SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

Evaluation of Higher-Order Coupling Schemes with FEniCS-preCICE

Niklas Vinnitchenko

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

Evaluation of Higher-Order Coupling Schemes with FEniCS-preCICE

Beurteilung von Kopplungsverfahren höherer Ordnung mit FEniCS-preCICE

Author:	Niklas Vinnitchenko
Supervisor:	Prof. Dr. Hans-Joachim Bungartz
Advisor:	M.Sc. (hons) Benjamin Rodenberg
Submission Date:	15.01.2024

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich, 15.01.2024 Niklas Vinnitchenko

Abstract

Multi-physics problems are often simulated with the help of partitioning to reduce the complexity of this task. Partitioning entails the coupling process. This thesis aims to evaluate the waveform iteration as a higher-order coupling scheme experimentally. For that, the theory of an already existing higher-order monolithic solver for the heat equation, implemented with FEniCS and Irksome, is dissected. Additionally, the partitioning process is described, and the idea of the waveform iteration is discussed. To solve the partitioned heat equation, the monolithic solver is modified to be compatible with FEniCS-preCICE. This allows the functionalites of the coupling library preCICE to be conveniently incorporated in the code. As a central technique to assess the accuracy of the computed solution, the method of manufactured solution is applied exemplarily to the heat equation. Conducted experiments focus on time integrators of higher order and waveform degrees of higher degree. They show that the waveform degree correlates with the convergence order of the solution.

Contents

1 Introduction

Water flowing over a heated plate, the beating of a heart, or a tank of a ship filled with fluid on the sea, all of these scenarios are physical processes that combine multiple physical fields. Such phenomena are commonly referred to as multi-physics scenarios or models. [16]
A prevalent method to tackle complex problems is the divide-and-conquer approach. By breaking down the problem into subproblems, it becomes more manageable.
Suppose your goal is to simulate a multi-physics scenario. In that case, a possible idea is to define the subproblems so that already existing software can solve each. This divide-and-conquer approach is known as *partitioning*.
While partitioning simplifies the solving process, the subproblems remain interdependent, meaning that partitioning always entails an additional step called *coupling* that must be considered. In the context of partitioning and coupling, a solver, such as an algorithm, of a subproblem is called *participant*. A distinction can be made between the two coupling types, volume coupling and surface coupling [1]. Yet volume coupling shall be of no interest in this thesis. Using surface coupling requires the assumption that information within the domain of a subproblem is not relevant to other problems. Instead, merely information on the surface of the domain is of interest. It is, therefore, sufficient to only exchange data on the surface, or in other words, on the boundary of the computational subdomain. If two or more subproblems have a common boundary, we call this boundary a coupling boundary.
A concrete example of a time-dependent coupling scheme is the *waveform iteration*, also called waveform relaxation. While the idea of waveform iteration is not new [17], it has great relevance in the domain of solving partitioned problems nowadays. The eponym of this scheme is the *waveform*. The waveform is a time-dependent function defined on each coupling boundary and interpolates time-discrete values provided on such a boundary. Waveforms are solely defined at nodes that are directly on the coupling boundary prescribed by the spatial discretization of the finite element method (FEM). This coupling scheme is, therefore, applicable to time-dependent problems that are numerically solved. [13][17]
This thesis aims to evaluate how the convergence order of a time-dependent partitioned problem behaves if the waveform iteration is used as a coupling scheme, assuming that the FEM is employed for spatial discretization.

The heat conduction equation, or short heat equation, serves as a time-dependent test case. It is solved with the FEM and time integrators of the Runge-Kutta family.
Software components used for solving the heat equation are the partial differential equation (PDE) solving library FEniCS paired with concepts of Irksome to enable easy usage of different time-stepping schemes.
To allow coupling, the computational domain of the heat equation is divided into two subdomains, leading to the partitioned heat equation. preCICE v3, which officially supports coupling with waveforms, is used for coupling. At the time of writing, an official release of version 3 was not published. Therefore, experiments were only carried out with one specific commit on the development branch of the preCICE repository[1].
In the paper [13], tests with moderate waveform degrees and convergence orders of time integrators already show that using the waveform iteration improves the convergence order if it is used instead of a constant interpolant. Thus, more tests are conducted with waveforms of higher degrees and higher-order time-stepping schemes.
After Chapter 2 presents basic concepts for solving partitioned time-dependent PDEs, explained practically oriented with examples, FEniCS, Irksome, and preCICE are introduced in Chapter 3 and Chapter 4 covers crucial aspects of implementing the coupling of the partitioned heat equation. The subsequent Chapter 5 explains how tests are conducted and addresses how the waveform iteration affects the convergence order of the solution.

[1]The commit can be viewed at this link: `https://github.com/precice/precice/pull/1914/commits/a174c66e669d4e0e700679185a577439833e6c4e`

2 Solving higher-order time-dependent partitioned PDEs

This section introduces concepts to solve higher-order time-dependent partitioned PDEs. While assuming that the FEM is used, no introduction to it is provided and the reader is referred to [4]. Rather, this section emphasizes strategies that are necessary explicitly for partitioned and time-dependent PDEs.
Beginning with time discretization, time integrators from the Runge-Kutta family are defined, and one member is exemplarily presented in Section 2.1. To obtain a unique solution of a given PDE, defining the variational formulation of a boundary value problem (BVP) with Dirichlet, Neumann and mixed boundary conditions is introduced subsequently. In Section 2.3, a strategy is developed to solve the heat equation with time-stepping schemes of arbitrary order.
The final topic of this chapter is partitioning (Section 2.4). We exemplarily investigate how the partitioned heat equation is defined and what the coupling procedure looks like. Dirichlet-Neumann coupling and waveform iteration are presented as examples, illustrating their relevance in the context of solving time-dependent partitioned PDEs.

2.1 Runge-Kutta methods

Differential equations that depend on the time derivative of the solution require methods that integrate time. A prominent method for that is the explicit Euler method, which is defined as

$$u_{n+1} = u_n + \delta t f(t_n, u_n),$$

where u_n is the solution of the n-th time step, δt is the time step size, and $f(t, u)$ is the first derivative in time.
However, the explicit Euler method converges only with order $\mathcal{O}(\delta t)$ and has furthermore no desirable stability properties [2]. As a result, it does not suffice to get a higher-order solution.
Thus, different time-stepping methods of higher order are necessary. Ideally, they can be obtained with arbitrary order algorithmically to avoid an implementation by hand. In this regard, the focus is directed exclusively towards Runge-Kutta methods, which

are introduced in the following. Like the explicit Euler method, which is also part of the Runge-Kutta family, Runge-Kutta methods are time integration methods.
An s-stage Runge-Kutta method is defined by

$$u_{n+1} = u_n + \delta t \sum_{i=1}^{s} b_i k_i \tag{2.1a}$$

$$k_i = f(t_n + c_i \delta t, u_n + \delta t \sum_{j=1}^{s} a_{ij} k_j). \tag{2.1b}$$

Every such method can be condensely represented in a Butcher Tableau, which is of the form

$$\begin{array}{c|c} c & \mathcal{A} \\ \hline & b^T \end{array} \tag{2.2}$$

where $b, c \in \mathbb{R}^s$ and $\mathcal{A} \in \mathbb{R}^{s \times s}$.
Solely choosing $\mathcal{A}, b$ and c arbitrarily does not define a Runge-Kutta method of a certain order. During the definition of a Butcher Tableau, the two vectors and the matrix must fulfill specific properties that can be viewed in [2].
From Equation 2.1 and Equation 2.2, we obtain for the explicit Euler method

$$\begin{array}{c|c} 1 & 0 \\ \hline & 1 \end{array}.$$

2.2 The variational form of a boundary value problem

PDEs can have multiple solutions, such as the Poisson equation

$$-\Delta u = f \text{ in } \Omega, \tag{2.3}$$

where Ω is the domain in which the equation is defined. Only formulating a PDE on some domain is usually insufficient to obtain a unique solution.
When seeking a unique solution that satisfies specific requirements, adding boundary conditions resolves the problem of non-uniqueness. The problem statement of finding the unique solution with an equation and boundary conditions on the boundary $\partial\Omega$ of Ω is called BVP. Different types of boundary conditions exist, but only the Dirichlet and Neumann boundary conditions are of interest in this thesis. [4]

2.2.1 Formulating a boundary value problem

This section aims to formulate Equation 2.3 as a BVP whose solution is u in Ω.
The first type of boundary condition is the Dirichlet boundary condition, often referred to as the essential boundary condition that prescribes a value of the unknown solution at $\partial\Omega$. A correctly defined boundary value problem with Dirichlet boundary conditions requires, therefore, additionally

$$u = g \text{ on } \partial\Omega, \tag{2.4}$$

where g is some function defined on $\partial\Omega$ prescribing one solution of u. The definition of the Poisson equation as a BVP would be

$$\begin{aligned} -\Delta u &= f \text{ in } \Omega \\ u &= g \text{ on } \partial\Omega. \end{aligned}$$

Neumann boundaries, called natural boundary conditions, prescribe the derivative normal to $\partial\Omega$ of u at the boundary $\partial\Omega$.

$$\frac{\partial u}{\partial n} = h \text{ on } \partial\Omega, \tag{2.5}$$

where h is some function defined on $\partial\Omega$.
Regarding Chapter 5, mixed boundary conditions are also important to look at. If a boundary value problem has mixed boundary conditions, different sorts of conditions are applied to different parts of the boundary.
Let for example Γ_1 and Γ_2 be a partition of $\partial\Omega$, i.e. $\Gamma_1 \cup \Gamma_2 = \partial\Omega$ and $\Gamma_1 \cap \Gamma_2 = \emptyset$, then mixed boundary conditions look exemplarily like

$$\begin{aligned} u &= g \text{ on } \Gamma_1 \\ \frac{\partial u}{\partial n} &= h \text{ on } \Gamma_2 \end{aligned}.$$

Both g and h are defined identically to the functions for the Dirichlet and Neumann boundary conditions. The boundary Γ_1 is thus a Dirichlet boundary and Γ_2 a Neumann boundary. [4]

2.2.2 Formulation of the variational form

The variational equation, often referred to as the weak form of a BVP, is the basis for deriving the FEM.[4] Formulating the variational form is a central aspect when solving a BVP with the FEM, which can also be seen in the software used later in Chapter 3. The software requires solely the weak form of a PDE and the boundary conditions from

the user to compute the solution. This motivates the introduction of the variational equation.
Consider Equation 2.3 with the Dirichlet boundary $g = 0$. The sought-after solution u is expected to be twice differentiable in all spatial directions within Ω. To deduce a problem's variational formulation, we first multiply a test equation v of the test function space V on both sides and then take the integral over the domain. [4]
Green's first identity

$$-\int_\Omega v\Delta u \,\mathrm{d}x = \int_\Omega \nabla v \cdot \nabla u \,\mathrm{d}x - \int_{\partial\Omega} v\frac{\partial u}{\partial n}\,\mathrm{d}s$$

leads to the equation

$$\int_\Omega \nabla v \cdot \nabla u \,\mathrm{d}x - \int_{\partial\Omega} v\frac{\partial u}{\partial n}\,\mathrm{d}s = \int_\Omega fv \,\mathrm{d}x.$$

The test function v can be chosen such that v vanishes on $\partial\Omega$. Thus, you would arrive at

$$\int_\Omega \nabla v \cdot \nabla u \,\mathrm{d}x = \int_\Omega fv \,\mathrm{d}x$$

which is called the variational formulation of the Poisson equation.
The equation prescribes that f only needs to be integrable and u not twice differentiable anymore - the prerequisites that were posed in the equation before are, therefore, weakened. [4]

2.3 Heat conduction equation

The heat equation is given by

$$\begin{aligned} \frac{\partial u}{\partial t} - \Delta u &= f \text{ in } \Omega \times (0, T], \\ u &= u_0 \text{ at } t = 0, \end{aligned} \tag{2.6}$$

where $\Omega \subseteq \mathbb{R}^d$ with $d \in \{1, 2, 3\}$, $T \in \mathbb{R}_+$ and u_0 is the initial value. [7] Defining the boundary conditions is omitted for now, but Equation 2.6 is regarded as a BVP in the following. Without any assumptions regarding the boundaries, the subsequent explanations can be done in a more general manner.
The heat equation is chosen as it is a rather trivial PDE that is time-dependent and thus suitable to evaluate the waveform iteration later. Before the testing, it is used to exemplarily show how the variational form can be defined with arbitrary time-stepping schemes and what a partitioned equation looks like.

2.3.1 Time integration of the heat equation

Following the approach as in Subsection 2.2.2, the variational form of Equation 2.6 is given by

$$\int_\Omega \frac{\partial u}{\partial t} v \,\mathrm{d}x + \int_\Omega \nabla u \nabla v \,\mathrm{d}x - \int_{\partial\Omega} v \frac{\partial u}{\partial n} \,\mathrm{d}s = \int_\Omega f v \,\mathrm{d}x. \tag{2.7}$$

To deduce time integration formulas for both inhomogeneous boundaries of a single type and mixed boundary conditions, specifications for the test functions, such as that they vanish at the boundary, are postponed.
Compared to the Poisson equation, the solution of Equation 2.6 is also dependent on the time derivative. As this is an additional unknown, it is necessary to eliminate it to solve the equation. There are two ansatzes which can be used to resolve the problem. The first is to replace $\frac{\partial u}{\partial t}$ with a rearranged time integration scheme. For instance, use the implicit or backward Euler given by

$$u_{n+1} = u_n + \delta t f(t_{n+1}, u_{n+1})$$

and solve it for the time derivative $f(t_{n+1}, u_{n+1})$ of u at the time t_{n+1} according to this scheme. In this case, the rearranged equation is $f(t_{n+1}, u_{n+1}) = \frac{u_{n+1} - u_n}{\delta t}$. Bear in mind that each time-stepping scheme describes the time derivative differently. Consequently, the resulting weak form differs from the following, assuming a different time-stepping scheme is used. After inserting the time derivative from the implicit Euler, Equation 2.7 becomes

$$\int_\Omega \frac{u_{n+1} - u_n}{\delta t} v \,\mathrm{d}x + \int_\Omega \nabla u_{n+1} \nabla v \,\mathrm{d}x - \int_{\partial\Omega} v \frac{\partial u_{n+1}}{\partial n} \,\mathrm{d}s = \int_\Omega f(t_{n+1}, \cdot) v \,\mathrm{d}x. \tag{2.8}$$

Replacing u with u_{n+1} and evaluating the right-hand side f of the heat equation at t_{n+1} is necessary because the inserted time derivative is defined for the end of the current time step, so $t_{n+1} = t_n + \delta t$. [3] Note that the normal vector of u_{n+1} is no unknown because the integral is only taken at the boundary, and therefore, the value can be determined with the boundary conditions.
Albeit this works fine with most explicit and simple implicit time-stepping methods, the definition of the variational form with implicit multi-stage methods is complex and error-prone.
And, due to the stiffness of the heat equation [3], one cannot expect useable results from the computations when using explicit time integration schemes. They do not fulfill stability properties that are required to solve equations such as the heat conduction equation numerically stably [2]. Hence, this approach is not expedient for getting higher-order solutions.

The second approach from [3] has a fundamentally different idea of dealing with the time derivative. Instead of replacing it, this method replaces u such that the sole unknown is the time derivative of u. As an example, a general two-stage Runge-Kutta method shall be used to show how this approach works.

$$\begin{array}{c|cc} c_1 & a_{11} & a_{12} \\ c_2 & a_{21} & a_{22} \\ \hline & b_1 & b_2 \end{array} \tag{2.9}$$

The Butcher Tableau of an arbitrary two-stage method is given in 2.9. For Runge-Kutta methods, the stages k_i represent time derivatives of the solution at different times, namely $t + c_i \delta t$. An equation must be formulated for each stage to assemble the discrete evolution according to Equation 2.1a. In that sense, a system of equations is defined. For 2.9, we get two equations for the two stages, k_1 and k_2.

$$\begin{aligned} \int_\Omega k_1 v_1 \,\mathrm{d}x + \int_\Omega \nabla(u_n + \delta t \sum_{j=1}^{s} a_{1j} k_j) \nabla v_1 \,\mathrm{d}x \\ - \int_{\partial\Omega} v_1 \frac{\partial u}{\partial n}(t_n + c_1 \cdot dt) \,\mathrm{d}s = \int_\Omega f(t_n + c_1 \delta t, \cdot) v_1 \,\mathrm{d}x \end{aligned} \tag{2.10a}$$

$$\begin{aligned} \int_\Omega k_2 v_2 \,\mathrm{d}x + \int_\Omega \nabla(u_n + \delta t \sum_{j=1}^{s} a_{2j} k_j) \nabla v_2 \,\mathrm{d}x \\ - \int_{\partial\Omega} v_2 \frac{\partial u}{\partial n}(t_n + c_2 \cdot dt) \,\mathrm{d}s = \int_\Omega f(t_n + c_2 \delta t, \cdot) v_2 \,\mathrm{d}x \end{aligned} \tag{2.10b}$$

The u and f are replaced similarly to the first ansatz. That means that the right-hand side is evaluated at the appropriate stage time and the variable u is replaced by $u_n + \delta t \sum_{j=1}^{s} a_{ij} k_j$ which is exactly the definition of u in this stage (see Equation 2.1b). Adding Equation 2.10a and Equation 2.10b leads to the weak form of the heat equation that uses a generic two-stage Runge-Kutta scheme. Mind that each equation has a different set of test functions $v_i \in V_i$.

A closer look at the two equations reveals that their structure is identical because the structure is defined by the variational form of the non-discretized PDE. If the weak form of a given equation is determined, formulating each equation follows the same scheme, namely by plugging in the definition of each stage as in our example. While the user is still responsible for the first part, the definition of the equations can be done by an algorithm. Consequently, using multi-stage methods implies neither more work nor higher complexity for the implementation.

The first presented approach is disregarded in the rest of this thesis, as this flexibility to get the weak form for different time-stepping schemes is not given.

2.3.2 Time integrator dependent boundary conditions

In the previous section, only the deduction of the weak form of Runge-Kutta time integrators is presented, while formulating the boundary conditions was skipped. As imposing the conditions is not as straightforward as shown in Subsection 2.2.1, this short section is dedicated to explaining how the different types of boundary conditions are handled.
In both equations of Equation 2.10, the term $\int_{\partial\Omega} \frac{\partial u}{\partial n}(t_n + c_i\delta t)v_i \, \mathrm{d}s$ explicitly defines the Neumann boundary conditions 2.5, if the directional derivative and v_i are nonzero. So, the Neumann conditions can be imposed directly using the equations deduced in the second ansatz of Subsection 2.3.1. [3]
For Dirichlet boundaries, we cannot use Equation 2.4 directly even though Equation 2.10 is deduced from the heat equation. Instead, the equation of the stage k_i demands the time derivative of u at the respective stage time $t_n + c_i\delta t$ as the Dirichlet boundary condition. Thus, we get as the Dirichlet boundary condition

$$\frac{\partial u}{\partial t}(t_n + c_i\delta t, \cdot) = \frac{\partial g}{\partial t}(t_n + c_i\delta t, \cdot) \text{ on } \partial\Omega. \tag{2.11}$$

Additionally, the terms from Equation 2.10 that describe the natural boundary conditions are dropped.
Mixed boundary conditions are imposed similarly as described in Subsection 2.2.1. If Γ_1 is a Dirichlet boundary, Equation 2.11 is only applied on Γ_1 instead of $\partial\Omega$ and if Γ_2 is a Neumann boundary, the integral $\int_{\partial\Omega} \cdot \, \mathrm{d}s$ from Equation 2.10 is the integral over Γ_2 instead, so $\int_{\Gamma_2} \cdot \, \mathrm{d}s$. [3]
Taking a closer look at how Dirichlet and Neumann boundary conditions are imposed is crucial in this context. Because it is a fallacy to assume that Neumann boundary values represent gradients of the time derivative of u just because Dirichlet boundary values represent time derivatives of u.

2.4 Partitioned heat conduction equation

The first step necessary to get to a partitioned PDE setup is partitioning the domain into at least two subdomains. Regarding the complexity, we dissect Ω only into two domains $\Omega_{\mathcal{D}}$ and $\Omega_{\mathcal{N}}$ as seen in Figure 2.1 and assume that Dirichlet boundaries are applied on $\partial\Omega$.
After the partitioning, the heat equation is solved on the two subdomains instead of Ω. Additional prescribed boundary conditions on the two new boundaries Γ_D and Γ_N that emerged during the partitioning are not allowed. They must be defined by the

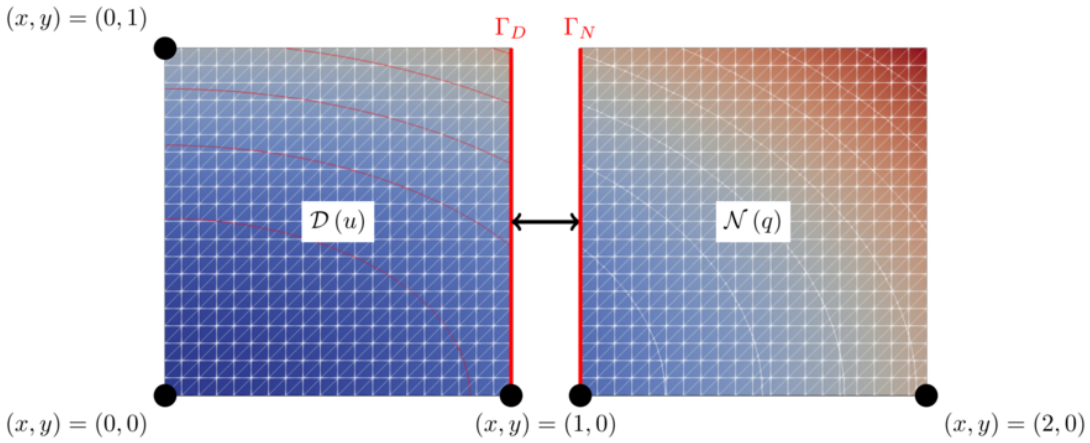

Figure 2.1: An exemplary partitioned setup on the domain $\Omega = [0,2] \times [0,1]$ with $\Omega_{\mathcal{D}} = [0,1] \times [0,1]$ and $\Omega_{\mathcal{N}} = [1,2] \times [0,1]$. In this case, Γ_D is a Dirichlet boundary, and Γ_N Neumann boundary accordingly. The figure is taken from [13].

solution that lives in the two subdomains. The data dependency between $\Omega_{\mathcal{D}}$ and $\Omega_{\mathcal{N}}$ is overcome by coupling.

2.4.1 Coupling

Many different coupling approaches and configurations exist. Implicit or explicit, parallel or serial, unidirectional or bidirectional, and surface or volume coupling schemes can be used. [1]
Not all schemes are presented, as only one configuration is utilized throughout the rest of the work. Omitted coupling schemes can be looked up in [1]. Which coupling scheme is used eventually is defined by the problem we want to solve and the boundary conditions at Γ_D and Γ_N [1].
For the partitioned heat equation, we will use Dirichlet boundary conditions on Γ_D, referred to as Dirichlet boundary, and Neumann boundary conditions on Γ_N, referred to as Neumann boundary. This specification leads to the Dirichlet-Neumann coupling, predestined for black-box coupling. Defining each participant as a black box is desirable because it minimizes the information each participant must provide. [6]

2.4.2 Dirichlet-Neumann coupling

The Dirichlet-Neumann coupling is an iterative coupling scheme shown in Algorithm 1. It is a slightly modified version of [10], which uses relaxation factors that depend on the current iteration. As currently, only constant under-relaxation is possible in preCICE with the coupling scheme we eventually use, this acceleration scheme is used in the presented algorithm. Additionally, a stopping criterion is introduced, imbued by a coupling scheme from [13] that defines a time-dependent Dirichlet-Neumann coupling, which is presented in Subsection 2.4.3.
Algorithm 1 is a serial and bidirectional coupling scheme because first, the Dirichlet participant must solve the problem (lines three to five), and only afterward the Neumann participant is allowed to compute the solution (lines six and seven). It is also bidirectional because the Dirichlet participant uses the results from the previous iteration of the Neumann problem, which is embedded into g_k. This coupling scheme is also implicit. After initialization, it reuses previous results to refine the current result, so it employs bootstrapping until some convergence measure is reached. In Figure 2.2, such an implicit, bidirectional, and serial scheme is generally depicted for two participants. $\mathcal{A}$ is, in the case of Algorithm 1 constant under-relaxation.
Recall that the heat equation is a parabolic PDE, which leads to the question of how boundary conditions for time-stepping schemes are obtained at the coupling surface during the coupling process. The remedy is to modify the Dirichlet-Neumann coupling to be compatible with time-dependent problems, resulting in the waveform iteration.

Algorithm 1 Dirichlet-Neumann coupling with constant under-relaxation [10][13]

1: Let g_0 at Γ_D be chosen appropriately.
2: Set k=0.
3: Solve the Dirichlet problem:
4: Get the solution $u_{\mathcal{D},k}$ on $\Omega_{\mathcal{D}}$ with boundary conditions g_k and the given boundaries at $\partial\Omega_{\mathcal{D}}$.
5: Compute $h_k = \frac{\partial u_{\mathcal{D},k}}{\partial n}$ with n directing in $\Omega_{\mathcal{N}}$ on the coupling surface Γ.
6: Solve the Neumann problem:
7: Get the solution $u_{\mathcal{N},k}$ on $\Omega_{\mathcal{N}}$ with boundary conditions h_k and the given boundaries at $\partial\Omega_{\mathcal{N}}$.
8: $g_{k+1} = \theta u_{\mathcal{N},k}|_{\Gamma_N} + (1-\theta)g_k$, with $\theta \in (0,1]$
9: **if** $\|g_{k+1} - g_k\|$ smaller than some tolerance **then**
10: return the solution $u_{\mathcal{N},k} \cup u_{\mathcal{D},k}$
11: **else**
12: Continue on line three with $k = k+1$
13: **end if**

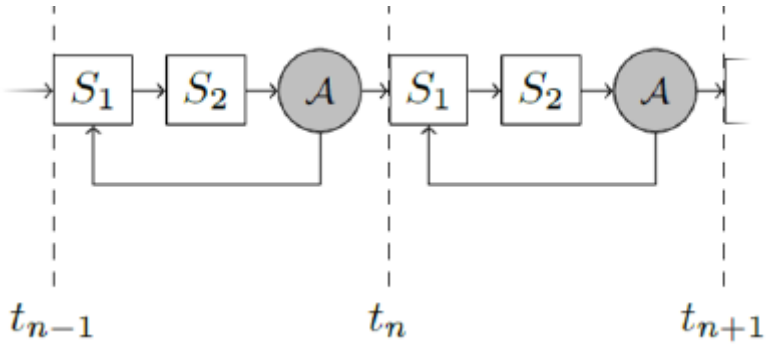

Figure 2.2: A schematic data flow of a serial bidirectional implicit coupling scheme. S_1 and S_2 are the participants that are coupled. First, S_1 is computed, and then S_2 (serial). S_2 uses results from S_1 and S_1 uses data from S_2 after applying the acceleration scheme $\mathcal{A}$ (bidirectional and implicit). $\Delta t = t_n - t_{n-1} = t_{n+1} - t_n$ and the dashed lines represent times at which S_1 and S_2 are coupled. [1]

2.4.3 Waveform iteration

To adequately explain the Dirichlet-Neumann waveform iteration, conditions and variables present in a problem to which the waveform iteration can be applied are first defined.
It makes sense to use the waveform iteration for time-dependent problems only, leading to the first two assumptions that the problem's evolution in the time interval $[0, T]$ with granularity Δt is sought. The value of Δt defines the frequency with which the solutions of the participants are synchronized, or in other words, coupled. The time interval is also called the time window, and T shall be an integer multiple of it. Each participant must deliver a value at the end of each time window.
In each time window, the partitioned heat equation is solved with a given time-stepping scheme where δt is not confounded with Δt. The time step size δt is defined for each participant and can be chosen independently of the coupling scheme. If $\delta t < \Delta t$, we speak from subcycling, i.e., if one participant computes more than one time step within one time window. If the time step size of the Neumann participant δt_N and of the Dirichlet participant δt_D are not equal additionally, this is called multirate time-stepping. For δt_N and δt_D, Δt is not required to be an integer multiple of them. As the reduction of δt will not impair the accuracy of the result, δt can be chosen smaller if necessary by $\delta t = min\{n \cdot \Delta t - t, \delta t\}$, where $(n-1) \cdot \Delta t \leq t < n \cdot \Delta t$; so the current simulation time t is in the n-th time window.

The idea of the waveform iteration is to provide boundary conditions that are interpolated in time within a time window. Providing such boundaries is motivated by time integrators that not exclusively use values at the beginning and the end of a time window, where the values are given due to the coupling process, but in between.
The interpolant of the time-dependent boundary conditions is called waveform. It is defined piecewise in one time window and for each participant. So Γ_D and Γ_N have independently defined waveforms. Interpolation conditions the waveform must fulfill are the value at the initial time of the current time window and the results from each time step. Waveforms of higher order can be created if at least subcycling is used. Participants then provide more data points of u at the boundary, leading to more interpolation conditions the waveform must fulfill.
A time-stepping scheme that requires boundary conditions between coupling times can then sample the waveform at arbitrary times of the current window. [13]
Algorithm 2 is not as intricately presented as in [13] because, for a compact representation, it is necessary to introduce a set of notations that would be beyond the scope of this thesis. Instead, the algorithm is didactically broken down into a higher-level algorithm. Additionally, the constant under-relaxation taken from [9], is directly included here unlike in the algorithm from [13].
Comparing Algorithms 1 and 2, the structure of both is only marginally different. Both must first solve the Dirichlet and then the Neumann problem; the stopping criterion and the acceleration scheme are also identical. Significant differences are the necessary time-stepping and the computation of the waveforms.
Be aware that the Dirichlet participant uses the waveform defined by the Neumann participant from the previous iteration, and the coupling happens during the time-stepping for both participants as they sample c_D^k and c_N^k that the respective other participant determines.

Algorithm 2 Dirichlet-Neumann coupling with waveform iteration [9][13]
Lines marked with * are only valid in the first time window. For every other window, the algorithm uses for c_D^0 a constant interpolant with the value of the end of the previous time window, $u_{D,0}^0$ and $u_{N,0}^0$ are the solutions at the end of the previous time window. t_{ini} is the initial time of the current time window.

1: Let c_D^0 at Γ be an initial guess*.
2: Let $u_{D,0}^0$ and $u_{N,0}^0$ be given as the initial condition*.
3: Set $k = 0$.
4: Solve the Dirichlet problem with n_D time steps, $u_{D,0}^k$ and c_D^k:
5: **for** $i \in \{1, ..., n_D\}$ **do**
6: Solve Dirichlet Problem at $t = t_{ini} + i \cdot \delta t_D$ and get $u_{D,i}^k$.
7: **end for**
8: From $(u_{D,1}^k, u_{D,2}^k, ..., u_{D,n_D}^k)$, retrieve interpolation variables $(c_{N,1}^k, c_{N,2}^k, ..., c_{N,n_D}^k)$, where $c_{N,i}^k = \frac{\partial u_{D,i}^k}{\partial n}, \forall i \in \{1, ..., n_D\}$
9: Interpolate with $(c_{N,1}^k, c_{N,2}^k, ..., c_{N,n_D}^k)$ to get the waveform c_N^k.
10: Solve the Neumann problem with n_N time steps, $u_{N,0}^k$ and c_N^k:
11: **for** $i \in \{1, ..., n_N\}$ **do**
12: Solve Neumann Problem at $t = t_{ini} + i \cdot \delta t_N$ and get $u_{N,i}^k$.
13: **end for**
14: From $(u_{N,1}^k, u_{N,2}^k, ..., u_{N,n_N}^k)$, retrieve interpolation variables $(c_{D,1}^k, c_{D,2}^k, ..., c_{D,n_N}^k)$
15: Interpolate with $(c_{D,1}^k, c_{D,2}^k, ..., c_{D,n_N}^k)$ to get the waveform $\tilde{c}_D^{k+1}$.
16: $c_D^{k+1} = \theta c_D^k + (1 - \theta)\tilde{c}_D^{k+1}$, with $\theta \; in (0, 1]$
17: **if** $\|c_D^{k+1}(t) - c_D^k(t)\|$ smaller than some tolerance **then**
18: return the solution $u_{N,k} \cup u_{D,k}$
19: **else**
20: Continue on line four with $k = k + 1$
21: **end if**

In Figure 2.3, the idea of waveforms is visualized. The scenario on the right side shows the effect of waveforms quite clearly. Both waveforms of $\mathcal{D}$ and $\mathcal{N}$ use the solutions which are computed with time step sizes $\delta t_{\mathcal{D}}$ and $\delta t_{\mathcal{N}}$ as interpolation conditions. Comparing the two waveforms, an obvious observation is that a more refined time-dependent boundary can be obtained by increasing the number of intermediate solutions.
If no waveform is used, the left scenario of Figure 2.3 arises. Only the last value of the time window is used as a reference, while the remaining results within the time window are ignored, resulting in a worse representation of the time-dependent result.

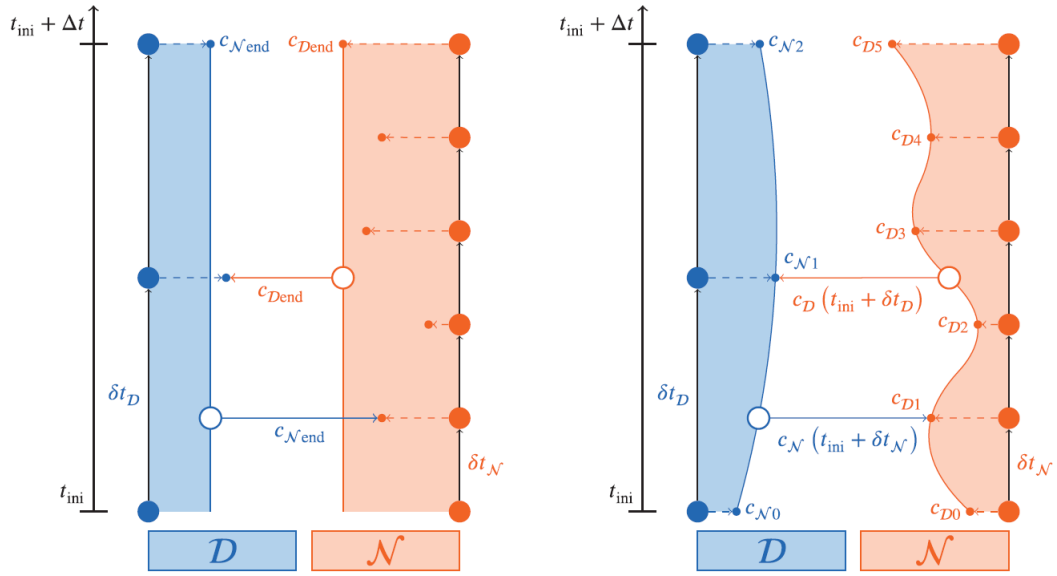

Figure 2.3: The two visualizations show the difference between a constant time interpolant and the waveform. On the left side, $\mathcal{D}$ and $\mathcal{N}$ no waveform is used, resulting in a constant interpolant of the boundary conditions.
On the right side, the waveform interpolates all results of the participants, resulting in a preferable representation of the boundary conditions. The number of intermediate steps depends on the definition of the time step sizes $\delta t_{\mathcal{D}}$ and $\delta t_{\mathcal{N}}$. [13]

3 Software

After introducing the necessary details to solve or implement higher-order partitioned partial differential equations, the software and its core features used for the code in this thesis are introduced.
FEniCS provides the functionality of solving PDEs automatically, and Irksome is a part of Firedrake, which is similar to FEniCS and is responsible for time discretization of time-dependent PDEs. As FEniCS does not provide time discretization methods and is used in the code of this thesis, some functionalities of Irksome are embedded into FEniCS code.
preCICE is the software component that couples partitioned solutions produced by FEniCS.
This section also serves the purpose of introducing the basic application programming interface (API) that is provided by the preCICE adapter FEniCS-preCICE used in the implementation later.

3.1 FEniCS

Information of the section is taken from [7].
FEniCS is an open-source software that is primarily a Python library that simplifies solving PDEs for users.
It consists of a collection of libraries itself and offers functionalities that cover most of the simulation pipeline, reaching from mesh creation to outputting the computed results into files. Computing solutions of PDEs with the FEM requires, therefore, in most cases, only the FEniCS project. Two of the most important packages of FEniCS are the DOLFIN and the UFL packages. Some of the functionalities required for FEM will be discussed in the following.
UFL, or Unified Form Language, provides operators and atomic expressions to define weak forms in code. The code representation differs only marginally from the mathematically correct representation so that it can be easily understood and written. For instance, the weak form of the Poisson equation

$$\int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}x - \int_\Omega f v \, \mathrm{d}x = 0$$

can be represented in UFL as

```
F = dot(grad(u), grad(v)) * dx - f * v * dx
```

DOLFIN is the main interface for the programmer. It is a wrapper for the functionalities of the other packages and represents the core software component as it is responsible for the communication among most of the packages and provides central data structures which are needed for using the FEM. Some of the data structures are meshes, function spaces, expressions, and boundary conditions.
FEniCS also offers an API that directly determines the solution of the weak form for given boundary conditions. The function `solve()` only demands the variational form defined as a UFL expression and the boundary conditions and returns the solution. The user can, therefore, conveniently string together functions that FEniCS provides to solve PDEs with FEM without needing to know the mathematical background of the FEM.

3.2 Irksome

Irksome is an open-source package of Firedrake. Firedrake is another project, similar to FEniCS, which allows the user to solve PDEs with the FEM. [5] The package is responsible for the time discretization, i.e., if you solve a time-dependent PDE with Firedrake, Irksome provides a convenient operator extending the UFL notation that takes care of time-stepping.
Extending the weak form from above with a time derivative, the resulting equation is

$$\int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}x - \int_\Omega f v \, \mathrm{d}x + \int_\Omega \frac{\partial u}{\partial t} v \, \mathrm{d}x = 0.$$

With Irksome and UFL, it can be easily represented as

```
F = dot(grad(u), grad(v)) * dx - f * v * dx + Dt(u)*v*dx
```

Such a simple modification is not possible when using FEniCS. Irksome modifies the weak form later according to a given time-stepping scheme, eventually replacing the `Dt` operator. [3] This modification is already described in the second ansatz of Subsection 2.3.1 that must be done by hand in FEniCS code. A set of time-stepping schemes from the Runge-Kutta family, such as Gauss-Legendre (GL), LobattoIIIC (LIIIC), and RadauIIA (RIIA) with arbitrary many stages, can be obtained from the `ButcherTableau` class of the Irksome package.
As in this thesis, Firedrake is not used, we cannot use Irksome for the time discretization as is, but we only use the class `ButcherTableau`[1] to conveniently use different

[1] In the GitHub repository of Irksome (`https://github.com/firedrakeproject/Irksome`), the class can be found in the file "irksome/ButcherTableaux.py".

classes of time-stepping schemes. Due to that, the code is versatilely applicable as the time-stepping scheme can be quickly changed by modifying a single line.

3.3 preCICE and its ecosystem

After the introduction of the software components which are needed for solving PDEs with the FEM, the last remaining component to solve partitioned PDEs shall enable the communication between subdomains.
This is what preCICE, or Precise Code Interaction Coupling Environment, is responsible for. It is an open-source coupling library for partitioned multi-physics simulations,

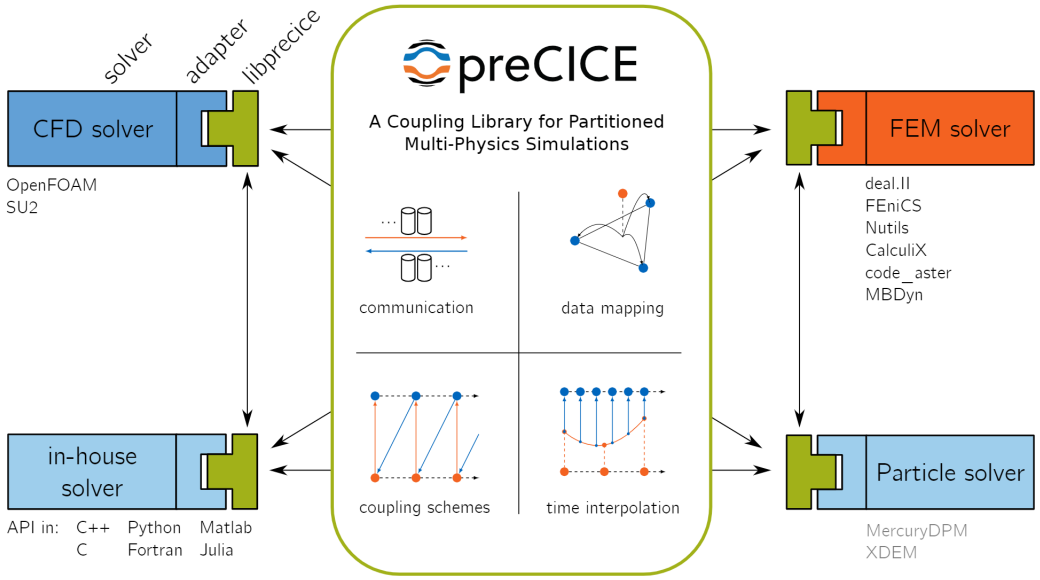

Figure 3.1: An overview about the core functionalities offered by preCICE and the general data flow. The diagram is taken from [1].

meaning it is the communication interface of two or more participants that together solve a multi-physics problem. The following paragraphs summarize the description of the functionalities of preCICE from [1] and give a concise overview of the most important API for the coupling process that is provided by the FEniCS-preCICE adapter. All participants are treated as black boxes, so neither preCICE nor other solvers can access internal details about a solver, such as spatial and time discretization. Core functionalities are, as seen in Figure 3.1, the communication, data mapping, coupling schemes, and time interpolation. Some of the functionalities can be configured to the needs of the user to achieve the best possible solution.

Adapters

Adapters provide a high-level API for solvers to communicate with preCICE. They define an intermediate software layer between preCICE and the solver itself that simplifies the incorporation of the coupling library. Therefore, participants that use solvers compatible with existing adapters interact only indirectly with the coupling library through an instance of the adapter class. One of the adapters is FEniCS-preCICE. As the name already suggests, this adapter is compatible with FEniCS solvers. It is used throughout the entire thesis, and most information required for this adapter is obtainable from [12].
If no such adapter exists for a used solver, the low-level API of preCICE can be directly used instead.

Communication

Communication comprises the sending and receiving of data to and from participants. As the participants are treated as black boxes, preCICE is unaware of what data each participant demands. Thus, participants provide preCICE data whenever they can and request data from preCICE whenever they need it. So, the coupling library offers an API for the communication.
The function `read_data(dt)` returns the value at the boundary relative to the current time window. If t is the initial time of the current time window, `read_data(dt)` returns the value of all FEM nodes at the coupling boundary at the time $t + dt$. preCICE takes the values from the waveform of the respective other participant, according to Algorithm 2.
Participants update the waveform by sending data to preCICE with the function `write_data(val)`, where `val` represents the computed solution of the participant of the whole spatial domain. `val` is used as the interpolation condition for the waveform at the current coupling time. The coupling time can be updated independently by each participant with `advance(dt)`. Calling this function advances the current coupling time by `dt` and does not influence the simulation time of other participants as they might use different time step sizes.

Coupling schemes and time interpolation

For coupling, the adapter provides the `coupling_expression` data structure. Each coupling boundary requires one coupling expression and can be initialized with `create_coupling_expression()`. This method returns a coupling expression for the coupling boundary. It is used as the boundary condition for the coupling boundary and is updated with `update_coupling_expression(coupling_expression, data)`, where

data is obtained by calling `read_data`.
In addition to the coupling expressions, the configuration file `precice-config.xml` exists that defines, among other things, the exchanged data during coupling and the coupling scheme that shall be used.
Listing 3.1 defines the waveform iteration as seen in Algorithm 2.

```
<data:scalar name="Temperature" waveform-degree="2" />
<data:scalar name="Heat-Flux" waveform-degree="2" />
        ...
<coupling-scheme:serial-implicit>
    <participants first="Dirichlet" second="Neumann" />
      <exchange ... from="Dirichlet" to="Neumann" ... />
      <exchange ... from="Neumann" to="Dirichlet" ... />
              ...
      <acceleration:constant>
      <relaxation value="0.375" />
    </acceleration:constant>
</coupling-scheme:serial-implicit>
```

Listing 3.1: This listing presents crucial sections of the preCICE configuration file to define the waveform iteration as a coupling scheme. In this case, a serial implicit bidirectional scheme that uses constant under-relaxation as the acceleration scheme is defined. The two participants are called "Dirichlet" and "Neumann" and the two values that are exchanged should both be represented as waveforms of degree two.

In order to configure preCICE properly, it is vital to categorize the coupling scheme used correctly.
Additional noteworthy attributes are `relative-convergence-measure` tol_{conv} which defines the stopping criterion of the coupling scheme, `max-time`, which defines the end time of the coupling and `time-window-size`, which defines the time window size Δt. After reaching the coupling time `max-time`, preCICE won't couple the participants anymore. While participants have no knowledge about this value, they can test if this time is reached by calling `is_coupling_ongoing()`. tol_{conv} is used as a value for the condition in line 17 of Algorithm 2.

Data mapping

Suppose participants have different representations of their data or use, in the case of the FEM, different meshes. In this case, preCICE modifies the representation of the data in such a way that it exactly corresponds to the requirements of each participant, and thus the user does not need to mind the compatibility of two solvers. This would be an example of data mapping. Users of the coupling library can again configure a specific data mapping approach by modifying the configuration file.

4 Implementation

This section serves the purpose of showing how an implementation of code looks like that can solve partitioned PDEs with higher-order using the software introduced in Chapter 3.
Again, the heat equation is chosen as an instructive example.
The presentation of the implementation starts with two ideas on how to numerically compute the derivative of a function whose term is unknown and that can be sampled only (Section 4.1). Computing the time derivative is necessary because the second ansatz from Subsection 2.3.1 requires time derivatives at the boundary. Still, the Dirichlet-Neumann waveform iteration uses the heat flux and the temperature, not the heat flux and the time derivative of the temperature. preCICE provides, therefore, only a waveform that interpolates the heat flux and the temperature in time, respectively. Both a finite difference approach and an ansatz where the unknown function is interpolated with samples of the unknown function are discussed.
Implementation details on how the heat equation can be solved with higher-order time-stepping schemes in FEniCS are omitted as it is scrutinized extensively in [18]. Instead, in Section 4.2, differences are highlighted between the code that solves the monolithic heat equation and the code that solves the partitioned heat equation with the waveform iteration. This includes how boundary conditions are determined on the coupling boundaries, how to modify preCICE such that waveforms of arbitrary degree can be used, and how the original simulation loop from [18] is modified to use preCICE as a coupling library.[1]

4.1 Time derivatives

This section solely serves the purpose of explaining how the derivative of preCICE's waveform can be obtained, as an API to retrieve it directly from preCICE is not implemented now. Therefore, the methods to get the derivatives are only a temporary workaround until this functionality is implemented.[2] Hence, in the following, two ideas are presented concisely.

[1] The implementation can be examined here: `https://github.com/precice/tutorials/pull/415`.
[2] The current development status can be viewed at `https://github.com/precice/precice/issues/1908`.

4.1.1 Finite differences

Using finite differences is a convenient way to approximate arbitrary derivatives of a function of arbitrary order. To determine the i-th derivative $f^{(i)}$ of some function f, with order $2N$ and equidistant sampling points of f, the sampling points are weighted with a coefficient. Afterward, the sum of all weighted function values determines the derivative.
The starting point to find the coefficients is to write the weighted sum of the derivative with the unknown coefficients c_k.

$$f^{(i)}(x) = \sum_{i=-N}^{N} c_i f(x + i \cdot h), \tag{4.1}$$

where h is the distance between two sampling points. As you can see, this ansatz uses the central difference scheme, but in the generalized case, the forward and backward difference schemes can also be applied.
The next step necessary to obtain c_i is to expand $c_i f(x + i \cdot h)$ for all $i \in -N, ..., N$ with the Taylor expansion up $\mathcal{O}(h^{2N+1})$. If the i-th derivative is sought, all other derivatives of f must vanish in the Taylor expansion, resulting in the following linear system of equations:

$$\begin{pmatrix} s_{-N}^0 & \dots & s_N^0 \\ \vdots & & \vdots \\ s_{-N}^{2N-1} & \dots & s_N^{2N} \end{pmatrix} \cdot \begin{pmatrix} c_{-N} \\ \vdots \\ c_N \end{pmatrix} = h^{-i} \begin{pmatrix} 0 \\ \vdots \\ i! \\ \vdots \\ 0 \end{pmatrix}, \tag{4.2}$$

where s_k^l is the l-th coefficient of the Taylor expansion at $f(x + k \cdot h)$.
The right-hand side for the n-th equation is defined by $\frac{i!}{h^i}\delta(n - i)$, where $\delta(\cdot)$ is the Kronecker delta. [15]
After computing the result of Equation 4.2, plugging it into Equation 4.1 leads to the sought-after derivative.
Let $[a, b]$ with $a, b \in \mathbb{R}$ be the domain where f is defined.
The central difference quotient cannot be used if the derivative of f is sought at either a or b. The remedy is to use forward divided differences for the derivative evaluation at a, so the ansatz of Equation 4.1 becomes,

$$f^{(i)}(x) = \sum_{i=0}^{2N} c_i f(x + i \cdot h),$$

For an evaluation at b, the backward variant can be used, so compute $\sum_{i=-2N}^{0}$ instead. However, the general structure of Equation 4.2 stays the same.

Implementation

The code of this ansatz is straightforward. After defining the order, the matrix and the right-hand side can be determined. A linear algebra library, like the `linalg` package of `scipy`, incurs the solving process. Once the linear system is solved, the coefficients can be reused throughout the entire program execution. If a derivative should be computed, it is merely necessary to insert the corresponding sampling distance h and the samples of the functions. To find a somewhat useful derivative, you can iteratively try to find a good approximation. Following the deduction of the general formula of the n-th order first derivative, smaller h *should* lead to better approximations, and thus, the difference between the approximation of the last and the current iteration should get smaller each time h is reduced. Due to that, it makes sense to compare two subsequent results and use it as a stopping criterion. First tests, seen in Figure 4.1, conducted in Matlab, show that defining a proper stopping criterion is more intricate than at first glance. Although the results look promising for the sizes of h used later in the tests, the derivative approximation is still a temporary workaround, and therefore, extensive investigations on ideal stopping criteria are ceased. Approximating the derivatives with B-Splines is a more convenient way where no such investigations are necessary. Hence, the finite difference ansatz is disregarded hereinafter.

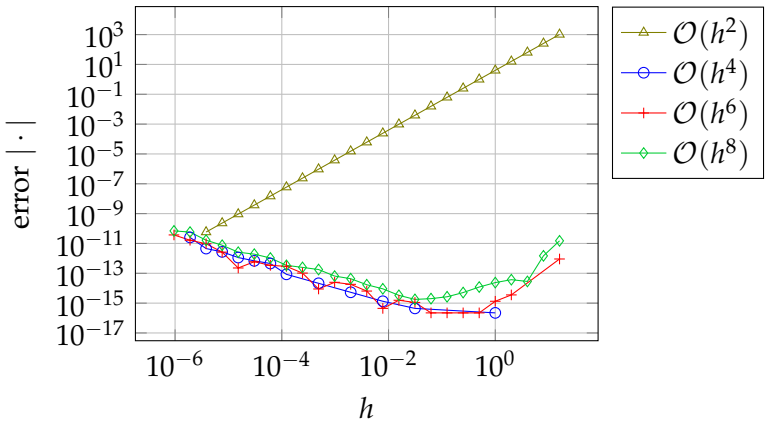

Figure 4.1: In this figure, errors resulting from the finite difference scheme with different convergence orders $\mathcal{O}(h^n)$ are depicted depending on h.
The function used for this test is $f(x) = x^4 + 3x^3 - 5x^2$, and the derivative is approximated at $x = 0.5$. For omitted data points, the error is zero.

4.1.2 B-Splines

A different method to approximate the first derivative is to sample the waveform of preCICE and reconstruct this time interpolant. In other words, we interpolate the waveform of preCICE. B-Splines have some nice properties, making them a preferred option over other interpolants. After finding the interpolant, representing the derivative, you can relatively cheaply evaluate this function at arbitrary points [14]. You additionally have the property that B-Splines are numerically stably constructed and well-conditioned for small B-Spline degrees [8]. Thus, they fulfill properties that are desired in numerics.

Implementation

As this method of approximating the derivatives is used in most tests, this implementation is described in more detail than Subsection 4.1.1. The key functions which are used are from the `scipy.interpolate` package:

```
splrep(x, y, w=None, xb=None, xe=None, k=3, task=0
       , s=None, t=None, full_output=0, per=0, quiet=1)
BSpline(t, c, k, extrapolate=True, axis=0)
BSpline.derivative(order)
```

The function `splrep` returns a triplet, which is required to initialize the class `BSpline` and defines a B-Spline representation corresponding to the function parameters.
The `BSpline` object holds this interpolant and offers the method `derivative()`, which enables the user to retrieve an arbitrary derivative, which then can be evaluated in the same interval as its antiderivative. As only the first derivative of the waveform is needed, `derivative(1)` is used.
The only parameter that needs to be determined during runtime is the set `y`. This parameter represents the function values of the function that should be interpolated, and therefore, it is necessary to evaluate the waveform of preCICE at the points prescribed by `x`. After the function call of `splrep`, an instance of the `BSpline` function is created, and lastly, the derivative is computed by the respective function call.
Once the derivative is determined, you can use it within the whole time window. That means even if you use a multi-stage time-stepping scheme, it is only necessary to evaluate the computed derivative at the corresponding time.

4.2 Solver for partitioned PDEs

As seen in Chapter 2, solving a partitioned PDE includes coupling and setting further boundary conditions in addition to aspects that need to be considered for solving a monolithic PDE.
[18] already shows how to implement higher-order implicit Runge-Kutta methods for the heat equation in a monolithic setup with FEniCS and the Butcher Tableau class provided by Irksome. This code is used as a basis for implementing the partitioned setup. Hence, steps equal to the reference are not explained in detail here.
That is why only code segments that underwent essential changes regarding partitioning and coupling are presented. How subproblems are tackled is explained first. Afterward, it is only necessary to assemble each code segment into one program that is compatible with the FEniCS-preCICE adapter and thus allows coupling.

4.2.1 Used version of preCICE

During the writing process of this thesis, preCICE v3 was still under development. No official release version of the library is therefore used. To circumvent the problem of getting incoherent testing results, the preCICE repository was not updated and rebuilt after starting the testing process. In the used version, preCICE only allows waveforms up to degree three. To allow waveforms of higher degree, the preCICE source code was modified.
Modifying the line

```
Time::MAX_WAVEFORM_DEGREE = 3;
```

of the file `src/time/Time.cpp` of the preCICE repository permits this.[3]
Not only is the new version of the coupling library under development, but also the adapters. Hence, the version of the used FEniCS-preCICE adapter is also not an official release version.[4]
The program is written with the preCICE Python bindings of version 3.0.0.0dev2.

4.2.2 Initialization of Boundary conditions

Implementing the boundary conditions, which are not coupling boundaries, is identical to the reference implementation. Dirichlet boundaries are defined by the time derivative as in Equation 2.11.

[3] Concretely, for the commit that is used in this thesis, here: `https://github.com/precice/precice/blob/76f871124a813c59394e4fdcc1da990dce8064ea/src/time/Time.cpp`

[4] The version of FEniCS-preCICE is the following commit: `https://github.com/precice/fenics-adapter/commit/6f998598aac79f923235bf1288fd9dd28ae07ef6`.

The coupling boundaries, however, are updated according to the used coupling scheme. Hence, it is necessary to obtain the boundary values from preCICE and set them as the boundary conditions. For an s-stage time-stepping scheme, s different boundary conditions must be imposed as seen in Equation 2.10. Consequently, s `coupling_expressions` must be initialized by one adapter. As a coupling expression created by an adapter does not modify the state of the adapter itself, multiple coupling expression instances can be created and used. A possible implementation for initializing the Dirichlet boundary conditions can be viewed in Listing 4.1.

```
# get time derivative of u
du_dt_expr = u_D_sp.diff(t_sp)
du_dt = [Expression(sp.ccode(du_dt_expr), degree=2,
            t=float(stage_times[i])) for i in range(tsm.num_stages)]
# set up coupling expressions for boundary conditions
coupling_expressions = \
    [precice.create_coupling_expression() for _ in range(tsm.num_stages)]

...

# initialize boundary conditions
bc = []
for i in range(tsm.num_stages):
    if problem is ProblemType.DIRICHLET:
        bc.append(DirichletBC(Vbigs[i], du_dt[i], remaining_boundary))
        bc.append(DirichletBC(Vbigs[i], coupling_expressions[i],
                    coupling_boundary))
    else:
        bc.append(DirichletBC(Vbigs[i], du_dt[i], remaining_boundary))
        F += vs[i] * coupling_expressions[i] * ds
```

Listing 4.1: A Code snippet that shows how boundary conditions can be initialized. In the first segment, the derivative of the boundary values is computed; below, the Dirichlet boundaries are initialized for both participants at the remaining_boundary. For the Neumann participant, the conditions of the coupling boundaries are enforced directly by adding them to F, whereas for the Dirichlet participant, the coupling_boundary is defined identically to the remaining_boundary.

`Vbigs` is an array that stores the function space for each stage of the time-stepping scheme. For each stage, boundary conditions need to be defined, which comprise the `coupling_boundary` and the `remaining_boundary`. The `remaining_boundary` represents the part of the boundary that is not the coupling boundary. If boundary conditions should be defined for a Dirichlet participant, they will all be Dirichlet boundary con-

ditions. Hence, the `remaining_boundary` of the `i`-th stage gets the prescribed time derivatives `du_dt[i]` of the solution and the `coupling_expressions` the computed time derivative from the waveform at the relative stage time.
The boundary conditions for the Neumann participant, so the participant of the Dirichlet-Neumann coupling which has one boundary with Neumann boundary conditions, needs a slightly different implementation for the coupling boundary as this is a case of mixed boundaries. We know from the waveform iteration we use that the Dirichlet side computes the heat flux as interpolation conditions for the waveform. Sampling the waveform on the Neumann side with `read_data` hence returns the heat flux. According to Equation 2.10, the Neumann conditions are explicitly given in the weak form. The weak form F is augmented correspondingly in the last line of Listing 4.1.

4.2.3 Simulation loop

The simulation loop of the partitioned ansatz has the same structure as Algorithm 3 that represents the sequence of the most important solving steps for a monolithic solver. Intuitively, it makes sense because each participant is a monolithic solver itself.
Essential differences are found in the definition of the time step size and the end time of the simulation, as well as in the update of the boundary conditions. The additional functionality for exchanging data must also be added to the simulation loop.

Algorithm 3 The high-level structure of the simulation loop. Regardless of a monolithic or partitioned setup, each step remains in the solving process while the implementation details may differ. In a partitioned setup, the only missing step would be the coupling process.

1: **while** simulation not complete **do**
2: Update boundary conditions.
3: Find the solution of the weak form obtained by Equation 2.10.
4: Assemble the discrete evolution to determine the sought-after solution.
5: Advance the time.
6: **end while**

Changed code segments

Instead of using constant time step sizes, which are defined before the runtime, the time steps of the participants, $\delta t_{\mathcal{D}}$ and $\delta t_{\mathcal{N}}$, need to be adapted to the time window size that preCICE prescribes. Due to coupling, the last time step of each participant

within a time window must end exactly at the time the time window ends. So, the δt a participant uses is defined as follows:

```
precice_dt = precice.get_max_time_step_size()
dt.assign(np.min([fenics_dt, precice_dt]))
```

Here, `dt` is the time step size of the participant, and `fenics_dt` is the local definition of the maximal time step size. `get_max_time_step_size()` returns the difference between the current simulation time and the time at which the current time window ends. From those two time step sizes, the minimum is chosen as the step size of the current iteration.
The problem we want to solve should be coupled during the whole simulation, so preCICE must couple it from start to end. `max-time` in the preCICE configuration file marks, therefore, the endpoint of the simulation. If `is_coupling_ongoing()` returns false, the simulation loop can be left.
After the solution of the current time step is computed and exchanged with preCICE, preCICE's internal coupling time is updated by `advance(dt)`.
Lastly, the implementation of the update of the boundary conditions at the coupling boundary remains. As this code change is strongly connected with the exchange of boundary conditions, the modifications regarding boundary conditions are described in the following paragraph.

Updating boundary conditions

Excluding the coupling boundaries, updating the boundary conditions is identical to the reference. For the coupling boundaries, it is necessary to determine if the participant has a Neumann or Dirichlet boundary. Updating the Neumann coupling boundary in Listing 4.2 is straightforward. For each stage of the time-stepping scheme, the waveform is sampled at the respective stage times and the coupling expression is updated accordingly.

```
for i in range(tsm.num_stages):
    precice.update_coupling_expression(
        coupling_expressions[i], precice.read_data(stage_times[i]))
```

Listing 4.2: The code snippet of the implementation that shows how the Neumann boundary conditions at coupling boundaries are updated.

The variable `coupling_expressions` is the same as in Listing 4.1.
Updating the boundary conditions of the Dirichlet participant is more intricate. We recall the theory from Subsection 2.3.2: the Dirichlet boundary needs the time derivative of the sought-after solution, so the derivative of the waveform is computed first to update the boundary conditions. In each time step, the program executes Listing 4.3. The

variable `bsplns_der` stores for each node `ky` at the coupling boundary the continuous time derivative of the waveform. The nodes are prescribed by the spatial discretization of the FEM. Updating the boundary conditions requires renewing the values at each prescribed node for each stage `i`. The value is determined by evaluating all B-Splines at the stage time of the `i`-th stage `stage_times[i]`.

```
for i in range(tsm.num_stages):
    # values of the derivative at the current time
    val = {}
    for ky in bsplns_der.keys():
        val[ky] = bsplns_der[ky](float(stage_times[i]))
    precice.update_coupling_expression(coupling_expressions[i], val)
```

Listing 4.3: The code that updates the coupling boundary for the Dirichlet participant. If preCICE provided a function that returns the derivative of the waveform directly, building the dictionary val for all stages would be not required.

Updating the waveforms is the residual code segment.
After each performed time step, the participant sends the result to preCICE with `write_data()` function. The Dirichlet participant computes the heat flux of his solution before calling the function, as the Neumann participant expects a waveform that interpolates the heat flux.

5 Testing methodology & Results

In this chapter, experiments with the implementation of the previous chapter are conducted regarding the convergence order.
First, the method of manufactured solutions (MMS) is introduced in Section 5.1. It is a method that is used throughout this section to verify if the program computes results with a certain convergence order.
Next, Section 5.2 discusses the methodology of the experiments. This includes the testing setup, so, among other things, restrictions on the computational domain, the partitioning and time step sizes. We also determine the ideal relative convergence measure to obtain sensible results.
With this setup, the results of the partitioned solver are compared with the results of the monolithic solver, which served as a reference in Chapter 4, first. Furthermore, we investigate how the choice of waveform degrees influences the overall convergence order. To this end, convergence studies with different time integrators paired with waveforms of different degrees are created.

5.1 Method of manufactured solutions

The MMS allows us to verify the convergence order of a numerical algorithm. Using this method requires manufacturing a solution that can be independent of the equation you want to solve. Prerequisites for such manufactured problems are non-triviality and a way to compare the results with the analytical solution. [11]
The paper [7] gives a basic example function for the heat equation, namely

$$u(x, y, t) = 1 + x^2 + 3y^2 + 1.2t^n, \tag{5.1}$$

where $n \in \mathbb{N}$.
If we want to verify that the solution of a time-stepping scheme of order n' has the same convergence order, we set n as n' and use the function u as our sought-after solution. Errors within machine precision indicate that the solver computes solutions with convergence order n'. This is a reasonable assumption because a time-stepping method of order n should produce error terms only for functions that exceed the n-th degree due to the deduction of Runge Kutta methods via Taylor expansion. [2]

This function and a slightly modified version of it are used throughout this section. The different form of u from [13], which will be used, is

$$u_{tri}(x, y, t) = 1 + sin(t)x^2 + 3y^2 + 1.2t.$$

Introducing this non-polynomial term leads to a time-dependent heat flux. A constant heat flux over the whole simulation time may hide faulty boundary conditions because these do not change over time. Faulty boundary conditions would, therefore, be concealed by boundary conditions that were correctly set once. Such a bug will be recognized when looking at the resulting errors in the computations. Wrong boundary conditions lead to higher errors than expected or even reduce the convergence order of the solution.
Furthermore, the sine function is non-zero for all derivative orders. Thus, no time-stepping scheme produces errors within machine precision for arbitrary large time steps.

5.2 Methodology

At this point, it should be clear that the accuracy of the computation depends on several parameters. The testing is organized in two parts to keep an overview of the sheer number of possible configurations and test cases. The first part consists of verifying the implementation, determining if the order of the time-stepping methods is preserved, and comparing it with the results of monolithic solvers from [18]. Secondly, it is tested how the convergence order of the solution reacts to different waveform degrees. As a byproduct of the tests, we can verify if the boundary conditions are imposed correctly. The function u_{tri} is used for this. To ensure that the scope of the tests is limited to the essentials, sensible restrictions are placed on the tests at the beginning.

The first restriction is applied to tol_{conv} of preCICE. Even though this variable can impact the runtime and accuracy of the program tremendously, it is fixed to a value of 10^{-11}, as long as the coupling scheme converges within 100 iterations. This value seems to be a good balance between strictness and computational cost since the comparison of the error behavior is solely dependent on the relative convergence measure shows that the error reduction from 10^{-11} onward was only marginal, as seen in Figure 5.1. In each test series, the time-stepping scheme, time window size, polynomial degree of the manufactured solution, and B-Spline degree were fixed.
The B-Spline degree is set to the highest possible order, namely five. It may seem counterproductive to interpolate a linear function piecewise quintic. Still, we can assume that even if the knots are all on a straight line within small perturbations, the

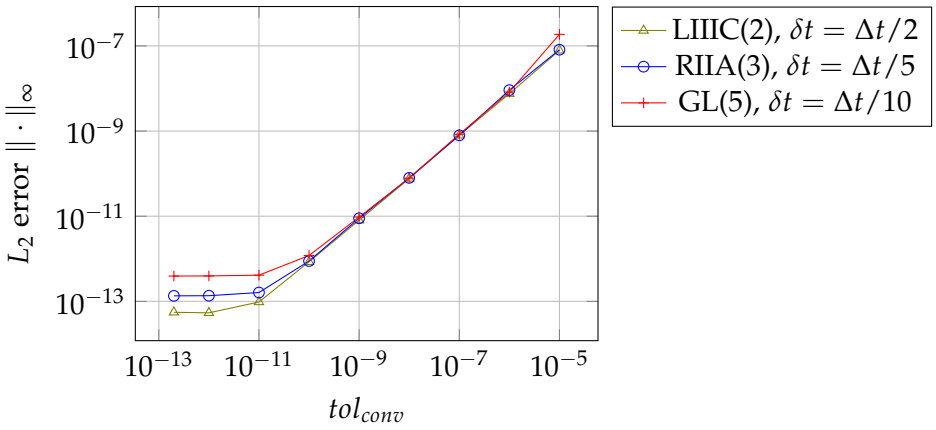

Figure 5.1: Errors dependent on the relative convergence measure with different time-stepping schemes and time step sizes. For a fixed time window size $\Delta t = 0.125$, the magnitude of the errors does not change significantly for $tol_{conv} < 10^{-11}$.

interpolant changes locally as this method is numerically stable.
If not other specified, the following setup is used:

- Figure 2.1 shows the domain used, including the partitioning.
- The time step sizes are chosen as $\delta t = \delta t_{\mathcal{N}} = \delta t_{\mathcal{D}}$
- Each partition of the domain has 11 nodes in each the x and y direction
- The waveform degrees are chosen as $p_{\mathcal{N}} = p_{\mathcal{D}} = p$, so both waveform degrees are identical
- The time window size is chosen minimal as $\Delta t = p \cdot \delta t$
- Relaxation factor of 0.375

5.3 Results

5.3.1 Comparsion to monolithic solutions

In the partitioned setup, creating tests identical to the reference [18] is impossible as we need to define more than just the time step size. The reference uses the Gauss-Legendre method with different stage numbers. This method shall also be used in the partitioned setup. First, we keep the time step size defined for the time-stepping schemes

identical to the size used for the monolithic results, so $\delta t = 0.03125$. Additionally, the setup from [18] requires $\Omega = [0,1] \times [0,1]$ and $T = 2$. The partitioning is chosen as $\Omega_{\mathcal{D}} = [0, 0.5] \times [0,1]$ and $\Omega_{\mathcal{N}} = [0.5, 1] \times [0,1]$. Second, the waveform degree is chosen to be equal to the convergence order of the time-stepping scheme. The second column of Table 5.1 defines the n of Equation 5.1. For now, waveform degrees less than the polynomial degree shall be neglected even if they provide at least equally precise results.
The error in the last row is by a magnitude of 10^6 worse than the result of the monolithic solution. The high tol_{conv} required for convergence of the coupling scheme is certainly

Monolithic results from [18]			Partitioned results
Number of stages	Polynomial order of RHS	error	error
1	2	1.098e-14	1.156e-13
2	4	7.456e-15	6.614e-14
3	6	7.228e-15	7.822e-13
4	8	7.612e-15	3.985e-12
5	10	7.907e-15	-
6	12	7.976e-15	-
7	14	7.793e-15	-
8	16	8.099e-15	2.976e-09*

Table 5.1: Comparison of errors between the monolithic and partitioned heat equation. The waveform degree is equal to the polynomial order of the right-hand side. Entries with no values cannot be computed with $\delta t = 0.03125$ because the simulation end time is no integer multiple of Δt. preCICE would prematurely terminate the program in this case. An issue regarding this is already been opened on the GitHub repository.[1]The entry marked with * has a relaxed convergence measure of $5 \cdot 10^{-8}$ as the coupling does not converge within 100 iterations otherwise.

the reason for the bad approximation. As seen in Figure 5.1, choosing $tol_{conv} \approx 10^{-8}$ instead of 10^{-11} may result in losing precision of about 10^3 for simulations that already end at $T = 1$. We can, therefore, not conclude from Table 5.1 that the solution of the last test has a convergence order lower than 16. If the convergence order of the solution computed with GL(8) actually is 16, which is tested later, we can deduce from that that the simulation is poorly configured, and thus, a direct comparison with monolithic solvers is not feasible.

[1]The issue can be accessed via `https://github.com/precice/precice/issues/1922`.

Reasons for the larger deviation from the analytical solution for all tests are that the time derivative needs to be approximated by the Dirichlet participant and the coupling process. Both steps produce errors that are to some extent unavoidable, like deviations arising from interpolation, but do not concern the monolithic solvers.
Still, the experiments of the first four rows approximate the analytical solution, especially when putting the magnitude of the error into perspective, only marginally worse than the monolithic solver. Furthermore, you can assume these values are exact within machine precision.

5.3.2 Correlation between waveform degree and convergence order

This section investigates how the waveform degree and the overall convergence order correlate. To this end, the function u_{tri} is used as the manufactured solution.
Now, merely computing the solution for one time step is insufficient to determine which convergence order the chosen preCICE and FEniCS configuration has, and, consequently, a convergence study is necessary. The following computations are done with different time step size sets $\mathcal{T}$ with $\delta t \in \mathcal{T}$. We define $\mathcal{T}$ as

$$\mathcal{T}_T(p_{max}) = \{\delta t = \frac{1}{2^l} \cdot \frac{T}{p_{max}} | l \in \{0, ..., 5\}\},$$

where $p_{max} \in \mathbb{N}$ is the maximal waveform degree used in a test series. We define that each test series comprises six computations with different time step sizes. δt is reduced by the factor 0.5 for each test. For the time window sizes, we choose for $\delta t = \frac{1}{2^i} \cdot \frac{T}{p_{max}}$, $\Delta t = \frac{1}{2^i} \cdot T$ independent of the waveform. So, we use a time window size that is not minimal for a given p and δt.

Time-stepping schemes of order four

Figure 5.2 shows that lower waveform degrees reduce the overall convergence of the solution. Though for $p = 4$, the errors converge with the same convergence order as the time stepping scheme, waveforms of degrees two and one reduce the convergence by a quadratic and approximately cubic order, respectively.

Time-stepping schemes of order eight

In Figure 5.3, tests with time-stepping schemes of eighth order are depicted. Regarding the convergence of the results, this plot provides the same conclusion as in Figure 5.2, excluding the errors of the first time step size. Increasing waveform degrees leads to a better convergence order.

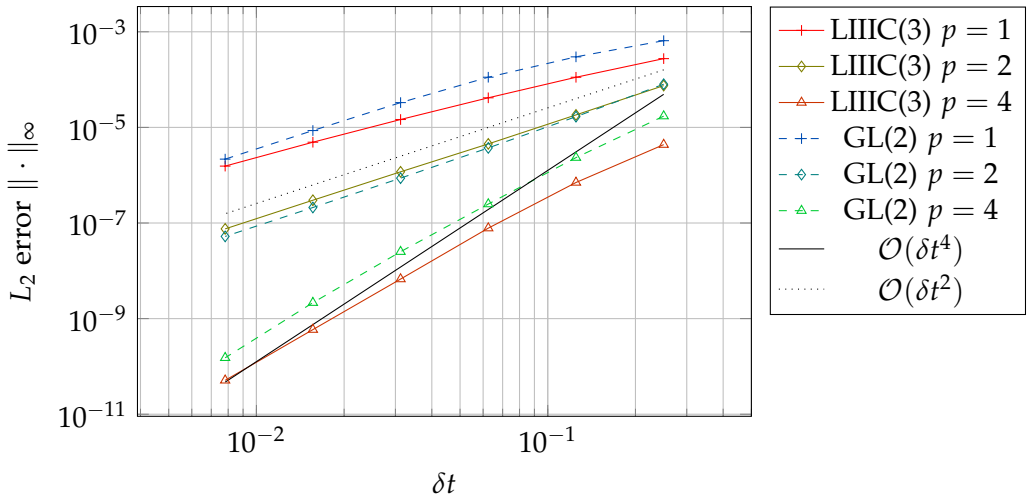

Figure 5.2: L_2 errors of different time-stepping schemes and different waveform degrees p. The convergence order of the result diminishes if the waveform degree is smaller than the order of the time-stepping scheme.
For this convergence tests $\mathcal{T}_1(4)$ is used.

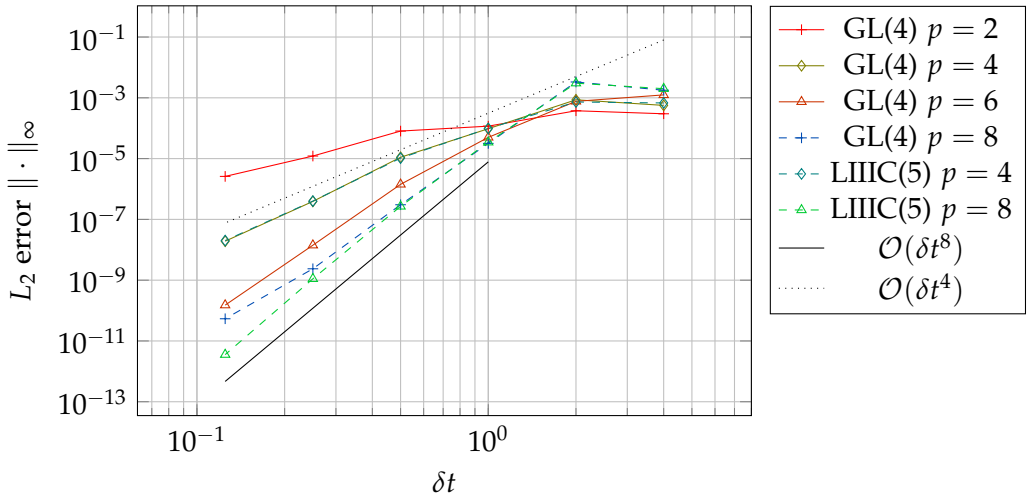

Figure 5.3: L_2 errors produced by LIIIC(5) and GL(4) with different waveform degrees p. For this convergence tests $\mathcal{T}_{32}(8)$ is used. Instead of $tol_{conv} = 10^{-11}$, $tol_{conv} = 10^{-9}$ is used as the coupling scheme did not converge for some tests.

Time-stepping schemes of order 16

Testing the convergence order for waveform degrees and time-stepping schemes of order 16 led, for multiple attempts, to convergence plots where a statement about the convergence order was difficult. The configuration of the test series in Figure 5.4 is the most satisfactory of all and is, hence, presented. In the plot, we can only recognize the convergence order to some extent in the data points of the two to three largest time step sizes for waveforms of large degree. All time-stepping schemes converge, as it seems, to a straight line that increases with decreasing time step sizes. As with increasing time steps, more computations are necessary, this line represents the unavoidable error in the applied configuration of the testing series. This unavoidable error increases with increasing computations since each calculation step on a machine produces an error.

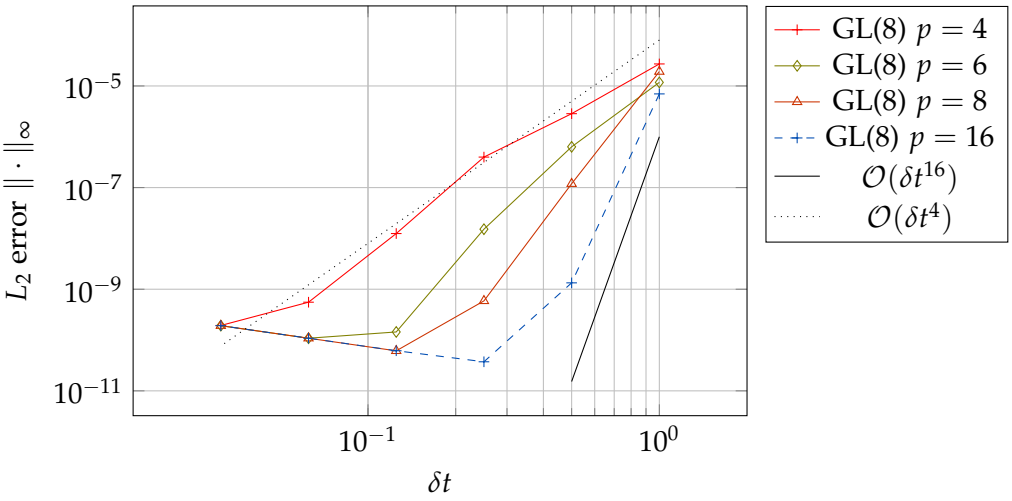

Figure 5.4: L_2 errors of the GL(8) method with different waveform degrees. The setup for this testing series is $\mathcal{T}_{16}(16)$, $tol_{conv} = 5 \cdot 10^{-8}$ and 0.5 as the relaxation factor.

5.3.3 Convergence results for minimal time window sizes

While conducting convergence tests for the previous chapter, a slightly modified testing setup led to a different and remarkable result regarding convergence.
If Δt is chosen minimally, the convergence behavior is not necessarily identical to Δt that is not minimal. The sole reason Δt is not minimized for all waveform degrees in Subsection 5.3.2 is to use a uniform set of time step sizes. A uniform set of time steps compatible with minimized time windows for all waveform degrees that should be

tested results in significantly smaller time steps. The computational effort is, however, unreasonable since the maximal time step would be T divided by the least common multiple of all waveforms used in a test series.
In this section, the two time-stepping schemes GL(2) and GL(4) and the results from the previous section are used to investigate how minimal time window sizes influence the convergence behavior of the error.
Figure 5.5 shows that minimizing Δt can have a positive effect on the convergence behavior. Instead of a quadratic convergence order, the experiment with the two-stage time-stepping scheme converges approximately cubic to the analytic result.
While the minimization process with GL(2) delivers better results, experiments with time-stepping schemes of higher order as in Figure 5.6 cannot reproduce the same properties.

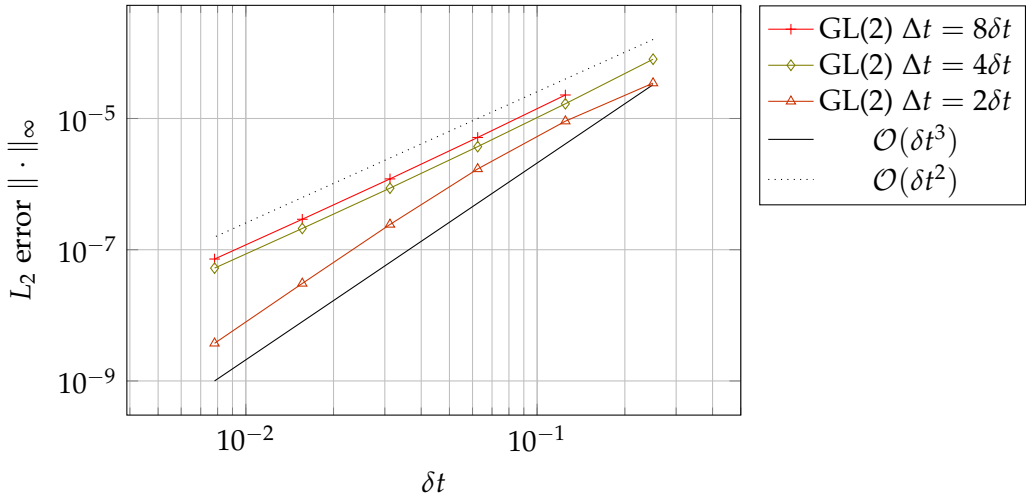

Figure 5.5: The convergence for the GL(2) method, waveform degree two and different time window sizes.Values of GL(2) $\Delta t = 4\delta t$ are taken from Figure 5.2. The convergence of the diamonds, which is the test series with the minimized time window, is approximately cubic. Larger time steps have, in contrast, apparently no negative influence on the convergence order.

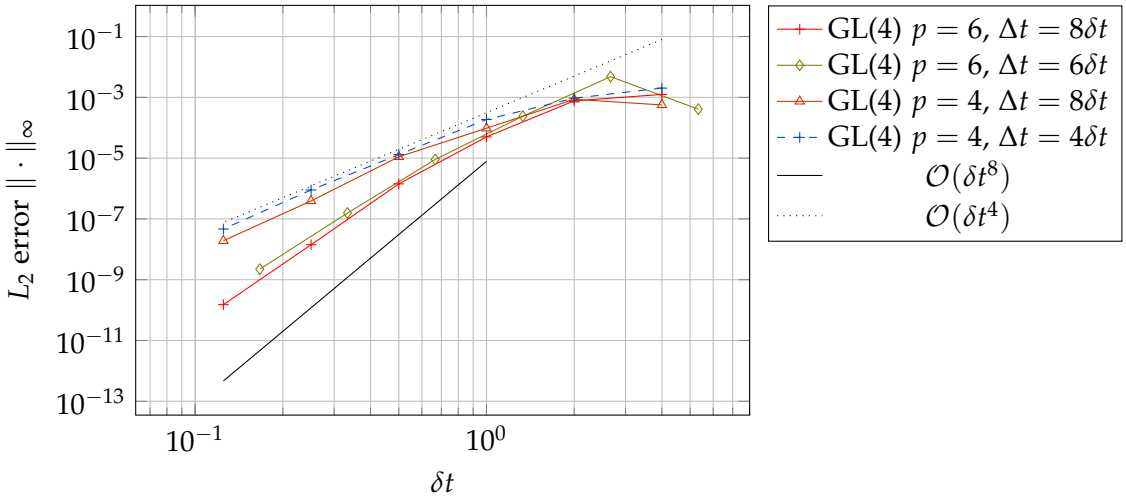

Figure 5.6: The convergence results for the non-minimized time windows are the same as in Figure 5.3. No improvement of the convergence can be detected in these experiments.

6 Conclusion & Outlook

To simulate multi-physics systems, the divide-and-conquer approach can be utilized to reduce the complexity of the problem statement to a bare minimum. Partitioning entails the coupling process that is not necessary when solving problems monolithically. Hence, to obtain higher-order solutions, the coupling should, at best, be no bottleneck for the convergence order.
In this thesis, the theory about solving time-dependent PDEs monolithically was discussed that is versatilely applicable to arbitrary Runge-Kutta methods and PDEs. Based on this, the time-dependent coupling method, called waveform iteration, was introduced. FEniCS and Irksome form the framework used in the implementation to easily develop a program that solves the monolithic heat equation, as seen in [18]. This code was augmented in this thesis to solve the partitioned heat equation instead.
Since preCICE offers only the waveform iteration as a higher-order coupling scheme, this coupling scheme was presented and used in testing.
In Chapter 5, we verified the convergence order of the results with the method of manufactured solutions. While the errors of the partitioned ansatz are higher than those produced by the monolithic solver we used as the basis for the implementation, we concluded that a one-to-one comparison is not sensible. However, in this comparison, the results of the partitioned heat equation were within machine precision for time-stepping schemes of orders two, four, six, eight, and 16.
Furthermore, this thesis investigated how the convergence of the solution and the waveform degree correlate. To this end, the Gauss-Legendre method of different orders paired with waveforms of different degrees was mainly employed. The overall outcome of these experiments is that smaller waveform degrees lead to worse convergence orders, while the convergence order of a time-stepping scheme remains unchanged if a waveform of the same degree as the convergence order is used. Still, the results demonstrate that choosing waveform degrees not coinciding with the convergence order of the time-stepping scheme is viable. Hence, waveform degrees are another variable for steering the convergence order of a partitioned solver.
For one configuration, the time window size also influences the convergence order positively. Choosing minimal time window sizes can lead to better convergence behavior compared to non-minimized time windows.

Regarding the experiments already conducted in [13], especially the results for orders six, eight and 16 give new insights about the convergence behavior using the waveform iteration as a coupling scheme with preCICE. Generally speaking, the experiments yield that the waveform iteration in preCICE can be used to compute solutions beyond the convergence order four if preCICE dismisses restricting the maximal waveform degree of three.

In this thesis, the tests focused on the correlation between the waveform degree and the convergence order while using the same time integrator, time step size and waveform degree. Experiments that have no restrictions regarding the equality of these parameters would supplement the tests conducted in Chapter 5, showing that the waveform iteration maintains the convergence order regardless of the configuration of participants. Furthermore, finding other configurations that lead to better convergence for minimized time windows is an interesting topic for further experiments since a pattern may materialize for such cases.
A highly intriguing enhancement for the code presented in Chapter 4, or rather preCICE, is an API to retrieve the time derivative directly from preCICE instead of a self-implemented workaround.
Ideally, this API returns the values of the coupling boundary equally to the already existing API for reading data, in order to use it directly. A possible function signature could look like

```
read_data(dt, order)
```

The already existing function is merely augmented by a second parameter that defines the desired order of the derivative of the waveform, inspired by the `BSpline` API. The code from Listing 4.3 shrinks then to

```
for i in range(tsm.num_stages):
    precice.update_coupling_expression(precice.read_data(stage_times[i], 1), val)
```

Additionally, the implementation presented in Subsection 4.1.2 ceases.

Abbreviations

API application programming interface

BVP boundary value problem

FEM finite element method

GL Gauss-Legendre

LIIIC LobattoIIIC

MMS method of manufactured solutions

PDE partial differential equation

RIIA RadauIIA

List of Figures

List of Tables

Bibliography

[1] G. Chourdakis et al. *preCICE v2: A sustainable and user-friendly coupling library [version 2; peer review: 2 approved]*. Open Res Europe 2022, 2022, pp. 3–8, 19. DOI: 10.12688/openreseurope.14445.2.

[2] Peter Deuflhard and Folkmar Bornemann. *Scientific Computing with Ordinary Differential Equations*. Springer New York, NY, 2002, 140–144, 145ff. DOI: 10.1007/978-0-387-21582-2.

[3] Patrick E. Farrell, Robert C. Kirby, and Jorge Marchena-Menéndez. "Irksome: Automating Runge–Kutta Time-Stepping for Finite Element Methods." In: *ACM Trans. Math. Softw.* 47.4 (Sept. 2021). ISSN: 0098-3500. DOI: 10.1145/3466168.

[4] Mark S. Gockenbach. *Understanding and Implementing the Finite Element Method*. Society for Industrial and Applied Mathematics, 2006, pp. 20, 21, 23, 29–33.

[5] David A. Ham et al. *Firedrake User Manual*. First edition. Imperial College London et al. May 2023. DOI: 10.25561/104839.

[6] Ulrich Küttler and Wolfgang A. Wall. "Fixed-point fluid structure interaction solvers with dynamic relaxation." In: *Computational Mechanics* 43.1 (Dec. 2008), pp. 61–72. DOI: 10.1007/s00466-008-0255-5.

[7] Hans P. Langtangen and Anders Logg. *Solving PDEs in Python - The FEniCS Tutorial I*. Sptringer Cham, 2017. DOI: 10.1007/978-3-319-52462-7.

[8] Tom Lyche and Knut Mørken. *Spline Methods Draft*. Department of Informatics, Centre of Mathematics for Applications University of Oslo, 2008, pp. 196–198.

[9] Bankim Mandal. *Convergence analysis of substructuring Waveform Relaxation methods for space-time problems and their application to Optimal Control Problems*. 2014, pp. 13, 15. DOI: 10.13097/archiveouverte/unige:46146.

[10] L.D. Marini and Alfio Quarteroni. "An iterative procedure for domain decomposition methods: A finite element approach." In: *First International Symposium on Domain Decomposition Methods for Partial Differential Equations*. Society for Industrial and Applied Mathematics, 1988.

[11] Patrick J. Roache. *Verification and Validation in Computational Science and Engineering*. Hermosa Publishers, 1998, pp. 67–69.

[12] Benjamin Rodenberg et al. "FEniCS–preCICE: Coupling FEniCS to other simulation software." In: *SoftwareX* 16 (2021), p. 100807. ISSN: 2352-7110. DOI: 10.1016/j.softx.2021.100807.

[13] Benjamin Rüth and Benjamin Uekermann et al. "Quasi-Newton waveform iteration for partitioned surface-coupled multiphysics applications." In: *International Journal for Numerical Methods in Engineering* 122.19 (2021), pp. 5236–5257. DOI: 10.1002/nme.6443. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.6443.

[14] Larry L Schumaker. *SPLINE FUNCTIONS: BASIC THEORY*. Wiley, 1981, pp. 194, 197.

[15] Cameron R. Taylor. *Finite Difference Coefficients Calculator*. https://web.media.mit.edu/~crtaylor/calculator.html. 2016.

[16] Benjamin Uekermann, Bernhard Gatzhammer, and Miriam Mehl. "Coupling algorithms for partitioned multi-physics simulations." In: *Informatik 2014*. Bonn: Gesellschaft für Informatik e.V., 2014, pp. 113–124. ISBN: 978-3-88579-626-8.

[17] Jacob White et al. "Waveform relaxation: theory and practice." In: *Transactions of the Society for Computer Simulation* 2 (June 1985).

[18] Nikola Wullenweber. *Higher-order time stepping schemes for solving partial differential equations with FEniCS*. 2021, pp. 35, 36.