. For all $p > w$, we have some error perturbation that results from truncating the exact (Schur-complement) coarse grid.

Approximate local coarse grid solve

Again, we do not invert $R_I^{(p)}A^{(0)}P^{(p)}$ exactly, but approximate $R_I^{(p)}A^{(0)}P^{(p)} \approx \widetilde{A}^{(1,p)}$. Specifically, we approximate again the powers Φ^m, which correspond to m

applications of the fine time integrator Φ, with a coarse operator Ψ. This results in the approximation $\widetilde{A}^{(1,p)}$ given by

$$\widetilde{A_c}^{(1,p)} := \overbrace{\begin{bmatrix} I & & & \\ -\Psi & I & & \\ & \ddots & \ddots & \\ & & -\Psi & I \end{bmatrix}}^{\min(p+1,w)} \left.\vphantom{\begin{bmatrix} I \\ I \\ \ddots \\ I \end{bmatrix}}\right\} {\scriptstyle \min(p+1,w)} .$$

Using this approximation, we can formulate the error propagation $\mathcal{E}_a$ using the approximated local coarse-grid inverse.

The definition of $R_I^{(p)} A^{(0)}$ is the same as in (4.25), but now

$$P_S^{(p)} \widetilde{A}^{(1,p)} = \left[\begin{array}{cccc} \multicolumn{4}{c}{\overbrace{\hphantom{\Phi^{m-1}\Psi^{w-1} \cdots \Phi^{m-1}\Psi^2\ \Phi^{m-1}\Psi\ \Phi^{m-1}}}^{w}} \\ \hdashline \Phi^0\Psi^{w-1} & \cdots\ \Phi^0\Psi^2 & \Phi^0\Psi & \Phi^0 \\ \vdots & \vdots & \vdots & \vdots \\ \Phi^{m-1}\Psi^{w-1} & \cdots\ \Phi^{m-1}\Psi^2 & \Phi^{m-1}\Psi & \Phi^{m-1} \\ \hdashline & & & \end{array}\right] \begin{array}{l} \}\ {\scriptstyle pm} \\ \left.\vphantom{\begin{array}{c} \Phi \\ \vdots \\ \Phi \end{array}}\right\} {\scriptstyle m} \\ \}\ {\scriptstyle N_t^{(0)}+1-(p+1)m} \end{array} .$$

As a result, we get a block matrix equivalent to (4.26), but this time with $m \times m$ block matrices $\widetilde{\mathcal{V}}_j$ and $\widetilde{\mathcal{D}}$ given by

$$\widetilde{\mathcal{V}}_j = \begin{bmatrix} \Phi^0\Psi^{(j+1)} & \mathbf{0} & \cdots & \mathbf{0} & -\Phi^0\Psi^j\Phi \\ \vdots & \vdots & & \vdots & \vdots \\ \Phi^{m-1}\Psi^{(j+1)} & \mathbf{0} & \cdots & \mathbf{0} & -\Phi^{m-1}\Psi^j\Phi \end{bmatrix}, \widetilde{\mathcal{D}} = \begin{bmatrix} \mathbf{0} & \cdots & \mathbf{0} & -\Phi^0\Psi^{w-1}\Phi \\ \vdots & & \vdots & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & -\Phi^{m-1}\Psi^{w-1}\Phi \end{bmatrix},$$

instead of $\mathcal{V}$ and $\mathcal{D}$, respectively. Note that $\mathcal{S}$ (as well as $\widetilde{\mathcal{V}}_j$) is the same as given in (4.16). Again, we use the structure of the matrices and calculate $m \times m$ block submatrices of $P_S^{(p)}(\widetilde{A}^{(1,p)})^{-1} R_I^{(p)} A^{(0)} P R_I$ given by

$$-\widetilde{\mathcal{V}}_j\mathcal{S} = \begin{bmatrix} \Phi^0\Psi^j(\Phi^m - \Psi) & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots \\ \Phi^{m-1}\Psi^j(\Phi^m - \Psi) & \mathbf{0} & \cdots & \mathbf{0} \end{bmatrix}, -\widetilde{\mathcal{D}}\mathcal{S} = \begin{bmatrix} \Phi^0\Psi^{w-1}\Phi^m & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots \\ \Phi^{m-1}\Psi^{w-1}\Phi^m & \mathbf{0} & \cdots & \mathbf{0} \end{bmatrix}.$$

Note that $-\widetilde{\mathcal{V}}_j\mathcal{S}$ is identical to (4.20). The error propagation operator with approximate coarse grid, $\mathcal{E}_a$, is then given by

$$\mathcal{E}_a = \begin{array}{cl} \begin{array}{ccc} \overbrace{\qquad}^{m} & \overbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad}^{m(N_t^{(1)}-1)} & \overbrace{\quad}^{1} \end{array} & \\ \begin{bmatrix} \mathbf{0} & \cdots & & & & & & & \mathbf{0} \\ \mathcal{Z}_0 & \mathbf{0} & \cdots & & & & & & \mathbf{0} \\ \vdots & \ddots & \ddots & \cdots & & & & & \mathbf{0} \\ \mathcal{Z}_{w-2} & \cdots & \mathcal{Z}_0 & \mathbf{0} & \cdots & & & & \mathbf{0} \\ \mathcal{W} & \mathcal{Z}_{w-2}\cdots & \mathcal{Z}_0 & & \mathbf{0} & \cdots & & & \mathbf{0} \\ \mathbf{0} & \ddots & \ddots & \vdots & \ddots & \ddots & & & \vdots \\ \mathbf{0} & \ddots & \mathcal{W} & \mathcal{Z}_{w-2} & \cdots & \mathcal{Z}_0 & \mathbf{0} & & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & \bar{\mathcal{W}} & \bar{\mathcal{Z}}_{w-2}\cdots & \bar{\mathcal{Z}}_0 & & & \mathbf{0} \end{bmatrix} & \begin{array}{l} \}\, m \\ \}\, (w-1)m \\ \}\, (N_t^{(1)}-w)m \\ \}\, 1 \end{array} \end{array}, \tag{4.27}$$

with block matrices $\mathcal{Z}_j = -\widetilde{\mathcal{V}}_j\mathcal{S}$ and $\mathcal{W} = -\widetilde{\mathcal{D}}\mathcal{S}$ and $\bar{\mathcal{W}}$ and $\bar{\mathcal{Z}}_j$ are again identical to first row of $\mathcal{W}$ and $\mathcal{Z}_j$, respectively.

4.4.4 Convergence bounds

We follow the framework from Section 4.3.4, drop the subscript a of $\mathcal{E}_a$, and analyze again the C-C principle submatrix of (4.27) given by

$$\mathcal{E}_{cc} = \begin{bmatrix} \mathbf{0} & & & & & & \\ (\Phi^m - \Psi) & \mathbf{0} & & & & & \\ \vdots & (\Phi^m - \Psi) & \ddots & & & & \\ \Psi^{w-2}(\Phi^m - \Psi) & \vdots & \ddots & \mathbf{0} & & & \\ \Psi^{w-1}\Phi^m & \Psi^{w-2}(\Phi^m - \Psi) & \dots & (\Phi^m - \Psi) & \mathbf{0} & & \\ & \ddots & \ddots & \vdots & \ddots & \ddots & \\ & & \Psi^{w-1}\Phi^m & \Psi^{w-2}(\Phi^m - \Psi) & \dots & (\Phi^m - \Psi) & \mathbf{0} \end{bmatrix}.$$

Consider the case of Φ and Ψ being simultaneously diagonalizable. Let U again denote the shared eigenvector matrix of Φ and Ψ, with eigenvalues $\mu \in \sigma(\Psi)$ and $\lambda \in \sigma(\Phi)$, where $\sigma(\Psi)$ and $\sigma(\Phi)$ denote the spectrum of Ψ and Φ, respectively.

Then $\widetilde{\mathcal{E}}_{cc}$ is defined for a fixed pair of eigenvalues $\{\mu, \lambda\}$:

$$\widetilde{\mathcal{E}}_{cc} :=$$

$$\begin{bmatrix} \mathbf{0} & & & & & & \\ (\lambda^m - \mu) & \mathbf{0} & & & & & \\ \vdots & (\lambda^m - \mu) & \ddots & & & & \\ \mu^{w-2}(\lambda^m - \mu) & \vdots & \ddots & \mathbf{0} & & & \\ \mu^{w-1}\lambda^m & \mu^{w-2}(\lambda^m - \mu) & \dots & (\lambda^m - \mu) & \mathbf{0} & & \\ & \ddots & \ddots & \vdots & \ddots & \ddots & \\ & & \mu^{w-1}\lambda^m & \mu^{w-2}(\lambda^m - \mu) & \dots & (\lambda^m - \mu) & \mathbf{0} \end{bmatrix}.$$

Following the strategy from Section 4.3.4, we obtain the bound for two-level AT-MGRIT with F-relaxation given by

$$\begin{aligned} \|\widetilde{\mathcal{E}}_{cc}\| \leq \|\widetilde{\mathcal{E}}_{cc}\|_1 &= |\lambda^m - \mu| \sum_{\ell=0}^{w-2} |\mu^\ell| + |\lambda^m \mu^{w-1}| \\ &= \frac{|\lambda^m - \mu|(1 - |\mu|^{w-1})}{1 - |\mu|} + |\lambda^m \mu^{w-1}|. \end{aligned} \tag{4.28}$$

Results are summarized in the following theorem.

Theorem 4.2 (Two-level convergence). *Let Φ and Ψ be simultaneously diagonalizable with eigenvectors U, and consider two-level AT-MGRIT with coarsening factor m and local coarse-grid size w. For a given CF-splitting of time points, error propagation takes the form*

$$\mathcal{E}^\ell := \begin{bmatrix} \mathcal{E}_{ff} & \mathcal{E}_{fc} \\ \mathcal{E}_{cf} & \mathcal{E}_{cc} \end{bmatrix}^\ell = \begin{bmatrix} \mathbf{0} & \mathcal{E}_{fc}\mathcal{E}_{cc}^{\ell-1} \\ \mathbf{0} & \mathcal{E}_{cc}^\ell \end{bmatrix}.$$

Further, assume Φ and Ψ are stable in an eigenvalue sense, that is, $|\mu|, |\lambda| < 1$ for all $\mu \in \sigma(\Psi)$, $\lambda \in \sigma(\Phi)$, and define

$$\varphi_{tg}(\mu, \lambda) := \frac{|\lambda^m - \mu|}{1 - |\mu|}$$

for eigenvalue pairs (with shared eigenvector) $\{\mu, \lambda\}$. Then,

$$\|\mathcal{E}_{cc}\|_{(\widetilde{U}\widetilde{U}^*)^{-1}} \leq \max_{\{\mu,\lambda\}} \left(\varphi_{tg}(\mu, \lambda) + |\mu^{w-1}| \left(|\lambda^m| - \varphi_{tg}(\mu, \lambda) \right) \right). \tag{4.29}$$

Proof. The proof follows from a simple expansion of (4.28). □

Corollary 4.3. *Under the same assumptions as in Theorem 4.2,*

$$\|\mathcal{E}_{cc}\|_{(\widetilde{U}\widetilde{U}^*)^{-1}} \leq \max_{\{\mu,\lambda\}} \left(\varphi_{tg}(\mu,\lambda) + |\mu^w| \left(1 - \varphi_{tg}(\mu,\lambda)\right)\right). \tag{4.30}$$

Proof. Note that

$$\frac{|\lambda^m - \mu|(1-|\mu|^{w-1})}{1-|\mu|} = \frac{|\lambda^m - \mu|(1-|\mu|^w)}{1-|\mu|} - |\mu^{w-1}||\lambda^m - \mu|.$$

In addition, we have $|\lambda^m \mu^{w-1}| = |\mu^{w-1}(\lambda^m - \mu) + \mu^w| \leq |\mu^{w-1}||\lambda^m - \mu| + |\mu^w|$. Plugging these two results into (4.28) yields an upper bound

$$\|\mathcal{E}_{cc}\|_{(\widetilde{U}\widetilde{U}^*)^{-1}} \leq \max_{\{\mu,\lambda\}} \left(\frac{|\lambda^m - \mu|(1-|\mu|^w)}{1-|\mu|} + |\mu^w|\right).$$

An analogous expansion as used in Theorem 4.2 completes the proof. □

Note that the first term, φ_{tg}, in (4.29) and (4.30), to $\mathcal{O}(1/N_t^{(1)})$, provides necessary and sufficient conditions for convergence of two-level MGRIT (4.22), while the second term introduces an error perturbation that results from truncating the coarse grid. Although Corollary 4.3 is less tight than Theorem 4.2, it provides a more intuitive description of convergence. Note that error modes which converge fast for traditional MGRIT, $\varphi_{tg}(\mu,\lambda) \approx 0$, lead to the largest perturbation of convergence for AT-MGRIT, $\approx |\mu^k|$ (this also suggests convergence will be better for a more "diffusive" coarse solver, that is, a coarse solver with generally smaller eigenvalues). In contrast, there will be much less degradation in convergence for modes that are relatively slow to converge for traditional two-level MGRIT.

This leads to a further important observation on convergence of AT-MGRIT: with some algebra,[1] one can show that the "error" subdiagonal, that is, the subdiagonal of $\widetilde{\mathcal{E}}_{cc}$ that lacks a $\lambda^m - \mu$ scaling, is propagated out of the matrix after $\lceil (N_t^{(1)} + 1)/w \rceil$ iterations (i. e., all matrix entries then have at least one power of $\lambda^m - \mu$). This suggests a natural heuristic to choose w:

Choice of w: *choose w at least large enough so that $\lceil (N_t^{(1)} + 1)/w \rceil$ approximates the number of iterations to converge for traditional two-level MGRIT.*

[1] There are various ways to show this; perhaps the most formal is in noting that multiplication of Toeplitz matrices such as $\widetilde{\mathcal{E}}_{cc}^{\ell}$ corresponds to finite discrete convolutions. One can also simply expand $\widetilde{\mathcal{E}}_{cc}^{p} = (\widehat{\mathcal{E}}_{cc} + \mu^{w-1}\lambda^m I_{-w})^p$, where I_{-w} is a diagonal of ones on the wth subdiagonal.

The number of iterations for two-level MGRIT convergence is determined by the slowest converging modes, which are in turn the least affected in convergence of AT-MGRIT by the perturbation term in (4.29) and (4.30) (i. e., we expect these modes to converge in a roughly similar number of iterations for AT-MGRIT). As mentioned previously, the fastest-converging modes for MGRIT, however, can suffer significant degradation of convergence in AT-MGRIT. Thus, we choose w so that the error perturbation for these terms is eliminated via nilpotency at (approximately) the same number of iterations as the slowest converging modes for standard MGRIT will have converged. After this nilpotency is achieved, these "fast" modes will rapidly converge (if they have not already) due to $\varphi_{tg}(\mu, \lambda) \ll 1$.

Last, it is important to remember that theory developed in this section is for two-level AT-MGRIT applied to linear problems. The resulting heuristic for choosing w provides a good starting point, but multilevel AT-MGRIT or application to nonlinear problems may require larger w.

4.5 Convergence criterion

All PinT methods presented are iterative methods that measure the quality of the solution at a certain point within the algorithm. Typically, the quality is expressed by an integer quantity, e. g., an arbitrary norm over the residual, where the lower the value, the better the quality. When this quality of the solution is good enough, i. e., the corresponding quantity is below a chosen threshold κ, the algorithm stops. The choice of the convergence criterion is a complex issue, and there are several approaches, e. g., the jump of the approximation between two iterations [41], determining convergence using finest-level information [70], the residual at single time points [46], or the space-time residual [30]. Unfortunately, however, there is no detailed study of the different convergence criteria. Here, we categorize the different criteria at a very high level into two groups: local and global criteria.

Local criteria measure the quality of the solution at a set of time points, considering each point individually and determining for each point whether the approximation is sufficient or not. Thus, local criteria may result in the convergence criterion already being met for some points, but not for others. Subsequent iterations of the iterative method have the option to consider only the time points that have not yet converged. The quality of the solution can be determined, for example, by the jump between two iterations or the residual, each measured in an arbitrary norm. If the measured result for a time point is below κ, the approximation is sufficient for this time point. Due to the evolutionary nature, an additional condition is often added, namely that the point immediately before the point under consideration must also have fallen below the threshold. This ensures that there is a dependence of the local criterion on the initial value.

A global criterion builds on a local criterion. First, the local criterion is determined for each time point and then a global measure is formed over all local criteria, e. g., using an arbitrary norm. If this global measure is smaller than κ, the algorithm terminates for all points simultaneously.

In the remainder of this thesis, we use global convergence criteria (unless otherwise stated), which is the typical choice for the MGRIT algorithm [30].

Chapter 5

Implementation

In the following chapter, we present the Python implementation PyMGRIT [53] of MGRIT, which was used for all of the following numerical experiments in Chapter 6 and Chapter 7, and was developed in the context of this thesis. The PyMGRIT framework includes many different variants of the MGRIT algorithm, ranging from different multigrid cycle types and relaxation schemes, various coarsening strategies, including time-only and space-time coarsening, and the ability to utilize different time integrators on different levels in the multigrid hierachy. It also includes an implementation of the AT-MGRIT algorithm. PyMGRIT supports both serial runs, suitable for prototyping and testing new approaches, as well as parallel runs using the Message Passing Interface (MPI). The code of the package can be found in the GitHub repository [52].

5.1 PyMGRIT

PyMGRIT is based on different classes, each of which specifies different required components; more precisely, the framework is based on four different class types:

- Solver: the solver class provides implementations of the solver, for example, MGRIT or AT-MGRIT.
- Vector: vector classes contain the solution at a single point in time. Every vector class must inherit from PyMGRIT's core Vector class.
- Application: application classes contain information about the problem we want to solve. Every application class must inherit from PyMGRIT's core Application class.

- GridTransfer: grid transfer classes contain information about the transfer of spatial grids between consecutive MGRIT levels. Every grid transfer class must inherit from `PyMGRIT`'s core `GridTransfer` class.

The three types of classes Vector, Application and GridTransfer are all based on abstract super classes. These classes independently create some structures that are valid for each type of class and also ensure that all necessary member variables and member functions exist in the respective child classes. A developer who wants to create and solve a problem that is not included in the `PyMGRIT` package usually only has to specify parts of the four classes. In most cases it is sufficient to write a vector class and an application class. The grid transfer class is primarily needed for the additional feature of spatial coarsening, and if spatial coarsening is not used, it is automatically handled by the framework and nothing needs to be done. In genereal, the solver classes can be used without modifications.

In the following, we demonstrate how an existing time stepping code for solving a time-dependent problem can easily be transformed for the usage in the `PyMGRIT` framework. Then, we show a typical main routine of an application code that uses a `PyMGRIT` solver to solve the problem. As an example, we consider a simple example of a scalar ODE. More specifically, our goal is to solve the Dahlquist test problem,

$$u' = \lambda u \quad \text{in } (0, 5], \tag{5.1}$$

with $u(0) = 1$ and $\lambda = -1$.

Listing 5.1 shows a typical time stepping code for solving problem (5.1) discretized by implicit Euler on a temporal mesh with 101 points. The code consists of three components: first, the initial condition `value = 1` is set in line 2. The variable `value` is further used to store the propagated solution at the current time. The second component consists of the time information in lines 3 and 4. The temporal grid contains $nt = 101$ points in the time interval $[0, 5]$ and is created using the `Numpy` [108] function `linspace`. In the last step, we iterate over all points in time and apply the time integration in form of implicit Euler. In summary, we have as components a variable that contains the solution at a point in time, time information belonging to the problem, and the time-integration loop.

To implement these components in `PyMGRIT`, the first step is to write a vector class `VectorDahlquist`, which stores the solution at a point in time and inherits from the `PyMGRIT` core class `Vector`. The member variable `value` contains the solution of a point in time. Furthermore, the following member functions have to be implemented:

- `set_values`: receives data values and overwrites the values of the vector data.

```
constant_lambda = -1 # set lambda to -1
value = 1  # initial solution value
nt = 101  # number of time points
t = np.linspace(0, 5, nt)  # nt evenly spaced numbers in interval [0,5]
# implicit Euler time integration
for i in range(1, nt):
    value = 1 / (1 - (t[i] - t[i - 1]) * constant_lambda) * value
```

Listing 5.1: Example time stepping code for problem (5.1), discretized by implicit Euler on a temporal mesh with 101 points.

```
class VectorDahlquist(Vector):
    def __init__(self, value):
        super().__init__()
        self.value = value

    def __add__(self, other):
        return VectorDahlquist(self.get_values() + other.get_values())

    def __sub__(self, other):
        return VectorDahlquist(self.get_values() - other.get_values())
```

Listing 5.2: Vector class for Dahlquist's test equation. Note that the definition of the class is not complete, the member functions `set_values`, `get_values`, `__mul__`, `clone`, `clone_zero`, `clone_rand`, `norm`, `pack`, and `unpack` are not shown.

- `get_values`: returns the vector data.
- `clone`: clones the object.
- `clone_zero`: returns a vector object initialized with zeros.
- `clone_rand`: returns a vector object initialized with random data.
- `__add__`: addition of two vector objects.
- `__sub__`: subtraction of two vector objects.
- `norm`: norm of a vector object.
- `__mul__`: multiplies a vector object by a float.
- `pack`: packs the vector data into a communicable object.
- `unpack`: reverse operation of `pack`.

The definition of the class `VectorDahlquist` with implementations of the constructor and of the functions `__add__`, `__sub__` is shown in Listing 5.2. The implementation of the other functions is straightforward and not specified in detail here.

Second, we write an application class `Dahlquist` which contains information about the problem we want to solve. This class contains information about the

```python
class Dahlquist(Application):
    def __init__(self, constant_lambda = -1, *args, **kwargs):
        super().__init__()
        self.c_lambda = constant_lambda
        self.vector_template = VectorDahlquist(0) # Data structure for any time point
        self.vector_t_start = VectorDahlquist(1) # Set the initial condition

    # Time integration routine
    def step(self,
             u_start: VectorDahlquist,
             t_start: float,
             t_stop: float) -> VectorDahlquist:
        tmp = 1/(1-(t_stop-t_start)*self.c_lambda)*u_start.get_values()
        return VectorDahlquist(tmp)
```

Listing 5.3: Application class for Dahlquist's test equation.

time grid and the step function and is shown in Listing 5.3. The time information is automatically provided by the PyMGRIT core class Application, from which every PyMGRIT application must inherit. The function step must be defined and contains the time integration routine, which is the same as in Listing 5.1 except for names and accesses. To compute the new solution, the function receives as parameters the solution of the previous time point, u_start, as well as the start point and the end point of the time integration step, t_start and t_stop, respectively. Furthermore, two mandatory member variables, vector_template and vector_t_start, must be created in the application class. The variable vector_template stores an instance of the corresponding Vector class, i. e., the DahlquistVector class, and vector_t_start defines the initial condition using the same class. This is all we need for our test problem. The application class is used in the following to solve the problem using PyMGRIT.

The main routine of an application code that uses PyMGRIT to solve Dahlquist's test problem is presented in Listing 5.4. The program uses a two-level MGRIT algorithm to solve the same problem as the time stepping code in Listing 5.1. The 101 time points are composed of one point for the start time $t = 0$ and 100 other time points. Note that one point for the start time is always included in the time interval in PyMGRIT. The structure of a main routine usually consists of three steps. First, the problem is created. Then, a multigrid hierarchy is constructed for this problem. Finally, the problem is solved using the MGRIT algorithm. In our example, for the first step, an instance of PyMGRIT's class Dahlquist is created in line 2 that describes the fine problem. The time domain is passed to the problem class by using the parameters t_start and t_stop for specifying the time interval bounds and the parameter nt for the number of time steps. Afterwards, for the second step, a multilevel hierarchy is constructed in line 5, based on the problem instance dahlquist. Here, the auxiliary function simple_setup_problem is used and a two-level hierarchy with a coarsening factor of two is chosen. The result is

```
1   # Create Dahlquist's test problem with 101 time steps in the interval [0, 5]
2   dahlquist_level_0 = Dahlquist(t_start=0, t_stop=5, nt=101)
3
4   # Construct a two-level multigrid hierarchy for the test problem using a
    coarsening factor of 2
5   dahlquist_multilevel_structure = simple_setup_problem(problem=dahlquist,
6                                                         level=2,
7                                                         coarsening=2)
8
9   # Set up the MGRIT solver for the test problem and set the solver tolerance
    to 1e-10
10  mgrit = Mgrit(problem=dahlquist_multilevel_structure)
11
12  # Solve the test problem
13  info = mgrit.solve()
```

Listing 5.4: Typical main routine of an application code that uses `PyMGRIT`.

a coarse level with 51 time points. Note that `PyMGRIT` provides several ways to pass the time domain information to a problem class and to create a multilevel hierarchy so that any kind of uniform or non-uniform coarsening strategy can be applied. For the third step, the MGRIT solver for the test problem is set up in line 10 as an instance of `PyMGRIT`'s core class `Mgrit` using the multilevel object `dahlquist_multilevel_structure`. Note that here only the problem hierarchy is passed to the constructor and all other settings of the MGRIT algorithm are set by default variables, but they can easily be set explicitly in the constructor. For example, the following parameters can be set: maximum number of iterations, convergence tolerance, usage of nested iteration strategy, relaxation scheme, cycle type, and others. For a complete list we refer to the documentation of `PyMGRIT` [51]. Finally, the problem is solved by calling the `solve` routine of the solver `mgrit` in line 13. The solver returns some statistical information about the run in the dictionary `info`.

The parallelization of the time dimension is handled completely automatically in the solver, so the user does not have to care about parallelization details. We will not go into implementation details here, but refer to Section 8.1.2 for the typical strategy of parallelization within PinT methods, which is also used in `PyMGRIT`, and to [52, 53] for more information about the package.

Chapter 6

Numerical experiments: AT-MGRIT investigations

In this chapter, we show numerical results presented in [55] for the AT-MGRIT algorithm for two problems. First, we consider a simple model problem, the one-dimensional (1D) heat equation, and investigate the numerical behavior of the algorithm for the new parameter w, the size of the local coarse grids. We also compare the theoretical results from Section 4.4.4 with the numerical results. The second problem, the 2D Gray-Scott example of a chemical reaction of two substances, presents a more challenging nonlinear problem. In particular, we consider the runtime results of the AT-MGRIT algorithm and compare them with the runtime results of the corresponding MGRIT variants. Finally, we briefly discuss the effect of spatial parallelism for different methods.

All simulations were performed on an Intel Xeon Phi Cluster consisting of four 1.4 GHz Intel Xeon Phi processors. The code for all experiments can be found in the `PyMGRIT` repository [52]. For all experiments, we use all possible resources for temporal parallelization, i. e., we do not use spatial parallelization (largely due to limited resources). For a brief discussion on the effect of spatial parallelism for the different algorithms, see Section 6.3.

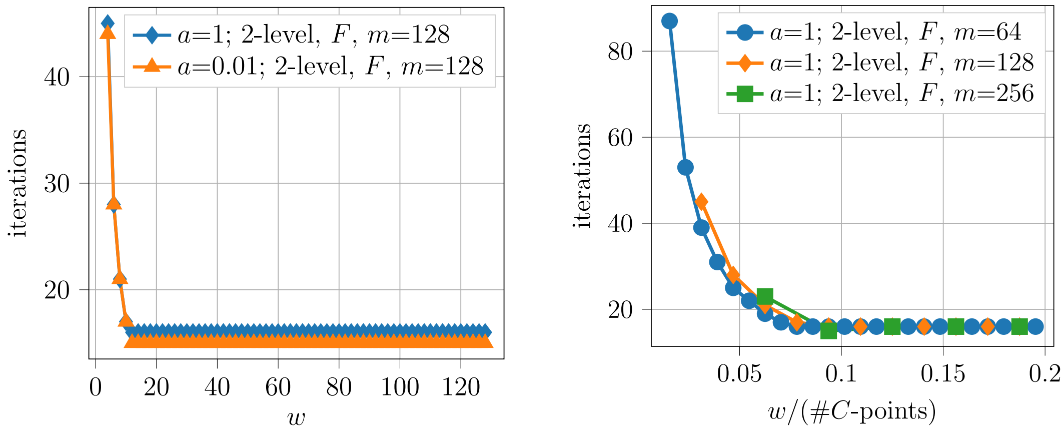
6.1 Heat equation

The parameter w defines the size of the local coarse grids and, thus, the number of sequential solves needed on the coarse grid. In the following, we consider the influence of the parameter w on the convergence of AT-MGRIT applied to a standard model problem for PinT methods, the 1D heat equation,

$$\mathbf{u}_t - a\mathbf{u}_{xx} = \mathbf{b}(\mathbf{x}, t) \quad \text{in} \quad [0,3] \times [0,\pi], \tag{6.1}$$

where a is the thermal conductivity, and subject to the initial condition $\mathbf{u}(x,0) = \sin(\pi x), 0 \leq x \leq 3$ and homogeneous Dirichlet boundary conditions in space. The forcing term is chosen as $\mathbf{b}(x,t) = -\sin(\pi x)(\sin(t) - \pi^2 \cos(t)), 0 \leq x \leq 3, 0 \leq t \leq \pi$, such that the exact solution is given by $\mathbf{u}(x,t) = \sin(\pi x)\cos(t)$ for $a = 1$. We first examine the behavior of w for $a = 1$ and $a = 0.01$, and then choose $a = 1$ for more detailed results.

We discretize (6.1) using second-order central finite differences with 1,025 degrees of freedom in space and on an equidistant time grid with 16,384 time points using implicit Euler. We investigate the behavior of the AT-MGRIT algorithm for the two-level case with F-relaxation, and choose different coarsening factors m and local grid sizes w. We restrict ourselves to the two-level case, since we want to study the effect of using local coarse grids of various sizes w. For all simulations, the convergence criterion is based on the discrete 2-norm of the absolute space-time residual with a tolerance of 10^{-7} and a random initial guess is chosen for all time points except for the initial condition. This choice guarantees that no knowledge of the right-hand side is used that could affect the convergence. Note that this is only a good choice for investigating the behavior of the algorithm and is not recommended in practice.



(a) Iterations to convergence as a function of w.

(b) Iterations to convergence as a function of w divided by the number of C-points.

Figure 6.1: Required iterations for AT-MGRIT variants for the 1D heat equation.

Figure 6.1(a) shows the required number of iterations to reach the convergence criterion for a two-level AT-MGRIT variant with F-relaxation and a coarsening factor of $m = 128$ as a function of size w for the two choices of the thermal conductivity. Note that while the variant with $w = 128$, which is equivalent to two-level MGRIT, performs 127 sequential time steps on the coarse level, equivalent convergence can be obtained with $w = 12$, which means 10× less coarse-grid solves. Note that the behavior is similar for both choices of a. Figure 6.1(b) presents iterations to convergence as a function of *the ratio* of local to global coarse-grid points. For three different coarsening factors, we see that convergence does not improve beyond the same ratio of $w/(\#C\text{-points})$, in this case about 0.08. Although this parameter is likely problem specific, Figure 6.1 does suggest the choice of w is relatively agnostic to that of the coarsening factor by posing it relative to the global coarse-grid size.

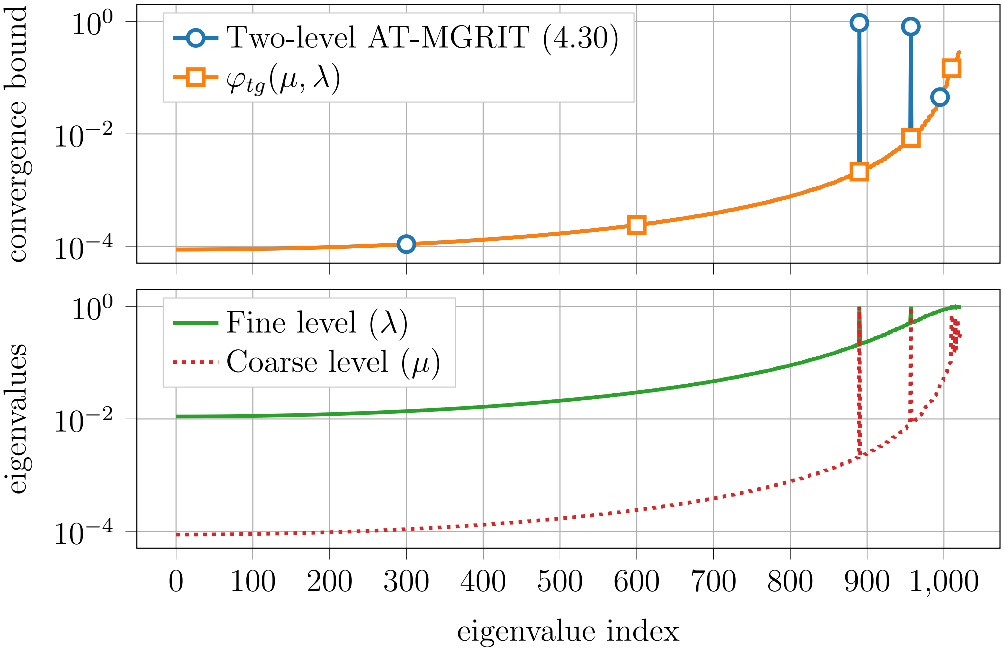


Figure 6.2: Theoretical bound on convergence rate based on (4.30), φ_{tg}, and eigenvalues sorted by φ_{tg}, for $m = 128$, $w = 12$ and $a = 1$.

To better understand the convergence behavior described in Section 4.4.4, for each spatial eigenvalue Figure 6.2 plots the theoretical convergence rate (4.30), the (asymptotic in $N_t^{(1)}$) two-grid rate φ_{tg}, and the corresponding eigenvalues for the coarse- and fine-propagator for two-level AT-MGRIT with $m = 128$ and $w = 12$. Notice that on an eigenvector basis, theoretical convergence of AT-MGRIT almost exactly matches that of two-level MGRIT, with (in this case) two exceptions, given by the blue dots with convergence ≈ 1. Each of these spatial eigenmodes correspond to a coarse-propagator eigenvalue $|\mu| \approx 1$ and $\varphi_{tg} \ll 1$, which (see Corollary 4.3) leads to a significant degradation in convergence (in a single-iteration sense). Figure 6.3 plots *observed* convergence behavior for two-level AT-MGRIT with coarsening factor $m = 128$ and various choices for w. The variant with $w = 128$ (i. e., two-level MGRIT) has uniform convergence behavior, while convergence for smaller w is split into three parts. Initially, convergence is much slower than for $w = 128$, due to the spatial eigenmodes with $|\mu^k| \approx 1$ and $\varphi_{tg} \ll 1$ discussed above. The smaller w is chosen, the slower is the convergence, since the convergence perturbation of $|\mu^k|(1 - \varphi_{tg})$ in Corollary 4.3 decreases with increased w. However, after almost exactly $(N_t^{(1)} + 1)/w$ iterations (see Theorem 4.2 and surrounding discussion), once the initial condition has been implicitly propagated over all local coarse grids, these problematic modes are eliminated via nilpotency and a drastic improvement in convergence can be seen for all three variants; e. g., for $w = 8$, one iteration suddenly reduces the residual norm almost four orders of magnitude. This rapid convergence lasts until the residual norm matches that of two-level MGRIT, and convergence rates thereafter follow two-level MGRIT. Comparing the theoretical and numerical results for $m = 12$, the theoretical bound (see Figure 6.2) is given by 0.94987 and the maximum numerical convergence factor between two iterations for the equivalent setting (see Figure 6.3) is given by 0.7418. Note, observed convergence is better than the theoretical bound due to many modes being rapidly attenuated. Only a few modes are very slow to converge, with rates likely close to the theoretical bound, but these modes degrade the average (across all error modes) convergence rate.

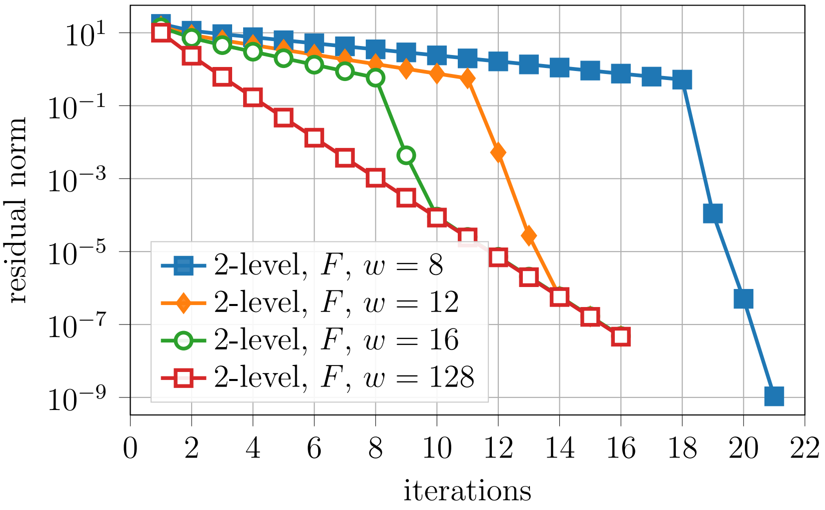


Figure 6.3: Residual norm as a function of iterations for two-level AT-MGRIT with coarsening factor $m = 128$ and $a = 1$.

6.2 Gray-Scott problem

We consider the 2D Gray-Scott problem [84] of a chemical reaction of two components $\mathcal{U}_c$ and $\mathcal{V}_c$, given by

$$\begin{aligned}\mathbf{u}_t &= D_u \Delta \mathbf{u} - \mathbf{u}\mathbf{v}^2 + F_r\left(1 - \mathbf{u}\right),\\ \mathbf{v}_t &= D_v \Delta \mathbf{v} + \mathbf{u}\mathbf{v}^2 - \left(K_r + F_r\right)\mathbf{u},\end{aligned}$$

where $\mathbf{u} = \mathbf{u}\left(\mathbf{x}, \mathbf{y}, t\right)$ and $\mathbf{v} = \mathbf{v}\left(\mathbf{x}, \mathbf{y}, t\right)$ are the concentration of $\mathcal{U}_c$ and $\mathcal{V}_c$, respectively, D_u and D_v are the diffusion rates, F_r is the feed rate, and K_r is the removal rate. For our simulations, we choose the spatial domain $[0, 2.5]^2$ with periodic boundary conditions, and the time interval $[0, 256]$. Further, we choose the parameters $F_r = 0.024$, $K_r = 0.06$, $D_u = 8 \times 10^{-5}$, and $D_v = 4 \times 10^{-5}$, and we consider the initial value

$$\begin{aligned}u\left(x, y, 0\right) &= 1 - 2\left(0.25 \sin\left(4\pi x\right)^2 \sin\left(4\pi y\right)^2\right), \qquad &(x, y) \in [1, 1.5]^2,\\ v\left(x, y, 0\right) &= 0.25 \sin\left(4\pi x\right)^2 \sin\left(4\pi y\right)^2, \qquad &(x, y) \in [1, 1.5]^2,\end{aligned}$$

and $u\left(x, y, 0\right) = 1$ and $v\left(x, y, 0\right) = 0$ otherwise. The problem is discretized using central finite differences with 128^2 points in space and on an equidistant time grid with 16,384 points using implicit Euler. We solve the resulting nonlinear problem using Newton's method of `PETSc` [6] with a relative and absolute tolerance of 10^{-10}.

We apply two-level and three-level AT-MGRIT variants, and compare the variants with the corresponding variants of the MGRIT algorithm. For the two-level variants, we apply F-relaxation and we choose the coarsening factor such that the number of coarse-grid points is equal to the number of processes used for the simulation enabling perfect parallelization on the fine level. For the three-level algorithms, we apply FCF-relaxation, non-uniform coarsening strategies with a coarsening factor of $m^{(0)} = 64$ between the fine and the intermediate level, and different factors between the intermediate and the coarse level. This choice allows perfect fine level parallelization when 256 processes are used, since there is exactly one interval of FC-points on each process. Furthermore, we use nested iterations to compute an improved initial guess. In the nested iteration strategy, MGRIT solves the global coarse-grid problem at the coarsest level, while AT-MGRIT uses the local coarse grids instead of the global grid. The convergence criterion for all variants is based on the discrete 2-norm of the space-time residual with a tolerance of 10^{-7}.

Table 6.1 shows the number of iterations and runtimes of the setup and solve phases of two-level AT-MGRIT and two-level MGRIT variants using N_P processes. The setup time consists of computing an improved initial guess and the solve time consists of applying the algorithm. The results show that iteration

Method	m	w	N_P	# Iters	Setup time	Solve time	Speedup w.r.t MGRIT
Two-level MGRIT	512	-	32	12	1,338 s	172,068 s	-
	256	-	64	10	2,308 s	89,288 s	-
	128	-	128	9	3,958 s	66,485 s	-
	64	-	256	7	7,646 s	66,272 s	-
Two-level AT-MGRIT	512	16	32	12	701 s	165,351 s	1.04
	256	32	64	10	1,167 s	78,230 s	1.15
	128	64	128	9	2,022 s	48,675 s	1.39
	64	128	256	7	3,812 s	39,895 s	1.69

Table 6.1: Iteration counts, setup times (for computing an improved initial guess), and runtimes of the solve phase of two-level AT-MGRIT and two-level MGRIT variants applied to the 2D Gray-Scott problem for various numbers of processes N_P.

counts of AT-MGRIT are equal to iteration counts of MGRIT with the same coarsening strategy. Furthermore, a finer coarse grid significantly reduces the number of iterations required. While 12 iterations are needed for the two-level variants with a coarse grid of only 32 points, this number is reduced to seven iterations for the variants with 256 coarse-grid points. However, this reduction in iterations is accompanied by significantly more expensive sequential coarse-grid solves, reflected in increasing setup times with increasing points on the coarse grid. However, if the number of points on the coarse grid doubles, the setup time does not double. This is because a smaller time step requires fewer Newton iterations and, thus, affects the duration of the application of each time integration. The setup time of each AT-MGRIT variant is about half as long as that of the corresponding two-level MGRIT variant due to the choice of w. Looking at the runtimes of the solve phase, we see that AT-MGRIT is always faster than the corresponding MGRIT variant, achieving a speedup of up to a factor of 1.69. Furthermore, we see that while the two-level MGRIT algorithm does not scale for more than 128 processes, since the serial part of the algorithm dominates the benefit of the additional parallelization of the fine level, the AT-MGRIT algorithm shows good parallel scaling up to 256 processes. Note that the selected coarsening factor was chosen to use a maximum of 256 processes, which is the maximum number of processes that can be utilized on the used cluster. To use more processes effectively, a different coarsening strategy must be applied. In this case, we expect that the work on the fine levels can be better parallelized, making the work on the coarsest level even more dominant and further increasing the advantage of AT-MGRIT compared to MGRIT.

Method	m	w	# Iters	Setup time	Solve time	Speedup w.r.t MGRIT
MGRIT	(64,16)	-	7	3,525 s	43,604 s	-
	(64,8)	-	7	3,498 s	38,420 s	-
	(64,4)	-	7	4,980 s	42,285 s	-
	(64,2)	-	6	8,075 s	45,131 s	-
AT-MGRIT	(64,16)	8	7	2,864 s	41,054 s	1.07
	(64,8)	16	7	2,174 s	33,688 s	1.17
	(64,4)	32	7	2,713 s	34,063 s	1.29
	(64,2)	64	7	4,247 s	38,571 s	1.24

Table 6.2: Iteration counts and runtimes of the setup and solve phase on 256 processes of three-level AT-MGRIT and MGRIT variants with FCF-relaxation and different non-uniform coarsening strategies applied to the 2D Gray-Scott problem.

Table 6.2 presents similar results to Table 6.1 for four different three-level variants of AT-MGRIT and MGRIT with FCF-relaxation on 256 processes. The number of iterations here does not depend as much on the coarsest grid as in the two-level case, but we still see that the MGRIT variant with the coarsening strategy $(64, 2)$, i. e., the variant with the most points on the second level, requires the fewest iterations. The corresponding AT-MGRIT variant needs one additional iteration, but after the sixth iteration the convergence criterion is slightly missed. A minimal increase in w would likely eliminate this extra iteration. In terms of solve times, we see that all variants of the AT-MGRIT algorithm are faster than the corresponding MGRIT variants, even the variant that requires an additional iteration. Again, the more points on the coarsest level, the higher the speedup of AT-MGRIT over MGRIT.

Note that for this problem, adding a coarser level to the three-level MGRIT variants does not guarantee further reduction of the runtime. Rather, adding a level may increase the runtime. For example, the four-level MGRIT with the coarsening strategy $(64, 8, 2)$, which adds another level with a coarsening factor of two to the variant with the coarsening factor $(64, 8)$ from Table 6.2, requires eight iterations and the overall runtime (setup and solve) is 52,370 s. This observed behavior resulting from the additional coarse grid is probably due to two reasons:

First, the nonlinear solver converges more slowly on the coarsest level due to the larger time step size, which significantly increases the runtime of each time integration on the coarsest level. Due to this increase and the additional work on the new intermediate level, the additional level of the four-level variant does not contribute to a runtime reduction compared to the three-level variant, but increases the runtime. In addition, the coarser discretization at the new level affects the convergence of the method, so that an additional iteration is required to reach the stopping criterion. Note that this behavior is problem dependent and may behave differently with other problems.

Method	m	w	# Iters	Setup time	Solve time	Speedup w.r.t MGRIT
AT-MGRIT	(64,8)	4	10	1,140 s	42,063 s	0.97
	(64,8)	6	9	1,357 s	38,978 s	1.04
	(64,8)	8	8	1,543 s	35,767 s	1.12
	(64,8)	10	8	1,716 s	36,509 s	1.10
	(64,8)	12	8	1,877 s	37,060 s	1.08
	(64,8)	14	8	2,047 s	37,786 s	1.05
	(64,8)	16	7	2,174 s	33,688 s	1.17

Table 6.3: Iteration counts and runtimes of the setup and solve phase on 256 processes of three-level AT-MGRIT with FCF-relaxation, coarsening factor $(64, 8)$, and different choices of w applied to the 2D Gray-Scott problem.

Table 6.3 extends the results from Table 6.2 and shows the effect of different choices of w for the three-level AT-MGRIT with coarsening factor $(64, 8)$. We see that the number of iterations increases slightly as w decreases. Despite the increasing number of iterations, the runtime for all $w \geq 6$ is smaller than the runtime of the corresponding MGRIT variant from Table 6.2. For smaller w, the runtime is larger compared to the MGRIT variant because the cost of the additional iterations is more expensive than the cost reduction due to the local coarse grids.

Figure 6.4 shows the overall runtime for one AT-MGRIT variant (blue line) and the corresponding MGRIT variant (orange line) as a function of the number of processes and the runtime of time stepping using only one process (black dashed line) which is about four days. For reference, the green dotted line indicates the behavior of perfect scaling based on the runtime of time stepping. While the runtime almost halves when going from 32 to 64 processes, the scaling curve starts

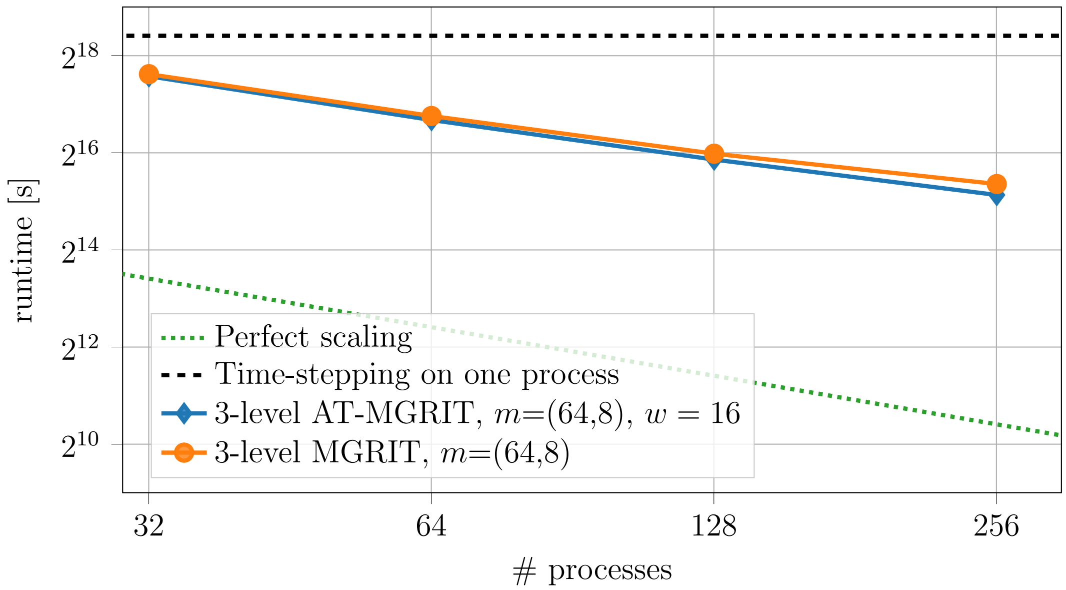


Figure 6.4: Strong scaling results for one three-level AT-MGRIT variant, the corresponding MGRIT variant, and sequential time stepping on one process applied to the 2D Gray-Scott problem. The green dotted line indicates the perfect scaling based on the runtime of time stepping.

to flatten slightly with a higher number of processes. This is mainly because only the fine level computations have an additional benefit from more processes due to the chosen coarsening strategy, and the runtime of the coarser levels becomes more and more dominant. However, compared to the corresponding MGRIT variant, the AT-MGRIT variant scales better due to its reduced work at the coarsest level.

6.3 Discussion of spatial parallelism

Here we demonstrate that the use of spatial parallelism has comparable effects on sequential time stepping (before saturation) as it does on MGRIT and AT-MGRIT. In particular, we emphasize that when spatial parallelism saturates, the observed near-perfect speedup obtained by spatial parallelism before saturation will extend to AT-MGRIT. To show this, we consider the same problem as for the experiments in Section 6.2.

Table 6.4 presents overall runtimes for using one and four processes in space for time stepping, two-level MGRIT, and two-level AT-MGRIT, the last two using 64 processes in time (same variants as in Table 6.1). We see that for all algorithms we get a speedup of about 3.8 by using four spatial processes compared to one process. Note that this problem scales well with spatial parallelization, and spatial parallelism (as in most cases) should be the first choice. However, spatial parallelization is exhausted at some point and temporal parallelization can then provide additional speedups.

Space processes	Time stepping one time process	Two-level MGRIT 64 time processes	Two-level AT-MGRIT 64 time processes
1	347,666 s	91,596 s	79,397 s
4	92,473 s	23,708 s	20,411 s

Table 6.4: Total runtimes using one and four processes in space for time stepping, MGRIT, and AT-MGRIT with $w = 32$, the latter two using a coarsening factor of 256 and 64 processes in time, applied to the 2D Gray-Scott problem.

Chapter 7

Numerical experiments: Induction machine

In this chapter, we present several numerical results [13, 54, 55] for the application of different MGRIT variants to eddy current simulations of an induction motor. First, we introduce the used model “im_3_kw” of an induction motor and present a variety of parameters that can be chosen in the model, such as different types of voltage sources. The solution of a dynamic system such as an induction motor excited with a periodic signal typically consists of a transient part followed by a (periodic) steady state that occurs when the transients are finally damped, see for example the time-domain torque evolution of the induction motor model “im_3_kw” in Figure 7.1.

In Section 7.2, we consider an induction motor driven by a PWM voltage source. When using a pulsed input signal, very small time steps are required to resolve the high frequency pulses. Therefore, we focus here mainly on simulating parts of the initial transient behavior of the machine, since considering a larger time interval would lead to very high computational costs due to the small time steps. For the simulations, we use the MGRIT algorithm and investigate the effect of spatial coarsening inside the method for the induction motor.

In Section 7.3 we consider the initial behavior of the machine until a periodic solution is reached. Therefore, we consider the induction motor driven by the sinusoidal voltage source, which allows us to consider larger time step sizes compared to the PWM voltage source. We first apply the MGRIT algorithm and the AT-MGRIT algorithm and compare them in Section 7.3.1. We then integrate the AT-MGRIT algorithm into an optimization algorithm that optimizes the height and width of the rotor bars with respect to the efficiency of the motor in Section 7.3.2.

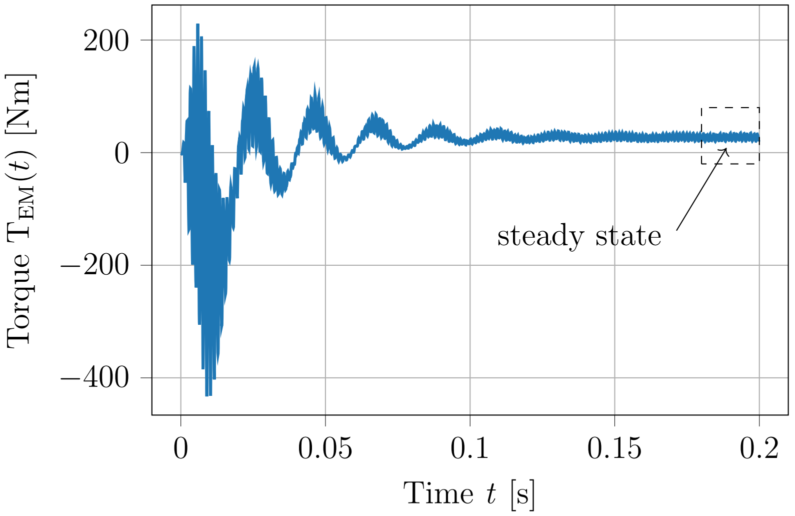


Figure 7.1: Time-domain evolution of the torque in the time interval [0,0.2] for the induction motor model "in_3_kw" with linear material properties, supplied by a three-phase voltage source with frequency $f = 50\,\mathrm{Hz}$ and amplitude $\hat{V} = 311.1\,\mathrm{V}$.

Note that instead of solving the initial value problem up to a point where periodic behavior is achieved, a problem with time-periodic boundary conditions instead of initial value conditions can also be considered. However, the consideration of time-periodic problems is beyond the scope of this thesis. For more information on time-periodic problems and PinT methods for dealing with them, we refer to [38], and to [8, 67] for the application of these methods to eddy current simulations. Furthermore, various other approaches have been proposed to accelerate the transient simulation of electrical machines, e. g., the time-periodic explicit error correction method [105, 106] or the extraction of a circuit model [9].

Again, all simulations were performed on an Intel Xeon Phi cluster consisting of four 1.4 GHz Intel Xeon Phi processors, and the code for all experiments is available at [52].

7.1 Numerical model

We model the semi-discrete eddy-current problem (3.36)-(3.37), supplied by a three-phase PWM voltage source, using the multi-slice finite element model "im_3_kw" of an electrical machine first presented in [48] and modified in [41] for the usage of a PWM voltage source. Specifically, the model "im_3_kw", depicted in Figure 3.8, models a 2D four-pole 3 kW squirrel-cage induction machine. Further, as typical for this kind of symmetric models, we consider only a quarter of the machine with periodic boundaries (Figure 7.3) to reduce simulation costs.

The induction motor is driven by either a sinusoidal or a PWM voltage source. More precisely, the sinusoidal voltage source is given by

$$v_{sin,j}(t) = \hat{V}\sin\bigl(2\pi f_{\sin} t - (j-1)\cdot 2\pi/3\bigr), \quad j = 1,2,3, \tag{7.1}$$

with $f_{\sin} = 50\,\mathrm{Hz}$, which corresponds to an electric period of 0.02 s. The PWM voltage source is given by

$$v_{pwm,j}(t) = \hat{V}\,\mathrm{sgn}\left[\bar{v}_{sin,j}(t) - c(t)\right], \quad j = 1,2,3, \tag{7.2}$$

with reference signals

$$\bar{v}_{sin,j}(t) = m_f v_{sin,j}(t), \quad j = 1,2,3, \tag{7.3}$$

with modulation factor $m_f = 0.8$ and carrier signal $c(t)$. The carrier signal, given by a bipolar trailing-edge modulation using a sawtooth carrier, is defined by

$$c(t) = \hat{V}(2(f_s t - \lfloor f_s t \rfloor) - 1), \tag{7.4}$$

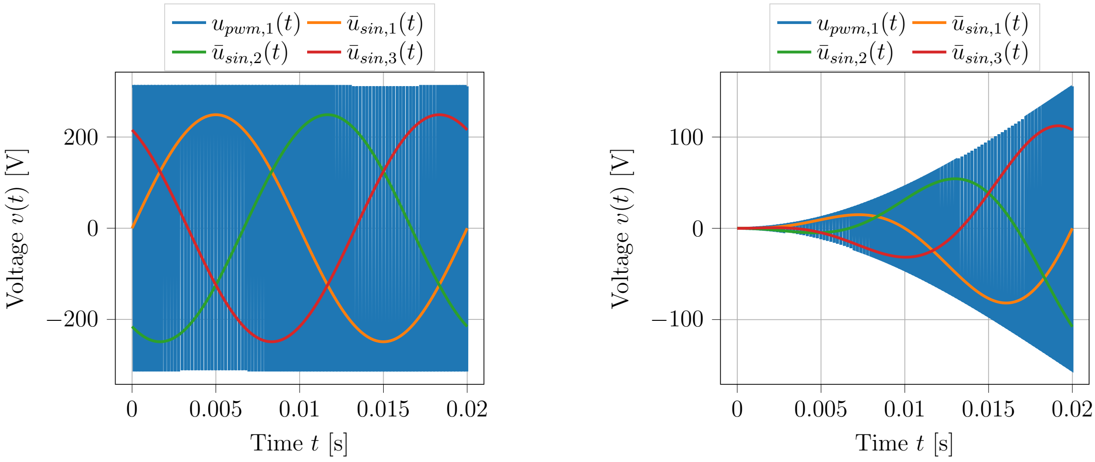
with $f_s = 20\,\mathrm{kHz}$. The peak voltage $\hat{V}$ is chosen to be 311.1 V for (7.1)-(7.4). The model provides a setting with linear material properties, i. e., in particular the reluctivity is modeled linearly, and a setting with nonlinear material properties. In the setting with nonlinear material properties the equations (7.1)-(7.4) are multiplied by a time-dependent factor

$$z\left(t\right) = \begin{cases} 0.5\left(1 - \cos\left(\pi\ \frac{t}{2}\right)\right), & t \in (0, 0.04], \\ 1, & \text{else}, \end{cases} \tag{7.5}$$

which reduces the initial transient behavior of the motor and allows to reach steady state faster [48]. Figure 7.2 shows the reference signal $\bar{v}_{sin,j}, j = 1,2,3$ and one resulting PWM voltage source $v_{pwm,1}(t)$ for a switching frequency of $f_\mathrm{s} = 5\,\mathrm{Hz}$ in the time interval $[0, 0.02]$. Figure 7.2(a) shows the four signals without applying the time-dependent factor and Figure 7.2(b) with applying the time-dependent factor (7.5).

The time integration is performed using the external library GetDP [27, 43], which implements the implicit Euler method and Newton's method with damping. In addition, GetDP computes several quantities of the machine after each application of the time integrator, e. g., the Joule losses (3.29) and the torque (3.27).

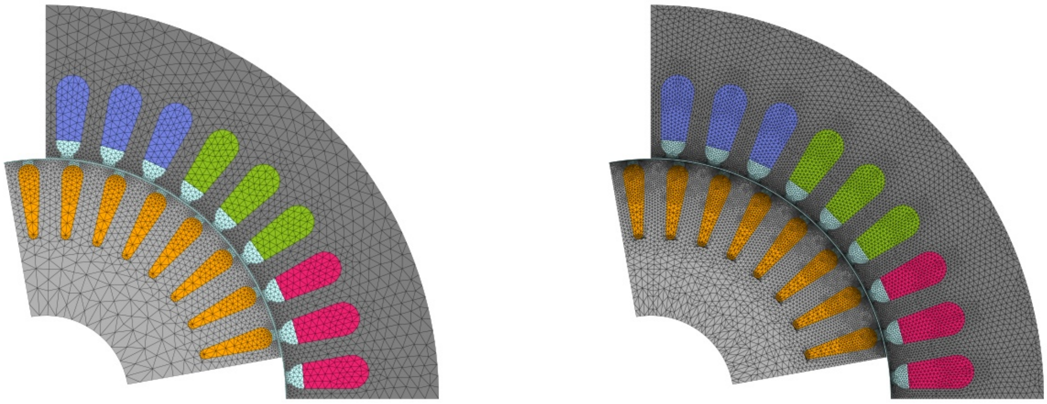
Gmsh [44, 45] is used to generate grid representations of the model. We consider a hierarchy of three nested spatial grids. The coarsest grid Ω_3 with $d_a = 4449$ degrees of freedom is represented in Figure 7.3(a). Two more precise discretizations are defined by the grids Ω_2 and Ω_1, which are obtained by refining the coarsest grid Ω_3 and yield grids with d_a = 17,496 and d_a = 69,384 degrees of freedom, respectively. The grid view of the intermediate spatial grid Ω_2 is shown in Figure 7.3(b).



(a) No relaxation.

(b) Relaxation.

Figure 7.2: Visualization of one phase of the PWM voltage source (7.2) and the three phases of the sinusoidal reference signal (7.3) in the time interval $[0, 0.02]$. In the right plot, the time-dependent factor (7.5) is applied.



(a) Grid view of Ω_3.

(b) Grid view of Ω_2.

Figure 7.3: Two of the three spatial discretizations generated by `Gmsh`. The finer grid Ω_2 shown in the right subplot is generated by refining the coarse grid Ω_3 depicted in the left subplot.

7.2 PWM voltage source

In the following, we present the numerical results of applying the MGRIT algorithm to the 2D induction machine model “im_3_kw” driven by the PWM voltage source (7.2) under no-load condition. In addition to the general convergence behavior and runtime results, we present the effects of spatial coarsening within MGRIT for the problem. For this purpose, we divide the numerical results into two sections: First, we investigate different coarsening strategies for spatial coarsening in Section 7.2.1 based on the model with linear material properties to save computational resources. Then, we apply the MGRIT algorithm based on the results to the induction motor model with nonlinear material properties.

For the transfer between the spatial grids, we use standard finite element interpolation or nodal interpolation. To avoid numerical instabilities at the boundaries between the stator and the rotor, we apply standard finite element interpolation for both regions separately. Recall from Section 3.4 that only the magnetic vector potential is a function that depends on space, and therefore only the magnetic vector potential is discretized on the different spatial meshes, and the other unknows are functions that are independent of space. Therefore, we define the spatial interpolation and restriction operators as block operators with two blocks as follows: We use standard finite element interpolation to interpolate the grid-dependent unknowns while injecting the grid-independent unknowns. Note that the resulting spatial transfer operators need to be computed only once.

For the MGRIT algorithm, we consider two- and five-level MGRIT variants and use both V- and F-cycles with different relaxation schemes. For the two-level variants, we use a coarsening strategy such that there are as many time steps at the coarse level as there are processes. In the multilevel case, we choose a non-uniform coarsening strategy. At the fine level we choose the same coarsening factor as in the two-level setting and then coarsening factor $m^{(\ell)} = 4, \ell = 1, 2, 3$ at the other levels. Furthermore, we consider and compare the direct and delayed spatial coarsening strategy presented in Section 4.3.2. In addition, the nested iteration strategy is used to generate an improved initial guess. For this prediction step, we use the reference signal (7.3) as a voltage source, based on the idea in [41]. For the iterations of the MGRIT algorithm, the discontinuous signal (7.2) is used at each grid level.

To resolve the pulses of the PWM voltage source, very fine time steps are required. More precisely, we choose a time step size of $\Delta t = 2^{-20}$ for the following numerical experiments. In Section 7.2.1, we consider the space-time domain $\Omega_1 \times [0, 0.03125]$, resulting in 2^{15} time steps. Moreover, we consider the space-time residual, measured in the discrete L_2-norm, as a convergence criterion for the MGRIT algorithm and choose a convergence tolerance of 10^{-4}. In Section 7.2.2 we consider 10,752 time steps, resulting in a final time $T \approx 0.01$, and use the spatial grid Ω_2 as fine spatial grid. Moreover, we use a less stringent convergence criterion, and the MGRIT algorithm stops when the relative difference of two successive iterates at the C-points of the finest level is smaller than 10^{-2}, i. e., when the maximum relative change in Joule losses for all C-points is smaller than 1%. Note that the considered time intervals are not sufficient to reach the steady-state of the machine.

7.2.1 Linear material model

Table 7.1 shows the number of iterations, setup time, solution time, and total time in seconds for the different MGRIT variants without spatial coarsening. The setup time consists of setting up the algorithm and generating an improved initial guess through nested iterations. Note that the same relaxation scheme is used for nested iterations as for the algorithm itself, i. e., the setup for the five-level F-cycle with F-relaxation is faster than that for the five-level F-cycle with FCF-relaxation. The two-level algorithm with F-relaxation reaches the desired tolerance after 12 MGRIT iterations. While FCF-relaxation is beneficial in the two-level setting, reducing both the number of iterations and the runtime, F-relaxation performs better than FCF-relaxation for F-cycles. This is due to the fact that in the two-level variant the stronger relaxation is applied only to the finest level, while in the multi-level variant the stronger relaxation is applied to multiple levels, which increases the amount of work within the multi-level variant more than in the two-level variant. However, the gain of the additional work in terms of the number of iterations required is the same in both cases. Comparing the total runtimes, the multilevel variants are faster than the two-level methods, with the five-level V-cycle being the fastest.

L	Cycle	Relax.	m	# Iters	Setup time	Solve time	Total time
2	V	F	(256)	12	9,523 s	166,265 s	175,788 s
2	V	FCF	(256)	7	9,515 s	125,149 s	134,664 s
5	V	FCF	(256,4,4,4)	8	3,293 s	77,146 s	80,439 s
5	F	F	(256,4,4,4)	12	2,396 s	85,117 s	87,513 s
5	F	FCF	(256,4,4,4)	7	3,291 s	87,660 s	90,951 s

Table 7.1: Results for the linear induction machine discretized on a space-time grid of size 69,384 × 2^{15} in terms of iterations, setup, and solve time on 256 processes for the different MGRIT variants without spatial coarsening.

Table 7.2 shows the effects of adding spatial coarsening to the MGRIT variants considered in Table 7.1. Note that for the five-level MGRIT variants, direct spatial coarsening means that Ω_1 is used at the finest level, Ω_2 at the second level, and Ω_3 at all remaining coarse levels. In delayed spatial coarsening, Ω_3 is used only at the coarsest level, Ω_2 at the second coarsest level, and Ω_1 at the three finest levels. In the two-level methods, Ω_2 or Ω_3 is used on the coarse grid. All variants with FCF-relaxation converge to the solution in the same number of iterations as the variants without spatial coarsening. Specifically for this problem, direct and delayed spatial coarsening do not differ in terms of iterations required. The situation is different for all variants with F-relaxation. Neither the two-level method with spatial coarsening converges to the desired tolerance in significantly fewer than $N_t^{(0)}/m$ iterations, nor does the five-level F-cycle.

L	Cycle	Relax.	m	SC strategy	# Iters	Total time	Speedup w.r.t. no SC
2	V	F	(256)	direct (Ω_2)	✗	-	-
2	V	FCF	(256)	direct (Ω_2)	7	76,157 s	1.77
2	V	FCF	(256)	direct (Ω_3)	7	49,551 s	2.72
5	V	FCF	(256,4,4,4)	delayed	8	74,686 s	1.08
				direct	8	66,549 s	1.21
5	F	F	(256,4,4,4)	delayed	✗	-	-
				direct	✗	-	-
5	F	FCF	(256,4,4,4)	delayed	7	75,881 s	1.20
				direct	7	60,689 s	1.50

Table 7.2: Results similar to those of Table 7.1 but with spatial coarsening; ✗ indicates no convergence to the desired tolerance in less than $N_t^{(0)}/m = 256$ iterations and speedup is measured relative to the same MGRIT variant without spatial coarsening.

To better understand the degradation of convergence when using F-relaxation, Figure 7.4 details the convergence behavior for all F-cycle variants considered in Table 7.2. Shown are the space-time residual norms over the first ten MGRIT iterations for all F-cycle variants. The dashed lines show the results of direct spatial coarsening and the solid lines show the application of the delayed approach. The FCF-relaxation variants show linear convergence behavior, and there is virtually no difference between the two spatial coarsening strategies. In contrast, the convergence of MGRIT with F-relaxation stagnates after five iterations. This effect most likely comes from the fact that certain information is not sufficiently transferred between the spatial grids. The overlap effect of the FCF-relaxation automatically fixes this insufficient transfer between spatial grids.

In terms of runtime, the results show that using a coarser spatial grid on the coarse levels can lead to a significant reduction in runtime. As expected, the more aggressive the spatial coarsening strategy and the more work on the coarse grids, the higher the gain of spatial coarsening. As a consequence, the two-level method benefits the most from spatial coarsening in these experiments.

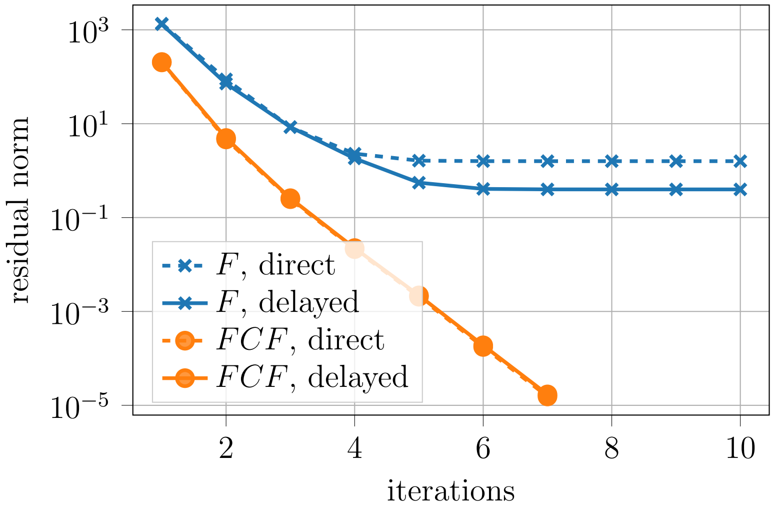


Figure 7.4: Convergence results for the different five-level F-cycle variants with direct (dashed lines) and delayed (solid lines) spatial coarsening.

7.2.2 Nonlinear material model

Based on the experiments in the previous chapter, we apply only the direct coarsening strategy in the following. Moreover, we consider (with one exception) only variants with FCF-relaxation, i. e., variants where the MGRIT algorithm converges within a reasonable number of iterations in the linear setting.

Table 7.3 shows iteration counts and runtimes of the different MGRIT variants, with timings split up again into setup and solve times. No results are given for the two-level method with FCF-relaxation and with spatial coarsening, since during the F-relaxation step of the first MGRIT iteration, the Newton solver failed to converge for at least one time step and, thus, the algorithm cannot be applied in this setting. Consequently, in the two-level case we consider only the version without spatial coarsening and there the version with only F-relaxation. Looking at the total runtimes of the different variants without spatial coarsening, the five-level V-cycle algorithm is fastest, followed by the five-level F-cycle variant which is about a factor of 1.5 times slower than considering V-cycles. Furthermore, five-level F-cycles are already about twice as fast as the two-level method. Adding spatial coarsening in the multilevel schemes, we can benefit over the two-level algorithm even more. Considering five-level V- and F-cycles with spatial coarsening, the factor in comparison with the runtime of the two-level method can be increased from 1.9 or 2.8 when applying F- or V-cycles without spatial coarsening, respectively, to a factor of about 3.8 or 4.4 when spatial coarsening is added.

Figure 7.5 shows total runtimes of the five convergent MGRIT variants considered

L	Cycle	Relax.	m	SC	# Iters	Setup time	Solve time	Total time
2	V	F	(256)	no	4	15,054 s	74,150 s	89,204 s
2	V	FCF	(256)	yes	✗	-	-	-
5	V	FCF	$(256, 4, 4, 4)$	no	3	5,870 s	25,509 s	31,379 s
				yes	3	1,297 s	18,763 s	20,060 s
5	F	FCF	$(256, 4, 4, 4)$	no	3	5,861 s	41,138 s	46,999 s
				yes	3	1,301 s	22,430 s	23,731 s

Table 7.3: Number of iterations, setup, solve and total time on 256 processes of various MGRIT variants applied to the full nonlinear model of the induction machine "im_3_kw" discretized on a space-time grid of size 17,496 × 10,753. ✗ indicates no convergence of at least one nonlinear spatial solve within `GetDP`.

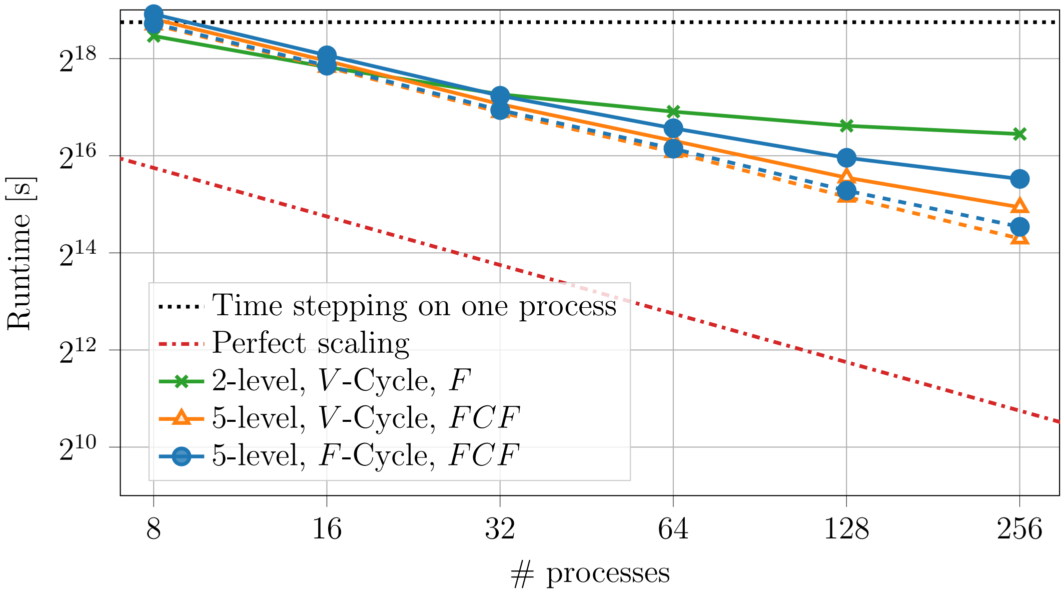


Figure 7.5: Total time-to-solution for the nonlinear induction machine "im_3_kw" using different MGRIT variants and sequential time stepping. Solid lines are runtimes without spatial coarsening, and dashed lines represent results with spatial coarsening. The dotted line shows the runtime of time stepping on one process, and the dashed dotted line indicates perfect scaling.

in Table 7.3 as a function of the number of processes, as well as the time-to-solution of the sequential block forward solve for reference purposes. For the multilevel variants, runtimes are shown for using spatial coarsening (dashed lines) and without spatial coarsening (solid lines). The runtime of all multilevel variants using eight processes as well as the time-to-solution of sequential time stepping is about five to six days, while using eight-way parallelism in the two-level method results in a runtime of only about four days. However, the multilevel variants show better strong parallel scaling and, thus, lead to faster time-to-solution at large process counts than the two-level method. Increasing the number of processes to 256, the total runtime can be reduced to about 8.72 hours or 5.57 hours, when considering MGRIT V-cycles without or with spatial coarsening, respectively. While F-cycles with spatial coarsening allow for a similar reduction, F-cycles without spatial coarsening using 256-way parallelism already take about half a day. The two-level variant reduces the runtime to about one day.

Table 7.4 details speedups and parallel efficiencies for the MGRIT variants. The speedup is computed relative to sequential time stepping, representing the algorithm with the minimum runtime on one process, i. e., the speedup using N_P processes is given by $S_{N_P} = T_{\text{TS}}/T_{\text{MGRIT}}(N_P)$. The parallel efficiency is measured as S_{N_P}/N_P for N_P processes. Using multilevel variants, we can benefit more over sequential time stepping at larger process counts than with the two-level variant. For example, while on 16 processes the speedup is similar for all methods, using 32 processes results in a speedup of up to a factor of 3.6 for a five-level variant, whereas the two-level method yields only a speedup of a factor of 2.8. Increasing the number of processes to 256, with the two-level method we achieve a speedup of a factor of about 5. In contrast, multilevel variants at least double the speedup factor. Considering spatial coarsening leads to the best speedup with the V-cycle algorithm being nearly 22 times faster than the sequential time stepping method. Moreover, the use of spatial coarsening in MGRIT F-cycle is crucial for improving the parallel scalability for more than 64 processes, since this allows reducing computations on coarse levels. While speedups and efficiencies degrade for F-cycles without spatial coarsening, the degredation is only modest when adding spatial coarsening. Note that we bind the temporal coarsening strategy to the maximum number of available processes and, thus, faster runtimes on eight to 128 processes may be possible by adjusting the temporal coarsening strategy to the number of processes.

7.3 Sinusoidal voltage source

In the following, we consider numerical results for the 2D induction machine with constant speed $\omega_{\text{mech}} = 1{,}420\,\text{rpm}$ and nonlinear material properties, using the sinusoidal voltage source (7.1) to cover a larger time interval. More specifically, we consider a time grid with $N_t = 16{,}384$ time points and a time step size $\Delta t \approx 1.2 \cdot 10^{-5}$ s, leading to a final time $T = 0.2$ s. In the spatial dimension, we consider

N_P	8	16	32	64	126	256
two-level cycle with F-relax.						
Speedup Efficiency	1.22 15.22%	1.90 11.85%	2.80 8.74%	3.59 5.60%	4.39 3.43%	4.93 1.93%
five-level V-cycle with FCF-relax.						
Speedup Efficiency	0.96 11.94%	1.74 10.86%	3.22 10.07%	5.43 8.48%	9.19 7.18%	14.02 5.48%
five-level F-cycle with FCF-relax.						
Speedup Efficiency	0.89 11.14%	1.60 10.02%	2.86 8.93%	4.54 7.10%	6.92 5.41%	9.36 3.66%
five-level V-cycle with FCF-relax. and SC						
Speedup Efficiency	1.04 12.96%	1.90 11.86%	3.62 11.31%	6.40 9.99%	12.01 9.44%	21.93 8.57%
five-level F-cycle with FCF-relax. and SC						
Speedup Efficiency	1.02 12.74%	1.86 11.63%	3.51 10.96%	6.08 9.50%	11.06 8.64%	18.54 7.24%

Table 7.4: Speedup and efficiency of different MGRIT variants using various number of processes. The speedup is given relative to the time-to-solution for the sequential time stepping on one process. Parallel efficiency is measured as S_{N_P}/N_P, where S_{N_P} is the speedup for N_P processes.

the grid Ω_3. Within this setting, the steady state is reached at the electric period $q = 10$ up to the tolerance of $< 5 \cdot 10^{-4}$ in terms of the relative error

$$\epsilon(q-1) = \frac{\left|T^{(q)}_{\text{EM,avg}} - T^{(q-1)}_{\text{EM,avg}}\right|}{\left|T^{(q)}_{\text{EM,avg}}\right|}, \quad \text{with} \quad T^{(q)}_{\text{EM,avg}} = \frac{1}{N_q} \sum_{i=1+(q-1)N_q}^{qN_q} T_{\text{EM}}(t_i)$$

denoting the average torque at the period q and $N_q = \lfloor N_t^{(0)}/q \rfloor$ being the number of time steps per period.

As seen in Section 7.2.2, a potential problem is that a nonlinear solve within `GetDP` does not converge due to a poor initial guess for the Newton method. When considering the larger time interval, this is also a problem; more specifically, when simulating the electric machine in the time-parallel setting, using too large time steps at the coarse level(s) may cause at least one nonlinear solution within `GetDP` not to converge. As a result, the time step size at the coarsest level cannot be chosen arbitrarily. Moreover, if we consider a coarsening strategy formed based on the available resources, the use of multilevel variants is limited. Therefore, we mainly use two-level methods in the following. If more resources were available, possibly more intermediate level would provide a further runtime advantage. Again, we use a coarsening strategy such that we have as many coarse grid points as processes. We terminate the method once the maximum change in Joule losses between two successive iterations is less than 1%.

7.3.1 AT-MGRIT

In the following, we present results for a two-level MGRIT variant and several two-level AT-MGRIT variants. For all experiments, we use an improved initial guess given by a global coarse-grid solve. For the following experiments 64 processes are used, which results in a coarsening factor of $m = 256$ for the MGRIT and AT-MGRIT variants. However, the resulting time step size on the coarse level results in at least one nonlinear solve to fail. To overcome this problem, we apply subcycling at the coarse level, i. e., we apply three smaller steps per time step at the coarse level, reducing the time step size and improving the accuracy of the solution.

Method	w	# Iters	Total time (Setup + Solve)	Speedup w.r.t MGRIT	Speedup w.r.t time stepping on one process
Two-level MGRIT	-	5	40,544 s	-	4.64
Two-level AT-MGRIT	12	8	39,480 s	1.03	4.77
	14	7	36,188 s	1.12	5.2
	16	6	32,710 s	1.24	5.75
	18	6	33,337 s	1.22	5.64
	20	6	33,996 s	1.19	5.53
	22	6	34,626 s	1.17	5.43
	24	5	30,582 s	1.33	6.15

Table 7.5: Iteration counts and total runtimes on 64 processes of two-level MGRIT and various two-level AT-MGRIT variants with a coarsening factor of 256 for the simulation of the induction machine.

Table 7.5 shows the number of iterations, total runtimes, and the speedup compared to sequential time stepping on one process for different AT-MGRIT variants and one MGRIT variant. Furthermore, the speedup compared to MGRIT is shown for all AT-MGRIT variants. Comparing the number of iterations, the two-level MGRIT and AT-MGRIT algorithm with $w = 24$ both require five iterations to convergence. For $16 \geq k \geq 22$, six iterations are needed to reach the convergence criterion, and for $w = 14$ and $w = 16$, seven and eight iterations, respectively. Despite the increased number of iterations for some variants, the total runtime of all AT-MGRIT variants is smaller than that of MGRIT, with the largest speedup by a factor of approximately 1.33. Note that the time for the setup phase is the same for all variants and is about 2,891 s. Note also that both algorithms treat the fine level identically, and the improvement comes only from using local coarse grids instead of one global coarse grid. For comparison, the simulation time using serial time stepping on one process is 188,123 s, which is more than two days. The fastest AT-MGRIT variant needs less than nine hours, which corresponds to a speedup of a factor of 6.15.

7.3.2 PinT optimization

In the following we incorporate the AT-MGRIT algorithm into a derivative-free constrained optimization procedure. More precisely, we consider the height $\hat{h}$ and the width $\hat{w}$ of the rotor bars as optimization variables and parameterize the domain $\Omega = \Omega(\hat{p})$ with these two parameters $\hat{p} = (\hat{h}, \hat{w})$. Our goal is to find the optimal width and height of the rotor bars, such that our objective function J is minimal under the constraint that the design variables lie in a set of admissible designs D_{ad}. Additionally, we require that the state equations (3.30)-(3.35) are fulfilled. As objective function we consider:

$$\min_{\mathbf{A}_3, \hat{p}} J(\mathbf{A}_3, \hat{p}) := -\frac{P_{\text{out}}(\mathbf{A}_3, \hat{p})}{P_{\text{in}}(\mathbf{A}_3, \hat{p})}, \tag{7.6}$$

with

$$\begin{aligned} P_{\text{out}}(\mathbf{A}_3, \hat{p}) &= \int\limits_{(q-1)T_q}^{qT_q} P_{\text{mech}}(\mathbf{A}_3, \hat{p}) \mathrm{d}t, \\ P_{\text{in}}(\mathbf{A}_3, \hat{p}) &= \int\limits_{(q-1)T_q}^{qT_q} \big[P_{\text{mech}}(\mathbf{A}_3, \hat{p}) + P_{\text{loss}}(\mathbf{A}_3, \hat{p})\big] \mathrm{d}t, \end{aligned}$$

where $q \in \mathbb{N}$ represents the period at which the steady state is reached and $T_q > 0$ is the length of the period. The objective J can be seen as a negative measure of efficiency, as it is given by the quotient of the output and the input power on the right-hand side in (7.6). Since P_{mech} and P_{loss} involve integrals over the parametrized domain (see (3.27)-(3.29)), they depend on the design $\hat{p}$. They both depend on the solution $\mathbf{A}_3$ of the state equation, which depends on the design $\hat{p}$ itself. In the optimization, we know that for every admissible design $\hat{p}$, there is a unique solution to our state equation, which we call $\mathbf{A}_3(\hat{p})$ and consider the reduced problem

$$\min_{\hat{p}} \hat{J}(\hat{p}) := J(\mathbf{A}_3(\hat{p}), \hat{p}) \quad \text{subject to} \quad \hat{p} \in D_{\text{ad}}, \tag{7.8}$$

which does not involve the state equation as a constraint anymore and where

$$D_{\text{ad}} := \{\hat{p} = (\hat{h}, \hat{w}) \in \mathbb{R}^2 \colon \hat{h}_{\text{l}} \leq \hat{h} \leq \hat{h}_{\text{u}},\ \hat{w}_{\text{l}} \leq \hat{w} \leq \hat{w}_{\text{u}}\}, \tag{7.9}$$

with $\hat{h}_{\text{l}}, \hat{h}_{\text{u}}, \hat{w}_{\text{l}}, \hat{w}_{\text{u}} \in \mathbb{R}$.

To solve the optimization problem, we use the derivative-free optimization algorithm Py-BOBYQA [19], which is a Python implementation of BOBYQA [87]. The idea of this algorithm is to use a model for the objective function and improve the model in every iteration to make it approximate the minimum of the objective function sufficiently accurately. Specifically, the algorithm solves the optimization problem using a trust region method that forms a quadratic interpolation model with $\hat{m} = 2\hat{n} + 1$ degrees of freedom. For a detailed description, we refer to [54, 87].

Based on the steady-state behavior of the induction motor, our goal is to optimize the size of the rotor bars, in particularly, their width and height using the objective function (7.8). By describing the geometry of the rotor bars with $\hat{n} = 2$ design variables, we have to simulate the behavior of the induction machine for $\hat{m} = 5$ different designs to compute an initial interpolation model and once in every optimization iteration. We set

$$\hat{h}_{\text{l}} = 0.007, \quad \hat{h}_{\text{u}} = 0.015, \quad \hat{w}_{\text{l}} = 0.0015, \quad \hat{w}_{\text{u}} = 0.0035$$

as the admissible design bounds in (7.9), choose $\hat{h}^{(0)} = 0.01425$ and $\hat{w}^{(0)} = 0.002$ as an initial design depicted in Figure 7.3 (left) and set the initial trust-region radius to 10^{-4}. The optimization algorithm terminates when the trust region is smaller than our chosen tolerance of 10^{-6}. For each objective function evaluation, we generate a mesh representation of the current geometry using `Gmsh`, where each mesh, depending on the geometry, consists of approximately 4,500 degrees of freedom. Then, a two-level AT-MGRIT with $m = 64$, $w = 100$, and F-relaxation is used to solve the problem.

The implementation uses a master/worker strategy for the optimization and the simulations, using one process for the optimization and 256 processes for each simulation, where all resources are used for temporal parallelization.

Optimization results

Figure 7.6 shows the negative values of the objective function evaluated during the optimization procedure on the left and an overview of the geometries considered

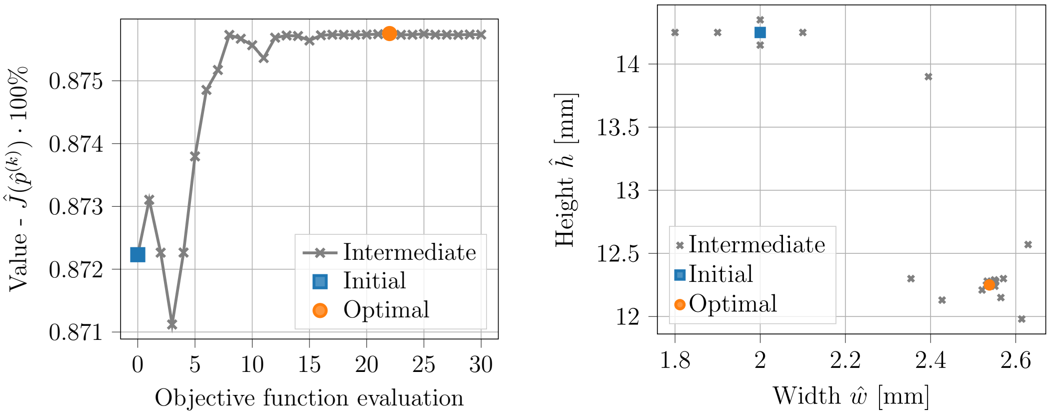


Figure 7.6: Optimization process over iterations: negative objective function values (left) and geometries (right).

on the right. In the 26 optimization iterations a total of 31 function evaluations were required: one evaluation in every iteration and an additional five in the first iteration for building the initial quadratic interpolation model. The optimal geometry $\bar{p}$ with $\bar{w} = 0.00254$ and $\bar{h} = 0.01226$ was found in the 18th iteration. In the following iterations, no better design was found, so the trust region was gradually reduced until it fell below the chosen termination criterion. Figure 7.7 shows the two geometries for the initial design (left) and the optimal design (right). Different structures of the geometry of the rotor bars (orange) can be seen: the bars in the optimal design are less high and significantly wider than those in the initial design. Overall, the optimal design increases the negative of the objective function and thus the efficiency of the electrical machine from 87.22% to 87.57%. Figure 7.8 gives a detailed comparison of the torque (left) and the Joule losses (right) between the initial and the optimal design in the steady state. Compared to the original design, both values were increased, with an average increase of 16.25% in torque and 20.03% in Joule losses for the optimal design.

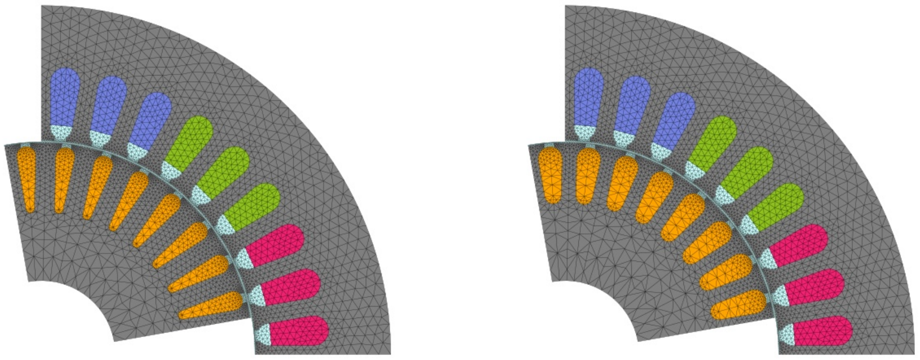


Figure 7.7: Grid view of the initial design (left) and the optimal design (right).

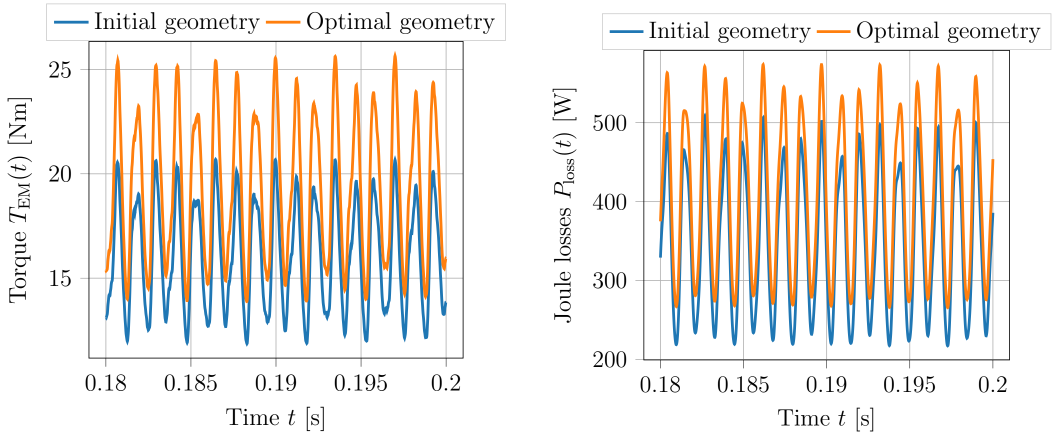


Figure 7.8: Torque (left) and Joule losses (right) for the initial and the optimal designs of a nonlinear “im_3_kw” induction machine model.

In each step of the optimization algorithm we use the AT-MGRIT algorithm to simulate the corresponding geometry. To determine the effect of the time-parallel method compared to sequential time stepping for solving the problem, we count the calls of the time integrator for both methods. In each iteration of the optimization algorithm, we achieve a theoretical speedup of up to 11.67 by using AT-MGRIT compared to the standard sequential time stepping. Note that for this application, counting the serial solves of the AT-MGRIT algorithm is a reasonable measure to determine the theoretical speedup, since the computational cost dominates the runtime cost of the algorithm and the communication cost is negligible compared to the computational cost.

Chapter 8

Performance Model

In the last two chapters, various numerical results were presented using different parameter settings for the MGRIT algorithm based on personal experience. It could be observed that for the MGRIT algorithm, the parameter space of the parameters to be chosen is very large and, moreover, the correct choice of the parameters has a great influence on both the convergence and the parallelizability of the method. Both factors have a significant impact on the runtime of the method, and ultimately reducing the runtime using parallel resources is the main goal of PinT methods. Here, we present an approach that can simplify the choice of parameters, more specifically, we describe a performance model [12] that can be used for different PinT methods and is able to cover the larger parameter space of the methods to predict the runtime for a given method and setting. In this way, parameter choices can be predicted theoretically without the need for time-consuming and costly parallel simulations. In the performance analysis, we focus on Parareal, PFASST, and MGRIT.

Various performance models exist for the different algorithms. In [5], several task-based implementations of the Parareal method are presented and performance models for the different implementations are established. The performance of the most common pipeline-based implementation of the Parareal algorithm was further investigated in [76, 94] and elsewhere. An event-based implementation with brief performance analysis was presented in [10]. The performance of the two-level PFASST algorithm was studied in [29, 76] and an extension for the multilevel case in [80]. For the MGRIT algorithm, there is a performance model based on the `XBRAID` framework [72] in [37] that uses the model to select the “best” configuration for the distribution of spatial and temporal parallelization. For the `PETSc` implementation of the MGRIT method, the performance of the linear MGRIT algorithm was investigated in [75] using a matrix-based model. However, all of these models rely on an explicit implementation of the algorithms. These implementations are very effectively chosen, but they only reflect the potential of the implementation and not the algorithm.

Algorithm 8.1: FAS MGRIT data driven formulation

```
foreach $i \in \{$C-points on level 0$\}$ do                          ▷ Initialize points
    $\mathbf{u}_i^0 \leftarrow \mathbf{0}$
    $\mathbf{g}_i^0 \leftarrow \mathbf{0}$
$\mathbf{u}_0^0 \leftarrow \mathbf{u}_0$                                  ▷ Set initial condition
for $k \leftarrow 1$ to $K+1$ do
    for $\ell \leftarrow$ 0 to L-2 do
        foreach $F$-interval on level $\ell$ do                        ▷ $F$-relax
            for $i \leftarrow$ first interval point to last interval point do
                $\mathbf{u}_i^{(\ell)} \leftarrow \mathbf{\Phi}(\mathbf{u}_{i-1}^{(\ell)}, T_{i-1}^{(\ell)}, T_i^{(\ell)}) + \mathbf{g}_i^{(\ell)}$
        repeat $\nu^{(\ell)}$ times
            foreach $i \in \{$C-points on level $\ell\} \setminus \{0\}$ do   ▷ $C$-relax
                $\mathbf{u}_i^{(\ell)} \leftarrow \mathbf{\Phi}(\mathbf{u}_{i-1}^{(\ell)}, T_{i-1}^{(\ell)}, T_i^{(\ell)}) + \mathbf{g}_i^{(\ell)}$
            foreach $F$-interval on level $\ell$ do                    ▷ $F$-relax
                for $i \leftarrow$ first interval point to last interval point do
                    $\mathbf{u}_i^{(\ell)} \leftarrow \mathbf{\Phi}(\mathbf{u}_{i-1}^{(\ell)}, T_{i-1}^{(\ell)}, T_i^{(\ell)}) + \mathbf{g}_i^{(\ell)}$
        foreach $i \in \{$C-points on level $\ell\}$ do               ▷ Spatial transfer
            $\mathbf{u}_{\varsigma^{(\ell)}(i)}^{(\ell+1)} \leftarrow R_s^{(\ell)}(\mathbf{u}_i^{(\ell)})$
            $\mathbf{v}_{\varsigma^{(\ell)}(i)}^{(\ell+1)} \leftarrow R_s^{(\ell)}(\mathbf{u}_i^{(\ell)})$
        foreach $i \in \{$C-points on level $\ell\} \setminus \{0\}$ do     ▷ FAS rhs
            $\mathbf{r}_i^l \leftarrow \mathbf{g}_i^{(\ell)} + \mathbf{\Phi}(\mathbf{u}_{i-1}^{(\ell)}, T_{i-1}^{(\ell)}, T_i^{(\ell)}) - \mathbf{u}_i^{(\ell)}$
            $\mathbf{g}_{\varsigma^{(\ell)}(i)}^{(\ell+1)} \leftarrow R_s^{(\ell)}(\mathbf{r}_i^l) - \mathbf{\Phi}(\mathbf{u}_{\varsigma^{(\ell)}(i)-1}^{(\ell+1)}, T_{\varsigma^{(\ell)}(i)-1}^{(\ell+1)}, T_{\varsigma^{(\ell)}(i)}^{(\ell+1)}) + \mathbf{u}_i^{(\ell+1)}$
    for $i \leftarrow 1$ to $N_t^{(L-1)}$ do                           ▷ Solve coarse system
        $\mathbf{u}_i^{(L-1)} \leftarrow \mathbf{\Phi}(\mathbf{u}_i^{(L-1)}, T_{i-1}^{(L-1)}, T_i^{(L-1)}) + \mathbf{g}_i^{(L-1)}$
    for $\ell \leftarrow$ L-2 to 0 do
        foreach $i \in \{$C-points on level $\ell\} \setminus \{0\}$ do     ▷ Correct
            $\mathbf{u}_i^{(\ell)} \leftarrow \mathbf{u}_i^{(\ell)} + P_s^{(\ell)}(\mathbf{u}_{\varsigma^{(\ell)}(i)}^{(\ell-1)} - \mathbf{v}_{\varsigma^{(\ell)}(i)}^{(\ell-1)})$
        foreach $F$-interval on level $\ell$ do                        ▷ $F$-relax
            for $i \leftarrow$ first interval point to last interval point do
                $\mathbf{u}_i^{(\ell)} \leftarrow \mathbf{\Phi}(\mathbf{u}_{i-1}^{(\ell)}, T_{i-1}^{(\ell)}, T_i^{(\ell)}) + \mathbf{g}_i^{(\ell)}$
    if convergence criterion is reached then
        break
```

Our performance model is based on task graphs and on a data-driven formulation of the algorithms, which is already given for Parareal in Algorithm 4.1 and for PFASST in Algorithm 4.2. A data-driven formulation of the MGRIT algorithm can be found in Algorithm 8.1. Note that the notation for the time integrator in Algorithm 8.1 is slightly different than in Section 4.3, and additionally expects the respective start and end points as parameters. Also, as in Section 4.3.2, it is assumed that all problem-dependent terms are inside the time integrator. Furthermore, the function $\zeta^{(\ell)} : \mathbb{R}_0^+ \to \mathbb{R}_0^+, \ell = 0, ..., L-2$ in Algorithm 8.1 maps between the indices of the time points of the different levels. For more details we refer to [12]. Similarly, the local and global convergence criteria from Section 4.5 can be formulated in a data-driven formulation [12]. In the following, we first introduce the model and show how a task graph can be created from a data-driven formulation of a PinT method. The prediction is then based on schedulings, i. e., the assignment of tasks to start times, based on typical parallelization strategies of PinT methods. Finally, we compare our predictions of the model with parallel simulation using four different PinT libraries.

8.1 Performance analysis

Our performance analysis of PinT algorithms is based on task graphs created for individual settings of the algorithms. An explicit task graph created in this way can be independently examined to determine a minimum lower bound on the runtime of the setting for the selected algorithm. We can also perform runtime analysis based on the task graph by assigning tasks with a specific starting point to the processes. This procedure is known as scheduling. Instead of solving the scheduling problem, i. e., finding a scheduling with minimal runtime, we present here a scheduling based on a known and typical distribution of time points among processes and implementations of PinT algorithms. This scheduling provides us with a runtime analysis for these typical implementation choices.

A task graph with communication costs included is a directed acyclic graph (DAG) $G = (V, E, \omega, c)$ [92], where $V = \{v_1, ..., v_n\}$ represents the set of tasks and the directed edges $E \subseteq V \times V$ represent the dependencies of the tasks, i. e., an edge (v_i, v_j) means that v_j cannot be executed until v_i is completed. The weighting function $\omega : V \to \mathbb{R}_0^+$ assigns a weight to each task and represents the computational cost incurred by the task. The cost function $c : E \to \mathbb{R}_0^+$ assigns a weight to each edge representing the communication cost between tasks. Let $P = \{p_1, \ldots, p_{N_P}\}$ be the set of N_P available processes and $A : V \to P$ be an allocation function that assigns each task in V to a process. Note that the communication cost function depends on an explicit assignment of tasks to processes, where the cost $c(v_i, v_j)$ for an edge $(v_i, v_j) \in E$ is 0 if $A(v_i) = A(v_j)$ and otherwise represents the communication cost between processes $A(v_i)$ and $A(v_j)$ based on the application and tasks v_i and v_j. A schedule is defined as a function

$S : V \to \mathbb{R}_0^+$ that assigns a starting point, i. e., a point in time when a processor starts executing that task, to each task subject to the following constraints:

1. For all $(v_i, v_j) \in E, S(v_j) \geq S(v_i) + \omega(v_i) + c(v_i, v_j)$
2. For all $v_i, v_j \in V, v_i \neq v_j, A(v_i) = A(v_j) \Rightarrow S(v_i) \geq S(v_j) + \omega(v_j) \vee S(v_j) \geq S(v_i) + \omega(v_i)$.

The first constraint guarantees that the dependencies of the tasks are respected. The second constraint restricts the use of resources and enforces that no process is assigned more than one task at a time. The makespan or runtime of a given allocation and schedule is defined by $\max_{v \in V}(S(v) + \omega(v))$. Setting the communication cost for each edge to zero, the minimum possible makespan for $N_P = \infty$ can be calculated by computing the longest path within a DAG.

In the following, we first describe how the data-driven algorithms and a fixed parameter setting can be used to construct a task graph. We then describe how typical parallelization strategies for PinT methods are implemented. These strategies can be transformed into a method for scheduling tasks, and finally this schedule can be used for runtime prediction.

8.1.1 Creating task graphs from PinT algorithms

We construct task graphs based on the PinT methods Parareal, PFASST, and MGRIT, with the procedure being the same for each of the three algorithms. Starting with an initial task that represents the information for the initial value, we run through the chosen algorithm and add a task at each point where computations occur. More precisely, if the same operation is performed at different time points or at the same time point in different iterations, a task is created for each computation. Also, for each task, we add edges to all tasks on which the computations within the task are based. Note that we do not model temporal parameters, i. e., $\mathcal{G}(u_{i-1}, t_{i-1}, t_i)$ is equivalent to $\mathcal{G}(u_{i-1})$ since these are fixed at the beginning and generally do not change thereafter.

Figure 8.1 shows an example of a DAG for the Parareal Algorithm 4.1 with $N_t = 3$ and without considering a convergence criterion. To keep the example simple, no edge costs are given. Tasks are specified by nodes, where the upper part of a node gives the description of the task and the lower part gives the cost. For the description, we use a left superscript that numbers and individualizes the nodes, and a right subscript that specifies the index of the time point processed by the task. The initial node is denoted by I, the coarse propagator by $\mathcal{G}$, the fine propagator by $\mathcal{F}$, the copy by C, and the correction by $+$. The cost of the fine propagator is chosen as two and all other costs are one. Note that these costs are not representative, but were chosen for illustrative purposes. A discussion of the appropriate weighting of these costs for PinT methods follows in Section 8.1.3.

A closer look at the graph shows the structure of the algorithm. Starting from left to right, first the initial value is copied, then an initial guess is computed based on the copied value, and then, three iterations of the algorithm are performed. Examining the subgraph that models the calculation of the initial guess, it is clear that for each time point the coarse operator is called and then the value is copied, with each call of the propagator depending on the copy of the previous time point. The longest path within the graph and, thus, the minimal achievable parallel runtime is ten and is given by $\{{}^{0}I \to {}^{1}C_0 \to {}^{8}\mathcal{F}_1 \to {}^{12}{+}_1 \to {}^{17}\mathcal{F}_2 \to {}^{20}{+}_2 \to {}^{23}\mathcal{F}_3 \to {}^{25}{+}_3\}$.

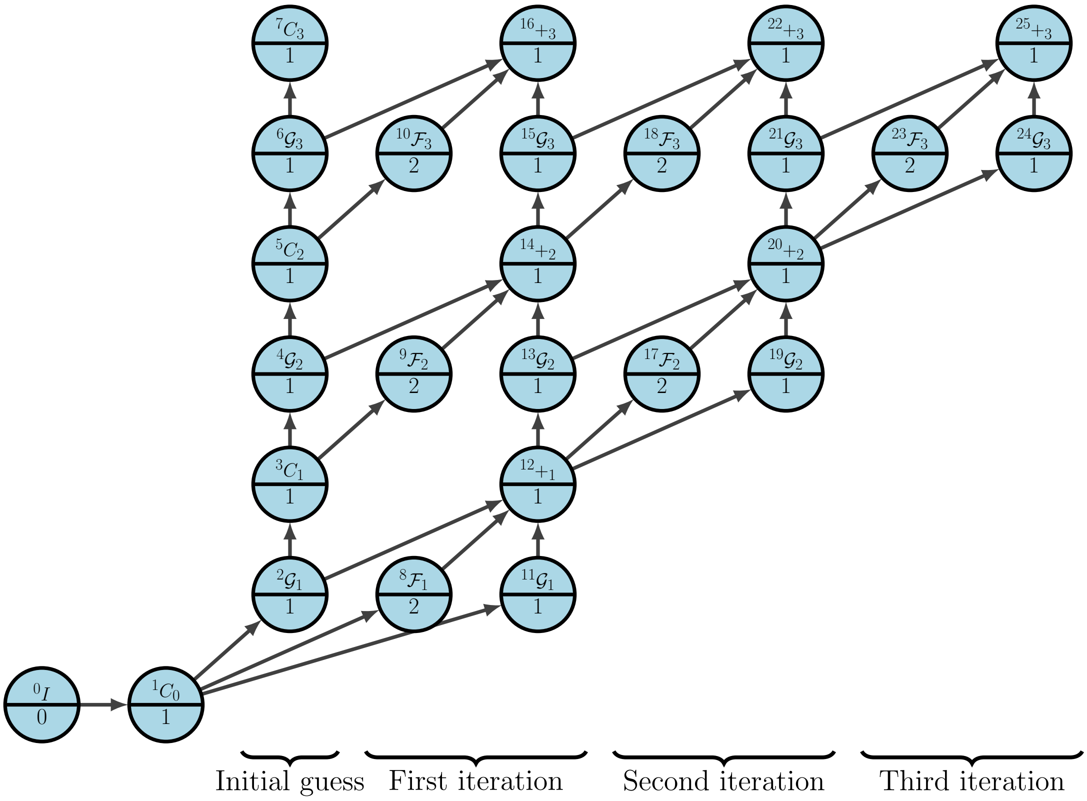


Figure 8.1: Example of the DAG for the Parareal method for $N_t = 3$. The nodes represent the individual tasks and, thus, the computations within the algorithm, with the upper part describing the task and the lower part giving the cost. The edge costs have been omitted for simplicity (the costs are zero everywhere).

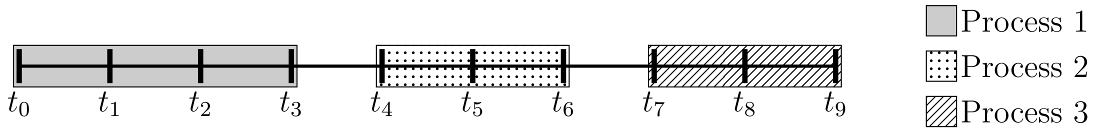


Figure 8.2: Example of a standard distribution of time points to processes in PinT algorithms for $N_t = 9$ and $N_P = 3$.

8.1.2 Adoption of typical PinT scheduling

PinT algorithms typically coordinate temporal processes by assigning time points to processes on a block-by-block basis, with each block consisting of approximately the same number of time points. This exploits the evolutionary nature of the underlying initial value problem, as information is only ever transported forward in time, keeping the number of required communications low. Figure 8.2 shows an example of such a distribution for $N_t = 9$ and $N_P = 3$. Note that the solution is given at time t_0, so each process must propagate the solution to the same number of time points. In multilevel algorithms, typically the distribution of the grid on the finest level also determines the distribution on the coarser grids. Thus, the distribution of time points at each level depends on the distribution at the finest level.

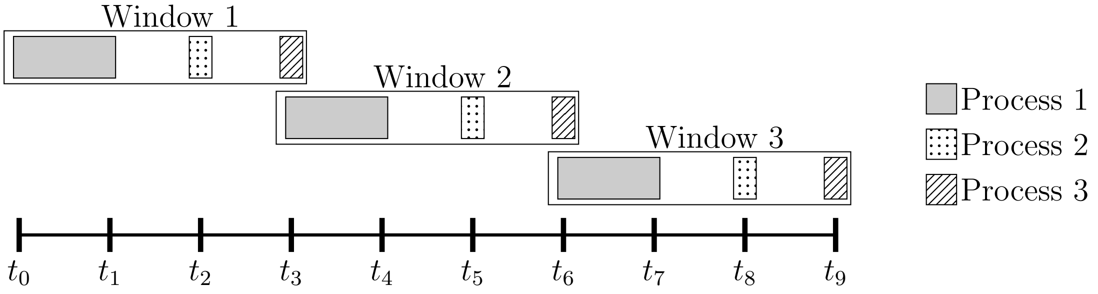


Figure 8.3: Example of the windowing strategy for $N_t = 9$, $N_P = 3$ and three windows. The windows are considered sequentially and the solution of the previous window is used as initial condition for the next window. In each window, the standard distribution is used.

Another approach is the so-called windowing strategy. Here, the entire time interval is first divided into windows, i. e., into time subintervals, which are then processed one after the other. The solution of the previous window always serves as initial condition for the next window. Each window is considered independently and within this window the blockwise distribution described above is used. Figure 8.3 shows an example for $N_t = 9$, $N_P = 3$ and three windows. However, the windowing strategy applies the corresponding PinT method several times in a row, so here, we focus on scheduling for the typical distribution. To adapt this strategy for windowing, the DAGs for each window must be properly connected and the scheduling has to be adjusted slightly.

Based on the common distribution, the following procedure can be used to determine a schedule: We run the chosen algorithm to maintain the order of operations, and assign a process to each task based on the distribution of time points. We also assign an earliest possible start time to the task based on the tasks already scheduled and their weights. If at a certain point within the algorithm there is no execution order for multiple tasks, e. g., in a `foreach`-loop, and multiple tasks are assigned to the same process, the task that satisfies all conditions and has the highest index, i. e., belongs to the latest time point, is always considered first. This strategy ensures that information about that task is available as early as possible. Due to the evolutionary nature of the underlying problem and the blockwise distribution of time points across processes, this strategy allows tasks to be processed first whose subsequent tasks may be on another process. In addition, because the points are distributed in blocks, the prerequisites for this task are often on the same process and the process can start without communication.

Figure 8.4 shows the example of a typical schedule based on this strategy for the Parareal example with $N_t = 3$ from Figure 8.1 and $N_P = 3$ processes. This scheduling for three processes requires a makespan of 19.

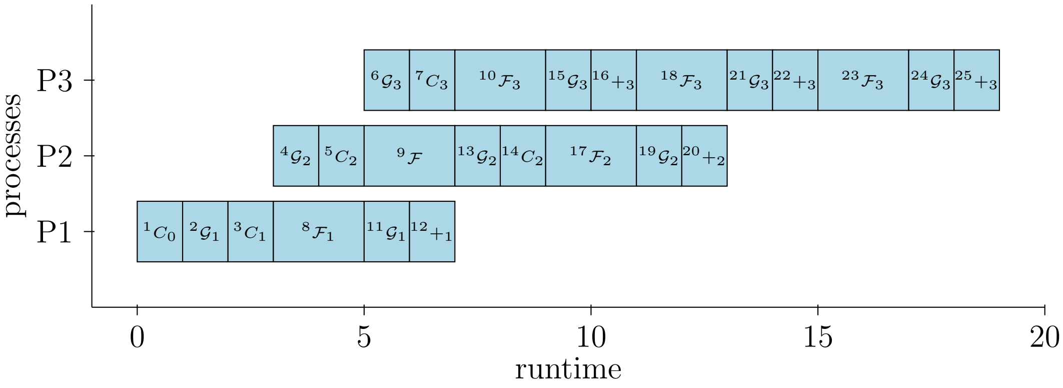


Figure 8.4: Schedule based on the typical distribution of time points across processes for PinT algorithms and for the Parareal example with $N_t = 3$ from Figure 8.1 and $N_P = 3$.

8.1.3 Discussion about appropriate weighting of task

All PinT algorithms presented here consider only the temporal dimension and are capable of solving various problems of the form (4.1), such as the heat equation or the advection equation. Up to this point, the spatial dimension has been considered as a black box, since the algorithms presented here operate relatively independently of the spatial problem. However, if we want to determine the cost of the required operations and communication of a given problem for the performance analysis, we need to consider the spatial problem. Both computation and communication in the temporal dimension depend, at least in part, on the size of the problem, the method used to solve the problem, and the spatial parallelization (including spatial communication). Since the model is based precisely on these costs and only works if these quantities can be predicted well, it is important to discuss how these costs can be determined.

There are generally two ways to determine the required costs for the operations and communication within the algorithms: The first way is to use a theoretical model to calculate the cost and the second way is to measure the runtimes. The advantage of a theoretical model is that one sets up an exact analysis for a given problem and, thus, has a model that is completely independent of actual calculations. Depending on the size, method of solving the problem, parallelization in space, etc., one can set up a very accurate estimate and even test the real implementation against the model. Therefore, the big disadvantage is also obvious: The effort needed to build the model and determine the cost is high. And this immense effort has to be made for every single problem, possibly even for different parameter settings of the same problem.

The other option that requires measuring of runtimes has the disadvantage that one has to perform the measurements for each type of operation within the algorithms. However, the algorithms presented here consist of only a few types of operations, and the runtime of some operations, such as a simple copy of data or an addition, do not necessarily need to be measured as well, since the runtime is very small and negligible. If we consider the Parareal Algorithm 4.1 as an example, we only need to determine the runtime of the fine propagator $\mathcal{F}$ and the coarse propagator $\mathcal{G}$. We assume that the runtime for an operator is the same for each time step, which means that we only need to make one measurement per type of operation, i. e., two time measurements for the Parareal algorithm. Note that this is generally not true for nonlinear problems, where the cost of each time step can vary significantly. Theoretically, the model can handle this varying cost per solution, although it makes it much more difficult to obtain the cost per task. For nonlinear problems, therefore, performance analysis is much more difficult and a challenge for future work. Moreover, the use of runtimes allows us to continue to consider the underlying spatial problem as a black box and it avoids many details such as implementation, spatial communication, etc., making it possible to cover a large number of different problems without much effort. Nevertheless, it is possible to make predictions for different distributions of spatial and temporal parallelization, since spatial parallelization is implicitly captured in the measurement. For example, in [37], a least squares problem based on measured runtimes for a varying number of processes in space was used to partition the cost of the spatial solve into computations and communications. The results were further used to optimize the distribution of all available processes across the space and time dimensions for the MGRIT method. Finally, both methods of determining the cost for the task diagram are possible, and both have advantages and disadvantages. There is nothing left but to differentiate depending on the use case.

8.2 Results

In the following, we compare the schedule-based performance analysis for PinT methods with four different PinT libraries. For the PFASST algorithm, we choose the Fortran library `LibPFASST` [68] and the Pyton library `pySDC` [100, 101], and for the MGRIT algorithm we choose the C implementation `XBRAID` and the Python implementation `PyMGRIT`. Although there are some implementations for Parareal, such as a Fortran implementation [94], these are often specialized for particular problems and therefore somewhat harder to access. In addition, the Parareal algorithm is the least complicated compared to the other two algorithms, so it is reasonable to assume that if the predictions for PFASST and MGRIT are accurate, the predictions for Parareal will also be accurate. Therefore, we limit our results to the PFASST and MGRIT methods.

One major challenge in comparing the different libraries with the model is the large parameter space. In Chapter 4, we focused on the basic forms of the algorithms, with brief comments on variations and extensions, although the libraries often offer further variations and extensions for the algorithms. At this point, however, we limit ourselves to the basic functionality of the methods. Table 8.1 gives an overview of all parameters that can currently be used for the model. Note that not all parameters are required for every variant of the algorithm, or that some combinations are not possible and/or useful.

Another major challenge in comparing the model with the different libraries is the underlying spatial problem, since all libraries are written in different programming languages and the different methods are typically used for different types of initial value problems. Depending on the spatial problem and the exact parameterization, the convergence behavior and, thus, the required number of iterations of the methods can vary significantly. However, this number of iterations is an important parameter for the model. Coupling the model with some sort of framework for method convergence is a possible direction for future work. To get around these problems and still be able to compare the performance model to the libraries, we implement a pseudo-problem for each library that does not solve a real problem, but simply sleeps for a period of time in the methods to be implemented for the algorithms. This choice allows us to have very large variability in comparing settings to the performance model, so we can cover a much larger test set and, thus, demonstrate the variability of the performance model without running the risk of making an inappropriate choice of parameters for the problem. The convergence and functionality of the methods have already been presented in this thesis and in many other publications.

The procedure is the same for all libraries: For each problem, all functions required for the pseudo-problem are implemented. The runtimes of the various functions are then measured. The measured costs are used as parameters for the model to predict the runtime of a complete run of a method. This predicted runtime is then compared to the actual parallel simulation times of the libraries. Note that the procedure for determining the runtime can be applied to other problems without much effort. Also note that the runtime for each call may change slightly even in this pseudo-problem, so there may be some overhead associated with each function call. This is likely to be larger for a real problem than for our pseudo-problem, so the discrepancy between model and runtime may be slightly larger for real problems.

The model was implemented in Python and is available on GitHub [50], along with the pseudo-problems for the four libraries. All experiments were performed on an Intel Xeon Phi cluster consisting of four 1.4 GHz Intel Xeon Phi processors.

8.2.1 PFASST

In this section, we compare the prediction of our model with the runtime of parallel computations using `LibPFASST` and `pySDC` for our pseudo-problem. The library `LibPFASST` implements the classical view of the PFASST algorithm. Among other things, the library provides a large parameter space for variations of the PFASST algorithm, several types of sweepers, and several variants of the prediction phase that computes an improved initial guess before the actual PFASST iterations begin. The `pySDC` library implements the multigrid view of the PFASST algorithm in Python. The library also provides a large parameter space, many pre-implemented components and spatial problems, and many other features. Both libraries use the windowing strategy when the number of time steps is greater than the number of processes in the time dimension, but the typical choice for the PFASST method is to use the same number of time steps and processes in time. In addition, both libraries use a local convergence criterion based on the residual computed in PFASST, taking into account both the residual at a given time point and the time point before it.

Type	Option	Parameter/Settings
General	Number of time intervals	N_t
	Number of processes in the time dimension	N_P
	Number of levels	L
	Number of iterations	K
	Convergence criterion	global or local
	Communication cost in time dimension at level ℓ	$\sigma_{\mathrm{C}}^{\ell}$
	Time required for convergence criterion	$\sigma_{\mathrm{CC}}^{\ell}$
PFASST	Number of sweeps on level ℓ	μ^{ℓ}
	Collocation nodes on level ℓ	M^{ℓ}
	Predictor type	Fine sweep or burn-in
	Skip fine-level sweep at start/end of iteration	True or False
	Time required for `Sweep` on level ℓ	$\sigma_{\mathrm{S}}^{\ell}$
	Time required for `FEvalAll` on level ℓ	$\sigma_{\mathrm{FA}}^{\ell}$
	Time required for `FEvalSingle` on level ℓ	$\sigma_{\mathrm{FS}}^{\ell}$
	Time required for `RestrictAll` on level ℓ	$\sigma_{\mathrm{RA}}^{\ell}$
	Time required for `RestrictSingle` on level ℓ	$\sigma_{\mathrm{RS}}^{\ell}$
	Time required for `InterpolateAll` on level ℓ	$\sigma_{\mathrm{IA}}^{\ell}$
	Time required for `InterpolateSingle` on level ℓ	$\sigma_{\mathrm{IS}}^{\ell}$
	Time required for `FAS` on level ℓ	$\sigma_{\mathrm{FAS}}^{\ell}$
MGRIT	Coarsening factor from level ℓ to level $\ell+1$	m^{ℓ}
	Cycle type	V or F
	Nested iterations	True or False
	Skip down	True or False
	Number of CF-relaxations on level ℓ	ν^{ℓ}
	Time required for operator Φ on level ℓ	σ_{Φ}^{i}
	Time required for spatial restriction on level ℓ	σ_{SR}^{i}
	Time required for spatial interpolation on level ℓ	σ_{SI}^{i}

Table 8.1: List of parameters that can currently be passed to the model to predict run times. Not all parameters are required for every run.

For the following experiments with `LibPFASST` and `pySDC`, we use an implicit SDC sweeper pre-implemented in both libraries and vary the number of collocation nodes. The implicit sweeper requires a function for an implicit time integration step, which we use to embed a level-dependent sleep period. Note that the performance model is not limited to this choice. To determine the runtime, the sweeper is called on a per-level basis, which is independent of the type of sweeper. In addition, we have implemented functions for the evaluation of the right-hand side, spatial interpolation, and spatial restriction, among others, for both libraries. All functions allow to control the period of sleep at all levels.

In the experiments, we restrict ourselves to the most common use cases of PFASST. Therefore, we choose four two-level settings and five four-level PFASST settings. For the two-level settings, we consider 16 time steps, 16 processes in time, and apply ten iterations of PFASST. We consider two settings with one sweep per level and different numbers of collocation nodes, and two settings with two sweeps at the fine level and only one at the coarse level, again with different numbers of collocation nodes per level. For the four-level PFASST variants, we use 32 time steps, 32 processes in time, and simulate ten iterations as well. Again, we use coarsening in time between the levels, with seven collocation nodes at the fine level, five at the next level, then three and two collocation nodes at the coarsest level per run, and vary the number of sweeps per level. For both libraries, we use a prediction phase before the actual algorithm starts. For `pySDC` we choose a variant that performs a fine sweep, and for `LibPFASST` we use the default variant of the prediction phase, which is a version with burn-in. For all other parameters, we choose the default setting of the respective libraries. Note that even with the same settings in terms of the number of sweeps and the number of collocation nodes per library, the runs differ due to the different prediction phase, PFASST algorithm views, and possible other parameters.

We also use the most typical use case for PFASST when we choose the cost of the pseudo-problem, i. e., the sleep period within the required functions. In most use cases, the costs for the spatial transfer operators and the evaluation are very small. Therefore, we set the sleep period for these three functions to zero and the cost per implicit solve to 0.05 s. Note that these are parameters for the pseudo-problem; we obtain the actual runtimes for the model by measuring the runtimes of the corresponding functions. This allows us to incorporate the minimum overhead incurred by each function call into our model and improve the predictions. We also set the communication cost and the cost of computing the convergence criterion for the model to zero, since both costs are negligible.

Table 8.2 and Figure 8.5 show the predicted runtimes of the performance model, the theoretical lower bound (L.b.) based on the task graph, and the runtime results of `LibPFASST` for the different settings and ten PFASST iterations, as well as the prediction phase. An array represents the selection of sweeps and collocation nodes at the different levels, where the first entry in the array represents the fine level, the next entry the next level, and so on. In all comparisons, we observe that the model predicts the runtimes of the parallel computation with minimal deviation. If we additionally consider the lower bound given by the shortest path within the DAG, we see that both the runtime of LibPFASST and the model reach this lower bound, i. e., the chosen parallelization strategy for the algorithm comes very close to the theoretical bound.

Table 8.3 and Figure 8.6 show the same results for the comparison with pySDC. Again, the parallel runtimes include the calculation of the ten PFASST iterations as well as the prediction phase. Actual runtimes of pySDC differ from the performance model by no more than 5%. While the predictions of the model for the two-level settings are very close to the theoretical lower bound, there are slight deviations for certain settings in the four-level settings. This is likely due to the multigrid view implemented in pySDC. It should be noted, however, that the lower bound generally assumes an infinite number of processes, so the comparison is based on potentially different processor counts. Moreover, an implementation to achieve the lower bound can be very challenging.

8.2.2 MGRIT

For the comparisons of the model with implementations of the MGRIT algorithm we use the two libraries XBRAID and PyMGRIT. The C library XBRAID with interfaces for C++, Fortran and Python allows covering many different variants of the MGRIT algorithm with a large number of options. In addition to the application of the MGRIT algorithm, it provides many examples and various spatial problems and it has also been used for problems outside the classical application domain, such as neural network training [49]. As a stopping criterion, XBRAID uses a global convergence criterion measured in an arbitrary norm based on the residual computed in the MGRIT algorithm. PyMGRIT is a software package in Python that implements the MGRIT algorithm. Besides many settings for the MGRIT algorithm, some examples of spatial problems and a prediction phase based on the nested iteration strategy, PyMGRIT offers the option to choose both local and global convergence criteria. In the library, the convergence criterion is computed independently of calculations within the method at the end of each iteration and can be based on the residual or the jump between iterations. Both libraries use the standard distribution of time points on processes for the parallelization of the MGRIT method. For both libraries, we have implemented a function for time integration, as well as spatial interpolation and spatial restriction for the pseudo-problem. For the three functions, the sleep period can be controlled per level.

In the following experiments, we consider typical use cases for MGRIT. In more detail, we consider 4,096 time steps, 256 temporal processes, and different five- and six-level MGRIT variants. For the five-level variants, we choose a non-uniform coarsening strategy, i. e., different coarsening factors per level. We choose a coarsening factor of 16 between the fine and the next coarser level and a factor of four for all other levels. This choice allows us to consider exactly one interval of F-points per process at the fine level, which optimizes parallelization at the fine level. For the six-level variants, we choose a uniform coarsening strategy with a coarsening factor of four at each level. We use both V- and F-cycles with different relaxation schemes. For V-cycles, we use an FCF-relaxation scheme typical for the MGRIT algorithm. For F-cycles, we use the simpler but cheaper F-relaxation. For `XBRAID`, we use the skip down library option, which skips the down cycle in the first MGRIT iteration. For `PyMGRIT`, we additionally use the nested iteration strategy, a prediction phase to compute an improved initial guess that is performed in addition to the MGRIT iterations. Furthermore, for `PyMGRIT`, we consider both global and local convergence criteria.

We restrict ourselves to the choice of temporal coarsening, i. e., we set the sleep periods for the spatial transfer operators equal to zero. For the sleep period within each time integration, we choose 0.05 s at each level. Again, we measure the runtimes of the functions independently of the sleep periods and use the measured values as parameters for the model to increase the accuracy of the prediction. We set the cost of computing the convergence criterion and the communication for the model to zero.

Table 8.4 and Figure 8.7 show the runtimes of `XBRAID` for different MGRIT settings for the pseudo-problem and prediction of the model, as well as the theoretical lower bound based on the task graph. Level-dependent parameters, such as the coarsening factor, are given as arrays of length $L-1$, since these parameters are given for all levels except the coarsest. The deviation of the measured runtime from the predictions is at most 5% and in particular at most 1% in most cases. More precisely, we have two settings, the five-level V- and F-cycles without skip-down, respectively, where the deviation is slightly larger than in the other cases. In these cases, there is an unexpected behavior of the cluster used, where the first communication between two processes on different nodes within `XBRAID` is extremely expensive, in particular much more expensive than when we test the communication cost independently of libraries. Unfortunately, we have not found a way to solve this problem. At this point, it is important to point out that this is a cluster-related problem, not a library-related problem. In general, this unexpected behavior could also be represented by appropriate communication costs in the model. However, from our point of view, a deviation of 5% is still very acceptable, so for simplicity we do not include these unexpectedly high communication costs in our model. Furthermore, it should be noted that this behavior probably occurs in all parallel runs when multiple nodes are used, while in all other cases the cost is hidden by an overlap of communication and computation.

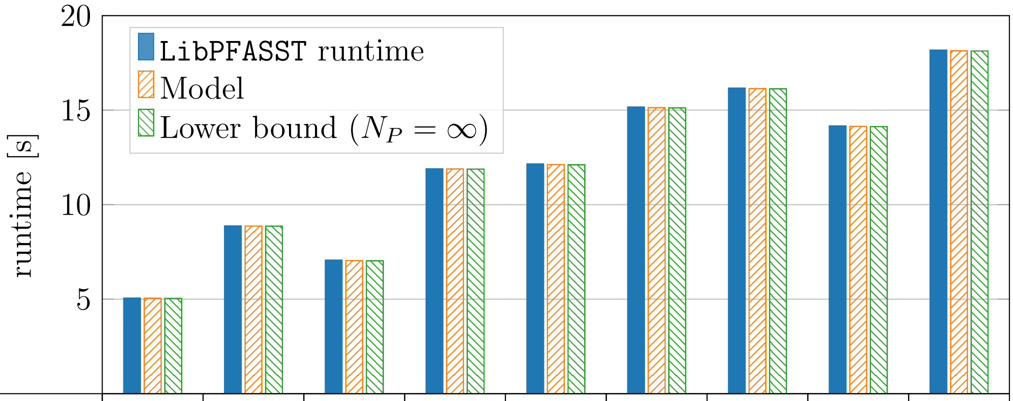
A look at the lower bounds shows that for the five-level settings, the minimum theoretical runtime is slightly lower than the runtime of the model. This is probably primarily due to the global convergence criterion in XBRAID, which is typically implemented as collective communication and, thus, blocking communication. The difference is slightly larger for the six-level settings, but this was to be expected given the choice of parameters. The chosen coarsening factor of four allows the effective utilization of more than 256 processes in the time dimension at the fine level, which would reduce the runtime.

Table 8.5 and Figure 8.8 show the results for predicted and parallel PyMGRIT runtimes. Comparing the two values, the actual runtimes differ from the predicted runtimes by at most 5%. It can be seen that the runtimes for the local convergence criterion are slightly better than for the global convergence criteria. Global convergence criteria are typically determined by collective communication, which incurs additional communication costs that we have omitted in the predictions. Again, the measured runtimes contain the same unexpected communication costs between nodes as described in the results of XBRAID.

Comparing the difference between the predictions and the lower bound for the five-level settings, we find that the difference is smaller when a local criterion is used than when a global criterion is used. For the six-level cases, the difference is again slightly larger, but this is again due to the fact that the selected number of processes in time does not exploit the full parallelization potential of the method. Another interesting result is that the lower bound for the same variants with and without nested iterations is more or less the same, which means that the prediction phase with nested iterations can theoretically be achieved without additional visible costs. However, this would require a suitable implementation. Investigating specific strategies that could enable more parallelization and implementing them accordingly are part of future work.

N_t	N_P	L	K	# Sweeps	collocation nodes	LibPFASST runtime	Model	L. b. ($N_P = \infty$)
16	16	2	10	$(1, 1)$	$(5, 3)$	5.06	5.05	5.04
16	16	2	10	$(1, 1)$	$(7, 5)$	8.88	8.87	8.86
16	16	2	10	$(2, 1)$	$(5, 3)$	7.07	7.04	7.03
16	16	2	10	$(2, 1)$	$(7, 5)$	11.9	11.89	11.88
32	32	4	10	$(1, 1, 1, 1)$	$(7, 5, 3, 2)$	12.16	12.12	12.11
32	32	4	10	$(2, 1, 1, 1)$	$(7, 5, 3, 2)$	15.17	15.13	15.12
32	32	4	10	$(1, 2, 1, 1)$	$(7, 5, 3, 2)$	16.17	16.14	16.13
32	32	4	10	$(1, 1, 2, 1)$	$(7, 5, 3, 2)$	14.17	14.14	14.13
32	32	4	10	$(1, 2, 2, 1)$	$(7, 5, 3, 2)$	18.18	18.14	18.13

Table 8.2: Predictions of the model and measured runtimes for the LibPFASST library for the pseudo-problem, with a sleep period of 0.05 s for each implicit solve and zero for all others. For the graphical view of the results, see Figure 8.5.

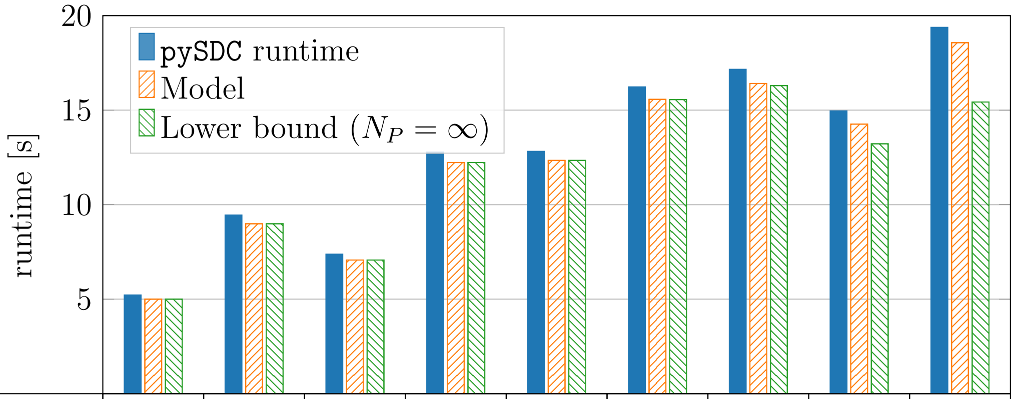


N	16	16	16	16	32	32	32	32	32
N_p	16	16	16	16	32	32	32	32	32
L	2	2	2	2	4	4	4	4	4
K	10	10	10	10	10	10	10	10	10
# Sweeps	(1,1)	(1,1)	(2,1)	(2,1)	(1,1,1,1)	(2,1,1,1)	(1,2,1,1)	(1,1,2,1)	(1,2,2,1)
collocation nodes	(5,3)	(7,5)	(5,3)	(7,5)	(7,5,3,2)	(7,5,3,2)	(7,5,3,2)	(7,5,3,2)	(7,5,3,2)

Figure 8.5: Graphical representation of the predictions of the model and measured runtimes for the LibPFASST library for the pseudo-problem, with a sleep period of 0.05 s for each implicit solve and zero for all others. For a tabular view of the results, see Table 8.2.

N_t	N_P	L	K	# Sweeps	collocation nodes	pySDC runtime	Model	L. b. ($N_P = \infty$)
16	16	2	10	$(1,1)$	$(5,3)$	5.22	5.00	5.00
16	16	2	10	$(1,1)$	$(7,5)$	9.45	8.99	8.99
16	16	2	10	$(2,1)$	$(5,3)$	7.38	7.07	7.07
16	16	2	10	$(2,1)$	$(7,5)$	12.78	12.23	12.23
32	32	4	10	$(1,1,1,1)$	$(7,5,3,2)$	12.82	12.34	12.34
32	32	4	10	$(2,1,1,1)$	$(7,5,3,2)$	16.23	15.57	15.56
32	32	4	10	$(1,2,1,1)$	$(7,5,3,2)$	17.16	16.41	16.30
32	32	4	10	$(1,1,2,1)$	$(7,5,3,2)$	14.96	14.26	13.22
32	32	4	10	$(1,2,2,1)$	$(7,5,3,2)$	19.38	18.57	15.43

Table 8.3: Predictions of the model and measured runtimes for the pySDC library for the pseudo-problem, with a sleep period of 0.05 s for each implicit solve and zero for all others. For a graphical view of the results, see Figure 8.6.

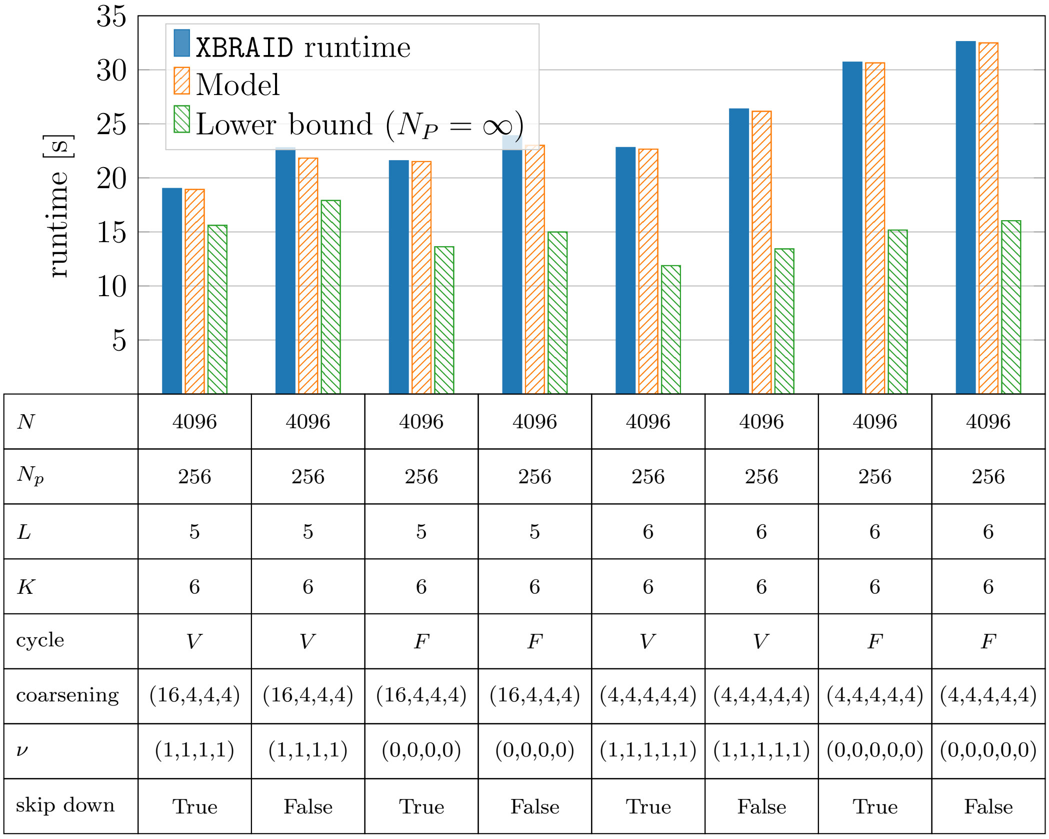


N	16	16	16	16	32	32	32	32	32
N_p	16	16	16	16	32	32	32	32	32
L	2	2	2	2	4	4	4	4	4
K	10	10	10	10	10	10	10	10	10
# Sweeps	(1,1)	(1,1)	(2,1)	(2,1)	(1,1,1,1)	(2,1,1,1)	(1,2,1,1)	(1,1,2,1)	(1,2,2,1)
collocation nodes	(5,3)	(7,5)	(5,3)	(7,5)	(7,5,3,2)	(7,5,3,2)	(7,5,3,2)	(7,5,3,2)	(7,5,3,2)

Figure 8.6: Graphical representation of the predictions of the model and measured runtimes for the pySDC library for the pseudo-problem, with a sleep period of 0.05 s for each implicit solve and zero for all others. For a tabular view of the results, see Table 8.3.

N_t	N_P	L	K	cyc.	coarsening	ν	skip down	XBRAID runtime	Model	L.b. ($N_P = \infty$)
4096	256	5	6	V	$(16, 4, 4, 4)$	$(1, 1, 1, 1)$	True	19.02	18.93	15.61
4096	256	5	6	V	$(16, 4, 4, 4)$	$(1, 1, 1, 1)$	False	22.79	21.83	17.91
4096	256	5	6	F	$(16, 4, 4, 4)$	$(0, 0, 0, 0)$	True	21.59	21.51	13.63
4096	256	5	6	F	$(16, 4, 4, 4)$	$(0, 0, 0, 0)$	False	23.90	23.01	14.99
4096	256	6	6	V	$(4, 4, 4, 4, 4)$	$(1, 1, 1, 1, 1)$	True	22.81	22.66	11.88
4096	256	6	6	V	$(4, 4, 4, 4, 4)$	$(1, 1, 1, 1, 1)$	False	26.38	26.16	13.43
4096	256	6	6	F	$(4, 4, 4, 4, 4)$	$(0, 0, 0, 0, 0)$	True	30.71	30.64	15.17
4096	256	6	6	F	$(4, 4, 4, 4, 4)$	$(0, 0, 0, 0, 0)$	False	32.61	32.49	16.04

Table 8.4: Predictions of the model and measured runtimes for the XBRAID library for the pseudo-problem, with a sleep period of 0.05 s for each time integration and zero for all others. For a graphical view of the results, see Figure 8.7.

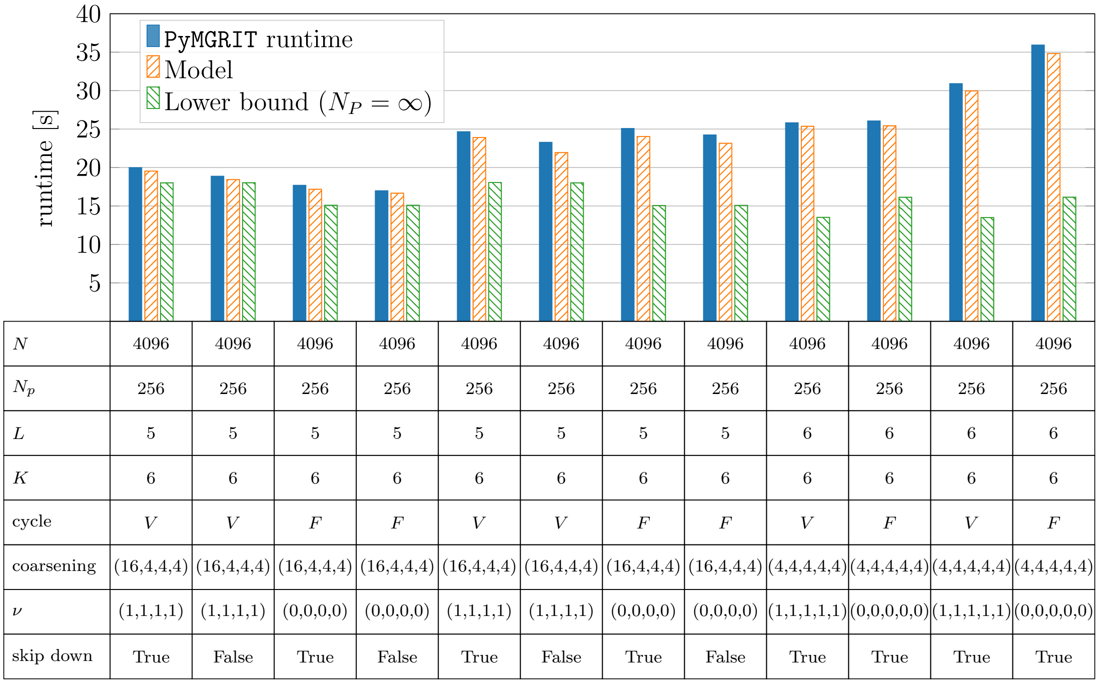


N	4096	4096	4096	4096	4096	4096	4096	4096
N_p	256	256	256	256	256	256	256	256
L	5	5	5	5	6	6	6	6
K	6	6	6	6	6	6	6	6
cycle	V	V	F	F	V	V	F	F
coarsening	(16,4,4,4)	(16,4,4,4)	(16,4,4,4)	(16,4,4,4)	(4,4,4,4,4)	(4,4,4,4,4)	(4,4,4,4,4)	(4,4,4,4,4)
ν	(1,1,1,1)	(1,1,1,1)	(0,0,0,0)	(0,0,0,0)	(1,1,1,1,1)	(1,1,1,1,1)	(0,0,0,0,0)	(0,0,0,0,0)
skip down	True	False	True	False	True	False	True	False

Figure 8.7: Graphical representation of the predictions of the model and measured runtimes for the XBRAID library for the pseudo-problem, with a sleep period of 0.05 s for each implicit solve and zero for all others. For a tabular view of the results, see Table 8.4.

N_t	N_P	L	K	cyc.	coarsening	ν	nest. iter.	conv. crit.	`PyMGRIT` runtime	Model	L.b. ($N_P = \infty$)
4096	256	5	6	V	$(16, 4, 4, 4)$	$(1, 1, 1, 1)$	T	local	19.99	19.53	18.01
4096	256	5	6	V	$(16, 4, 4, 4)$	$(1, 1, 1, 1)$	F	local	18.87	18.43	18.02
4096	256	5	6	F	$(16, 4, 4, 4)$	$(0, 0, 0, 0)$	T	local	17.69	17.17	15.09
4096	256	5	6	F	$(16, 4, 4, 4)$	$(0, 0, 0, 0)$	F	local	16.98	16.66	15.09
4096	256	5	6	V	$(16, 4, 4, 4)$	$(1, 1, 1, 1)$	T	global	24.66	23.91	18.06
4096	256	5	6	V	$(16, 4, 4, 4)$	$(1, 1, 1, 1)$	F	global	23.28	21.94	18.01
4096	256	5	6	F	$(16, 4, 4, 4)$	$(0, 0, 0, 0)$	T	global	25.08	24.04	15.07
4096	256	5	6	F	$(16, 4, 4, 4)$	$(0, 0, 0, 0)$	F	global	24.24	23.15	15.08
4096	256	6	6	V	$(4, 4, 4, 4, 4)$	$(1, 1, 1, 1, 1)$	T	local	25.82	25.36	13.52
4096	256	6	6	F	$(4, 4, 4, 4, 4)$	$(0, 0, 0, 0, 0)$	T	local	26.06	25.43	16.13
4096	256	6	6	V	$(4, 4, 4, 4, 4)$	$(1, 1, 1, 1, 1)$	T	global	30.91	29.97	13.49
4096	256	6	6	F	$(4, 4, 4, 4, 4)$	$(0, 0, 0, 0, 0)$	T	global	35.93	34.82	16.14

Table 8.5: Predictions of the model and measured runtimes for the `PyMGRIT` library for the pseudo-problem, with a sleep period of 0.05 s for each time integration and zero for all others. For a graphical view of the results, see Figure 8.8.



N	4096	4096	4096	4096	4096	4096	4096	4096	4096	4096	4096	4096
N_p	256	256	256	256	256	256	256	256	256	256	256	256
L	5	5	5	5	5	5	5	5	6	6	6	6
K	6	6	6	6	6	6	6	6	6	6	6	6
cycle	V	V	F	F	V	V	F	F	V	F	V	F
coarsening	(16,4,4,4)	(16,4,4,4)	(16,4,4,4)	(16,4,4,4)	(16,4,4,4)	(16,4,4,4)	(16,4,4,4)	(16,4,4,4)	(4,4,4,4,4)	(4,4,4,4,4)	(4,4,4,4,4)	(4,4,4,4,4)
ν	(1,1,1,1)	(1,1,1,1)	(0,0,0,0)	(0,0,0,0)	(1,1,1,1)	(1,1,1,1)	(0,0,0,0)	(0,0,0,0)	(1,1,1,1,1)	(0,0,0,0,0)	(1,1,1,1,1)	(0,0,0,0,0)
skip down	True	False	True	False	True	False	True	False	True	True	True	True

Figure 8.8: Graphical representation of the predictions of the model and measured runtimes for the `PyMGRIT` library for the pseudo-problem, with a sleep period of 0.05 s for each implicit solve and zero for all others. For a tabular view of the results, see Table 8.5.

Chapter 9

Conclusions & Outlook

In this work, we applied the time-parallel MGRIT algorithm to the simulation of an induction machine. The main results of this work are the extension of the scope of PinT methods, the development of a new variant of the MGRIT algorithm, and the provision of a new library that implements the MGRIT framework. In addition, a new performance model for PinT methods is presented.

Applying the MGRIT algorithm to the model “im_3_kW” of a three-phase induction machine driven by a PWM voltage source reduced the required simulation runtime by a factor of up to 14 compared to the traditional sequential time stepping using 256 processes for temporal parallelization. The use of spatial coarsening between levels of the MGRIT algorithm allows for a further reduction in runtime, resulting in a speedup by a factor of up to 21.9 compared to the sequential time stepping. This shows that parallelization of the time dimension is a promising approach to reduce the runtime of time-dependent simulations for complex problems, especially when spatial parallelization is exhausted.

By incorporating the AT-MGRIT algorithm into a geometry optimization of the induction motor, an improved geometry in terms of motor efficiency was achieved, and the time-parallel simulation in each optimization step provided a shorter time to solution compared to the classical time stepping method.

The release of the `PyMGRIT` package, which contains the code for all numerical experiments in this work, makes the work accessible to the PinT community and provides an easy entry point for researchers interested in using the MGRIT algorithm for time-dependent simulations.

A challenge in using the MGRIT algorithm, as with other PinT methods, is the large parameter space that allows for different variations of the method, with the right choice of parameters having a significant impact on the efficiency of the method. The presented task-based performance model for the three PinT methods Parareal, PFASST, and MGRIT provides runtime prediction for a given setting and can therefore help to choose appropriate parameters without performing expensive simulations. We used the model to predict runtimes for simulations and compared these predictions to actual parallel runtimes using four different PinT libraries. In all test cases for all libraries, the actual runtime deviated from the expected value only by a maximum of 5%.

In future work, it would be interesting to further explore the task-based view of PinT methods. For example, the model can be used to find the optimal parameter setting for a given problem in terms of the expected simulation time and to choose the distribution of processes over spatial and temporal dimensions. A future challenge is certainly the additional prediction of the required number of iterations of the method for particular problems and settings, but coupling the model with the generalized convergence study tool for PinT methods [42] could be promising. In addition, the task graph gives us many other possibilities that are independent of predictions. First, if we look at the lower bounds of the algorithms, we can see that for some settings, especially for the MGRIT algorithm, there is still some gap between the achieved runtimes and the lower bound. A task-based implementation of the algorithm could potentially close this gap. Such an implementation also has the potential to take into account not only static load balancing, but also dynamic load balancing, which may be necessary to avoid idle times on more complicated problems. In addition, this global view of task graphs could be an approach to create a framework that combines multiple PinT methods. This would make the proper selection of an appropriate PinT method, as well as the application of temporal parallelization to complex problems, such as the induction motor model studied, even more accessible to academia and industry.

List of Figures

List of Tables

List of Algorithms

List of Notations

Throughout this thesis, scalars are denoted by lower-case letters, vectors are denoted by bold lower-case letter and matrices and constants are denoted by upper-case letters. In addition, the following abbreviations and notations are used across all chapters:

Governing application

$\mathbf{B}$	Magnetic flux density
$\mathbf{H}$	Magnetic field strength
$\mathbf{D}$	Electric flux density
$\mathbf{E}$	Electric field strength
$\mathbf{J}$	Electric current density
$\mathbf{J}_s$	Electric source current density
$\mathbf{A}$	Magnetic vector potential
ϕ	Electric scalar potential
ϱ	Electric charge density
ϵ	Electric permittivity
σ	Electric conductivity
R	Resistance
L	Inductance
C	Capacitance
v	Voltage
i	Current
ω_{sync}	Synchronous speed
ω_{mech}	Mechanical speed
s	Slip
T_{EM}	Electromagnetic torque
P_{mech}	Mechanical power
P_{loss}	Joule losses
J	Moment of inertia
θ	Rotor angle

Mathematical notation

$\mathbb{R}$	Set of real numbers
$\mathbb{R}^d$	d-dimensional Euclidean space
$\mathbb{R}_0^+$	The set of positive real numbers with zero
$\mathbb{Z}$	The set of integer numbers
$\mathbb{Z}_+ = \mathbb{N}$	The set of positive integer numbers
$\mathbb{C}$	The set of complex numbers
$\mathbb{C}^-$	The set of complex numbers with negative real part
Ω	Domain
$\partial\Omega$	Boundaries of the domain Ω
$C^k(\Omega)$	The set of k times continuously differentiable functions on Ω
$C^k(\Omega)_0$	The set of functions $\phi \in C^k(\Omega)$ having compact support in Ω
$C^k(\Omega)_0'$	The dual space of $C^k(\Omega)_0$
$L^p(\Omega), 1 \leq p < \infty$	The set of functions ϕ on Ω for which $\|\phi\|^p$ is Lebesgue integrable
$W^{s,p}(\Omega)$	The fundamental Sobolev spaces
$H^s(\Omega)$	Hilbert spaces

Acronyms

1D	one-dimensional
2D	two-dimensional
3D	three-dimensional
AC	alternating current
AT-MGRIT	asynchronous truncated multigrid-reduction-in-time
DAE	differential-algebraic equation
DC	direct current
FAS	full aproximation storage
MGRIT	multigrid-reduction-in-time
ODE	ordinary differential equation
PDE	partial differential equation
PFASST	parallel full approximation scheme in space and time
PinT	parallel-in-time
PWM	pulse-width modulation
SOR	successive over-relaxation
DAG	directed acyclic graph

Bibliography

[1] A. Alonso Rodríguez and A. Valli, *Eddy Current Approximation of Maxwell Equations*, vol. 4 of Modeling, Simulation and Applications, Springer.

[2] A. Arkkio, *Analysis of induction motors based on the numerical solution of the magnetic field and circuit equations*, PhD thesis, Helsinki University of Technology, 1987.

[3] F. Arscott and A. Filippov, *Differential Equations with Discontinuous Righthand Sides: Control Systems*, Mathematics and its Applications, Springer Netherlands, 2013.

[4] U. M. Ascher and L. R. Petzold, *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*, Society for Industrial and Applied Mathematics, USA, 1st ed., 1998.

[5] E. Aubanel, *Scheduling of Tasks in the Parareal Algorithm*, Parallel Computing, 37 (2011), p. 172–182.

[6] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, D. Karpeyev, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, and H. Zhang, *PETSc users manual*, Technical Report, Argonne National Laboratory, 2020.

[7] R. Bank, R. Falgout, T. Jones, T. A. Manteuffel, S. F. McCormick, and J. W. Ruge, *Algebraic multigrid domain and range decomposition (AMG-DD/AMG-RD)*, SIAM Journal on Scientific Computing, 37 (2015), pp. S113–S136.

[8] D. Bast, I. Kulchytska-Ruchka, S. Schöps, and O. Rain, *Accelerated Steady-State Torque Computation for Induction Machines using Parallel-In-Time Algorithms*, IEEE Transactions on Magnetics, (2019).

[9] A. Bermúdez, D. Gómez, M. Piñeiro, and P. Salgado, *A novel numerical method for accelerating the computation of the steady-state in induction machines*, Computers and Mathematics with Applications, (2019).

[10] L. Berry, W. Elwasif, J. Reynolds-Barredo, D. Samaddar, R. Sanchez, and D. Newman, *Event-based parareal: A data-flow based implementation of parareal*, Journal of Computational Physics, 231 (2012), pp. 5945–5954.

[11] H. Black, *Modulation theory*, Bell Telephone Laboratories series, Van Nostrand, 1953.

[12] M. Bolten, S. Friedhoff, and J. Hahne, *Task Graph-Based Performance Analysis of Parallel-in-Time Methods.* Available at SSRN: `https://ssrn.com/abstract=4201056`, 2022 (Submitted).

[13] M. Bolten, S. Friedhoff, J. Hahne, and S. Schöps, *Parallel-in-time simulation of an electrical machine using MGRIT*, Computing and Visualization in Science, 23 (2020), pp. Paper No. 14, 14.

[14] M. Bolten, D. Moser, and R. Speck, *A multigrid perspective on the parallel full approximation scheme in space and time*, Numerical Linear Algebra with Applications, 24 (2017).

[15] D. Braess, *Finite Elements: Theory, Fast Solvers, and Applications in Solid Mechanics*, Cambridge University Press, 2007.

[16] A. Brandt, *Multi-level adaptive solutions to boundary-value problems*, Mathematics of Computation, 31 (1977), pp. 333–390.

[17] H. Brezis, *Functional Analysis, Sobolev Spaces and Partial Differential Equations*, Universitext, Springer New York, 2010.

[18] K. Burrage, *Parallel and Sequential Methods for Ordinary Differential Equations*, Clarendon Press, USA, 1995.

[19] C. Cartis, J. Fiala, B. Marteau, and L. Roberts, *Improving the Flexibility and Robustness of Model-Based Derivative-Free Optimization Solvers*, ACM Transactions on Mathematical Software, 45 (2019).

[20] A. J. Christlieb, C. B. Macdonald, and B. W. Ong, *Parallel High-Order Integrators*, SIAM Journal on Scientific Computing, 32 (2010), pp. 818–835.

[21] J. Cortial and C. Farhat, *A time-parallel implicit method for accelerating the solution of non-linear structural dynamics problems*, International Journal for Numerical Methods in Engineering, 77 (2009), pp. 451–470.

[22] F. Danieli and S. MacLachlan, *Multigrid Reduction in Time for non-linear hyperbolic equations*, 2021.

[23] H. De Sterck, R. D. Falgout, S. Friedhoff, O. A. Krzysik, and S. P. MacLachlan, *Optimizing multigrid reduction-in-time and Parareal coarse-grid operators for linear advection*, Numerical Linear Algebra with Applications, 28 (2021).

[24] P. Deuflhard, *Newton methods for nonlinear problems: affine invariance and adaptive algorithms*, Springer, 2004.

[25] P. Deuflhard, W. Rheinboldt, and F. Bornemann, *Scientific Computing with Ordinary Differential Equations*, Texts in Applied Mathematics, Springer New York, 2012.

[26] V. A. Dobrev, T. Kolev, N. A. Petersson, and J. B. Schroder, *Two-Level Convergence Theory for Multigrid Reduction in Time (MGRIT)*, SIAM Journal on Scientific Computing, 39 (2017), pp. S501–S527.

[27] P. Dular and C. Geuzaine, *GetDP: A General Environment for the Treatment of Discrete Problems*. `http://www.getdp.info`, Online; accessed December 27, 2022.

[28] A. Dutt, L. Greengard, and V. Rokhlin, *Spectral deferred correction methods for ordinary differential equations*, BIT Numerical Mathematics, 40 (2000), pp. 241–266.

[29] M. Emmett and M. Minion, *Toward an efficient parallel in time method for partial differential equations*, Communications in Applied Mathematics and Computational Science, 7 (2012), pp. 105 – 132.

[30] R. D. Falgout, S. Friedhoff, T. V. Kolev, S. P. MacLachlan, and J. B. Schroder, *Parallel time integration with multigrid*, SIAM Journal on Scientific Computing, 36 (2014), pp. C635–C661.

[31] R. D. Falgout, A. Katz, T. V. Kolev, J. B. Schroder, A. Wissink, and U. M. Yang, *Parallel Time Integration with Multigrid Reduction for a Compressible Fluid Dynamics Application*, Technical Report, Lawrence Livermore National Laboratory, 2015.

[32] R. D. Falgout, M. Lecouvez, and C. S. Woodward, *A parallel-in-time algorithm for variable step multistep methods*, Journal of Computational Science, 37 (2019).

[33] R. D. Falgout, T. A. Manteuffel, B. O'Neill, and J. B. Schroder, *Multigrid Reduction in Time for Nonlinear Parabolic Problems: A Case Study*, SIAM Journal on Scientific Computing, 39 (2017), pp. S298–S322.

[34] C. Farhat, J. Cortial, C. Dastillung, and H. Bavestrello, *Time-parallel implicit integrators for the near-real-time prediction of linear structural dynamic responses*, International Journal for Numerical Methods in Engineering, 67 (2006), pp. 697–724.

[35] S. Friedhoff, J. Hahne, I. Kulchytska-Ruchka, and S. Schöps, *Exploring Parallel-in-Time Approaches for Eddy Current Problems*, in Progress in Industrial Mathematics at ECMI 2018, vol. 30 of The European Consortium for Mathematics in Industry, Springer, 2020.

[36] S. Friedhoff and B. S. Southworth, *On "optimal" h-independent convergence of parareal and multigrid-reduction-in-time using runge-kutta time integration*, Numerical Linear Algebra with Applications, 28 (2021).

[37] H. Gahvari, V. A. Dobrev, R. D. Falgout, T. V. Kolev, J. B. Schroder, M. Schulz, and U. M. Yang, *A Performance Model for Allocating the Parallelism in a Multigrid-in-Time Solver*, in 2016 7th International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS), 2016, pp. 22–31.

[38] M. Gander, Y.-L. Jiang, B. Song, and H. Zhang, *Analysis of Two Parareal Algorithms for Time-Periodic Problems*, SIAM Journal on Scientific Computing, 35 (2013).

[39] M. J. Gander, *50 years of Time Parallel Time Integration*, in Multiple Shooting and Time Domain Decomposition, Springer, 2015, pp. 69–113.

[40] M. J. Gander and E. Hairer, *Nonlinear Convergence Analysis for the Parareal Algorithm*, in Domain Decomposition Methods in Science and Engineering XVII, Berlin, Heidelberg, 2008, Springer Berlin Heidelberg, pp. 45–56.

[41] M. J. Gander, I. Kulchytska-Ruchka, I. Niyonzima, and S. Schöps, *A New Parareal Algorithm for Problems with Discontinuous Sources*, SIAM Journal on Scientific Computing, 41 (2019), pp. B375–B395.

[42] M. J. Gander, T. Lunet, D. Ruprecht, and R. Speck, *A unified analysis framework for iterative parallel-in-time algorithms*, 2022.

[43] C. Geuzaine, *GetDP: a general finite-element solver for the de Rham complex*, PAMM, 7 (2007).

[44] C. Geuzaine and J.-F. Remacle, *Gmsh: A three-dimensional finite element mesh generator with built-in pre- and post-processing facilities.* `http://www.gmsh.info`, Online; accessed December 27, 2022.

[45] C. Geuzaine and J.-F. Remacle, *Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities*, International Journal for Numerical Methods in Engineering, 79 (2009), pp. 1309–1331.

[46] S. Götschel and M. L. Minion, *Parallel-in-Time for Parabolic Optimal Control Problems Using PFASST*, in Domain Decomposition Methods in Science and Engineering XXIV, Cham, 2018, Springer International Publishing, pp. 363–371.

[47] D. Griffiths, P. Griffiths, and R. College, *Introduction to Electrodynamics*, Prentice Hall, 1999.

[48] J. Gyselinck, L. Vandevelde, and J. Melkebeek, *Multi-slice FE modeling of electrical machines with skewed slots-the skew discretization error*, Magnetics, IEEE Transactions on, 37 (2001), pp. 3233 – 3237.

[49] S. Günther, L. Ruthotto, J. B. Schroder, E. C. Cyr, and N. R. Gauger, *Layer-Parallel Training of Deep Residual Neural Networks*, SIAM Journal on Mathematics of Data Science, 2 (2020), pp. 1–23.

[50] J. Hahne, *Github repository for the PinT performance model.* `https://github.com/pymgrit/performance_model`, Online; accessed December 27, 2022.

[51] J. Hahne and S. Friedhoff, *Documentation for* `PyMGRIT`. `https://pymgrit.github.io/pymgrit/`, Online; accessed December 27, 2022.

[52] ——, *Github repository for* `PyMGRIT`. `https://github.com/pymgrit/pymgrit`, Online; accessed December 27, 2022.

[53] J. Hahne, S. Friedhoff, and M. Bolten, *Algorithm 1016: PyMGRIT: A Python Package for the Parallel-in-Time Method MGRIT*, ACM Transactions on Mathematical Software, 47 (2021).

[54] J. Hahne, B. Polenz, I. Kulchytska-Ruchka, S. Friedhoff, S. Ulbrich, and S. Schöps, *Parallel-in-time optimization of induction motors*, 2022 (Submitted).

[55] J. Hahne, B. S. Southworth, and S. Friedhoff, *Asynchronous truncated multigrid-reduction-in-time*, SIAM Journal on Scientific Computing, (2022), pp. S281–S306.

[56] E. Hairer, S. Nørsett, and G. Wanner, *Solving Ordinary Differential Equations I: Nonstiff problems*, Springer, Berlin, 2nd ed., 2000.

[57] E. Hairer, S. P. Nørsett, and G. Wanner, *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*, Springer Series in Computational Mathematics, Springer, 2 ed., 2002.

[58] A. Hessenthaler, *Multilevel convergence analysis : parallel-in-time integration for fluid-structure interaction problems with applications in cardiac flow modeling*, PhD thesis, Universität Stuttgart, 2018.

[59] A. Hessenthaler, B. S. Southworth, D. Nordsletten, O. Röhrle, R. D. Falgout, and J. B. Schroder, *Multilevel Convergence Analysis of Multigrid-Reduction-in-Time*, SIAM Journal on Scientific Computing, 42 (2020), pp. A771–A796.

[60] C.-W. Ho, A. Ruehli, and P. Brennan, *The Modified Nodal Approach to Network Analysis*, Circuits and Systems, IEEE Transactions on, 22 (1975), pp. 504 – 509.

[61] G. Horton and S. Vandewalle, *A Space-Time Multigrid Method for Parabolic Partial Differential Equations*, SIAM Journal on Scientific Computing, 16 (1995), pp. 848–864.

[62] A. Howse, H. De Sterck, R. D. Falgout, S. MacLachlan, and J. B. Schroder, *Parallel-In-Time Multigrid with Adaptive Spatial Coarsening for The Linear Advection and Inviscid Burgers Equations*, SIAM Journal on Scientific Computing, 41 (2019), pp. A538–A565.

[63] N. Ida and J. P. A. Bastos, *Electromagnetics and calculation of fields*, Springer, 2 ed., 1997.

[64] L. Kronsjö, *A note on the "nested iterations" methods*, Nordisk Tidskrift for Informationsbehandling, 15 (1975), pp. 107–110.

[65] L. Kronsjö and G. Dahlquist, *On the design of nested iterations for elliptic difference equations*, Nordisk Tidskrift for Informationsbehandling, 12 (1972), pp. 63–71.

[66] I. Kulchytska-Ruchka, *Parallel-in-Time Simulation of Electromagnetic Energy Converters*, PhD thesis, Technische Universität Darmstadt, 2021.

[67] I. Kulchytska-Ruchka and S. Schöps, *Efficient Parallel-in-Time Solution of Time-Periodic Problems Using a MultiHarmonic Coarse Grid Correction*, SIAM Journal on Scientific Computing, 43 (2021), pp. C61–C88.

[68] LBNL, *Github repository for `LibPFASST`*. `https://github.com/libpfasst/LibPFASST`, Online; accessed December 27, 2022.

[69] M. Lecouvez, R. D. Falgout, C. S. Woodward, and P. Top, *A parallel multigrid reduction in time method for power systems*, in 2016 IEEE Power and Energy Society General Meeting (PESGM), 2016, pp. 1–5.

[70] B. Lepsa and A. Sandu, *An Efficient Error Control Mechanism for the Adaptive 'parareal' Time Discretization Algorithm*, in Proceedings of the 2010 Spring Simulation Multiconference, SpringSim '10, Society for Computer Simulation International, 2010.

[71] J.-L. Lions, Y. Maday, and G. Turinici, *Résolution d'EDP par un schéma en temps "pararéel"*, Comptes Rendus de l'Académie des Sciences. Série I. Mathématique, 332 (2001), pp. 661–668.

[72] LLNL, *Website for XBraid*. `https://www.llnl.gov/casc/xbraid`, Online; accessed December 27, 2022.

[73] T. Lunet, J. Bodart, S. Gratton, and X. Vasseur, *Time-parallel simulation of the decay of homogeneous turbulence using Parareal with spatial coarsening*, Computing and Visualization in Science, 19 (2018), pp. 31–44.

[74] J. C. Maxwell, *A dynamical theory of the electromagnetic field*, Philosophical Transactions of the Royal Society of London, 155 (1865), pp. 459–512.

[75] V. Mele, E. M. Constantinescu, L. Carracciuolo, and L. D'Amore, *A PETSc parallel-in-time solver based on MGRIT algorithm*, Concurrency and Computation: Practice and Experience, 30 (2018).

[76] M. Minion, *A hybrid parareal spectral deferred corrections method*, Communications in Applied Mathematics and Computational Science, 5 (2010), pp. 265 – 301.

[77] W. Mitchell and T. Manteuffel, *Advances in implementation, theoretical motivation, and numerical results for the nested iteration with range decomposition algorithm*, Numerical Linear Algebra with Applications, 25 (2018).

[78] W. B. Mitchell, R. Strzodka, and R. D. Falgout, *Parallel performance of algebraic multigrid domain decomposition*, Numerical Linear Algebra with Applications, 28 (2021).

[79] E. Moon and E. C. Cyr, *Parallel Training of GRU Networks with a Multi-Grid Solver for Long Sequences*, in International Conference on Learning Representations, 2022.

[80] D. Moser, *A multigrid perspective on the parallel full approximation scheme in space and time*, PhD thesis, Universität Kassel, 2018.

[81] A. S. Nielsen, G. Brunner, and J. S. Hesthaven, *Communication-aware adaptive Parareal with application to a nonlinear hyperbolic system of partial differential equations*, Journal of Computational Physics, 371 (2018), pp. 483–505.

[82] J. Nievergelt, *Parallel Methods for Integrating Ordinary Differential Equations*, Communications ACM, 7 (1964), p. 731–733.

[83] B. Ong and J. Schroder, *Applications of time parallelization*, Computing and Visualization in Science, 23 (2020).

[84] J. E. Pearson, *Complex Patterns in a Simple System*, Science, 261 (1993), pp. 189–192.

[85] C. Pechstein, *Multigrid-Newton-Methods for Nonlinear Magnetostatic Problems*, Master's thesis, Universität Linz, 2004.

[86] C. Pechstein and B. Jüttler, *Monotonicity-preserving interproximation of b–h-curves*, Journal of Computational and Applied Mathematics, 196 (2006), pp. 45–57.

[87] M. J. D. Powell, *The BOBYQA algorithm for bound constrained optimization without derivatives*, Cambridge NA Report NA2009/06, University of Cambridge, Cambridge, (2009), pp. 26–46.

[88] A. Preumont, *Mechatronics: Dynamics of Electromechanical and Piezoelectric Systems*, Solid Mechanics and Its Applications, Springer Netherlands, 2006.

[89] J. C. Rautio, *The Long Road to Maxwell's Equations*, IEEE Spectrum, 51 (2014), pp. 36–56.

[90] T. Reis, *Mathematical Modeling and Analysis of Nonlinear Time-invariant RLC Circuits*, Hamburger Beiträge zur angewandten Mathematik, 2013.

[91] R. Riaza, *Differential-algebraic Systems: Analytical Aspects And Circuit Applications*, World Scientific Publishing Company, 2008.

[92] Y. Robert, *Task Graph Scheduling*, in Encyclopedia of Parallel Computing, D. Padua, ed., Springer US, 2011, pp. 2013–2025.

[93] D. Ruprecht, *Convergence of Parareal with spatial coarsening*, PAMM, 14 (2014), pp. 1031–1034.

[94] ——, *Shared Memory Pipelined Parareal*, in Euro-Par 2017: Parallel Processing, Springer International Publishing, 2017, pp. 669–681.

[95] S. J. Salon, *Finite Element Analysis of Electrical Machines*, Kluwer, 1995.

[96] S. Schöps, H. De Gersem, and T. Weiland, *Winding Functions in Transient Magnetoquasistatic Field-Circuit Coupled Simulations*, COMPEL: The International Journal for Computation and Mathematics in Electrical and Electronic Engineering, 32 (2013), pp. 2063–2083.

[97] J. B. Schroder, *Parallelizing Over Artificial Neural Network Training Runs with Multigrid*, Technical Report, Lawrence Livermore National Laboratory, 2017.

[98] J. B. Schroder, R. D. Falgout, C. S. Woodward, P. Top, and M. Lecouvez, *Parallel-in-Time Solution of Power Systems with Scheduled Events*, in 2018 IEEE Power Energy Society General Meeting (PESGM), 2018, pp. 1–5.

[99] B. S. Southworth, *Necessary Conditions and Tight Two-level Convergence Bounds for Parareal and Multigrid Reduction in Time*, SIAM Journal on Matrix Analysis and Applications, 40 (2019), pp. 564–608.

[100] R. Speck, *Github repository for `pySDC`*. `https://github.com/Parallel-in-Time/pySDC`, Online; accessed December 27, 2022.

[101] ——, *Algorithm 997: PySDC—Prototyping Spectral Deferred Corrections*, ACM Transactions on Mathematical Software, 45 (2019).

[102] R. Speck, M. Knobloch, S. Lührs, and A. Gocht, *Using Performance Analysis Tools for a Parallel-in-Time Integrator*, in Parallel-in-Time Integration Methods, Springer International Publishing, 2021, pp. 51–80.

[103] J. Stoer, R. Bartels, W. Gautschi, R. Bulirsch, and C. Witzgall, *Introduction to Numerical Analysis*, Texts in Applied Mathematics, Springer New York, 2013.

[104] K. Stüben, *An introduction to algebraic multigrid*, in Multigrid, Academic press, 2001, pp. 413–523.

[105] Y. Takahashi, K. Fujiwara, T. Iwashita, and H. Nakashima, *Parallel Finite-Element Method Based on Space-Time Domain Decomposition for Magnetic Field Analysis of Electric Machines*, IEEE Transactions on Magnetics, 55 (2019), pp. 1–4.

[106] Y. Takahashi, T. Tokumasu, K. Fujiwara, T. Iwashita, and H. Nakashima, *Parallel TP-EEC Method Based on Phase Conversion for Time-periodic Nonlinear Magnetic Field Problems*, IEEE Transactions on Magnetics, 51 (2015).

[107] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*, SIAM, 1997.

[108] S. van der Walt, S. C. Colbert, and G. Varoquaux, *The NumPy Array: A Structure for Efficient Numerical Computation*, Computing in Science Engineering, 13 (2011), pp. 22–30.

[109] F. Vasca and L. Iannelli, *Dynamics and Control of Switched Electronic Systems: Advanced Perspectives for Modeling, Simulation and Control of Power Converters*, Advances in Industrial Control, Springer London, 2012.