Bathymetry reconstruction in shallow water using PDE-constrained optimisation and its acceleration through parallel-in-time methods

Vom Promotionsausschuss der
Technischen Universität Hamburg

zur Erlangung des akademischen Grades

Doktorin der Naturwissenschaften (Dr. rer. nat.)

genehmigte Dissertation (Monografie)

von
Judith Angel

aus
Hamburg

2025

1. Gutachter: Prof. Dr. Daniel Ruprecht
2. Gutachter: Prof. Dr. Jörn Behrens

Tag der mündlichen Prüfung: 13. Oktober 2025

doi:https://doi.org/10.15480/882.16012
ORCiD: https://orcid.org/0009-0008-2098-4883

Summary

The problem of reconstructing a bathymetry from measurements of the free surface can be formulated as an optimisation problem constrained by partial differential equations (PDEs). In this thesis, we describe the water depth by the shallow water equations (SWE) and use the 'first optimise, then discretise'-approach to solve this problem numerically. Then we investigate the use of parallel-in-time methods to accelerate the computations.

In Chapter 2 we explain theory and different approaches on PDE-constrained optimisation problems, followed by solution techniques such as optimisation algorithms. Furthermore, we introduce the SWE and the numerical solver that is being used for all PDEs in this work.

These methods are being applied to different problems in Chapter 3. First, we consider SWE with periodic boundary conditions. The adjoint SWE and gradient of the reduced objective functional are being derived and then used to infer a sinusoidal and a Gaussian-shaped bathymetry. Results for the gradient descent method and the L-BFGS algorithm are being compared. Then we work with time-dependent boundary conditions that correspond to an experiment in a wave flume. We infer a bathymetry with roughly Gaussian shape that was placed in the wave flume, where we use the previously introduced methods. Results for simulated and measured observations are being compared. It is also shown that the errors in the reconstruction are in line with results from existing literature.

Chapter 4 introduces several parallel-in-time methods and investigates their convergence for the forward and adjoint SWE. We apply the Parareal method to both equations and show results for spatial coarsening for the advection-diffusion equation and the SWE. As a more suitable method for hyperbolic problems we use Asymptotic Parareal, for which the SWE has to be reformulated by using a perturbation height instead of the water depth. For the adjoint problem we use the ParaExp method and investigate its convergence for the adjoint equation.

The results for serial and time-parallel optimisations are being compared in Chapter 5. There we reconstruct bathymetries for SWE with periodic boundary conditions and for the wave flume experiment.

Acknowledgements

There are many people who helped me a lot during my time as a PhD student. First of all, I give great thanks to my supervisor Daniel Ruprecht and my co-supervisor Sebastian Götschel who gave me all the support that I needed, taught me a lot and for all the good conversations that we had. Special thanks also goes to my co-advisor from the research training group RTG2583 and second referee, Jörn Behrens, for his valuable input and fruitful discussions.

A big thank you also goes to Marko Lindner and Christian Seifert for helpful discussions, nice conversations and everything I learned from them already during my studies. Furthermore, I would like to thank all my colleagues, especially Julio Urizarna, Joscha Fregin, Ikrom Akramov, Vamika Rathi, Sophie Externbrink, Thibaut Lunet, Katharina Klioba, Marco Wolkner, Jens-Peter Zemke for many helpful discussions, for coffee chats and everything else that made my time at TUHH wonderful. Thank you for helping me carry sofas, for taking care of my furry friend Maja and for all the fun that we had together! I am also very grateful to Marten Hollm who performed the measurements at the wave flume. Thank you to the technicians at the Institute of Mechanics and Ocean Engineering who turned skateboard ramps into a bathymetry that we could use for the experiment. Thank you also to Juliane Rosemeier for the great time and work together in Exeter, to Stefan Güttel for the idea of a stopping criterion for nonlinear ParaExp and to Beth Wingate for helpful discussions and the suggestion of using a linear bottom friction term in the shallow water equations.

Thanks also to the European Union and the BMFTR (former BMBF) for their financial support. We acknowledge the support by the Deutsche Forschungsgemeinschaft (DFG) within the Research Training Group GRK 2583 “Modeling, Simulation and Optimization of Fluid Dynamic Applications”.

Last but not least, I thank my friends and family, especially my parents, grandparents and godparents and everyone I did not mention by name for all the support and encouragement!

Contents

Chapter 1

Introduction

For the prediction of flooding as well as for some applications in science and engineering like the prediction of meteorological tsunamis or hazardous materials in estuaries [1, 2], the knowledge of the bathymetry, that is the bottom topography, is indispensable. Existing measuring techniques such as side scan sonars or airborne LiDAR are usually time consuming and costly [3]. Besides, when using side scan sonars there is the risk of stranding and the use of airborne LiDAR is limited to areas where flying is not prohibited [3]. An alternative is to analyse satellite images. This technique is called satellite derived bathymetry (SDB) [4]. It can gather data from large areas much faster than oceanographic vessels or aeroplanes [5]. The disadvantage of this method is that the quality of the measurement data suffers from atmospheric effects, sun reflection and the ripple of the water surface [6]. This drives the need for different methods to infer bathymetries, for instance by measurements of the water height, usually called the free surface elevation. In this thesis, this problem will be formulated as an optimisation problem that is constrained by shallow water equations (SWE). This problem will be solved with use of iterative methods, where at each iteration partial differential equations (PDEs) have to be solved. As this requires the computation of a lot of time steps in total, a parallelisation in time can be a promising approach to accelerate computations.

The theoretical background for PDE-constrained optimisation and all used algorithms is explained in Chapter 2. A functional is defined which measures the error of the solution of the PDE to some desired state or an observation. This functional has to be minimised under the constraint that the PDE has to be fulfilled. To this end, we introduce suitable iterative minimisation algorithms. There are two approaches on PDE-constrained optimisation problems, namely the 'first optimise, then discretise' and the 'first discretise, then optimise'-approach. In this thesis, we optimise first which means to derive optimality conditions in function space. This involves the derivation of a continuous adjoint equation and gradient of the functional. In each iteration of the minimisation algorithm, the state equations and adjoint equations have to be solved in order to compute the gradient for the descent direction. This is illustrated in Figure 1.1. Starting with an initial guess c_0, the control is being updated in each iteration of the optimisation algorithm. This approach gives more flexibility on how to solve the PDE and its adjoint. When discretising first, the adjoint problem is discrete already and this allows for usage of techniques such as automatic differentiation. For a deeper understanding of this topic, we refer to Hinze et al. [7] and Tröltzsch [8]. They give a good overview and explanations on how to solve optimi-

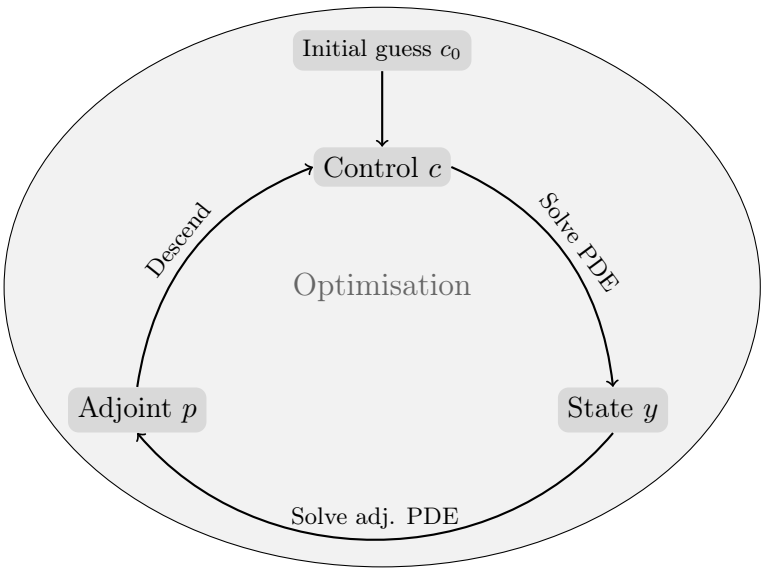

Figure 1.1: Optimisation with PDE constraints.

sation problems with parabolic and elliptic PDEs as constraints. However, for hyperbolic problems, there is not much information about the regularity of the solutions [8]. There is some theory on optimisation constrained by hyperbolic equations [9, 10, 11, 12, 13, 14], but solving these kind of problems remains challenging as shocks in the solution can occur. Nevertheless, we avoid these by considering only smooth bathymetries and short time horizons. Thus, we can use known iterative optimisation algorithms which we will introduce in Chapter 2. Furthermore, we state the nonlinear one-dimensional SWE with linear bottom friction term and explain the solvers that are being used for the forward and adjoint SWE.

Chapter 3 focuses on bathymetry reconstruction as an optimisation problem with SWE as constraints. Different bathymetries are being inferred from observations of the free surface. We work with periodic as well as time-dependent Dirichlet boundary conditions that arise from an experimental setup in a wave flume. We will also see reconstructions from real measurement data and compare the errors in the reconstruction for different types of observation. The continuous adjoint and gradient that we derive are then being discretised using spectral methods. Most other authors who reconstruct bathymetries from an observation of the free surface discretise before optimising. Hajduk et al. [15] discretise the SWE with a Discontinuous Galerkin method and then use a pseudo-time stepping scheme to infer the bathymetry. Monnier et al. [16] discretise the two-dimensional SWE with Finite Volume methods and use Data Assimilation to reconstruct the topography under the water as well as in dry areas. Gessese and Sellier [17] use a one-shot technique to reconstruct the bathymetry from measurements of the free surface. They substitute the bathymetry in the SWE, solve it numerically for the free surface elevation and water depth and then compute the bathymetry by resubstitution. Vasan and Deconinck [18] use the Euler equations for the bathymetry reconstruction. They use observations of the water wave height and its first two time derivatives to numerically compute the bathymetry via the solution of a set of two coupled equations. In a later paper, Vasan et al. [19] infer a bathymetry via two separate inverse problems. The 'first optimise, then discretise'-

approach is being used by Sanders and Katopodes [2]. They determine a continuous adjoint of the SWE with nonlinear bottom friction term using the method of Lagrange multipliers like it is being done in this thesis. But instead of the bathymetry, like it is done in this work, they optimise the boundary conditions in order to avoid flooding in river and estuarine systems. Thus, the paper by Angel et al. [20] where part of this thesis is based on is the first to use the 'first optimise, then discretise'-approach for bathymetry reconstruction with nonlinear SWE.

Chapter 4 treats parallel-in-time (PinT) methods for SWE and its adjoint. First, we consider the classical Parareal method that was introduced by Lions, Maday and Turinici [21] and apply it to the forward and adjoint SWE. Very few people have applied Parareal to SWE as the method is known to have issues with hyperbolic problems [22]. This is also confirmed numerically for the forward and adjoint SWE in Section 4.1.2. Authors who have worked with Parareal for SWE use slightly adapted Parareal methods. Nielsen [23] uses the Communication-aware Adaptive Parareal and Caldas Steinstraesser [24] uses a variant of Parareal with reduced-order models. As an option to improve Parareal convergence for SWE we consider the so-called Asymptotic Parareal that was developed by Haut and Wingate [25]. It uses a transformation on the SWE from which the *modulation equation* is obtained and applies averaging on this equation. Asymptotic Parareal has drawn some attention in the past years [26, 27, 28]. It has been applied to SWE with rotation, but we use a non-rotating SWE here. Our equation has to be reformulated in terms of a perturbation height instead of the water depth which appears in the original SWE. When applying Asymptotic Parareal on this equation we find that the averaging can significantly improve Parareal convergence. Researchers have also applied other PinT methods to SWE recently [29, 30, 31, 32, 33], but they work with SWE on the sphere and have only considered Asymptotic Parareal, but not Parareal for the modulation equation solely. In this thesis, Asymptotic Parareal, Parareal for the modulation equation and for the original SWE will be compared in terms of iterations. We will see, that only using the modulation equation can already improve Parareal convergence.

In order to deal with the adjoint equation, we apply the nonlinear ParaExp method [34], a variant of the ParaExp algorithm which was developed by Gander and Güttel [35]. It seems to be a promising method as ParaExp works well for linear hyperbolic problems [35] and our adjoint problem is linear, but with time-dependent coefficients. However, its convergence proves to be disappointing.

There is a lot of recent work for the PinT solution of optimisation problems with PDE constraints. But many of them use the 'first discretise, then optimise'-approach, like Hahne et al. [36] who use the PinT method multigrid reduction in time. Also in the work on linear-quadratic optimisation problems by Heinkenschloss and Kroeger [37] the optimality system is first being discretised and then diagonalised in order to solve it parallel in time. Parpas and Muir [38] optimise neural networks using PinT methods. Authors who optimise first are e.g. Heinzelreiter and Pearson [39] who derive a PinT preconditioner for fluid flow problems and Ulbrich [40], who proposes the use of Parareal as a preconditioner for the forward and adjoint problem. He shows that Parareal has potential for the parallel solution of parabolic optimal control problems, as only few Parareal iterations are needed in order to keep the needed number of iterations for the conjugate gradient method very moderate [41]. Here we deal with hyperbolic problems such that we cannot apply this method. Götschel and Minion [42] use the PinT method PFASST within an optimisation

problem, also constrained by parabolic equations. They showed that a warm start, i.e. using the solution from the previous optimisation iteration as initial guess for the PinT method, is very useful in this context. Appel and Alexandersen [43] use a one-shot Parareal method for topology optimisation of transient heat flow. Skene et al. [44] use ParaExp to accelerate the computation of the continuous adjoint of a general nonlinear equation. Constanzo et al. [45] generalise this method to the Navier-Stokes equations.

In Chapter 5 the previously introduced PinT methods will be applied to the forward and adjoint equation in order to perform a PinT optimisation. We will see that using PinT methods does not harm the quality of the reconstructed bathymetries, even though the solution of the PDEs is less accurate compared to the serial computations. We show results of the serial and PinT optimisation for periodic boundary conditions, where we use Asymptotic Parareal for the reformulated SWE and ParaExp for the adjoint equation. We compare the reconstructions and see that the quality does not suffer a lot from the slightly larger discretisation error that we get from using PinT methods. Asymptotic Parareal cannot be applied to the problem with time-dependent boundary conditions due to the way it has to be implemented within the software that we use. Hence, we will perform the PinT reconstruction using the original Parareal algorithm and obtain relatively good convergence for the forward SWE. But Parareal for the adjoint converges too slowly throughout the optimisation to obtain any speed-up. The same is true for ParaExp such that the main challenge remains to efficiently parallelise the adjoint equation in time.

Chapter 2

Optimal control problems for hyperbolic PDEs and solution techniques

The problem of bathymetry reconstruction can be formulated as an optimisation problem constrained with PDEs, in our case the SWE. In this chapter, all methods that are being used in this thesis will be explained. First, we will define a general optimisation problem with PDE constraints and shortly discuss approaches on how to solve it. Then we will look at different minimisation algorithms. After that, the SWE will be introduced, followed by explanations on a spectral methods framework which we use to solve the SWE and its adjoint.

2.1 PDE-constrained optimisation

2.1.1 Theoretical background

Consider a partial differential equation with initial and boundary conditions

$$e(y, c) = 0, \tag{2.1}$$

where $e : Y \times C \to Z$ is an operator and Y, C, Z are real Banach spaces with norm $\|\cdot\|$. We call $y \in Y$ the *state* and $c \in C$ the *control* and assume that y is a solution to the PDE (2.1) for a given control c. In order to express this dependence of the state on the control we sometimes write $y(c)$. The space-time domain will always be denoted by $Q = \Omega \times [0, T]$ in the following. Equation (2.1) could be the heat equation for example, that is

$$\begin{aligned} y_t &= y_{xx} + c \quad \text{in } Q \\ y &= 0 \quad \text{on } \Gamma \\ y(\cdot, 0) &= y_0 \quad \text{in } \Omega, \end{aligned} \tag{2.2}$$

where Γ denotes the boundary of Ω. In this case the function y describes the temperature in a room and the control c the heating. Suppose we have a desired state or an observation y_{obs} and aim to find the control for which the solution y of Equation (2.1) is as close as possible to y_{obs}. That is, we want to minimise the error $\|y - y_{\text{obs}}\|$. To this end, we define

an *objective functional* J that measures this distance of a state to the observation and also contains a regularisation term for the control. Let $\lambda > 0$ and define the objective functional

$$J(y, c) = \frac{1}{2}\|y - y_{\text{obs}}\|^2 + \frac{\lambda}{2}\|c\|^2. \tag{2.3}$$

The regularisation will ensure better smoothness properties of the optimal control given that $\lambda > 0$ [7]. In some applications, it can also be interpreted as e.g. energy costs [8] which is also true for our example with the heat equation. There, we try to get as close as possible to the desired temperature y while saving costs for heating. In general, we call a control *optimal* if it locally minimises the objective functional such that the PDE (2.1) holds [8], i.e. it solves the *PDE-constrained optimisation problem*. The parameter λ should be chosen positive [8] and sufficiently small, such that the main focus is still on the minimisation of the error $\|y - y_{\text{obs}}\|$. Suppose there is a control c^* for which this error is zero, but with $\|c^*\|$ being large. If the regularisation parameter is chosen too large, the optimal control will possibly be a control c^{**} with $\|c^{**}\| \ll \|c^*\|$ but the error to the observation only being slightly larger. In the case that the control is an actual bathymetry that we want to reconstruct this would mean that the result of the optimisation is rather useless. Often there are restrictions on the control [7, 8], i.e. a set of admissible controls is being defined which we call C_{ad}. The problem of finding the optimal control can then be formulated as the minimisation problem with PDE constraints

$$\min_{c \in C_{\text{ad}}} J(y, c) \quad \text{subject to} \quad e(y, c) = 0. \tag{2.4}$$

If the operator e is continuously Fréchet differentiable and e_y has a bounded inverse, then, by the implicit function theorem there is a control-to-state operator that maps a control c to the locally unique solution $y(c)$ [7]. Using the control-to-state operator we can eliminate the argument y in J and obtain the *reduced objective functional*

$$f(c) := J(y(c), c). \tag{2.5}$$

For the numerical solution of Problem (2.4) we will use minimisation algorithms that use the gradient of the reduced objective functional f to determine a descent direction and iteratively move towards the minimum. The gradient can be computed using the solution of the so-called *adjoint equations*. We derive these using the formal Lagrange technique [8]. Let $p = (p_1, \ldots, p_d) \in Z^*$ be Lagrange multipliers, where $d \in \mathbb{N}$ is the number of components in Equation (2.1). In the case of the heat equation (2.2) we have $d = 3$. We want to minimise the scalar product of p and the PDE (2.1). Thus, the Lagrange multipliers punish a violation of the PDE constraints. We define the *Lagrange function* as

$$\mathcal{L}(y, c, p) = J(y, c) - \langle p, e(y, c)\rangle_{Z^*,Z}, \tag{2.6}$$

where $\langle \cdot, \cdot \rangle_{Z^*,Z}$ is the dual pairing. Suppose we have an optimal control c^* and corresponding solutions y^* of the state equations and p^* of the adjoint equations. Then for the optimal point (y^*, c^*, p^*) the optimality conditions

$$\mathcal{L}_p(y^*, c^*, p^*)\varphi = 0 \quad \forall \varphi \in C_0^\infty(Q) \tag{2.7}$$

$$\mathcal{L}_y(y^*, c^*, p^*)\varphi = 0 \quad \forall \varphi \in C_0^\infty(Q) \tag{2.8}$$

$$\langle \mathcal{L}_c(y^*, c^*, p^*), c - c^*\rangle_{C^*,C} \geq 0 \quad \forall\ c \in C_{ad} \tag{2.9}$$

have to be satisfied [8]. Condition (2.7) yields the state equations, from the second Equation (2.8) the adjoint equations can be derived. These are often referred to as the *backward equations* as they run backwards in time. Likewise, the state equations are often referred to as the *forward equations*. The *variational inequality* (2.9) can be used to derive the gradient of the reduced objective functional which we need for the minimisation algorithms. The corresponding derivation for the specific problems that will be considered in this work will be shown in detail in Chapter 3. Note that inequality (2.9) turns into an equation in the case $C_{ad} = C$ [7].

There are different approaches on how to solve PDE-constrained optimisation problems. One possibility is to use the "First discretise, then optimise"-approach (FDTO). This means, the forward problem and objective functional are being discretised to obtain a finite dimensional optimisation problem. This has the advantages that the discrete adjoint and gradient are consistent and it is possible to use automatic differentiation software [46]. But sometimes it is necessary to treat the adjoint problem differently [46]. This is only possible with the "First optimise, then discretise"-approach (FOTD). It depends on the specific optimisation problem which one of the two approaches is better, so there is no general recommendation on which one to choose [7].

2.1.2 Optimisation algorithms

In this section, we will introduce iterative methods to solve the minimisation problem (2.4) using FOTD. Figure 2.1 shows the general procedure for this approach. The state equations are being solved forward in time. The result is then used to compute the solution of the adjoint equation which is backwards in time, and its solution is needed for the computation of the gradient. This is what has to be done in each iteration of the optimisation algorithm, where we mainly discuss the gradient descent method, but also shortly introduce alternatives. Quasi-Newton methods use an approximation on the second derivative and consequently they are expected to converge faster to the minimum [47], Gradient descent has the advantage that it is much easier to implement and still gives satisfactory results as we will see later. Suppose we have discretised the forward and adjoint problem. We

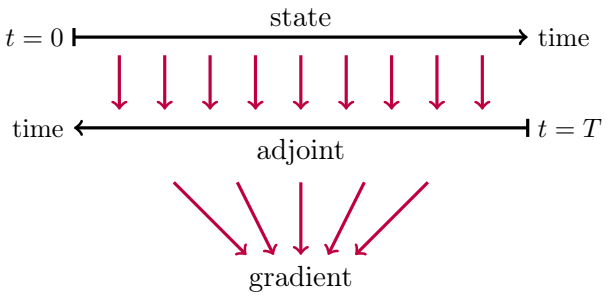

Figure 2.1: Illustration of the computation of the adjoint solution and gradient during the optimisation.

denote all discrete instances with a bar, e.g. $\bar{y} \in \mathbb{R}^n$ for $n \in \mathbb{N}$ as the discretised state. We also use the notation $\bar{f}_j := \bar{f}(\bar{c}_j)$ and $\bar{c}_{j+1} = \bar{c}_j - \alpha_j \nabla \bar{f}_j$.

2.1.2.1 Gradient descent method

The well-known gradient descent method is one of the simplest algorithms to minimise a function and often the method of choice for highly nonlinear problems, despite its slow convergence [8, p. 73]. The minimum of the function is being computed by taking a step in the direction of the anti-gradient in each iteration. Algorithm 1 gives a detailed description of the method. In Lines 2 and 3 the solution of the forward and backward problem are being computed for the current control $\bar{c}$. These results are needed for the computation of the gradient in Line 4. At iteration j, a step length α_j for the descent direction is chosen

Algorithm 1 Gradient descent method

Require: $\bar{y}_{\text{obs}}, \bar{c}_0, \varepsilon$
1: **for** $j = 0, 1, 2, \ldots$ **do**
2: $\quad \bar{y} \leftarrow \text{solveForwardProblem}(\bar{c})$
3: $\quad \bar{p} \leftarrow \text{solveAdjointProblem}(\bar{y}, \bar{c}, \bar{y}_{\text{obs}})$
4: $\quad v \leftarrow \text{computeGradient}(\bar{p}, \bar{y}, \bar{c}, \bar{y}_{\text{obs}})$
5: $\quad$ **if** $||v|| < \varepsilon$ **then**
6: $\quad\quad$ break
7: $\quad \alpha \leftarrow \text{chooseStepsize}(v)$
8: $\quad \bar{c} \leftarrow \bar{c} - \alpha v$

using a line search in Line 7 such that for some $\beta \in \mathbb{R}$, e.g. $\beta = 10^{-4}$, it satisfies the Armijo condition [47]

$$\bar{f}_{j+1} \leq \bar{f}_j - \beta\alpha_j \|\nabla \bar{f}_j\|_2^2, \tag{2.10}$$

where $\nabla \bar{f}_j := \nabla \bar{f}(\bar{c}_j)$. This condition demands that the next value $\bar{f}_{j+1}$ should be below the line $l(\alpha_j) = \bar{f}_j - \beta\alpha_j \|\nabla \bar{f}_j\|_2^2$ [47], which is illustrated in Figure 2.2. The line search

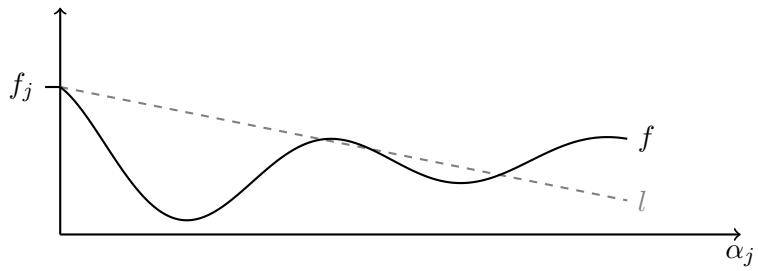

Figure 2.2: Sketch of the Armijo condition. All values α_j for which the value $\bar{f}_{j+1}$ is smaller than $l(\alpha_j)$ are acceptable.

method to determine the step size works as follows. Starting with some initial step length, we halve the value of α_j as long as the Armijo conditions do not hold. Once the condition is satisfied, the algorithm proceeds with the chosen step size. In case that there is no such value α_j found, the gradient descent algorithm stops. This would mean that, with the given gradient v, we cannot reach a control for which the objective functional has a significantly smaller value. This can happen if the discretisation of the gradient is not accurate enough or the tolerance ε was chosen too small. After determining the step size, the current control is being updated in Line 8 of Algorithm 1 by taking a step in direction of the anti-gradient. The algorithm stops as soon as the norm of the gradient is smaller than some previously chosen tolerance ε.

2.1.2.2 Alternatives to gradient descent

Quasi-Newton methods like the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm [48] can be a good alternative to the gradient descent method. This method requires more effort in the implementation and needs more memory, but it can have a superlinear convergence rate [47, Chapter 7]. In the BFGS method an approximation of the inverse Hessian is computed. This can require a lot of memory and computational cost as possibly large, non-sparse matrices have to be computed and saved [47]. This problem is addressed by the limited-memory (L-BFGS) algorithm [47, Chapter 7.2] that only saves some vectors to compute an approximation on the inverse of the Hessian which we denote by H_j in iteration j of the optimisation algorithm. In each iteration, there is information added and the oldest saved information deleted. The downside of this method is that the convergence rate is only linear [47].

L-BFGS works as follows. Let $d_j := H_j \nabla \bar{f}_j$, i.e. $-d_j$ is the descent direction. We denote the updated control by $\bar{c}_{j+1} := \bar{c}_j - \alpha_j d_j$ and the new value of the objective functional by $\bar{f}_{j+1} := \bar{f}(\bar{c}_{j+1})$. Let

$$s_j := \bar{c}_{j+1} - \bar{c}_j, \qquad y_j := \nabla \bar{f}_{j+1} - \nabla \bar{f}_j. \tag{2.11}$$

The initial approximation H_j^0 of the inverse Hessian can be chosen freely, but $H_j^0 = \gamma_j I$ has proven to be effective [47], where

$$\gamma_j := \frac{s_{j-1}^T y_{j-1}}{y_{j-1}^T y_{j-1}}. \tag{2.12}$$

Then the matrix-vector product $H_j \nabla \bar{f}_j =: r$ is computed using the L-BFGS *two-loop recursion* [47] which is shown in Algorithm 2. Within the two-loop recursion the last m values of s_j and y_j are needed for the approximation H_j. The larger m is chosen, the better is the approximation, but this would of course require more memory. For choosing

Algorithm 2 L-BFGS two-loop recursion

Require: $m, y_i, s_i, \nabla \bar{f}_j$
$q \leftarrow \nabla \bar{f}_j$
for $i = j-1, j-2, \ldots, j-m$ **do**
  $\rho_i \leftarrow \frac{1}{y_j^T s_i}$
  $a_i \leftarrow \rho_i s_i^T q$
  $q \leftarrow q - a_i y_i$
$r \leftarrow H_j^0 q$
for $i = j-m, j-m+1, \ldots, j-1$ **do**
  $b \leftarrow \rho_i y_i^T r$
  $r \leftarrow r + s_i(a_i - b)$

step lengths in a BFGS method, one should use Wolfe or strong Wolfe conditions instead of Armijo line search in order to ensure stable updates [47]. We take the following from Nocedal and Wright [47]. Note the changed sign in the update as we defined $-d_j$ as the descent direction. The strong Wolfe conditions, which we will use for L-BFGS, read

$$\bar{f}(\bar{c}_j - \alpha_j d_j) \leq \bar{f}(\bar{c}_j) - \beta_1 \alpha_j \nabla \bar{f}(\bar{c}_j)^T d_j \tag{2.13a}$$

$$|\nabla \bar{f}(\bar{c}_j - \alpha_j d_j)^T d_j| \leq \beta_2 |\nabla \bar{f}(\bar{c}_j)^T d_j| \tag{2.13b}$$

with $0 < \beta_1 < \beta_2 < 1$. Equation (2.13a) is called the *sufficient decrease condition* and is a generalisation of the Armijo condition (2.10). Equation (2.13b) demands that the slope of the reduced objective functional for step length α_j should be at least β_2 times the slope for step length zero and is therefore called the *curvature condition*. With this condition, we can expect further decrease of f when moving along the descent direction $-d_j$.

There exists a step length satisfying the strong Wolfe conditions if f is smooth and bounded from below [47, Lemma 3.1]. An initial step length of 1 is a natural choice for a Newton direction [47]. Typical choices for the other parameters would be $\beta_1 = 10^{-4}$ and $\beta_2 = 0.9$ for Newton or quasi-Newton methods [47]. For all computations with L-BFGS in the following chapters these values are being used.

2.1.2.3 Taylor remainder test

The computed gradient and adjoint can be verified using the Taylor remainder convergence test [49]. It is based on the following observation. For any perturbation $\tilde{c}$ we have that

$$|f(c + h\tilde{c}) - f(c)| \to 0 \quad \text{at } \mathcal{O}(|h|), \tag{2.14}$$

but

$$|f(c + h\tilde{c}) - f(c) - h\tilde{c}^T \nabla f(c)| \to 0 \quad \text{at } \mathcal{O}(|h|^2). \tag{2.15}$$

This means, if the discretised version of Equation (2.15) holds true for the numerical gradient v, it must have been computed correctly and consequently also the adjoint solution. This test is very sensitive to small errors in the adjoint [49]. Numerically, the Taylor test can be performed as described in Algorithm 3. Starting with an initial perturbation h_{ini} which is halved in each iteration, the corresponding remainder and its logarithm to basis 2, denoted by err_i, is being computed. The order is the difference of two consecutive errors err_i and err_{i-1}. Finally, the algorithm checks if all the computed orders are close to 2.

Algorithm 3 Taylor remainder convergence test

1: **for** $i = 1$ to $i_{\max}$ **do**
2: $\quad h \leftarrow h_{\text{ini}} \cdot 2^{-i}$
3: $\quad remainder \leftarrow |\bar{f}(\bar{c} + h\tilde{c}) - \bar{f}(\bar{c}) - h\tilde{c}^T v|$
4: $\quad err_i \leftarrow log(remainder, 2)$
5: **for** $i = 1$ to $i_{\max} - 1$ **do**
6: $\quad order_i \leftarrow err_i - err_{i-1}$
7: **if** all$(order) > 2 - \epsilon$ **then return** True

2.2 Shallow water equations

SWE are hyperbolic PDEs that can be used to describe e.g. tsunami waves, dam break waves as well as dynamics in the atmosphere [50]. Le Méhauté [51, Chapter 15, Figure 15-7] gives a good overview on different types of waves and states that shallow water waves are those with a ratio of at most 0.05 from water depth to wavelength. Here, we use the one-dimensional SWE to describe the water depth h and depth-averaged velocity u

in a channel with rectangular cross-section. In the following, the spatial domain is a one-dimensional interval $[L, R] =: \Omega$. We consider a possibly non-flat bottom topography b - usually called *bathymetry* - which does not change over time. This will be the control in our PDE-constrained optimisation problem.

The probably most common form of the nonlinear SWE is the conservative form [52]

$$\begin{pmatrix} h \\ hu \end{pmatrix}_t + \begin{pmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \end{pmatrix}_x = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \tag{2.16}$$

where g is the gravitational acceleration. A non-flat bathymetry b will lead to an additional source term $-ghb_x$ in the second component of the right hand side [53]. As we will use discretisation methods that do not require the SWE to be given in conservation law, we will use the non-conservative form. This simplifies the derivation of the corresponding adjoint equation. By plugging in h_t from the first into the second equation and dividing by h - supposing it is non-zero - we obtain the non-conservative form of the SWE. Often, a bottom friction term is included [54, 55]. Here, we will use a linear bottom friction term and denote the corresponding coefficient by κ. Then the SWE with initial conditions read

$$\begin{aligned} \begin{pmatrix} h \\ u \end{pmatrix}_t + \begin{pmatrix} hu \\ \frac{1}{2}u^2 + gh \end{pmatrix}_x &= \begin{pmatrix} 0 \\ -gb_x - \kappa u \end{pmatrix} \\ h(\cdot, 0) &= h_0 \quad \text{on } \Omega \\ u(\cdot, 0) &= 0 \quad \text{on } \Omega. \end{aligned} \tag{2.17}$$

The derivative of the bathymetry that appears in the source term of the SWE has an influence on the water depth. We want to use this fact to infer the bathymetry from observations of the free surface elevation $H = h + b$. We denote the observation by H_{obs}. A sketch of the relation between the bathymetry b, the water depth h and the free surface elevation H can be found in Figure 2.3. In the PDE-constrained optimisation problem, the

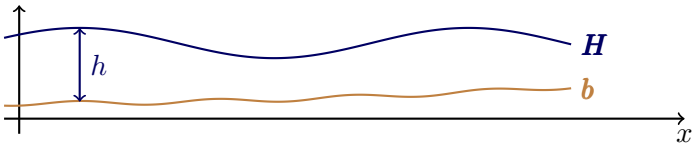

Figure 2.3: Free surface elevation H, water depth h and bathymetry b.

control is the bathymetry b and the state is the free surface elevation H. We will work on the SWE with different boundary conditions. The derivation of the corresponding adjoint problems that is needed to solve the PDE-constrained optimisation problem will be shown in the following chapters. The next section will focus on the numerical methods that we use to compute the solutions of the SWE and its adjoint.

2.3 Numerical solution of the PDEs

For the discretisation of forward and adjoint SWE a suitable numerical method has to be chosen. Here we will use spectral methods for the spatial discretisation. These converge fast but are limited to simple geometries [56]. The latter is not an issue in our case as

we work on one-dimensional finite intervals. Spectral methods work best for cases where smooth solutions are expected [57]. SWE can develop shocks after some time, but we only work with setups where this is not the case.

2.3.1 Spectral methods

The following explanations on spectral methods are taken from Burns et al. [56] and Boyd [57]. Spectral methods discretise functions using a set of basis functions. Consider an orthogonal basis $\{\phi_n\}$ with associated inner product $(\cdot,\cdot)_\phi$ such that

$$(\phi_n, \phi_m)_\phi = \delta_{n,m}. \tag{2.18}$$

Then, a function f can be represented by

$$f(x) = \sum_{n=0}^{\infty} f_n^\phi \phi_n(x), \tag{2.19}$$

where

$$f_n^\phi = \frac{(\phi_n, f)_\phi}{(\phi_n, \phi_n)_\phi} \tag{2.20}$$

are the coefficients. One example is the Fourier basis

$$\phi_k(x) = \exp(ikx) \tag{2.21}$$

on the interval $[0, 2\pi]$. The best choice for a basis are usually Chebyshev polynomials, unless the spatial domain is periodic, in which case the Fourier basis is better [57]. In all cases, the derivatives of the basis functions should be a linear combination of few basis functions. For instance, the first spatial derivatives of the Fourier basis functions are

$$\partial_x \phi_k(x) = ik\phi_k(x). \tag{2.22}$$

In this way, we obtain a discretisation of spatial derivatives in a PDE. The resulting semi-discrete equation can be solved with appropriate numerical solvers.

2.3.2 Runge-Kutta methods

An ordinary differential equation

$$y' = f(t, y), \quad y(t_0) = y_0 \tag{2.23}$$

can be solved numerically using *Runge-Kutta methods*. Let $b_i,\ a_{ij} \in \mathbb{R},\ i, j = 1, \ldots, s$ and $c_i = \sum_{j=1}^{s} a_{ij}$ and let $h > 0$ be the step length, i.e. $t_{n+1} = t_n + h$. An s-stage Runge-Kutta method is defined by [58]

$$\begin{aligned} k_i &= f\left(t_n + c_i h, y_n + h \sum_{j=1}^{s} a_{ij} k_j\right), \quad i = 1, \ldots, s \\ y_{n+1} &= y_n + h \sum_{i=1}^{s} b_i k_i, \end{aligned} \tag{2.24}$$

where k_i are called the *stages* [59]. The method is called *implicit* if $a_{ij} \neq 0$ for $i \leq j$ [59]. When applying the scheme, we obtain approximations

$$y(t_n) \approx y_n. \tag{2.25}$$

When using spectral methods, the solution of the semi-discrete PDE can be computed using Runge-Kutta methods.

2.3.3 Dedalus

For our computations, we use the software package Dedalus [56] which is a spectral framework. It is open source and written in Python. Besides boundary value problems and generalised eigenvalue problems, Dedalus can be used to solve initial value problems of the form

$$\begin{aligned} u_t + Lu &= \mathcal{N}(u) \\ u(0) &= u_0 \end{aligned} \tag{2.26}$$

with L being a linear operator and $\mathcal{N}$ denoting the nonlinearity of the equation. Dedalus allows to enter the equations as strings as can be seen in Figure 2.4. Everything that is written on the left hand side in the equations will be solved implicitly and the right hand side will be treated explicitly. Several time steppers, e.g. Runge-Kutta methods of different orders, are already included in the Dedalus package. One advantage of Dedalus

```
problem.add_equation("dt(h) = -dx(hu)")
problem.add_equation("dt(u) = -1/2*dx(u**2)
+ g*dx(h) - g*dx(b)")
```

Figure 2.4: Shallow water equations in Dedalus.

is that spatial derivatives are being determined analytically, because these can also be expressed in the used basis, see e.g. Equation (2.22). They can be computed with the built-in function `Differentiate`.

Depending on the boundary conditions of the problem that needs to be solved, there are different types of bases available. For periodic boundary conditions, one should choose a Fourier basis, either `RealFourier` or `ComplexFourier`. For other types of boundary conditions such as Dirichlet, the `Chebyshev` basis can be used. The Dedalus documentation[1] provides further and more detailed information and example scripts. Dedalus also allows for parallelisation with Message Passing Interface (MPI) if the spatial domain is at least two-dimensional. This does not apply in our case though as the spatial domain that is considered for the bathymetry reconstruction is only one-dimensional.

[1] https://dedalus-project.readthedocs.io/en/latest/

Chapter 3

Bathymetry reconstruction

In this chapter, we will derive adjoint equations and gradients in order to numerically perform a bathymetry reconstruction with SWE for different types of boundary conditions. We will use the methods introduced in Chapter 2, using the FOTD approach. In Section 3.1 we will consider SWE with periodic boundary conditions and reconstruct a sinusoidal and a Gaussian shaped bathymetry from observations that were previously computed. Section 3.2 will treat SWE with time-dependent Dirichlet boundary conditions which arise from an experimental setup in a wave flume. After inferring the bathymetry from simulated observations, we will use actual measurements for the reconstruction.

3.1 Shallow water equations with periodic boundary conditions

The derivation of the adjoint problem in the case of periodic boundary conditions is simpler than for other boundary conditions. We can include the periodic conditions in the function space. Let $Q = \Omega \times [0,T]$, $\Omega = [L,R]$ with $L, R \in \mathbb{R}$. In this section, we assume the state function and adjoint to be in $H^1_\#(\Omega) \times H^1([0,T])$, where $H^1_\#(\Omega)$ is the Sobolev space that contains all H^1-functions f with $f(L) = f(R)$. The bathymetry b is assumed to be in $H^2_\#(\Omega)$, because we need a higher regularity, as we will see later.

3.1.1 Formulation of the PDE-constrained optimisation problem

We will formulate the optimisation problem constrained by the SWE (2.17) and derive the corresponding continuous adjoint problem and the continuous gradient. To this end, we apply the Lagrange technique as explained in Chapter 2.

3.1.1.1 Objective functional and Lagrangian

We observe the state function $H = h + b$ on the whole interval $[0, T]$ and in all of Ω. This observation is denoted by H_{obs}. We define the objective functional

$$\begin{aligned} J(b, H) :=& \frac{\gamma}{2} \int_Q (H - H_{\text{obs}})^2 \, d(x,t) + \frac{\delta}{2} \int_\Omega (H(\cdot, T) - H_{\text{obs}}(\cdot, T))^2 \, dx \\ &+ \frac{\lambda_1}{2} \int_\Omega b^2 \, dx + \frac{\lambda_2}{2} \int_\Omega b_x^2 \, dx \end{aligned} \tag{3.1}$$

with $\gamma = \delta = 0.5$ and $\lambda_{1/2} > 0$. The first two terms measure the error to the observation. Note that in the first term the set $\{T\} \times \Omega$ has measure zero and therefore does not contribute to the value of the integral. This means, that the mismatch at the final time has no influence on the value of the first term. Thus, we include the second term in order to take the mismatch to the observation at the final time into account. We will also see in the following that this term contributes to the initial condition for the adjoint problem. The last two terms in the functional J are regularisation terms which are needed for better smoothness properties of the optimal control [8]. For $\lambda_1 = \lambda_2$ we would regularise with the H^1-norm of the bathymetry, but in order to have more flexibility we allow the parameters to be different. We do not put any restrictions on the control and write our optimisation problem as

$$\min_{b \in H^2_\#(\Omega)} J(b, H) \quad \text{subject to (2.17)}. \tag{3.2}$$

As explained in Chapter 2 we will use the Lagrangian to derive the adjoint equation and the gradient. With Lagrange multipliers $p_i,\ i \in \{1, 2, 3\}$, the Lagrangian reads

$$\begin{aligned} \mathcal{L}(h, u, b, p) = & \frac{\gamma}{2} \int_Q (h + b - H_{\text{obs}})^2 \, d(x,t) + \frac{\delta}{2} \int_\Omega (h(\cdot, T) + b - H_{\text{obs}}(\cdot, T))^2 \, dx \\ &+ \frac{\lambda_1}{2} \int_\Omega b^2 \, dx + \frac{\lambda_2}{2} \int_\Omega b_x^2 \, dx - \int_Q (h_t + (hu)_x) p_1 \, d(x,t) \\ &- \int_Q \left(u_t + \left(\frac{1}{2} u^2 + gh \right)_x + g b_x + \kappa u \right) p_2 \, d(x,t) \\ &- \int_\Omega (h(\cdot, 0) - (H_{\text{obs}}(\cdot, 0) - b)) p_3 \, dx. \end{aligned} \tag{3.3}$$

The first part of the Lagrangian is the objective functional. After moving the right hand side of the SWE (2.17) to the left, the second part of the Lagrange function is the L^2 scalar product of the Lagrange multipliers with the left hand sides of the SWE.

3.1.1.2 Derivation of the adjoint equations and the descent direction

For the numerical computation of the descent direction we need the adjoint equations as the corresponding solution contributes to the gradient of the reduced objective functional. The adjoint problem can be derived from the necessary optimality conditions (2.8) which read

$$\begin{aligned} \mathcal{L}_h(h, u, b, p)\varphi = 0 \\ \mathcal{L}_u(h, u, b, p)\varphi = 0 \end{aligned} \tag{3.4}$$

for all $\varphi \in C_0^\infty(Q)$. Thus, we have to compute the partial derivatives of the Lagrange function. In order to move derivatives away from φ we use integration by parts and obtain

$$\begin{aligned}
\mathcal{L}_h(h,u,b,p)\varphi &= \gamma \int_Q (h+b-H_{\text{obs}})\,\varphi \; d(x,t) \\
&\quad + \delta \int_\Omega (h(\cdot,T)+b-H_{\text{obs}}(\cdot,T))\,\varphi(\cdot,T) \; dx \\
&\quad - \int_Q \varphi_t p_1 \; d(x,t) - \int_Q (\varphi u)_x \, p_1 \; d(x,t) - \int_Q g\varphi_x p_2 \; d(x,t) \\
&\quad - \int_\Omega \varphi(\cdot,0) p_3 \; dx \\
&= \gamma \int_Q (h+b-H_{\text{obs}})\,\varphi \; d(x,t) \\
&\quad + \delta \int_\Omega (h(\cdot,T)+b-H_{\text{obs}}(\cdot,T))\,\varphi(\cdot,T) \; dx \\
&\quad + \int_Q \varphi p_{1,t} \; d(x,t) - \int_\Omega [\varphi p_1]_0^T \; dx \\
&\quad + \int_Q \varphi u p_{1,x} \; d(x,t) - \int_0^T [\varphi u p_1]_L^R \; dt \\
&\quad + \int_Q g\varphi p_{2,x} \; d(x,t) - \int_0^T [g\varphi p_2]_L^R \; dt - \int_\Omega \varphi(\cdot,0) p_3 \; dx.
\end{aligned} \tag{3.5}$$

Using the optimality condition (2.8) we set this to zero for all $\varphi \in C_0^\infty(Q)$. This yields

$$\begin{aligned}
&\int_Q \gamma\,(h+b-H_{\text{obs}})\,\varphi + \varphi p_{1,t} + \varphi u p_{1,x} + g\varphi p_{2,x} \; d(x,t) = 0 \\
\Leftrightarrow &\int_Q (\gamma\,(h+b-H_{\text{obs}}) + p_{1,t} + u p_{1,x} + g p_{2,x})\,\varphi \; d(x,t) = 0
\end{aligned}$$

and using the fundamental lemma of calculus of variations we get the first part of the adjoint equation

$$p_{1,t} + u p_{1,x} + g p_{2,x} = -\gamma\,(h+b-H_{\text{obs}})\,. \tag{3.6}$$

As the term in (3.5) is zero for all $\varphi \in C_0^\infty(Q)$, this is also true if we leave out the condition $\varphi(\cdot,T)=0$. By doing this, we obtain the final condition for p_1

$$\begin{aligned}
&\int_\Omega \delta\,(h(\cdot,T)+b-H_{\text{obs}}(\cdot,T))\,\varphi(\cdot,T) \; dx - \int_\Omega \varphi(\cdot,T) p_1(\cdot,T) \; dx = 0 \\
&\Leftrightarrow p_1(\cdot,T) = \delta\,(h(\cdot,T)+b-H_{\text{obs}}(\cdot,T))\,.
\end{aligned} \tag{3.7}$$

If we now let $\varphi(\cdot,0)$ vary we get

$$\int_\Omega \varphi(\cdot,0) p_1(\cdot,0) \; dx - \int_\Omega \varphi(\cdot,0) p_3 \; dx = 0 \tag{3.8}$$

and thus we have

$$p_1(\cdot,0) = p_3. \tag{3.9}$$

For the second part of the adjoint equation we need the derivative

$$\begin{aligned}\mathcal{L}_u(h,u,b,p)\varphi &= -\int_Q (h\varphi)_x p_1 \; d(x,t) - \int_Q \left(\varphi_t + (u\varphi)_x + \kappa\varphi\right) p_2 \; d(x,t) \\ &= \int_Q h\varphi p_{1,x} \; d(x,t) - \int_0^T [h\varphi p_1]_L^R \; dt + \int_Q \varphi p_{2,t} \; d(x,t) - \int_\Omega [\varphi p_2]_0^T \; dx \\ &\quad + \int_Q u\varphi p_{2,x} \; d(x,t) - \int_0^T [u\varphi p_2]_L^R \; dt - \int_Q \kappa\varphi p_2 \; d(x,t).\end{aligned} \tag{3.10}$$

Setting this to zero for all $\varphi \in C_0^\infty(Q)$, we get

$$\int_Q h\varphi p_{1,x} + \varphi p_{2,t} + u\varphi p_{2,x} - \kappa\varphi p_2 \; d(x,t) = 0$$

and using again the fundamental lemma of calculus of variations we get the second part of the adjoint equation

$$p_{2,t} + hp_{1,x} + up_{2,x} - \kappa p_2 = 0. \tag{3.11}$$

Now we let $\varphi(\cdot, T)$ vary to obtain the final condition for p_2, which is

$$p_2(\cdot, T) = 0. \tag{3.12}$$

Note that as we have final conditions for the adjoint problem, it runs backwards in time as already mentioned in Chapter 2. To reformulate it as an initial value problem, we define a new (backward) time variable $\tau := T - t$ and the corresponding functions

$$\begin{aligned}\tilde{p}(x,\tau) &:= p(x, T-t) \\ \tilde{h}(x,\tau) &:= h(x, T-t) \\ \tilde{u}(x,\tau) &:= u(x, T-t).\end{aligned}$$

Now we can rewrite the adjoint equations as the initial value problem

$$\begin{aligned}\tilde{p}_{1,\tau} - \tilde{u}p_{1,x} - g\tilde{p}_{2,x} &= \gamma\left(\tilde{h} + b - \tilde{H}_{\text{obs}}\right) \\ \tilde{p}_{2,\tau} - \tilde{h}\tilde{p}_{1,x} - \tilde{u}\tilde{p}_{2,x} &= -\kappa\tilde{p}_2, \\ \tilde{p}_1(x,0) &= \delta\left(\tilde{h}(\cdot,0) + b - \tilde{H}_{\text{obs}}(\cdot,0)\right) \\ \tilde{p}_2(x,0) &= 0.\end{aligned} \tag{3.13}$$

As mentioned beforehand, we will need the solution of the adjoint problem (3.13) for the computation of the gradient which we will derive now. We obtain the L^2-gradient from

the condition $\mathcal{L}_b(h,u,b,p)\delta b = 0$. To this end, we compute the partial derivative

$$\begin{aligned}
\mathcal{L}_b(h,u,b,p)\delta b &= \gamma \int_\Omega \left(\int_0^T h - H_{\text{obs}}\ dt + bT \right) \delta b\ dx + \delta \int_\Omega \left(h(\cdot,T) + b - H_{\text{obs}}(\cdot,T)\right) \delta b\ dx \\
&\quad + \lambda_1 \int_\Omega b\delta b\ dx + \lambda_2 \int_\Omega b_x(\delta b)_x\ dx - \int_\Omega \int_0^T g p_2\ dt(\delta b)_x\ dx - \int_\Omega \delta b p_3\ dx \\
&= \gamma \int_\Omega \left(\int_0^T h - H_{\text{obs}}\ dt + bT \right) \delta b\ dx \\
&\quad + \int_\Omega \left(\delta\left(h(\cdot,T) + b - H_{\text{obs}}(\cdot,T)\right) + \lambda_1 b - \lambda_2 b_{xx}\right) \delta b\ dx \\
&\quad + \lambda [b_x \delta b]_L^R + g \int_\Omega \left(\int_0^T p_2\ dt \right)_x \delta b\ d(x,t) - g \left[\int_0^T \delta b p_2\ dt \right]_L^R \\
&\quad - \int_\Omega \delta b p_3\ dx \\
&= \int_\Omega \left(\gamma \int_0^T h - H_{\text{obs}}\ dt + \gamma bT \right) \delta b\ dx \\
&\quad + \int_\Omega \left(\delta\left(h(\cdot,T) + b - H_{\text{obs}}(\cdot,T)\right) + \lambda_1 b - \lambda b_{xx}\right) \delta b\ dx \\
&\quad + g \int_\Omega \int_0^T p_{2,x}\ dt \delta b\ d(x,t) - \int_\Omega \delta b p_3\ dx,
\end{aligned} \tag{3.14}$$

where we use the periodic boundary conditions. Setting this to zero we obtain

$$\begin{aligned}
\tilde{v} := \gamma \int_0^T h - H_{\text{obs}}\ dt + \gamma bT + \delta\left(h(\cdot,T) + b - H_{\text{obs}}(\cdot,T)\right) + \lambda_1 b - \lambda_2 b_{xx} \\
+ g \int_0^T p_{2,x}\ dt - p_1(\cdot,0).
\end{aligned} \tag{3.15}$$

As the bathymetry b is a H^1-function, the true gradient of the reduced objective functional is the H^1-gradient. We can determine it using the Fréchet-Riesz representation theorem [60]. We have

$$\begin{aligned}
\langle \mathcal{L}_b, \delta b \rangle_{H^1_\#(\Omega)^*, H^1_\#(\Omega)} &= (v, \delta b)_{H^1_\#(\Omega)} \\
&= \int_\Omega v\delta b\ dx + \int_\Omega \nabla v \nabla \delta b\ dx \\
&= \int_\Omega v\delta b\ dx - \int_\Omega \Delta v \delta b\ dx + \int_\Gamma v \delta b\ ds \\
&= \int_\Omega (v - \Delta v)\delta b\ dx \\
&= (v - \Delta v, \delta b)_{L^2(\Omega)}.
\end{aligned} \tag{3.16}$$

As we have $v, \delta b \in H^1_\#(\Omega)$, the integral over the boundary is zero. With the identification theorem, $\mathcal{L}_b(h,u,b,p)$ is identified with $\tilde{v}$ in $L^2(\Omega)$ and with v in $H^1_\#(\Omega)$. As the functional $\mathcal{L}_b$ is not only in $H^1_\#(\Omega)^* = H^{-1}(\Omega)$, but also in $L^2(\Omega)$, the dual pairing

$\langle \mathcal{L}_b, \delta b \rangle_{H^1_\#(\Omega)^*, H^1_\#(\Omega)}$ is equal to $\langle \mathcal{L}_b, \delta b \rangle_{L^2(\Omega)^*, L^2(\Omega)}$ [61, p. 283]. We used the latter to derive $\tilde{v}$ in Equation (3.14), so we have the equality

$$-\Delta v + v = \tilde{v}. \tag{3.17}$$

The solution v of this stationary PDE is the H^1-gradient.

3.1.2 Implementation

We will use the numerical methods that were explained in Chapter 2. We use gradient descent as well as L-BFGS to iterate towards the minimum of the reduced objective functional. In each iteration of the optimisation, we compute the solution of the forward and adjoint equation with Dedalus. We use the `RealFourier` basis, which is the basis in Dedalus that has to be used for real periodic functions. From the time steppers available in Dedalus we choose the Runge-Kutta method with the highest order, that is `RK443`. For the time integrals that appear in the L^2-gradient $\tilde{v}$ we use the Scipy function `integrate.simpson` and all derivatives are computed with the Dedalus function `Differentiate`. The equation for the H^1-gradient can be defined as a linear boundary value problem in Dedalus as shown in Figure 3.1 and does not require much time to solve as it is a stationary problem.

```
1 v = dist.Field(bases=xbasis)
2 vTilde = dist.Field(bases=xbasis)
3 def dx(A): return d3.Differentiate(A, xcoord)
4 vTilde['g'] = L2grad
5 problem = d3.LBVP([v], namespace=locals())
6 problem.add_equation("-dx(dx(v)) + v = vTilde")
```

Figure 3.1: Computation of the H^1-gradient in Dedalus.

3.1.3 Reconstructions

In the following we will see reconstructed bathymetries from previously computed observations. That is, we plug in the exact bathymetry b_{ex} into the forward equation (2.17) and numerically compute a fine solution (h_f, u_f). Then the observation is

$$H_{\text{obs}} = h_f + b_{\text{ex}}. \tag{3.18}$$

In the following we always define $\Omega = [L, R] = [0, 10]$, $T = 10$, $g = 9.81$ and $\kappa = 0.2$. The parameters in the objective functional we choose to be $\gamma = \delta = 0.5$ and $\lambda_1 = \lambda_2 = 10^{-7}$. To compute the observation we use 130 spatial grid points and a time step of $\delta t_f = 5 \times 10^{-5}\,\text{s}$. For the optimisation we use $M = 64$ spatial grid points and a time step of $\delta t = 0.01\,\text{s}$. Here, the tolerance for the stopping criterion for gradient descent and L-BFGS is set to $\varepsilon = 10^{-7}$ and the initial step size for the line search to $\alpha_{\text{ini}} = 16$.

3.1.3.1 Sinusoidal bathymetry

Consider the exact bathymetry

$$b_{\text{ex}}(x) = 0.05 \sin(0.2\pi x) + 0.1 \qquad \forall x \in \Omega \tag{3.19}$$

and the initial condition

$$h_0(x) = 0.05 \exp(-0.2(x-5)^2) + 0.35 - b(x) \qquad \forall x \in \Omega. \tag{3.20}$$

Note that the initial condition h_0 changes throughout the reconstruction process as the bathymetry b is different at each iteration. In order to show how much information is available for the bathymetry reconstruction the difference of the water height $H(b_{\text{ex}})$ for the sinusoidal bathymetry to the water height for zero bathymetry $H(0)$ is shown in Figure 3.2. The difference in water heights has amplitudes of at most 1.5 cm, whereas the amplitude in the initial condition is 5 cm.

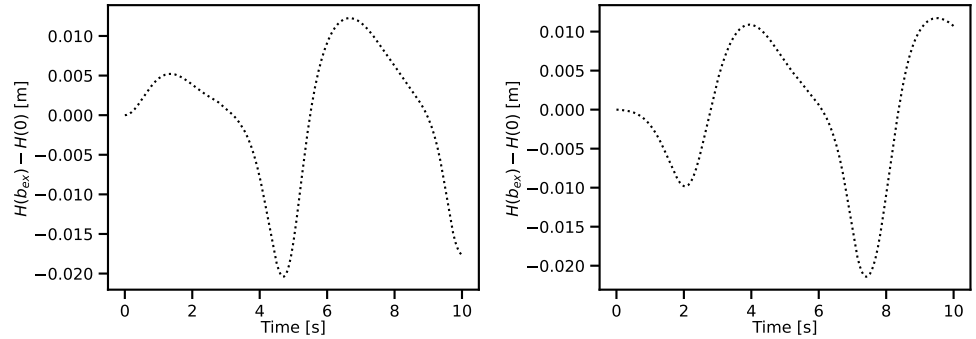

Figure 3.2: Difference in water height with and without the sinusoidal bathymetry (3.19) in the middle of the spatial domain (left) and the end of the spatial domain (right).

For the optimisation, we set the starting value for the bathymetry to

$$b_0(x) = 0 \tag{3.21}$$

for $x \in \Omega$. The errors, runtimes and number of iterations for the following reconstructions are collected in Table 3.1. The bathymetry b_{ex} and its reconstruction from gradient descent are shown in Figure 3.3 on the left. There is only a small difference between the bathymetries visible. The relative ℓ^2-error of the reconstruction is approximately 2.11%. On the right of Figure 3.3 the relative errors are plotted against the iteration counter. At first, the error decreases rapidly and then more and more slowly. The values of the objective functional, which are shown on the left of Figure 3.4 show a similar behaviour. As the mismatch decreases, the regularisation term gets closer to the value $f(b_{\text{ex}})$ at the exact bathymetry. This is to be expected if the optimisation works correctly, because the value $f(b_{\text{ex}})$ only consists of the regularisation up to rounding errors. Gradient descent took about 4.03 hours to run and terminated after 2579 iterations, when the norm of the gradient reached its tolerance ε. The norms of the gradient are shown in Figure 3.4 (right).

We perform another reconstruction with exactly the same parameters using L-BFGS with a memory of 10 descent directions. The corresponding reconstruction is shown in Figure 3.5 on the left together with the exact bathymetry. There are only small differences visible compared to the reconstruction with gradient descent. The plot with the relative ℓ^2-errors in Figure 3.5 is quite different, but this is not surprising since L-BFGS uses a different descent direction than gradient descent. Besides the fact that the values of the objective in Figure 3.6 on the left do not form a smooth looking graph, they decrease monotonically as they should. L-BFGS stopped after only 13 iterations as no step size

could be found in the line search with Wolfe conditions. This took about 135 seconds, which is only 1.4% of the runtime of gradient descent. But L-BFGS yields almost the same reconstruction. The relative ℓ^2-error in the reconstruction is 5.06% for L-BFGS versus 2.48% for gradient descent. Using more memory in L-BFGS does not improve this error.

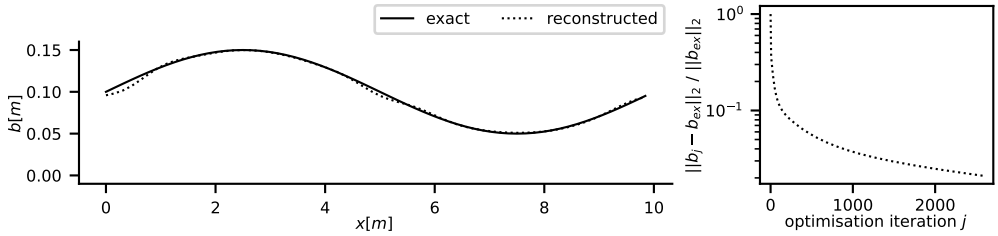

Figure 3.3: Reconstruction of the sinusoidal bathymetry (3.19) (left) and relative errors against iteration in gradient descent (right) for initial guess $b = 0$.

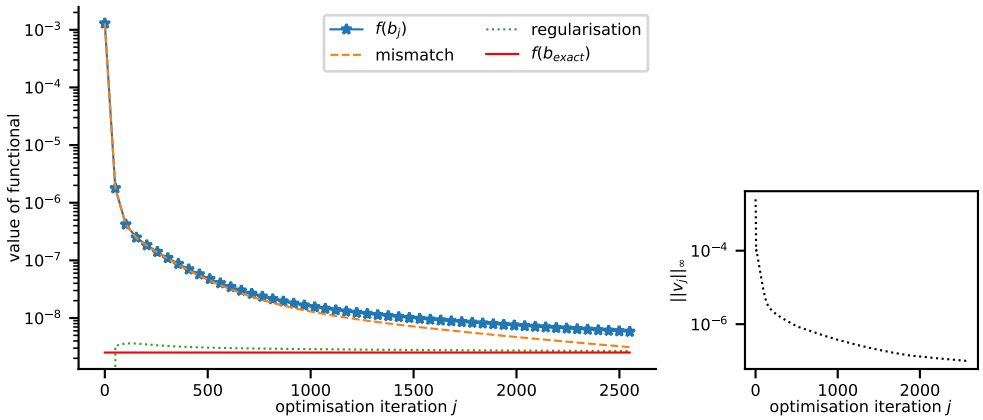

Figure 3.4: Norms of the gradients (right) and values of the objective functional (left) for the optimisation with gradient descent for initial guess $b = 0$. The values of the objective functional are the sum of the mismatch (orange dashed) and the regularisation term (green dashed). The value for the exact bathymetry is indicated by the red line.

Now we change the initial guess for the bathymetry to

$$b_0(x) = 0.1 \tag{3.22}$$

for $x \in \Omega$ and reconstruct the same bathymetry again with gradient descent and L-BFGS. Gradient descent took 2.67 hours runtime and terminated after 1527 iterations, because the stopping criterion was met. The relative error of the reconstruction, which can be seen in Figure 3.7, is 1.52%. The reconstruction with L-BFGS, which is shown in Figure 3.8, has almost the same relative ℓ^2-error, that is 1.58%. The algorithm stopped after 46 iterations as no more step size was found and took 319 seconds to run. Hence, L-BFGS was again much faster than gradient descent and yielded almost the same result. In comparison to the zero initial guess for the bathymetry, the error in the reconstruction went down for both gradient descent and L-BFGS, but for L-BFGS the difference is larger. Here, the initial guess is the average height of the true bathymetry and thus closer to it than the zero initial guess. This could be a reason for the better reconstruction.

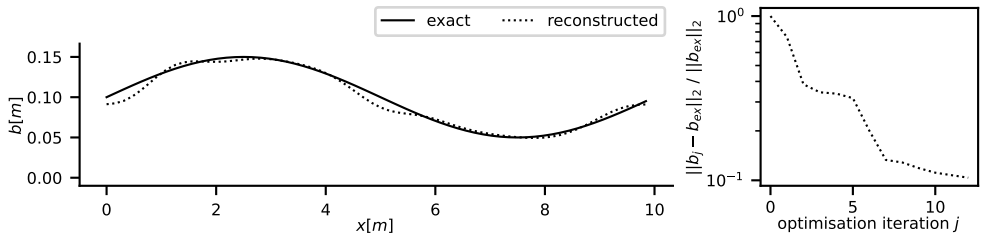

Figure 3.5: Reconstruction of the sinusoidal bathymetry (3.19) (left) and relative errors against iteration in L-BFGS (right) for initial guess $b = 0$.

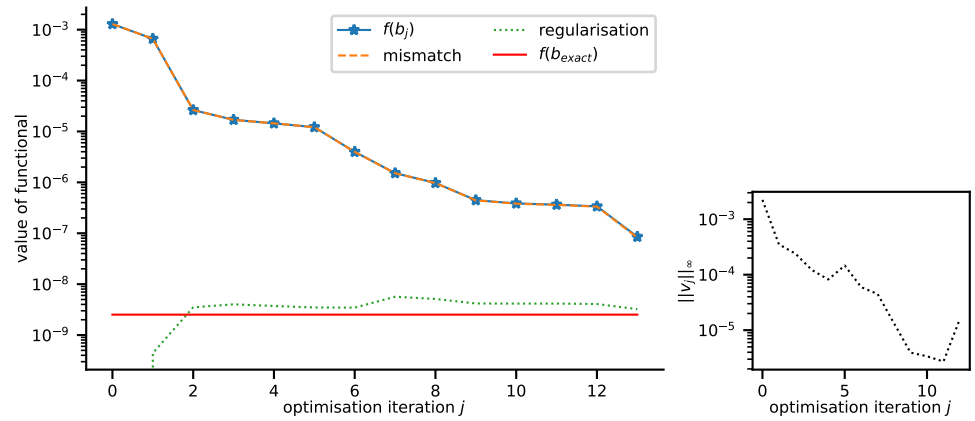

Figure 3.6: Norms of the gradients (right) and values of the objective functional (left) for the optimisation with L-BFGS for initial guess $b = 0$. The values of the objective functional are the sum of the mismatch (orange dashed) and the regularisation term (green dashed). The value for the exact bathymetry is indicated by the red line.

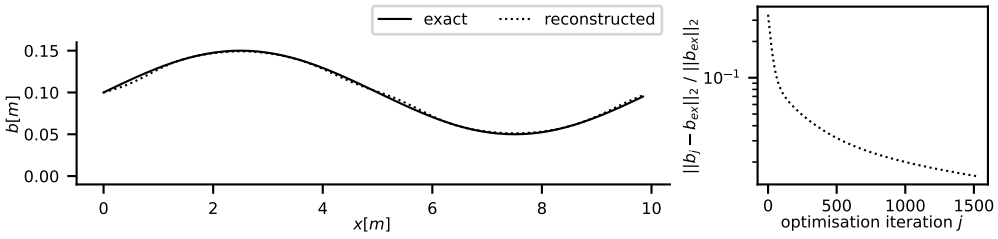

Figure 3.7: Reconstruction of the sinusoidal bathymetry (3.19) (left) and relative errors against iteration in gradient descent (right) for initial guess $b = 0.1$.

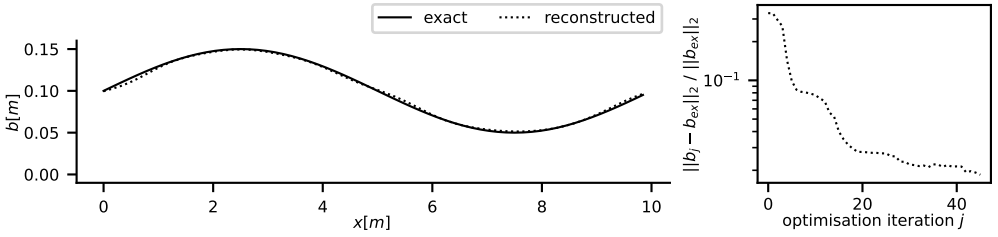

Figure 3.8: Reconstruction of the sinusoidal bathymetry (3.19) (left) and relative errors against iteration in L-BFGS (right) for initial guess $b = 0.1$.

Initial guess	Method	ℓ^2-error	Runtime [s]	Iterations
$b = 0$	Gradient descent	2.11%	14507	2579
	L-BFGS	5.06%	135	13
$b = 0.1$	Gradient descent	1.52%	9601	1527
	L-BFGS	1.58%	319	46

Table 3.1: Relative ℓ^2-errors, runtime and number of iterations for the reconstruction of the sinusoidal bathymetry (3.19) for gradient descent and L-BFGS.

3.1.3.2 Gaussian shape

Now consider a bathymetry that has a Gaussian shape, i.e.

$$b_{\text{ex}}(x) = 0.1e^{-(x-6)^2} \qquad \forall x \in \Omega. \tag{3.23}$$

The difference in the water height compared to zero bathymetry is even smaller than for the sinusoidal bathymetry, see Figure 3.9. That means, we have less information for the reconstruction, which makes it more challenging.

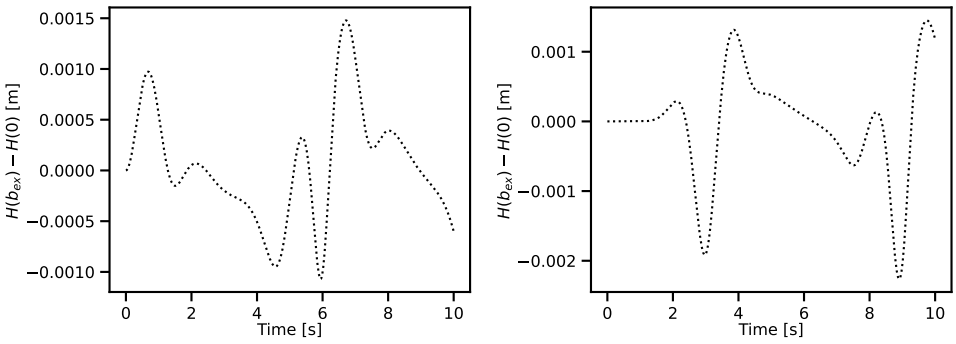

Figure 3.9: Difference in water height with and without the Gaussian shaped bathymetry (3.23) in the middle of the spatial domain (left) and the end of the spatial domain (right).

We use the same parameters and initial conditions as before, starting the optimisation with

$$b_0 = 0 \qquad \text{on } \Omega. \tag{3.24}$$

The reconstruction with gradient descent is shown in Figure 3.10 on the left. The exact bathymetry is matched very well except for some smaller oscillations. The algorithm stopped after 2689 iterations, because the stopping criterion was fulfilled. The runtime was very long, taking around 4.2 hours. The relative ℓ^2-errors, which can be seen in Figure 3.10 (right), decrease similarly as for the reconstruction of the sinusoidal bathymetry in Figure 3.3. The error of the final reconstruction is about 8.2%. The values of the objective functional in Figure 3.11 decrease fast for the first few iterations and then more and more slowly. The value of the regularisation term almost reaches the value $f(b_{\text{ex}})$. The mismatch crosses this value towards the end of the optimisation. All step sizes were chosen by the step size control to be the largest possible value which is $\alpha_{\text{ini}} = 16$.

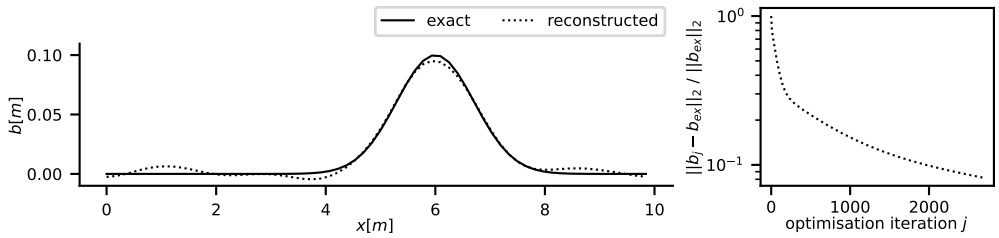

Figure 3.10: Reconstruction of the Gaussian bathymetry (3.23) (left) and relative errors against iteration in gradient descent (right).

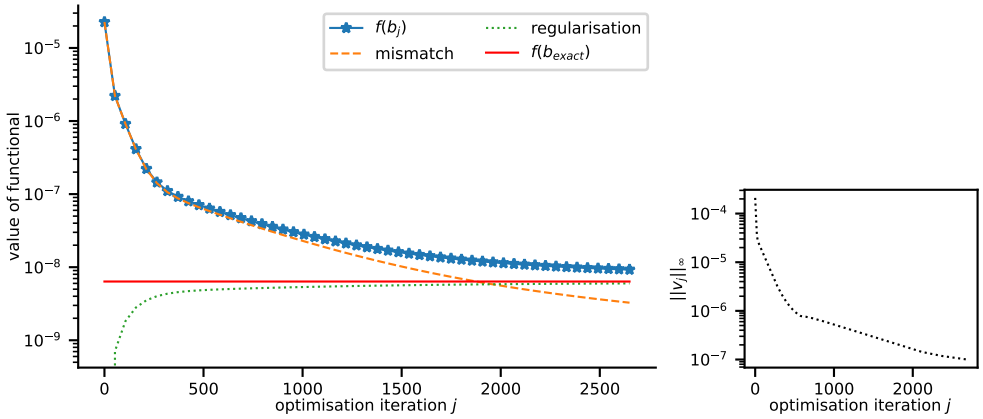

Figure 3.11: Norms of the gradients (right) and values of the objective functional (left) for the optimisation with gradient descent. The values of the objective functional are the sum of the mismatch (orange dashed) and the regularisation term (green dashed). The value for the exact bathymetry is indicated by the red line.

Figure 3.12 (left) shows the reconstruction that was obtained when using L-BFGS. It has very small differences to the bathymetry that was inferred using gradient descent. After 30 iterations L-BFGS stopped with a relative ℓ^2-error of 7.14%, even smaller than the error of the gradient descent reconstruction. The runtime of 245 seconds is only about 1.6% of the runtime for gradient descent. The relative ℓ^2-errors to the exact bathymetry, see Figure 3.12 (right), go down much faster than in the gradient descent optimisation. From Figure 3.13 it can be observed that the value of the objective functional gets very close to $f(b_{\text{ex}})$. L-BFGS stopped, because the stopping criterion was fulfilled. In Figure 3.13 (right) it can be seen that the norm of the gradient reaches the tolerance $\varepsilon = 10^{-7}$. The errors of the reconstruction, runtimes and number of iterations are also shown in Table 3.2. Compared to the sinusoidal bathymetry, the errors are larger. The reason is probably that the signal from which the bathymetry is inferred is about ten times smaller, see Figure 3.2 and Figure 3.9 for the differences in the water height with and without bathymetry.

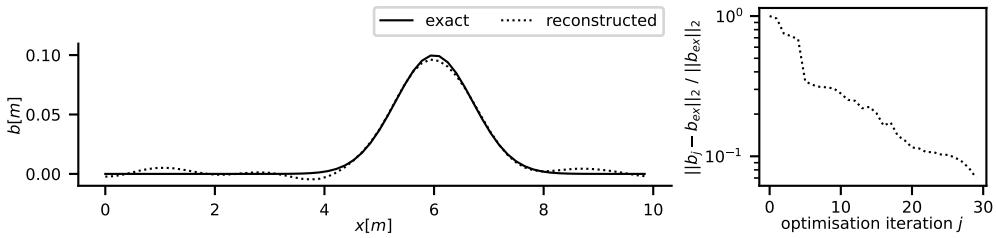

Figure 3.12: Reconstruction of the Gaussian bathymetry (3.23) (left) and relative errors against iteration in L-BFGS (right).

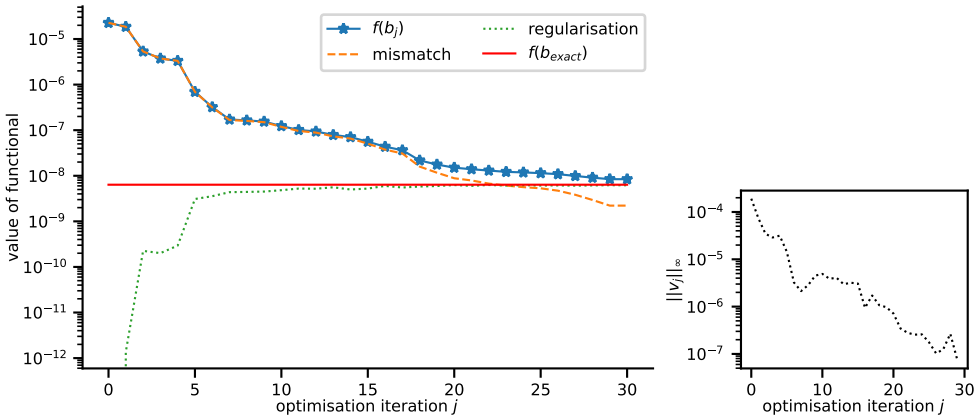

Figure 3.13: Norms of the gradients (right) and values of the objective functional (left) for the optimisation with L-BFGS. The values of the objective functional are the sum of the mismatch (orange dashed) and the regularisation term (green dashed). The value for the exact bathymetry is indicated by the red line.

Initial guess	Method	ℓ^2-error	Runtime [s]	Iterations
$b = 0$	Gradient descent	8.20%	15098	2689
	L-BFGS	7.14%	245	30

Table 3.2: Relative ℓ^2-errors, runtime and number of iterations for the reconstruction of the Gaussian bathymetry (3.23) for gradient descent and L-BFGS.

3.2 Bathymetry reconstruction in a wave flume

In the previous section we have seen that for periodic boundary conditions the FOTD-approach yields fairly good reconstructions from little information. The Gaussian bathymetry could be reconstructed with error less than 10% even though the difference in wave height with and without bathymetry was only around 1% of the largest wave amplitude. In this section we use this approach for other boundary conditions related to an experimental setup, which leads to additional challenges. We reconstruct a roughly Gaussian shaped bathymetry in a wave flume based at the Institute of Mechanics and Ocean Engineering at Hamburg University of Technology, see Figure 3.14.

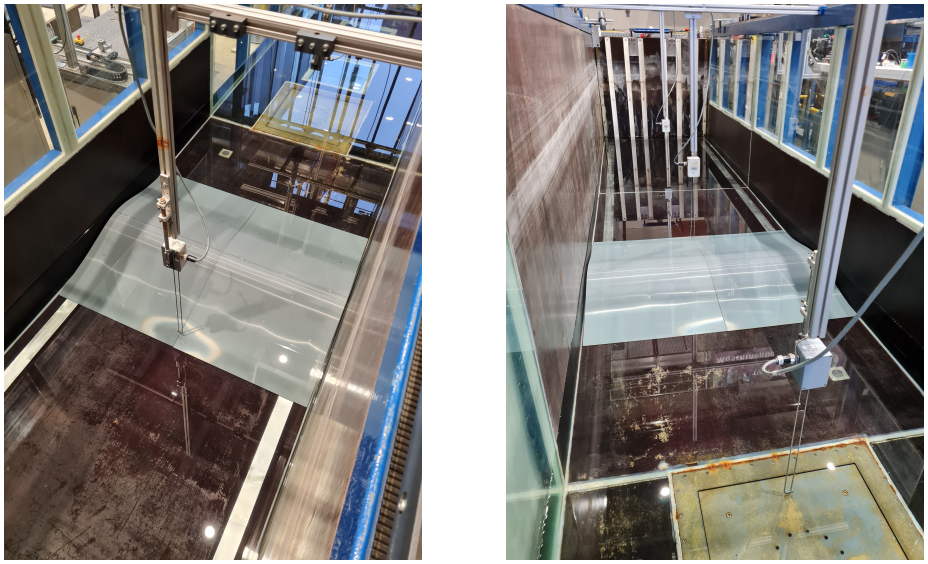

Figure 3.14: Photographs of the wave flume in the Institute of Mechanics and Ocean Engineering at Hamburg University of Technology with installed bathymetry from two different perspectives [62].

First, we will use a simulated observation and then experimental data. As in the previous section, we compute a very fine solution of the forward problem for the exact bathymetry and use the corresponding free surface elevation as our observation H_{obs}. Then, we use the experimental data we obtain from measurements in the wave flume. All simulations are being computed with Dedalus. We will assume all functions h, u, b and the observation to be in the space $H^1(\Omega)$. Compared to the previous section, the main difference in the problem formulation are the boundary conditions which have to be chosen according to the experimental setup. Most of the content in this section is published in Angel et al. [20]. Also the code that was used for the discretisation and optimisation in the paper is available online [63].

3.2.1 Experimental setup

First, we describe the wave flume and the bathymetry which we want to infer from data of the water depth. For additional information, we refer to the very detailed description of the experiment that is available online [62]. A sketch of the wave flume is depicted in

Figure 3.15. The flume has a total length of 12 m, where at time 0 s the water is at rest. After 30 s, waves are being generated by a wave flap that moves in a previously determined frequency within the flap angle γ_{wf}. The flap is positioned at 0 m and at the end of the flume, there is a "beach" installed to mitigate reflections from the end of the flume. There are four sensors in the water at positions 1.5 m, 3.5 m, 5.5 m, 7.5 m that can detect the free surface level over time. In the experiment, measurements of the free surface were taken over a time of 100 s, where the flap starts moving after 30 s.

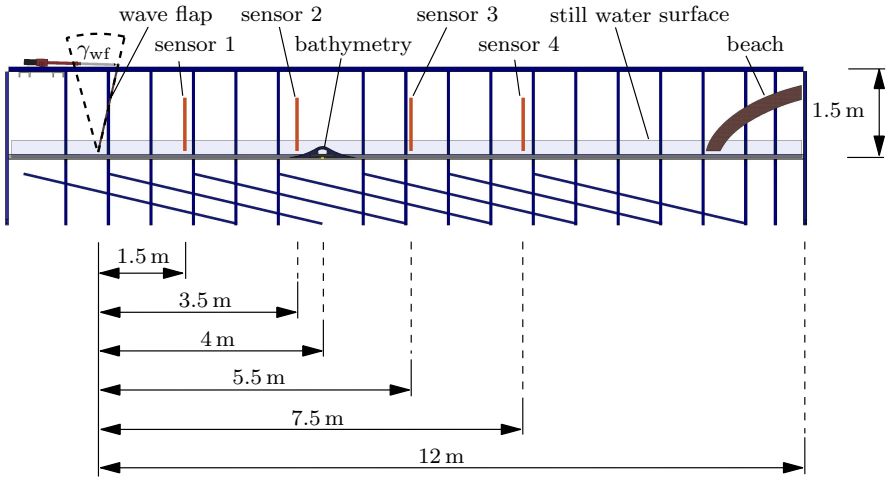

Figure 3.15: Sketch of the wave flume in the Institute of Mechanics and Ocean Engineering at Hamburg University of Technology with the installed bathymetry, sensor positions and wave flap with flap angle γ_{wf} [20].

The water at rest has a depth of 0.3 m, so the SWE could yield a sufficiently accurate description of the water depth and velocity. This will be verified in Subsection 3.2.5. For the reconstruction, a hill of a roughly Gaussian shape with height 0.2 m was placed into the flume, with the maximum of the hill at 4 m. To construct the physical hill, three skateboard ramps were attached next to each other and covered with a very thin PVC plate, which can be seen in Figure 3.16. Weights were placed below the PVC plate to prevent the bathymetry from floating. In order to obtain observations for the bathymetry recon-

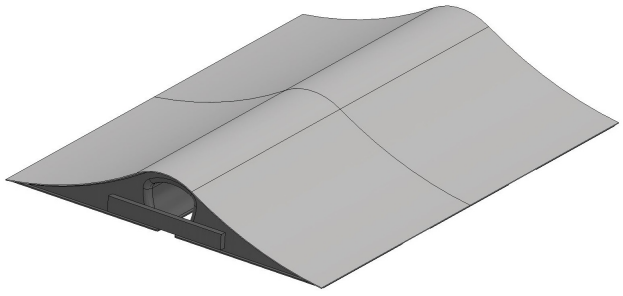

Figure 3.16: Sketch of the bathymetry that was used in the experiment [62].

struction, measurements with and without bathymetry were performed 20 times each. To minimise the effect of noise within the optimisation, we took the mean of all measurements

and used this as the observation H_{obs}. Measurement data from one of the experiments is shown in Figure 3.17.

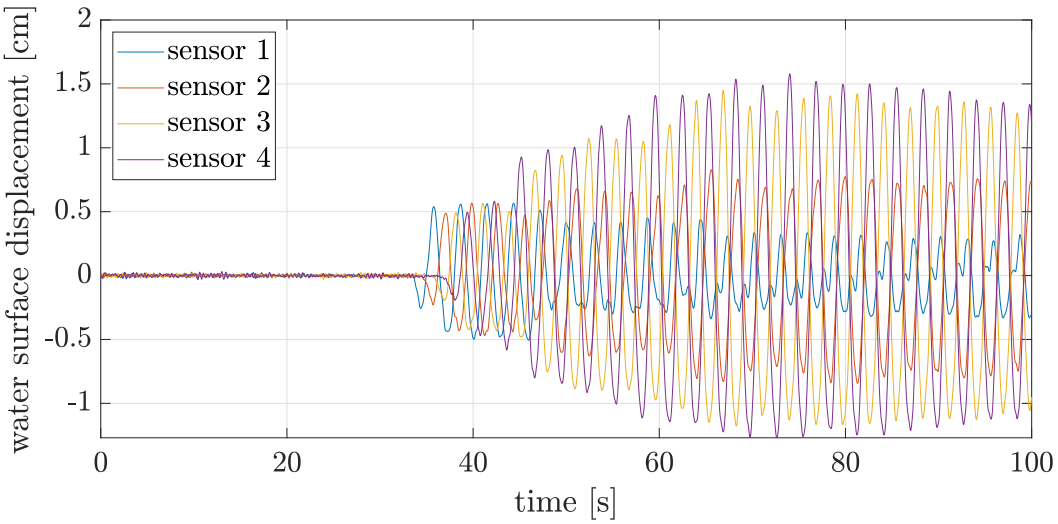

Figure 3.17: Measurement data from all four sensors [62].

Note that after about 45 s the wave amplitudes at sensor 3 and 4 increase significantly. The reason are most likely wave reflections that occur despite the beach being installed at the end of the wave flume. These additional waves are not visible in the measurement data from sensor 1 which is the closest to the wave flap.

3.2.2 Forward problem

To model the water height and velocity in the wave flume we use Equation (2.17). The spatial domain is set to $\Omega = [L, R] = [1.5, 15]$, because we define the left boundary condition for the water depth to be determined by measurements from sensor 1 at position $x = 1.5\,\text{m}$. The right boundary was determined by trial and error such that the numerical solution is as close as possible to the measurements. The boundary condition for the velocity is set to zero. Then our forward problem reads

$$\begin{aligned}
&\begin{pmatrix} h \\ u \end{pmatrix}_t + \begin{pmatrix} hu \\ \frac{1}{2}u^2 + gh \end{pmatrix}_x = \begin{pmatrix} 0 \\ -gb_x - \kappa u \end{pmatrix} \quad \text{on } Q \\
&h(\cdot, 0) = H_{\text{obs}}(\cdot, 0) - b \quad \text{on } \Omega \\
&u(\cdot, 0) = 0 \quad \text{on } \Omega \\
&h(L, \cdot) = H_{\text{obs}}(L, \cdot) - b(L) \quad \text{on } [0, T] \\
&u(R, \cdot) = 0 \quad \text{on } [0, T],
\end{aligned} \tag{3.25}$$

where $g = 9.81$ is the gravitational acceleration and $\kappa = 0.2$ the bottom friction coefficient. Note that the left boundary condition is time-dependent as it is determined by the sensor data from sensor 1, i.e. by $H_{\text{obs}}(L, \cdot)$. We only use the measurements from 10 seconds after the waves were generated, thus in the interval $[30\,\text{s}, 40\,\text{s}]$. This is because the reflections from the end of the channel are only noticeable after about 45 s and these are hard to model.

Another advantage of keeping the time interval small is that this keeps computation times shorter. However, this short time suffices for obtaining a decent reconstruction of the bathymetry. We define the time 30 s as $t = 0$ such that we work on the time interval $[0, T] = [0, 10]$. This implies that $H_{\text{obs}}(\cdot, 0)$ is the water height at rest, i.e. 0.3 m, such that in our case the initial condition for the water depth is $h(\cdot, 0) = 0.3 - b$. The initial velocity $u(\cdot, 0) = 0$ is a reasonable choice in this setup, because the water is at rest at time $t = 0$.

3.2.3 Optimisation problem

For the case where we have an observation given on the whole spatial domain, the objective functional can be defined exactly the same as in the previous section, where we considered SWE with periodic boundary conditions. The objective functional will be adapted later when we consider an observation that is limited some discrete points in space. Let the parameters $\gamma, \delta, \lambda_1, \lambda_2 > 0$ and define the objective functional as

$$\begin{aligned} J(b, H) :=& \frac{\gamma}{2} \int_Q (H - H_{\text{obs}})^2 \, d(x, t) + \frac{\delta}{2} \int_\Omega (H(\cdot, T) - H_{\text{obs}}(\cdot, T))^2 \, dx \\ &+ \frac{\lambda_1}{2} \int_\Omega b^2 \, dx + \frac{\lambda_2}{2} \int_\Omega b_x^2 \, dx. \end{aligned} \tag{3.26}$$

The problem of bathymetry reconstruction reads

$$\min_b J(b, H) \quad \text{subject to} \quad (3.25). \tag{3.27}$$

As before, we will derive the continuous adjoint equations and gradient and discretise them with Dedalus.

3.2.4 Derivation of adjoint equations and gradient

As in the previous section, we use the Lagrangian approach to determine the adjoint equations and the gradient. Compared to Section 3.1, the Lagrange function has additional terms due to the boundary conditions, because these cannot be included in the function space $H^1(\Omega)$ like the periodic boundary conditions. Our Lagrange multipliers are denoted

$p_1, \dots, p_4$. The Lagrange function for our optimisation problem reads

$$\begin{aligned}\mathcal{L}(h,u,b,p) = \frac{\gamma}{2}\int_Q (h+b-H_{\text{obs}})^2 \; d(x,t) \\ + \frac{\delta}{2}\int_\Omega (h(\cdot,T)+b-H_{\text{obs}}(\cdot,T))^2 \; dx \\ + \frac{\lambda_1}{2}\int_\Omega b^2 \; dx + \frac{\lambda_2}{2}\int_\Omega b_x^2 \; dx \\ - \int_Q (h_t+(hu)_x)p_1 \; d(x,t) \\ - \int_Q (u_t + uu_x + gh_x + gb_x + \kappa u)\, p_2 \; d(x,t) \\ - \int_0^T (h(L,\cdot) - H_{\text{obs}}(L,\cdot) + b(L))p_3 \; dt \\ - \int_\Omega (h(\cdot,0) - (H_{\text{obs}}(\cdot,0)-b))p_4 \; dx.\end{aligned} \tag{3.28}$$

The necessary optimality condition

$$\nabla_{(h,u)}\mathcal{L}(h,u,b,p) = 0 \tag{3.29}$$

yields the adjoint equations. We compute the partial derivatives of the Lagrange function and use integration by parts. We get

$$\begin{aligned}\mathcal{L}_h(h,u,b,p)\varphi &= \gamma\int_Q (h+b-H_{\text{obs}})\,\varphi \; d(x,t) \\ &\quad + \delta\int_\Omega (h(\cdot,T)+b-H_{\text{obs}}(\cdot,T))\,\varphi(\cdot,T) \; dx \\ &\quad - \int_Q \varphi_t p_1 \; d(x,t) - \int_Q (\varphi u)_x p_1 \; d(x,t) - \int_Q g\varphi_x p_2 \; d(x,t) \\ &\quad - \int_0^T \varphi(L,\cdot)p_3 \; dt - \int_\Omega \varphi(\cdot,0)p_4 \; dx \\ &= \gamma\int_Q (h+b-H_{\text{obs}})\,\varphi \; d(x,t) \\ &\quad + \delta\int_\Omega (h(\cdot,T)+b-H_{\text{obs}}(\cdot,T))\,\varphi(\cdot,T) \; dx \\ &\quad + \int_Q \varphi p_{1,t} \; d(x,t) - \int_\Omega [\varphi p_1]_0^T \; dx + \int_Q \varphi u p_{1,x} \; d(x,t) \\ &\quad - \int_0^T [\varphi u p_1]_L^R \; dt + \int_Q g\varphi p_{2,x} \; d(x,t) - \int_0^T [g\varphi p_2]_L^R \; dt \\ &\quad - \int_0^T \varphi(L,\cdot)p_3 \; dt - \int_\Omega \varphi(\cdot,0)p_4 \; dx\end{aligned} \tag{3.30}$$

which has to be zero for all $\varphi \in C_0^\infty(Q)$ due to the optimality condition (3.29). This leads to

$$\int_Q (\gamma\,(h+b-H_{\text{obs}}) + p_{1,t} + up_{1,x} + gp_{2,x})\,\varphi \; d(x,t) = 0.$$

Using the fundamental lemma of calculus of variations, we obtain the first part of the adjoint problem

$$p_{1,t} + up_{1,x} + gp_{2,x} = -\gamma\left(h + b - H_{\text{obs}}\right). \tag{3.31}$$

Now we let single conditions on the function φ vary in order to derive final and boundary conditions for the adjoint equation. By allowing any value for $\varphi(\cdot, T)$ in Equation (3.30) we obtain the final condition for the first adjoint variable

$$p_1(\cdot, T) = \delta\left(h(\cdot, T) + b - H_{\text{obs}}(\cdot, T)\right). \tag{3.32}$$

Letting the initial value $\varphi(\cdot, 0)$ vary we get

$$p_1(\cdot, 0) = p_4 \tag{3.33}$$

and leaving out the condition for $\varphi(R, \cdot)$ we get

$$gp_2(R, \cdot) = 0. \tag{3.34}$$

Leaving out $\varphi(L, \cdot) = 0$ yields an expression for the Lagrange multiplier p_3, which is

$$p_3 = u(L, \cdot)p_1(L, \cdot) + gp_2(L, \cdot). \tag{3.35}$$

The other partial derivative is

$$\begin{aligned}
\mathcal{L}_u(h, u, b, p)\varphi &= -\int_Q (h\varphi)_x p_1 \; d(x,t) - \int_Q \left(\varphi_t + (u\varphi)_x + \kappa\varphi\right) p_2 \; d(x,t) \\
&= \int_Q h\varphi p_{1,x} \; d(x,t) - \int_0^T \left[h\varphi p_1\right]_L^R \; dt + \int_Q \varphi p_{2,t} \; d(x,t) \\
&\quad - \int_\Omega \left[\varphi p_2\right]_0^T \; dx + \int_Q u\varphi p_{2,x} \; d(x,t) \\
&\quad - \int_0^T \left[u\varphi p_2\right]_L^R \; dt - \int_Q \kappa\varphi p_2 \; d(x,t).
\end{aligned} \tag{3.36}$$

Thus, the second part of the adjoint equation is

$$p_{2,t} + hp_{1,x} + up_{2,x} = \kappa p_2. \tag{3.37}$$

We let $\varphi(\cdot, T)$ vary and obtain the final condition for the second adjoint variable

$$p_2(\cdot, T) = 0. \tag{3.38}$$

If we omit the condition $\varphi(L, \cdot) = 0$, we get

$$\begin{aligned}
u(L, \cdot)p_2(L, \cdot) &= -p_1(L, \cdot)h(L, \cdot) \\
\Leftrightarrow p_1(L, \cdot) &= -\frac{u(L, \cdot)}{h(L, \cdot)} p_2(L, \cdot).
\end{aligned} \tag{3.39}$$

With Equation (3.35) this implies that

$$p_3 = \left(g - \frac{(u(L, \cdot))^2}{h(L, \cdot)}\right) p_2(L, \cdot). \tag{3.40}$$

In order to solve the adjoint problem numerically, we need to transform it into an initial value problem. This we can do by setting $\tau := T - t$ and defining the backward variables

$$\begin{aligned}\tilde{p}_{1/2}(x,\tau) &:= p_{1/2}(x, T-t)\\ \tilde{h}(x,\tau) &:= h(x, T-t)\\ \tilde{u}(x,\tau) &:= u(x, T-t)\\ \tilde{H}_{\text{obs}}(x,\tau) &:= H_{\text{obs}}(x, T-t).\end{aligned}$$

The time transformed adjoint problem is

$$\begin{aligned}&\tilde{p}_{1,\tau} - \tilde{u}\tilde{p}_{1,x} - g\tilde{p}_{2,x} = \gamma\left(\tilde{h} + b - \tilde{H}_{\text{obs}}\right)\\ &\tilde{p}_{2,\tau} - \tilde{h}\tilde{p}_{1,x} - \tilde{u}\tilde{p}_{2,x} = -\kappa\tilde{p}_2\\ &\tilde{p}_1(\cdot,0) = \delta\left(\tilde{h}(\cdot,0) + b - \tilde{H}_{\text{obs}}(\cdot,0)\right)\\ &\tilde{p}_2(\cdot,0) = 0\\ &\tilde{p}_1(L,\cdot) = -\frac{\tilde{u}(L,\cdot)}{\tilde{h}(L,\cdot)}\tilde{p}_2(L,\cdot)\\ &\tilde{p}_2(R,\cdot) = 0.\end{aligned} \tag{3.41}$$

Now we need to determine the L^2-gradient, because we need it to compute the H^1-gradient. The L^2-gradient can be obtained by computing the derivative $\mathcal{L}_b$ and using integration by parts as before. The derivative of the Lagrangian with respect to b is

$$\begin{aligned}\mathcal{L}_b(h,u,b,p)\delta b = \gamma &\int_\Omega \int_0^T (h + b - H_{\text{obs}})\ dt\ \delta b\ dx\\ &+ \delta \int_\Omega (h(\cdot,T) + b - H_{\text{obs}}(\cdot,T))\,\delta b\ dx\\ &+ \lambda_1 \int_\Omega b\delta b\ dx + \lambda_2 \int_\Omega b_x(\delta b)_x\ dx - \int_\Omega \int_0^T gp_2\ dt\ (\delta b)_x\ dx\\ &- \int_0^T \delta b(L)p_3\ dt - \int_\Omega p_4\delta b\ dx\end{aligned} \tag{3.42}$$

After applying integration by parts and demanding no change of the bathymetry at the boundaries, we obtain

$$\begin{aligned}\mathcal{L}_b(h,u,b,p)\delta b = &\int_\Omega \left(\int_0^T \gamma(h + b - H_{\text{obs}}) + gp_{2,x}\ dt\right)\delta b\ dx\\ &+ \int_\Omega \left(\delta\left(h(\cdot,T) + b - H_{\text{obs}}(\cdot,T)\right) + \lambda_1 b - \lambda_2 b_{xx} - p_4\right)\delta b\ dx.\end{aligned} \tag{3.43}$$

Hence, the L^2-gradient is

$$\begin{aligned}\tilde{v} := &\int_0^T \gamma(h + b - H_{\text{obs}}) + gp_{2,x}\ dt\\ &+ \delta\left(h(\cdot,T) + b - H_{\text{obs}}(\cdot,T)\right) + \lambda_1 b - \lambda_2 b_{xx} - p_4.\end{aligned} \tag{3.44}$$

With $v(L) = v(R) = 0$, the H^1-gradient is the solution v of the problem

$$\begin{aligned} v - \Delta v &= \tilde{v} \\ v(R) &= 0 \\ v(L) &= 0. \end{aligned} \tag{3.45}$$

3.2.5 Discretisation and comparison with measurement data

As mentioned beforehand, all PDEs will be solved with Dedalus. We use a `Chebyshev` basis with $M = 64$ grid points and a time step of 1×10^{-3} s. Before we solve the PDE-constrained optimisation problem, we have to ensure that the numerical solution is sufficiently close to the measurement data. From the $n = 20$ measurements with the Gaussian-shape bathymetry we take the average and compare it to the numerical solution of the SWE. The corresponding simulated water height and mean of the measurements with bathymetry are shown in Figure 3.18. As the data from sensor 1 is used for the left boundary condition, it is equal to the numerical solution and therefore not shown in the plot.

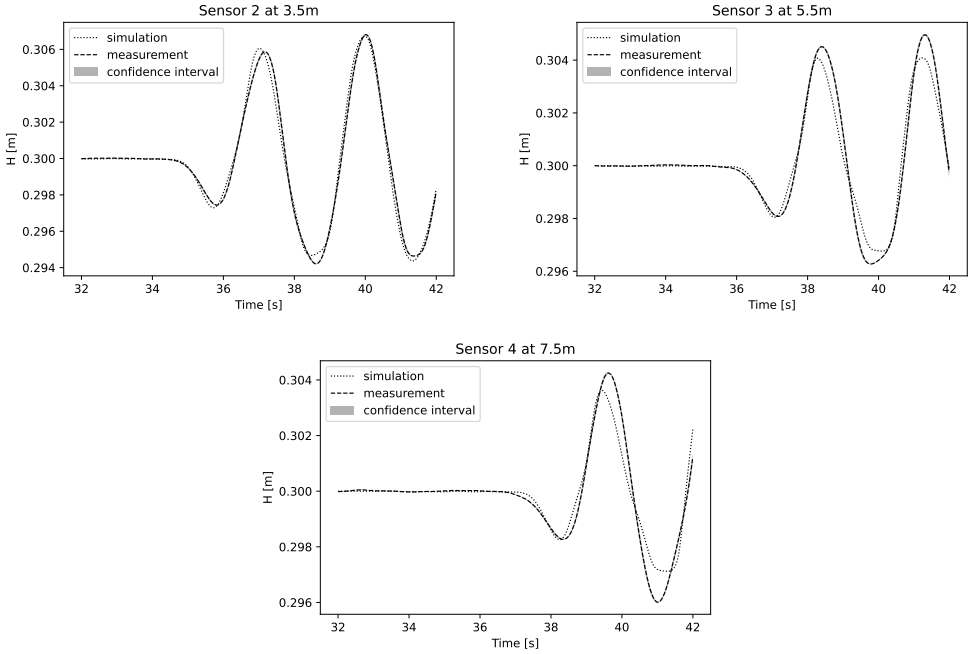

Figure 3.18: Average of measurements (dashed) with 95% confidence interval (grey) and numerical solution (dotted). [20]

A 95% confidence interval is plotted as well, but it is hardly distinguishable from the dashed line which shows the mean of all measurements. The confidence interval was determined as follows. We assume the observations to be normally distributed and stochastically independent. Let $\bar{x}$ be the mean value of the measurements and s_x the standard deviation. With certainty $1 - \alpha$, $\alpha \in (0, 1)$, the true value lies in the confidence interval [64], which is

given by [65]

$$\left[\bar{x} \pm \frac{s_x t_{n-1,1-\alpha/2}}{\sqrt{n}}\right]. \tag{3.46}$$

For a 95% confidence interval, we have $\alpha = 0.05$ and consequently have to use the t-value 1.729 [65].

The simulation is the closest to the measurement data at sensor 2 with a relative ℓ^2-error of 1.9×10^{-3} and slightly underestimates the wave amplitude at sensor 3 and sensor 4 with relative ℓ^2-errors of 1.6×10^{-3} and 1.3×10^{-3}, respectively. The good match of the measurement data with the simulation is promising for a reasonable reconstruction of the bathymetry from the experimental data.

3.2.6 Reconstructed bathymetries from different types of observations

We will move step by step towards the reconstruction from experimental data. First, we will use simulated observations. This is easier, as in this case we have no modelling error. We start with results for an observation that was computed numerically by solving the SWE for the bathymetry that we want to reconstruct. First, the computed observation is used on the whole spatial domain, also with added random noise. Then, we use a scenario that is closer to the experimental setup where the measurements are only given at the sensor positions. We perform reconstructions, where we only use the computed observation at a few points in the spatial domain. This is more challenging, because less information about the free surface level is available. Here we also add noise to the observation. After this, we show reconstructions from experimental data. There, we have an additional modelling error in the observation. At first, we use the gradient descent method for all optimisations. In all plots, the reconstructions will only be shown on an interval from 1.5 m to 10 m for improved visibility. Outside this domain there are only smaller oscillations in the reconstructions that are negligible for the plots. All errors were computed on the full interval $[1.5, 15]$ as this is the spatial domain we are working on.

3.2.6.1 Reconstruction from simulated observations

In this part, we examine the quality of the bathymetry reconstruction from an observation that was computed from the solution the forward problem for the exact bathymetry. The exact bathymetry here means an interpolation of the data that were measured at the actual bathymetry, see Figure 3.19.

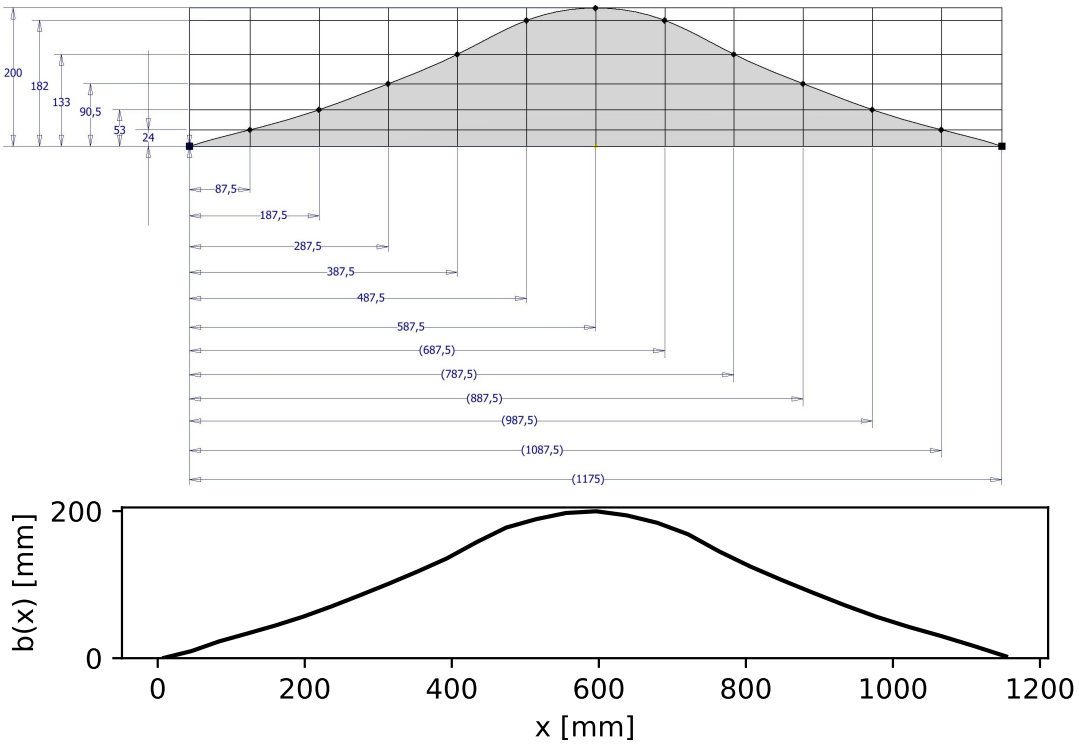

Figure 3.19: Measured points (top) and Dedalus representation (bottom) of the bathymetry that was used in the experiment. The Dedalus representation was obtained by manually copying the coordinates from the data points in the upper sketch and connecting them by using the SciPy interpolation function `PchipInterpolator` [20].

We computed the observation using $M = 128$ spatial grid points and chose a relatively fine time step of 5×10^{-5} s. The resolution needs to be different from the one used in the optimisation, otherwise this would mean to ignore the discretisation error, which is a so-called inverse crime [66].

We deal with the different spatial resolution by applying the `change_scales` function incorporated in Dedalus. The discretised functions, e.g. the water depth h can be saved in a Dedalus `Field` and these can be scaled onto a grid with a different number of grid points using `change_scales`. In time, we interpolate with the Scipy function `CubicSpline`. The regularisation parameters were chosen to be $\lambda_1 = 10^{-7}$ and $\lambda_2 = 10^{-6}$. A small change of these values does only lead to a small change in the reconstruction, but these values produce the best results in our case. The other coefficients in the objective functional we set to $\gamma = \delta = 0.5$. We will also add noise to the observation and show the corresponding reconstructed bathymetries. In the presence of noise in the observation we increase the second regularisation parameter to $\lambda_2 = 10^{-5}$, because the noise leads to additional small hills in the reconstruction. The noise was normally distributed with a standard deviation of 5% of the maximal local wave height, where we took the maximum over the time interval $[0, 10]$.

Observation on the whole spatial domain. In the following we used the computed observation on the complete spatial domain with and without added noise. Figure 3.20

shows the corresponding reconstructions along with the exact bathymetry. Except for some smaller oscillations, the reconstructed hill is very close to the exact bathymetry. The relative ℓ^2-error is 29.44% for the case without noise. In the case where noise was added to the observation, there are some smaller artefacts on the right hand side of the spatial domain, but we still obtain a good approximation on the hill at 4 m.

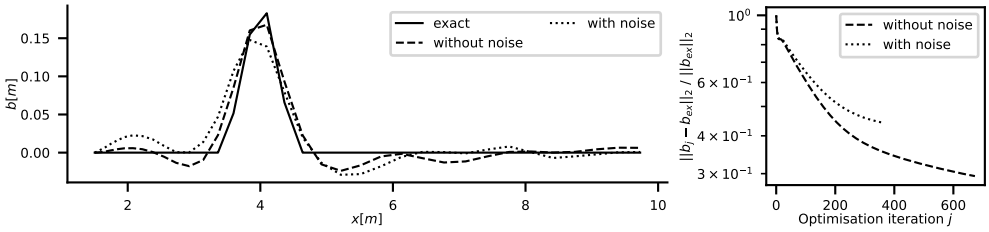

Figure 3.20: Reconstructed bathymetries from simulated data and relative ℓ^2-errors to exact bathymetry [20].

The corresponding reconstruction has a relative error of 44.25%. Here, the algorithm stopped before reaching the tolerance $\varepsilon = 10^{-7}$ for the norm of the gradient, because no step size was found for which the value of the objective functional was significantly smaller. This means, that the approximation on the anti-gradient was no longer a descent direction. A possible explanation is that at some point the mismatch to the observation is mainly noise. It might be hard to change the bathymetry in such a way that this random noise in the free surface level is produced. When using an observation on a coarser grid, the algorithm runs until the stopping criterion is fulfilled.

Observation at sensor positions. Here, we use the simulated observation at several equidistant points in space and then at the positions where the real sensors in the experiment are placed. Let x_i, $i = 1, \dots, m_p$, be the sensor positions and $H_{\text{obs},i}$ the corresponding observations. These are obtained by evaluating the interpolation of the observation at the sensor positions. As the observation is no longer given on the whole spatial domain, we have to adapt the objective functional. Formally, we would have Dirac deltas for each observation point. To approximate these, we choose Gaussians with peak at the sensor positions and maximum value being the observed values $H_{\text{obs},i}$. The variance of the Gaussians is set to $\sqrt{0.045}$ such that these are narrow, but still significantly larger than the distance of two adjacent grid points.

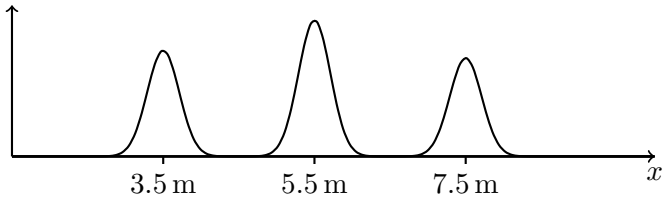

Figure 3.21: Sketch of the Gaussians in the objective functional.

Then the objective functional changes to

$$\begin{aligned} J(b,H) :=& \frac{\gamma}{2}\int_Q \left(\sum_{i=1}^{m_p}(H(x_i,t)-H_{\mathrm{obs},i}(t))\exp\left(-\frac{1}{2}\frac{(x-x_i)^2}{0.045}\right)\right)^2 d(x,t) \\ &+ \frac{\delta}{2}\int_\Omega \left(\sum_{i=1}^{m_p}(H(x_i,T)-H_{\mathrm{obs},i}(T))\exp\left(-\frac{1}{2}\frac{(x-x_i)^2}{0.045}\right)\right)^2 dx \\ &+ \frac{\lambda_1}{2}\int_\Omega b^2\,dx + \frac{\lambda_2}{2}\int_\Omega b_x^2\,dx. \end{aligned} \tag{3.47}$$

In this case the L^2-gradient reads

$$\begin{aligned} \tilde{v} :=& \int_0^T \gamma\left(\sum_{i=1}^{m_p}(H(x_i,t)-H_{\mathrm{obs},i}(t))\exp\left(-\frac{1}{2}\frac{(x-x_i)^2}{0.045}\right)\right) + gp_{2,x}\,dt \\ &+ \delta\left(\sum_{i=1}^{m_p}(H(x_i,T)-H_{\mathrm{obs},i}(T))\exp\left(-\frac{1}{2}\frac{(x-x_i)^2}{0.045}\right)\right) + \lambda_1 b - \lambda_2 b_{xx} - p_4. \end{aligned} \tag{3.48}$$

Figure 3.22 shows reconstructions with and without noise from a simulated observation at several data points which we also call "sensor positions" in the plots. The reconstructed bathymetry from noisy data shows some large artefacts next to the real bathymetry. But the reconstruction from the observation without noise is very close to the one from the full observation that we saw previously in Figure 3.20. It has a relative ℓ^2-error to the exact bathymetry of 31.33% which is a bit larger than the error of 29.44% for the full observation. The reconstruction from noisy data in Figure 3.22 has a relative error of 55.52%.

Now we increase the regularisation parameter to $\lambda_1 = 10^{-4}$. The corresponding reconstruction is shown in Figure 3.23. The error in the reconstruction is 71.59% which is higher than the error for the smaller regularisation parameter. Here the effect of the regularisation with the norm of the bathymetry b is visible. The artefacts between $x = 5\,\mathrm{m}$ and $x = 9\,\mathrm{m}$ are much smaller, but also the maximum of the hill at the position of the exact bathymetry.

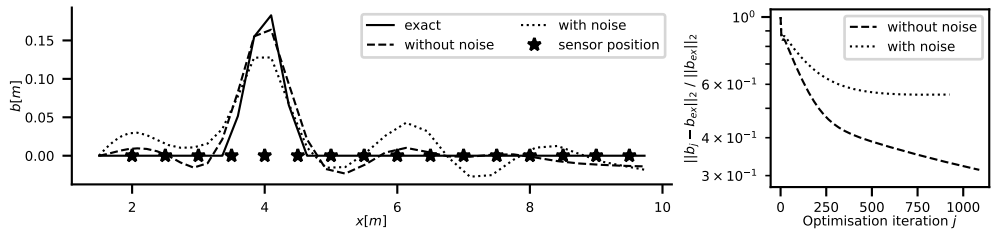

Figure 3.22: Reconstructed bathymetries from simulated data at sensor positions (left) and relative ℓ^2-errors to exact bathymetry (right).

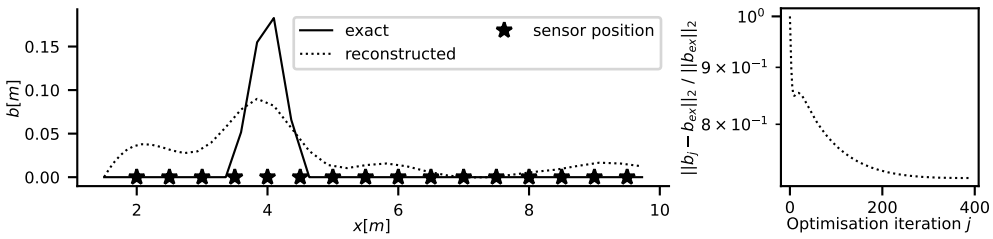

Figure 3.23: Reconstructed bathymetries from simulated data with noise at sensor positions (left) and relative ℓ^2-errors to exact bathymetry for $\lambda_1 = 10^{-4}$ (right).

Now we limit the number of used points for the observation to the positions of sensor 2, 3 and 4 and again perform a second reconstruction with noise added to the observation. Both results are shown in Figure 3.24. The errors of 57.84% without noise and 68.56% with noise are higher than when using more observation points, see Table 3.3. This is not surprising as there is less information available from which the bathymetry can be inferred. The maximum of the reconstructed hill is smaller than the maximal height of the true bathymetry, but still positioned correctly. In contrast to Figure 3.22 the effect of noise in the observation seems to be much less. There is hardly any difference in the ℓ^2-errors to the exact bathymetry in Figure 3.24.

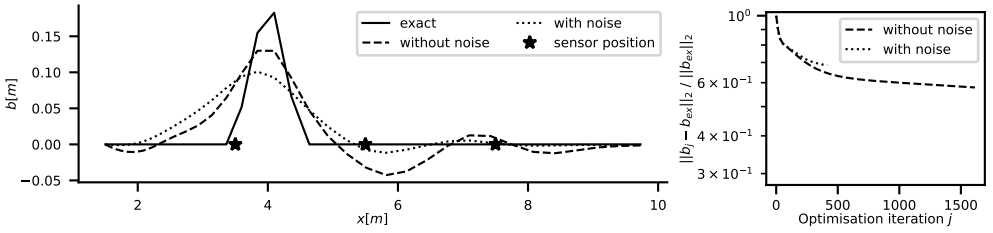

Figure 3.24: Reconstructed bathymetries from simulated data at sensor positions (left) and relative ℓ^2-errors to exact bathymetry (right) [20].

3.2.6.2 Reconstruction from measurements

Inferring the bathymetry from experimental data is more challenging than from simulated observations, as we have a model error additionally. Furthermore, the impact of the bathymetry on the water depth is in fact very small, see Figure 3.25, which shows the difference between the average of the 20 experiments with and the average of the 20 experiments without bathymetry. The 95% confidence interval is shown in grey. Here we used the t-value for 38 degrees of freedom which is 1.686 [67], because this is a two-sample problem with two different experiments, see Dümbgen [65]. The maximum difference in the water depths is around 1.5 mm at sensor 2 and 3 mm at sensor 3 and 4. For the reconstruction, we used the same parameters as in the case with a noisy observation. Figure 3.26 shows the reconstructed bathymetry which looks similar to the ones in Figure 3.24, but a with a larger error, especially at around 8 m, where the artefact has become larger. The larger reconstruction error is not surprising due to the modelling error. The maximum height on the other hand is slightly closer to the true bathymetry with 0.111 m compared to 0.105 m in the noisy case in Figure 3.24. Remarkable is that the error to the measurement data is the largest at sensor 4. We performed another reconstruction with the data from only

sensor 2 and 3, which is shown in Figure 3.27. The fact that this reconstruction has a smaller error to the exact bathymetry shows that the information from sensor 4 is not only useless for the reconstruction but even harming. As can be seen in Figure 3.27 on the left, the small hill in the reconstruction at around 8 m has vanished. Also, the relative ℓ^2-error is significantly smaller (83.76% versus 93.89%). It can be observed from Figure 3.18 that at sensor 4 the difference from the measurement to the simulation is larger than for the other sensors. This could be the reason for the artefact at around 8 m.

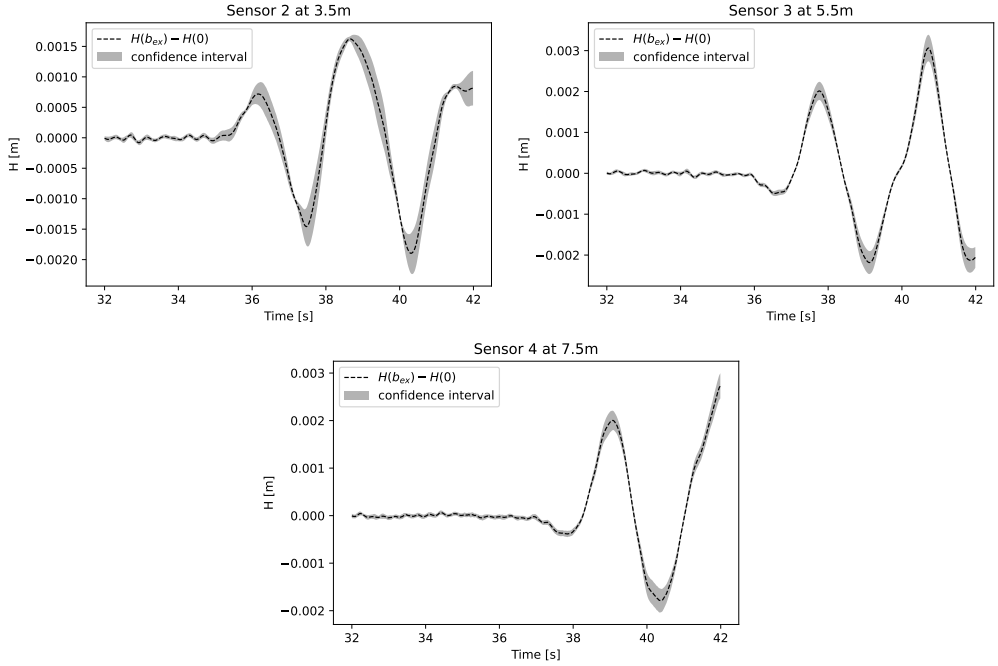

Figure 3.25: Difference between the averages of measurements with and without bathymetry (dashed) and 95% confidence interval (grey). [20]

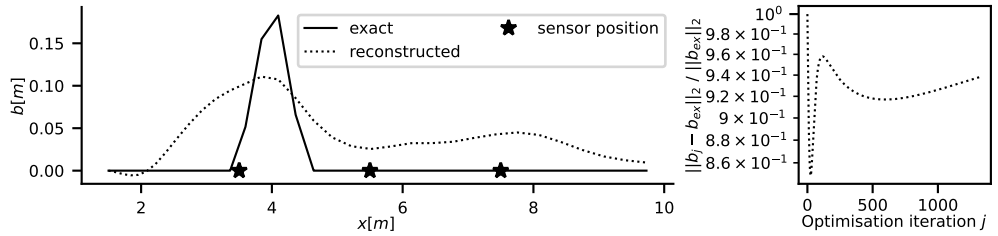

Figure 3.26: Reconstructed bathymetry from measurements (left) and relative ℓ^2-errors to the exact bathymetry (right) [20].

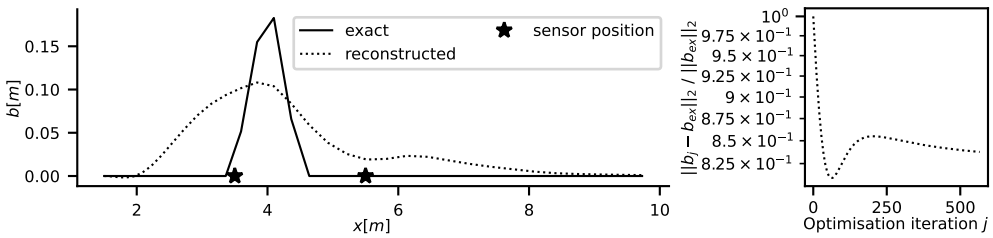

Figure 3.27: Reconstructed bathymetry from measurements at sensor 2 and 3 (left) and relative ℓ^2-errors to the exact bathymetry (right).

	Observation			
Type	Used sensor positions	ℓ^2-error	ℓ^∞-error	NRMSE
Simulated	16 equidistant pos.	31.33	17.19	05.48
	2,3,4	57.84	35.35	10.12
	2,3	59.08	34.91	10.34
	2	65.60	43.33	11.48
	3	89.92	67.83	15.73
Simulated + noise	16 equidistant pos.	69.91	31.12	12.23
		55.52	30.04	09.71
	2,3,4	68.56	49.23	11.99
	2,3	74.53	49.02	13.04
	2	72.21	48.71	12.63
	3	85.16	74.61	14.90
Measured	2,3,4	93.89	51.65	16.42
	2,3	83.76	51.20	14.65
	2	90.43	55.64	15.82
	3	97.82	59.47	17.11

Table 3.3: Relative error in the ℓ^2- and ℓ^∞-norm and normalised root mean squared error (NRMSE) between reconstructed bathymetry and the model representation of the exact bathymetry for reconstructions using data from different sensors [20].

In order to compare the results with other publications, we additionally consider the normalised root mean squared error (NRMSE), defined as [68]

$$\text{NRMSE} = \frac{\sqrt{\frac{1}{M}\sum_{i=1}^{M}(b_i - b_{\text{ex},i})^2}}{b_{\text{ex, max}} - b_{\text{ex, min}}}. \tag{3.49}$$

Table 3.3 shows relative ℓ^2- and ℓ^∞-norm errors and the NRMSE for simulated and measured observations on different sensor positions. With an NRMSE of around 14% the results are in line with those listed by Hao et al. [68, Table 2] which have NRMSEs between 20.6% and 31.8%. From the table, we also observe that the relative ℓ^∞-errors in the

case "simulated + noise" and "measured" are very similar for used sensors 2 and 3 and for used sensor 2. As already observed in Figures 3.26 and 3.27, for the measured observations using all three sensors gives a larger ℓ^2-error than using only sensors 2 and 3. Even for the simulated observation, sensor 4 seems to contribute very little to the reconstruction, as the error for all three sensors is almost the same as for only two sensors. For the case of 16 sensor positions with noise there are two rows for two different runs with exactly the same parameters. Because the noise consist of randomly added numbers, the quality of the reconstructions can vary a lot. This is also the reason for the different numbers for noisy observations compared to Angel et al. [20]. The best way would be to run the optimisation many times and compute the average error, but this is not done here as the runtimes are very long.

Reconstructions using L-BFGS. For the periodic boundary conditions in Section 3.1 L-BFGS could significantly improve runtimes while yielding reconstructions that have almost the same error as for the reconstructions by gradient descent. Now we will investigate if this holds also true for the bathymetry reconstruction from experimental data. We choose the following parameters. We save $m = 5$ of the previous descent directions in each iteration of L-BFGS and use a stopping tolerance of $\varepsilon = 10^{-7}$. All other parameters are the same as for gradient descent. The step sizes for the update in L-BFGS are being determined by a line search algorithm with strong Wolfe conditions. The reconstruction

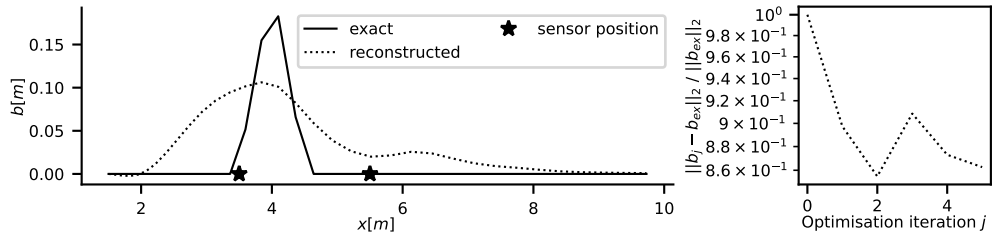

Figure 3.28: Reconstructed bathymetry using L-BFGS from measurements at sensor 2 and 3 (left) and relative ℓ^2-errors to the exact bathymetry (right).

with L-BFGS in Figure 3.28 is very similar to the one obtained with gradient descent in Figure 3.27, but with a slightly larger relative ℓ^2-error of 85.4% to the exact bathymetry. There is a large improvement in the runtime when using L-BFGS, which took about 81 minutes for 6 iterations compared to gradient descent with approximately 23.2 hours for 567 iterations. Thus, L-BFGS took only around 6% of the runtime of gradient descent. The L-BFGS algorithm stopped, because no more step size was found in the line search with Wolfe conditions.

Possibilities for further research. As we have seen, the position of the sensors have an effect on the quality of the reconstruction. This is an optimisation problem by itself that could be worked on. Furthermore, known benchmark problems [69, Chapter 3.1.3-3.1.5] could be used for a similar reconstruction, but this would require the opportunity to use a different wave flume that can produce waves which fit those problems. Also, using a different forward model like Euler equations would be a possibility for further investigation. We have seen that for the SWE the simulation error increases towards the end of the wave flume. This could be because the SWE do not capture the physics of the wave flume well

enough. From Figure 3.18 it can be observed that a wave needs roughly 1.25 s to travel from one sensor to the next. Thus, we get an approximate velocity of $1.6\,\frac{\mathrm{m}}{\mathrm{s}}$ for the waves. The time from one wave peak to the next is around 3 s such that we obtain a wavelength of about 4.8 m. According to Le Méhauté [51], shallow water waves have a ratio from water depth to wave length of at most 0.05, but with the water depth of 0.3 m that we have here, we have a ratio of 0.0625. Hence, the waves in the experiment are at the edge of being shallow. The Euler equations could give a better fitting model for the waves as these do not take the depth-average of the velocity.

Chapter 4

Parallel-in-time solution of SWE and corresponding adjoint equations

The reconstruction of bathymetries could be accelerated a lot if we found faster ways to solve the forward and adjoint equation, as this needs to be done in each iteration of the optimisation algorithm. In general, an option is to parallelise in space. But there is a saturation in performance due to Amdahl's law and this drives the need for additional parallelisation in time [70, 23, 71]. Recently, there has been a lot of research in parallel-in-time algorithms and in some cases these algorithms can yield good speed-up [70]. But hyperbolic problems such as SWE are challenging for parallel-in-time (PinT) methods. This was reported by Gander and Vandewalle [72] and Farhat and Chandesris [73]. Thus, new PinT methods need to be developed that are suitable for hyperbolic problems. In this chapter, we will look at such PinT methods and numerically investigate the convergence for the forward and adjoint SWE. For PDE-constrained optimisation problems, time parallelisation has a big potential in case of speed-up as for each solve of the forward and adjoint equations many time steps are being performed. Solving one PDE requires N_t time steps. For τ_{fwd} and τ_{adj} being the time needed for one time step for the forward and adjoint problem respectively, we have total costs of $j_{\text{opt}}(N_t\tau_{\text{fwd}} + N_t\tau_{\text{adj}})$, where j_{opt} is the number of iterations in the optimisation algorithm. A spatial parallelisation could reduce τ_{fwd} and τ_{adj} by some factor to $\tau_{\text{fwd}}s_{\text{fwd}}$ and $\tau_{\text{adj}}s_{\text{adj}}$ for $s_{\text{fwd}}, s_{\text{adj}} < 1$. Time parallelisation could additionally decrease the total costs $N_t\tau_{\text{fwd}}s_{\text{fwd}}$ and $N_t\tau_{\text{adj}}s_{\text{adj}}$ for the solution of the forward and adjoint problem by some other factors.

4.1 Parareal for SWE

Parareal is a PinT method that was invented by Lions, Maday and Turinici [21]. It is one of the simplest PinT methods. In the following, this algorithm will be explained. Then we will show numerical results for Parareal for the forward and adjoint SWE. As these are hyperbolic problems, there is no fast convergence to be expected for the original Parareal. Nevertheless, speed-up has been observed for Communication-aware Adaptive Parareal (CAAP) which was developed by Nielsen et al. [23]. They use adaptive time-steps and a specific combination of coarse and fine propagator that operate on the same grid. They use a WENO solver for the fine propagator and a Roe solver for the coarse propagator, which has a lower order of discretisation. However, the convergence of CAAP is strongly

affected by the choice of the coarse propagator and relies on the use of a special scheduler. Another disadvantage of this method is the load imbalance caused by the adaptive time steps. In Section 4.2 a different variant of Parareal will be introduced that is suitable for SWE and does not have these restrictions. The convergence of this variant will be compared numerically to the original Parareal.

4.1.1 The Parareal algorithm

The Parareal method is an iterative PinT algorithm for initial value problems that combines coarse and fine resolutions [21]. Consider an initial value problem

$$\begin{aligned} u_t &= f(u) \quad \text{on } [0,T] \\ u(0) &= u_0. \end{aligned} \tag{4.1}$$

Let the time domain $[0,T]$ be divided into time slices of length ΔT such that the number

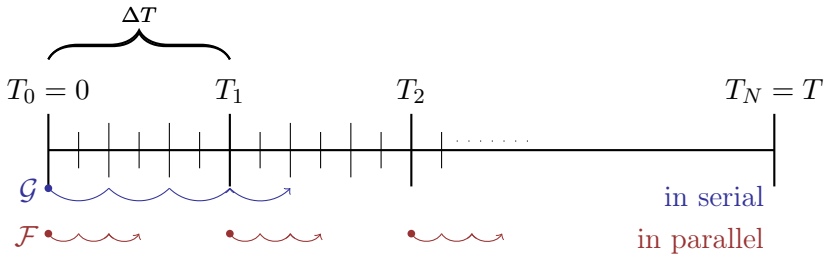

Figure 4.1: Coarse propagator (blue) and fine propagator (red) on the time domain with time slices of length ΔT.

of time slices N equals the number of processes for the parallel computation. We define a coarse propagator $\mathcal{G}$ which should be a cheap integrator with a coarse time step Δt, and a fine, more accurate propagator $\mathcal{F}$ with time step δt. By applying $\mathcal{G}$ on the underlying initial value Problem (4.1) we obtain an initial guess for the solution u, see Figure 4.1. In the first Parareal iteration $k = 0$, this yields initial values for all time slices such that the fine propagator can compute a fine solution in each time slice in parallel. In the k'th Parareal iteration the Parareal solution U_n^k at time step n can be determined as [74]

$$U_n^k = \mathcal{G}(U_{n-1}^k) + \mathcal{F}(U_{n-1}^{k-1}) - \mathcal{G}(U_{n-1}^{k-1}), \quad n = 1, ..., N. \tag{4.2}$$

The computation of $\mathcal{G}(U_{n-1}^k)$ is called the prediction step, whereas Equation (4.2) is the correction. The fine solution $\mathcal{F}(U_{n-1}^{k-1})$ can be computed in parallel by assigning one time slice to each process. For a stopping criterion the *defect*

$$\Delta^k := U^k - U^{k-1}, \tag{4.3}$$

i.e. the difference of two successive Parareal solutions can be used. One usually demands that the defect at all interfaces of the time slices is small enough, that is

$$\|U_n^k - U_n^{k-1}\|_\infty < \epsilon_P \quad \forall n = 1, \ldots, N \tag{4.4}$$

for some tolerance ϵ_P.

The detailed single steps of the method can be found in Algorithm 4. In Lines 2 to 4 the initial prediction for the Parareal solution is being computed using the coarse propagator $\mathcal{G}$. This has to be done in serial, because each process needs the initial guess on its time slice. There are two ways to do this. The first option is that every process l computes the initial prediction from T_0 until T_{l+1}. The other way is that process l computes the solution from T_l to T_{l+1} and all but the last process send the coarse solution at T_{l+1} on to process $l+1$. Note that the second approach requires some time for communication, but this can be negligible if the arrays to be sent are not too large. After computing the initial prediction, in Lines 7 and 8 the fine solution is computed in parallel, i.e. the `for`-loop should be parallelised. After that the prediction step is computed in Line 10 and the correction in Line 11. Here the `for`-loop starts only at `n=k` as the Parareal solution at time point k equals the fine solution [75]. The stopping criterion is found in Line 12. The algorithm always converges after at most $k_{\max} = N$ iterations to the sequential solution of the problem [76]. Depending

Algorithm 4 Simple Version of Parareal

1: $U_0^0 \leftarrow \tilde{U}_0^0 \leftarrow y_0$
2: **for** $n = 1$ to N **do**
3: $\quad \tilde{U}_n^0 \leftarrow \mathcal{G}(\tilde{U}_{n-1}^0)$
4: $\quad U_n^0 \leftarrow \tilde{U}_n^0$
5: **for** $k = 1$ to K **do**
6: $\quad U_0^k \leftarrow y_0$
7: $\quad$ **for** $n = 1$ to N **do**
8: $\quad\quad \hat{U}_n^{k-1} \leftarrow \mathcal{F}(U_{n-1}^{k-1})$
9: $\quad$ **for** $n = k$ to N **do**
10: $\quad\quad \tilde{U}_n^k \leftarrow \mathcal{G}(U_{n-1}^k)$
11: $\quad\quad U_n^k \leftarrow \tilde{U}_n^k + \hat{U}_n^{k-1} - \tilde{U}_n^{k-1}$
12: $\quad$ **if** $\|U_n^k - U_n^{k-1}\|_\infty < \epsilon_P \ \forall n$ **then**
13: $\quad\quad$ **break**

on the computation time needed by the coarse propagator the pipelined Parareal [77] can lead to improved speed up. It is implemented such that as soon as a process has finished computing the coarse solution on its time slice, it can already start computing the fine solution.

4.1.2 Parareal for forward and adjoint SWE

When implementing Parareal for the forward and adjoint equation, we can use the fact that the adjoint is backwards in time. Besides, we can implement it in a general way such that the same code can be applied for the forward and backward problem.

Note that within the optimisation there is the possibility of a so-called *warm start* for Parareal. Instead of computing an initial prediction with the coarse propagator each time, we can use the Parareal solution from the previous optimisation iteration. Especially for the adjoint this can be useful towards the end of the optimisation, where the adjoint solution does not change a lot from one optimisation step to the next.

4.1.2.1 Different order of ranks for the adjoint equation

Using Parareal for the forward and adjoint equation within the optimisation algorithm could be realised as follows. In each iteration we solve the forward problem first and then

use the result for the computation of the adjoint problem. The latter is backwards in time with the forward solution as coefficients, so we should also use the processes in backward order. This is illustrated in Figure 4.2. If we solve the primal problem with Parareal, then

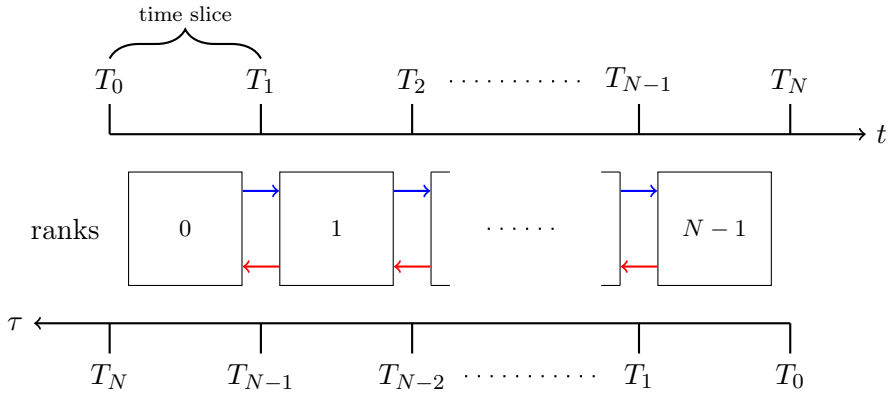

Figure 4.2: Communication in Parareal for forward and adjoint problem and ranks assigned to the corresponding time slices. The blue arrows indicate MPI messages sent during the computation of the forward problem and the red arrows for the adjoint.

process 0 will know the forward solution on the first time slice, process 1 on the second time slice and so on. Thus, the last process needs to compute the adjoint solution on the first time slice in 'backward' time τ which corresponds to the last time slice in t. That means, that for the adjoint problem we use the ranks in reverse order. Algorithm 5 shows how Parareal can be applied on both the forward and the adjoint problem. For this thesis, the parallelisation is realised using MPI, using the Python package `mpi4py`. For the forward problem, the first process is `rank` 0, so `first` is `True` for `rank` 0. The last one is `rank` $n-1$, so the variable `last` is `True` for this rank. This is exactly the other way round for the adjoint problem. We use the stopping criterion in Equation (4.4). If it is satisfied for process n, the flag `converged` of process n will be set to `True` in Line 5. Also, the flag `prev_converged` of the next process will then be set to `True` in Line 5. Note that, as for the adjoint problem the order of the processes is backwards, here the next process would be $n-1$. This is indicated by `nextPro`. After the algorithm converges, each process will know the Parareal solution on its time slice.

4.1.2.2 Convergence for SWE

Consider the SWE (2.17) with periodic boundary conditions. We will apply Parareal on the forward and adjoint problem and numerically investigate its convergence. As these are hyperbolic problems, we expect bad convergence behaviour.

Example 4.1. *Let b_{ex} be the bathymetry from the wave flume experiment in Section 3.2. We set the parameters $T = 3$, $[L, R] = [0, 10]$, $g = 9.81$, $D = 0.3$. For an arbitrary bathymetry b we define the initial conditions*

$$\begin{aligned} h_0(x) &= 0.05De^{-0.05(x-5)^2}\sin(0.2\pi(x-5)) + 0.95D - b(x) \\ u_0(x) &= 0 \end{aligned} \tag{4.5}$$

for $x \in [L, R]$.

Algorithm 5 Pipelined Parareal for forward and adjoint problem

Require: y_0
$U_0^0 \leftarrow \tilde{U}_0^0 \leftarrow y_0$
$converged \leftarrow$ False
if first is True **then**
    $prev\_converged =$ True
else
    $prev\_converged =$ False
$U_0^0 \leftarrow y_0$
$\tilde{U}_0^0 \leftarrow y_0$
if first is True **then**
    $\tilde{U}_1^0 \leftarrow G(\tilde{U}_0^0)$
    $U_1^0 \leftarrow \tilde{U}_1^0$
    comm.Send($\tilde{U}_1^0$, dest=$nextPro$)
else
    comm.Recv($\tilde{U}_n^0$, source=$prevPro$)
    $\tilde{U}_{n+1}^0 \leftarrow G(\tilde{U}_n^0)$
    $U_{n+1}^0 \leftarrow \tilde{U}_{n+1}^0$
if last is False **then**
    comm.Send($\tilde{U}_{n+1}^0$, dest=$nextPro$)
for $k = 0, \ldots, K_{\max}$ **do**
    $U_0^{k+1} \leftarrow y_0$
    $\hat{U}_{n+1}^k \leftarrow F(U_n^k)$
    if first is False **then**
        if prev_converged is False **then**
            comm.Recv(U_n^{k+1}, source=$prevPro$)
            comm.recv(prev_converged, source=$prevPro$)
        else
            $U_n^{k+1} \leftarrow U_n^k$
    $\tilde{U}_{n+1}^{k+1} \leftarrow G(U_n^{k+1})$
    $U_{n+1}^{k+1} \leftarrow \tilde{U}_{n+1}^{k+1} + \hat{U}_{n+1}^k - \tilde{U}_{n+1}^k$
    if last is False **then**
        comm.Send(U_{n+1}^{k+1}, dest=$nextPro$)
    if $||U_{n+1}^{k+1} - U_{n+1}^k|| < tol$ and prev_converged is True **then**
        $converged \leftarrow True$
    if last is False **then**
        comm.send($converged$, dest=$nextPro$)
    if converged is True **then**
        break

Consider Example 4.1. First of all, we compute an observation. That is, we compute a very fine solution for $b = b_{ex}$ using a time step of $5 \cdot 10^{-5}$, 130 spatial grid points and the third order Runge-Kutta method `RK443` that is included in the Dedalus package. The computed observation is needed for the initial condition and the source term in the adjoint problem (3.13). Then we compute the solution of the forward and adjoint problem with Parareal using 8 time slices. For the coarse propagator we choose the time step $\Delta t = 0.05$ and for the fine propagator $\delta t = 0.001$. Both use the Runge-Kutta method `RK443` and 64

spatial grid points. The stopping tolerance is set to $\epsilon_P = 10^{-6}$. We compute the forward solution for $b = 0$ and use this and the observation as input for the adjoint problem. This is what has to be done in the first iteration of the optimisation algorithm if we started with zero initial guess for the bathymetry. The Parareal defect and error to the sequential solution are shown in Figure 4.3. For both the forward and adjoint equation Parareal converges in a way that is typical for hyperbolic problems, see e.g. Gander [22]. For both

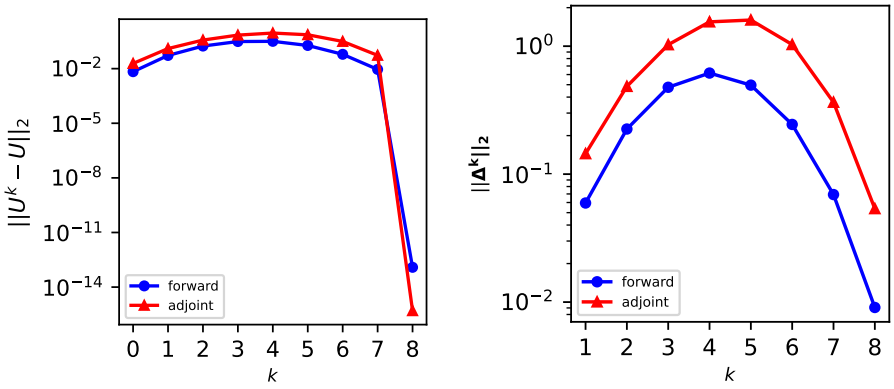

Figure 4.3: Left: Error Parareal solution U^k to sequential solution U for forward and backward problem. Right: Norm of Parareal defect Δ^k for forward and backward problem.

the forward and adjoint problem the defect and error to sequential solution increase before going down eventually after about 4 iterations. This effect has already been observed when applying Parareal to hyperbolic problems [78]. The maximum number of iterations was needed for Parareal to converge.

For the wave flume problem the convergence is a bit better, see Figure 4.4. For the forward problem one could possibly obtain speed-up when setting the tolerance higher and still get a sufficiently accurate solution. But the adjoint problem already starts with large errors which do not decrease a lot such that lifting the tolerance would not help. The Parareal defect stays above 10^{-2} for all iterations and the error to the sequential solution is far away from 10^{-3} for all but the last iteration. When changing the stopping tolerance to 5×10^{-4}, Parareal for the forward problem stops after only one iteration, but the adjoint still takes the maximum number of iterations. Thus, there is a chance that Parareal or some variant of it could work well for the forward problem, but not for the adjoint. In the following, we will only consider the forward SWE and propose to use a different PinT method for the adjoint, which will be explained later.

4.1.3 Coarsening in space

In the previous subsection, the difference of the coarse and fine propagator was only the time discretisation. Another approach in Parareal is to use a coarser discretisation in space for the coarse propagator than for the fine. We have seen that Parareal often converges very slowly for SWE. Still, it is possible to 'cheat' and take a rather small time step for the coarse propagator [79]. That will lead to early convergence of Parareal. Nevertheless, we will see in the following that for the SWE even a very small change in the spatial discretisation in the coarse propagator leads to very bad convergence. Similar results can

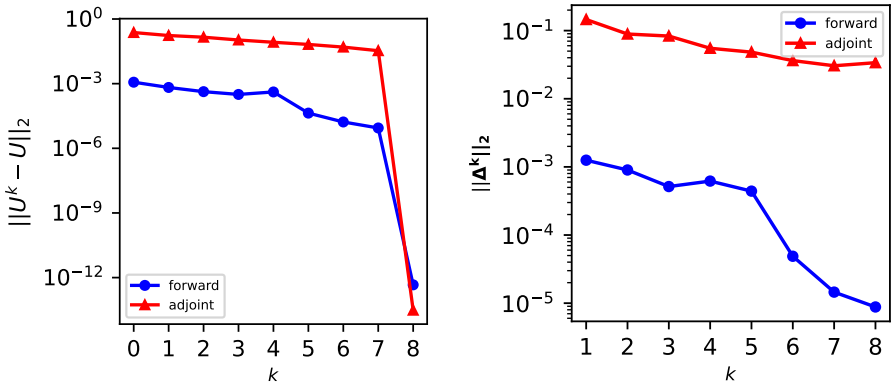

Figure 4.4: Left: Error Parareal solution U^k to sequential solution U for forward and backward problem. Right: Norm of Parareal defect Δ^k for forward and backward problem.

be found in Angel et al. [80].

When using spatial coarsening, solutions on the fine and on the coarse mesh have to be put on the same grid as they need to be added in the Parareal correction (4.2). That is, for $m_c < m_f$ we have to define a restriction operator

$$\mathcal{R} : \mathbb{C}^{m_f} \mapsto \mathbb{C}^{m_c}$$

and a prolongation operator

$$\mathcal{P} : \mathbb{C}^{m_c} \mapsto \mathbb{C}^{m_f}.$$

These can be defined by interpolation methods. Note that the order of interpolation has an influence on Parareal convergence [81]. Thus, we will plot Parareal convergence for different interpolation orders. A common approach is to compute the Parareal correction on the fine level as follows. Let $\tilde{\mathcal{G}}$ be the coarse integrator. Then, we can define the coarse propagator to operate as

$$\mathcal{G}(y) = \mathcal{P}\tilde{\mathcal{G}}(\mathcal{R}y) \tag{4.6}$$

and use this in the Parareal correction (4.2) as suggested by Fischer et al. [82], that is

$$U_n^k = \mathcal{P}\tilde{\mathcal{G}}(\mathcal{R}U_{n-1}^k) + \mathcal{F}(U_{n-1}^{k-1}) - \mathcal{P}\tilde{\mathcal{G}}(\mathcal{R}U_{n-1}^{k-1}). \tag{4.7}$$

Alternatively, one can use

$$\tilde{U}_n^k = \tilde{\mathcal{G}}(\mathcal{R}U_{n-1}^k) + \mathcal{R}\mathcal{F}(U_{n-1}^{k-1}) - \tilde{\mathcal{G}}(\mathcal{R}U_{n-1}^{k-1}) \tag{4.8a}$$

$$U_n^k = \mathcal{P}\tilde{U}_n^k + \mathcal{F}(U_{n-1}^{k-1}) - \mathcal{P}\mathcal{R}\mathcal{F}(U_{n-1}^{k-1}), \tag{4.8b}$$

which is the so-called *micro-macro Parareal* [83]. The Parareal correction acts on the coarse (macroscopic) level in Equation (4.8a) and this is lifted to the fine level in Equation (4.8b). The aim of the term $\mathcal{F}(U_{n-1}^{k-1}) - \mathcal{P}\mathcal{R}\mathcal{F}(U_{n-1}^{k-1})$ is to add information on the fine scale which is lost due to the restriction on the macroscopic level [83]. Here, the use of either the predictor-corrector format or micro-macro Parareal did not make any noticeable difference,

in contrast to Philippi and Slawig [83], where micro-macro Parareal was used for an ocean-circulation model. There was no explanation found, why only micro-macro Parareal works for the ocean circulation model and the correction (4.7) caused instabilities [84]. In his Dissertation, Philippi reported that the two versions of the Parareal correction yielded the same convergence results for the Burger's equation [84].

4.1.3.1 Parareal with spatial coarsening for SWE

We consider periodic boundary conditions (2.17) and the initial condition and the bathymetry b_{ex} from Example 4.1. For the fine and coarse propagator we use Dedalus with the `RealFourier` basis. We use the Scipy function `make_interp_spline` for the restriction and prolongation operators. When using spatial coarsening, we choose the order of interpolation out of $\{1, 3, 5\}$. For the fine and coarse propagator we take the same time stepping method `RK443` and a time step of $\Delta t = \delta t = 1 \times 10^{-3}\,\text{s}$ for both. The only difference in the propagators is the number of spatial grid points with $M_c = 62$ versus $M_f = 64$. We set the tolerance for the stopping criterion to $\epsilon_P = 10^{-6}$. For comparison we first show convergence plots for Parareal without spatial coarsening but with different time steps for the coarse and fine propagator $\Delta t = 1 \times 10^{-2}\,\text{s}$ and $\delta t = 1 \times 10^{-3}\,\text{s}$. Note that the time step Δt is five times smaller than the time step that we chose in Subsection 4.1.2, where Parareal only converged after the maximum number of iterations. The errors to the sequential solution are shown in Figure 4.5 for 8 and 16 cores. We only show results for polynomial interpolation of degree 3 for the prolongation and restriction operators. Changing the degree does not make any noticeable difference in this case. As mentioned beforehand, the convergence behaviour is very good for a hyperbolic problem, which is because the coarse propagator is too accurate [79]. But even in this case a very small reduction in the spatial resolution leads to a convergence behaviour that leaves no hope for any speed-up, as can be seen in Figure 4.6. Compared to only coarsening in time in Figure 4.5 there is a big difference in Parareal convergence with only one iteration versus the maximum number of iterations (8/16).

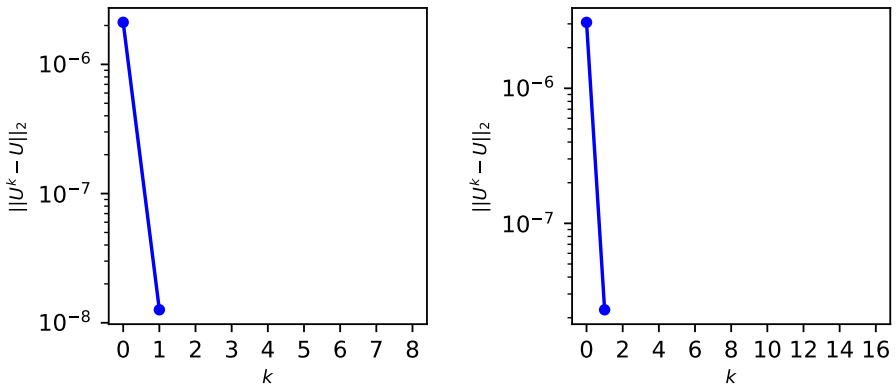

Figure 4.5: Error to the sequential solution for SWE for 8 time slices (left) and 16 time slices (right). No spatial coarsening was used. The difference in the coarse propagator is the larger time step of $\Delta t = 10^{-2}$. Here the polynomial order of interpolation did not make any difference, therefore only results for polynomial degree 3 are shown.

The order of interpolation does only lead to slight changes in the error to the sequential solution and the defect. The biggest change is from order 1 to order 3, where for the first few iterations the error is smaller for degree 3. But then the errors are - at least partly - larger than for first order interpolation and then again smaller. The reason for this behaviour should be further investigated. For all cases the convergence is too slow such that we cannot get any speed-up. The following sections show parallel-in-time methods that could be better suited for the forward and adjoint SWE.

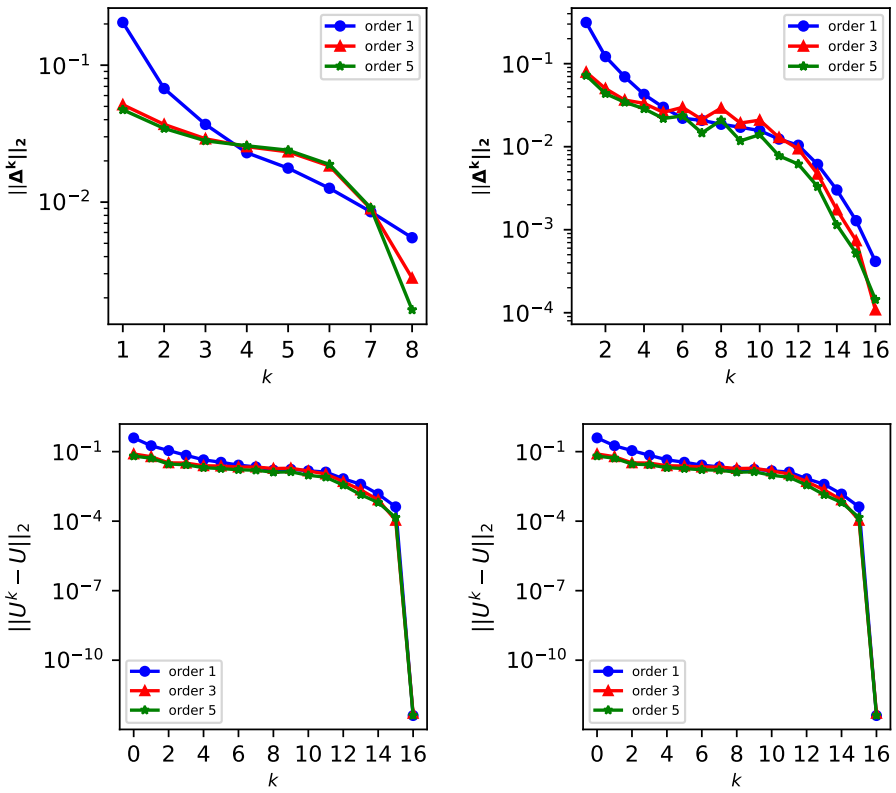

Figure 4.6: Defect (top) and error to the sequential solution (bottom) for SWE for 8 time slices (left) and 16 time slices (right). Spatial coarsening was done by reducing the number of spatial grid points from 64 to 62. In each plot, the blue circles show the defect for linear interpolation, the red triangles for cubic interpolation and the green stars for interpolation degree 5.

4.2 Parareal with averaging for shallow water equations

Parareal with phase averaging, also called *Asymptotic Parareal*, is a variant of Parareal that was developed by Haut and Wingate [25]. The idea is to mitigate small oscillations in the coarse solution using so-called *phase-averaging* such that it is possible to use a larger time step for the coarse propagator [27]. The slow structure in the problem is still preserved, as the smaller features are averaged [85]. Asymptotic Parareal was already shown to be useful for SWE without bathymetry in the sense that is allows for larger time steps in the coarse propagator [27]. There is also work on SWE on the sphere [26], where a framework is presented including a stable matrix exponential specific to Finite Elements and a parallel

phase-averaging procedure. Asymptotic Parareal can be applied on any equation that has the form

$$\mathbf{u_t} + \frac{1}{\varepsilon}\mathfrak{L}\mathbf{u} = \mathcal{N}(\mathbf{u}), \quad \mathbf{u}(0) = \mathbf{u}_0, \tag{4.9}$$

where $\mathfrak{L}$ is a linear operator that is usually assumed to be skew-hermitian [25, 27] and responsible for temporal oscillations in the solution [85]. The operator $\mathcal{N}$ is the nonlinearity of the equation and $\varepsilon > 0$. Decreasing the parameter ε causes smaller oscillations in the system, leading to increased oscillatory stiffness [86]. We can eliminate the linear term by applying the transformation [85]

$$\mathbf{w}(t) = e^{\frac{1}{\varepsilon}\mathfrak{L}t}\mathbf{u}(t). \tag{4.10}$$

This yields the *modulation equation*

$$\dot{\mathbf{w}}(\mathbf{t}) = \mathbf{e}^{\frac{\mathbf{1}}{\varepsilon}\mathfrak{L}\mathbf{t}}\mathcal{N}(\mathbf{e}^{-\frac{\mathbf{1}}{\varepsilon}\mathfrak{L}\mathbf{t}}\mathbf{w}) \tag{4.11}$$

that is being solved by the fine propagator. In order to allow for larger time steps for the coarse propagator, it solves an averaged version of this equation. This is the so-called *averaged equation* [25]

$$\frac{d\bar{\mathbf{w}}}{dt} = \frac{1}{\eta}\int_{-\frac{\eta}{2}}^{\frac{\eta}{2}} \rho\left(\frac{s}{\eta}\right) e^{\frac{1}{\varepsilon}\mathfrak{L}(s+t)}\mathcal{N}(e^{-\frac{1}{\varepsilon}\mathfrak{L}(s+t)}\mathbf{w})\, ds. \tag{4.12}$$

The smooth kernel function ρ has to be chosen appropriately such that the length of the averaging window $\eta > 0$ is as small as possible [25]. This reduces the computational costs [86] which should then be asymptotically smaller than $1/\varepsilon$ [25].

4.2.1 Reformulation of the SWE

The one-dimensional nonlinear SWE which we use in our bathymetry reconstruction reads

$$\begin{aligned} h_t + (hu)_x &= 0 \\ u_t + uu_x + gh_x &= -gb_x - \kappa u. \end{aligned} \tag{4.13}$$

In order to apply Asymptotic Parareal, we need to write it in the form of Equation (4.9). To this end, we define a *perturbation height* Φ which is the deviation from a constant D which could be the average water height for example. Precisely, we set $D + \Phi - b = h$. This is illustrated in Figure 4.7. With the perturbation height Φ we can rewrite Equation (4.13)

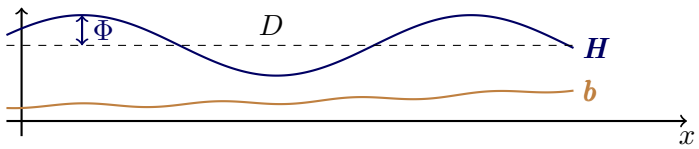

Figure 4.7: Free surface elevation H, average surface level D, perturbation Φ and bathymetry b.

as

$$\begin{aligned} \Phi_t + Du_x &= -(\Phi u)_x + (bu)_x \\ u_t + g\Phi_x &= -uu_x - \kappa u. \end{aligned} \tag{4.14}$$

Thus, with the notation in Equation (4.9) we have the operators

$$\mathfrak{L} = \begin{pmatrix} 0 & D\partial_x \\ g\partial_x & 0 \end{pmatrix}, \qquad \mathcal{N} = \begin{pmatrix} -(\Phi u)_x + (bu)_x \\ -uu_x - \kappa u \end{pmatrix}. \tag{4.15}$$

Here, we have $\varepsilon = 1$, where for rotating SWE Asymptotic Parareal has been proven to be very accurate [25].

4.2.2 Averaging for the SWE

In Peddle et al. [27, Figure 5.1] the transformed fine solution of a rotational SWE without bathymetry is compared to the transformed solution of the averaged equation. That is, the solution of the fine and coarse propagator are being illustrated. In their paper, the two fine and coarse solution are hardly distinguishable. We will show corresponding plots for the non-rotational SWE with non-zero bathymetry for two different bathymetries and initial conditions which we will see again in Chapter 5 for PinT bathymetry reconstructions.

Example 4.2. *For $[L, R] = [0, 10]$, $T = 10$, $g = 9.81$, $\kappa = 0.2$ and $D = 0.3$ let b be the hill from the experiment in Section 3.2 given by the measurement points in Figure 3.19 and define the initial conditions*

$$\begin{aligned} \Phi(x, 0) &= 0.015 \exp(-0.05(x-5)^2) \sin(0.2\pi(x-5)) - 0.015 \\ u(x, 0) &= 0 \end{aligned}$$

for $x \in [L, R]$.

We compute the solution of Equation (4.14) for the case of Example 4.2 with the fine and coarse propagator. The fine propagator yields a solution $\mathbf{w} = (w_1, w_2)$ of the modulation equation (4.11) and the coarse propagator a solution $\bar{\mathbf{w}} = (\bar{w}_1, \bar{w}_2)$ of the averaged equation (4.12). By applying the exponential of the linear operator $e^{-\mathfrak{L}t}$ we obtain the corresponding solutions of the SWE (4.14) that we call Φ and $\bar{\Phi}$, respectively. Figure 4.8 shows the coarse solution (top) computed with averaging window $\eta = 0.4$, its transformation (middle) and the transformation of the fine solution (bottom). There are small differences between Φ and $\bar{\Phi}$ visible, the relative ℓ^2-error is around 1%. But we could use a very large time step for the coarse propagator, that is $\Delta t = 0.1$ which is 100 times larger than the fine time step $\delta t = 0.001$. Using the coarse time step for the fine propagator produces a solution that is rather useless due to instabilities, see Figure 4.9. Thus, only averaging allows us to use a coarser time step in this example.

We also consider a sinusoidal bathymetry with a Gaussian initial condition which is the second example that we will use for a bathymetry reconstruction later.

Example 4.3. *For $[L, R] = [0, 10]$, $T = 10$, $g = 9.81$, $\kappa = 0.2$ and $D = 0.3$ let*

$$\begin{aligned} b(x) &= 0.05 \sin(0.2\pi x) + 0.1 \\ \Phi(x, 0) &= 0.05 \exp(-0.2(x-5)^2) + 0.05 \\ u(x, 0) &= 0 \end{aligned}$$

for $x \in [L, R]$.

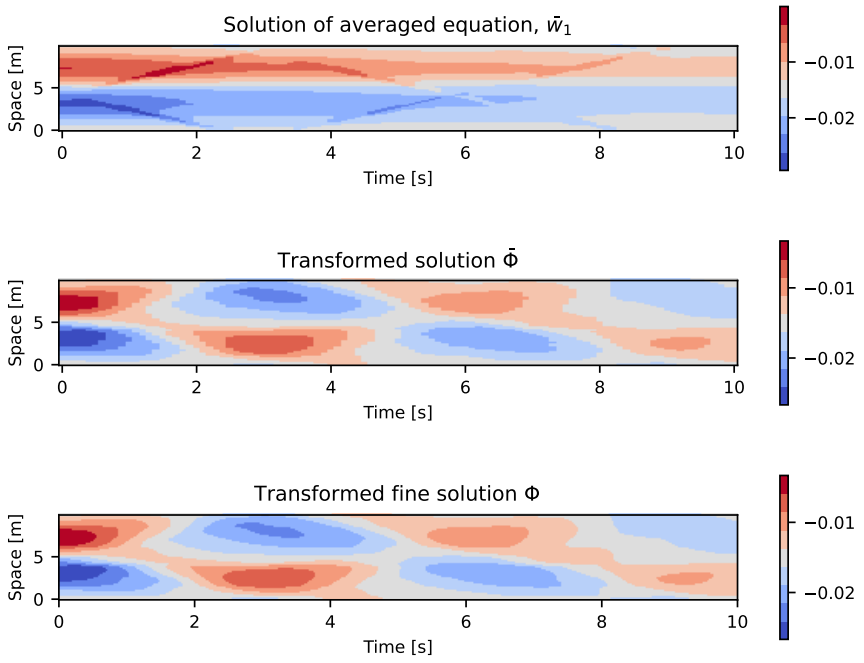

Figure 4.8: First solution component of the averaged equation $\bar{w}_1$ (top), its transformation $\bar{\Phi}$ (middle) and transformed solution Φ of the fine propagator (bottom) for Example 4.2. For the solution of the averaged equation ($\eta = 0.4$) the time step was 100 times larger than for the fine solution.

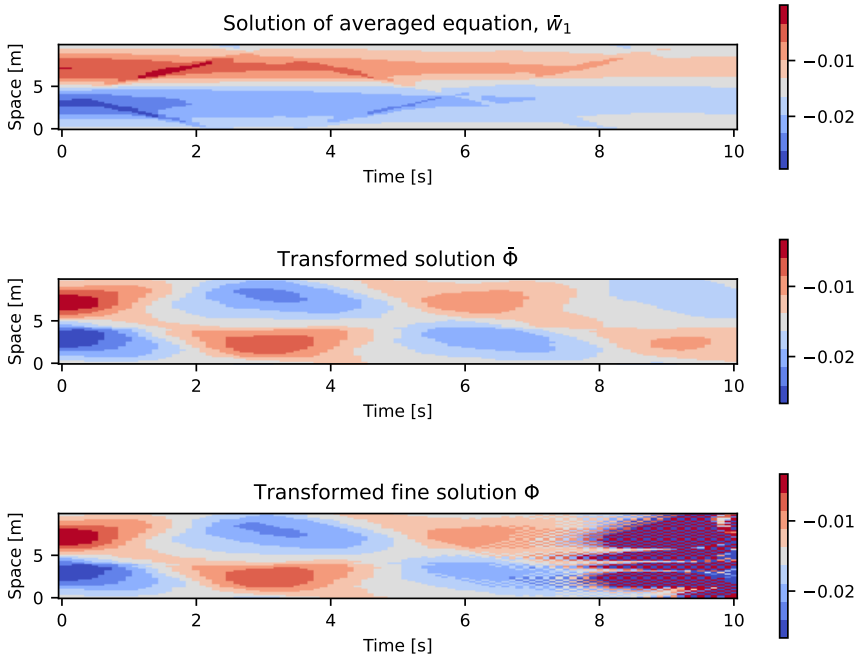

Figure 4.9: First solution component of the averaged equation $\bar{w}_1$ (top), its transformation $\bar{\Phi}$ (middle) and transformed solution Φ of the fine propagator (bottom) for Example 4.2. The fine solution was computed using the coarse time step of $\delta t = \Delta t = 0.1$. Instabilities in Φ are clearly visible after $t = 6$.

Figure 4.10 shows that for Example 4.3 the solution of coarse propagator $\bar{\Phi}$ is quite close to the fine solution Φ, whereas the fine solution requires a smaller time step of at least $\delta t = 0.09$. Taking a coarse time step of 0.1 for the fine propagator leads to an unstable solution, even worse than for Example 4.2, see Figure 4.9. This is why no corresponding plot is shown for Example 4.3.

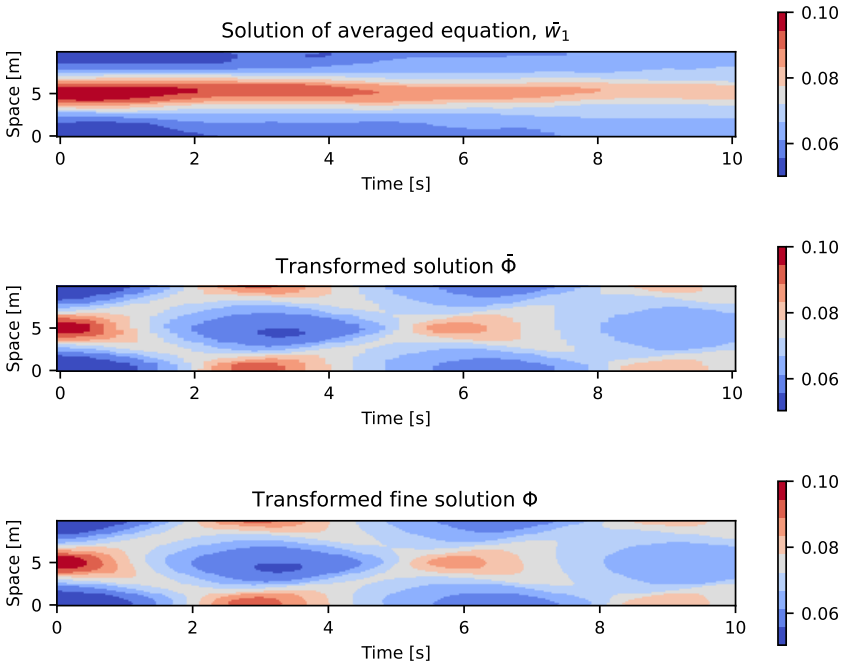

Figure 4.10: First solution component of the averaged equation $\bar{w}_1$ (top), its transformation $\bar{\Phi}$ (middle) and transformed solution Φ of the fine propagator (bottom) for Example 4.3. For the solution of the averaged equation ($\eta = 0.4$) the time step was 100 times larger than for the fine solution.

4.2.3 Time-dependent boundary conditions

In Section 3.2 a hill is reconstructed from measurement data. As there is noise in the sensor data from which the boundary condition is determined, we have temporal oscillations in the solution of the SWE. In this case, averaging could be very helpful. There is no literature about how to use averaging in the case of time-dependent boundary conditions as in Equation (3.25). In Dedalus, the matrix discretising the left hand side has two additional rows and columns at the end. This means, that the last two entries of the solution vector $\mathbf{u} \in \mathbb{R}^M$ should contain the boundary values. For other discretisation methods this could be different, but here we use the corresponding notation. According to the transformation in Equation (4.10) the modulation equation has the boundary conditions

$$\begin{aligned} w_{1,\text{left}} &= (e^{\frac{1}{\varepsilon}\mathfrak{L}t}\mathbf{u}(t))_{M-1} \\ w_{2,\text{right}} &= (e^{\frac{1}{\varepsilon}\mathfrak{L}t}\mathbf{u}(t))_{M}. \end{aligned} \tag{4.16}$$

For the averaged equation one could as well use phase averaging for the boundary values, that is

$$\begin{aligned} \bar{w}_{1,\text{left}} &= \frac{1}{\eta}\int_{-\frac{\eta}{2}}^{\frac{\eta}{2}} \rho\left(\frac{s}{\eta}\right)\left(e^{\frac{1}{\varepsilon}\mathfrak{L}(s+t)}\mathbf{u}(s+t)\right)_{M-1}\, ds \\ \bar{w}_{2,\text{right}} &= \frac{1}{\eta}\int_{-\frac{\eta}{2}}^{\frac{\eta}{2}} \rho\left(\frac{s}{\eta}\right)\left(e^{\frac{1}{\varepsilon}\mathfrak{L}(s+t)}\mathbf{u}(s+t)\right)_{M}\, ds. \end{aligned} \tag{4.17}$$

These could be implemented using Dedalus' `GeneralFunction` as shown in Figure 4.11. But for each boundary condition there has to be a spatial derivative on the left hand side

```
problem.add_equation("w1(x='left') = avrg1(t)")
problem.add_equation("w2(x='right') = avrg2(t)")
```

Figure 4.11: Boundary conditions with `GeneralFunction` `avrg1` and `avrg2`.

of the PDE in Dedalus. Because of the right hand side of Equations (4.11) and (4.12) being nonlinear, it is not possible to move derivatives to the left hand side. These have to be implemented one `GeneralFunction` each as well, see Figure 4.12. One would need to

```
mod_problem.add_equation("dt(w1) = aF1(t, w1, w2)"
mod_problem.add_equation("dt(w2) = aF2(t, w1, w2)"
```

Figure 4.12: Modulation equation with `GeneralFunction` `aF1` and `aF2`.

find some solution to 'cheat' a derivative onto the left hand side in order to obtain stable solutions in Dedalus. This would require modifications inside the Dedalus code, which is not feasible for this work. Therefore, no results for this approach will be shown.

4.2.4 The choice of the averaging window

Depending on the averaging window η also slower oscillations are being mitigated and this increases the averaging error [27]. On the other hand, the timestepping error decreases with increasing η, because a larger averaging window makes the problem less stiff [27]. The error of the coarse solution to the fine is the sum of the timestepping and averaging error. Hence, there is an optimal averaging window where the error to the fine solution is the smallest [27]. Finding this optimal η is not the main focus of this work. But we show plots of the solution of Equation (4.14) for Example 4.3. We first choose a rather small averaging window of $\eta_1 = 0.4$ and then a much larger value of $\eta_2 = 4$. The effect of changing the averaging window can be seen in Figure 4.13. For η_1, there are small temporal oscillations visible which are clearly being mitigated for the larger value of η_2. In Φ, there is hardly any difference noticeable for the two different averaging windows. In the next subsection we will see that the averaging window can have an influence on the number of Parareal iterations.

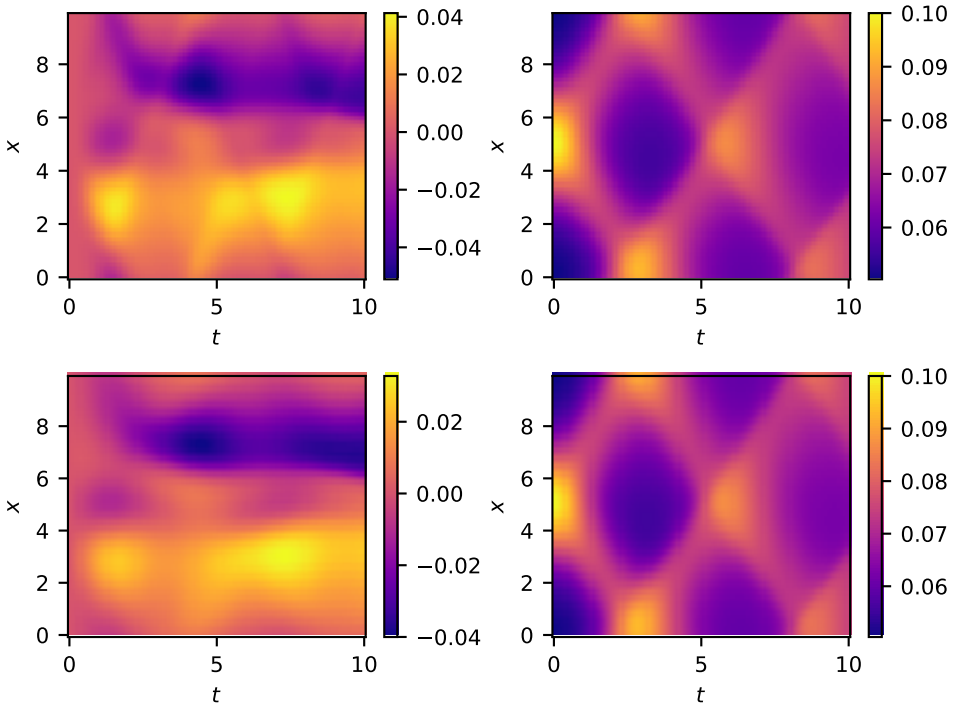

Figure 4.13: Second solution component w_2 of the modulation equation (left) and Φ of the SWE (right) for different averaging windows $\eta_1 = 0.4$ (top) and $\eta_2 = 4$ (bottom). The bathymetry and initial condition are from Ex. 4.3.

4.2.5 Comparison of Parareal with and without averaging

We will investigate convergence of Asymptotic Parareal for the SWE in Equation (4.14) by comparing its number of iterations to Parareal without averaging. For Parareal without averaging there are two options. Firstly, we can take use the transformation (4.10) as it is used for Asymptotic Parareal, but let the coarse propagator solve the modulation equation (4.11) without averaging such that the only difference between coarse and fine propagator is the time step. This is expected to slow down the coarse propagator, as the phase-averaging allows us to use a coarser time step. But this could possibly be compensated by earlier convergence of Parareal. We will call this Parareal for the modulation equation. The second option is to just solve the SWE (4.14) with Parareal and not use the transformation (4.10) at all. This is referred to as Parareal for the original equation.

There is not any comparison like this found in literature. A fair comparison of runtimes would require an optimised implementation of the averaging which is not the main topic in this work. In our case there is two main reasons for long runtimes. On the one hand, the right hand side in Equation (4.12) is computed in a nested for-loop that should be parallelised. On the other hand, the way of implementing the right hand sides of Equation (4.11) and (4.12) slows down the computations a lot. In the Dedalus equations everything needs to be Dedalus operators, fields or numbers. As there are no Dedalus operators for the specific right hand sides of our equations, these have to be implemented using the Dedalus `GeneralFunction`. It takes another user-defined function and produces a Dedalus operator as output, but this takes quite some time. Consequently, we will only investigate perfor-

mance of Parareal in terms of number of iterations, the error to the sequential solution and Parareal defect Δ^k per iteration k. We will look at a few different examples.

First, consider Example 4.2. We set the averaging window to $\eta = 0.4$, use a time step of $\delta t = 0.001$ for the fine propagator and $\Delta t = 0.05$ for the coarse propagator. Figure 4.14 shows convergence for Asymptotic Parareal, Parareal for the modulation equation without averaging and Parareal for the original equation. For the given example, solving the mod-

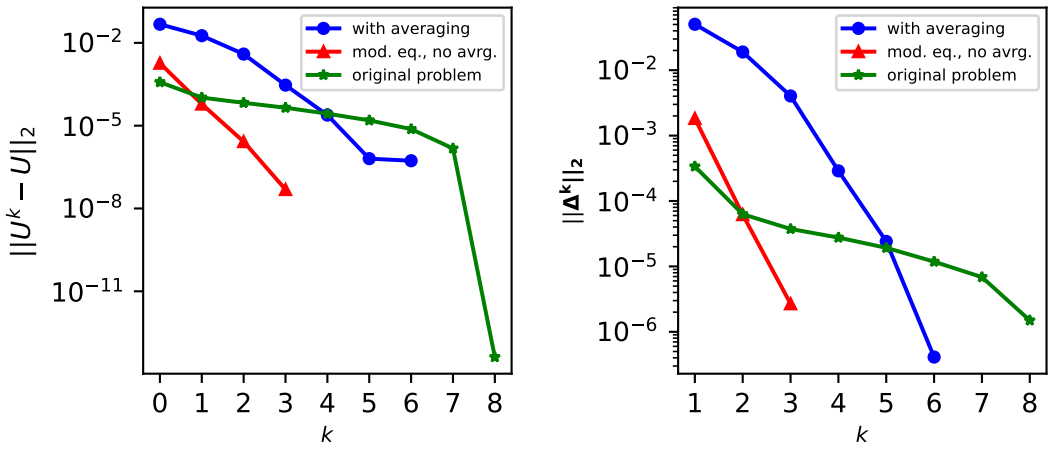

Figure 4.14: Error to sequential solution (left) and defect for Parareal with averaging (blue dots), Parareal for the modulation equation but without averaging (red triangles) and Parareal for the original equation (green stars) for Example 4.2.

ulation equation instead of the original equation leads to a large improvement of Parareal convergence from the maximum number of iterations to only 4 iterations. With averaging the convergence is a bit slower with 6 iterations. At iteration $k = 0$, i.e. for the initial guess, the error to the sequential solution is the highest for Parareal with averaging. A reason might be that the averaging window η is not optimal. However, the error goes down far more rapidly than for Parareal for the original equation.

Let us consider another example where the convergence behaviour is different.

Example 4.4. *Let $T = 3$, $[L, R] = [0, 10]$, $g = 9.81$, $\kappa = 0.2$, $D = 0.3$ and define the bathymetry and initial condition*

$$\begin{aligned} b(x) &= 0.015 \sin(\pi x) + 0.015 \\ \Phi(x, 0) &= 0.2 \exp(-0.2(x-5)^2) + 0.05 \\ u(x, 0) &= 0 \end{aligned}$$

for $x \in [L, R]$.

We use a time step of $\delta t = 0.001$ for the fine propagator, $\Delta t \approx 0.05$ for the coarse propagator and an averaging window of $\eta = 0.6$. The errors of the solution of Parareal with averaging, Parareal for the modulation equation without averaging and Parareal for the original equation for Example 4.4 is shown in Figure 4.15. The convergence behaviour of Parareal is very different compared to the previous example. For the modulation equation without averaging the errors increase before going down, so convergence is even slightly

worse than for the original equation. However, with averaging the errors decrease monotonically and Parareal converges after 6 iterations already. Without averaging, very slow

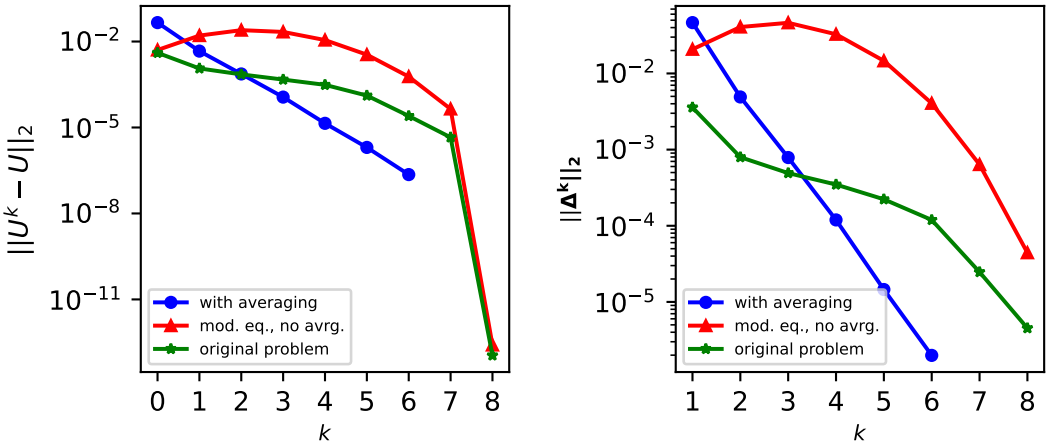

Figure 4.15: Error to sequential solution (left) and defect for Parareal with averaging (blue dots), Parareal for the modulation equation without averaging (red triangles) and Parareal for the original equation (green stars) for Example 4.4.

convergence can be observed which was already observed for different hyperbolic problems [72].

Now we take a look at numbers of iterations for the three cases and for different averaging windows. Table 4.1 shows the number of iterations for Asymptotic Parareal, Parareal for the modulation equation and Parareal for the original equation on 8 time slices. For all examples, the convergence of Parareal for the original equation is very bad,

		Parareal iterations				
	Δt	$\eta = 0.2$	$\eta = 0.4$	$\eta = 0.8$	Mod. eq.	Original eq.
Ex. 4.2	0.1	6	6	8	-	8
	0.05	5	6	6	3	8
	0.01	5	6	6	2	2
Ex. 4.3	0.1	6	5	8	-	8
	0.05	4	5	5	3	8
	0.01	4	5	5	1	2
Ex. 4.4	0.125	8	8	8	-	8
	0.054	6	6	7	8	8
	0.01	5	6	7	2	2

Table 4.1: Number of Parareal iterations for Asymptotic Parareal ($\eta \in \{0.2, 0.4, 0.8\}$), Parareal for the modulation equation and Parareal for the original equation. The tolerance for the stopping criterion is $\epsilon_P = 10^{-6}$, the fine time step is $\delta t = 1 \times 10^{-3}$ s and the number of spatial grid points $M = 64$. The number of used time slices is 8.

except for the finest time step for Δt. Apparently, in this case the coarse propagator is too accurate, leading to convergence after only two iterations. We have already observed

this phenomenon in Subsection 4.1.3. This was similar for Example 4.1, see Figure 4.5. Parareal convergence for the modulation equation without averaging is most of the times better than for the original equation. But for the coarse time step $\Delta t = 0.1$ it did not converge at all due to instabilities. This is indicated by the minus in Table 4.1. Asymptotic Parareal needs fewer iterations than the original SWE, except for the finest coarse time step $\Delta t = 0.01$. The only case where it is better than Parareal for the modulation equation is Example 4.4 for $\Delta t = 0.054$ with only 6 (for $\eta \in \{0.2, 0.4\}$) versus 8 iterations. Another observation is that the number of iterations tends to become fewer for smaller values of the averaging window η. This fits to the previous observation that the chosen averaging windows are most likely not optimal, except for Example 4.3 for $\Delta t = 0.1$, where $\eta = 0.4$ seems to be a good choice.

To sum up, the modulation equation already improves Parareal convergence except for Example 4.4. Changing the initial condition or the bathymetry in this example to one of the other two examples led to better convergence of Parareal for the modulation equation. Hence, the reason for bad convergence here seems to be both the initial condition and bathymetry. This is the only example where Asymptotic Parareal wins in terms of iterations. The dependence of convergence behaviour on the initial condition and bathymetry should be further investigated.

4.3 Solving the adjoint problem parallel in time

The adjoint problem has time dependent coefficients which is why it does not fit into the form that is required for Asymptotic Parareal. Hence, we have to use another method to solve the adjoint parallel in time. We would like to make use of the linearity of the equation. There is a PinT method that is particularly well suited for linear hyperbolic problems, which is the ParaExp algorithm. It is a PinT method for initial value problems that was first introduced by Gander and Güttel [35]. Originally, the method can be applied on linear initial value problems of the form

$$\mathbf{u}'(t) = A\mathbf{u}(t) + \mathbf{g}(t), \quad \mathbf{u}(0) = \mathbf{u}_0, \quad t \in [0, T] \tag{4.18}$$

with $A \in \mathbb{C}^{m \times m}$, $\mathbf{u}(t) \in \mathbb{C}^m$ and a source term $\mathbf{g}(t) \in \mathbb{C}^m$. The method is based on the decomposition of the problem into an inhomogeneous and a homogeneous part (i.e. $\mathbf{g} \equiv 0$). The latter problem has the solution $\mathbf{u}(t) = \mathbf{u}_0 e^{At}$. Hence, exponential integration is applied to solve the homogeneous part, which is done by using Krylov subspace methods. Parallel efficiency above 50% was observed in numerical experiments which is more than Parareal can achieve [35]. As the adjoint problems are linear, we hope to speed up the computation of its solution with ParaExp.

4.3.1 Adjoint problem

As we will use Equation (4.14) for the bathymetry reconstruction, we need the corresponding adjoint equation. It can be derived the same way as in the previous chapters. We use

the objective functional

$$\begin{aligned} J(b,\Phi) =& \frac{\gamma}{2}\int_Q (\Phi + D - H_{\text{obs}})^2 \; d(x,t) + \frac{\delta}{2}\int_\Omega (\Phi(\cdot,T) + D - H_{\text{obs}}(\cdot,T))^2 \; dx \\ &+ \frac{\lambda_1}{2}\int_\Omega b^2 \; dx + \frac{\lambda_2}{2}\int_\Omega b_x^2 \; dx \end{aligned} \tag{4.19}$$

and define the Lagrangian

$$\begin{aligned} \mathcal{L}(\Phi,u,b,p) = J(b,\Phi) &- \int_Q (\Phi_t + Du_x + (\Phi u)_x - (bu)_x)\, p_1 \; d(x,t) \\ &- \int_Q (u_t + g\Phi_x + uu_x + \kappa u)\, p_2 \; d(x,t) \\ &- \int_\Omega (\Phi(\cdot,0) + D - H_{\text{obs}}(\cdot,0))\, p_3 \; dx. \end{aligned} \tag{4.20}$$

We compute the partial derivatives and use integration by parts and the periodic boundary conditions. We get

$$\begin{aligned} \mathcal{L}_\Phi(\Phi,u,b,p)\varphi &= \gamma\int_Q (\Phi + D - H_{\text{obs}})\varphi \; d(x,t) + \delta\int_\Omega (\Phi(\cdot,T) + D - H_{\text{obs}}(\cdot,T))\varphi \; dx \\ &\quad - \int_Q (\varphi_t + (\varphi u)_x)\, p_1 \; d(x,t) - \int_Q g\varphi_x p_2 \; d(x,t) \\ &\quad - \int_\Omega \varphi(\cdot,0) p_3 \; dx \\ &= \gamma\int_Q (\Phi + D - H_{\text{obs}})\varphi \; d(x,t) + \delta\int_\Omega (\Phi(\cdot,T) + D - H_{\text{obs}}(\cdot,T))\varphi \; dx \\ &\quad + \int_Q \varphi p_{1,t} \; d(x,t) - \int_\Omega [\varphi p_1]_0^T \; dx + \int_Q \varphi u p_{1,x} \; d(x,t) + \int_Q g\varphi p_{2,x} \; d(x,t) \\ &\quad - \int_\Omega \varphi(\cdot,0) p_3 \; dx \end{aligned} \tag{4.21}$$

and

$$\begin{aligned} \mathcal{L}_u(\Phi,u,b,p)\varphi &= -\int_Q (D\varphi_x + (\Phi\varphi)_x - (b\varphi)_x)\, p_1 \; d(x,t) - \int_Q (\varphi_t + (\varphi u)_x + \kappa\varphi)\, p_2 \; d(x,t) \\ &= \int_Q (D + \Phi - b)\, p_{1,x}\varphi \; d(x,t) + \int_Q \varphi p_{2,t} \; d(x,t) - \int_\Omega [\varphi p_2]_0^T \\ &\quad + \int_Q \varphi u p_{2,x} \; d(x,t) - \int_Q \kappa\varphi p_2 \; d(x,t), \end{aligned} \tag{4.22}$$

from which we obtain the adjoint equation and corresponding final conditions. As before, we use the time transformation $\tau := T - t$. Let $\tilde{\Phi}$, $\tilde{u}$ be the time transformed variables from the forward problem. Define $\tilde{p}_1, \tilde{p}_2$ as the backward adjoint variables. Then the adjoint

problem that corresponds to Problem (4.14) reads

$$\begin{aligned} \tilde{p}_{1,\tau} &= \tilde{u}\tilde{p}_{1,x} + g\tilde{p}_{2,x} + \gamma(D + \tilde{\Phi} - \tilde{H}_{\text{obs}}) \\ \tilde{p}_{2,\tau} &= \tilde{u}\tilde{p}_{2,x} + (D + \tilde{\Phi} - b)\tilde{p}_{1,x} - \kappa\tilde{p}_2 \\ \tilde{p}_1(\cdot, 0) &= \delta(\tilde{\Phi}(\cdot, 0) + D - \tilde{H}_{\text{obs}}(\cdot, 0)) \\ \tilde{p}_2(\cdot, 0) &= 0. \end{aligned} \tag{4.23}$$

Note that inserting $h = D + \Phi - b$ in Equation (3.13) yields the same adjoint problem as in Equation (4.23). For the waterchannel problem in Section 3.2 the only difference in the adjoint problem are the boundary conditions. They read

$$\begin{aligned} \tilde{p}_1(L, \cdot) &= -\frac{\tilde{u}(L, \cdot)}{D + \tilde{\Phi}(L, \cdot) - b(L)}\tilde{p}_2(L, \cdot) \\ \tilde{p}_2(R, \cdot) &= 0. \end{aligned} \tag{4.24}$$

The adjoint problem has the form

$$\mathbf{u}'(t) = A(t)\mathbf{u}(t) + \mathbf{g}(t), \quad \mathbf{u}(0) = \mathbf{u}_0, \quad t \in [0, T]. \tag{4.25}$$

This problem almost has the form of Equation (4.18) but with a time dependent matrix $A(t)$. As the solution of the homogeneous problem is not $\mathbf{u}_0 e^{A(t)t}$ we cannot use the original ParaExp for the adjoint problem. We have to choose a variant of this method that is suitable for our problem.

4.3.2 The nonlinear ParaExp algorithm

A variant of ParaExp for nonlinear problems that can also be applied to problems with a time dependent matrix $A(t)$ was introduced in 2018 by Gander, Güttel and Petcu [34]. It can be used for problems of the form

$$\mathbf{u}'(t) = A\mathbf{u}(t) + B(\mathbf{u}(t)) + \mathbf{g}(t), \quad \mathbf{u}(0) = \mathbf{u}_0, \quad t \in [0, T] \tag{4.26}$$

with nonlinear operator $B : \mathbb{C}^m \to \mathbb{C}^m$. The adjoint problem (4.25) can be rewritten as

$$\mathbf{u}'(t) = A_0\mathbf{u}(t) + (A(t) - A_0)\mathbf{u}(t) + \mathbf{g}(t), \quad \mathbf{u}(0) = \mathbf{u}_0, \quad t \in [0, T] \tag{4.27}$$

with e.g. $A_0 = A(0)$. Then it fits the form in Equation (4.26) with $B := A(t) - A_0$. In detail, the algorithm works as follows. Like for Parareal, we split the time interval into time slices $[T_{n-1}, T_n],\ n = 1, \ldots, N$, where N is the number of used cores. Consider iteration k, where k is the ParaExp iteration counter. In the first step, the nonlinear ParaExp solves the linear problems

$$\begin{aligned} &(\mathbf{w}_n^k)'(t) = A_0\mathbf{w}_n^k(t) \\ &\mathbf{w}_n^k(T_{n-1}) = \mathbf{u}_{n-1}^{k-1}(T_{n-1}) - \sum_{j=1}^{n-1} \mathbf{w}_j^{k-1}(T_{n-1}),\ \mathbf{w}_1^k(T_0) = \mathbf{u}_0 \end{aligned} \tag{4.28}$$

on the time interval $[T_{n-1}, T]$ in parallel which can be solved efficiently using Krylov subspace methods [34]. These methods compute an orthonormal basis $V_m = [v_1, \ldots, v_m]$

of the Krylov subspace and an upper Hessenberg matrix H_m. Gallopoulos and Saad [87] state that, with $\mathbf{e}_1$ being the first unit vector, we can approximate the solution of the linear problem by

$$e^{tA_0}\mathbf{w}_n^k \approx \|\mathbf{w}_n^k\| V_m e^{tH_m}\mathbf{e}_1. \tag{4.29}$$

The matrix exponential of H_m should be much faster to compute than e^{A_0} as the Hessenberg matrix is only of size $m \times m$. The second step in the ParaExp iteration is to solve the nonlinear problems

$$\begin{aligned}(\mathbf{u}_n^k)'(t) &= A_0\mathbf{u}_n^k(t) + (A(t) - A_0)\mathbf{u}_n^k(t) + \mathbf{g}(t)\\ \mathbf{u}_n^k(T_{n-1}) &= \sum_{j=1}^{n} \mathbf{w}_n^k(T_{n-1})\end{aligned} \tag{4.30}$$

on the time interval $[T_{n-1}, T_n]$ in parallel for each process n. Then, the approximate solution in iteration k is

$$\mathbf{u}^k(t) = \mathbf{u}_n^k(t), \quad t \in [T_{n-1}, T_n). \tag{4.31}$$

4.3.3 Convergence of ParaExp for the adjoint problem

In order to investigate convergence of the nonlinear ParaExp for the adjoint equations, we will plot the error to the sequential solution for each ParaExp iteration. We consider both types of boundary conditions that were treated so far in this thesis. For ParaExp, there is no stopping criterion found in literature. A possibility would be to demand that the increment is smaller than some tolerance $\epsilon_{PE} > 0$, like in Parareal. That is

$$\|\mathbf{u}^{k+1}(t) - \mathbf{u}^k(t)\| \leq \epsilon_{PE}. \tag{4.32}$$

4.3.3.1 Periodic boundary conditions

Consider a computed observation for Example 4.1 with time step $5 \cdot 10^{-5}$ and 130 grid points. For the computation of the solution with Asymptotic Parareal we use the same initial condition but the bathymetry $b = 0$ and choose the following parameters. The tolerance for the Parareal stopping criterion is $\epsilon_P = 10^{-6}$. The fine time step is $\delta t = 0.001$, the coarse time step is $\Delta t = 0.05$, the averaging window $\eta = 0.2$ and the number of spatial grid points $M = 64$. The number of time slices is chosen as $P = 8$ for both Parareal and nonlinear ParaExp. The forward solution and the observation determine the coefficients and initial condition for the adjoint equation (4.23) that we will solve with nonlinear ParaExp. We set the stopping tolerance ϵ_{PE} to machine precision and solve the linear problem with SciPy's function `expm` for reasons of simplicity. Figure 4.16 shows the ParaExp solution on the left and the relative ℓ^2-error to the sequential solution for all iterations on the right. The convergence of ParaExp in Figure 4.16 is a bit worse than the results for the nonlinear wave equation with strong nonlinearity in Gander et al. [34]. In the kth iteration the error to the sequential solution stays above 10^{-5} for all time slices $[T_{n-1}, T_n]$, $n = k+1, \ldots, N$. Now we set the tolerance to $\epsilon_{PE} = 10^{-3}$ and compute the solution again with nonlinear ParaExp to see if there is any hope for speed-up. The algorithm terminates after 4 iteration as can be seen in Figure 4.17. The solution looks the same as for the full number of iterations. Speed-up is still to be investigated, it is hard to say whether half of the maximum number of iterations is enough to obtain speed-up.

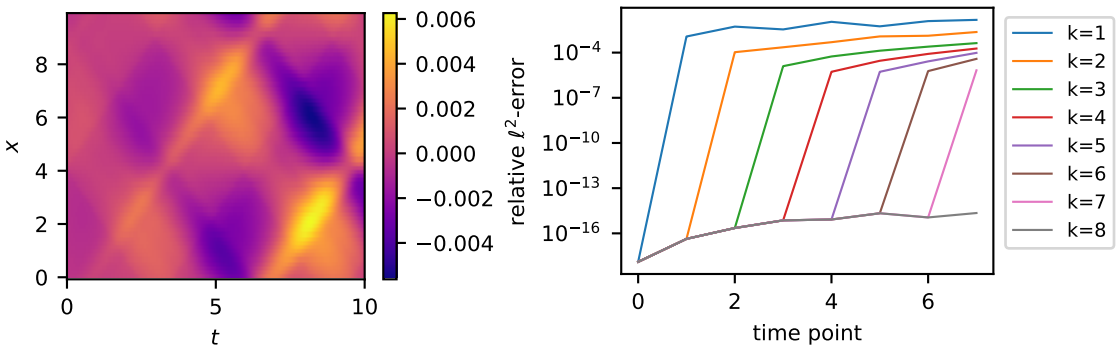

Figure 4.16: Solution p_1 for the adjoint equation with periodic boundary conditions (left) and error of the ParaExp solution to the sequential solution versus time slice for different number of iterations k (right).

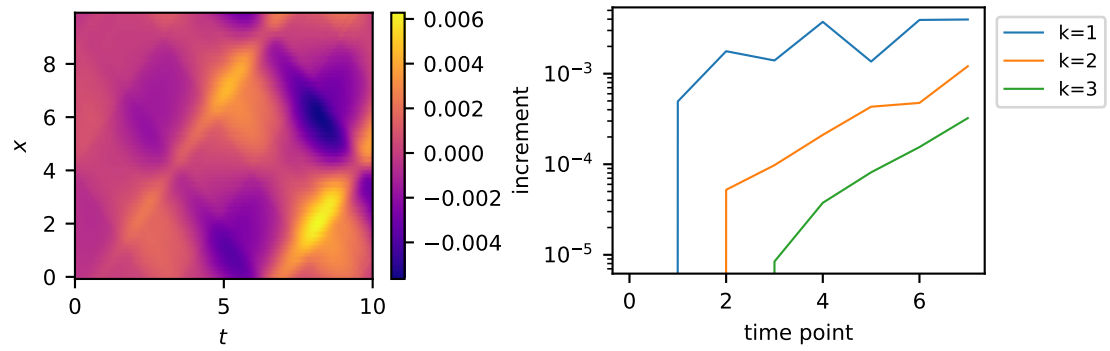

Figure 4.17: Solution p_1 for the adjoint equation with periodic boundary conditions (left) and ParaExp increment versus time slice for different number of iterations k (right).

4.3.3.2 Wave flume

Additionally, we apply ParaExp on the adjoint equation for the wave flume setup. The only difference to Equation (4.23) are the boundary conditions which are given by Equation (4.24). For this adjoint equation the convergence behaviour is similar, see Figure 4.18, even though the adjoint solution looks totally different. But in this case, the error drops below 10^{-5} for all time slices after iteration $k = 5$. This seems promising, but despite using the same stopping tolerance as for the periodic boundary conditions, ParaExp takes the maximum number of iterations to terminate. Apparently, the stopping criterion is not very useful for this example. Even though the error to the sequential solution for later times decreases slightly faster than for the periodic boundary conditions, the increment goes down very slowly. It hardly reaches the tolerance $\epsilon_{PE} = 10^{-3}$ in the last iteration.

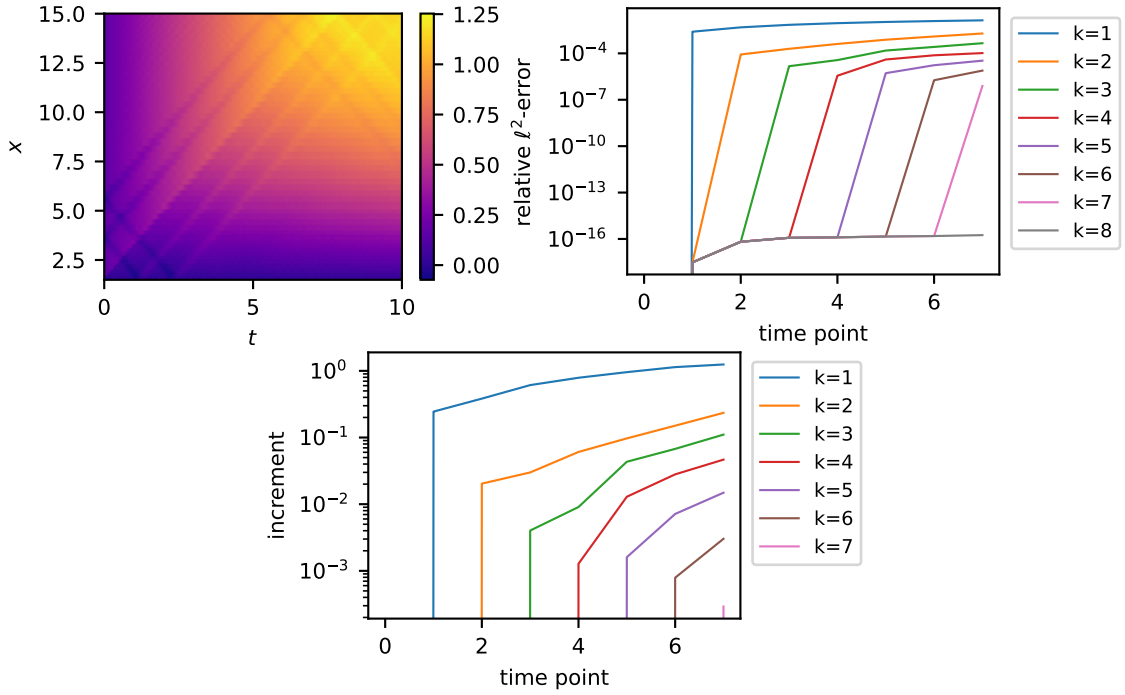

Figure 4.18: Solution p_1 for the adjoint equation that is used for the reconstruction of the hill in the wave flume (top left), error of the ParaExp solution to the sequential solution versus time slice k (top right) and increment versus time slice (bottom) for different number of iterations.

Chapter 5

Bathymetry reconstruction with ParaExp and (asymptotic) Parareal

In this chapter, we will investigate the quality of the bathymetry reconstruction when using parallel-in-time methods for solving the forward and adjoint equation. The difference of the PinT solutions to the sequential solutions for forward and adjoint problem could have an effect on the optimisation as this leads to slightly different descent directions. At the beginning of the optimisation this has most likely no effect, because a not very accurate descent direction will still lead to a significant decrease of the value of the objective functional. But as the method gets closer to the minimum of the objective functional f, the norm of the adjoint solution gets smaller. Consequently, the error of the PinT solution to the sequential solution has a larger effect on the descent direction.

Parallelisation of the minimisation algorithm

For the time parallel versions of gradient descent and L-BFGS, the computation of the gradient has to be rewritten as a parallel program. As we use the Python package `mpi4py`, we show the corresponding MPI pseudocode. Algorithm 6 is a more detailed and parallel version of Line 4 in Algorithm 1, where the gradient is computed in serial. The subscript

Algorithm 6 Gradient computation

Require: b, H_{obs}
1: $H^{loc} \leftarrow \text{solvePDE}(b)$
2: $(p_1^{loc}, p_2^{loc}) \leftarrow \text{solveAdjointPDE}(H^{loc}, b)$
3: $v_e^{loc} \leftarrow g \cdot \text{quadr}(p_{2,x}^{loc}, dt)$
4: $v_J^{loc} \leftarrow \gamma \cdot \text{quadr}(H^{loc} - H_{\text{obs}}^{loc}, dt)$
5: **if** $rank == size - 1$ **then**
6: $\quad v_J^{loc} \leftarrow v_J^{loc} + \delta \cdot (H^{loc}[-1] - H_{\text{obs}}^{loc}[-1])$
7: **if** $rank == 0$ **then**
8: $\quad v_J^{loc} \leftarrow v_J^{loc} - p_1^{loc}[0]$
9: $v_e \leftarrow \text{comm.allreduce}(v_e^{loc}, \text{op=MPI.SUM})$
10: $v_J \leftarrow \text{comm.allreduce}(v_J^{loc}, \text{op=MPI.SUM})$
11: $v_J \leftarrow v_J + \lambda_1 b - \lambda_2 b_{xx}$
12: $\tilde{v} \leftarrow v_e + v_J$
13: $v \leftarrow \text{solveLBVP}(\tilde{v})$

"loc" indicates that the corresponding variable or execution only takes place locally on each process. After computing the solutions of forward and adjoint problem in parallel in Lines 1 and 2, all needed derivatives are being computed locally on each core. The latter is not shown in an extra line in the pseudocode. Then, in Lines 3 and 4 a quadrature rule is being applied in order to approximate the integrals in (3.15), e.g. Simpson's rule. In the next four lines, last rank `size-1` adds the mismatch at the final time and the first rank `0` subtracts p_1 at time $t = 0$. Then, using `allreduce` in Lines 9 and 10, all previously computed parts of the L^2-gradient are being gathered. As all ranks know the bathymetry b and the factors $\lambda_{1/2}$ for the regularisation terms, the terms $\lambda_1 b$ and $-\lambda_2 b_{xx}$ can be added in Line 11 after the `allreduce`. In Line 13 the linear boundary value problem (3.17) is being solved on each rank using the previously computed L^2-gradient. This is the way it could be implemented to save some time for communication, but for the numerical results a simpler implementation is being used. In Lines 1 and 2 it is also possible to use the global solution by using an `Allgather` in the function `solvePDE`.

5.1 Periodic boundary conditions

Firstly, we perform a PinT reconstruction using the forward equation from Chapter 4 which is a reformulation of the SWE (2.17). We use Asymptotic Parareal to parallelise the forward solution and ParaExp for the adjoint. Consider Equation (4.14) with periodic boundary conditions on $[L, R] = [0, 10]$ and the objective functional (4.19).

5.1.1 Reconstructions

We reconstruct two different types of bathymetries using L-BFGS. Then we will compare relative errors to the exact bathymetries to the errors of the corresponding serial reconstructions. As our implementation of Asymptotic Parareal in Dedalus is not very efficient, it does not make much sense evaluating runtimes. This is left for future work. The idea is to show that the results of the PinT optimisation are sufficiently close to the reconstruction from the serial run. All PinT optimisations are being computed on eight cores, i.e. with eight time slices.

5.1.1.1 Sinusoidal bathymetry

Consider Example 4.3. This is exactly the same problem as in Section 3.1. For the optimisation, we choose the following parameters. The coefficients in the objective functional are $\gamma = \delta = 0.5$ and $\lambda_1 = \lambda_2 = 10^{-7}$. We use tolerances $\varepsilon = 10^{-6}$ for L-BFGS, $\epsilon_P = 10^{-5}$ for Asymptotic Parareal and $\epsilon_{PE} = 10^{-4}$ for ParaExp and save ten previous descent directions in L-BFGS. The number of spatial grid points is $M = 64$, the time step for the fine propagator is $\delta t = 1 \times 10^{-3}\,\mathrm{s}$ and the approximate time step $\Delta t = 0.05\,\mathrm{s}$ for the coarse propagator. The averaging window we choose to be $\eta = 0.2$. The initial guess for the bathymetry is

$$b = 0 \qquad \text{on } [0, 10]. \tag{5.1}$$

Figure 5.1 shows the L-BFGS reconstruction after the last iteration (left) and the relative ℓ^2-errors to the exact bathymetry throughout the optimisation process (right). The quality of the serial reconstruction with a relative ℓ^2-error of 5.1% is almost the same as for the

corresponding optimisation problem in Section 3.1. The only difference here is the use of the perturbation height instead of water depth. Apparently, this does not make any difference for the optimisation. The PinT reconstruction did not work, because the solution of the modulation equation was unstable at some point in the line search with Wolfe conditions. Apparently, Asymptotic Parareal for the forward equation can get unstable for some bathymetries and this should be further investigated. The serial optimisation stopped after 13 iterations as no more step size was found.

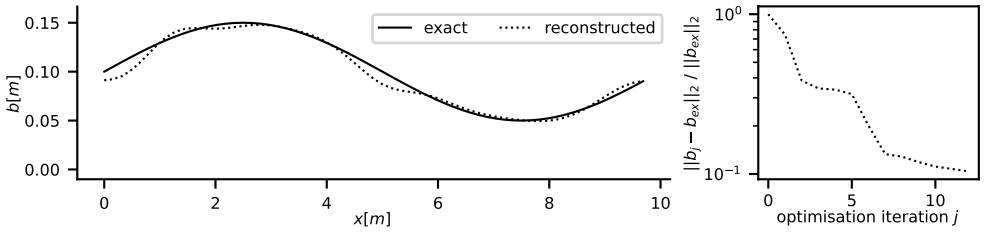

Figure 5.1: Reconstruction of the bathymetry (left) and relative errors against iteration (right) for the serial optimisation (dotted) with L-BFGS.

5.1.1.2 Gaussian shaped bathymetry

Consider Example 4.2 where the bathymetry is defined by a Gaussian. We choose the same parameters, tolerances and initial guess as for the sinusoidal bathymetry. Figure 5.2 shows the serial and PinT reconstruction on the left plot and the relative ℓ^2-errors to the exact bathymetry on the right. Both the serial and PinT reconstruction stopped, because no step size was found in the line search with Wolfe conditions. The relative ℓ^2-error to the exact bathymetry after the last iteration is about 46.39% for the serial reconstruction versus 67.57% for the bathymetry obtained by the PinT optimisation which stopped six iterations earlier than the serial L-BFGS. For all optimisations iteration Asymptotic Parareal converged after two to four iterations. ParaExp always needed five to seven iterations to converge.

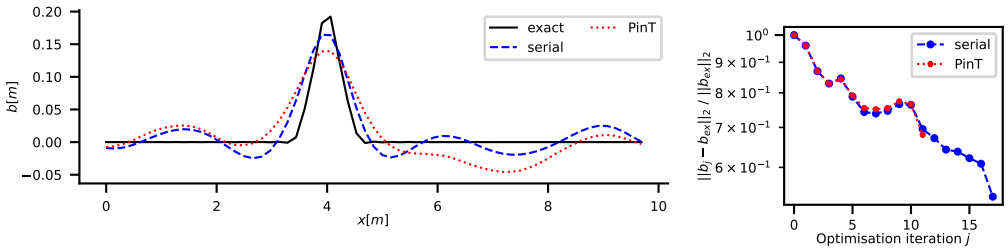

Figure 5.2: Reconstruction of the bathymetry (left) and relative errors against iteration (right) for the PinT (red dotted) and serial optimisation (blue dashed) with L-BFGS.

5.2 Reconstruction of the bathymetry in the wave flume

In Section 4.2, we discussed the issues with the implementation of the setup for the experiment within asymptotic Parareal. Thus, we will perform a PinT bathymetry reconstruc-

tion from the experimental data with vanilla Parareal for the forward and ParaExp for the adjoint problem. We use the data from the first two sensors as this gave the best results.

5.2.1 Equations and gradient

The SWE used for the reconstruction of the hill in the wave flume from Section 3.2 reads

$$\begin{aligned}
&\begin{pmatrix} h \\ u \end{pmatrix}_t + \begin{pmatrix} hu \\ \frac{1}{2}u^2 + gh \end{pmatrix}_x = \begin{pmatrix} 0 \\ -gb_x - \kappa u \end{pmatrix} \quad \text{on } Q \\
&h(\cdot, 0) = H_{\text{obs}}(\cdot, 0) - b \quad \text{on } [0, T] \\
&h(L, \cdot) = H_{\text{obs}}(L, \cdot) - b(L) \quad \text{on } \Omega \\
&u(R, \cdot) = 0 \quad \text{on } \Omega
\end{aligned}$$

and the corresponding adjoint problem is

$$\begin{aligned}
&\tilde{p}_{1,\tau} - \tilde{u}\tilde{p}_{1,x} - g\tilde{p}_{2,x} = \gamma\left(\tilde{h} + b - \tilde{H}_{\text{obs}}\right) \\
&\tilde{p}_{2,\tau} - \tilde{h}\tilde{p}_{1,x} - \tilde{u}\tilde{p}_{2,x} = -\kappa\tilde{p}_2 \\
&\tilde{p}_1(\cdot, 0) = \delta\left(\tilde{h}(\cdot, 0) + b - \tilde{H}_{\text{obs}}(\cdot, 0)\right) \\
&\tilde{p}_2(\cdot, 0) = 0 \\
&\tilde{p}_1(L, \cdot) = -\frac{\tilde{u}(L, \cdot)}{\tilde{h}(L, \cdot)}\tilde{p}_2(L, \cdot) \\
&\tilde{p}_2(R, \cdot) = 0.
\end{aligned}$$

This is the same as Problem (4.23), where we used the substitution $h = D + \Phi - b$. Thus, we can apply the nonlinear ParaExp accordingly. We use the objective functional from Section 3.2 for the reconstruction from measurements, i.e.

$$\begin{aligned}
J(b, H) :=& \frac{\gamma}{2}\int_Q \left(\sum_{i=1}^{m_p}(H(x_i, t) - H_{\text{obs},i}(t))\exp\left(-\frac{1}{2}\frac{(x - x_i)^2}{0.045}\right)\right)^2 d(x, t) \\
&+ \frac{\delta}{2}\int_\Omega \left(\sum_{i=1}^{m_p}(H(x_i, T) - H_{\text{obs},i}(T))\exp\left(-\frac{1}{2}\frac{(x - x_i)^2}{0.045}\right)\right)^2 dx \\
&+ \frac{\lambda_1}{2}\int_\Omega b^2\, dx + \frac{\lambda_2}{2}\int_\Omega b_x^2\, dx.
\end{aligned}$$

The corresponding H^1-gradient is given by Equations (3.48) and (3.45) in Section 3.2.

5.2.2 Reconstructions

We will reconstruct the hill in the wave flume in four ways, that is with Parareal for the forward and ParaExp for the adjoint problem, Parareal for both problems, Parareal for the forward problem and the adjoint in serial and both problems in serial. All results will be compared.

We use the following parameters. The coefficients in the objective functional we choose to be $\gamma = \delta = 0.5$, $\lambda_1 = 10^{-7}$ and $\lambda_2 = 10^{-5}$. The number of spatial grid points is

$M = 64$, the time step for the fine propagator is $\delta t = 1 \times 10^{-3}$ s and the approximate time step $\Delta t = 0.05$ s for the coarse propagator. The tolerances for the stopping criteria are $\varepsilon = 10^{-7}$ for L-BFGS and $\epsilon_{P,f} = 5{\cdot}10^{-4}$ for Parareal for the forward problem, $\epsilon_{P,a} = 5{\cdot}10^{-5}$ for Parareal for the adjoint problem and $\epsilon_{PE} = 10^{-3}$ for ParaExp. With smaller tolerances the PinT methods need almost always the maximum number of iterations such that there would be no chance to speed up the optimisation. The idea is to accept less accurate solutions of the PDEs in favour of speed-up in case that the quality of the reconstruction does not suffer a lot.

Let b_{ex} be the hill that was used in the experiment in Section 3.2. We use eight time slices for the PinT reconstruction. The serial and PinT optimisations yield almost the

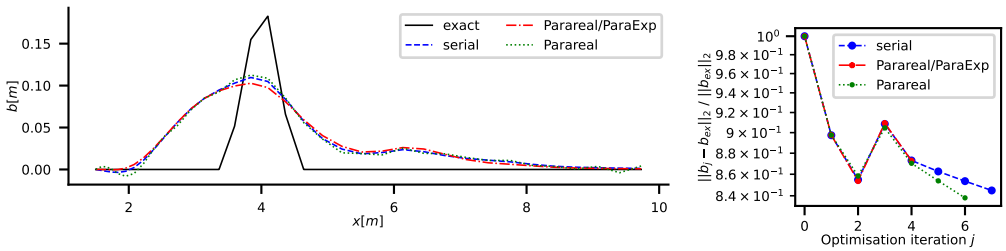

Figure 5.3: Reconstruction of the bathymetry (left) and relative errors against iteration in L-BFGS (right) for the serial (blue dashed) and PinT optimisation with Parareal for the forward problem and ParaExp for the adjoint problem (red dash-dotted) and Parareal for both problems (green dotted).

same results as can be seen in Figure 5.3. All errors and runtimes are shown in Table 5.1. The reconstruction from the serial optimisation has a relative ℓ^2-error of 84.04%, whereas for the PinT reconstruction with Parareal and ParaExp the relative error is 86.27%. For most computations Parareal converged after only one iteration, but ParaExp needed most of the times 8 iterations, sometimes only two. This means that most PinT solutions had a larger error than the corresponding serial solutions which also changes the descent direction in the optimisation. But apparently it does not change a lot such that L-BFGS takes about the same path for PinT as for the serial run. It seems that this difference gets more significant towards the end of the optimisation and that is why the PinT optimisation stops earlier than the serial one. Using a smaller tolerance for Parareal and ParaExp would increase the accuracy of the solutions and yield a reconstruction that is closer to the serial reconstruction. The reconstruction with Parareal for the forward problem and computing the adjoint solution in serial has almost the same error (84.02%) in the reconstruction as the serial reconstruction. But the optimisation was a bit slower, taking 1649 s versus 1420 s for the serial reconstruction.

As ParaExp performs worse than expected, we additionally compare with a reconstruction where Parareal is also used for the solution of the adjoint equation. Surprisingly, the reconstruction with Parareal for the forward and adjoint problem is not only better than the one with ParaExp, but even better than the serial reconstruction which has a relative ℓ^2-error of 84.04%. Additionally, Parareal for the adjoint needs mostly four iterations, which is fewer than ParaExp needed. The optimisation with Parareal stops one iteration earlier and still has a smaller relative ℓ^2-error of 82.59% in the inferred bathymetry. Apparently, the loss in accuracy yields a better descent direction in that sense that the

reconstruction has a lower error. This is possible if the discretisation error corrects the model error in some sense. The mismatch $\|H(b_{\text{ex}}) - H_{\text{obs}}\|_2$ cannot be zero in general as it is limited by the model error for $H = h(b_{\text{ex}}) + b_{\text{ex}}$, where $h(b_{\text{ex}})$ is the exact solution of the SWE. If $\hat{h}$ denotes the numerical solution of the SWE we can write $h(b_{\text{ex}}) = \hat{h} + e$ for a discretisation error e. Let ξ be the model error. Then $H(b_{\text{ex}}) - H_{\text{obs}} = \hat{h} + e + b_{\text{ex}} + \xi$ and in a lucky case the discretisation error e could correct some of the model error ξ.

When looking at the runtimes, in Figure 5.3 we obtain speed-up when using Parareal for both the forward and adjoint problem. The PinT reconstruction with only Parareal which took about 1067 s was faster than the serial reconstruction that had a runtime of 1420 s. This seems like Parareal can be very useful in the optimisation. But this is only true if the error of a serial reconstruction using the time step from the coarse propagator is notably larger. That is why we perform another reconstruction with coarser time steps. Let $\delta t = 0.05$ be the time step for the serial optimisation and the fine propagator and $\Delta t = 0.1$ be the coarse time step. The corresponding reconstructions are shown in Figure 5.4 on the left. The serial reconstruction has a relative error of 84.94% which is slightly larger than the error for the serial run with finer time step in Figure 5.3. But the run time is significantly shorter with only 52 s. Thus, looking at the results, the coarser time step is definitely to be preferred. The reconstruction using Parareal forward and backward with the coarser time steps has a similar error than the other reconstructions, that is 85.56%. Unfortunately, we do not get any speed-up in this case. The total runtime for

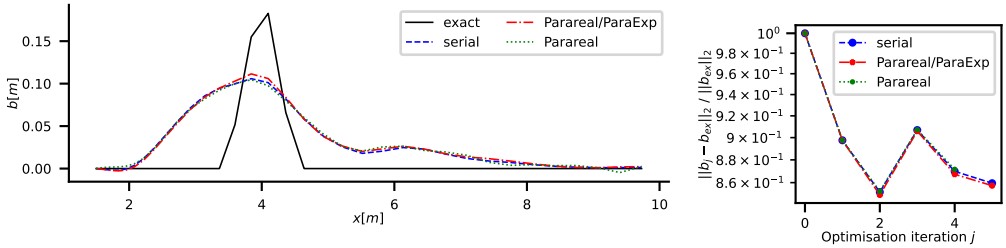

Figure 5.4: Reconstruction of the bathymetry (left) and relative errors against iteration in L-BFGS (right) for the serial reconstruction (blue dashed) and the PinT optimisation with Parareal for the forward and ParaExp for the adjoint problem (red dash-dotted) and Parareal for both forward and adjoint problem (green dotted) for a coarser time step.

the optimisation using Parareal is about 99 s. The reason that we have no speed-up here is that solving the adjoint needs mostly 7 Parareal iterations and a lot of time was lost in the step size control. At the end, no step size was found such that L-BFGS stopped after five iterations already. Consequently, if it is known that a time step of 0.05 s suffices for reconstructing the bathymetry, it makes not much sense using Parareal for both PDEs. Parareal for the forward problem and ParaExp for the adjoint performs a bit better here with a relative error of 83.66% and a runtime of 65 s, even though ParaExp always needed eight iterations to converge. The best approach here is Parareal for the forward equation and solving the adjoint in serial. The corresponding reconstruction has a relative error of 83.66% and the shortest runtime of all four cases, that is 47 s.

Discretisation					Iterations		
δt	Forward	Adjoint	ℓ^2-error	RT [s]	L-BFGS	PinT fwd.	PinT adj.
0.001	serial	serial	84.04%	1420	8	-	-
	Parareal	serial	84.02%	1649	8	1.27	-
	Parareal	ParaExp	86.27%	2988	5	1.07	6.43
	Parareal	Parareal	82.59%	1067	7	1.45	3.21
0.05	serial	serial	84.94%	52	6	-	-
	Parareal	serial	83.66%	47	6	2.35	-
	Parareal	ParaExp	83.66%	65	6	2.35	8.00
	Parareal	Parareal	85.56%	99	5	2.68	5.89

Table 5.1: Relative ℓ^2-errors, runtime (RT), number of L-BFGS iterations and average numbers of iterations for the PinT methods.

Table 5.1 shows average iteration numbers for the PinT methods within the optimisation. Parareal for the forward problem converges relatively good, but not for the adjoint. In terms of iterations, ParaExp performs worse than Parareal, but for the adjoint Parareal convergence is also not very good in the case $\delta t = 0.05$, where the average number of Parareal iterations is 5.89. Hence, the efficient parallelisation of the adjoint problem remains the main challenge here. This might require the development of new methods or improvement of existing approaches, which is left for future work. A possible approach could be the warm start, i.e. to use the adjoint from the previous optimisation iteration as an initial guess for ParaExp.

Chapter 6

Conclusions

We have derived the adjoint problems and gradients for nonlinear SWE with periodic and time-dependent Dirichlet boundary conditions. For serial optimisations in Chapter 3 we have seen that the bathymetry can be reconstructed with relative ℓ^2-errors down to less than 1% using the gradient descent method or L-BFGS. It could be observed that noise in the data can lead to artefacts in the reconstructions, but the hill that is to be inferred was still positioned correctly in the reconstruction. The same is true when using fewer observation points. Though the bathymetry reconstruction in a wave flume from experimental data is even more difficult, the maximum of the reconstructed hill was at the correct position. This is even true when using the measurement data from only two sensors. Literature on the 'first optimise, then discretise'-approach (FOTD) for bathymetry reconstruction with nonlinear SWE has so far not been published except in Angel et al. [20]. The findings imply that our method is fairly robust. A direct comparison with 'first discretise, then optimise' should be subject to future research to investigate whether the derivation of adjoint equations and gradient needed for FOTD is worth the effort. Besides, reconstructions from the experimental data could possibly be improved by using Euler equations instead of SWE to describe the movement of the water in the wave flume.

In Chapter 4 we have observed bad convergence of Parareal for the forward and adjoint SWE as it has often be observed for hyperbolic problems. It is possible to 'cheat' with a small time step for coarse and fine propagator and obtain early convergence of Parareal [79]. But when using spatial coarsening even in this case Parareal converges very slowly. In addition, differences in the Parareal errors for different orders of interpolation were observed for some numerical examples. This was already shown in Ruprecht [81] for the linear advection equation and a different type of discretisation. However, for SWE we have seen that these differences are very small. As a suitable method for hyperbolic problems we have investigated Asymptotic Parareal. The number of needed iterations for Asymptotic Parareal were compared to Parareal for the modulation equation and the original SWE, which has not been done before. The convergence of Asymptotic Parareal proved to be much better than for Parareal for SWE in some cases, and even using the modulation equation can already lead to earlier Parareal convergence. Speed-up is still to be investigated. Theoretically, the computation of the averaging integral can be parallelised and the method could be implemented in a more efficient way than it was done for this thesis. This is left for future work. In order to find a more appropriate method to solve the adjoint problem we applied nonlinear ParaExp. The hope for good convergence

was justified by the linearity of the adjoint equation and the fact that ParaExp can work very well for linear hyperbolic problems. Nevertheless, the nonlinear ParaExp converged worse than expected when applied on the adjoint SWE.

Chapter 5 showed that the novel PinT bathymetry reconstruction works well despite additional errors for the forward and adjoint problem cause by the time-parallelisation. Thus, an optimisation using Asymptotic Parareal could be a good option in cases where convergence of vanilla Parareal is poor. Within the optimisation, the latter often converged surprisingly good. A prediction of how well Parareal converges would be very helpful, such that we could use Asymptotic Parareal instead in cases of late Parareal convergence. For the adjoint problem there is still a more suitable method to be found. We observed speed-up for the optimisation with Parareal for the forward and adjoint wave flume problem when using a 'too' fine time step. But a larger time step is also sufficient to obtain a reconstruction of similar quality and for this case there could be no speed-up observed. It could still be advantageous optimise using Parareal. Because we use FOTD, even solving only the forward problem parallel in time and the adjoint in serial would be possible and could speed up the optimisation. In addition, there are many other possibilities for research. The FOTD approach gives a lot of flexibility as the forward and adjoint problem can be discretised independently. Besides, one could use fewer Parareal or ParaExp iterations at the beginning of the optimisation when less accuracy is needed for the descent direction. For ParaExp, one could even try using the linear ParaExp for the adjoint problem and taking e.g. the average of u and h instead of the time-dependent variables that occur in the adjoint equation. Furthermore, one could decrease the time step later in the optimisation to obtain better reconstructions.

Bibliography

[1] A. Rabinovich, "Twenty-Seven Years of Progress in the Science of Meteorological Tsunamis Following the 1992 Daytona Beach Event," *Pure and Applied Geophysics*, vol. 177, pp. 1193–1230, 2020.

[2] B. Sanders and N. Katopodes, "Adjoint Sensitivity Analysis for Shallow-Water Wave Control," *Journal of Engineering Mechanics-asce - J ENG MECH-ASCE*, vol. 126, 09 2000.

[3] T. Sagawa, Y. Yamashita, T. Okumura, and T. Yamanokuchi, "Satellite derived bathymetry using machine learning and multi-temporal satellite images," *Remote Sensing*, vol. 11, no. 10, p. 1155, 2019.

[4] M. Ashphaq, P. K. Srivastava, and D. Mitra, "Review of near-shore satellite derived bathymetry: Classification and account of five decades of coastal bathymetry research," *Journal of Ocean Engineering and Science*, vol. 6, no. 4, pp. 340–359, 2021.

[5] W. H. F. Smith and D. T. Sandwell, "Conventional bathymetry, bathymetry from space, and geodetic altimetry," *Oceanography*, vol. 17, no. 1, pp. 8–23, 2004.

[6] E. Evagorou, A. Argyriou, N. Papadopoulos, C. Mettas, G. Alexandrakis, and D. Hadjimitsis, "Evaluation of satellite-derived bathymetry from high and medium-resolution sensors using empirical methods," *Remote Sensing*, vol. 14, no. 3, p. 772, 2022.

[7] M. Hinze, R. Pinnau, M. Ulbrich, and S. Ulbrich, *Optimization with PDE Constraints*. Springer-Verlag GmbH, 2009.

[8] F. Tröltzsch, *Optimale Steuerung partieller Differentialgleichungen*, vol. 2. Springer, 2005.

[9] P. Schäfer Aguilar, *Numerical approximation of optimal control problems for hyperbolic conservation laws*. PhD thesis, Technische Universität Darmstadt, Darmstadt, 2021.

[10] A. Bressan, "Hyperbolic systems of conservation laws," *Oxford Lecture Series in Mathematics and its applications*, vol. 20, 2000.

[11] A. Bressan and W. Shen, "Optimality conditions for solutions to hyperbolic balance laws," *Control Methods in PDE – Dynamical Systems*, p. 129–152, Jan. 2007.

[12] S. Pfaff and S. Ulbrich, "Optimal boundary control of nonlinear hyperbolic conservation laws with switched boundary data," *SIAM Journal on Control and Optimization*, vol. 53, no. 3, pp. 1250–1277, 2015.

[13] S. Ulbrich, "On the existence and approximation of solutions for the optimal control of nonlinear hyperbolic conservation laws," in *Optimal Control of Partial Differential Equations: International Conference in Chemnitz, Germany, April 20-25, 1998*, pp. 287–299, Springer, 1999.

[14] S. Ulbrich, "A sensitivity and adjoint calculus for discontinuous solutions of hyperbolic conservation laws with source terms," *SIAM journal on control and optimization*, vol. 41, no. 3, pp. 740–797, 2002.

[15] H. Hajduk, D. Kuzmin, and V. Aizinger, "Bathymetry Reconstruction Using Inverse ShallowWater Models: Finite Element Discretization and Regularization," in *Lecture Notes in Computational Science and Engineering*, p. 223–230, Springer International Publishing, 2020.

[16] J. Monnier, F. Couderc, D. Dartus, K. Larnier, R. Madec, and J.-P. Vila, "Inverse algorithms for 2D shallow water equations in presence of wet dry fronts: Application to flood plain dynamics," *Advances in Water Resources*, vol. 97, pp. 11–24, 2016.

[17] A. Gessese and M. Sellier, "A Direct Solution Approach to the Inverse Shallow-Water Problem," *Mathematical Problems in Engineering*, vol. 2012, p. 1–18, 2012.

[18] V. Vasan and B. Deconinck, "The inverse water wave problem of bathymetry detection," *Journal of Fluid Mechanics*, vol. 714, p. 562–590, 2013.

[19] V. Vasan, Manisha, and D. Auroux, "Ocean-depth measurement using shallow-water wave models," *Studies in Applied Mathematics*, vol. 147, no. 4, pp. 1481–1518, 2021.

[20] J. Angel, J. Behrens, S. Götschel, M. Hollm, D. Ruprecht, and R. Seifried, "Bathymetry reconstruction from experimental data using PDE-constrained optimisation," *Computers & Fluids*, vol. 278, p. 106321, 2024.

[21] J.-L. Lions, Y. Maday, and G. Turinici, "A "parareal" in time discretization of PDE's," *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics*, vol. 332, pp. 661–668, 2001.

[22] M. J. Gander, "Analysis of the Parareal Algorithm Applied to Hyperbolic Problems using Characteristics," *Bol. Soc. Esp. Mat. Apl.*, vol. 42, pp. 21–35, 2008.

[23] A. Nielsen, G. Brunner, and J. Hesthaven, "Communication-aware adaptive Parareal with application to a nonlinear hyperbolic system of partial differential equations," *Journal of Computational Physics*, vol. 371, 2018.

[24] J. G. Caldas Steinstraesser, *Coupling large and small scale shallow water models with porosity in the presence of anisotropy.* PhD thesis, Université de Montpellier, 2021.

[25] T. Haut and B. Wingate, "An Asymptotic Parallel-in-Time Method for Highly Oscillatory PDEs," *SIAM Journal on Scientific Computing*, vol. 36, 03 2014.

[26] H. Yamazaki, C. J. Cotter, and B. A. Wingate, "Time-parallel integration and phase averaging for the nonlinear shallow-water equations on the sphere," *Quarterly Journal of the Royal Meteorological Society*, July 2023.

[27] A. G. Peddle, T. Haut, and B. Wingate, "Parareal Convergence for Oscillatory PDEs with Finite Time-Scale Separation," *SIAM Journal on Scientific Computing*, vol. 41, no. 6, pp. A3476–A3497, 2019.

[28] T. S. Haut, T. Babb, P. G. Martinsson, and B. A. Wingate, "A high-order time-parallel scheme for solving wave propagation problems via the direct construction of an approximate time-evolution operator," *IMA Journal of Numerical Analysis*, vol. 36, no. 2, pp. 688–716, 2016.

[29] F. P. Hamon, M. Schreiber, and M. L. Minion, "Parallel-in-time multi-level integration of the shallow-water equations on the rotating sphere," *Journal of Computational Physics*, vol. 407, p. 109210, Apr. 2020.

[30] M. Schreiber, N. Schaeffer, and R. Loft, "Exponential integrators with parallel-in-time rational approximations for shallow-water equations on the rotating sphere," *Parallel Computing*, 2019.

[31] M. Schreiber and R. Loft, "A parallel time integrator for solving the linearized shallow water equations on the rotating sphere," *Numerical Linear Algebra with Applications*, vol. 26, no. 2, p. e2220, 2019.

[32] M. Schreiber and R. Loft, "A parallel time integrator for solving the linearized shallow water equations on the rotating sphere," *Numerical Linear Algebra with Applications*, 2018.

[33] J. G. Caldas Steinstraesser, P. da Silva Peixoto, and M. Schreiber, "Parallel-in-time integration of the shallow water equations on the rotating sphere using Parareal and MGRIT," *Journal of Computational Physics*, vol. 496, p. 112591, 2024.

[34] M. J. Gander, S. Güttel, and M. Petcu, "A nonlinear ParaExp algorithm," in *Lecture Notes in Computational Science and Engineering*, pp. 261–270, Springer International Publishing, 2018.

[35] M. J. Gander and S. Güttel, "PARAEXP: A Parallel Integrator for Linear Initial-Value Problems," *SIAM Journal on Scientific Computing*, vol. 35, no. 2, pp. C123–C142, 2013.

[36] J. Hahne, B. Polenz, I. Kulchytska-Ruchka, S. Friedhoff, S. Ulbrich, and S. Schöps, "Parallel-in-time optimization of induction motors," *Journal of Mathematics in Industry*, vol. 13, June 2023.

[37] M. Heinkenschloss and N. J. Kroeger, "A new diagonalization based method for parallel-in-time solution of linear-quadratic optimal control problems," *ESAIM: Control, Optimisation and Calculus of Variations*, vol. 30, p. 62, 2024.

[38] P. Parpas and C. Muir, "Predict globally, correct locally: Parallel-in-time optimization of neural networks," *Automatica*, vol. 171, p. 111976, Jan. 2025.

[39] B. Heinzelreiter and J. W. Pearson, "Diagonalization-Based Parallel-in-Time Preconditioners for Instationary Fluid Flow Control Problems." Unpublished, 2024.

[40] S. Ulbrich, *7. Generalized SQP Methods with "Parareal" Time-Domain Decomposition for Time-Dependent PDE-Constrained Optimization*, pp. 145–168. SIAM, 2007.

[41] S. Ulbrich, *Preconditioners Based on "Parareal" Time-Domain Decomposition for Time-Dependent PDE-Constrained Optimization*, pp. 203–232. Cham: Springer International Publishing, 2015.

[42] S. Götschel and M. L. Minion, "An efficient parallel-in-time method for optimization with parabolic PDEs," *SIAM Journal on Scientific Computing*, vol. 41, pp. C603–C626, Jan. 2019.

[43] M. Appel and J. Alexandersen, "One-shot parareal approach for topology optimisation of transient heat flow." Unpublished, 2024.

[44] C. Skene, M. Eggl, and P. Schmid, "A parallel-in-time approach for accelerating direct-adjoint studies," *Journal of Computational Physics*, vol. 429, p. 110033, Mar. 2021.

[45] S. Costanzo, T. Sayadi, M. F. de Pando, P. Schmid, and P. Frey, "Parallel-in-time adjoint-based optimization – application to unsteady incompressible flows," *Journal of Computational Physics*, p. 111664, Oct. 2022.

[46] M. D. Gunzburger, *Perspectives in Flow Control and Optimization*. Society for Industrial and Applied Mathematics, 2002.

[47] J. Nocedal and S. J. Wright, *Numerical Optimization*. Springer New York, NY, 2006.

[48] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical programming*, vol. 45, no. 1, pp. 503–528, 1989.

[49] P. E. Farrell, D. A. Ham, S. W. Funke, and M. E. Rognes, "Automated Derivation of the Adjoint of High-Level Transient Finite Element Programs," *SIAM Journal on Scientific Computing*, vol. 35, no. 4, p. C369–C393, 2013.

[50] E. F. Toro, *Computational Algorithms for Shallow Water Equations*. Springer Cham, 2024.

[51] B. Le Méhauté, *An introduction to hydrodynamics and water waves*. Springer Science & Business Media, 1976.

[52] R. J. LeVeque, *Finite Volume Methods for Hyperbolic Problems*. Cambridge Texts in Applied Mathematics, Cambridge University Press, 2002.

[53] V. Desveaux and A. Masset, "A fully well-balanced scheme for shallow water equations with Coriolis force," *Communications in Mathematical Sciences*, vol. 20, no. 7, pp. 1875–1900, 2022.

[54] N. Matckevich and L. Chubarov, "Exact solutions to shallow water equations for a water oscillation problem in an idealized basin and their use in verifying some numerical algorithms," *Numerical Analysis and Applications*, vol. 12, pp. 234–250, July 2019.

[55] T. Dyakonova and A. Khoperskov, "Bottom friction models for shallow water equations: Manning's roughness coefficient and small-scale bottom heterogeneity," *Journal of Physics: Conference Series*, vol. 973, p. 012032, Mar. 2018.

[56] K. J. Burns, G. M. Vasil, J. S. Oishi, D. Lecoanet, and B. P. Brown, "Dedalus: A flexible framework for numerical simulations with spectral methods," *Physical Review Research*, vol. 2, no. 2, p. 023068, 2020.

[57] J. P. Boyd, *Chebyshev and Fourier spectral methods*. Dover Publications, 2001.

[58] E. Hairer, S. P. Nørsett, G. Wanner, *et al.*, "Solving ordinary differential equations i [electronic resource]: Nonstiff problems," 2000.

[59] J. C. Butcher, "Coefficients for the study of Runge-Kutta integration processes," *Journal of the Australian Mathematical Society*, vol. 3, no. 2, p. 185–201, 1963.

[60] G. Chacón, H. Rafeiro, and J. C. Vallejo, *Functional Analysis*. Berlin, Boston: De Gruyter, 2017.

[61] L. C. Evans, *Partial differential equations*, vol. 19. American Mathematical Society, 1998.

[62] J. Angel, J. Behrens, S. Götschel, M. Hollm, D. Ruprecht, and R. Seifried, "Data artefact: Bathymetry reconstruction from experimental data with PDE-constrained optimisation," 2024.

[63] J. Angel, D. Ruprecht, and S. Götschel, "Waterchannel," Apr. 2025.

[64] H.-O. Georgii, *Introduction to Probability and Statistics*. Berlin, Boston: De Gruyter, 2013.

[65] L. Dümbgen, *Einführung in die Statistik*. Springer, 2016.

[66] J. Kaipio and E. Somersalo, "Statistical inverse problems: Discretization, model reduction and inverse crimes," *Journal of Computational and Applied Mathematics*, vol. 198, no. 2, pp. 493–504, 2007. Special Issue: Applied Computational Inverse Problems.

[67] "Tabelle der t-Verteilung." `https://www.statistik.tu-dortmund.de/fileadmin/user_upload/Lehrstuehle/Oekonometrie/Lehre/WiSoOekoSS17/tabelletV.pdf`, 2017. Accessed 26 March 2024.

[68] Z. Hao, F. Chen, X. Jia, X. Cai, C. Yang, Y. Du, and F. Ling, "GRDL: A New Global Reservoir Area-Storage-Depth Data Set Derived Through Deep Learning-Based Bathymetry Reconstruction," *Water Resources Research*, vol. 60, no. 1, p. e2023WR035781, 2024.

[69] O. Delestre, C. Lucas, P.-A. Ksinant, F. Darboux, C. Laguerre, T.-N.-T. Vo, F. James, and S. Cordier, "SWASHES: a compilation of shallow water analytic solutions for hydraulic and environmental studies," *International Journal for Numerical Methods in Fluids*, vol. 72, no. 3, pp. 269–300, 2013.

[70] B. W. Ong and J. B. Schroder, "Applications of time parallelization," *Computing and Visualization in Science*, vol. 23, pp. 1–15, 2020.

[71] M. J. Gander, "50 years of time parallel time integration," in *Multiple Shooting and Time Domain Decomposition Methods: MuS-TDD, Heidelberg, May 6-8, 2013*, pp. 69–113, Springer, 2015.

[72] M. J. Gander and S. Vandewalle, "Analysis of the parareal time-parallel time-integration method," *SIAM Journal on Scientific Computing*, vol. 29, no. 2, p. 556–578, 2007.

[73] C. Farhat and M. Chandesris, "Time-decomposed parallel time-integrators: theory and feasibility studies for fluid, structure, and fluid-structure applications," *International Journal for Numerical Methods in Engineering*, vol. 58, no. 9, pp. 1397–1434, 2003.

[74] L. Baffico, S. Bernard, Y. Maday, G. Turinici, and G. Zérah, "Parallel-in-time molecular-dynamics simulations," *Phys. Rev. E*, vol. 66, p. 057701, 2002.

[75] M. J. Gander and T. Lunet, *Time Parallel Time Integration*, ch. 2, p. 39–84. SIAM, 2024.

[76] G. A. Staff and E. M. Rønquist, "Stability of the parareal algorithm," in *Domain Decomposition Methods in Science and Engineering* (R. Kornhuber and et al., eds.), vol. 40 of *Lecture Notes in Computational Science and Engineering*, (Berlin), pp. 449–456, Springer, 2005.

[77] E. Aubanel, "Scheduling of Tasks in the Parareal Algorithm," *Parallel Computing*, vol. 37, pp. 172–182, 2011.

[78] M. Iizuka and K. Ono, "Influence of the phase accuracy of the coarse solver calculation on the convergence of the parareal method iteration for hyperbolic PDEs," *Computing and Visualization in Science*, 2018.

[79] S. Götschel, M. Minion, D. Ruprecht, and R. Speck, "Twelve ways to fool the masses when giving parallel-in-time results," in *Springer Proceedings in Mathematics & Statistics*, pp. 81–94, Springer International Publishing, 2021.

[80] J. Angel, S. Götschel, and D. Ruprecht, "Impact of spatial coarsening on parareal convergence for the linear advection equation," 2024. Accepted for publication in Scientific Computing and Software - Go20 CSCS 2024.

[81] D. Ruprecht, "Convergence of Parareal with spatial coarsening," *PAMM*, vol. 14, no. 1, pp. 1031–1034, 2014.

[82] P. F. Fischer, F. Hecht, and Y. Maday, "A parareal in time semi-implicit approximation of the Navier-Stokes equations," in *Domain Decomposition Methods in Science and Engineering* (R. Kornhuber and et al., eds.), vol. 40 of *Lecture Notes in Computational Science and Engineering*, (Berlin), pp. 433–440, Springer, 2005.

[83] B. Philippi and T. Slawig, "A Micro-Macro Parareal Implementation for the Ocean-Circulation Model FESOM2," 2023. Unpublished.

[84] B. Philippi, *Investigation of Parallel-In-Time Algorithms for Fluid Flow Problems of Varying Complexity: From Burger's Equation to an Operational Ocean Circulation Model*. PhD thesis, Kiel, 2024.

[85] J. Rosemeier, T. Haut, and B. Wingate, "Multilevel Parareal Algorithm with Averaging for Oscillatory Problems," *SIAM Journal on Scientific Computing*, vol. 46, no. 4, pp. A2709–A2736, 2024.

[86] A. Peddle, *Components of Nonlinear Oscillation and Optimal Averaging for Stiff PDEs*. PhD thesis, Exeter, 2018.

[87] E. Gallopoulos and Y. Saad, "On the parallel solution of parabolic equations," in *Proceedings of the 3rd international conference on Supercomputing*, pp. 17–28, 1989.