

Tobias Schwedes
David A. Ham
Simon W. Funke
Matthew D. Piggott

Mesh dependence in PDE- constrained optimisation

An application in tidal turbine array layouts

July 14, 2017

Springer

Preface

Implementations of optimisation methods often rely on the assumption that the underlying problem is posed in the Euclidean space. In many applications however, the optimisation control is a function over some Hilbert space rather than being a number or a vector in $\mathbb{R}^n$. Consideration of this point is important since the choice of the underlying Hilbert space determines the notion of angles and distances, induced by its associated inner product. Since such quantitative concepts are typically used by algorithms, for example to compute a search direction or to decide whether convergence criteria have been met, it seems logical that the Euclidean assumption might lead to problems in the performance of these algorithms in cases where the control is not Euclidean.

One example of this is in the case of gradient-based methods, that is methods that use the gradient of an objective functional. It is a consequence of the Riesz representation theorem that the representation of the derivative of a functional depends on the underlying inner product. Methods which do not respect the inherent or natural control space and its associated inner product become dependent on the underlying discretisation of the domain – with detrimental consequences for the algorithm's performance. The exploration and analysis of this phenomenon, that we term mesh dependence, in theory and numerical simulations represent the core of this work.

Scenarios whose controls are functions lying in some Hilbert space regularly arise when solving optimisation problems for a physical system which is subject to constraints in terms of partial differential equations (PDEs). A classical example of such an optimisation problem is: what is the optimal shape of an aeroplane in terms of maximum fuel efficiency or lift-to-drag ratio under the influence of air flow around the body? It seems natural to represent the shape by a continuous function. Applications of PDE-constrained optimisation appear in a wide range of other fields of industry as well as in medical research and in economics.

This work provides a discussion and analysis which quantifies the performance loss induced by assuming the Euclidean inner product for gradient-based methods for PDE-constrained optimisation problems with continuous controls. The authors also hope to promote best practice for the development and usage of optimisation

tools in the context of problems posed in Hilbert spaces as a consequence on their analysis.

The intended audience of this book mainly consists of graduate students and researchers in computational science, mathematics, engineering and physics with an interest in PDE-constrained optimisation and their real-world applications. The application oriented character of the book may also make it of interest to users of optimisation tools and practitioners that work with PDE-constrained problems. Developers of optimisation tools or users interested in best practice for mesh-independent methods are a further potential audience.

The reader is provided with a gentle introduction into the field of PDE-constrained optimisation in Chap. 1. This includes an introduction to solving PDEs using the finite element method, to adjoints used for the effective computation of derivatives and an overview of some popular gradient-based optimisation methods. In addition, this chapter introduces the relationship between primal and dual space, which underpins the analysis in the subsequent chapter. In Chap. 2 the impact of mesh-dependence is treated theoretically for a simple optimisation problem, as well as numerically for a slightly more advanced generic PDE-constrained optimisation problem. In Chap. 3, a case study considering the optimisation of tidal turbine array layouts in the Pentland Firth, Scotland, demonstrates the relevance of the analysis from the preceding chapter. This further represents a valuable example of a real-world application of optimisation methods used in the context of PDE-constrained problems.

Tobias Schwedes, David A. Ham, Simon W. Funke, Matthew D. Piggott
March 2017

Contents

1	Introduction to PDE-constrained optimisation	1
1.1	The Finite element method	3
1.1.1	Poisson's equation	3
1.1.2	Burgers' equation	14
1.2	Hilbert spaces	21
1.2.1	Dual spaces and the Riesz representation theorem	21
1.2.2	Fréchet derivatives	23
1.2.3	Partial derivatives and the chain rule	24
1.3	The relation between primal and dual finite element spaces	24
1.3.1	Example: application of assembled finite element operators	26
1.3.2	The primal and dual norms for L^2 and H^1	26
1.3.3	Convergence criteria for Newton's method	28
1.4	Adjoint and tangent linear equations	29
1.4.1	A finite-dimensional example	30
1.4.2	The infinite dimensional case	32
1.4.3	Higher-order derivatives	34
1.4.4	Example: Adjoint Poisson's equation	35
1.4.5	Example: Adjoint Burgers' equation	36
1.4.6	Taylor tests for adjoint implementations	37
1.5	Optimisation methods	38
1.5.1	Steepest descent method	38
1.5.2	Inexact line search and the Wolfe conditions	39
1.5.3	Line search Newton-CG	40
1.5.4	BFGS	41
1.5.5	Primal log-barrier interior point method	47
1.6	Optimal control of the Poisson equation	48
1.7	References	50
2	Mesh dependence in PDE-constrained optimisation	53
2.1	Literature overview	54
2.2	A generic optimisation problem	56

2.2.1	Formulation	56
2.2.2	Finite element discretisation	57
2.2.3	Iteration count using L^2 inner product	58
2.2.4	Iteration count estimate using the ℓ^2 inner product	60
2.2.5	Numerical experiments	66
2.3	Mesh dependence in the optimal control of the Poisson equation . . .	67
2.3.1	Experiments on deterministically refined meshes	68
2.3.2	Experiments on randomly refined meshes	70
2.4	Final remark	75
2.5	References	75
3	An application: Optimising the layout of tidal turbine arrays	79
3.1	Introduction	79
3.2	Problem formulation	81
3.2.1	Design parameters and box constraints	81
3.2.2	The PDE constraint	82
3.2.3	The class of turbine density functions	83
3.2.4	Relationship between turbine density and seabed friction . . .	84
3.2.5	The functional of interest	84
3.3	Shallow water equations	85
3.3.1	Physical principles	86
3.3.2	Navier-Stokes equations	87
3.3.3	Shallow water equations	88
3.4	Aspects of numerical solution	91
3.4.1	Solving the shallow water equations	91
3.4.2	Adjoint equations	92
3.4.3	Implementation	93
3.5	Mesh dependence in tidal turbine array layout optimisation	94
3.5.1	Mesh refinement	97
3.5.2	Numerical experiments	98
3.6	Tidal resources assessment in the Pentland Firth, Scotland	101
3.7	References	106
Index		109

Chapter 1

Introduction to PDE-constrained optimisation

The use of computational models based on the numerical solution of partial differential equations (PDEs) to simulate physical processes is a powerful complement to physical experiments. Simulations can be undertaken to consider scenarios for which experiments are impossible, such as climate physics or the dynamics of black holes and galaxies. However, a particular strength of simulations is in answering the inverse problems which pervade science and engineering: to which inputs of my system are my output results most sensitive? Is this system stable or unstable? Which configuration or design produces the best outcome? This last question can be rephrased as the core problem of PDE-constrained optimisation: which input parameters minimise some output measurement, given the constraint that the system state must be a solution to a given PDE.

In engineering, the need for optimisation emerges regularly when seeking improved designs. A classical example is concerned with a key element in aeronautical engineering:

What is the optimal shape of an aerofoil?

In the pioneering work of [20], this question is considered as an optimisation problem governed by the Euler equations for compressible flow. A similar design problem is investigated in Chap. 3 of this work. It considers the situation where a large number of tidal turbines are to be deployed within an array in order to extract energy from tidal currents. The investigated optimisation problem is:

What is the optimal spatial distribution of the tidal turbines?

A common task in the geosciences is to determine unknown parameters such that a computer model best reproduces existing measurements [13]. For example, satellite imagery can provide detailed surface information about the ocean, but in general little is known about its interior [35]. The ECCO2 project incorporates most oceanographic and meteorologic measurements available to date into an ocean simulator to create an accurate description of the time-evolving state of the ocean [25]. Here, the problem formulation is:

What is the state of the ocean at the beginning of the observation interval that minimises the difference between simulation and measurements?

Despite the variety of these tasks, they can all be formulated as optimisation problems constrained by PDEs. In this work we will employ the finite element method to produce numerical solutions, so the PDEs will be stated in weak, or variational, form. The generic form of a PDE-constrained optimisation problem is then:

$$\begin{aligned} & \min_{\substack{u \in V \\ m \in M}} J(u, m) \\ & \text{subject to } F(u, m; v) = 0 \quad \forall v \in V', \end{aligned} \tag{1.1}$$

where M is the space of all possible values of optimisation parameters, and V, V' are suitable function spaces – indeed in most cases $V = V'$. Further, $J : M \otimes V \rightarrow \mathbb{R}$ is the functional of interest that is to be minimised, and $F(u, m; v) = 0$ is a PDE parameterised by m with solution u for all values of the test function v .

As an example, reconsider the optimal design of the aerofoil. In this case, the parameter m contains the parameterised shape of the aerofoil, for example the coefficients of its Bézier curve. The physics are described by the Euler equations, which yield the pressure and velocity for a given aerofoil design. Finally, the functional of interest J computes some performance metric for the aerofoil, for example by evaluating the drag-lift ratio. The issue where a performance metric is to be maximised while formulation (1.1) actually describes a minimisation problem is easily resolved by seeking to minimise the negative of J instead.

The necessary ingredients for solving such optimisation problems numerically are the following: first, we need to be able to approximate the solution u of $F(u, m; v) = 0$, the underlying PDE. To that end, the finite element method is introduced in Sect. 1.1. There are a number of factors motivating the adoption of the finite element method in this work rather than any of the other commonly employed numerical methods for PDEs. The finite element method combines geometric flexibility, the availability of a wide range of discrete function spaces enabling practitioners to choose those with optimal properties for a given problem, and a particularly elegant mathematical representation which facilitates rigorous analysis. In particular, the finite element method makes the role of particular discrete function spaces, and their dual spaces, explicit in the discretisation. As will become apparent, the relationship between these function spaces and mesh dependence is critical. The choice of the finite element method also presents the opportunity to present the derivation of the adjoint PDE in Sect. 1.4 in the setting of a variational problem. This is in contrast to the common practice in works such as [17] of deriving the adjoint of the strong form of the PDE. It is hoped that the variational derivation presented here will be particularly useful to those who employ the finite element method. However, it is important to note that the issues considered in this work carry over to other discretisation approaches: it is not possible to avoid mesh dependence simply by switching to a finite difference or finite volume formulation.

Second, we must be able to compute derivatives of J with respect to the parameters m . Many problems arising in PDE-constrained optimisation have the common

feature that the solution of the governing PDE is computationally demanding and that a large number of parameters need to be optimised. Naïve differentiation methods scale linearly with the number of parameters, which makes them computationally too costly for many applications. However, these drawbacks can be overcome through use of the adjoint approach for computing derivatives, which is introduced in Sect. 1.4.

Finally, suitable optimisation methods must be employed. In many applications the situation arises in which the parameters m denote a function rather than a set of numbers. For example, the shape of the aerofoil in the above mentioned example can be considered as a function. Optimisation methods that are formulated on abstract spaces, including functions spaces and Euclidean spaces, are introduced in Sect. 1.5.

1.1 The Finite element method

The finite element method approximates the solution of the weak form of a PDE by replacing the infinite-dimensional spaces of solution functions and test functions with finite-dimensional subspaces. This is achieved by decomposing the underlying domain into a collection of subdomains, or cells, on each of which the admissible functions are a defined set of polynomials. Discrete integral operators are evaluated on each cell and the results recombined to form and solve the global problem over the entire domain. In this section the concept of finite elements is introduced using two simple but representative PDEs, namely, Poisson's equation and the time-dependent viscous Burgers equation. The latter is nonlinear and its numerical solution additionally requires a discretisation of the temporal dimension. The former is linear and is the classical example used when introducing finite elements. It is therefore the starting point of this section.

1.1.1 Poisson's equation

The Poisson equation,

$$-\Delta u = f \quad \text{on} \quad \Omega, \quad (1.2)$$

is the simplest elliptic partial differential equation and will serve as a prime example for the derivation of the finite element method. Here we will assume that we are in two spatial dimensions and thus

$$\Delta := \frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_2^2}, \quad (1.3)$$

denotes the Laplace operator (or Laplacian) for functions on $\mathbb{R}^2$. Moreover, f is a real-valued source function on the domain $\Omega \subset \mathbb{R}^2$ and u the associated real-valued solution. The solution u also satisfies boundary conditions defined on the boundary $\partial\Omega$ of the domain. For the sake of simplicity we only introduce the two most common boundary conditions given by:

$$\text{Dirichlet-type:} \quad u = g_D \quad \text{on} \quad \partial\Omega_D, \quad (1.4)$$

$$\text{Neumann-type:} \quad \frac{\partial u}{\partial n} = g_N \quad \text{on} \quad \partial\Omega_N, \quad (1.5)$$

where $\partial\Omega_D \cup \partial\Omega_N = \partial\Omega$ and $\partial\Omega_D \cap \partial\Omega_N = \emptyset$. Furthermore, $n = (n_1, n_2)^T$ denotes the unit outward normal vector to $\partial\Omega$ and

$$\frac{\partial u}{\partial n} = \left(\frac{\partial u}{\partial x_1}, \frac{\partial u}{\partial x_2} \right) \cdot (n_1, n_2)^T. \quad (1.6)$$

Thus, a Neumann boundary condition prescribes the derivative of the solution in the direction pointing directly outwards relative to the domain, whereas a Dirichlet boundary condition prescribes the value of the solution itself on the boundary. In the case where $\Omega_D = \emptyset$ we speak of pure Neumann boundary conditions, and if $\Omega_N = \emptyset$ we speak of pure Dirichlet boundary conditions. Poisson's equation 1.2 together with the boundary conditions (1.4, 1.5) is called a boundary value problem.

1.1.1.1 Derivation of weak solutions

A function u that is sufficiently smooth and satisfies (1.2) as well as (1.4, 1.5) is called a classical solution to the boundary value problem. In the case of Dirichlet boundary conditions, a classical solution u must be twice differentiable on Ω and continuous on the closure $\bar{\Omega}$, i.e. $u \in C^2(\Omega) \cap C^0(\bar{\Omega})$. It can be shown analytically that for sufficiently smooth source data f and boundary $\partial\Omega$, a classical solution exists and is unique. However, many problems lack these smoothness or regularity conditions such that the classical formalism becomes unsuitable. This can be the case for non-convex domains; see [9, p.11-13]. Further, consider the case in which f in (1.2) is a discontinuous function. Were there a classical solution u , one would have $f = \Delta u \in C^0(\Omega)$ contradicting the assumption on the discontinuous nature of f .

Discontinuous sources are physically perfectly reasonable. In thermodynamics, the Poisson equation describes the steady-state temperature profile u under some heating source f . If $f = 1$ on one part of the domain, and $f = 0$ on the rest of the domain, this corresponds to a constant heating applied to only part of the domain.

In order to overcome the limitations of classical solutions, we relax the regularity properties required of the solution by adopting an alternative formulation of the boundary value problem. To this end, we suppose that the solution $u : \Omega \rightarrow \mathbb{R}$ belongs to some suitable function space V . We multiply both sides of (1.2) by an arbitrary test function $v \in V'$ from some appropriate set of test functions and integrate

over the whole domain. The problem thus becomes, find $u \in V$ such that

$$-\int_{\Omega} \Delta u(x) v(x) \, dx = \int_{\Omega} f(x) v(x) \, dx \quad \forall v \in V'. \quad (1.7)$$

Applying integration by parts to the left hand side yields

$$\int_{\Omega} \nabla u(x) \cdot \nabla v(x) \, dx - \int_{\partial\Omega} \frac{\partial u}{\partial n}(s) v(s) \, ds = \int_{\Omega} f(x) v(x) \, dx, \quad (1.8)$$

where $\nabla := (\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2})$ denotes the nabla operator. A function u satisfying (1.8) is called a weak solution since it requires less smoothness than a classical one. Indeed, a weak solution u does not need to be twice differentiable. Instead regularity conditions on the spaces V and V' required to make (1.8) well-posed are formulated in terms of the integrability of functions and their first derivatives. One important function space in this context is the space of Lebesgue square-integrable functions

$$L^2(\Omega) := \left\{ u : \Omega \rightarrow \mathbb{R} \mid \int_{\Omega} u^2(x) \, dx < \infty \right\}, \quad (1.9)$$

with the associated inner product

$$\langle u, v \rangle_{L^2(\Omega)} := \int_{\Omega} u(x) v(x) \, dx, \quad (1.10)$$

and norm

$$\|u\|_{L^2(\Omega)} := \left(\langle u, u \rangle_{L^2(\Omega)} \right)^{1/2} = \sqrt{\int_{\Omega} u^2(x) \, dx}. \quad (1.11)$$

Applying pure Neumann conditions to (1.8) yields

$$\int_{\Omega} \nabla u(x) \cdot \nabla v(x) \, dx - \int_{\partial\Omega} g_N(s) v(s) \, ds = \int_{\Omega} f(x) v(x) \, dx. \quad (1.12)$$

Note that any solution to (1.12) is only defined up to a constant. Uniqueness holds if some of the Neumann boundary is replaced by a Dirichlet boundary.

For (1.12) to be well-defined we require every integral term to be finite. For the forcing term to be finite obviously requires $f, v \in L^2(\Omega)$. For the boundary term we additionally require that $g_N \in L^2(\partial\Omega)$ and the so called trace inequality, which relates an integral over the boundary to an integral over the whole domain; see [8, Sect. 1.5.1] for details. For the Laplacian term we note that if $\frac{\partial u}{\partial x_i}, \frac{\partial v}{\partial x_i} \in L^2(\Omega)$ for all i then

$$\left| \int_{\Omega} \nabla u(x) \cdot \nabla v(x) \, dx \right| = \left| \sum_i \int_{\Omega} \frac{\partial u}{\partial x_i}(x) \frac{\partial v}{\partial x_i}(x) \, dx \right| \quad (1.13)$$

$$\leq \sum_i \left| \left\langle \frac{\partial u}{\partial x_i}, \frac{\partial v}{\partial x_i} \right\rangle_{L^2(\Omega)} \right| \quad (1.14)$$

$$< \infty. \quad (1.15)$$

From these considerations, we conclude that a natural space for both the weak solution u of (1.12) as well as the associated test functions v to exist in is the Sobolev space $H^1(\Omega)$ defined by

$$H^1(\Omega) := \left\{ u : \Omega \rightarrow \mathbb{R} \left| u \in L^2(\Omega), \frac{\partial u}{\partial x_i} \in L^2(\Omega) \, \forall i \right. \right\}. \quad (1.16)$$

In summary, while a classical solution of the boundary value problem (1.2) and (1.4, 1.5) for Poisson's equation must be twice differentiable, a weak solution need only be square integrable and have square-integrable first derivatives.

1.1.1.2 Weak solutions for higher-order PDEs

In a manner analogous to that presented above for Poisson's equation, higher-order partial differential equations whose classical solutions require higher-order differentiability (triple or more) can be rewritten in their weak integral form by applying integration by parts. The regularity conditions posed on the (weak) solution thereby reduce again to integrability conditions on the solution itself and derivatives of lower order than in the classical formalism. Consider for example the biharmonic equation

$$\Delta^2 u = f, \quad (1.17)$$

with $\Delta^2 = \sum_{i=1}^2 \sum_{j=1}^2 \partial_i^2 \partial_j^2$. Multiplying (1.17) by a test function, integrating, and applying integration by parts twice naturally leads to integrability conditions on the second derivatives of the solution. A natural choice for the space of solutions is the Sobolev space $H^2(\Omega)$ given by

$$H^2(\Omega) := \left\{ u : \Omega \rightarrow \mathbb{R} \left| u \in L^2(\Omega), \frac{\partial u}{\partial x_i} \in L^2(\Omega), \frac{\partial^2 u}{\partial x_i \partial x_j} \in L^2(\Omega) \, \forall i, j \right. \right\}. \quad (1.18)$$

1.1.1.3 Weak derivatives

The partial derivatives appearing in $H^1(\Omega)$ and $H^2(\Omega)$ are not understood in the usual strong sense. Recall the usual definition

$$\frac{\partial u}{\partial x_i}(x) = \lim_{\varepsilon \rightarrow 0} \frac{u(x + \varepsilon e_i) - u(x)}{\varepsilon}, \quad (1.19)$$

where e_i denotes the unit vector in the x_i direction for $i = 1, 2$. Instead, in a manner analogous to the weak formulation, they are understood in a spatially averaged way with respect to a set of test functions. These derivatives are therefore called weak derivatives, and can be defined for locally integrable functions, i.e. that are integrable over compact sets inside Ω . The space of these functions is denoted by $L^1_{\text{loc}}(\Omega)$. We say that $u \in L^1_{\text{loc}}(\Omega)$ has a weak derivative with respect to x_i , denoted by $D_w^{x_i} u$, if there is a function $w_{x_i} \in L^1_{\text{loc}}(\Omega)$ such that

$$\int_{\Omega} w_{x_i}(x) v(x) dx = - \int_{\Omega} u(x) \frac{\partial v}{\partial x_i}(x) dx \quad \forall v \in C_0^\infty(\Omega), \quad (1.20)$$

where $C_0^\infty(\Omega)$ is the set of smooth functions on Ω with compact support. In that case, we set $D_w^{x_i} u = w_{x_i}$. Definition (1.20) can be easily generalised for higher-order partial derivatives, appearing for instance in $H^2(\Omega)$.

It follows immediately from the integration by parts formula that any differentiable function is also weakly differentiable. Weak derivatives are useful since they allow for important classes of functions to have a (weak) derivative where classically they would not. For instance, $H^1(\Omega)$ contains piecewise linear polynomials, which are an essential class of functions used for finite element approximations. We shall return to this later when finite element spaces are introduced.

1.1.1.4 Weak formulation

In order to solve the generic integral form (1.8) of the boundary value problem with boundary conditions (1.4, 1.5), let us assume that $\int_{\partial\Omega_D} ds > 0$, i.e. the boundary conditions are not purely Neumann. Following [8, Sect. 1.2], we define the solution and test function spaces by

$$H_D^1 := \{u \in H^1(\Omega) \mid u = g_D \text{ on } \partial\Omega_D\}, \quad (1.21)$$

$$H_0^1 := \{v \in H^1(\Omega) \mid v = 0 \text{ on } \partial\Omega_D\}, \quad (1.22)$$

respectively. The Dirichlet boundary conditions are contained in the set H_D^1 of weak solutions while the test function space vanishes over the corresponding part of the boundary. The Neumann boundary conditions do not explicitly restrict the solution or the test functions.

The complete weak formulation of the boundary value problem can now be specified as follows: find $u \in H_D^1(\Omega)$ such that

$$\int_{\Omega} \nabla u(x) \cdot \nabla v(x) dx - \int_{\partial\Omega_N} g_N(s) v(s) ds = \int_{\Omega} f(x) v(x) dx \quad \forall v \in H_0^1(\Omega). \quad (1.23)$$

The reader might reasonably ask what the physical interpretation of a weak solution to a PDE might be in contrast to a classical one. In the classical formalism, the PDE solution is prescribed by the equations of the PDE at every point of the domain and its boundary. However, any PDE is just a mathematical representation that approximates an actual physical phenomenon, whose observations can never be perfectly localised. It is therefore physically more meaningful, and conforms with what is actually observed in an experiment of the real world, to instead consider what happens on average over a small region of space. This is precisely what is described in the weak formulation of a PDE. This can be seen if we take a smooth test function v whose support is a small neighbourhood of some point $x_0 \in \Omega$ and whose integral is equal to one. In this case

$$\int_{\Omega} u(x)v(x)dx \quad (1.24)$$

represents a spatial average of the values of u in a small neighbourhood of x_0 . Further, the solutions to many physical problems contain discontinuities such as shock waves. The weak formulation allows for the existence of these solutions in a mathematically rigorous framework.

1.1.1.5 Galerkin finite element method

The function spaces H_D^1 and H_0^1 from (1.21, 1.22) are of infinite dimension. To see this for H_0^1 suppose $\Omega = (0, \pi)^2 \subset \mathbb{R}^2$, then the set of functions $\{\lambda_k\}_{k \in \mathbb{N}}$ with

$$\lambda_k(x_1, x_2) := \sin(kx_1) \cos\left(kx_2 + \frac{\pi}{2}\right) \quad \forall (x_1, x_2) \in \Omega, \quad (1.25)$$

are linearly independent, and $\lambda_k \in H_0^1(\Omega) \quad \forall k \in \mathbb{N}$. Thus, $H_0^1(\Omega)$ cannot be of finite dimension.

Since computational resources are finite, numerically solving the weak form (1.23) requires a finite-dimensional approximation. Let us therefore choose an appropriate subspace $V_{0,h} \subset H_0^1$ of test functions of dimension $d \in \mathbb{N}$, which we call a finite element space. Any $v_h \in V_{0,h}$ can be represented with respect to a set of basis functions $\{\varphi_1, \dots, \varphi_d\}$ for $V_{0,h}$, which we shall call shape functions, and coefficients $v_{h,1}, \dots, v_{h,d} \in \mathbb{R}$, by

$$v_h = \sum_{i=1}^d v_{h,i} \varphi_i. \quad (1.26)$$

The Dirichlet boundary condition in the solution is interpolated by an additional set of shape functions $\{\varphi_{d+1}, \dots, \varphi_{d+d_D}\}$ and associated coefficients $g_{h,i}$. Thus, the finite element solution $u_h \in V_{D,h}$, where $V_{D,h} \subset H_D^1$ is spanned by the set of basis functions $\{\varphi_{d+1}, \dots, \varphi_{d+d_D}, \varphi_{d+1}, \dots, \varphi_{d+d_D}\}$, can be expressed in a unique way via the coefficients $u_{h,1}, \dots, u_{h,d}$ and $g_{h,d+1}, \dots, g_{h,d+d_D}$ by

$$u_h = \sum_{i=1}^d u_{h,i} \varphi_i + \sum_{i=d+1}^{d+d_D} g_{h,i} \varphi_i. \quad (1.27)$$

Replacing $u \in H_D^1(\Omega)$ and $v \in H_0^1(\Omega)$ in the weak formulation (1.23) by $u_h \in V_{D,h}$ and $v_h \in V_{0,h}$, respectively, yields the finite-dimensional approximation we are looking for, which is commonly termed the Galerkin finite element approximation, i.e. find $u_h \in V_{D,h}$ such that

$$\int_{\Omega} \nabla u_h(x) \cdot \nabla v_h(x) dx - \int_{\partial\Omega_N} g_N(s) v_h(s) ds = \int_{\Omega} f(x) v_h(x) dx \quad \forall v_h \in V_{0,h}. \quad (1.28)$$

Making use of the representations (1.26) and (1.27) of v_h and u_h with respect to their basis functions, respectively, (1.28) is equivalent to

$$\begin{aligned} \sum_{j=1}^d u_{h,j} \int_{\Omega} \nabla \varphi_j \cdot \nabla \varphi_i dx &= \int_{\Omega} \varphi_i f dx + \int_{\partial\Omega_N} \varphi_i g_N ds \\ &\quad - \sum_{j=d+1}^{d+d_D} u_{h,j} \int_{\Omega} \nabla \varphi_j \cdot \nabla \varphi_i dx \quad \forall i = 1, \dots, d. \end{aligned} \quad (1.29)$$

This system of equations can be rewritten in matrix form, which we term the Galerkin system and is given by

$$\mathbf{A} \mathbf{u} = \mathbf{f}, \quad (1.30)$$

where $\mathbf{u} := (u_{h,1}, \dots, u_{h,d})^T$, the entry A_{ij} of the i th row and j th column of $\mathbf{A}$ is given by

$$A_{ij} = \int_{\Omega} \nabla \varphi_j \cdot \nabla \varphi_i dx, \quad (1.31)$$

and the i th component f_i of the vector $\mathbf{f}$ is given by

$$f_i = \int_{\Omega} \varphi_i f dx + \int_{\partial\Omega_N} \varphi_i g_N ds - \sum_{j=d+1}^{d+d_D} u_{h,j} \int_{\Omega} \nabla \varphi_j \cdot \nabla \varphi_i dx. \quad (1.32)$$

The matrix $\mathbf{A}$ is usually referred to as the *stiffness matrix*, while $\mathbf{f}$ is often termed the *load vector*.

Spatially, the domain $\Omega \subset \mathbb{R}^2$ is divided into a mesh of polygons such as triangles or quadrilaterals, or indeed other two-dimensional shapes; these are termed the elements or cells. The finite element spaces are chosen such that the basis functions $\varphi_1, \dots, \varphi_d$ have compact support, i.e. they are each only locally nonzero on the mesh, thereby creating local matrix and vector contributions which are in general far smaller than $\mathbf{A}$ and $\mathbf{f}$. The local matrix and vector quantities can easily be computed

and can then be assembled into the global Galerkin system that models the entire problem.

1.1.1.6 Triangular Lagrange finite elements

For the sake of simplicity and following [8, Sect. 1.3.1], let us assume that the domain Ω is polygonal, i.e. Ω can be tessellated entirely with triangles $\Delta_k \subset \Omega$ for $k = 1, \dots, K$, i.e.

$$\bigcup_{k=1}^K \bar{\Delta}_k = \bar{\Omega}, \quad (1.33)$$

$$\Delta_k \cap \Delta_\ell = \emptyset \quad \forall k \neq \ell. \quad (1.34)$$

We shall call the set $\{\Delta_1, \dots, \Delta_K\}$ a triangulation and denote it by $\mathcal{T}_h$. The edges and vertices of the set of triangles form a mesh, and we shall refer to each triangle as a cell or an element. The finite element spaces we shall consider are the continuous piecewise polynomials of degree p . This means that the space restricted to any cell is the space of polynomials of degree p on that cell, and with the space constrained to be continuous along the interior edges and at every vertex of the mesh. The natural basis functions for this space are polynomials of degree p which take the value 1 at a particular nodal point and vanish at the nodal points for all other basis functions. The simplest choice for such basis functions is the set $P_1(\mathcal{T}_h)$ of piecewise linear polynomials. Any $\varphi_j \in P_1(\mathcal{T}_h)$ is a linear function on each of the surrounding elements of vertex j and is identically equal to zero on all other elements. Figure 1.1 shows an example of a P_1 basis function which is nonzero on six elements of a triangular mesh. Clearly, φ_j is not differentiable (at vertex j and along the border of the patch of its surrounding elements). However, it is simple to show that φ_j is weakly differentiable and therefore that $P_1(\mathcal{T}_h) \subset H^1(\Omega)$.

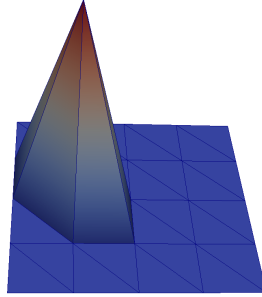


Fig. 1.1: P_1 basis function

The definition of the set $P_2(\mathcal{T}_h)$ of piecewise quadratic functions requires additional nodes located at the mid-point of each mesh edge. As with P_1 , a basis function $\varphi_j \in P_2 \subset H^1(\Omega)$ is nonzero only on the elements surrounding node j . More pre-

cisely, any $\varphi_j \in P_2(\mathcal{T}_h)$ is a quadratic function on each of the elements adjacent to node j and is identically equal to zero on all other elements. Further, φ_j is equal to one at node j and equal to zero on all other nodes in the mesh. Figure 1.2 shows the two types of P_2 basis functions, the first one being equal to 1 on an edge mid-point of an element and the second one being equal to 1 on a vertex. Finite elements can be defined in a similar manner for piecewise polynomial functions of arbitrary order $p \in \mathbb{N}$.

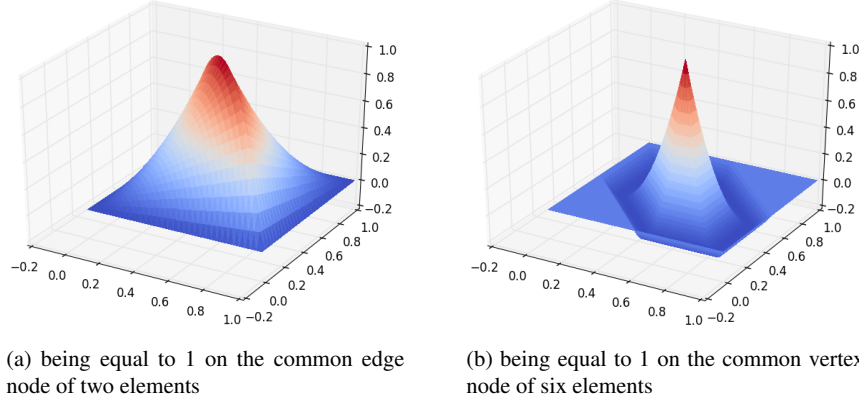


Fig. 1.2: P_2 basis functions

The Galerkin matrix A created by this choice of basis functions is sparse. For example, on a perfectly regular mesh, each interior vertex is surrounded by six triangles, so the matrix rows associated with the P_1 nodes at each interior vertex have seven non-zeros. More generally the number of non-zeros on each row depends only on the local mesh topology: it does not increase as the mesh is refined or the domain extended. Similarly, the rows corresponding to the interior vertex nodes of P_2 on a completely regular mesh have 19 nonzeros, while the rows associated with edge nodes have 9 nonzeros. The sparsity of A is exploited to solve the Galerkin system (1.30) efficiently. A discussion of efficient sparse linear solvers however is beyond the scope of this work, but the interested reader is referred to [6], [8] and [32] for details.

1.1.1.7 Assembly of the Galerkin system

Having created a triangulation of the domain Ω with an associated set of basis functions $\varphi_1, \dots, \varphi_d$, the linearity of integration can be exploited to decompose the assembly of the Galerkin system (1.30) into a local integral on each cell and boundary facet. For example, we can rewrite the global stiffness matrix (1.31) as

$$\begin{aligned}
A_{ij} &= \int_{\Omega} \nabla \varphi_j \cdot \nabla \varphi_i \, dx, \\
&= \sum_{\Delta_k \in \mathcal{T}_h} \int_{\Delta_k} \nabla \varphi_j \cdot \nabla \varphi_i \, dx.
\end{aligned} \tag{1.35}$$

On every element Δ_k there are only a small number $n_k \in \mathbb{N}$ of basis functions which are nonzero. For instance a P_1 space has only three such basis functions per element while for P_2 , there are six. For each cell k , we can define a function, $s_k(\hat{i})$, which maps the local basis function index $\hat{i}$ to the corresponding global index i . From this we can define the element scatter matrix $S_k[i, j] = \delta(s_k(\hat{i}), i)$, and hence write the global stiffness matrix as a sum of local contributions:

$$A = \sum_k S_k A_k S_k^T, \tag{1.36}$$

with the local stiffness matrix on cell k given by

$$A_k[\hat{i}, \hat{j}] = \int_{\Delta_k} \nabla \varphi_{s_k(\hat{i})} \cdot \nabla \varphi_{s_k(\hat{j})} \, dx. \tag{1.37}$$

Note that the local stiffness matrix is a dense $n_k \times n_k$ matrix which contributes a sparse update to A with sparsity given by the element scatter matrix. In practice, the element scatter matrix is not actually constructed. Instead, the scatter function s_k is employed directly to add contributions to the appropriate entries of A . Any essential boundary conditions are neglected in the local stiffness matrix. They are instead applied by modifying the global stiffness matrix A , either during or after the addition of the local contributions. This process applies *mutatis mutandis* to the assembly of the global load vector from local contributions.

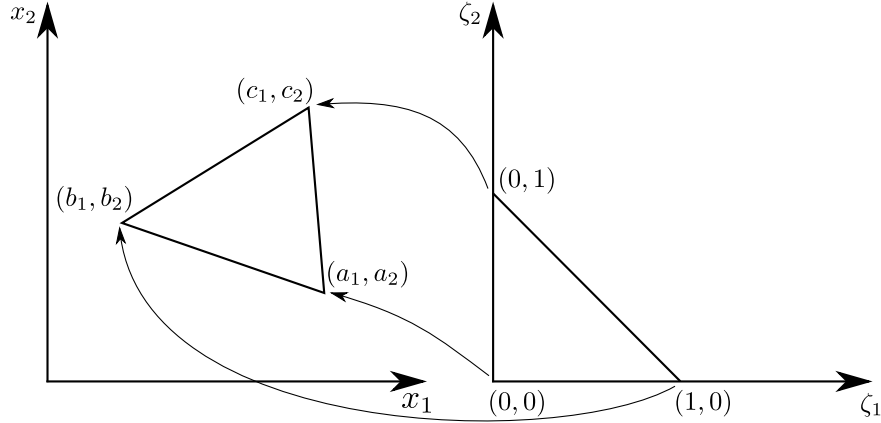


Fig. 1.3: Diffeomorphic mapping ρ_k for the P_1 element Δ^k

A convenient mechanism for the construction of the local stiffness matrix is to evaluate all element integrals on a single reference element Δ^* which is tied to each global element Δ_k by a diffeomorphic mapping, ρ_k . Here, Δ^* is formed by the vertices $(0,0)$, $(0,1)$ and $(1,0)$. As illustrated by Fig. 1.3, Δ^* is mapped onto the local element Δ_k with vertices $a = (a_1, a_2)$, $b = (b_1, b_2)$ and $c = (c_1, c_2)$ with ρ_k defined by,

$$\rho_k(\zeta_1, \zeta_2) = a\tau_1(\zeta_1, \zeta_2) + b\tau_2(\zeta_1, \zeta_2) + c\tau_3(\zeta_1, \zeta_2) \quad (1.38)$$

$$= (x_1, x_2), \quad (1.39)$$

where:

$$\tau_1(\zeta_1, \zeta_2) = 1 - \zeta_1 - \zeta_2, \quad (1.40)$$

$$\tau_2(\zeta_1, \zeta_2) = \zeta_1, \quad (1.41)$$

$$\tau_3(\zeta_1, \zeta_2) = \zeta_2. \quad (1.42)$$

This enables us to define a set of n_k local basis functions for our finite element space over Δ^* : $\{\psi_i\}$ with the property that $\psi_i(\zeta) = \phi_{s_k(i)}(\rho_k(\zeta))$, where $\zeta = (\zeta_1, \zeta_2)$. For clarity, the associated $\phi_{s_k(i)}$ are also called global basis functions. The local basis functions enable us to pull back the local stiffness contributions to the reference space:

$$\int_{\Delta_k} \nabla \phi_{s_k(i)} \cdot \nabla \phi_{s_k(j)} \, dx = \int_{\Delta^*} (J_k^T \nabla_\zeta \psi_i) \cdot (J_k^T \nabla_\zeta \psi_j) |J| \, d\zeta, \quad (1.43)$$

where $\nabla_\zeta := \left(\frac{\partial}{\partial \zeta_1}, \frac{\partial}{\partial \zeta_2} \right)^T$ is the gradient operator in local coordinates and $J := \nabla_\zeta \rho$ is the cell Jacobian¹:

$$\begin{aligned} J_{\alpha,\beta} &= \frac{\partial x_\alpha}{\partial \zeta_\beta} \\ &= a_\alpha \frac{\partial \tau_1}{\partial \zeta_\beta} + b_\alpha \frac{\partial \tau_2}{\partial \zeta_\beta} + c_\alpha \frac{\partial \tau_3}{\partial \zeta_\beta}. \end{aligned} \quad (1.44)$$

The identity $\nabla = J^T \nabla_\zeta$ follows immediately by the chain rule. The details of the assembly process are beyond the scope of this work, however the interested reader is referred to [2, Sect. 0.6], [8, Sect. 1.4], and [23, Chap. 6].

1.1.1.8 Example problem

Let us consider the boundary value problem (1.2, 1.4, 1.5) for Poisson's equation with parameters

¹ Note that the Jacobian, J , is a totally separate concept from the functional to be optimised from (1.1), which is also written J .

$$\Omega = [0, 1]^2, \quad (1.45)$$

$$\partial\Omega_D = \{x_1 = 1\} \cup \{x_2 = 1\}, \quad (1.46)$$

$$\partial\Omega_N = \Omega \setminus \partial\Omega_D, \quad (1.47)$$

$$g_D = 0 \quad \text{on} \quad \partial\Omega_D, \quad (1.48)$$

$$g_N = 0 \quad \text{on} \quad \partial\Omega_N, \quad (1.49)$$

$$f(x_1, x_2) = \chi_{\{x_1+x_2 < 1\}} \sin(4\pi x_1) \cos(4\pi x_2) \quad \forall (x_1, x_2) \in \Omega. \quad (1.50)$$

A plot of the source function f is displayed in Fig. 1.4a. Since f is discontinuous, there is no classical solution to this problem. However, it can be proved that there is a weak solution u and that u is unique, which is a direct consequence of Lax-Milgram's theorem. For details, the interested reader is referred to [2, Sect. 2.7].

An approximate solution u_h to the weak boundary value problem (1.23) with parameters (1.45–1.50) is computed numerically using the finite element method embedded within the Firedrake framework [31]. Firedrake is an automated system for the portable solution of PDEs using finite elements. It allows for a succinct implementation for a large variety of discretisations and PDEs. For an introductory tutorial to the usage of Firedrake, we refer to [11]. The code used to generate the corresponding solution in Python can be found in [33]. Here, the weak solution u_h is approximated using P_1 finite elements. A plot of u_h is displayed in Fig. 1.4b.

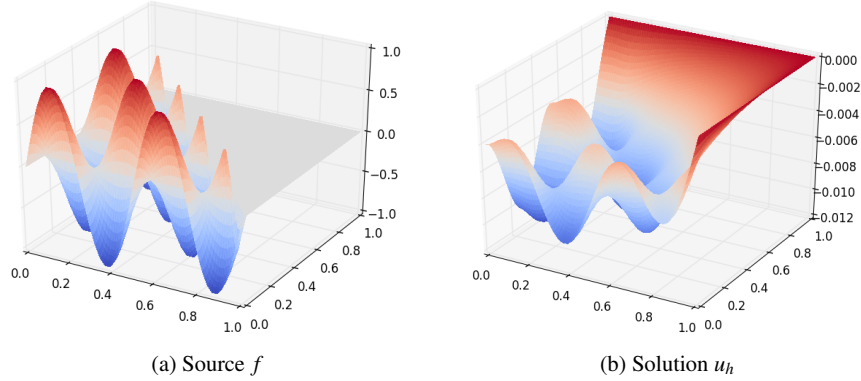


Fig. 1.4: Plots of the discontinuous source function f and the finite element solution u_h to the weak boundary value problem (1.23) for Poisson's equation with parameters (1.45–1.50)

1.1.2 Burgers' equation

The viscous form of Burgers' equation

$$\frac{\partial u}{\partial t} + (u \cdot \nabla) u - \eta \Delta u = 0 \quad \text{on } [0, T] \times \Omega, \quad (1.51)$$

is a fundamental nonlinear partial differential equation describing the advection and diffusion of u , a velocity field. Here, $\eta > 0$ denotes an assumed to be constant viscosity or diffusion coefficient and $u : [0, T] \times \Omega \rightarrow \mathbb{R}^2$ the associated solution field over the time interval $[0, T]$, $T > 0$, and the domain $\Omega \subset \mathbb{R}^2$. Further,

$$\Delta u := (\Delta u_1, \Delta u_2)^T, \quad (1.52)$$

denotes the vector Laplacian of $u := (u_1, u_2)^T$. By simple vector computations, one finds

$$(u \cdot \nabla) u = \left(u_1 \frac{\partial u_1}{\partial x_1} + u_2 \frac{\partial u_1}{\partial x_2}, u_1 \frac{\partial u_2}{\partial x_1} + u_2 \frac{\partial u_2}{\partial x_2} \right)^T. \quad (1.53)$$

The solution of (1.51) is assumed here to satisfy Neumann boundary conditions given by

$$(n \cdot \nabla) u = 0 \quad \text{on } [0, T] \times \partial \Omega, \quad (1.54)$$

for which the vector term can simply be written as

$$(n \cdot \nabla) u = \left(\frac{\partial u_1}{\partial n}, \frac{\partial u_2}{\partial n} \right)^T. \quad (1.55)$$

Since the problem at hand is time dependent, an initial condition, i.e. a condition for the state of the system at time $t = 0$, is given by

$$u(0, x) = u_0(x), \quad (1.56)$$

for a given function $u_0 : \Omega \rightarrow \mathbb{R}^2$. Combining Burgers' equation (1.51) with the initial condition (1.56) forms a so called initial value problem. Together with the boundary condition (1.54), it is also a boundary value problem.

1.1.2.1 Weak formulation

Using vector multiplication to multiply both sides of (1.51) by a test function $v : \Omega \rightarrow \mathbb{R}^2$ from some appropriate test function space and integrating over the whole domain yields

$$\int_{\Omega} \frac{\partial u}{\partial t} \cdot v \, dx + \int_{\Omega} (u \cdot \nabla) u \cdot v \, dx - \eta \int_{\Omega} \Delta u \cdot v \, dx = 0. \quad (1.57)$$

Applying the integration-by-parts formula to the Laplace term yields

$$\int_{\Omega} \frac{\partial u}{\partial t} \cdot v \, dx + \int_{\Omega} (u \cdot \nabla) u \cdot v \, dx + \eta \int_{\Omega} \nabla u : \nabla v \, dx = 0, \quad (1.58)$$

where

$$\nabla u : \nabla v := \nabla u_1 \cdot \nabla v_1 + \nabla u_2 \cdot \nabla v_2. \quad (1.59)$$

The boundary term arising from integrating by parts has disappeared here due to the application of the Neumann boundary condition (1.54).

1.1.2.2 Time discretisation

Since we are dealing with a time dependent problem, we need to discretise (1.161) in time. For simplicity and stability, the backward Euler discretisation is used. For a given number of time steps N and associated time step size $dt = T/N$, we solve

$$\begin{aligned} \int_{\Omega} \frac{u^{n+1} - u^n}{dt} \cdot v \, dx + \int_{\Omega} (u^{n+1} \cdot \nabla) u^{n+1} \cdot v \, dx \\ + \eta \int_{\Omega} \nabla u^{n+1} : \nabla v \, dx = 0 \quad \forall n = 0, \dots, N-1, \end{aligned} \quad (1.60)$$

where $u^n = u(n \cdot dt, \cdot)$ and thus $u^0 = u(0, \cdot) = u_0$. Note that every u^n is now a time independent function, i.e. $u^n : \Omega \rightarrow \mathbb{R}^2 \, \forall n = 0, \dots, N-1$. The natural space for the solution and test functions in (1.60) is the tensor product Sobolev space $H^1(\Omega)^2 := H^1(\Omega) \otimes H^1(\Omega)$, i.e.

$$H^1(\Omega)^2 = \left\{ u : \Omega \rightarrow \mathbb{R}^2 \left| u_i, \frac{\partial u_i}{\partial x_j} \in L^2(\Omega) \, \forall i, j \right. \right\}. \quad (1.61)$$

The complete weak formulation of the Burgers' initial value problem can now be written as follows: find $u^{n+1} \in H^1(\Omega)^2$ such that

$$\begin{aligned} \int_{\Omega} \frac{u^{n+1} - u^n}{dt} \cdot v \, dx + \int_{\Omega} (u^{n+1} \cdot \nabla) u^{n+1} \cdot v \, dx \\ + \eta \int_{\Omega} \nabla u^{n+1} : \nabla v \, dx = 0 \quad \forall v \in H^1(\Omega)^2 \, \forall n = 0, \dots, N-1. \end{aligned} \quad (1.62)$$

1.1.2.3 Galerkin approximation

Consider a finite-dimensional subspace $V_h \subset H^1(\Omega)^2$. Any $u_h \in V_h$ can be represented as

$$u_h = \sum_{i=1}^d u_{h,i} \varphi_i. \quad (1.63)$$

In contrast to the previous, scalar, Poisson equation, the solution to Burgers' equation is vector-valued. We can construct a suitable finite element space by taking the tensor product of two scalar valued finite element spaces. Suppose $W_h \subset H^1(\Omega)$ is some such space with basis $\omega_1, \dots, \omega_d$. Then a basis for $V_h = W_h \otimes W_h$ is given by:

$$\{e_1 \omega_i \mid 0 < i \leq d\} \cup \{e_2 \omega_i \mid 0 < i \leq d\}, \quad (1.64)$$

where e_1 and e_2 are the unit vectors in the x_1 and x_2 directions respectively.

The Galerkin approximation of Burgers' equation is a finite-dimensional approximation of the weak formulation (1.62), which can be expressed as follows: Find $u_h^{n+1} \in V_h$ such that

$$\begin{aligned} \int_{\Omega} \frac{u_h^{n+1} - u_h^n}{dt} \cdot v_h \, dx + \int_{\Omega} (u_h^{n+1} \cdot \nabla) u_h^{n+1} \cdot v_h \, dx \\ + \eta \int_{\Omega} \nabla u_h^{n+1} : \nabla v_h \, dx = 0 \quad \forall v_h \in V_h \quad \forall n = 0, \dots, N-1. \end{aligned} \quad (1.65)$$

1.1.2.4 Residual form

Unlike the Poisson problem, the weak formulation (1.62) of Burgers' initial value problem is nonlinear in the solution u . As a consequence, the discretised problem does not simply result in a matrix system which can be solved using linear algebra techniques. Instead, Newton-like nonlinear solvers are typically employed. The first step is to express the system in what is termed *residual form*. For the weak form of Burgers' equation (1.65) this is: find u_h^{n+1} such that

$$F(u_h^{n+1}; v_h) = 0 \quad \forall v_h \in V_h, \quad \forall n = 0, \dots, N-1, \quad (1.66)$$

where the *residual*, F , is defined by

$$F(u_h^{n+1}; v_h) = \int_{\Omega} \frac{u_h^{n+1} - u_h^n}{dt} \cdot v_h \, dx + \int_{\Omega} (u_h^{n+1} \cdot \nabla) u_h^{n+1} \cdot v_h \, dx \quad (1.67)$$

$$+ \eta \int_{\Omega} \nabla u_h^{n+1} : \nabla v_h \, dx. \quad (1.68)$$

The semicolon indicates that F is linear in the argument to the right of the semicolon. Note that F is not linear in the first argument. Since the residual is linear in v_h , it is sufficient that (1.66) holds for every test function in the basis of V_h . That is, the problem reduces to finding $u_h^{n+1} \in V_h$ such that

$$F(u_h^{n+1}; \varphi_j) = 0 \quad \forall j = 1, \dots, d, \quad \forall n = 0, \dots, N-1. \quad (1.69)$$

1.1.2.5 Linearisation

The residual form (1.66) can be employed to formulate the linearisation of the problem as required by Newton-like methods.

The linearisation of the residual F requires its differentiation with respect to its first argument u_h^{n+1} . For the sake of simplicity we shall omit the upper ‘ n ’ index for the time discretisation in the following, i.e. u_h^{n+1} is replaced by u_h . Since at every time step $n = 0, \dots, N-1$ an equation with equivalent structure is solved, there is no loss of generality in what follows. Since u_h is not an element of a Euclidean space but is a function, a more general notion of differentiability than for functions on Euclidean spaces is needed. The Gâteaux derivative of $F(\cdot; v_h)$ at u_h is given by

$$dF_{u_h}(u_h; v_h, \hat{u}) = \lim_{\varepsilon \rightarrow 0} \frac{F(u_h + \varepsilon \hat{u}; v_h) - F(u_h; v_h)}{\varepsilon} \quad \forall \hat{u} \in V_h. \quad (1.70)$$

In other words, $dF_{u_h}(u_h; v_h, \hat{u})$ denotes the directional derivative of $F(\cdot; v_h)$ with respect to u_h , at the current value of u_h and in the “direction” $\hat{u}$. The subscript will be omitted when there is no ambiguity about the variable with respect to which the derivative is being taken. A more formal introduction to Gâteaux derivatives is given in Sect. 1.2.

The Gâteaux derivative of the Burgers’ residual can be explicitly computed by

$$dF(u_h; v_h, \hat{u}) = \lim_{\varepsilon \rightarrow 0} \left[\int_{\Omega} \left(\frac{\hat{u}}{dt} + (\hat{u} \cdot \nabla) u_h + ((u_h + \varepsilon \hat{u}) \cdot \nabla) \hat{u} \right) \cdot v_h \right. \quad (1.71)$$

$$\left. + \eta \nabla \hat{u} : \nabla v_h \, dx \right] \quad (1.72)$$

$$= \int_{\Omega} \left(\frac{\hat{u}}{dt} + (\hat{u} \cdot \nabla) u_h + (u_h \cdot \nabla) \hat{u} \right) \cdot v_h + \eta \nabla \hat{u} : \nabla v_h \, dx. \quad (1.73)$$

It is easy to see that dF_{u_h} is linear in both v_h and $\hat{u}$. This linearity will be used to solve the residual form of Burgers’ initial value problem.

1.1.2.6 Solving the residual form

Since the derivative of the residual is available, a straightforward way to solve the residual form (1.66) is to apply Newton’s method. Just as in the Euclidean case, Newton’s method on function spaces consists of approximating the function, in this case the residual, by its Taylor expansion only up to the linear term and solving for the update that will set the approximation to zero. The linear Taylor series approximation of $F(\cdot; v_h)$ at a perturbed solution $u_h + \hat{u}$ is given by

$$F(u_h + \hat{u}; v_h) \approx F(u_h; v_h) + dF(u_h; v_h, \hat{u}) \quad \forall v_h \in V_h. \quad (1.74)$$

Thus, if we write the $(k+1)$ th Newton iterate u_h^{k+1} as

$$u_h^{k+1} = u_h^k + \hat{u}, \quad (1.75)$$

then the update $\hat{u} \in V_h$ is the solution to

$$dF(u_h^k; v_h, \hat{u}) = -F(u_h^k; v_h) \quad \forall v_h \in V_h. \quad (1.76)$$

Due to the linearity of $dF(u_h^k; v_h, \hat{u})$ in $\hat{u}$, Eq. (1.76) is simply a linear finite element problem. More precisely, using the linearity of $dF(u_h^k; v_h, \hat{u})$ in $\hat{u}$ and the linear independence of different basis functions in V_h this can be written as

$$\sum_{j=1}^d dF(u_h^k; \varphi_i, \varphi_j) \hat{u}_j = -F(u_h^k; \varphi_i) \quad \forall i = 1, \dots, d, \quad (1.77)$$

where $\hat{u}_1, \dots, \hat{u}_d$ are the coefficients of the expansion of $\hat{u}$ in terms of the basis functions. In matrix form, the last equation can be written as

$$\hat{A} \hat{\mathbf{u}} = \hat{\mathbf{f}}, \quad (1.78)$$

where the entry in the i th row and j th column of $\hat{A}$ is given by

$$\begin{aligned} \hat{A}_{ij} &= dF(u_h^k; \varphi_i, \varphi_j) \\ &= \int_{\Omega} \left(\frac{\varphi_j}{dt} + (\varphi_j \cdot \nabla) u_h^k + (u_h^k \cdot \nabla) \varphi_j \right) \cdot \varphi_i + \eta (\nabla \varphi_j : \nabla \varphi_i) dx \quad \forall i, j = 1, \dots, d, \end{aligned} \quad (1.79)$$

$$(1.80)$$

$\hat{\mathbf{u}} = (\hat{u}_1, \dots, \hat{u}_d)$, and $\hat{\mathbf{f}} = (\hat{f}_1, \dots, \hat{f}_d)$ with

$$\hat{f}_i = -F(u_h^k; \varphi_i) \quad \forall i = 1, \dots, d. \quad (1.81)$$

As was the case for Poisson's equation (1.30), this is a linear system which can be solved using standard linear algebra techniques. In essence, Newton's method converts the nonlinear system into a (hopefully convergent) series of linear problems. More sophisticated nonlinear solvers such as line search Newton methods are available, but the core process of linearising the equations to produce a sequence of linear finite element problems is common to these approaches.

Under certain conditions, Newton's method is guaranteed to generate a sequence $(u_h^k)_k$ of approximations that converges to the correct solution of the original nonlinear problem. However, we still need to decide when to stop the algorithm and accept the most recent update as a solution. This requires more involved consideration of the appropriate norms to employ. We therefore postpone its consideration to Sect. 1.3.3.

1.1.2.7 An example problem

Let us now consider the Burgers' initial value problem (1.51, 1.54, 1.56) with parameters

$$\Omega = [0, 1]^2, \quad (1.82)$$

$$T = 1, \quad (1.83)$$

$$\eta = 0.01, \quad (1.84)$$

$$u_0(x_1, x_2) = \left(\exp \left[-20 \cdot \left((x_1 - \frac{1}{2})^2 + (x_2 - \frac{1}{2})^2 \right) \right], 0 \right) \quad \forall (x_1, x_2) \in \Omega. \quad (1.85)$$

An approximate solution u_h to the weak formulation (1.62) of Burgers' initial value problem with parameters (1.82–1.85) is computed numerically using finite elements in Firedrake. The code that was used to generate the corresponding solution in Python can be found in [33]. Here, the vector-valued weak solution u_h is approximated using P_2 vector-valued finite elements. A plot of the first vector component u_{1_h} of u_h at successive points in time is displayed in Fig. 1.5.

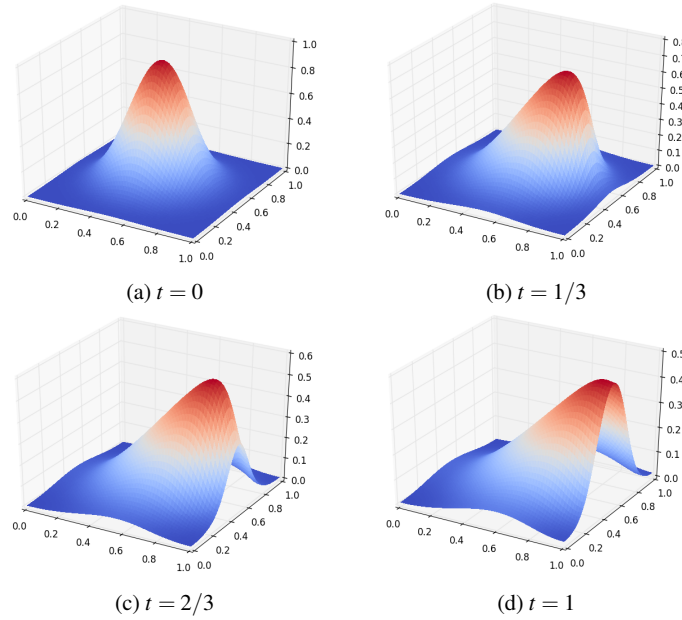


Fig. 1.5: Plots of the first vector component u_{1_h} of the finite element solution u_h to the weak Burgers' initial value problem (1.62) with parameters (1.82–1.85) at successive points in time

1.2 Hilbert spaces

The structure of a Hilbert space arises naturally in the finite element method since the weak formulation of a PDE involves an inner product with the test functions. One of the key properties of a Hilbert space is its close relationship with its dual space. The key result characterising this relationship is the Riesz representation theorem.

A Hilbert space is a complete vector space equipped with an inner product, $\langle \cdot, \cdot \rangle$. The availability of the inner product makes it possible to measure the angles between, and the length of, vectors in a Hilbert space. Completeness is the property that any Cauchy sequence in a Hilbert space converges to a limit in that space. This ensures that there are no “missing” limits with the consequence that the usual techniques of limit calculus can be applied. Here, we only consider Hilbert spaces over the real field, but most of the results here can be generalised to Hilbert spaces over other (for example complex) fields.

The inner product on a Hilbert space induces a norm defined for any $u \in H$ by:

$$\|u\|_H = \sqrt{\langle u, u \rangle}. \quad (1.86)$$

1.2.1 Dual spaces and the Riesz representation theorem

Let $F : H \rightarrow \mathbb{R}$ be a linear functional on a Hilbert space H . We call F continuous if

$$\exists c > 0 : |F(u)| \leq c \|u\|_H \quad \forall u \in H. \quad (1.87)$$

The set $\mathcal{L}(H, \mathbb{R})$ of continuous linear functionals on H is called the dual space of H and is also denoted by H^* , i.e.

$$H^* := \{F : H \rightarrow \mathbb{R} \mid F \text{ is linear and continuous}\}. \quad (1.88)$$

The Hilbert space H induces on its dual space H^* an operator norm $\|\cdot\|_{H^*}$ defined by:

$$\|F\|_{H^*} := \sup_{\|u\|_H=1} |F(u)| \quad \forall F \in H^*. \quad (1.89)$$

The one fundamental relationship between a Hilbert space H and its dual space H^* is that these spaces are isometrically isomorphic to each other. In more comprehensible terms, this property is expressed in the following theorem.

Theorem 1.1 (The Riesz representation theorem). *Let H be a Hilbert space with inner product $\langle \cdot, \cdot \rangle_H$. For any $F \in H^*$ there is a unique $u_F \in H$ such that*

$$F(v) = \langle u_F, v \rangle_H \quad \forall v \in H,$$

and that $\|F\|_{H^*} = \|u_F\|_H$. Conversely, for any $u \in H$ the functional F_u defined by

$$F_u(v) := \langle u, v \rangle_H \quad \forall v \in H, \quad (1.90)$$

is an element of H^* .

The map $\mathcal{R}_H : H^* \rightarrow H$ that sends F to its unique representative $u_F \in H$ is called the Riesz map. Further, $\mathcal{R}_H(F) = u_F$ is called the Riesz representer of F in H .

In simple terms, the theorem states that any element of the dual space H^* can be uniquely represented by an element in the primal space H and vice versa, while their respective norms are equivalent.

The Riesz representation theorem is of essential importance for this work and will be used repeatedly. It is therefore appropriate to provide a proof, which follows [1, Sect. 6.1].

Proof (Riesz representation theorem). It is enough to show that the inverse mapping of the Riesz map $\tau := \mathcal{R}_H^{-1}$ given by

$$\tau(u)(v) \mapsto \langle u, v \rangle_H \quad \forall u, v \in H, \quad (1.91)$$

is an isometric isomorphism. Let $u \in H$. Obviously, $\tau(u)$ is linear. The Cauchy-Schwarz inequality yields

$$|\tau(u)(v)| \leq \|u\|_H \cdot \|v\|_H \quad \forall v \in H. \quad (1.92)$$

Therefore, $\tau(u)$ is continuous, which means $\tau(u) \in H^*$, and $\|\tau(u)\|_{H^*} \leq \|u\|_H$ by definition of the operator norm. But since $\|\tau(u)(u)\| = \|u\|^2$, we also have $\|\tau(u)\|_{H^*} \geq \|u\|_H$. Hence, τ is an isometry. Let $u_0 \in H$ such that $\tau(u_0) = 0 \in H^*$. Then,

$$0 = \|\tau(u_0)\|_{H^*} = \|u_0\|_H, \quad (1.93)$$

and thus $u_0 = 0 \in H$. Hence, τ is injective due to its linearity in u . It remains to show that τ is surjective. Consider $0 \neq F_0 \in H^*$ and let P be the orthogonal projection from H onto the closed null space

$$\mathcal{N}(F_0) := \{u \in H : F_0(u) = 0\}. \quad (1.94)$$

The existence and uniqueness of P is ensured by the projection theorem [1, Sect. 4.3]. Let $e \in H$ such that $F_0(e) = 1$, and define

$$u_0 := e - Pe. \quad (1.95)$$

Hence, $F_0(u_0) = F_0(e) - F_0(Pe) = 1$, which implies $u_0 \neq 0$. Using the characteristic property of the projection, i.e. $e - Pe \in \mathcal{N}(F_0)^\perp$, we have

$$\langle u_0, v \rangle_H = 0 \quad \forall v \in \mathcal{N}(F_0). \quad (1.96)$$

As $F_0(u - F_0(u)u_0) = F_0(u) - F_0(u) \cdot 1 = 0$, (1.96) implies that

$$\langle u_0, u \rangle_H = \langle u_0, u - F_0(u)u_0 \rangle_H + \langle u_0, F_0(u)u_0 \rangle_H \quad (1.97)$$

$$= \langle u_0, F_0(u)u_0 \rangle_H \quad (1.98)$$

$$= F_0(u)\|u_0\|^2 \quad \forall u \in H. \quad (1.99)$$

Dividing by $\|u_0\|^2$ yields

$$F_0(u) = \left\langle \frac{u_0}{\|u_0\|^2}, u \right\rangle_H = \tau \left(\frac{u_0}{\|u_0\|^2} \right)(u) \quad \forall u \in H, \quad (1.100)$$

which concludes the proof.

1.2.2 Fréchet derivatives

Fréchet differentiability is a generalisation of the differentiability from functions on Euclidean space to more general spaces such as Hilbert spaces. Using the Fréchet derivative it is possible to extend core results from Euclidean analysis such as Taylor's theorem to a more general setting.

Let $F : U \subset H \rightarrow \mathbb{R}$ be a functional on an open set $U \neq \emptyset$ of some Hilbert space H . We call F directionally differentiable if

$$dF(u; h) := \lim_{\varepsilon \rightarrow 0^+} \frac{F(u + \varepsilon h) - F(u)}{\varepsilon}, \quad (1.101)$$

exists for all $u, h \in U$. We call F Gâteaux-differentiable if F is directionally differentiable and the directional derivative $dF(u; \cdot) : h \mapsto dF(u; h)$ is continuous and linear, i.e. $dF(u; \cdot) \in H^* \forall u \in U$. Further, we call F Fréchet-differentiable if F is Gâteaux-differentiable and if

$$\lim_{\|h\|_H \rightarrow 0} \frac{\|F(u + h) - F(u) - dF(u; h)\|}{\|h\|_H} = 0 \quad \forall u \in U. \quad (1.102)$$

Since $dF(u; \cdot) \in H^*$,

$$dF(u; v) = \langle \mathcal{R}(F'(u)), v \rangle_H \quad \forall v \in U, \quad (1.103)$$

where $\mathcal{R}(dF(u; \cdot))$ denotes the Riesz representer of $dF(u; \cdot)$ in H . That is, the Fréchet derivative of a functional on H can be represented with respect to an element of H .

If F is Fréchet-differentiable and $dF(u; \cdot) \in \mathcal{L}(U, \mathbb{R})$ is also Fréchet-differentiable, then F is called twice Fréchet-differentiable. We denote by $dF^{(2)}(u; \cdot, \cdot) \in \mathcal{L}(U, \mathcal{L}(U, \mathbb{R}))$ the second Fréchet derivative of F at u . The n th Fréchet derivative of F for any $n \in \mathbb{N}$ can be defined accordingly.

The following result generalises Taylor's theorem to functionals on Hilbert spaces. Analogous statements hold for operators between Banach spaces. For a detailed proof, the interested reader is referred to [14, Theorem 5].

If F is n -times Fréchet-differentiable, we have

$$\begin{aligned} F(u+h) &= F(u) + dF(u;h) + \frac{1}{2}d^{(2)}F(u;h,h) + \dots \\ &\quad + \frac{1}{n!}dF^{(n)}(u;\underbrace{h,\dots,h}_{n \text{ times}}) + \mathcal{O}(\|h\|_H^{n+1}) \quad \forall u, h \in U. \end{aligned} \quad (1.104)$$

1.2.3 Partial derivatives and the chain rule

In the derivation of the adjoint to a PDE, it will also be necessary to differentiate between partial and total derivatives. Define $F : U \times U \rightarrow \mathbb{R}$ with U an open subset of some Hilbert space H , and $u : U \rightarrow U$. Then

$$\partial F_m(u(m), m; h) = \lim_{\varepsilon \rightarrow 0^+} \frac{F(u(m), m + \varepsilon h) - F(u(m), m)}{\varepsilon}, \quad (1.105)$$

is the partial derivative of F with respect to m while

$$dF_m(u(m), m; h) = \lim_{\varepsilon \rightarrow 0^+} \frac{F(u(m + \varepsilon h), m + \varepsilon h) - F(u(m), m)}{\varepsilon}, \quad (1.106)$$

is the corresponding total derivative. As is the case in Euclidean space, the partial and total derivatives are related by the chain rule, which in this case is given by

$$dF_m(u(m), m; \cdot) = \partial F_u(u(m), m; du_m(m; \cdot)) + \partial F_m(u(m), m; \cdot). \quad (1.107)$$

Omitting function arguments, this produces the familiar form

$$dF_m = \partial F_u du_m + \partial F_m. \quad (1.108)$$

1.3 The relation between primal and dual finite element spaces

As previously seen for Burgers' equation, a finite element problem can be formulated in terms of its residual. The root of the residual corresponds to a solution of the problem, which leads to the archetypal formulation of a finite element problem, find $u \in U$ such that

$$F(u; v) = 0 \quad \forall v \in V, \quad (1.109)$$

where U and V are suitable finite element spaces. In general, we call U the trial function space and V the test function space.

The weak formulation is derived by multiplying the original PDE with a test function v and integrating over the domain. Due to the linearity of integration $F(u; v)$ is therefore linear in v for any given $u \in U$. Further, for many PDEs it is straightforward to show that $F(u; \cdot)$ is continuous and therefore that $F(u; \cdot) \in V^*$. If we recall the definition of the operator (or dual) norm, then (1.109) can be written: find $u \in U$ such that

$$\|F(u; \cdot)\|_* = 0. \quad (1.110)$$

Similarly, the archetypal linear problem, find $u \in U$ such that

$$a(u, v) = L(v) \quad \forall v \in V, \quad (1.111)$$

can equivalently be written, find $u \in U$ such that

$$\|a(u, \cdot) - L(\cdot)\|_* = 0. \quad (1.112)$$

In other words the weak form of a PDE is actually an equation between linear operators, with equality defined in terms of the appropriate operator norm.

Naturally, this is also reflected in the nature of the assembled matrices and vectors which result from actually invoking the finite element method. Recall that $v \in V$ can be represented with respect to a basis $\mathcal{P} = \{\phi_1, \dots, \phi_d\}$:

$$v = \sum_{i=1}^d v_i \phi_i. \quad (1.113)$$

When we represent a function within a finite element space as a vector, it is of course this set of coefficients v_i that we are storing within an implementation in code.

$\mathcal{P}$ also induces a unique dual basis for V^* , $\mathcal{P}^* = \{\phi_1^*, \dots, \phi_d^*\}$. The defining property of the dual basis is its orthogonality to $\mathcal{P}$:

$$\phi_i^*(\phi_j) = \delta_{ij} \quad \forall i, j = 1, \dots, d. \quad (1.114)$$

This immediately results in the fundamental identity relating the dual basis, $\mathcal{P}^*$, and the coefficients of the primal basis, $\mathcal{P}$:

$$\forall v \in V, v = \sum_{i=1}^d v_i \phi_i \iff v_i = \phi_i^*(v) \quad \forall i = 1, \dots, d. \quad (1.115)$$

Conversely, assembling a finite element operator as a vector simply amounts to expressing the operator as a linear combination of the basis functions for the dual space. In this case, the primal basis functions define the coefficients:

$$\forall F \in V^*, F = \sum_{i=1}^d F_i \phi_i^* \iff F_i = F(\phi_i) \quad \forall i = 1, \dots, d. \quad (1.116)$$

1.3.1 Example: application of assembled finite element operators

A straightforward consequence of the above is that we can explain why the application of a finite element operator to a function is equivalent to the (ℓ^2) dot product of the associated assembled vectors. Suppose we have $v(= \sum_i v_i \phi_i) \in V$ and $F(= \sum_i F_i \phi_i^*) \in V^*$. Then

$$\begin{aligned}
 F(v) &= \sum_{i=1}^d F_i \phi_i^* \left(\sum_{j=1}^d v_j \phi_j \right) \\
 &= \sum_{i,j=1}^d F_i v_j \phi_i^*(\phi_j) \\
 &= \sum_{i,j=1}^d F_i v_j \delta_{ij} \\
 &= \sum_{i=1}^d F_i v_i \\
 &= \mathbf{F} \cdot \mathbf{v},
 \end{aligned} \tag{1.117}$$

where $\mathbf{F}$ and $\mathbf{v}$ are the vectors of basis function coefficients corresponding to F and v , respectively. It is helpful in this context to follow [21] and introduce an operator, $\mathcal{J} : \mathbb{R}^{|V|} \rightarrow V$, defined by

$$\mathcal{J}(\mathbf{v}) = \sum_{i=1}^d v_i \phi_i. \tag{1.118}$$

In other words, $\mathcal{J}^{-1}$ takes functions $v \in V$ to their representation as a vector of basis function coefficients. We define a similar operator for the dual space, $\mathcal{J}^* : \mathbb{R}^{|V|} \rightarrow V^*$:

$$\mathcal{J}^*(\mathbf{F}) = \sum_{i=1}^d F_i \phi_i^*. \tag{1.119}$$

1.3.2 The primal and dual norms for L^2 and H^1

We will return to the choice of an appropriate norm in a subsequent chapter, however it may be helpful in understanding the relationship between the primal function space V and its dual V^* to work through the evaluation of a particular operator norm. For this purpose, we choose the most basic norm on finite element spaces, L^2 , and let trial and function space coincide. Recall that for $u \in V \subset L^2$,

$$\|u\|_{L^2}^2 = \int_{\Omega} u^2 \, dx. \tag{1.120}$$

Equivalently, the L^2 inner product is given by:

$$\langle u, v \rangle_{L^2} = \int_{\Omega} uv \, dx \quad \forall u, v \in V. \quad (1.121)$$

Simply by writing $u = \sum_i u_i \phi_i$, $v = \sum_j v_j \phi_j$, we can transform this into the matrix expression

$$\langle u, v \rangle_{L^2} = \mathbf{u}^T \mathbf{M} \mathbf{v}, \quad (1.122)$$

where

$$M_{ij} = \int_{\Omega} \phi_i \phi_j \, dx, \quad (1.123)$$

is commonly referred to as the mass matrix. Now, for any $u \in V$, we can define a functional $F_u \in \mathcal{L}(L^2, \mathbb{R})$ via

$$F_u(\cdot) = \langle u, \cdot \rangle_{L^2}. \quad (1.124)$$

It immediately follows that

$$\mathcal{J}^{*-1}(F_u) = \mathbf{F}_u = \mathbf{M} \mathbf{u}. \quad (1.125)$$

Conversely, for any $F \in V^*$, the uniqueness of the Riesz representation guarantees that

$$\mathcal{J}^{-1}(\mathcal{R}_{L^2}(F)) = \mathbf{M}^{-1} \mathcal{J}^{*-1}(F) = \mathbf{M}^{-1} \mathbf{F}. \quad (1.126)$$

In other words, the L^2 Riesz representer of a functional F is obtained by multiplying the corresponding coefficient vector by the inverse mass matrix.

The Riesz representation theorem also enables us to understand the role of the mass matrix in evaluating the L^2 operator norm. Recall that, for any $F \in V^*$

$$\|F\|_{L^{2*}} = \|\mathcal{R}_{L^2}(F)\|_{L^2}, \quad (1.127)$$

which implies that

$$\begin{aligned} \|F\|_{L^{2*}}^2 &= \langle \mathcal{J}(\mathbf{M}^{-1} \mathbf{F}), \mathcal{J}(\mathbf{M}^{-1} \mathbf{F}) \rangle_{L^2} \\ &= \mathbf{F}^T \mathbf{M}^{-1} \mathbf{M} \mathbf{M}^{-1} \mathbf{F} \\ &= \mathbf{F}^T \mathbf{M}^{-1} \mathbf{F} \\ &= F(\mathcal{R}_{L^2}(F)). \end{aligned} \quad (1.128)$$

An analogous argument to that given above applies to finite element subspaces of H^1 with inner product given by

$$\langle u, v \rangle_{H^1} = \int_{\Omega} \nabla u \cdot \nabla v + uv \, dx. \quad (1.129)$$

In this case, if $V \subset H^1$ and $\mathbf{F} = \mathcal{J}^{*-1}(F)$ with $F \in V^*$ then

$$\mathcal{R}_{H^1}(F) = \mathbf{A}^{-1} \mathbf{F}, \quad (1.130)$$

where

$$A_{ij} = \int_{\Omega} \nabla \phi_i \cdot \nabla \phi_j + \phi_i \phi_j \, dx. \quad (1.131)$$

1.3.3 Convergence criteria for Newton's method

Having established the relationship between finite element solutions and residuals, and their corresponding norms, we can now return to the question of convergence criteria for nonlinear solvers, which we postponed from Sect. 1.1.2.6.

One possibility is use the magnitude of the residual as the stopping criterion, possibly normalised by the magnitude of the residual at the initial guess. The residual measures the extent to which the current iteration fails to be a solution to the discretised PDE, and this approach essentially coincides with the usual approach for determining convergence in linear systems. Since the residual is an functional in the dual space to the solution, the natural norm in which to evaluate this residual is the dual norm appropriate to the finite element space under consideration. However, as shown in the previous section, this amounts to solving a matrix system, which seems an expensive operation given that the result merely indicates convergence and does not actually improve the solution in any way.

One widely adopted work-around for this solution is to employ a more conveniently calculated norm. The obvious and usual choice is to observe that the vector of coefficients of the residual vector is isomorphic to a vector in $\mathbb{R}^d$ and to employ the ℓ^2 norm of that vector:

$$\|F\|_{\ell^2} = \sqrt{\mathbf{F} \cdot \mathbf{F}}, \quad (1.132)$$

where $\mathbf{F} = \mathcal{J}^{*-1}(F)$. It is simple to show that this is indeed a norm on V^* . However, what has been lost by discarding the factor of M^{-1} or A^{-1} is the notion that the norm in which we measure the residual is a discrete approximation to the norm on the original, continuous, Sobolev space. In fact, as the example of the stiffness matrix in Sect. 1.1.1.7 shows, the discretisation is explicit in the mesh cell Jacobian entries that appear in the transformation of the integral measure and, in the H^1 case, in the gradient operator. This situation is, in fact, a close analogy of the problem that we analyse in much more depth in Chapt. 2. Rather than restate that analysis here, we will simply note that the impact of discarding the mesh operator from the norm is that the value of the ℓ^2 norm of the residual is entirely dependent on the mesh chosen. This has a number of undesirable consequences, including that the number of iterations required to achieve convergence is mesh-dependent, and that candidate solutions on different meshes which differ by arbitrarily small amounts in the primal norm, can result in residuals whose ℓ^2 norms differ by arbitrarily large factors. In short, the ℓ^2 norm is simply not a good indicator of the magnitude of the residual. Note that this problem arises through the infinite-dimensional formulation of the underlying problem. On finite-dimensional spaces, all norms are equivalent.

If the operator norm is inconvenient to evaluate, and other norms applied to the residual are poor indicators of convergence, how then should convergence be mea-

sured? One solution is to use the magnitude of the update step $\hat{u}$, which is an estimator for the error in the solution at the previous step of the iteration. Note that $\hat{u} \in V$ so the natural norm only involves multiplication by a sparse and easily assembled matrix. Given a tolerance $\varepsilon > 0$, one terminates the algorithm if

$$\frac{\|u_h^{k+1} - u_h^k\|_V}{\|u_h^k\|_V} < \varepsilon. \quad (1.133)$$

The best norm to use depends on the Sobolev space in which the solution is sought. For the examples we have considered, this is the H^1 norm. For a much more in-depth analysis of nonlinear solvers for numerical PDE problems, the reader is referred to [7]. The Riesz map also emerges as an important consideration in the preconditioning of the linear systems which emerge from the numerical solution of PDEs. The interested reader is referred to [21].

1.4 Adjoint and tangent linear equations

Recall the generic formulation of a PDE-constrained optimisation problem

$$\min_{\substack{u \in V \\ m \in M}} J(u, m) \quad (1.134)$$

$$\text{subject to } F(u, m; v) = 0 \quad \forall v \in V', \quad (1.135)$$

where $J : M \otimes V \rightarrow \mathbb{R}$ is the objective functional, $m \in M$ is the control variable, and $u \in V$ is the weak solution of the parametrised PDE. In the context of the adjoint equations (1.135) is referred to as the *forward model*. We assume that the forward model yields a unique solution for any control value, so that we can define the solution operator $u(\cdot) : M \rightarrow V$ which maps a control value to the associated solution of the forward model. Of course the explicit form of $u(m)$ is not known for many PDEs, but a finite element solver might be used instead as an approximation. Substituting the solution operator into (1.134) yields the *reduced functional*

$$\tilde{J}(m) = J(u(m), m), \quad (1.136)$$

and the associated *reduced optimization problem*

$$\min_{m \in M} \tilde{J}(m). \quad (1.137)$$

Most efficient optimisation algorithms for solving (1.137) require evaluations of the functional gradient with respect to the control variable, i.e. $d\tilde{J}_m(m; \cdot) = dJ_m(u(m), m; \cdot) \in M^*$. Table 1.1 lists common techniques for computing the gradient, together with an estimate of their computational expense. For PDE constrained optimisation, the number of input parameters (the dimension of M) is typically large, typically ranging from 10^2 when optimising the positioning of wind turbines to

Method	Number of PDE solves
Finite difference approximation	i
Complex step approach	i
Tangent linear approach	i
Adjoint approach	o

Table 1.1: Number of PDE solves required to compute the derivative of a computer model with i input and o output parameters for different methods. The PDE solves are typically the principle computational cost factor, and hence provide a good estimate for the total expense. PDE constrained optimization have $o = 1$ and $i \gg 1$, and hence the adjoint approach is by far the most efficient method

$> 10^9$ when optimising the initial state of the ocean in the ECCO2 ocean model [34]. At the same time, the number of output values is one (the functional value). Examining table 1.1 reveals that the adjoint approach is the most effective option for evaluating the functional gradient as the required number of PDE solves is independent of the number of input parameters.

1.4.1 A finite-dimensional example

To understand how the adjoint approach computes the functional derivative so efficiently, let us consider a finite dimensional example. Consider the functional

$$J(u, m) := j^T u + \frac{1}{2} m^T m, \quad (1.138)$$

with $j \in \mathbb{R}^n$ given and a parameter vector $m \in \mathbb{R}^m$. Furthermore, let $u \in \mathbb{R}^n$ be the solution of the linear system

$$Au = Bm, \quad (1.139)$$

with $A \in \mathbb{R}^{n \times n}$ invertible and $B \in \mathbb{R}^{n \times m}$.

Our aim is to compute the total derivative of the functional (1.138) with respect to m . Since A is invertible, we can solve (1.139) for u and substitute it into (1.138) to obtain the reduced functional:

$$\tilde{J}(m) := j^T A^{-1} B m + \frac{1}{2} m^T m. \quad (1.140)$$

From that, we can compute the total derivative with respect to m :

$$d\tilde{J}(m) = j^T A^{-1} B + m^T. \quad (1.141)$$

Depending on the order in which we evaluate these terms, we obtain the tangent linear or the adjoint approach:

Tangent linear approach

1. Solve the tangent linear system $A\mu = B$ for the tangent linear solution $\mu \in \mathbb{R}^{n \times m}$.
2. Evaluate the gradient with $j^T \mu + m^T \in \mathbb{R}^m$.

Adjoint approach

1. Solve the linear system $A^T \lambda = j$ for the adjoint solution $\lambda \in \mathbb{R}^n$.
2. Evaluate the gradient with $\lambda^T B + m^T \in \mathbb{R}^m$.

The two different approaches are visualised in figure 1.6. The computational cost of the tangent linear approach is dominated by the solution of the tangent linear system $A\mu = B$. Figure 1.6 reveals that the right hand side of this linear system is a matrix with m columns, corresponding to the number of parameters. One option to solve this system is to invert the matrix A explicitly (for example with a LU decomposition) and to perform a matrix-matrix multiplication to compute μ . Alternatively, m iterative solves can be performed for each column vector in B . However, in both cases, the computational cost scales linearly with m .

In contrast, the right hand side of the adjoint equation $A^T \lambda = j$ consists of a single column vector. Hence, the computation of the adjoint solution λ requires only one linear solve, independent of the dimension of m . Since the dominant computational cost is the solution of the linear systems, the adjoint approach yields a computational cost that is practically independent of m .

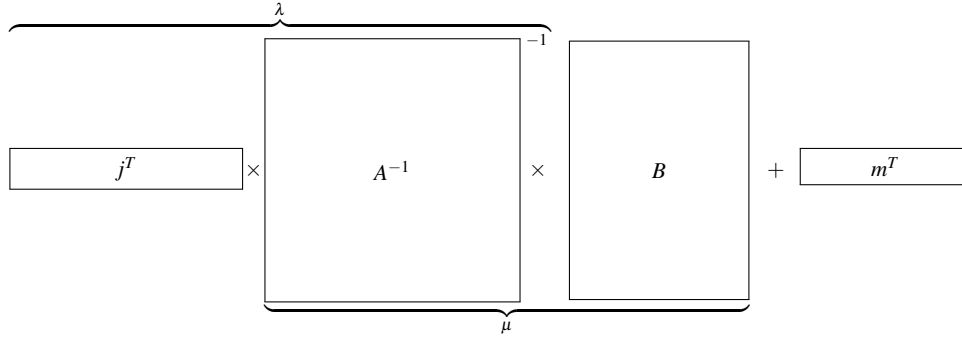


Fig. 1.6: The dimensions of the terms in the gradient (1.141) visualised. The tangent linear solution $\mu \in \mathbb{R}^{n \times m}$ requires the solution of m linear systems, while the adjoint solution $\lambda \in \mathbb{R}^n$ requires only one linear solve

1.4.2 The infinite dimensional case

Let us return to the case where the function spaces in (1.135) are infinite dimensional. For brevity we omit operator arguments that is we write $J \in \mathbb{R}$ instead of $J(u, m) \in \mathbb{R}$ and $F \in V'^*$ instead of $F(u, m; \cdot) \in V'^*$.

Taking the derivative of Eq. (1.136) and applying the chain rule on its right hand side yields:

$$dJ_m = \partial J_u du_m + \partial J_m \in M^*. \quad (1.142)$$

Here we assumed that the solution operator $u(m)$ is continuously Fréchet differentiable, which follows if J and F are continuously Fréchet differentiable and if the linearised PDE operator $\partial F_u(u(m), m; \cdot, \cdot)$ is invertible for any $m \in M$, see [19, Sect. 1.4.2] and [19, Sect. 1.6]. If Eq. (1.142) was used directly to compute the functional gradient, all three terms on the right hand side would be needed. For the terms involving J this is straightforward: the functional of interest is typically given explicitly and its partial derivatives can therefore be readily derived. However, $du_m \in \mathcal{L}(M, V)$ is typically not known explicitly, since u is the solution of the PDE.

The state equation (1.135) provides us a way to compute du_m : Since $F(u(m), m) = 0$ for all $m \in M$, also its derivative $d_m F(u(m), m)$ must be zero. Applying the chain rule on the left hand side yields:

$$\partial F_u du_m + \partial F_m = 0 \in M^*. \quad (1.143)$$

Since it was assumed that ∂F_u is invertible, we can substitute (1.143) into (1.142) to obtain an equation for the functional gradient:

$$dJ_m = -\overbrace{\partial J_u \partial F_u^{-1} \partial F_m}^{=\lambda^*} + \partial J_m. \quad (1.144)$$

$=\mu$

The order in which the right hand side is evaluated leads to two different approaches: one involves solving the *tangent linear model* for μ , and the other one solves the *adjoint model* for $\lambda \in V'$.

1.4.2.1 The tangent linear approach

The tangent linear approach first solves the weak tangent linear model for $\mu \in \mathcal{L}(M, V)$:

$$\partial F_u(u, m; v, \mu) = \partial F_m(u, m; v) \quad \forall v \in V', \quad (1.145)$$

and then uses (1.144) to compute the functional derivative with

$$dJ_m(u, m; \cdot) = -\partial J_u(u, m; \mu(\cdot)) + \partial J_m(u, m; \cdot).$$

Note that the linearised forward model (1.143) and the tangent linear model (1.145) are equivalent apart from the sign, and hence $\mu = -du_m$.

1.4.2.2 The adjoint approach

The adjoint approach solves the equation: Find $\lambda \in V'$ such that:

$$\partial F_u^*(u, m; v, \lambda) = \partial J_u(u, m; v) \quad \forall v \in V. \quad (1.146)$$

or equivalently,

$$\partial F_u(u, m; \lambda, v) = \partial J_u(u, m; v) \quad \forall v \in V. \quad (1.147)$$

Here, the adjoint equation is stated in the familiar form of a variational problem, with a bilinear operator on the left hand side and a linear operator on the right hand side and can be solved with the finite element method. The adjoint operator is based on the linearisation of the original PDE operator, so its computational cost is expected to be equal or less than that of the original PDE.

Once the adjoint solution has been computed, the functional derivative is obtained with

$$dJ_m(u, m; \cdot) = -\partial F_m(u, m; \cdot, \lambda) + \partial J_m(u, m; \cdot). \quad (1.148)$$

1.4.2.3 Alternative approaches

There exist alternative approaches to compute the functional gradient. Two common methods are the finite difference and the complex-step approaches which will be briefly discussed in this section.

The finite difference approach approximates the functional gradient in the direction δm by computing the difference quotient. The central difference formula for example yields:

$$d\tilde{J}_m(m; \delta m) = \frac{\tilde{J}(m + h\delta m) - \tilde{J}(m - h\delta m)}{2h} + \mathcal{O}(|h|^2) \quad \text{as } h \rightarrow 0.$$

The main advantage of the finite difference approach is its easy implementation: the model can be treated as a black box since only functional evaluations are required. However, the determination of a suitable step length h can be difficult: from a mathematical perspective, h should be chosen as small as possible to improve the approximation, but due to numerical cancellation the smallest suitable h value is bounded by round-off errors [27, pp. 166–169].

The complex-step approach [24] avoids this problem by using complex calculus. It considers the reduced functional in the complex plane and uses the Cauchy-Riemann equations to derive following approximation of the directional derivative:

$$d\tilde{J}_m(m; \delta m) = \frac{\text{Im}(\tilde{J}(m + ih\delta m))}{h} + \mathcal{O}(|h|^2) \quad \text{as } h \rightarrow 0.$$

Since no difference operation is performed, this evaluation is not subject to subtractive cancellation errors. From an implementation perspective the complex-step approach is more intrusive, since the underlying code must be modified to support complex numbers.

Both the finite difference and the complex step approach only yield the derivatives in a particular direction. To obtain the full functional derivative, the directional derivatives for all basis functions in the parameter space must be computed separately. As a consequence the computational complexity increases linearly with the dimension of M . Nevertheless, the finite difference method is popular due to its straightforward implementation and is a useful verification tool.

1.4.3 Higher-order derivatives

It is possible to compute higher-order derivatives by recursively applying the adjoint or tangent linear approach. One application of second-order information is in the context of optimisation with PDE constraints, for example for design optimisation [16] and optimal control [18, 30]. For brevity we only state a short derivation of the second order adjoint equations here. A more detailed derivation of second-order adjoints can be found for example in [15, Sect. 13.4], [19, Sect. 1.6.5], [36, Sect. 4.6], and [5].

The action of the Hessian H is the transpose of the total derivative of the first derivative in a particular direction δm . Applying the total derivative to (1.148) and omitting the arguments for brevity, we find:

$$\begin{aligned} (H\delta m)^* &= d(dJ_m)_m \delta m = -\dot{\lambda}^* \partial F_m \\ &\quad - \lambda^* d(\partial F_m)_m \delta m \\ &\quad + d(\partial J_m)_m \delta m, \end{aligned} \quad (1.149)$$

where

$$\dot{\lambda} = d\lambda_m \delta m \quad (1.150)$$

is the tangent linearisation of the adjoint solution λ in the direction δm . $\dot{\lambda}$ denotes the second order adjoint solution, which is derived by taking the total derivative of the adjoint equation (1.147):

$$\begin{aligned} \partial F_u^* \dot{\lambda} &= (\partial^2 J_{uu} \dot{u})^* + (\partial^2 J_{mu} \delta m)^* \\ &\quad - (\partial^2 F_{uu} \dot{u})^* \lambda - (\partial^2 F_{mu} \delta m)^* \lambda. \end{aligned}$$

Evaluating the directional Hessian is computationally quite more expensive than a gradient evaluation. In particular, to compute the functional, only the forward PDE needs to be solved. To compute the functional's gradient, the forward and adjoint PDEs need to be solved. However, to compute the Hessian in one direction, the forward and adjoint PDEs, a tangent linear PDE and a second order adjoint PDE

need to be solved. This comparable high cost will play an important role for the design of optimisation algorithms in section 1.5.

1.4.4 Example: Adjoint Poisson's equation

Recall from (1.12) the variational form of the Poisson's equation with pure Neumann conditions: find $u \in V = H^1(\Omega)$ such that

$$\int_{\Omega} \nabla u \cdot \nabla v \, dx - \int_{\partial\Omega} g_N v \, ds = \int_{\Omega} f v \, dx \quad \forall v \in V' = H^1(\Omega). \quad (1.151)$$

with $f \in L^2(\Omega)$ and together with the functional $\frac{1}{2} \int_{\Omega} u \cdot u \, dx + \frac{\alpha}{2} \int_{\Omega} f^2 \, dx$ with $\alpha \geq 0$.

We can write this problem in the canonical form (1.135) by defining following operators:

$$F(u, f; v) = \int_{\Omega} \nabla u \cdot \nabla v \, dx - \int_{\partial\Omega} g_N v \, ds - \int_{\Omega} f v \, dx, \quad (1.152)$$

$$J(u, f) = \frac{1}{2} \int_{\Omega} u \cdot u \, dx + \frac{\alpha}{2} \int_{\Omega} f^2 \, dx. \quad (1.153)$$

For the operators in the adjoint equation (1.147) we obtain

$$\partial F_u^*(u, m; v, \lambda) = \partial F_u(u, m; \lambda, v) = \int_{\Omega} \nabla v \cdot \nabla \lambda \, dx, \quad (1.154)$$

$$\partial J_u(u, f; v) = \int_{\Omega} u v \, dx. \quad (1.155)$$

Hence, the adjoint Poisson's equation for the functional is: Find $\lambda \in V'$ such that

$$\int_{\Omega} \nabla v \cdot \nabla \lambda \, dx = \int_{\Omega} u \cdot v \, dx \quad \forall v \in V, \quad (1.156)$$

or in strong form:

$$-\Delta \lambda = u \quad \text{in } \Omega, \quad (1.157)$$

$$\frac{\partial \lambda}{\partial n} = 0 \quad \text{on } \partial\Omega. \quad (1.158)$$

Given λ , the functional derivative can be evaluated as:

$$dJ_f(u, f; \cdot) = -\partial F_f(u, f; \cdot, \lambda) + \partial J_f(u, f; \cdot) = \int_{\Omega} (\lambda + \alpha f) \cdot dx. \quad (1.159)$$

Similarly, we could also evaluate the functional derivative with respect to the Neumann boundary values:

$$dJ_{g_N} = -\partial F_{g_N} + \partial J_{g_N} = \int_{\partial\Omega} \lambda \cdot ds. \quad (1.160)$$

1.4.5 Example: Adjoint Burgers' equation

Next we derive the adjoint equation of the time-dependent viscous Burgers' equation with Neumann boundary conditions. The weak formulation of the Burgers' equation was derived in Sect. 1.1.2.1: find $u \in V = H^1(\Omega)^2$ such that

$$\int_{\Omega} \left(\frac{\partial u}{\partial t} \cdot v + (u \cdot \nabla) u \cdot v + \eta \nabla u : \nabla v \right) dx = 0 \quad \forall v \in V' = H^1(\Omega)^2. \quad (1.161)$$

At this point we have two choices. Either we discretise the equation in time as in section 1.1.2.2, and derive the adjoint from the semi-discretised equation. Or derive the adjoint equation directly from the non-discretised form (1.161). We follow the latter strategy and extend the variational formulation such that our test and trial spaces are functions in time and space. Consequently, we extend the variational form by a time-integral and incorporate the initial condition into the variational formulation: Find $u \in L^2(0, T; V) \cap H^1(0, T; V^*)$ such that

$$\begin{aligned} & \int_0^T \int_{\Omega} \left(\frac{\partial u}{\partial t} \cdot v + (u \cdot \nabla) u \cdot v + \eta \nabla u : \nabla v \right) dx dt \\ & + \int_{\Omega} (u(0) - u_0) \cdot v(0) dx = 0 \quad \forall v \in L^2(0, T; V') \cap H^1(0, T; V'^*). \end{aligned}$$

The adjoint PDE operator is obtained by linearising in direction λ , and forming its transpose by swapping test and trial functions:

$$\int_0^T \int_{\Omega} \left(\frac{\partial v}{\partial t} \cdot \lambda + (v \cdot \nabla) u \cdot \lambda + (u \cdot \nabla) v \cdot \lambda + \eta \nabla v : \nabla \lambda \right) dx dt + \int_{\Omega} \lambda(0) \cdot v(0) dx. \quad (1.162)$$

Integrating the time-derivative term by parts yields:

$$\int_0^T \int_{\Omega} \left(-\frac{\partial \lambda}{\partial t} \cdot v + (v \cdot \nabla) u \cdot \lambda + (u \cdot \nabla) v \cdot \lambda + \eta \nabla v : \nabla \lambda \right) dx dt + \int_{\Omega} \lambda(T) \cdot v(T) dx \quad (1.163)$$

Note that the integration by parts results in the term $-\int_{\Omega} \lambda(0) \cdot v(0) dx$ which cancels with the last term in (1.162). (1.163) is the weak adjoint operator for the Burger's equation and can be discretised and solved with the finite element method. The strong form can also be obtained through integration by parts. For a functional $J(u, m)$ one obtains:

$$\begin{aligned}
-\frac{\partial \lambda}{\partial t} - (u \cdot \nabla) \lambda + (\nabla u)^* \lambda - \eta \Delta \lambda &= \partial J_u^* \quad \text{in } [0, T] \times \Omega, \\
(n \cdot \nabla) u &= 0 \quad \text{on } [0, T] \times \partial \Omega, \\
\lambda &= 0 \quad \text{on } \{T\} \times \Omega.
\end{aligned} \tag{1.164}$$

The adjoint of the Burgers' equation (1.164) reveals some general properties about the adjoint of a time-dependent problem: the resulting PDE is linear with initial conditions at the end of the time interval. Consequently it is solved backwards in time. Furthermore, the adjoint PDE depends on the forward solution u , and therefore the forward model must be solved beforehand.

The derivation and implementation of the adjoint system can be laborious for complex PDEs. To avoid the need for explicit adjoint derivations, special software tools have been developed that aim automate the derivation process.

1.4.6 Taylor tests for adjoint implementations

Verification, or correctness testing, is an essential component of the implementation of any numerical algorithm on a computer. This is particularly true for the technically complex software required to implement the numerical solver for an adjoint PDE. A Taylor test computes approximate convergence rates of Taylor remainders which can be used to compare to theoretical convergence rates based on Taylor's theorem. Solving the adjoint PDE is a mechanism for evaluating the Fréchet derivative of the reduced functional with respect to the control parameters, so the applicable form of Taylor's theorem is that given in Sect. 1.2.

Recall that $\tilde{J}(m) := J(u(m), m)$ is the reduced functional and let δm be an arbitrary perturbation of the control m . Applying Taylor's theorem to this situation, we find for $\mathbb{R} \ni h \rightarrow 0$

$$\left| \tilde{J}(m + h\delta m) - \tilde{J}(m) \right| = \mathcal{O}(h), \text{ and} \tag{1.165}$$

$$\left| \tilde{J}(m + h\delta m) - \tilde{J}(m) - h d\tilde{J}(m; \delta m) \right| = \mathcal{O}(h^2). \tag{1.166}$$

The expression in (1.165) is referred to as the first-order Taylor remainder while that in (1.166) is called the second-order Taylor remainder.

Let us consider the situation where computer implementations of $\tilde{J}$ and $d\tilde{J}$ are given and it is claimed that $d\tilde{J}$ correctly implements the Fréchet derivative of $\tilde{J}$. One may numerically test this claim by computing the second-order Taylor remainder for a fixed but arbitrary choice of δm , and an exponentially decreasing sequence of values of h . If the implementation is correct then for sufficiently small h , the Taylor remainder will decrease as $\mathcal{O}(h^2)$.

Similarly, one can employ higher order Taylor remainders to test implementations of higher order Fréchet derivatives. For example the third-order remainder can be used to test implementations of the second derivative:

$$\left| \tilde{J}(m + h\delta m) - \tilde{J}(m) - h d\tilde{J}(m; \delta m) - \frac{1}{2} h^2 d^{(2)}\tilde{J}(m; \delta m, \delta m) \right| = \mathcal{O}(h^3). \quad (1.167)$$

Clearly in this case the expected convergence rate is $\mathcal{O}(h^3)$. Since the Taylor test for the second derivative requires a correctly implemented first derivative, it is highly advisable to first conduct a Taylor test of the first derivative.

1.5 Optimisation methods

The preceding sections introduced the theory and practice of solving PDE problems using the finite element method, and the efficient computation of derivatives of functionals of the associated PDE solutions. On this basis it is now possible to introduce algorithms which employ these solution techniques and the resulting gradient information to solve PDE-constrained optimisation problems. Here we will introduce several iterative approaches which are well-known in the optimisation community. In contrast with most introductory works, we will formulate these algorithms in a general Hilbert space, rather than assuming that the optimisation problem is set in Euclidean space.

1.5.1 Steepest descent method

The steepest descent method is an optimisation algorithm which is based upon the observation that a differentiable function decreases fastest in the proximity of the current iterate in the direction of its negative gradient. This principle naturally generalises to functionals. As before, we formulate the optimisation problem in terms of the reduced functional:

$$\min_{m \in M} \tilde{J}(m) \quad (1.168)$$

where $\tilde{J} : M \rightarrow \mathbb{R}$ is the reduced functional defined over a control parameter drawn from a Hilbert space M . For any sufficiently small $\alpha > 0$ and with $m' := m - \alpha \mathcal{R}(d\tilde{J}(m; \cdot))$ for any $m \in M$ not a local minimiser of $\tilde{J}$, we obtain $\tilde{J}(m') < \tilde{J}(m)$. Recall that $\mathcal{R}(d\tilde{J}(m; \cdot)) \in M$ is the Riesz representer of the the Fréchet derivative of $\tilde{J}$ at point m . Iterating this approach yields

$$m_{k+1} = m_k - \alpha_k \mathcal{R}(d\tilde{J}(m_k; \cdot)) \quad \forall k \in \mathbb{N}_0, \quad (1.169)$$

such that $\tilde{J}(m_{k+1}) \leq \tilde{J}(m_k)$.

If $d\tilde{J} : M \rightarrow M^*$ is uniformly continuous on bounded sets and $\tilde{J}$ is uniformly convex, there is a unique global minimum. If further α_k satisfies the so called Wolfe conditions, the sequence $\{m_k\}$ converges strongly to the global minimum [12].

In Eq. (1.169), there are several possible choices for the step size length α_k . The simplest is $\alpha_k = \alpha \forall k \in \mathbb{N}_0$ for a sufficiently small, fixed $\alpha > 0$. Another choice, which is introduced next, is to apply a so called exact line search.

The Taylor series of $\tilde{J}(m_{k+1})$ around the point m_k is given by

$$\tilde{J}(m_{k+1}) = \tilde{J}(m_k + \alpha_k \mathcal{R}(\mathrm{d}\tilde{J}(m_k; \cdot))) \quad (1.170)$$

$$\begin{aligned} &= \tilde{J}(m_k) - \alpha_k \mathrm{d}\tilde{J}(m_k; \mathcal{R}(\mathrm{d}\tilde{J}(m_k; \cdot))) \\ &\quad + \frac{\alpha_k^2}{2} \mathrm{d}\tilde{J}^{(2)}(m_k; \mathcal{R}(\mathrm{d}\tilde{J}(m_k; \cdot)), \mathcal{R}(\mathrm{d}\tilde{J}(m_k; \cdot))) \\ &\quad + \alpha_k^3 \mathcal{O}(\|\mathcal{R}(\mathrm{d}\tilde{J}(m_k; \cdot))\|_M^3). \end{aligned} \quad (1.171)$$

Note that the last term on the right hand side vanishes in the case where f is quadratic, i.e. $\tilde{J}^{(n)} \equiv 0$ for any $n > 2$. Equivalently, simply dropping the error term results in a quadratic approximation to $\tilde{J}(m_{k+1})$. Differentiating this approximation with respect to α_k and setting the result equal to zero leads to

$$\hat{\alpha}_k = \frac{\mathrm{d}\tilde{J}(m_k; \mathcal{R}(\mathrm{d}\tilde{J}(m_k; \cdot)))}{\mathrm{d}\tilde{J}^{(2)}(m_k; \mathcal{R}(\mathrm{d}\tilde{J}(m_k; \cdot)), \mathcal{R}(\mathrm{d}\tilde{J}(m_k; \cdot)))}. \quad (1.172)$$

Under suitable hypotheses, among others convexity, $\hat{\alpha}_k$ defines the unique step length which minimises the quadratic approximation of $\tilde{J}(m_{k+1})$. If $M = \mathbb{R}^n$, the positive definiteness of the Hessian of $\tilde{J}$ is sufficient. The iterative approach induced by the step length $\alpha_k = \hat{\alpha}_k$ is called exact line search. Algorithm 1 shows the pseudocode for the steepest descent method with an exact line search.

Algorithm 1: Steepest descent with exact line search in a Hilbert space M

Input: Initial m_0 , convergence tolerance $\varepsilon > 0$;
while $\|\mathcal{R}(\mathrm{d}\tilde{J}(m_k; \cdot))\|_M > \varepsilon$ **do**
 $\alpha_k \leftarrow \mathrm{d}\tilde{J}(m_k; \mathcal{R}(\mathrm{d}\tilde{J}(m_k; \cdot))) / \mathrm{d}\tilde{J}^{(2)}(m_k; \mathcal{R}(\mathrm{d}\tilde{J}(m_k; \cdot)), \mathcal{R}(\mathrm{d}\tilde{J}(m_k; \cdot)))$;
 $m_{k+1} \leftarrow m_k - \alpha_k \mathcal{R}(\mathrm{d}\tilde{J}(m_k; \cdot))$;
end

1.5.2 Inexact line search and the Wolfe conditions

The exact line search procedure requires the evaluation of the functional Hessian which, as is discussed below, may not be easily or affordably computable. Consequently, more heuristic approaches have been developed, along with appropriate criteria under which they generate convergent optimisation schemes.

All of the methods we will consider produce iterative updates to the control m , of the form:

$$m_{k+1} = m_k + \alpha_k p_k. \quad (1.173)$$

α_k , a scalar, is referred to as the *step length* while p_k , which lies in the same space as m , is the *search direction*. The Wolfe conditions [37] are conditions on α_k which, given suitable conditions on the functional to be minimised, are sufficient to show that the optimisation converges. A more complete discussion of the Wolfe conditions is to be found in [3].

The first condition is the sufficient decrease condition:

$$\tilde{J}(m_{k+1}) \leq \tilde{J}(m_k) + \lambda \alpha_k d\tilde{J}(m_k; p_k) \quad (1.174)$$

for a small parameter $\lambda > 0$. This condition, also called the Armijo condition, effectively prevents large steps which only cause small functional decreases. The second condition is the curvature condition:

$$|d\tilde{J}(m_{k+1}; p_k)| \leq \beta |d\tilde{J}(m_k; p_k)| \quad (1.175)$$

A popular and effective inexact line search strategy is to progressively decrease candidate α_k values until these conditions are met. This approach is known as backtracking.

1.5.3 Line search Newton-CG

As the name suggests, the Line search Newton conjugate gradient (Newton-CG) algorithm is Newton's method augmented by a line search for the update size, and employing the conjugate gradient method to invert the Hessian. Newton's method amounts to approximating the system to be optimised by the leading terms of the Taylor series and solving for the fixed point given by the Jacobian being zero:

$$0 = d\tilde{J}(m_{k+1}; \cdot) \quad (1.176)$$

$$= d\tilde{J}(m_k + p_k; \cdot) \quad (1.177)$$

$$\approx d\tilde{J}(m_k + p_k; \cdot) + d^{(2)}\tilde{J}(m_k; p_k, \cdot). \quad (1.178)$$

The procedure is to solve (1.178) for the search direction p_k and then to employ a line search to establish the step size.

The Newton-CG algorithm in $\mathbb{R}^n$ is given in [26, Sect. 7.1], while its generalisation to functionals on Hilbert spaces, achieved by carefully applying the correct inner products and norms, is given in algorithm 2. In this algorithm, the inner iteration sequence converges to the Newton step p_k in (1.178) and at each outer iteration, a tolerance $\varepsilon_k > 0$ for the accuracy of the computed p_k is specified. In the inner iteration, which corresponds to the conjugate gradient method, the search directions are denoted by d_j . The fallback case for negative curvature, $d^{(2)}\tilde{J}(m_k; d_j, d_j) \leq 0$, ensures that the search direction p_k is always descending.

Algorithm 2: Line Search Newton-CG in Hilbert space

```

Input: Initial  $m_0$  ;
for  $k = 0, 1, \dots$  do
    Define tolerance  $\varepsilon_k > 0$ ;
     $z_0 \leftarrow 0$ ;  $r_0 \leftarrow \mathcal{R}(\mathrm{d}\tilde{f}(m_k; \cdot))$ ;  $d_0 \leftarrow -r_0$ ;
    for  $j = 0, 1, \dots$  do
        if  $\mathrm{d}^{(2)}\tilde{f}(m_k; d_j, d_j) \leq 0$  then
            if  $j = 0$  then
                 $p_k \leftarrow -\mathcal{R}(\mathrm{d}\tilde{f}(m_k; \cdot))$ ;
                break;
            else
                 $p_k \leftarrow z_j$ ;
                break;
            end
        end
         $\beta_j \leftarrow \langle r_j, r_j \rangle_M / \mathrm{d}^{(2)}\tilde{f}(m_k; d_j, d_j)$ ;
         $z_{j+1} \leftarrow z_j + \beta_j d_j$ ;
         $r_{j+1} \leftarrow r_j + \beta_j \mathcal{R}(\mathrm{d}^{(2)}\tilde{f}(m_k; d_j, \cdot))$ ;
        if  $\|r_{j+1}\|_M < \varepsilon_k$  then
             $p_k \leftarrow z_{j+1}$ ;
            break;
        end
         $\gamma_{j+1} \leftarrow \langle r_{j+1}, r_{j+1} \rangle_M / \langle r_j, r_j \rangle_M$ ;
         $d_{j+1} \leftarrow -r_{j+1} + \gamma_{j+1} d_j$ ;
    end
    Compute  $\alpha_k$  using an appropriate line search;
     $m_{k+1} \leftarrow m_k + \alpha_k p_k$ ;
end

```

1.5.4 BFGS

Newton-CG and other Krylov-based methods require many evaluations of the exact Hessian of the objective functional. Recall that, for a PDE-constrained optimisation problem, this in turn requires composed solutions of a (potentially time-varying) adjoint problem. To address this issue, it may be possible to employ optimisation algorithms which avoid the need to evaluate the Hessian.

In its original, Euclidean formulation the Broyden-Fletcher-Goldfarb-Shanno method is an optimisation algorithm for problems of the type

$$\min_{x \in \mathbb{R}} f(x) \quad (1.179)$$

with $f : \mathbb{R}^n \rightarrow \mathbb{R}$.

The BFGS method is one of the most successful and applied algorithms in unconstrained nonlinear optimisation [26]. As a quasi-Newton method, it does not require evaluations of the Hessian of the objective function, but only of the gradient and of the objective function itself. An approximation for the Hessian is computed using

the difference between successive iterates and the corresponding gradient vectors. In this respect, quasi-Newton methods can be understood as a generalisation of the secant method for finding roots of first derivatives. One advantage of the Hessian approximation is that it is always positive definite while the Hessian itself might merely be positive semi-definite and therefore not invertible.

In a manner analogous with Newton's method, we consider a quadratic model for the objective function. At a current iterate $x_k \in \mathbb{R}^n$, this model is given by:

$$q_k(x) = f(x_k) + \nabla f(x_k) \cdot (x - x_k) + \frac{1}{2}(x - x_k)^T \cdot B_k \cdot (x - x_k), \quad (1.180)$$

where $B_k \in \mathbb{R}^{n \times n}$ denotes the approximation of the Hessian of f evaluated at m_k . Since q_k is convex, one easily computes the unique minimum in the substituted variable $p_k = x - x_k$ as

$$p_k = -B_k^{-1} \cdot \nabla f(x_k). \quad (1.181)$$

Taking (1.181) as the line search direction, one may proceed as in Newton's method to compute the next iterate x_{k+1} by

$$x_{k+1} = x_k + \alpha_k p_k,$$

where $\alpha_k > 0$ denotes the step length satisfying sufficient decrease and curvature conditions given by the Wolfe conditons [26, Sect. 3.1]. The major difference from Newton's method is that B_k is an approximation to the Hessian and not the Hessian itself. The update of the quadratic model with $k \rightarrow k+1$ is achieved by updating the Hessian approximation $B_k \rightarrow B_{k+1}$. In fact, the BFGS algorithm explicitly constructs and updates $H_k \equiv B_k^{-1}$. This avoids the need to solve the linear system in (1.181) at each iteration.

How, then, do we construct and update H_k ? It is a direct consequence of Taylor's theorem in $\mathbb{R}^n$ that

$$\nabla^2 f(x_k) \cdot (x_{k+1} - x_k) \approx \nabla f(x_{k+1}) - \nabla f(x_k) \quad \text{for } \|x_{k+1} - x_k\| \ll 1, \quad (1.182)$$

where $\nabla^2 f(x_k) \in \mathbb{R}^{n \times n}$ denotes the Hessian of f evaluated at x_k . We impose the analagous condition on B_{k+1} resulting in the so called secant equation,

$$B_{k+1} \cdot (x_{k+1} - x_k) = \nabla f(x_{k+1}) - \nabla f(x_k), \quad (1.183)$$

or, equivalently,

$$H_{k+1} \cdot (\nabla f(x_{k+1}) - \nabla f(x_k)) = x_{k+1} - x_k, \quad (1.184)$$

The Wolfe conditions on α_k are sufficient to guarantee a solution to (1.184) [26, Sect. 6.1].

Choosing H_{k+1} as the closest matrix H to H_k with respect to a weighted Frobenius norm under the constraints that H is symmetric and satisfies (1.184) yields the

unique solution:

$$H_{k+1} = V_k^T H_k V_k + \rho_k s_k s_k^T, \quad (1.185)$$

where

$$s_k = x_{k+1} - x_k = \alpha_k p_k, \quad (1.186)$$

$$y_k = \nabla f(x_{k+1}) - \nabla f(x_k), \quad (1.187)$$

$$\rho_k = \frac{1}{y_k^T s_k}, \quad (1.188)$$

$$V_k = \mathbf{I} - \rho_k y_k s_k^T. \quad (1.189)$$

One could also define a BFGS algorithm based on B_k instead of H_k . Using the Sherman-Morrison-Woodbury formula [26, Appendix] applied to (1.185), we obtain for the Hessian approximation B_{k+1} :

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}. \quad (1.190)$$

We refer to the Eqs. (1.185) and (1.190) as the BFGS updating formulae. Observe that if B_k is symmetric then the update formula guarantees that B_{k+1} will be too. For more details of the derivations above and for algorithm 3, we refer to [26, Chap. 6].

Algorithm 3: BFGS in $\mathbb{R}^n$

Input: Initial x_0 , convergence tolerances $\varepsilon_\nabla, \varepsilon > 0$, initial inverse Hessian approximation H_0 ;

$k \leftarrow 0$

while $\|\nabla f_k\| > \varepsilon_\nabla$ *and* $(f_k - f_{k+1})/\max(|f_{k+1}|, |f_k|, 1) > \varepsilon$ **do**

$p_k \leftarrow -H_k \nabla f_k$;

 Compute α_k using an appropriate line search;

$s_k \leftarrow \alpha_k p_k$;

$x_{k+1} \leftarrow x_k + s_k$;

$y_k \leftarrow \nabla f_{k+1} - \nabla f_k$;

 Compute H_{k+1} according to (1.185);

$k \leftarrow k + 1$

end

1.5.4.1 BFGS in Hilbert spaces

We now return to the problem of minimising the reduced functional:

$$\min_{m \in M} \tilde{J}(m) \quad (1.191)$$

where M is a Hilbert space with inner product $\langle \cdot, \cdot \rangle_M$ and $\tilde{J}$ is assumed to be a twice continuously Fréchet-differentiable real-valued functional. The extension of the BFGS method to this situation simply requires careful attention to the appropriate application of this inner product. The quadratic model now becomes:

$$q_k(p) = \tilde{J}_k + y_k(p) + \frac{1}{2}B_k(p, p), \quad (1.192)$$

where $B_k \in \mathcal{L}(M, M^*)$ is invertible and $y_k = d\tilde{J}(m_k) \in M^*$. Note that we can consider functions in $\mathcal{L}(M, M^*)$ to have either one argument in M and return values in M^* , or to have two arguments in M and return values in $\mathbb{R}$. We will use whichever version is most applicable in a given circumstance. The linear and continuous operator B_k is an approximation of the second Fréchet derivative of $\tilde{J}$. In order to find a p_k such that

$$q_k(p_k) \leq q_k(p) \quad \forall p \in M, \quad (1.193)$$

it is necessary that

$$dq_k(p_k; \cdot) = y_k(\cdot) + B_k(p_k, \cdot) = 0. \quad (1.194)$$

This is typically referred to as the first order optimality condition. Therefore, the search direction we require is

$$p_k = -B_k^{-1}(y_k), \quad (1.195)$$

where $B_k^{-1} \in \mathcal{L}(M^*, M)$. The linearity and boundedness of B_k^{-1} can be demonstrated as follows. For any $m, n \in M$, there are $m^*, n^* \in M^*$ such that $B_k^{-1}m^* = m$ and $B_k^{-1}n^* = n$. Using the linearity of B_k yields

$$B_k(m + \gamma n) = B_k m + \gamma B_k n \quad (1.196)$$

$$\Leftrightarrow B_k(B_k^{-1}m^* + \gamma B_k^{-1}n^*) = m^* + \gamma n^* \quad \forall \gamma \in \mathbb{R}. \quad (1.197)$$

Applying B_k^{-1} from the left hand side shows that B_k^{-1} is linear. Further, the open mapping theorem states that any surjective bounded linear operator between two Banach spaces is an open map, i.e. maps open sets onto open sets. Thus, for any $U \subset M$ open we have $(B_k^{-1})^{-1}(U) = B_k(U)$ open in M^* . Hence, B_k^{-1} is continuous.

A symmetric rank two update that meets the secant condition is the Hilbert version of the BFGS formula (1.190), given by

$$B_{k+1}(m, n) = B_k(m, n) - \frac{B_k(s_k, m)B_k(s_k, n)}{B_k(s_k, s_k)} + \frac{y_k(m)y_k(n)}{y_k(s_k)}, \quad (1.198)$$

where the function arguments are $m, n \in M$. The updating formula corresponding to the approximation H_{k+1} of the Hessian inverse in (1.185) is given by

$$H_{k+1}(m^*) = H_k(m^*) - \frac{m^*(s_k)}{y_k(s_k)} H_k(y_k) - \frac{y_k(H_k(m^*))}{y_k(s_k)} s_k \quad (1.199)$$

$$+ \frac{m^*(s_k) y_k(H_k(y_k))}{(y_k(s_k))^2} s_k + m^*(s_k) s_k \quad (1.200)$$

$$= \left(\text{Id}_M - \frac{s_k}{y_k(s_k)} y_k \right) \left(H_k(m^*) - H_k(y_k) \frac{m^*(s_k)}{y_k(s_k)} \right) + m^*(s_k) s_k \quad (1.201)$$

$$= \left(\text{Id}_M - \frac{s_k}{y_k s_k} y_k \right) \left(H_k \left(m^* - \frac{m^*(s_k)}{y_k(s_k)} y_k \right) \right) + m^*(s_k) s_k \quad (1.202)$$

$$= \mathcal{V}_k(H_k(\mathcal{V}_k^*(m^*))) + m^*(s_k) s_k, \quad (1.203)$$

where $m^* \in M^*$ arbitrary and $H_k \in \mathcal{L}(M^*, M)$. Here, $\text{Id}_M : M \rightarrow M$ is the identity operator on the space M . Further, $\mathcal{V}_k : \mathcal{L}(M, M)$ and $\mathcal{V}_k^* : \mathcal{L}(M^*, M^*)$ are given by

$$\mathcal{V}_k(m) = m - \frac{s_k}{y_k(s_k)} y_k(m), \quad (1.204)$$

$$\mathcal{V}_k^*(m^*) = m^* - \frac{m^*(s_k)}{y_k(s_k)} y_k, \quad (1.205)$$

where $m \in M$ and $m^* \in M^*$ arbitrary.

Using the Riesz representation theorem, formula (1.198) can be written as

$$B_{k+1}(m, n) = B_k(m, n) - \frac{\langle \mathcal{R}(B_k(s_k)), m \rangle_M \langle \mathcal{R}(B_k(s_k)), n \rangle_M}{\langle \mathcal{R}(B_k s_k), s_k \rangle_M} + \frac{\langle \mathcal{R}(y_k), m \rangle_M \langle \mathcal{R}(y_k), n \rangle_H}{\langle \mathcal{R}(y_k), s_k \rangle_M}. \quad (1.206)$$

Of course we are primarily interested in the case where H is finite-dimensional. In the particular case where $M = \mathbb{R}^n$, we have $\langle a, b \rangle_M = a^T b$ and $\mathcal{R}(a) = a^T$. In this special case (1.206) is equivalent to (1.190). In contrast, if M is a finite dimensional subspace of, for example, H^1 or L^2 then (1.206) demands that the corresponding inner product is employed.

The question arises which algorithm among all Hilbert space BFGS versions, or equivalently, what inner product to choose for numerically solving an optimisation problem. Letting ourselves lead by the structure of the problem, it is natural to choose the inner product space that corresponds to the space of the optimisation controls. The same space should be the basis for any finite-dimensional numerical method.

It turns out that optimisation algorithms that do not respect the natural and correct inner product of the control space lead to mesh-dependent and generally suboptimal solutions. Chapt. 2 and Chap. 3 are dedicated to a deeper discussion of these phenomena.

1.5.4.2 Limited memory Hessian approximation

The inverse Hessian approximation, H_k , introduced in the previous section for $\mathbb{R}^n$ is a dense $n \times n$ matrix. The storage of this matrix is impractical for all but the smallest problems. Fortunately, this is also unnecessary since only the action of this matrix is required. Instead, the sequence of vectors y_k and s_k can be stored, and $-H_k \nabla f_k$ computed on the fly using the recursion formula for H_k without ever constructing the matrix. Of course this still requires an increasingly large number of vectors to be stored and the computation of the matrix action becomes increasingly expensive. In contrast, [3] present an approach in which only information from the most recent iterations is stored. The justification for this approach, which is similar in character to restarted GMRES, is that the vectors essentially encode curvature information, and that the curvature information from the most recent iterations is likely to have the greatest impact on the Hessian behaviour of the current iteration [26, Sect. 7.2].

By repeated application of the BFGS formula (1.185), one can easily see that H_k is given recursively by

$$\begin{aligned}
 H_k = & (V_{k-1}^T \cdots V_{k-m}^T) H_{k-m} (V_{k-m} \cdots V_{k-1}) \\
 & + \rho_{k-m} (V_{k-1}^T \cdots V_{k-m+1}^T) s_{k-m} s_{k-m}^T (V_{k-m+1} \cdots V_{k-1}) \\
 & + \rho_{k-m+1} (V_{k-1}^T \cdots V_{k-m+2}^T) s_{k-m+1} s_{k-m+1}^T (V_{k-m+2} \cdots V_{k-1}) \\
 & \vdots \\
 & + \rho_{k-2} V_{k-1}^T s_{k-2} s_{k-2}^T V_{k-1} \\
 & + \rho_{k-1} s_{k-1} s_{k-1}^T,
 \end{aligned} \tag{1.207}$$

where $1 \leq m \leq k$ and H_{k-m} is the approximation of the Hessian inverse at the $(k-m)$ th iterate. The main idea of the limited BFGS method is now to replace H_{k-m} in (1.207) by some initialisation matrix $\gamma_k \mathbf{I}$ with $\gamma_k > 0$. The reader is referred to [3, Sect. 3] and [4, Sect. 3], for a more compact representation of H_k under further curvature conditions.

In a Hilbert space setting, (1.207) becomes

$$\begin{aligned}
H_k(m^*) &= (\mathcal{V}_{k-1} \circ \dots \circ \mathcal{V}_{k-m}) (H_{k-m}(\mathcal{V}_{k-m}^* \circ \dots \circ \mathcal{V}_{k-1}^*(m^*))) \\
&+ \frac{\mathcal{V}_{k-1} \circ \dots \circ \mathcal{V}_{k-m+1}}{y_{k-m} s_{k-m}} (s_{k-m} \cdot (\mathcal{V}_{k-m+1}^* \circ \dots \circ \mathcal{V}_{k-1}^*(m^*))(s_{k-m})) \\
&+ \frac{\mathcal{V}_{k-1} \circ \dots \circ \mathcal{V}_{k-m+2}}{y_{k-m+1} s_{k-m+1}} (s_{k-m+1} \cdot (\mathcal{V}_{k-m+2}^* \circ \dots \circ \mathcal{V}_{k-1}^*(m^*))(s_{k-m+1})) \\
&\vdots \\
&+ \frac{\mathcal{V}_{k-1}}{y_{k-2} s_{k-2}} (s_{k-2} \cdot \mathcal{V}_{k-1}^*(m^*)(s_{k-2})) \\
&+ \frac{s_{k-1} m^*(s_{k-1})}{y_{k-1} s_{k-2}},
\end{aligned} \tag{1.208}$$

where $m^* \in M^*$ arbitrary.

1.5.5 Primal log-barrier interior point method

The concept of barrier methods for constrained optimisation relies on reformulating the constrained problem as a sequence of unconstrained problems. In that, a composite function is minimised which accounts both for the original objective function as well as the constraints. The interior point method used in 3.5 is closely related to the classical barrier interior point method summarised in [38], which is introduced in what follows. Consider the following inequality constrained nonlinear optimisation problem

$$\begin{aligned}
&\min_{m \in M} \tilde{J}(m) \\
&\text{subject to } c_i(m) \geq 0,
\end{aligned} \tag{1.209}$$

where $\tilde{J}: M \rightarrow \mathbb{R}$ and $c_i: M \rightarrow \mathbb{R}$ for $i = 1, \dots, n$. The optimisation method introduced in the following aims on solving (1.209) iteratively by requiring their iterates to belong to the interior of the feasible region, i.e. to satisfy the inequality constraints strictly. For (1.209), we can define the so called logarithmic barrier functional $\beta^\mu: M \rightarrow \mathbb{R}$ by

$$\beta^\mu(m) = \tilde{J}(m) - \mu \sum_{i=1}^n \log(c_i(m)), \tag{1.210}$$

where $\mu > 0$ is referred to as the barrier parameter. Note that the logarithmic terms are well-defined for $c_i(m) > 0$ but tend to infinity for $c_i(m)$ approaching zero. Intuitively, for small μ the minimum of β^μ is close to a solution of (1.209). In the case $H = \mathbb{R}^n$ and under sufficient optimality conditions on a minimiser $\hat{m}$ of (1.209), one can show ([38]) that for a monotonically decreasing sequence of sufficiently

small values of μ , there is a sequence m_μ of unconstrained minimisers of the barrier function (1.210) with

$$\lim_{\mu \rightarrow 0} m_\mu = \hat{m}. \quad (1.211)$$

Under appropriate smoothness assumptions, the sequence $\{x_\mu\}$ of points define a smooth curve and converges non-tangentially from the strict interior of the feasible region to the minimiser $\hat{m}$ [38].

The Fréchet derivative of the logarithmic barrier functional with respect to $m \in H$ is given by

$$d\beta^\mu(m; \cdot) = d\tilde{J}(m; \cdot) - \sum_{i=1}^n \frac{\mu}{c_i(m)} dc_i(m; \cdot). \quad (1.212)$$

Using the product rule for differentiation, the barrier Hessian is computed as

$$d^{(2)}\beta^\mu(m; \cdot, \cdot) = d^{(2)}\tilde{J}(m; \cdot, \cdot) - \sum_{i=1}^n \frac{\mu}{c_i(m)} d^{(2)}c_i(m; \cdot, \cdot) \quad (1.213)$$

$$+ \sum_{i=1}^n \frac{\mu}{c_i^2(m)} dc_i(m, \cdot) \cdot dc_i(m, \cdot) \quad (1.214)$$

To compute the step $p_k \in H$ to move from a current iterate m_μ^k to m_μ^{k+1} , i.e.

$$m_\mu^{k+1} = m_\mu^k + p_k, \quad (1.215)$$

we apply Newton's method to the quadratic model of the barrier function. This leads to solving the so called primal Newton barrier equation

$$d^{(2)}\beta^\mu(m; p_k, \cdot) = -d\tilde{J}(m; \cdot) + \sum_{i=1}^n \frac{\mu}{c_i(m)} dc_i(m; \cdot). \quad (1.216)$$

Iterating this leads to an approximation of a stationary point of the barrier function (1.210). In the case $H = \mathbb{R}^n$ and under suitable differentiability assumption on β^μ , it can be shown that $\{m_\mu^k\}$ indeed converges to a stationary point and thus satisfies the necessary conditions to minimise the barrier function.

1.6 Optimal control of the Poisson equation

In this section, the principle of PDE-constrained optimisation is illustrated with a simple and generic example. The problem chosen is the optimal control of a system constrained by the Poisson equation. Physically, this problem can be interpreted as finding the ideal heat source in order to translate the state of the system into a desired temperature profile. Mathematically, the problem is to minimise the following

tracking type functional

$$\min_{m \in M} \frac{1}{2} \|u - d\|_{L^2(\Omega)}^2 + \frac{\alpha}{2} \|m\|_M^2, \quad (1.217)$$

subject to the Poisson equation with Dirichlet boundary conditions given by

$$-\Delta u = m \quad \text{in } \Omega \quad (1.218)$$

$$u = 0 \quad \text{on } \partial\Omega, \quad (1.219)$$

where $\Omega = [0, 1]^2$, $M = L^2(\Omega)$ or $H^1(\Omega)$, $u : \Omega \rightarrow \mathbb{R}$ is the unknown temperature, $m \in M$ is the unknown control function acting as source term ($m(x) > 0$ corresponds to heating and $m(x) < 0$ corresponds to cooling), $d \in L^2(\Omega)$ is the given desired temperature profile and $\alpha \in [0, \infty)$ is a Tikhonov regularisation parameter. For a proof of existence and uniqueness of the solution, we refer to [29, Sect. 1.5]. Using the formulation from 1.4, the objective functional for this problem is

$$J(u, m) := \frac{1}{2} \|u - d\|_{L^2(\Omega)}^2 + \frac{\alpha}{2} \|m\|_M^2. \quad (1.220)$$

An iterative approach towards a solution is achieved using methods introduced previously. In a first step, a solution u_0 to the PDE constraints for a given user-defined initial control m_0 is computed using finite elements. In a second step, the Fréchet derivative of the reduced functional $\tilde{J}$ of J with respect to m_0 is computed using the adjoint approach. Using the derivative of $\tilde{J}$, an optimisation method is then applied in a third step, producing a new, better control m_1 . This procedure is then iterated until a convergence criterion is met.

Based on this approach, a numerical solution of (1.217) satisfying (1.218, 1.219) using linear Lagrange finite elements for both m and u can be computed. Results are shown in Fig. 1.7. The parameters for the problem are

$$\Omega = [0, 1]^2, \quad (1.221)$$

$$d(x_1, x_2) = \frac{1}{2\pi^2} \sin(\pi x_1) \sin(\pi x_2), \quad (1.222)$$

$$\alpha = 10^{-6}, \quad (1.223)$$

$$m_0 = 0. \quad (1.224)$$

The underlying optimisation method is an implementation of the limited memory BFGS algorithm from Sect. 1.5.4 provided by Moola [28]. While many widely used optimisation packages assume that the problem is posed in $\mathbb{R}^n$, this implementation respects the native structure of problems posed in arbitrary Hilbert spaces. In other words, the inner product native to the control space, is applied in the determination of search directions and convergence criteria. Respecting the inner products leads to mesh-independent convergence of the optimisation algorithm, which is explored in more detail in Chap. 2. For solving the finite element problem, the FEniCS frame-

work [23] was used. Similar to Firedrake, FEniCS enables the automated solution of differential equations. For an open access FEniCS tutorial we refer to [22]. Further, we made use of dolfin-adjoint [10] for the computation of adjoints. For the Python code used to produce the numerical solutions in this section, we refer to [33].

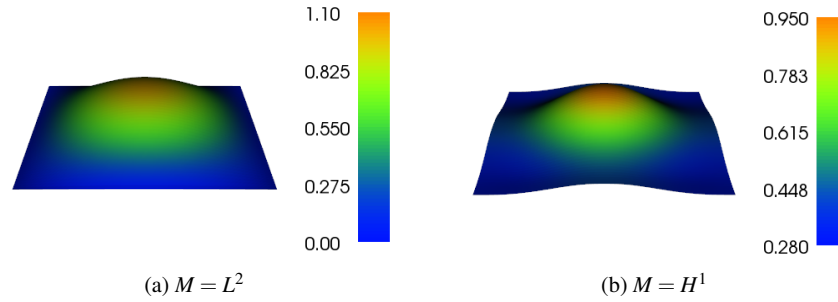


Fig. 1.7: Plots of $\hat{m}$ minimising (1.217) under the PDE-constraints (1.218, 1.219)

1.7 References

- [1] Hans Wilhelm Alt. *Linear functional analysis: an application-oriented introduction*. Springer, 2016.
- [2] S. C. Brenner and L. R. Scott. *The Mathematical Theory of Finite Element Methods*. 3rd Edition, Springer Science+Business Media, LLC, 2008.
- [3] J. Nocedal Byrd R. H. and R. B. Schnabel. “Representations of quasi-Newton Matrices and their Use in Limited Memory Methods”. In: *Mathematical Programming* 63 (1994), pp. 129–156.
- [4] Richard H Byrd et al. “A limited memory algorithm for bound constrained optimization”. In: *SIAM Journal on Scientific Computing* 16.5 (1995), pp. 1190–1208.
- [5] Alexandru Cioaca, Mihai Alexe, and Adrian Sandu. “Second-order adjoints for solving PDE-constrained optimization problems”. In: *Optimization Methods and Software* 27.4-5 (2012), pp. 625–653. DOI: 10.1080/10556788.2011.610455.
- [6] Timothy A Davis. *Direct methods for sparse linear systems*. Vol. 2. Siam, 2006.
- [7] Peter Deuffhard. *Newton methods for nonlinear problems: affine invariance and adaptive algorithms*. Vol. 35. Springer Series in Computational Mathematics. Springer, 2011.
- [8] H. C. Elman, D. J. Silvester, and A. J. Wathen. *Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics*. USA: Oxford University Press, 2005.

- [9] Howard C Elman, David J Silvester, and Andrew J Wathen. *Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics*. Oxford University Press (UK), 2014.
- [10] P. E. Farrell et al. “Automated derivation of the adjoint of high-level transient finite element programs”. In: *SIAM Journal on Scientific Computing* (2012). accepted.
- [11] *Firedrake documentation*. URL: <http://firedrakeproject.org/documentation.html>.
- [12] Fernando Andrés Gallego, John Jairo Quintero, and Juan Carlos Riano. “Convergence of the steepest descent method with line searches and uniformly convex objective in reflexive Banach spaces”. In: *Mathematical Communications* 20.2 (2015), pp. 161–173.
- [13] Michael Ghil and Paola Malanotte-Rizzoli. “Data Assimilation in Meteorology and Oceanography”. In: *Advances in Geophysics*. Ed. by Renata Dmowska and Barry Saltzman. Vol. 33. Elsevier, 1991, pp. 141–266. DOI: 10.1016/S0065-2687(08)60442-2.
- [14] Lawrence M Graves. “Riemann integration and Taylor’s theorem in general analysis”. In: *Transactions of the American Mathematical Society* 29.1 (1927), pp. 163–177.
- [15] A. Griewank and A. Walther. *Evaluating derivatives: Principles and techniques of algorithmic differentiation*. Second. Philadelphia, PA, USA: SIAM, 2008. ISBN: 0898714516.
- [16] P. Guillaume and M. Masmoudi. “Computation of high order derivatives in optimal shape design”. In: *Numerische Mathematik* 67.2 (1994), pp. 231–250.
- [17] M. D. Gunzburger. *Perspectives in Flow Control and Optimization*. Advances in Design and Control. Philadelphia, PA, USA: SIAM, 2003. ISBN: 089871527X.
- [18] M. Hinze and K. Kunisch. “Second Order Methods for Optimal Control of Time-Dependent Fluid Flow”. In: *SIAM Journal on Control and Optimization* 40.3 (2001), pp. 925–946. DOI: 10.1137/S0363012999361810.
- [19] M. Hinze et al. *Optimization with PDE constraints*. Vol. 23. Mathematical Modelling: Theory and Applications. Berlin, Heidelberg, New York: Springer-Verlag, 2009. ISBN: 978-1-4020-8838-4.
- [20] A. Jameson. “Aerodynamic design via control theory”. In: *Journal of Scientific Computing* 3.3 (1988), pp. 233–260.
- [21] Robert C Kirby. “From functional analysis to iterative methods”. In: *SIAM review* 52.2 (2010), pp. 269–293.
- [22] H.P. Langtangen and A. Logg. *Solving PDEs in Python: The FEniCS Tutorial I*. Simula SpringerBriefs on Computing. Springer International Publishing, 2017.
- [23] Anders Logg, Kent-Andre Mardal, and Garth Wells. *Automated solution of differential equations by the finite element method: The FEniCS book*. Vol. 84. Springer Science & Business Media, 2012.

- [24] J. N. Lyness and C. B. Moler. “Numerical differentiation of analytic functions”. In: *SIAM Journal on Numerical Analysis* 4.2 (1967), pp. 202–210.
- [25] D. Menemenlis et al. “ECCO2: High resolution global ocean and sea ice data synthesis”. In: *Mercator Ocean Quarterly Newsletter* 31 (2008), pp. 13–21. DOI: 10.1007/s10236-006-0082-1..
- [26] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Science+Business Media, LLC, 2006.
- [27] J. Nocedal and S. J. Wright. *Numerical optimization*. 2nd ed. Berlin, Heidelberg, New-York: Springer-Verlag, 1999.
- [28] M Nordaas and Simon W. Funke. *The Moola optimisation package*. <https://github.com/funsim/moola>. 2016.
- [29] Rene Pinnau and Michael Ulbrich. *Optimization with PDE constraints*. Vol. 23. Springer Science & Business Media, 2008.
- [30] R. L. Raffard and C. J. Tomlin. “Second order adjoint-based optimization of ordinary and partial differential equations with application to air traffic flow”. In: *Proceedings of the 2005 American Control Conference*. 2005, pp. 798–803. DOI: 10.1109/ACC.2005.1470057.
- [31] Florian Rathgeber et al. “Firedrake: Automating the Finite Element Method by Composing Abstractions”. In: *ACM Trans. Math. Softw.* 43.3 (Dec. 2016), 24:1–24:27. ISSN: 0098-3500. DOI: 10.1145/2998441. URL: <http://doi.acm.org/10.1145/2998441>.
- [32] Yousef Saad. *Iterative methods for sparse linear systems*. Siam, 2003.
- [33] Tobias Schwedes, David Ham, and Simon Wolfgang Funke. *Code for simulations in Chapter 1 of ResearchBrief*. Dec. 2016. DOI: 10.5281/zenodo.223099. URL: <https://doi.org/10.5281/zenodo.223099>.
- [34] Detlef Stammer and Carl Wunsch. *Computational requirements for ECCO in support of CLIVAR and GODAE*. The ECCO Report Series 2. Boston, USA: Scripps Institution of Oceanography, Massachusetts Institute of Technology, 1999.
- [35] D. Stammer et al. “Ocean state estimation and prediction in support of oceanographic research”. In: *Oceanography* 2.13 (2000), pp. 51–56.
- [36] D. Thevenin and G. Janiga. *Optimization and computational fluid dynamics*. Berlin, Heidelberg, New-York: Springer-Verlag, 2008. DOI: 10.1007/978-3-540-72153-6.
- [37] Philip Wolfe. “Convergence Conditions for Ascent Methods”. In: *SIAM Review* 11.2 (1969), pp. 226–235.
- [38] Margaret Wright. “The interior-point revolution in optimization: history, recent developments, and lasting consequences”. In: *Bulletin of the American mathematical society* 42.1 (2005), pp. 39–56.

Chapter 2

Mesh dependence in PDE-constrained optimisation

Computationally solving an optimisation problem on Hilbert spaces requires discretisation and the application of an optimisation method to the resulting finite dimensional problem. In this context, mesh-independent convergence of the optimisation algorithm means that, for a discretisation given on a sufficiently fine mesh, the number of iterations required to solve the optimisation problem to a given tolerance is bounded. Conversely, for a mesh dependent algorithm there exist arbitrarily fine meshes which will require an arbitrarily large number of iterations to achieve convergence.

As we will discuss in Sect. 2.1, employing a suitable inner product in the optimisation algorithm is a necessary condition for achieving mesh-independent convergence. However, many well-established continuous optimisation packages are not designed for function based optimisation and are hard-coded to apply the Euclidean (ℓ^2) inner product (for example the MATLAB Optimization Toolbox [33], TAO [9], SNOPT [16], `scipy.optimize` [10, 24] and IPOPT [25, 42]). One can intuitively comprehend the inaccuracy in this practice for function based optimisation: since the inner product defines angles and distances, drawbacks in the convergence of optimisation methods using an inner product associated with a space different from the control space are to be expected. For example, this error manifests in suboptimal search directions. The optimisation methods introduced in Sect. 1.5 all evaluate the gradient of the objective functional at each iteration to determine the search direction. Such methods are therefore called gradient based. The representation of the derivative associated with the functional of interest is inner product dependent, which is an immediate consequence of the Riesz representation theorem. Applying optimisation packages which hard-code the ℓ^2 inner product to problems posed over other Hilbert spaces therefore results in mesh dependent convergence rates. The emergence of inner product aware optimisation packages is recent [35, 43]. Given the prevalence of ℓ^2 optimisation packages, it is germane to attempt to quantify the extent of mesh dependence in optimisation problems. In other words, to what extent is mesh independence a mathematical nicety, and to what extent an essential component of an acceptable optimisation algorithm?

This chapter analyses mesh-dependent algorithms together with their associated performance loss. Representing the functional derivative with respect to the inner product induced by the continuous control space leads to mesh-independent convergence. This, in the opinion of the authors, motivates the use of optimisation methods allowing for problem-specific inner products.

In Sect. 2.1, we examine the existing work on mesh dependence. In the following Sect. 2.2 an analytic estimate is derived for the iteration count in the case of a simple and generic discretised optimisation problem. The system analysed is the steepest descent method applied to a finite element problem. The estimate is based on Kantorovich's inequality and on an upper bound for the condition number of Galerkin mass matrices. Computational experiments are shown which validate the iteration number estimate. Sect. 2.3 presents similar numerical results for a more complex optimisation problem constrained by a partial differential equation. The experiments performed explore mesh-dependent convergence based on uniform and non-uniform mesh refinement including randomly refined meshes.

2.1 Literature overview

Considerable contributions considering mesh independence properties have been achieved for Newton methods. For many of those algorithms it has been possible to establish so called mesh independence principles. That is, for sufficiently small mesh size and under conditions closely related to assumptions guaranteeing convergence of the discretised algorithm, the method is mesh-independent. The proof of the mesh independence principle in [41] is mainly based on [3], which establishes a mesh independence principle for Newton's method on a large class of operator equations and discretisations. Before [41], this result has only been proved for particular classes of boundary value problems [2, 4], which are then used to design mesh refinement schemes [1, 2, 34]. The result from [3] is extended to nonlinear operator equations and their discretisations for merely Hölder continuous operators in [7]. Another extension of [3] to norm constrained Gauss Newton methods is proved in [19]. In [11], the uniform Lipschitz conditions required in [3] are weakened, and an asymptotic version of the mesh independence principle is established. The result is based on a new Newton-Mysovskii theorem in affine invariant formulation. A mesh independence principle for generalised Newton methods for a class of semi-smooth operator equations and a class of control constrained optimal control problems is introduced in [22]. Further, [5] establishes a mesh independence principle for Newton's method applied to stable and consistent discretisations of generalised operator equations. The results are applied to control constrained optimal control problems for ODEs. Mesh independence for Newton's method applied to discretised variational inequalities or generalised equations is proved in [12] under weaker conditions than in [5]. mesh independence conditions that are easier to check in applications than those in [5] are introduced in a subsequent paper [6]. Based on that mesh independence is shown for a Lagrange Newton method applied

to a nonlinear optimal control problem. A further mesh independence result for a class of unconstrained optimal control problems and the BFGS method in Hilbert spaces is proved in [28]. The same authors present a mesh independence principle for discretisations of optimal control problems and gradient projection method based on a condition on the convergence of gradients is proved in [27]. In [40] the concept of semismoothness is generalised from finite dimensions to nonsmooth superposition operators in Banach spaces. The results hold for the large class of function based nonlinear complementarity problems. Based on the semismoothness results, a Newton-type method applied to nonsmooth infinite-dimensional operator equations is developed, assuring mesh-independent convergence of the solution to the discretised problem.

Results on mesh independence have been further achieved based on well chosen refinement strategies. Under local smoothness conditions, a mesh refinement scheme relating coarse and fine mesh sizes guaranteeing mesh-independent convergence of Newton-type methods is discussed in [8]. In [26] it is studied how the local convergence behaviour of quasi-Newton methods in infinite dimensional formulation is affected by the refinement strategy. A similar result is achieved in [18] for the Gauss-Newton method.

Recent work on mesh-independent algorithms for PDE-constrained problems may be summarised as follows. [21] introduces first and second order optimisation algorithms for PDE-constraint optimisation that are inner product aware. Additionally to iterative solution approaches, a one-shot SQP method is introduced. In contrast with iterative methods, one-shot methods solve the underlying PDE simultaneously with the optimisation problem, thereby avoiding the repeated solution of the forward model. One-shot methods are beyond the scope of this book, but the interested reader is referred to [38, 39]. In [41] an augmented Lagrangian SQP method for unconstrained optimal control problems with possibly nonlinear state equations in Hilbert spaces is proposed, for which a mesh independence principle is proved.

Mesh independence properties which rely on the finite element discretisations of the continuous problem have been derived in the literature. [23] develops a mesh-independent bi-directional evolutionary structural optimisation method for topology optimisation. Based on finite element discretisation, a mesh independence filter using nodal variables is introduced that determines the addition of elements and the removing of structural details below a particular length scale. Further recent work on mesh independence in finite element methods focusses on achieving this characteristic in the solution of the primary finite element problem [17, 29, 32]. This is, in fact, very closely related to the problem considered here: solving the finite element problem amounts to minimising an appropriate norm of the residual, and the task of mapping the residual gradient from the dual space back into the primal function space is similar to employing the correct inner products in calculating updates for the optimisation problem.

[29] derives bounds for the condition number of finite element operator matrices based on the structure that the discretisation inherits from the underlying continuous Hilbert space. The condition number depends on the underlying discretisation due to a non-isometric embedding of the operator into Euclidean space. Spectral

bounds that are independent of the finite element subspace (that is in particular of the mesh) are achieved by applying the Riesz map as preconditioner. This abstract preconditioning principle transfers directly to the problem considered here: mesh-independent convergence requires that the Fréchet derivative of the objective functional, which defines the search direction in the optimisation, is represented with respect to the correct Hilbert space. This is equivalent to employing an ℓ^2 representation and preconditioning with the Riesz map. In the case of the Hilbert space L^2 , the Riesz map of the discretised problem is the inverse of the Galerkin mass matrix.

More generally, [17] shows that the choice of a preconditioner is equivalent to the choice of an inner product on the underlying space. That work investigates the dependence of conjugate gradient and MINRES methods in Hilbert spaces on the underlying scalar product. In finite dimensions, the naturally induced scalar product is related to every other inner product by the application of a positive definite matrix. The Riesz map preconditioner corresponds to an application of that matrix's inverse. For L^2 , this is once again the inverse of the Galerkin mass matrix.

[17, 29, 32] state conditions under which mesh dependence can be avoided, rather than discussing the drawbacks of the opposite. Considering that many well-established optimisation packages assume that the problem is formulated in Euclidean space, in what follows we instead investigate of the magnitude of the resulting performance loss.

2.2 A generic optimisation problem

In this section, we study the finite element discretisation of a generic optimisation problem formulated in a Hilbert function space. If we solve this problem using the steepest descent method with respect to the ℓ^2 inner product, the convergence rate is mesh-dependent. For this situation we analytically derive an iteration number estimate that reveals that the number of iterations is polynomial in the ratio between the largest and smallest directional element sizes of the mesh. Conversely, if the optimisation is conducted in the inner product induced by the Hilbert space, the optimisation converges in exactly one iteration, independently of the computational mesh employed. Numerical experiments confirm the analytical estimate in the special case of the Hilbert space L^2 . The results from this section are based on [36].

2.2.1 Formulation

In order to discuss how the choice of the inner product for the representation of the functional derivative impacts upon the convergence of the optimisation with respect to the underlying discretisation, it is helpful to consider an example as simple as possible while maintaining the essential features of the problem. While such an optimisation problem is an exemplar, it suggests similar dependence on the inner

product and the discretisation for more complex optimisation problems, including those with PDE constraints. Consider the minimisation problem,

$$\min_{u \in H} \alpha \langle u, u \rangle_H + \beta \langle u, v \rangle_H + \gamma, \quad (2.1)$$

where $\alpha > 0$, $\beta, \gamma \in \mathbb{R}$ and $v \in H$. Further, $H \subset \{u : \Omega \rightarrow \mathbb{R}\}$, with bounded domain $\Omega \subset \mathbb{R}^n$ and $n \in \mathbb{N}$, is a Hilbert Sobolev space, i.e. $H = W^{k,2}(\Omega)$ with $k \in \mathbb{N}$. The derivation of the steepest descent iteration count estimate for (2.1) can be reduced to the consideration of the simple optimisation problem

$$\min_{u \in L^2(\Omega)} \left\{ f(u) = \frac{1}{2} \langle 1 - u, 1 - u \rangle_{L^2} \right\}, \quad (2.2)$$

where $\langle \cdot, \cdot \rangle_{L^2}$ denotes the inner product of the space $L^2 := L^2(\Omega)$. This simplification is justified by the common finite element approach and the quadratic structure of both discretisations. Details are found later on. It is easy to see that (2.2) is well posed, with $u \equiv 1$ as the unique solution.

In the subsequent sections we consider the solution of (2.2) using the steepest descent algorithm with exact line search and the finite element method. In doing so, two approaches are compared: In the first the continuous formulation of f in (2.2) is employed in order to compute the gradient used for the optimisation method, and then the finite element discretisation is applied. In the second, the discretisation is performed first, whereupon the gradient is computed with respect to the ℓ^2 inner product of the coefficient vectors. We will see that the formulation used in the second approach is mesh-dependent by a scaling of the local finite element mass matrix. Results of the optimisation procedure using both approaches are compared for non-uniform meshes.

2.2.2 Finite element discretisation

Let us assume that the domain Ω is polygonal and that $\mathcal{T}_h$ is a triangulation of Ω in the sense of 1.1.1.6. Let $\mathcal{P}^m(\mathcal{T}_h) \subset L^2(\Omega)$ denote the associated Lagrangian finite element space of order $m \in \mathbb{N}$. For simplicity, we write $\mathcal{P}^m = \mathcal{P}^m(\mathcal{T}_h)$. Any $u \in \mathcal{P}^m$ can be expressed by a set of basis functions $\{\varphi_1, \dots, \varphi_d\}$ of $\mathcal{P}^m$ and coefficients $u_1, \dots, u_d \in \mathbb{R}$ as

$$u = \sum_{i=1}^d u_i \varphi_i. \quad (2.3)$$

Applying this finite element discretisation to the argument in f from (2.2) leads to

$$f(u) = \int_{\Omega} \frac{1}{2} (1 - u(x))^2 dx \quad (2.4)$$

$$= \frac{1}{2} \int_{\Omega} \left(\sum_{i=1}^d (1 - u) \varphi_i(x) \right)^2 dx \quad (2.5)$$

$$= \frac{1}{2} \sum_{i,j=1}^d (1 - u_i)(1 - u_j) \int_{\Omega} \varphi_i(x) \varphi_j(x) dx, \quad (2.6)$$

where we used the fact that

$$\sum_{i=1}^d \varphi_i = 1. \quad (2.7)$$

Given the mass matrix M with $M_{ij} := \int_{\Omega} \varphi_i \varphi_j dx$, the finite element discretised version of the original optimality problem (2.2) can be expressed as

$$\min_{u \in \mathcal{P}^m} f(u) = \min_{\mathbf{u} \in \mathbb{R}^d} \left\{ f(\mathbf{u}) = \frac{1}{2} (\mathbf{1} - \mathbf{u})^T M (\mathbf{1} - \mathbf{u}) \right\}, \quad (2.8)$$

where $\mathbf{u} = (u_1, \dots, u_d)$, $\mathbf{1} = (1, \dots, 1) \in \mathbb{R}^d$. Here, we identified $f|_{\mathcal{P}^m}$ with the function

$$\mathbf{u} \mapsto \frac{1}{2} (\mathbf{1} - \mathbf{u})^T M (\mathbf{1} - \mathbf{u}). \quad (2.9)$$

This makes sense since for any $u \in \mathcal{P}^m$ there is a unique $\mathbf{u} \in \mathbb{R}^d$ such that (2.3) holds true.

2.2.3 Iteration count using L^2 inner product

In order to solve problem (2.2) in its continuous form, we apply the steepest descent method while representing the gradient according to the Riesz representation theorem with respect to the L^2 inner product. Initialised by $u_0 \in L^2(\Omega)$, the first iterate u_1 is given by $u_1 = u_0 - \alpha \mathcal{R}_{L^2}(\mathrm{d}f(u_0))$, where $\mathcal{R}_{L^2}(\mathrm{d}f(u_0))$ denotes the Riesz representer of the first Fréchet derivative $\mathrm{d}f(u_0)$ in L^2 . We compute $\mathrm{d}f(u) \in \mathcal{L}(L^2, \mathbb{R})$ as

$$\mathrm{d}f(u; v) = -\frac{1}{2} \left(\langle v, 1 - u \rangle_{L^2} + \langle 1 - u, v \rangle_{L^2} \right) \quad (2.10)$$

$$= -\langle 1 - u, v \rangle_{L^2} \quad \forall v \in L^2(\Omega), \quad (2.11)$$

where we used the symmetry of $\langle \cdot, \cdot \rangle_{L^2}$. Hence,

$$\mathcal{R}_{L^2}(\mathrm{d}f(u)) = -(1 - u). \quad (2.12)$$

The second Fréchet derivative $d^{(2)}f(u) \in \mathcal{L}(L^2, \mathcal{L}(L^2, \mathbb{R}))$ is given by

$$d^{(2)}f(u; v, w) = \langle v, w \rangle_{L^2} \quad \forall v, w \in L^2(\Omega). \quad (2.13)$$

For any $n > 2$, we find $d^{(n)}f(u) = 0$. Applying Taylor's theorem for function spaces at $u_1 = u_0 - \alpha \mathcal{R}(df(u_0))$ using (2.12) yields

$$f(u_1) = f(u_0) - \alpha df(u_0; \mathcal{R}_{L^2}(df(u_0))) \quad (2.14)$$

$$+ \frac{1}{2} \alpha^2 d^{(2)}f(u_0; \mathcal{R}_{L^2}(df(u_0)), \mathcal{R}_{L^2}(df(u_0))) \quad (2.15)$$

$$= \left(\frac{1}{2} - \alpha + \frac{1}{2} \alpha^2 \right) \|1 - u_0\|_{L^2}^2. \quad (2.16)$$

One easily computes that the minimum of $f(u_1)$ is found for $\alpha^* = 1$, i.e. α^* is the step size of an exact line search, and $u_1 \equiv 1$. Hence, the solution of the optimality problem (2.2) is found after one steepest descent iteration, independently of the initial $u_0 \in L^2(\Omega)$. This still holds for the finite element discretised version (2.8) of problem (2.2). This can be seen from the fact that for any $u_0 = \sum_{i=1}^d u_{0,i} \phi_i \in \mathcal{P}^m$ it holds

$$\mathcal{R}_{L^2}(df(u_0)) = u_0 - 1 = \sum_{i=1}^d (u_{0,i} - 1) \cdot \phi_i, \quad (2.17)$$

and thus equals the interpolated L^2 represented derivative of f at u_0 . Hence, the first steepest descent iterate for the discretised optimisation problem (2.8) equals the constant function 1, which is the optimum.

It may be noted that the evaluation of the functional derivative with respect to the appropriate inner product generally requires the solution of a linear system. Let the finite element discretisation of some $g : L^2(\Omega) \rightarrow \mathbb{R}$ be given by

$$g(\mathbf{u}) = \frac{1}{2} \mathbf{u}^T \mathbf{A} \mathbf{u}. \quad (2.18)$$

Then, the appropriate derivative is computed as

$$\mathcal{R}_{L^2}(dg(\mathbf{u})) = M^{-1} \mathbf{A} \mathbf{u}, \quad (2.19)$$

where M^{-1} denotes the inverse of the mass matrix. To solve (2.19) one needs to solve the linear system of the form $M \mathbf{v} = \mathbf{A} \mathbf{u}$.

2.2.4 Iteration count estimate using the ℓ^2 inner product

In this section, an iteration count estimate is derived for the steepest descent method using gradients with respect to the ℓ^2 inner product, computed based on the finite element discretised version (2.8) of the optimality problem (2.2). Since M is symmetric, one computes

$$df(\mathbf{u}; \mathbf{v}) = -(\mathbf{1} - \mathbf{u})^T M \mathbf{v} = \langle -M(\mathbf{1} - \mathbf{u}), \mathbf{v} \rangle_{\ell^2} \quad \forall \mathbf{v} \in \mathbb{R}^d, \quad (2.20)$$

where $\langle \cdot, \cdot \rangle_{\ell^2}$ denotes the ℓ^2 inner product, i.e. $\langle \mathbf{u}, \mathbf{v} \rangle_{\ell^2} := \mathbf{u}^T \mathbf{v} = \sum_{i=1}^d u_i v_i$ for any $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$. Thus

$$\mathcal{R}_{\ell^2}(df(\mathbf{u})) = \nabla f(\mathbf{u}) = -M(\mathbf{1} - \mathbf{u}). \quad (2.21)$$

Notice that $\mathcal{R}_{\ell^2}(df)$ differs from $\mathcal{R}_{L^2}(df)$ in (2.12) by the multiplication of the mass matrix M . Since M reflects the structure of the underlying mesh, that is, the spatial distribution of elements and their sizes, it is expected that the convergence of the steepest descent method using $\mathcal{R}_{\ell^2}(df)$ is mesh-dependent.

Using (2.21), the $(k+1)$ th iterate of the steepest descent method is given by $\mathbf{u}_{k+1} = \mathbf{u}_k + \alpha_k M(\mathbf{1} - \mathbf{u}_k)$. In order to minimise

$$f(\mathbf{u}_{k+1}) = \frac{1}{2} \left(\mathbf{1} - (\mathbf{u}_k + \alpha_k M(\mathbf{1} - \mathbf{u}_k)) \right)^T M \left(\mathbf{1} - (\mathbf{u}_k + \alpha_k M(\mathbf{1} - \mathbf{u}_k)) \right), \quad (2.22)$$

with respect to α_k , we set $\frac{\partial f}{\partial \alpha_k}(\mathbf{u}_{k+1}) = 0$ such that

$$\alpha_k = \frac{(\mathbf{1} - \mathbf{u}_k)^T M^2 (\mathbf{1} - \mathbf{u}_k)}{(\mathbf{1} - \mathbf{u}_k)^T M^3 (\mathbf{1} - \mathbf{u}_k)}. \quad (2.23)$$

Note that the Hessian of f equals M . Considering the second derivative of $f(\mathbf{u}_{k+1})$ with respect to α_k and using the fact that mass matrices are positive definite shows that α_k as given in (2.23) is the unique minimiser, i.e. α_k is the step size of an exact line search.

Using steepest descent with exact line search on a strongly convex quadratic function, we may apply Kantorovich's lemma from [31, Sect. 8.6]. The lemma provides an recursive error estimate in f at the k^{th} iterate with respect to the condition number $\kappa(M)$ of M . Since M is normal, its condition number is given by

$$\kappa(M) = \frac{\lambda_{\max}^M}{\lambda_{\min}^M}, \quad (2.24)$$

where $\lambda_{\max}^M$ and $\lambda_{\min}^M$ denote the maximum and minimum eigenvalues of M , respectively. Applying the Lemma of Kantorovich yields

$$f(\mathbf{u}_k) - f(\mathbf{u}^*) \leq \left[\frac{\kappa(M) - 1}{\kappa(M) + 1} \right]^2 \left(f(\mathbf{u}_{k-1}) - f(\mathbf{u}^*) \right) \quad (2.25)$$

$\leq \dots$

$$\leq \left[\frac{\kappa(M) - 1}{\kappa(M) + 1} \right]^{2k} \left(f(\mathbf{u}_0) - f(\mathbf{u}^*) \right) \quad (2.26)$$

$$= \left[\frac{\kappa(M) - 1}{\kappa(M) + 1} \right]^{2k} f(\mathbf{u}_0), \quad (2.27)$$

where $\mathbf{u}^* = (1, \dots, 1) \in \mathbb{R}^d$ is the optimal value such that $f(\mathbf{u}^*) = 0$. It is noted that Kantorovich's estimate is actually valid for the steepest descent solution to (2.1) for any Hilbert space H and its subspaces. Thus, in particular not only for finite element discretisations.

An obvious question is how to determine $\kappa(M)$ from the underlying discretisation of the domain. Corollary 1 of [14] gives an upper bound for the condition number of mass matrices depending on the minimum and maximum eigenvalues of their local mass matrices. Let M_K be the local mass matrix associated with element K , and let $\lambda_{\max}^{M_K}$ and $\lambda_{\min}^{M_K}$ denote the maximum and minimum eigenvalues of M_K , respectively. Applying the estimate to M , we obtain

$$\kappa(M) \leq p_{\max} \frac{\max_K \lambda_{\max}^{M_K}}{\min_K \lambda_{\min}^{M_K}}, \quad (2.28)$$

where $p_{\max}$ is the maximum number of elements around any nodal point. In order to determine $\lambda_{\max}^{M_K}$ and $\lambda_{\min}^{M_K}$, respectively, we consider the bijective transformation T_K from a reference element $(\hat{K}, \hat{\mathcal{P}} = \{\psi_1, \dots, \psi_p\}, \hat{\mathcal{N}} = \{\hat{N}_1, \dots, \hat{N}_p\})$ to the local element $(K, \mathcal{P}^K = \{\phi_1^K, \dots, \phi_p^K\}, \mathcal{N}^K = \{N_1^K, \dots, N_p^K\})$, where $\phi_l^K \circ T_K = \psi_l$. Let us assume affine transformations, i.e.

$$\begin{aligned} T_K : \hat{K} &\rightarrow K \\ x &\mapsto J_K x + y_K, \end{aligned} \quad (2.29)$$

for $J_K \in \mathbb{R}^{n \times n}$ invertible and $y_K \in \mathbb{R}^n$. Applying singular value decomposition to the Jacobian matrix J_K of the transformation T_K , there are unitary $U, V \in \mathbb{R}^{n \times n}$ such that

$$J_K = U H_K V, \quad (2.30)$$

where $H_K \in \mathbb{R}^{n \times n}$ is a diagonal matrix with positive real numbers $h_1^K, \dots, h_n^K$ on the diagonal. Thus,

$$|\det(J_K)| = |\det(H_K)| = h_1^K \cdot \dots \cdot h_n^K. \quad (2.31)$$

The factors $h_1^K, \dots, h_n^K$ describe the scaling of lengths in orthogonal directions between reference element $\hat{K}$ and local element K , respectively. Using the transforma-

tion theorem, an entry μ_{lm}^K with $l, m = 1, \dots, p$ of the local mass matrix M_K associated with element K can therefore be written as

$$\mu_{lm}^K = \int_K \phi_l^K(x) \phi_m^K(x) dx \quad (2.32)$$

$$= \int_{\tilde{K}} \phi_l^K(T_K(\zeta)) \phi_m^K(T_K(\zeta)) |\det(J_K)| d\zeta \quad (2.33)$$

$$= h_1^K \cdots h_n^K \int_{\tilde{K}} \psi_l(\zeta) \psi_m(\zeta) d\zeta \quad (2.34)$$

$$= h_1^K \cdots h_n^K \cdot \hat{\mu}_{lm}. \quad (2.35)$$

Consequently, the local mass matrix M_K has the form

$$M_K = h_1^K \cdots h_n^K \hat{M}, \quad (2.36)$$

where $\hat{M} = (\hat{\mu}_{lm})_{l,m=1,\dots,p}$ is the mass matrix associated with the reference element $\tilde{K}$. Hence, the eigenvalues of M_K are proportional to the eigenvalues of $\hat{M}$, with proportionality factor equal to $h_1 \cdots h_n$. Note that the matrix $\hat{M}$ is independent of the underlying element K .

Let $\lambda_{\max}^{\hat{M}}$ and $\lambda_{\min}^{\hat{M}}$ denote the largest and smallest eigenvalues of $\hat{M}$, respectively. The estimate (2.28) can now be written as

$$\kappa(M) \leq p_{\max} \frac{\lambda_{\max}^{\hat{M}}}{\lambda_{\min}^{\hat{M}}} \frac{\max \left\{ \prod_{i=1}^n h_i^K : K \right\}}{\min \left\{ \prod_{i=1}^n h_i^K : K \right\}} \quad (2.37)$$

$$\leq p_{\max} \frac{\lambda_{\max}^{\hat{M}}}{\lambda_{\min}^{\hat{M}}} \prod_{i=1}^n \frac{\max_K h_i^K}{\min_K h_i^K}. \quad (2.38)$$

Due to the role of $h_1^K, \dots, h_p^K$ as scaling factors of the element K relative to the reference element $\tilde{K}$ in orthogonal directions, the expression

$$\max \left\{ \prod_{i=1}^n h_i^K : K \right\} / \min \left\{ \prod_{i=1}^n h_i^K : K \right\} \quad (2.39)$$

may be considered as a measure of non-uniformity in the mesh.

In the particular case where the maximum and minimum of the scaling factors h_i^K in orthogonal directions coincide for $i = 1, \dots, n$, we may set $h_{\max} := \max_K h_i^K$ and $h_{\min} := \min_K h_i^K$ for any i , respectively, such that (2.38) simplifies to

$$\kappa(M) \leq p_{\max} \frac{\lambda_{\max}^{\hat{M}}}{\lambda_{\min}^{\hat{M}}} \left(\frac{h_{\max}}{h_{\min}} \right)^n. \quad (2.40)$$

Assuming that the underlying mesh contains significant non-uniformity, i.e. $h_{\max}/h_{\min} \gg 1$, the following approximation holds true:

$$\log \left(\frac{p_{\max} \frac{\lambda_{\max}^{\hat{M}}}{\lambda_{\min}^{\hat{M}}} \left(\frac{h_{\max}}{h_{\min}} \right)^n - 1}{p_{\max} \frac{\lambda_{\max}^{\hat{M}}}{\lambda_{\min}^{\hat{M}}} \left(\frac{h_{\max}}{h_{\min}} \right)^n + 1} \right) = \sum_{v=1}^{\infty} \frac{(-1)^v}{v} \left(\frac{p_{\max} \frac{\lambda_{\max}^{\hat{M}}}{\lambda_{\min}^{\hat{M}}} \left(\frac{h_{\max}}{h_{\min}} \right)^n - 1}{p_{\max} \frac{\lambda_{\max}^{\hat{M}}}{\lambda_{\min}^{\hat{M}}} \left(\frac{h_{\max}}{h_{\min}} \right)^n + 1} \right)^v \quad (2.41)$$

$$= \sum_{v=1}^{\infty} -\frac{2^v}{v} \cdot \left(p_{\max} \frac{\lambda_{\max}^{\hat{M}}}{\lambda_{\min}^{\hat{M}}} \left(\frac{h_{\max}}{h_{\min}} \right)^n + 1 \right)^{-v} \quad (2.42)$$

$$\approx -2 \cdot \left(p_{\max} \frac{\lambda_{\max}^{\hat{M}}}{\lambda_{\min}^{\hat{M}}} \left(\frac{h_{\max}}{h_{\min}} \right)^n + 1 \right)^{-1}. \quad (2.43)$$

Setting (2.27) smaller or equal to ε , one then obtains

$$k \geq -\frac{1}{4} \log(\varepsilon/f(\mathbf{u}_0)) \cdot \left(p_{\max} \frac{\lambda_{\max}^{\hat{M}}}{\lambda_{\min}^{\hat{M}}} \left(\frac{h_{\max}}{h_{\min}} \right)^n + 1 \right) =: \hat{k}. \quad (2.44)$$

Thus, after $\hat{k}$ iterations the error of the steepest descent method is smaller or equal to ε . Careful interpretation of this result yields that an iteration number satisfying that the associated error is smaller or equal to ε is at most polynomial in the ratio between largest and smallest directional element size of order equal to the domain dimension n . The subsequent numerical simulations show that this dependency is also observed numerically.

2.2.4.1 Remark

The simplification of considering (2.2) instead of (2.1) in the context of deriving (2.44) demands some justification. With the notation used above, discretising

$$\alpha \langle u, u \rangle_H + \beta \langle u, v \rangle_H + \gamma, \quad (2.45)$$

using finite elements leads to

$$\alpha \sum_{i,j=1}^d u_i u_j \langle \varphi_i, \varphi_j \rangle_H + \beta \sum_{i,j=1}^d u_i v_j \langle \varphi_i, \varphi_j \rangle_H + \gamma = \alpha \mathbf{u}^T M_H \mathbf{u} + \beta \mathbf{u}^T M_H \mathbf{v} + \gamma, \quad (2.46)$$

where $v_1, \dots, v_n$ are the coefficients of the interpolant $\mathcal{I}_{\mathcal{T}}(v) = \sum_{i=1}^n v_i \varphi_i$ of $v \in H$, and $M_H = (\langle \varphi_i, \varphi_j \rangle_H)_{i,j}$. In the special case of $H = L^2([0, 1])$, M_H coincides with the Galerkin mass matrix M . Since adding constants does not change the solution of

an optimisation problem, minimising the right hand side of (2.46) is equivalent to minimising

$$\frac{1}{2} \mathbf{u}^T \tilde{M}_H \mathbf{u} + \mathbf{u}^T \tilde{\mathbf{v}}, \quad (2.47)$$

where $\tilde{M}_H = 2\alpha M_H$ and $\tilde{\mathbf{v}} = \beta M_H \mathbf{v}$. Analogously, minimising (2.47) is equivalent to minimising

$$\frac{1}{2} \mathbf{u}^T \tilde{M}_H \mathbf{u} + \mathbf{u}^T \tilde{\mathbf{v}} + \frac{1}{2} \mathbf{u}^{*T} \tilde{M}_H \mathbf{u}^* = \frac{1}{2} (\mathbf{u} - \mathbf{u}^*)^T \tilde{M}_H (\mathbf{u} - \mathbf{u}^*), \quad (2.48)$$

where $\mathbf{u}^*$ is the optimal solution, for which obviously $\tilde{M}_H \mathbf{u}^* = -\tilde{\mathbf{v}}$. Since Kantorovich's inequality is valid for any $\mathbf{u}^*$ and the estimate in (2.25) depends only on the condition number of the Hessian matrix of the quadratic problem, one may conclude that the iteration count estimate is valid for minimising (2.48) in the same way as for the minimisation of

$$\frac{1}{2} (\mathbf{u} - \mathbf{1})^T \tilde{M}_H (\mathbf{u} - \mathbf{1}). \quad (2.49)$$

We need to show that (2.44) applies for $\tilde{M}_H$ just as it does for M . For brevity we write substitute $\tilde{M}_H$ for M_H . Let $\langle \cdot, \cdot \rangle_{H(K)}$ and $\langle \cdot, \cdot \rangle_{H(\hat{K})}$ be defined as the inner products of the same integral type as H , with integration domains equal to K and $\hat{K}$, respectively. The local finite element matrix $M_{H,K}$ corresponding to M_H is then given by the entries

$$\mu_{lm}^{H,K} = 2\alpha \cdot \langle \phi_l^K, \phi_m^K \rangle_{H(K)}, \quad (2.50)$$

where ϕ_l^K with $l = 1, \dots, p$ denote the local basis functions associated with element K . In a manner analogous to the proof of [14, Theorem 1], one can show that the condition number estimate (2.28) holds true for M_H , i.e.

$$\kappa(M_H) \leq p_{\max} \frac{\max_K \lambda_{\max}^{M_{H,K}}}{\min_K \lambda_{\min}^{M_{H,K}}}. \quad (2.51)$$

A similar upper bound can be obtained for the Hilbert space $H = H_0^1(\Omega)$ (which we excluded in our initial consideration) with $\langle u, v \rangle_H := \int_{\Omega} \nabla u \nabla v dx$. In that case, $M_H = S$ is the stiffness matrix, i.e.

$$S_{ij} := \int_{\Omega} \nabla \phi_i \nabla \phi_j dx. \quad (2.52)$$

Thus, it might be that $\min_K \lambda_{\min}^{M_{H,K}} = 0$ such that (2.51) becomes trivial. Let for instance $H = H_0^1([0, 1])$, $\langle u, v \rangle_H := \int_0^1 u'(x) v'(x) dx$ and linear continuous Lagrange finite elements. The reference basis functions are given by $\phi_1(x) = x$ and $\phi_2(x) = 1 - x$. Thus,

$$S = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}, \quad (2.53)$$

which has eigenvalues $\{0, 2\}$. In such situations, [14, Theorem 3] is very useful since it provides a non-trivial upper bound as

$$\kappa(S) \leq p_{\max} \frac{\max_K \lambda_{\max}^{S_K}}{\lambda_1 \min_K \lambda_{\min}^{M_K}}, \quad (2.54)$$

where λ_1 is the smallest eigenvalue of the negative Laplacian from the boundary value problem

$$\begin{aligned} -\Delta u &= \lambda u & \text{on } \Omega \\ u &= 0 & \text{on } \partial\Omega. \end{aligned} \quad (2.55)$$

Following [20], λ_1 can be expressed as

$$\lambda_1 = \inf_{\substack{u \in H_0^1(\Omega), \\ u \neq 0}} \frac{\int_{\Omega} |\nabla u(x)|^2 dx}{\int_{\Omega} u^2(x) dx} > 0. \quad (2.56)$$

For the simple case of $\Omega = [0, 1]$, the general solution of (2.55) is easily derived as $u(x) = \sin(\sqrt{\lambda}x)$. Hence, $\lambda_1 = \pi^2$.

The reference finite element matrix $\hat{M}_H$ corresponding to M_H is given by the entries

$$\hat{\mu}_{lm}^H = 2\alpha \cdot \langle \psi_l, \psi_m \rangle_{H(\hat{K})}, \quad (2.57)$$

where ψ_l with $l = 1, \dots, p$ denote the basis functions associated with the reference element $\hat{K}$. Since the inner products are of integral type, the transformation theorem applies, analogously to (2.32)-(2.35), for the mapping from the reference element to the local element, i.e.

$$\mu_{lm}^{H,K} = 2\alpha \cdot \langle \phi_l, \phi_m \rangle_{H(K)} \quad (2.58)$$

$$= 2\alpha \cdot h_1^K \cdots h_n^K \langle \psi_l, \psi_m \rangle_{H(\hat{K})} \quad (2.59)$$

$$= h_1^K \cdots h_n^K \hat{\mu}_{lm}^H. \quad (2.60)$$

Since the remaining derivation does not change, it is easy to see that formula (2.44) is valid for the finite element discretised version of (2.1), when $f(\mathbf{u}_0)$ is replaced by $f(\mathbf{u}_0) - f(\mathbf{u}^*)$, and $\lambda_{\max}^{\hat{M}}, \lambda_{\min}^{\hat{M}}$ are replaced by $\lambda_{\max}^{\hat{M}_H}, \lambda_{\min}^{\hat{M}_H}$, respectively.

2.2.5 Numerical experiments

This section verifies the iteration count estimates by solving the optimisation problem (2.2) numerically. The following simulations were performed using the FEniCS automated finite element system [30] and the optimisation framework dolfin-adjoint [13, 15].

The first experiment investigates the number of optimisation iterations required to solve (2.2) under *non-uniform mesh refinement*. The meshes are constructed iteratively: the $(r + 1)$ -times refined mesh is derived from the r -times refined mesh by doubling the mesh resolution at the boundary of the r -times refined mesh. That way, $h_{\max}/h_{\min}$ doubles after every refinement. Figure (2.1) shows the evolution of iteration numbers for the ℓ^2 gradient representation with respect to $h_{\max}/h_{\min}$ using linear Lagrange elements in one, two and three dimensions, and their corresponding analytical iteration count estimates. All optimisations started at an initial control value of $u = 0$. The results confirm the polynomial dependency between iteration number and $h_{\max}/h_{\min}$ in the estimate (2.44). Analogous dependencies were found for other and higher order finite elements implemented, emphasising that the iteration number evolution indeed exhibits the polynomial relationship given in (2.44), independently of the finite element choice.

The second experiment considers the iteration numbers under *uniform refinement*, that is the mesh is refined by an equal factor throughout the domain such that the ratio $h_{\max}/h_{\min}$ remains unaltered. In this case, we do not expect the iteration number to increase according to (2.44). This result is confirmed in table 2.1, which even shows a slight decrease in iteration numbers for increasing refinement levels.

Note that the iteration numbers from Fig. (2.1) and table 2.1 compare to an iteration number of one when the functional derivative of f is represented with respect to L^2 corresponding to the control variable space, independent of the mesh or the dimension of the underlying domain.

Table 2.1: Iterations of the steepest descent method with exact line search and ℓ^2 represented gradients for Lagrange finite elements of different order (CG1-5) on uniformly refined meshes in 1D depending on the refinement level (number of elements). The error threshold is given by $\varepsilon = 10^{-6}$ in the functional value

<i>Refinement</i>	<i>CG1</i>	<i>CG2</i>	<i>CG3</i>	<i>CG4</i>	<i>CG5</i>
1 (96)	13	20	19	37	43
2 (192)	12	20	16	36	41
3 (384)	11	19	15	35	40
4 (768)	10	19	13	35	38
5 (1536)	8	19	13	35	37

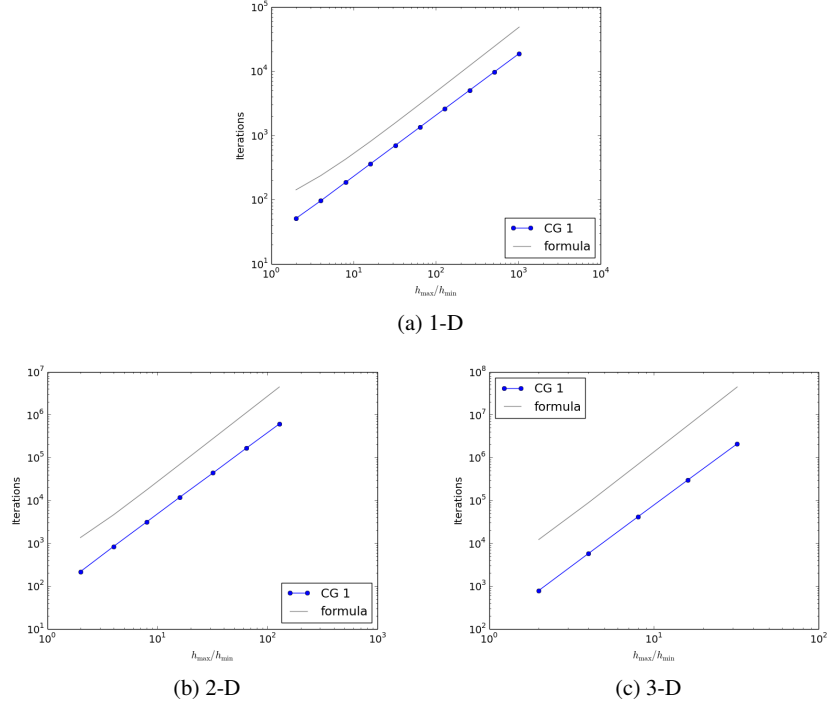


Fig. 2.1: Iterations of the steepest descent method with exact line search and ℓ^2 represented gradients, using linear Lagrange finite elements on non-uniform refined meshes with respect to $h_{\max}/h_{\min}$ on a log-log scale, and with error threshold $\varepsilon = 10^{-15}$ in the functional value

2.3 Mesh dependence in the optimal control of the Poisson equation

The iteration count estimate from 2.2 is derived for a particularly simple case in the expectation that the impact of the use of non-native incorrect inner product on more complex problems will be at least as severe. A common and more realistic case for function based optimisation to occur is PDE-constraint optimisation. To exemplify this, we consider the optimal control problem constrained by the Poisson equation introduced in Sect. 1.6. As in before, the PDE-constraints are eliminated in every optimisation iteration by solving (1.218) and (1.219) using linear Lagrange finite elements and an adjoint solve producing the Fréchet derivative of the reduced functional. The results will show that the iteration count is mesh-dependent when the ℓ^2 inner product is employed in computing this derivative, while mesh-independent convergence is achieved by employing the L^2 inner product. Convergence criteria

for all algorithms in this section are posed on the gradient, and are identical, i.e. are independent of the optimisation method or the inner product employed. We do that by computing the functional derivative with respect to the appropriate inner product outside the ℓ^2 employing algorithms. For the Python code used to produce the numerical results from this section, we refer to [37].

2.3.1 Experiments on deterministically refined meshes

In this section, mesh dependence is analysed numerically for a hierarchy of meshes that arise by the refinement scheme stated in 2.2.5. We solve this problem with four optimisation software packages: 1) the limited BFGS method [10] as implemented in Scipy [24]; 2) the Limited Memory, Variable Metric (LMVM) method from the TAO library [9] which uses a BFGS approximation with a Moré-Thuente line search; and 3) the interior point line search filter method described in [42] and implemented in IPOPT [25] which uses the BFGS method to approximate the Hessian of the Lagrangian and not of the functional itself; 4) the BFGS implementation from Moola [35]. The first three optimisation software packages assume an Euclidean space while the Moola allows user-specific inner products and Riesz maps¹

Table (2.2) displays iteration counts for the different algorithm implementations solving (1.217) subject to (1.218) and (1.219), discretised by linear Lagrange finite elements, for $H = L^2$ and $H = H^1$ with $\Omega = [0, 1]^2$, respectively. In both cases, the inner product insensitive methods exhibit mesh dependence. This can be seen from the increase in iteration numbers for increasing ratio between largest and smallest element size $h_{\max}/h_{\min}$, reflecting the non-uniformity in the discretisation. Iteration counts increase roughly by factors between 4-25 when comparing the least and most non-uniform meshes. Relating this to large scale optimisation problems, it is obvious that the ability of efficiently dealing with non-uniformity in the discretisation can very well decide upon the computational feasibility of solving the problem. The optimisation method that respects the inner product of the control space is approximately mesh-independent.

Table 2.3 illustrates the limitations of the analysis made in 2.2: non-uniform mesh refinement is sufficient to induce mesh dependence in Euclidean algorithms, but it is not necessary. Mesh dependence is also observed for uniform mesh refinement in the Euclidean algorithms, however for $H = L^2$ the dependence is less significant, far less in the case of BFGS. This highlights the point that mesh dependence is not limited to the case of non-uniformity in the mesh, but can arise whenever the inner product employed is not the one implied by the problem setting.

¹ Very recent TAO versions have support for user-specific inner products.

Table 2.2: Iterations of different common optimisation method implementations with Python interfaces, using different inner products and non-uniform mesh refinement. The error threshold is given by $\varepsilon = 10^{-7}$ in the gradient norm $\|\mathcal{R}_H(J')\|_H$. History length for Hessian approximation equal to 10, except for LMVM in TAO, where it is equal to 5

inner product	Implementation	$h_{\max}/h_{\min}$					
		4	8	16	32	64	128
ℓ^2	Scipy (BFGS)	27	47	75	97	133	189
ℓ^2	TAO (LMVM)	42	60	77	111	131	155
ℓ^2	IPOPT (Interior Point)	28	38	65	88	117	139
L^2	Moola (BFGS)	22	20	22	23	23	27

(a) $H = L^2(\Omega)$

inner product	Implementation	$h_{\max}/h_{\min}$					
		4	8	16	32	64	128
ℓ^2	Scipy (BFGS)	25	53	118	222	244	594
ℓ^2	TAO (LMVM)	38	38	110	144	141	384
ℓ^2	IPOPT (Interior Point)	31	38	85	127	182	443
H^1	Moola (BFGS)	4	4	4	4	4	4

(b) $H = H^1(\Omega)$

Table 2.3: Iterations of different common optimisation method implementations with Python interfaces, using different inner products and non-uniform mesh refinement. The error threshold is given by $\varepsilon = 10^{-7}$ in the gradient norm $\|\mathcal{R}_H(J')\|_H$. History length for Hessian approximation equal to 10, except for LMVM in TAO, where it is equal to 5

inner product	Implementation	refinement level					
		0	1	2	3	4	5
ℓ^2	Scipy (BFGS)	63	85	80	74	63	61
ℓ^2	TAO (LMVM)	48	78	100	99	113	106
ℓ^2	IPOPT (Interior Point)	41	62	73	55	77	117
L^2	Moola (BFGS)	24	23	25	23	25	23

(a) $H = L^2(\Omega)$

inner product	Implementation	refinement level					
		0	1	2	3	4	5
ℓ^2	Scipy (BFGS)	11	18	29	34	91	153
ℓ^2	TAO (LMVM)	11	21	25	49	111	230
ℓ^2	IPOPT (Interior Point)	11	18	22	34	90	256
H^1	Moola (BFGS)	4	4	4	4	4	4

(b) $H = H^1(\Omega)$

2.3.2 Experiments on randomly refined meshes

To test mesh dependence properties on more complex meshes that might be exhibited in practical applications, e.g. mesh refinement, experiments on randomly refined meshes are performed. In the following, numerical experiments on such meshes are performed using two optimisation algorithms that were introduced in 1.5. The outcomes are compared to experiments on uniformly refined meshes.

2.3.2.1 Newton-CG

In this section, we analyse mesh-dependent convergence of the Newton-CG algorithm presented in Sect. 1.5 on randomly refined meshes. For the gradient of the objective both the ℓ^2 and the L^2 representation are considered and the performance of the resulting optimisation procedure is compared. Experiments are likewise performed on uniformly refined meshes to provide a reference. Discontinuous Lagrange finite elements are applied for the discretisation of the controls.

The following two series of meshes are compared: For the first series, which state the randomly refined meshes, the $(i + 1)$ th mesh is given by the randomly refined i th mesh according to the following refinement schema: Given an initial mesh, each element of the domain is labelled for refinement randomly with a probability equal to 0.35. The refined mesh is created by refining the labelled elements of the initial mesh and adjacent elements if necessary in order to prevent hanging nodes in the mesh. The resulting meshes for $\Omega = [0, 1]^2$ are displayed in Figs. 2.2. For the second series of meshes, the mesh is uniformly refined. This means every element is divided into four smaller elements of the same size.

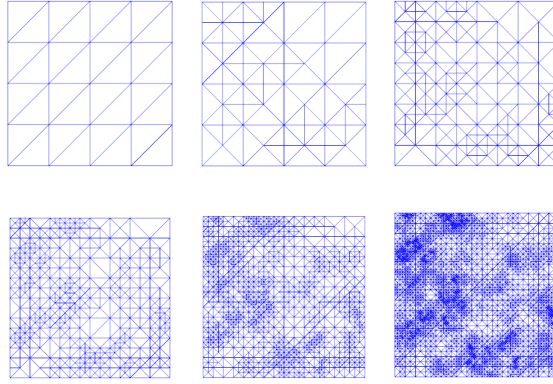


Fig. 2.2: Successively, randomly refined meshes used for the simulations with results given in Fig. 2.3

Plots of the numerical solution of (1.217) subject to (1.218) and (1.219) for $H = L^2$ on the corresponding randomly refined meshes are displayed in Fig. (2.3). Since there are no significant differences in the solution produced by using the ℓ^2 and the L^2 gradient representation for the initial mesh and the first five successively refined meshes, only the solution produced by using the L^2 representation is plotted. Since the solution quality is essentially identical, it is meaningful to make direct performance comparisons between the methods. Performance results corresponding to two different mesh scenarios described above are illustrated in tables 2.4 and 2.4.

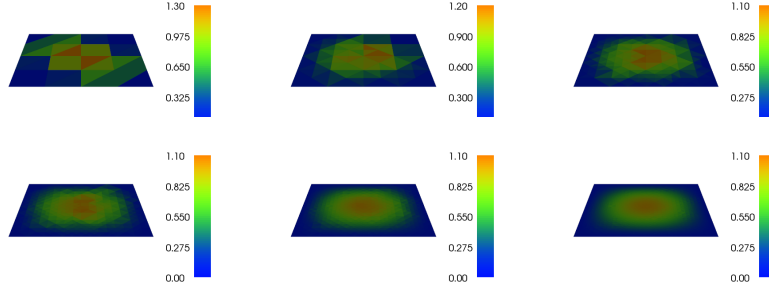


Fig. 2.3: Numerical solution of (1.217) using the L^2 representation of the functional derivative for successively randomly refined meshes, corresponding to Fig. 2.2. Similar outcomes were obtained using the ℓ^2 representation

From table 2.4 the following conclusions can be made: in a manner consistent with the observations for the steepest descent method in Sect. 2.2, the number of iterations remains the same for increasingly non-uniformly refined meshes in the case where the gradient used for the optimisation is represented with respect to the L^2 inner product. On the other hand, the iterations increase when refining the initial mesh non-uniformly if an ℓ^2 representation of the gradient is used. Indeed, from the third iteration upwards the iterations roughly double after every refinement. In addition to the increase of iterations using an ℓ^2 inner product representation instead of L^2 , the average run time per iteration is roughly twice as large from the first refinement level on. One possible explanation for this is that the computation of the ℓ^2 representation of the gradient requires the evaluation of a product with the Galerkin mass matrix, which is more costly when the mass matrix is non-uniform along the diagonal. In Sect. 2.2, it was pointed out that the mass matrix represents the finite element discretisation of the domain in the sense that entries correspond to a multiple of the mesh size. That means the more non-uniform the mesh is, the more non-uniform is the corresponding mass matrix. This explanation seems to be confirmed by comparing the results for the ℓ^2 gradient representation in table 2.4 to the ones in table 2.5: In the case of successively uniformly refined meshes, the run times only increase roughly like the ones where the L^2 representation is used.

Table 2.4: Iterations and run times of the Newton-CG method non-uniformly refined meshes depending on the refinement level. Refinement level 0 corresponds to a uniform mesh. Convergence criteria given by a relative step size of 10^{-3} and gradient L^2 norm of 10^{-7}

Refinement Level	Iterations		Run Time (s)	
	ℓ^2	L^2	ℓ^2	L^2
0 (32)	3	3	4.46	4.52
1 (86)	8	3	39.90	7.32
2 (251)	11	3	58.01	8.60
3 (757)	14	3	80.86	9.42
4 (2368)	33	3	238.50	11.91
5 (7518)	61	3	767.22	18.93
6 (23927)	-	3	-	43.21

Table 2.5: Iterations and run times of the Newton-CG method for uniformly refined meshes depending on the refinement level. Refinement level 0 corresponds to a uniform mesh. The convergence criteria given by a relative step size of 10^{-3} and gradient L^2 norm of 10^{-7}

Refinement Level	Iterations		Run Time (s)	
	ℓ^2	L^2	ℓ^2	L^2
0 (32)	3	3	4.72	4.71
1 (128)	3	3	6.36	6.74
2 (512)	3	3	7.79	8.18
3 (2048)	3	3	9.74	10.37
4 (8192)	-	3	-	19.19

From table 2.4, one can further see that the optimisation method fails on the mesh in the sixth refinement level (and all subsequent levels, not displayed) using an ℓ^2 inner product representation. In this case failing means that in the line search algorithm used in Newton-CG no appropriate step size is found, i.e. the proposed step sizes do not satisfy the curvature conditions given by the Wolfe equations and are therefore refused. In particular, the step sizes do not satisfy a sufficient decrease towards the search direction which, as stated in the Newton-CG algorithm, itself is a function of the gradient. Since the gradient in the ℓ^2 representation is scaled by the Galerkin mass matrix, it becomes closer to zero as the mesh gets finer. For a sufficiently refined mesh, one may therefore expect every proposed step size to be refused and the algorithm to fail. In particular, this effect is coupled to the mesh size and not the non-uniformity of the mesh. As a matter of fact, table 2.5 shows that the algorithm using the ℓ^2 representation fails likewise for uniformly refined meshes from the fourth refinement level (and all subsequent levels, not displayed).

In Fig. 2.4, steepest descent steps of the reduced functional $\tilde{J}(m) = J(u, m)$ with respect to step sizes ε are compared using the two different functional derivative representations. Due to the scaling of functional derivative by the Galerkin mass matrix, one can see that the ℓ^2 -represented gradient is small compared to the L^2 -represented gradient, which is reflected by the different sizes of the steepest descent steps. Furthermore, Fig. 2.5 illustrates that the scaling by the Galerkin mass matrix in the ℓ^2 representation not only leads to generally smaller gradients compared to the L^2 representation, but also that the ℓ^2 represented gradient has a different geometrical structure. This generally leads to different search directions in the optimisation method. Having sufficient time and resources, the method using the ℓ^2 gradient representation should still converge to the unique minimum since $\|\cdot\| = 0$ for the gradient is a valid convergence criteria in every norm. Indeed, this observation is

made in all numerical results from this chapter. However, when considering optimality problems that are not well-posed, including the turbine density optimisation problem (3.1), it is a priori not clear that by using the ℓ^2 representation the correct solution of the optimisation problem is found. Indeed, it will be shown in the following chapter that numerical solutions of this far more complex PDE-constrained optimisation problem derived by an optimisation method using the ℓ^2 representation may compute non-optimal and in parts highly degenerate solutions.

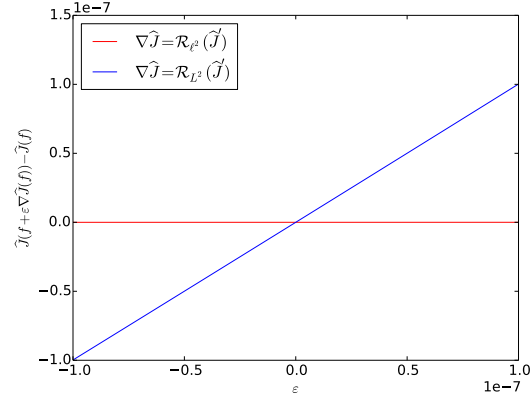


Fig. 2.4: Steepest descent steps of the reduced functional $\tilde{J}$ with respect to ℓ^2 and L^2 Riesz gradient representations (normalised) under small perturbations of the control function f on the sixth randomly refined mesh (refinement level at which the line search fails due to refused step size)

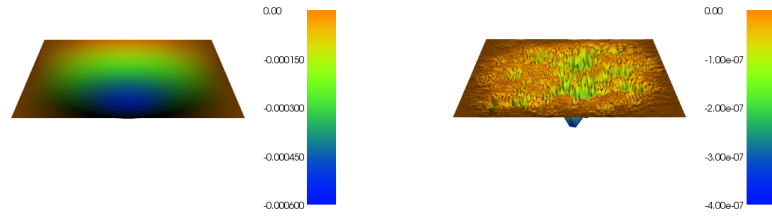


Fig. 2.5: Plots of the L^2 (left) and ℓ^2 (right) representations of the derivative of the reduced functional on the sixth randomly refined mesh (refinement level at which the line search fails due to refused step size)

2.3.2.2 BFGS

In this section, numerical results are discussed for the optimality problem (1.217) subject to (1.218) and (1.219) using the limited memory BFGS algorithm presented in Sect. 1.5.4 for randomly refined meshes. The implementation of the BFGS method is `scipy.optimize` [24], which is a limited memory BFGS method that also allows for bound constraints [10]. The functional derivative of the reduced functional $\tilde{J}$ from (1.220) is represented with respect to the ℓ^2 inner product, which is hard-coded within the implementation of the algorithm. Again, we use discontinuous Lagrange finite elements to discretise the controls.

The results for iterations and run times displayed in table 2.6 using the ℓ^2 gradient representation in the BFGS method confirm mesh-dependent behaviour for non-uniformly refined meshes, similar to the Newton-CG algorithm considered in Sect. 2.3.2.1. Although the BFGS method is for uniform meshes and slightly non-uniform meshes is still faster than the Newton-CG method that uses the correct L^2 inner product representation, the BFGS algorithm is outperformed by the Newton-CG algorithm using the L^2 gradient representation for meshes that are more than once randomly refined. This is interesting since the implementation of the BFGS used here is generally considered to be a faster algorithm than any implementation of the Newton-CG algorithm. The results in table 2.6 show that this is actually only true for sufficiently uniform meshes given that other algorithms have the choice of user-defined inner product spaces to their disposal.

Furthermore, the results from table 2.7 indicate mesh dependence also in the case of uniformly refined meshes for the lower refinement levels. Therefore, we can conclude that mesh-dependent behaviour is not exclusively coupled to the non-uniformity of the mesh, but may be caused by other aspects of the choice of non-native inner products. However, the increase in the iteration number slows down considerably with the increase of the refinement.

Table 2.6: Iterations and run times of the BFGS method on non-uniformly refined meshes depending on the refinement level, corresponding to ℓ^2 case in table 2.4

Ref. Level	Iterations	Run Time (s)
0 (32)	28	1.84
1 (86)	95	5.79
2 (251)	143	9.03
3 (757)	188	13.82
4 (2368)	204	19.00
5 (7518)	262	40.41
6 (23927)	295	128.14

Table 2.7: Iterations and run times of the BFGS method on non-uniformly refined meshes depending on the refinement level, corresponding to ℓ^2 case in table 2.5

Ref. Level	Iterations	Run Time (s)
0 (32)	28	1.84
1 (128)	49	2.98
2 (512)	82	6.01
3 (2048)	64	6.24
4 (8192)	70	11.72

2.4 Final remark

The previous chapter can be seen as a first step in quantifying the computational expense of disrespecting inner products. It illustrates the importance of employing the inner product induced by the control space for optimisations by deriving a rigorous estimate of the cost of failing to do so for a simple and generic problem. Simulations confirm the derived dependency of iteration counts mesh non-uniformity, while similar numerical results are found for a more complex PDE-constrained optimisation problem. These results provide an intuitive and analytically coherent model of mesh dependence and its corresponding errors for this class of problems. The ability to employ mesh refinement to resolve features of interest is a core advantage of the finite element method. Effectively dealing with non-uniformity in the mesh should therefore be a core capability of the optimisation method.

2.5 References

- [1] EL Allgower, K Böhmer, and S Mc Cormick. “Discrete correction methods for operator equations”. In: *Numerical Solution of Nonlinear Equations*. Springer, 1981, pp. 30–97.
- [2] EL Allgower and SF McCormick. “Newton’s method with mesh refinements for numerical solution of nonlinear two-point boundary value problems”. In: *Numerische Mathematik* 29.3 (1978), pp. 237–260.
- [3] EL Allgower et al. “A mesh-independence principle for operator equations and their discretizations”. In: *SIAM Journal on Numerical Analysis* 23.1 (1986), pp. 160–169.
- [4] Eugene L Allgower, F St McCormick, and Daniel V Pryor. “A general mesh independence principle for Newton’s method applied to second order boundary value problems”. In: *Computing* 23.3 (1979), pp. 233–246.
- [5] Walter Alt. “Discretization and mesh-independence of Newton’s method for generalized equations”. In: *Lecture Notes in Pure and Applied Mathematics* (1997), pp. 1–30.
- [6] Walter Alt. “Mesh-independence of the Lagrange–Newton method for nonlinear optimal control problems and their discretizations”. In: *Annals of Operations Research* 101.1-4 (2001), pp. 101–117.
- [7] Ioannis K Argyros. “A mesh-independence principle for nonlinear operator equations and their discretizations under mild differentiability conditions”. In: *Computing* 45.3 (1990), pp. 265–268.
- [8] Owe Axelsson. “On mesh independence and Newton-type methods”. In: *Applications of Mathematics* 38.4 (1993), pp. 249–265.
- [9] Steven J Benson et al. *TAO user manual (revision 1.8)*. Tech. rep. ANL/MCS-TM-242, Mathematics and Computer Science Division, Argonne National Laboratory, 2005.

- [10] Richard H Byrd et al. “A limited memory algorithm for bound constrained optimization”. In: *SIAM Journal on Scientific Computing* 16.5 (1995), pp. 1190–1208.
- [11] Peter Deufhard and Florian A Potra. “Asymptotic mesh independence of Newton-Galerkin methods via a refined Mysovskii theorem”. In: *SIAM Journal on Numerical Analysis* 29.5 (1992), pp. 1395–1412.
- [12] A. L. Dontchev, W. W. Hager, and V. M. Veliov. “Uniform Convergence and Mesh Independence of Newton’s Method for Discretized Variational Problems”. In: *SIAM Journal on Control and Optimization* 39.3 (2000), pp. 961–980.
- [13] P. E. Farrell et al. “Automated derivation of the adjoint of high-level transient finite element programs”. In: *SIAM Journal on Scientific Computing* 35 (2013), pp. C359–393. DOI: 10.1137/120873558.
- [14] I Fried. “Bounds on the Extremal Eigenvalues of the Finite Element Stiffness and Mass Matrices and their Spectral Condition Number”. In: *Journal of Sound and Vibration* 22.4 (1972), pp. 407–418.
- [15] Simon W Funke and Patrick E Farrell. “A framework for automated PDE-constrained optimisation”. In: *arXiv preprint arXiv:1302.3894* (2013).
- [16] Philip E Gill, Walter Murray, and Michael A Saunders. “SNOPT: An SQP algorithm for large-scale constrained optimization”. In: *SIAM review* 47.1 (2005), pp. 99–131.
- [17] Andreas Günnel, Roland Herzog, and Ekkehard Sachs. “A note on preconditioners and scalar products in Krylov subspace methods for self-adjoint problems in Hilbert space”. In: *Electronic Transactions on Numerical Analysis* 41 (2014), pp. 13–20.
- [18] M Heinkenschloss, M Laumen, and EW Sachs. “Gauss-Newton methods with grid refinement”. In: *Estimation and Control of Distributed Parameter Systems*. Springer, 1991, pp. 161–174.
- [19] Matthias Heinkenschloss. “Mesh independence for nonlinear least squares problems with norm constraints”. In: *SIAM Journal on Optimization* 3.1 (1993), pp. 81–117.
- [20] Antoine Henrot. “Minimization problems for eigenvalues of the Laplacian”. In: *Nonlinear Evolution Equations and Related Topics*. Springer, 2003, pp. 443–461.
- [21] Roland Herzog and Karl Kunisch. “Algorithms for PDE-constrained optimization”. In: *GAMM-Mitteilungen* 33.2 (2010), pp. 163–176.
- [22] Michael Hintermüller and Michael Ulbrich. “A mesh-independence result for semismooth Newton methods”. In: *Mathematical Programming* 101.1 (2004), pp. 151–184.
- [23] X Huang and YM Xie. “Convergent and mesh-independent solutions for the bi-directional evolutionary structural optimization method”. In: *Finite Elements in Analysis and Design* 43.14 (2007), pp. 1039–1049.
- [24] Eric Jones, Travis Oliphant, Pearu Peterson, et al. *SciPy: Open source scientific tools for Python*. <http://www.scipy.org>. 2001–.

- [25] Yoshiaki Kawajir et al. *Introduction to Ipopt: A tutorial for downloading, installing, and using Ipopt*. <https://projects.coin-or.org/Ipopt/>. 2015.
- [26] CT Kelley and Ekkehard W. Sachs. “Approximate quasi-Newton methods”. In: *Mathematical Programming* 48.1-3 (1990), pp. 41–70.
- [27] CT Kelley and EW Sachs. “Mesh independence of the gradient projection method for optimal control problems”. In: *SIAM journal on control and optimization* 30.2 (1992), pp. 477–493.
- [28] CT Kelley and EW Sachs. “Quasi-Newton methods and unconstrained optimal control problems”. In: *SIAM Journal on Control and Optimization* 25.6 (1987), pp. 1503–1516.
- [29] Robert C Kirby. “From functional analysis to iterative methods”. In: *SIAM review* 52.2 (2010), pp. 269–293.
- [30] Anders Logg, Kent-Andre Mardal, and Garth Wells. *Automated solution of differential equations by the finite element method: The FEniCS book*. Vol. 84. Springer Science & Business Media, 2012.
- [31] D. G. Luenberger and Y. Yinyu. *Linear and Nonlinear Programming*. 3rd Edition, Springer Science+Business Media, LLC, 2008.
- [32] Kent-Andre Mardal and Ragnar Winther. “Preconditioning discretizations of systems of partial differential equations”. In: *Numerical Linear Algebra with Applications* 18.1 (2011), pp. 1–40.
- [33] *MATLAB and Optimization Toolbox Release 2016a, The MathWorks, Inc., Natick, Massachusetts, United States*. <http://mathworks.com>. 2016.
- [34] SF McCormick. “A revised mesh refinement strategy for Newton’s method applied to nonlinear two-point boundary value problems”. In: (1978), pp. 15–23.
- [35] M Nordaas and Simon W. Funke. *The Moola optimisation package*. <https://github.com/funsim/moola>. 2016.
- [36] Tobias Schwedes, Simon W. Funke, and David A. Ham. “An iteration count estimate for a mesh-dependent steepest descent method based on finite elements and Riesz inner product representation”. In: *arXiv preprint arXiv:1606.08069* (2016).
- [37] Tobias Schwedes, David Ham, and Simon Wolfgang Funke. *Code for simulations in Chapter 2 of ResearchBrief*. Dec. 2016. DOI: 10.5281/zenodo.224242. URL: <https://doi.org/10.5281/zenodo.224242>.
- [38] Shlomo Ta’asan. *‘One Shot’ Methods for Optimal Control of Distributed Parameter Systems I: Finite Dimensional Control*. Tech. rep. DTIC Document, 1991.
- [39] Shlomo Ta’asan, G Kuruvila, and M Salas. “Aerodynamic design and optimization in one shot”. In: *30th Aerospace Sciences Meeting and Exhibit*. 1992, p. 25.
- [40] Michael Ulbrich. “Semismooth Newton methods for operator equations in function spaces”. In: *SIAM Journal on Optimization* 13.3 (2002), pp. 805–841.

- [41] Stefan Volkwein. “Mesh-independence for an augmented Lagrangian-SQP method in Hilbert spaces”. In: *SIAM Journal on Control and Optimization* 38.3 (2000), pp. 767–785.
- [42] Andreas Wächter and Lorenz T Biegler. “On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming”. In: *Mathematical programming* 106.1 (2006), pp. 25–57.
- [43] J. Young. *Optizelle: An open source software library designed to solve general purpose nonlinear optimization problems*. www.optimojoe.com, Open source software. 2016.

Chapter 3

An application: Optimising the layout of tidal turbine arrays

3.1 Introduction

The motivation for this application comes from the desire to generate renewable energy from the tides in the Earth's oceans. Tides are a consequence of the gravitational attractions experienced within the Earth-Moon-Sun system. As this system evolves in time the cumulative gravitational forces vary and this has the effect of driving ocean (or tidal) currents which periodically and locally increase and decrease the depth of the oceans at global scales. As these tidal motions move from the deep ocean into the shallower continental shelf regions, conservation of volume accelerates the currents. They may then be even further accelerated as they interact with coastal features such as headlands and as they are constrained through narrow straits between islands. We thus have a situation where significant amounts of kinetic energy is contained within the coastal ocean relatively close to locations on shore where electricity is required. This has naturally led to significant interest in exploiting this resource through the renewable generation of electricity from tidal currents. Tides have the significant advantage over other renewable energy sources, such as solar, wind or wave, in that they are periodic and predictable, and hence do not suffer from supply reliability drawbacks. This tidal energy can be captured through tidal turbines in the case of the kinetic energy of currents discussed above, as well as barrages or lagoons in the case of the potential energy contained within tidal highs and lows. Here we consider the former case where potentially hundreds of tidal turbines may be deployed in arrays (Fig. 3.1), in much the same way that arrays of wind turbines operate.

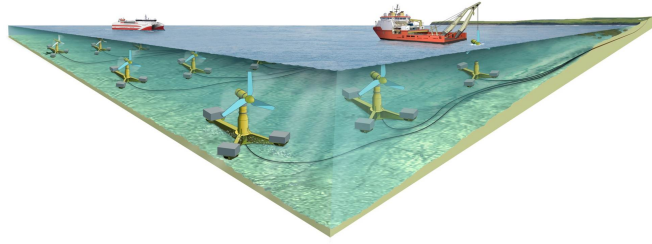


Fig. 3.1: Illustration of a tidal stream generator array. Image credit: MeyGen Limited.

Once tidal turbines are installed within a tidal current they will alter the flow across a range of scales. For example, a turbulent wake will be formed in the immediate lee of the turbine where the current speed is reduced, with locally accelerated flow above, below and to either side of the turbine and its wake (the bypass flow region). The power generated by a turbine is approximately dependent on the cube of the current speed it experiences. Hence, whether a down-stream turbine is installed within a wake zone or within the locally accelerated bypass flow will have a significant effect on the power yield of that turbine. Furthermore, large arrays can not only affect the local currents, but also the tidal resource at a regional scale. This regional effect limits the overall energy that can be extracted from a tidal array.

The optimal design of a tidal turbine array, in terms of the array location, the total number of turbines making up the array, and their individual locations (or micro-siting), is thus important in terms of array power yield and hence economic viability. The turbine layout optimisation problem is however very challenging: it has a large number of design parameters and, as discussed above, is coupled to the nonlinear hydrodynamics across a range of scales. Different approaches have been proposed to solve this problem, see for example [6, 9, 15, 17, 18], and for an overview see [20].

The goal of this chapter is to formulate the array layout problem as a PDE-constrained optimisation problem and solve it with the techniques presented earlier in this book. The hydrodynamic model will be based on the nonlinear shallow water equations, discretised with the finite element method on an unstructured mesh. One option to incorporate tidal turbines into the model is through locally enhanced seabed friction values at each turbine location. However, this approach requires relatively high mesh resolution and hence computational cost, because the size of a turbine is typically small with respect to the region considered for the (tidal/resource-scale) simulation. To mitigate this cost, here we represent the entire tidal array configuration via a turbine density function [5, 11, 19]. This density function is then used to locally increase the seabed friction coefficient to account for the impact of the local density of turbines on the flow. This “continuous turbine approach” allows for the optimisation of large arrays, or even multiple arrays close enough to interact with one another, within a regional context with relatively coarse meshes. An opti-

mised turbine density function can then be used to infer the individual locations of turbines, and/or used as an initial guess at the array layout for a more costly discrete micro-siting optimisation [11].

3.2 Problem formulation

We treat the problem of finding the optimal configuration of a continuous turbine density function (or equivalently an enhanced seabed friction field) as a PDE-constraint optimisation problem:

$$\begin{aligned} & \max_{u, \eta, d} J(u, \eta, d), \\ & \text{subject to } F(u, \eta, d) = 0, \\ & 0 \leq d \leq d_u, \end{aligned} \tag{3.1}$$

where u is the horizontal (depth-averaged) velocity vector, η is the free surface displacement from a state of rest, d is the turbine density function, $J(u, \eta, d) \in \mathbb{R}$ is the functional of interest, $F(u, \eta, d) = 0$ represents the nonlinear shallow water equations, and d_u is an upper bound for the turbine density function. The choice of the function space D containing d is essential for this work and is discussed in Sect. 3.2.3.

Assuming that through the PDE constraint equation $F(u, \eta, d) = 0$, any given $d \in D$ is uniquely mapped to a solution u, η through the shallow water equations, we may regard u, η as implicit functions of the design parameters d , that is $u \equiv u(d)$ and $\eta \equiv \eta(d)$. By substituting this into the functional of interest J , one obtains the reduced optimisation problem

$$\begin{aligned} & \max_{d \in D} J(u(d), \eta(d), d), \\ & \text{subject to } 0 \leq d \leq d_u. \end{aligned} \tag{3.2}$$

3.2.1 Design parameters and box constraints

The design parameter d is a continuous turbine density function on the domain of interest Ω . More specifically, $d|_A \equiv 0$ for some area $A \subset \Omega$ implies that no turbines are deployed inside A , whereas $d|_A \gg 0$ implies that turbines are densely deployed within A . The turbine density field can be regarded as an approximation for the actual placement configuration of turbines within the array. The integrated turbine density function additionally provides an approximation for the number of turbines N of the configuration:

$$N = \int_{\Omega} d(x) \, dx. \quad (3.3)$$

The point-wise bound constraint in (3.2) restricts d to feasible values. Firstly, the turbine density may only take non-negative values. Secondly, non-zero values are only expected in areas that are suitable for turbine placement. For instance, if an area A is too steep, too shallow, or environmentally protected, no turbines should be placed there, i.e. $d_u|_A \equiv 0$. All other locations combine to form valid array area(s), denoted Ω_{array} . In addition, the turbine density is assumed to be bounded above by a constant $\bar{d} > 0$. This bound takes into account that the number of turbines per unit area is limited, for example by a minimum feasible distance between any two turbines, or by hydrodynamic, installation and maintenance constraints. We conclude that

$$d_u(x) = \begin{cases} \bar{d} & \text{if } x \in \Omega_{\text{array}} \\ 0 & \text{if } x \in \Omega \setminus \Omega_{\text{array}}. \end{cases} \quad (3.4)$$

3.2.2 The PDE constraint

The optimisation problem (3.1) is subject to physical laws which are incorporated via the PDE $F(u, \eta, d) = 0$. Here, the PDE represents the nonlinear shallow water equations in steady state, that is

$$\begin{aligned} u \cdot \nabla u - \nu \nabla^2 u + g \nabla \eta + \frac{c_b + c_t(d)}{H} \|u\| u &= 0, \\ \nabla \cdot (Hu) &= 0, \end{aligned} \quad (3.5)$$

where $u : \Omega \rightarrow \mathbb{R}^2$ and $\eta : \Omega \rightarrow \mathbb{R}$ denote the depth-averaged horizontal velocity vector and free-surface displacement respectively, $H : \Omega \rightarrow \mathbb{R}$ is the total water depth (that is, $H = \eta + h$, where h is the water depth at rest, see Fig. 3.2), $g \approx 9.81 \text{ms}^{-2}$ is the gravitational acceleration, $\nu > 0$ is the kinematic viscosity coefficient, $c_b : \Omega \rightarrow [0, \infty)$ denotes the natural or background seabed friction value which may be spatially varying if this knowledge/data is available, and $c_t(d) : \Omega \rightarrow [0, \infty)$ is the additional ‘friction’ exerted on the flow by the presence of the turbines as encoded by the turbine density field d . The shallow water equations will be derived in detail in Sect. 3.3.

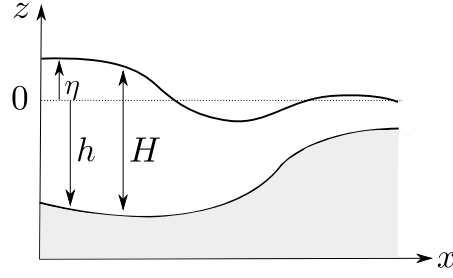


Fig. 3.2: Variables for the shallow water equations

3.2.3 The class of turbine density functions

When considering the optimisation problem (3.1), the question arises as to what class of functions are reasonable to consider for the turbine density d . At first glance, we expect the total number of turbines given by (3.3) to be finite, that is $d \in L^1(\Omega)$. If we were to assume that $d \in L^2$, a simple calculation using Cauchy-Schwarz and the finiteness of Ω convinces us that this implies that $d \in L^1(\Omega)$:

$$\|d\|_{L^1(\Omega)} = \int_{\Omega} d(x) dx = \langle d, 1 \rangle_{L^2(\Omega)} \leq \|d\|_{L^2(\Omega)} \cdot |\Omega|^{1/2} < \infty,$$

where $(\cdot, \cdot)_{L^2(\Omega)}$ denotes the inner product in $L^2(\Omega)$, and $\|\cdot\|_{L^2(\Omega)}$ and $\|\cdot\|_{L^1(\Omega)}$ denote the norms in $L^2(\Omega)$ and $L^1(\Omega)$, respectively. $L^2(\Omega)$ has the advantage over $L^1(\Omega)$ of being a Hilbert space, that is, it possesses an inner product and strong functional analytical results are valid such as the Riesz representation theorem. Therefore, functionals on $L^2(\Omega)$ have a unique representative in $L^2(\Omega)$ for their Fréchet derivative. This becomes particularly relevant when solving (3.1) using gradient-based methods which naturally require that representative. The question arises why not considering $d \in W^{k,2}(\Omega)$ for $k > 1$, such as $H^1(\Omega)$ for example. Such functions have weak derivatives that are $L^2(\Omega)$ -integrable. That means they satisfy smoothness assumptions that we do not necessarily expect from, nor wish to impose on, the turbine density function. In fact, the continuous turbine approach is a practical means to approximate the optimal distribution of real, i.e. discrete individual, turbines. In reality, we naturally expect to observe discontinuities in the turbine distribution as at every point of the domain there is either a turbine in place or not, as well as the fact that the domain is made up of discrete regions where turbines are permitted. Therefore, we shall use a space that reflects this non-smoothness. In this context and for the reasons mentioned above, we believe that the choice of $L^2(\Omega)$ as the class of turbine density functions is sensible.

3.2.4 Relationship between turbine density and seabed friction

Referring to [11], the force exerted on the flow due to the enhanced friction term $c_t(d)$ produced by the turbine field in the shallow water equations (3.5) is given by

$$F_{\text{array}} = \int_{\Omega_{\text{array}}} \rho c_t(d(x)) \|u(x)\| u(x) dx. \quad (3.6)$$

Using a common parameterisation for the drag force exerted by an individual turbine in 3D, the drag force exerted by an array of N turbines can be approximated as

$$F_{\text{array}} = \sum_{i=1}^N \frac{1}{2} \rho C_t A_t \|u_{\infty,i}\| u_{\infty,i}, \quad (3.7)$$

where C_t denotes the thrust coefficient corresponding to an individual turbine and A_t its cross-sectional area. Here we assume that all turbines are identical and are always aligned with the flow and so have common values for C_t and A_t . Further, $u_{\infty,i}$ denotes the free-stream or undisturbed velocity, i.e. the velocity that would be encountered at the i th turbine's location without the presence of the turbine. Here we simply take this to be the velocity at the turbine location from the model and note that this is a reasonable approximation as the local velocity of the model gets closer to the free-stream velocity the more the turbine density field is 'spread' through use of a relatively coarse computational mesh. See [1, 11, 12] for further discussions on this point, including corrections that are possible as mesh resolution is increased.

Re-formulating the drag force from its discrete version in (3.7) into a continuous one by replacing the sum by an integral containing the turbine density field yields

$$F_{\text{array}} = \int_{\Omega_{\text{array}}} \frac{1}{2} \rho C_t A_t d(x) \|u(x)\| u(x) dx. \quad (3.8)$$

Equating with (3.6) leads to an expression for the friction function c_t in terms of the turbine density function d :

$$c_t(d) = \frac{1}{2} C_t A_t d. \quad (3.9)$$

3.2.5 The functional of interest

The functional of interest J in the optimisation problem (3.1) refers to the quantity that is to be maximised. In what follows we define two reasonable expressions for the functional of interest.

3.2.5.1 Power functional

One natural choice for the functional of interest is the total power which the array (or here the turbine density field) extracts from the flow. From (3.6) and the formula for mechanical power, $P = F_{\text{array}} \cdot u$, this functional can be expressed as:

$$J_{\text{power}}(u, d) = \int_{\Omega_{\text{farm}}} \rho c_t(d(x)) \|u(x)\|^3 dx. \quad (3.10)$$

Note that the actual electricity produced by the turbine array is only a fraction of 3.10, as the functional as written also includes energy losses due to wake mixing effects behind turbines. Furthermore, one needs to account for the power losses due to power transformation to electrical energy, losses due to transporting the energy from turbines to the grid, and by the presence of turbine support structures which contribute part of the turbine induced friction.

3.2.5.2 Profit functional

In the context of evaluating possible turbine arrays from an economic perspective, a reasonable choice for the functional of interest is the profit of an array represented by its density function, generated over its entire lifetime. A simple choice for a profit functional is

$$\begin{aligned} J_{\text{profit}}(u, d) &= \text{Revenue}(u, d) - \text{Cost}(d) \\ &= IkT J_{\text{power}}(u, d) - C \int_{\Omega} d(x) dx, \end{aligned} \quad (3.11)$$

where $J_{\text{power}}(u, d)$ as given in (3.10) is the extracted power, T a turbine's average lifetime, $k \in (0, 1)$ is a turbine efficiency coefficient, I is some income factor stating the financial value of the power generated, and C is the cost of installing and maintaining one turbine. Recall from (3.3) that the integral of d gives the total number of installed turbines. See [11] for further details.

3.3 Shallow water equations

In this section, for completeness, we review the derivation of the shallow water equations which are the partial differential equations that describe the flow of the system we consider. The shallow water equations are a two-dimensional (depth-averaged) approximation to the three-dimensional Navier-Stokes equations and are valid for scenarios in which the horizontal length scale is much greater than the vertical length scale.

3.3.1 Physical principles

The shallow water equations are based on depth-integrating or averaging the Navier-Stokes equations and therefore are a consequence of the principles of conservation of mass and momentum. Thus, we begin by briefly deriving these principles.

3.3.1.1 Conservation of mass

One essential law of physics is the conservation of mass. It states that the rate of change of the total mass within an arbitrary control volume V equals the net mass flux across its boundary ∂V . Mathematically, this can be expressed as

$$\frac{d}{dt} \int_V \rho \, dx = - \int_{\partial V} \rho u \cdot n \, dA, \quad (3.12)$$

where ρ is the density of the fluid, u is the fluid velocity and n is the outward pointing unit normal vector to ∂V . Let us assume that V does not vary in time and that ρ is smooth. Applying Gauss's theorem to the right hand side allows us to write

$$\int_V \left(\frac{\partial \rho}{\partial t} - \nabla \cdot (\rho u) \right) dx = 0. \quad (3.13)$$

As V was an arbitrary control volume, we can conclude the pointwise relationship

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho u) = 0. \quad (3.14)$$

Note on the other hand that Eq. (3.14) trivially implies (3.12), and hence is equivalent to mass conservation. The PDE (3.14) is referred to as the continuity equation.

3.3.1.2 Conservation of momentum

Following Newton's second law, the rate of change of the total momentum in a control volume V equals the net flux of momentum across ∂V plus any internal forces exerted within V and any external forces exerted on ∂V . Mathematically this can be written as

$$\frac{d}{dt} \int_V \rho u \, dx = - \int_{\partial V} (\rho u) u \cdot n \, dA + \int_V \rho f_{\text{int}} \, dx + \int_{\partial V} S n \, dA,$$

where f_{int} denotes the internal force density and S denotes the Cauchy stress tensor. Examples of internal forces are those due to gravity, Coriolis (which is actually a fictitious force that is required when we make the usual ocean modelling assumption of considering our domain in a non-inertial, i.e. here a rotating, reference frame), electric or magnetic forces. Assuming that the momentum ρu is smooth and applying

Gauss's theorem results in

$$\int_V \left(\frac{\partial}{\partial t}(\rho u) + \nabla \cdot (\rho u u^T) - \rho f_{\text{int}} - \nabla \cdot S \right) dx = 0. \quad (3.15)$$

Since again V was chosen arbitrarily, we deduce that

$$\frac{\partial}{\partial t}(\rho u) + \nabla \cdot (\rho u u^T) - \rho f_{\text{int}} - \nabla \cdot S = 0. \quad (3.16)$$

The PDE (3.16) encodes the concept of conservation of momentum and is referred to as the momentum equation.

3.3.2 Navier-Stokes equations

In the following, we derive the Navier-Stokes equations for an incompressible Newtonian fluid using the principles of conservation of mass and momentum. For a Newtonian fluid, we have the stress tensor

$$S = -pI_3 + \tau, \quad (3.17)$$

where p denotes the pressure, I_3 the identity matrix in $\mathbb{R}^3$ and τ is the deviatoric stress tensor. Stokes' stress constitutive equation used for incompressible fluid states that

$$\tau = \mu \left(\nabla u + (\nabla u)^T \right), \quad (3.18)$$

where μ is the, assumed constant, dynamic viscosity coefficient of the fluid ($\mu = \rho \nu$ where ν is termed the kinematic viscosity). Thus, we find for the divergence of the Cauchy stress tensor:

$$\nabla \cdot S = -\nabla p + \mu \nabla^2 u. \quad (3.19)$$

Neglecting all internal forces besides gravity, we have $f_{\text{int}} = g$, where g is the gravitational acceleration vector. In addition to incompressibility, which states that ρ is independent of p , we assume that ρ is a constant in time and throughout the fluid, e.g. ignoring thermal effects. Applying (3.19), we therefore may infer from the mass continuity equation (3.14) and the momentum equation (3.16) the following system of PDEs:

$$\begin{aligned} \nabla \cdot u &= 0, \\ \frac{\partial u}{\partial t} + \nabla \cdot (u u^T) - \nu \nabla^2 u + \frac{1}{\rho} \nabla p - g &= 0. \end{aligned} \quad (3.20)$$

The equation pair (3.20) are known as the isothermal Navier-Stokes equations for an incompressible Newtonian fluid. If f contains another internal forcing vector, i.e. $f_{\text{int}} = g + f$, we get an additional f term on the right hand side of (3.20).

3.3.3 Shallow water equations

In what follows we sketch the derivation of the shallow water equations via the depth-integration, or averaging, of the Navier-Stokes equations (3.20). For a more complete derivation and mathematical discussions see [21], [3]. With $u = (u_1, u_2, u_3)^T$, the momentum equation in (3.20) may be written in non-vector form as

$$\begin{aligned} \frac{\partial u_1}{\partial t} + \sum_{i=1}^3 u_i \frac{\partial u_1}{\partial x_i} - \nu \sum_{j=1}^3 \frac{\partial^2 u_1}{\partial x_j^2} + \frac{1}{\rho} \frac{\partial p}{\partial x_1} &= 0, \\ \frac{\partial u_2}{\partial t} + \sum_{i=1}^3 u_i \frac{\partial u_2}{\partial x_i} - \nu \sum_{j=1}^3 \frac{\partial^2 u_2}{\partial x_j^2} + \frac{1}{\rho} \frac{\partial p}{\partial x_2} &= 0, \\ \frac{\partial u_3}{\partial t} + \sum_{i=1}^3 u_i \frac{\partial u_3}{\partial x_i} - \nu \sum_{j=1}^3 \frac{\partial^2 u_3}{\partial x_j^2} + \frac{1}{\rho} \frac{\partial p}{\partial x_3} + g &= 0, \end{aligned} \quad (3.21)$$

where g now represents the magnitude of the gravitational acceleration vector. Further, the continuity equation in (3.20) may be written as

$$\frac{\partial u_1}{\partial x_1} + \frac{\partial u_2}{\partial x_2} + \frac{\partial u_3}{\partial x_3} = 0. \quad (3.22)$$

The shallow water equations are based on the essential assumption that the horizontal length scale of the dynamics, e.g. its wavelength, is much larger than the vertical length scale, e.g. the water depth at rest. Based upon this and via a scaling analysis of the terms in the vertical momentum equation we may conclude that vertical acceleration as well as viscous effects are small, and that this equation reduces to the so-called hydrostatic balance relation

$$\frac{1}{\rho} \frac{\partial p}{\partial x_3} = -g, \quad (3.23)$$

This states that pressure in our system is well approximated by its hydrostatic value, i.e. the weight of water above the location in question. Integrating (3.23) in the vertical, i.e. with respect to x_3 , yields

$$p = \rho g(\eta - x_3) + p_a, \quad (3.24)$$

where η is the free-surface displacement from a state of rest (Fig. 3.2), and $p_a = p(x_1, x_2, \eta)$ denotes the atmospheric pressure. For the sake of simplicity we further assume that $\nabla p_a = 0$. For applications where this term becomes relevant, e.g. storm

surge modelling, it can easily be incorporated as an additional source term in the final equations. Differentiating (3.24) in the horizontal yields

$$\frac{\partial p}{\partial x_1} = \rho g \frac{\partial \eta}{\partial x_1}, \quad \frac{\partial p}{\partial x_2} = \rho g \frac{\partial \eta}{\partial x_2}. \quad (3.25)$$

Noting that this pressure gradient forcing is independent of depth, x_3 , allows us to conclude from the horizontal momentum equations that if the horizontal velocities are initially independent of depth then they will remain so. While this is the case we assume here, and is entirely consistent with the simulation of tidal dynamics at large scales, it should be noted that this assumption would break down at small scales including when the flow interacts with individual turbines. The use of fully-3D equation sets and models in the design of turbine arrays is the subject of ongoing and future work, e.g. see [1].

We can thus interpret the horizontal velocities as depth-averaged values, or depth-integrals following multiplication by the layer depth, and within the horizontal momentum equations ignore all derivatives with respect to x_3 . More rigorous derivations, including arriving at a similar result from the vertical integration of the momentum equations, may be found in [21], [3]. We are thus left with new shallow water momentum equations

$$\begin{aligned} \frac{\partial u_1}{\partial t} + \sum_{i=1}^2 u_i \frac{\partial u_1}{\partial x_i} - \nu \sum_{j=1}^2 \frac{\partial^2 u_1}{\partial x_j^2} + g \frac{\partial \eta}{\partial x_1} &= 0, \\ \frac{\partial u_2}{\partial t} + \sum_{i=1}^2 u_i \frac{\partial u_2}{\partial x_i} - \nu \sum_{j=1}^2 \frac{\partial^2 u_2}{\partial x_j^2} + g \frac{\partial \eta}{\partial x_2} &= 0. \end{aligned} \quad (3.26)$$

We can now also remove the presence of the vertical derivative of u_3 in the continuity equation (3.22) by integrating over the total water depth:

$$\begin{aligned} \int_{-h}^{\eta} \nabla \cdot \mathbf{u} \, dx_3 &= \int_{-h}^{\eta} \left(\frac{\partial u_1}{\partial x_1} + \frac{\partial u_2}{\partial x_2} \right) dx_3 + u_3|_{x_3=\eta} - u_3|_{x_3=-h} \\ &= \frac{\partial}{\partial x_1} \int_{-h}^{\eta} u_1 \, dx_3 + \frac{\partial}{\partial x_2} \int_{-h}^{\eta} u_2 \, dx_3 - \left(u_1|_{x_3=\eta} \frac{\partial \eta}{\partial x_1} + u_1|_{x_3=-h} \frac{\partial h}{\partial x_1} \right) \\ &\quad - \left(u_2|_{x_3=\eta} \frac{\partial \eta}{\partial x_2} + u_2|_{x_3=-h} \frac{\partial h}{\partial x_2} \right) + u_3|_{x_3=\eta} - u_3|_{x_3=-h}, \end{aligned} \quad (3.27)$$

where h denotes the bathymetry, i.e. the water depth at rest. The following so-called kinematic boundary conditions state that the normal components of the flow at the bottom of the domain ($x_3 = -h$) and at the free surface ($x_3 = \eta$) are equal to zero:

$$\begin{aligned} \left(u_1 \frac{\partial h}{\partial x_1} + u_2 \frac{\partial h}{\partial x_2} + u_3 \right) \Big|_{x_3=-h} &= 0, \\ \left(\frac{\partial \eta}{\partial t} + u_1 \frac{\partial \eta}{\partial x_1} + u_2 \frac{\partial \eta}{\partial x_2} - u_3 \right) \Big|_{x_3=\eta} &= 0. \end{aligned} \quad (3.28)$$

This states that particles cannot traverse these surfaces. Since the horizontal velocities are assumed to be independent of x_3 we have

$$\frac{\partial}{\partial x_i} \int_{-h}^{\eta} u_i dx_3 = \frac{\partial}{\partial x_i} (Hu_i) \quad \forall i = 1, 2, \quad (3.29)$$

where $H = h + \eta$ denotes the total water depth. Applying the boundary conditions (3.28) to (3.27) then results in

$$\frac{\partial \eta}{\partial t} + \frac{\partial}{\partial x_1} (Hu_1) + \frac{\partial}{\partial x_2} (Hu_2) = 0. \quad (3.30)$$

Equation (3.30) is termed the depth-averaged continuity equation. In 2D vector notation (3.30) and (3.26) may be written as

$$\frac{\partial \eta}{\partial t} + \nabla \cdot (Hu) = 0, \quad (3.31)$$

$$\frac{\partial u}{\partial t} + u \cdot \nabla u - \nu \nabla^2 u + g \nabla \eta = 0, \quad (3.32)$$

which we term the nonlinear shallow water equations. It is straightforward to see that in the case where we have an additional term F in the momentum equation (3.16) representing a force or source vector, it has to be added in the momentum equation (3.32). In this work, we have an additional bottom friction term given by the quadratic drag

$$F = -\frac{c_b + c_t(d)}{H} \|u\|u, \quad (3.33)$$

where c_b is the natural bottom friction coefficient and $c_t(d)$ the coefficient corresponding to the friction field induced by the turbine density d . Thus, the shallow water momentum equation we consider in this work takes the form

$$\frac{\partial u}{\partial t} + u \cdot \nabla u - \nu \nabla^2 u + g \nabla \eta + \frac{c_b + c_t(d)}{H} \|u\|u = 0. \quad (3.34)$$

Finally, the steady form of the system (3.31) and (3.34) follows from neglecting the time derivative terms and is often considered in situations where the velocity can be assumed to be time independent, or the fully time-dependent simulation is numerically too costly. With this simplification, we arrive at the PDE system stated in (3.5).

3.4 Aspects of numerical solution

This section discusses details of the numerical solution of problem (3.2). First, the shallow water equations are stated in their weak form and solved by applying a finite element discretisation (Sect. 3.4.1). From the resulting discrete solutions, the discrete adjoint equations are derived (Sect. 3.4.2). Their solution provides the discretised derivative of the functional of interest with respect to the turbine density, which is used in several gradient-based optimisation methods. Section 3.4.3 provides an illustration of what happens during one iteration of the optimisation loop performed in order to solve (3.2), describes the Python libraries used and performs an implementation verification via a Taylor remainder test.

3.4.1 Solving the shallow water equations

In what follows, we briefly explain the derivation of the weak formulation of the shallow water equations (3.5) in steady state and discuss how their weak form is solved using the finite element method. One derives the weak formulation of the shallow water equations by multiplying them by test functions $\Psi \in V$ and $\Phi \in Q$, respectively, integrating over the domain Ω and applying integration by parts to the viscosity and divergence terms. Using the notation $\langle \cdot, \cdot \rangle_A := \langle \cdot, \cdot \rangle_{L^2(A)}$ for $A \subset \overline{\Omega}$, this yields

$$\begin{aligned} \langle u \cdot \nabla u, \Psi \rangle_{\Omega} + \nu \langle \nabla u, \nabla \Psi \rangle_{\Omega} + g \langle \nabla \eta, \Psi \rangle_{\Omega} + \left\langle \frac{c_b + c_t(d)}{H} \|u\| u, \Psi \right\rangle_{\Omega} &= 0, \\ - \langle Hu, \nabla \Phi \rangle_{\Omega} + \langle Hu \cdot n, \Phi \rangle_{\partial \Omega_{\text{In}}} &= 0, \end{aligned} \quad (3.35)$$

where $\partial \Omega_{\text{In}}$ denotes the inflow boundary, on which strong Dirichlet conditions are imposed for the normal component of the velocity vector u . To ensure that all terms in (3.35) are well-defined, a natural choice for the test function spaces is $V = [H^1(\Omega)]^2$ and $Q = H^1(\Omega)$. The domain boundary can be decomposed into $\partial \Omega = \partial \Omega_{\text{In}} \cup \partial \Omega_{\text{Out}} \cup \partial \Omega_{\text{Coast}}$, where $\partial \Omega_{\text{In}}$ denotes the inflow boundary, $\partial \Omega_{\text{Out}}$ the outflow boundary and $\partial \Omega_{\text{Coast}}$ the coastal boundaries. For the weak formulation (3.35), we have used the following integration by parts relation:

$$\langle \nabla \cdot (Hu), \Phi \rangle_{\Omega} = - \langle Hu, \nabla \Phi \rangle_{\Omega} + \langle Hu \cdot n, \Phi \rangle_{\partial \Omega} \quad (3.36)$$

$$= - \langle Hu, \nabla \Phi \rangle_{\Omega} + \langle Hu \cdot n, \Phi \rangle_{\partial \Omega_{\text{In}}}, \quad (3.37)$$

where we applied an either weakly or strongly imposed homogeneous Dirichlet boundary conditions on $\partial \Omega_{\text{Coast}}$. Further, a strong Dirichlet boundary condition on η was imposed on the outflow boundary $\partial \Omega_{\text{Out}}$, hence the test function Φ vanishes on the boundary and thus the integral over $\partial \Omega_{\text{Out}}$ vanishes as well. Integration by parts has also been applied to the viscous term:

$$-\langle \Delta u, \Psi \rangle_{\Omega} = \langle \nabla u, \nabla \Psi \rangle_{\Omega} - \langle \nabla u \cdot n, \Psi \rangle_{\partial \Omega} \quad (3.38)$$

$$= \langle \nabla u, \nabla \Psi \rangle_{\Omega}. \quad (3.39)$$

The inflow boundary term disappears since strong Dirichlet boundary conditions are imposed on $\partial \Omega_{\text{In}}$, and therefore the test function Ψ vanishes. The outflow boundary term equals zero due to the velocity-pressure duality which, because of the strong Dirichlet boundary condition on η , imposes a zero Neumann boundary condition on u .

Problem (3.35) is solved using a finite element approach. The function spaces V and Q are replaced by finite dimensional subspaces $V_h \subset V$ and $Q_h \subset Q$ using a triangulation of the domain. V_h and Q_h are chosen as the Taylor-Hood P_2 - P_1 mixed finite element pair which satisfies the desirable inf-sup or Ladyzhenskaya-Babuka-Breezi (LBB) stability condition; see [2, Sect. 12.5] for details.

3.4.2 Adjoint equations

The adjoint equations can be derived with the approach presented in Sect. 1.4. The adjoint shallow water equations are

$$\begin{aligned} (\nabla \cdot u) \lambda_u - u \cdot \nabla \lambda_u - \nu \nabla^2 \lambda_u - H \nabla \lambda_{\eta} + \frac{c_b + c_t(d)}{H} \left(\|u\| \lambda_u + \frac{u \cdot \lambda_u}{\|u\|} u \right) &= \frac{\partial J^*}{\partial u}, \\ -g \nabla \cdot \lambda_u - \nabla \lambda_{\eta} \cdot u - \frac{c_b + c_t(d)}{H^2} \|u\| u &= 0, \end{aligned} \quad (3.40)$$

where λ_u is the adjoint velocity and λ_{η} the adjoint free-surface displacement. For a detailed derivation of (3.40), we refer to [10, Appendix C]. The functional gradient for a given turbine density field d is computed in three steps:

1. Solve the weak formulation of the shallow water equations given by (3.35) for u and η .
2. Solve (3.40) for λ_u and λ_{η} . Using the same finite element discretisation as in step 1 ensures that the gradient is consistent with the discrete shallow water model.
3. Evaluate the functional gradient using (1.148). The only term in the shallow water equations which explicitly depends on d is the one containing $c_t(d)$ with $c_t = \frac{1}{2} C_t A_t d$. Therefore,

$$\frac{\partial F}{\partial d}(u, \eta, d; \lambda_u, \lambda_{\eta}) = \frac{1}{2} \frac{C_t A_t}{H} \|u\| u \cdot \lambda_u,$$

and the gradient is evaluated as

$$\frac{dJ}{dd}(u, \eta, d) = -\frac{1}{2} \frac{C_t A_t}{H} \|u\| u \cdot \lambda_u + \frac{\partial J}{\partial d}(u, \eta, d).$$

3.4.3 Implementation

The implementation of the turbine layout optimiser is available in the open-source package OpenTidalFarm. Figure 3.3 visualises the optimisation loop and the software components that are used internally.

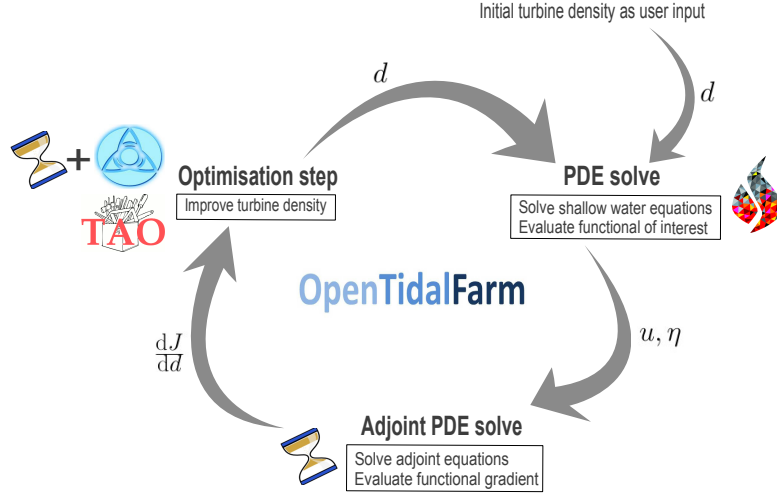


Fig. 3.3: The optimisation loop of the turbine layout designer: Initialised by a user-defined turbine density d , the FEniCS framework [13] is employed to solve the shallow water equations. In the next step, the adjoint shallow water equations and the functional gradient are derived and calculated using dolfin-adjoint [8]. In the optimisation step, we use either the interior point method from Optizelle [22] or the LMVM method provided by TAO [14]. This yields an improved turbine density, which is given to the PDE step and the procedure is iterated until a convergence criteria is met

To test the correctness of the adjoint equation and the functional gradient, a Taylor convergence test can be performed as described in Sect. 1.4.6. The Taylor convergence tests are performed here for three different choices of turbine friction functions, defined by

$$\begin{aligned}
c_{t,1}(x,y) &= 0.03, \\
c_{t,2}(x,y) &= 3 \cdot 10^{-9} \cdot (1 + x^2 + 2y^2), \\
c_{t,3}(x,y) &= 0.01 \cdot (2 + \sin(2\pi x) - \cos(2\pi y)),
\end{aligned}$$

for $(x,y) \in \Omega_{\text{array}} = [1500, 2500]^2$, and equal to zero on $\Omega \setminus \Omega_{\text{array}}$ with $\Omega = [0, 4000]^2$. By applying the relation $d_i = 2c_{t,i}/(C_t A_t)$ with $C_t = 0.6$ and $A_t = 314.15\text{m}^2$, this corresponds to the turbine density functions given by

$$\begin{aligned}
d_1(x,y) &= 3.183 \cdot 10^{-4}, \\
d_2(x,y) &= 3.183 \cdot 10^{-11} \cdot (1 + x^2 + 2y^2), \\
d_3(x,y) &= 1.061 \cdot 10^{-4} \cdot (2 + \sin(2\pi x) - \cos(2\pi y)),
\end{aligned}$$

for $(x,y) \in \Omega_{\text{array}}$ and equal to zero on $\Omega \setminus \Omega_{\text{array}}$, where the physical unit m^{-2} is omitted.

The example turbine densities d_1, d_2 and d_3 all satisfy the inequality condition in (3.1) with d_u given in (3.4) with $\bar{d} \approx 6.25 \cdot 10^{-4} \text{m}^{-2}$. The $\bar{d}$ value is derived from $\bar{d} = 1/D_{\min}^2$ with a realistic minimal distance $D_{\min} \approx 40\text{m}$ between individual turbines. Moreover, d_1, d_2 and d_3 produce outcomes for the power functional in a range that we also encounter in our more realistic simulations.

All other parameters relevant for the turbine array optimisation problem, particularly the model parameters used in the shallow water equations, can be found in table 3.4, except the viscosity which is chosen here as $\nu = 50 \text{m}^2/\text{s}$. In the model, homogeneous Dirichlet boundary conditions for u are applied at the Northern and Southern boundaries to the domain. On the western boundary, an inflow is generated by imposing $\eta = 0.1$ while at the eastern outflow boundary $\eta = 0$ is imposed. The turbine density is represented by cubic Lagrange finite elements $\mathcal{P}^3$ and the underlying mesh contains 10,202 elements.

The test results for the first and second-order Taylor remainders for d_1, d_2 and d_3 are given in tables 3.1, 3.2 and 3.3, respectively. The first-order Taylor remainders decrease roughly by half between subsequent evaluations, and the second-order Taylor remainders decrease roughly by four. The convergence order should be equal to one for first-order Taylor remainders and two for second-order Taylor remainders. The results of the Taylor remainder demonstrate the expected orders of convergence, giving confidence that the adjoint and gradient computations are correct.

3.5 Mesh dependence in tidal turbine array layout optimisation

In this section, we analyse mesh dependent convergence in the PDE-constraint optimality problem presented in Sect. 3.2 with respect to different Riesz gradient representations for the functional of interest. Here, the functional of interest

h	$R_0(h\delta d)$	order	$R_1(h\delta d)$	order
0.01	$2.412 \cdot 10^6$		$4.020 \cdot 10^5$	
0.005	$1.302 \cdot 10^6$	0.89	$1.055 \cdot 10^5$	1.93
0.0025	$0.677 \cdot 10^6$	0.94	$0.270 \cdot 10^5$	1.96
0.00125	$0.345 \cdot 10^6$	0.97	$0.068 \cdot 10^5$	1.98
0.000625	$0.174 \cdot 10^6$	0.99	$0.017 \cdot 10^5$	1.99

Table 3.1: Taylor reminders $R_0 = |\hat{J}(d_1 + h\delta d) - J(d_1)|$ and $R_1 = |J(d_1 + h\delta d) - J(d_1) - h\nabla J(d_1)\delta d|$ for the turbine layout example with functional given by (3.10). The perturbation direction δd is a $\mathcal{P}^3$ function nodal values given by random samples from $[0, 1]$

h	$R_0(\delta d)$	order	$R_1(\delta d)$	order
0.01	$1.620 \cdot 10^6$		$2.752 \cdot 10^5$	
0.005	$0.876 \cdot 10^6$	0.89	$0.720 \cdot 10^5$	1.93
0.0025	$0.455 \cdot 10^6$	0.94	$0.184 \cdot 10^5$	1.97
0.00125	$0.232 \cdot 10^6$	0.97	$0.046 \cdot 10^5$	1.98
0.000625	$0.117 \cdot 10^6$	0.99	$0.012 \cdot 10^5$	1.99

Table 3.2: Taylor reminders $R_0 = |\hat{J}(d_2 + \delta d) - J(d_2)|$ and $R_1 = |J(d_2 + \delta d) - J(d_2) - \nabla J(d_2)\delta d|$ for the turbine layout example with functional given by (3.10). The perturbation direction δd is a $\mathcal{P}^3$ function nodal values given by random samples from $[0, 1]$

h	$R_0(\delta d)$	order	$R_1(\delta d)$	order
0.01	$4.192 \cdot 10^6$		$6.965 \cdot 10^5$	
0.005	$2.261 \cdot 10^6$	0.89	$1.834 \cdot 10^5$	1.93
0.0025	$1.175 \cdot 10^6$	0.94	$0.471 \cdot 10^5$	1.96
0.00125	$0.599 \cdot 10^6$	0.97	$0.19 \cdot 10^5$	1.98
0.000625	$0.302 \cdot 10^6$	0.99	$0.030 \cdot 10^5$	1.99

Table 3.3: Taylor reminders $R_0 = |\hat{J}(d_3 + \delta d) - J(d_3)|$ and $R_1 = |J(d_3 + \delta d) - J(d_3) - \nabla J(d_3)\delta d|$ for the turbine layout example with functional given by (3.10). The perturbation direction δd is a $\mathcal{P}^3$ function nodal values given by random samples from $[0, 1]$

is chosen as the profit functional from (3.11), using a profit margin $m = 0.4$, i.e. $J_{\text{profit}} = 0.4 \cdot \text{Revenue}$, and a cost coefficient $C/(IkT) = 452.39 \text{ kW}$. The other relevant parameters for our simulations are stated in table 3.4. Further, we define the domain $\Omega = [0, 2000] \times [0, 1000]$ and the farm area $\Omega_{\text{array}} = [750, 1250] \times [350, 650]$ with an initial turbine friction equal to zero (no turbines placed within the array). In the model, weak homogeneous Dirichlet boundary conditions for u are applied on the Northern and Southern boundaries of the domain. On the western side boundary, an inflow is generated by imposing $u = (2, 0)^T$ strongly, with the outflow boundary condition on the eastern boundary strongly imposed by $\eta = 0$. The turbine density is represented by discontinuous Galerkin finite elements. The underlying meshes used contain between 1258 and 8604 elements, depending on the mesh refinement (see Sect. 3.5.1). The optimisation problem is solved using a primal log-barrier interior point method [22] that allows for user-defined inner products. The optimisation is terminated when the $L^2(\Omega)$ norm of the functional gradient, computed using the $L^2(\Omega)$ inner product, is smaller than 10^{-7} .

The reader is reminded that this is a continuous optimisation problem, that is the turbine density with respect to which the optimisation is performed is a member of an infinite-dimensional function space, namely given by $L^2(\Omega)$. A discussion of this choice of function class was given in Sect. 3.2.3. Besides the PDE-constraint, which here is given by the nonlinear shallow water equations in steady-state form, the optimisation problem is subject to inequality constraints, that is a lower and upper bound for the turbine density function. For the optimisation step, the interior point method presented in Sect. 1.5.5 is applied, where its implementation allows the choice of the underlying inner product. Throughout this section, linear Lagrange finite elements were used for the spatial discretisation of the turbine density function.

Table 3.4: Parameters for the shallow water equations and the tidal turbine farm used for the continuous turbine density optimisation

Parameter	Symbol	Value	Units
Water density	ρ	1000	kg/m^3
Viscosity	ν	5	m^2/s
Water depth	h	50	m
Gravity	g	9.81	m/s^2
Natural bottom friction	c_b	0.0025	—
Turbine radius		10	m
Minimum distance between turbines		40	m
Maximum turbine density	$\bar{d}$	$6.25 \cdot 10^{-4}$	m^{-2}
Maximum turbine friction		0.059	—
Thrust coefficient	C_t	0.6	—
Turbine cross section	A_t	314.15	m^2

3.5.1 Mesh refinement

For the numerical analysis of the mesh dependent behaviour when solving this optimisation problem, two approaches for a non-uniform refinement of the underlying meshes are considered.

In the first approach, meshes are refined according to a random refinement rule: starting with a mesh that is uniform in the turbine farm area in the middle of the rectangular mesh, the $(i + 1)^{th}$ mesh is given by randomly refining the farm region in the i^{th} mesh, i.e. the refinement schema introduced in 2.3.2.1 is applied. The probability with which an element is labelled for refinement equals 0.15. The resulting randomly refined meshes used in the experiments are presented in Fig. 3.4.

In the second refinement approach, only a region around a single location within the mesh is refined. Starting with a mesh that is uniform in the turbine area, the resolution in element size around this point increases by a factor of five for every refinement. For the sake of saving space and due to the smaller complexity of this refinement schema, only one of these so-called point refined meshes is displayed, see Fig. 3.5.

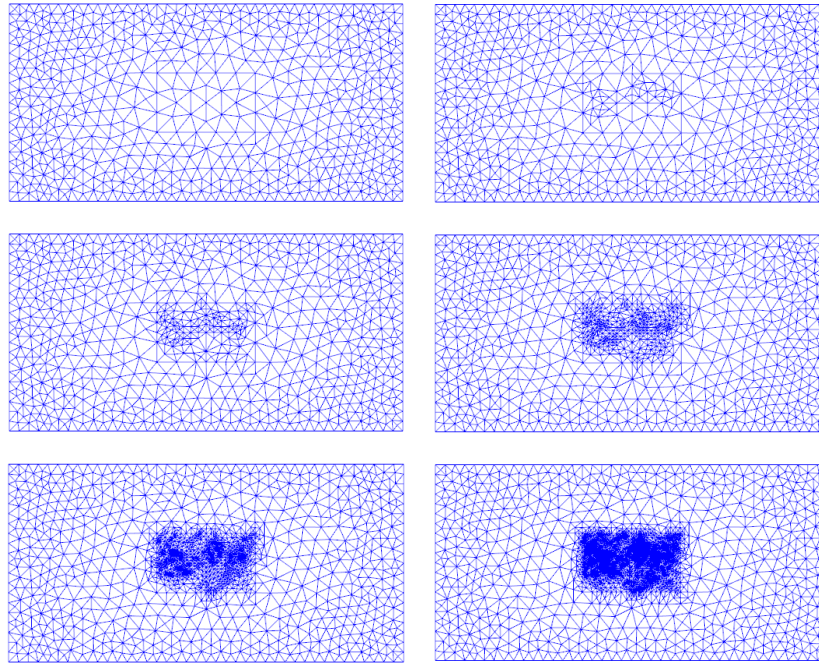


Fig. 3.4: Successively, randomly refined meshes in the turbine array area used for the simulations with results given in Fig. 3.8 and 3.9

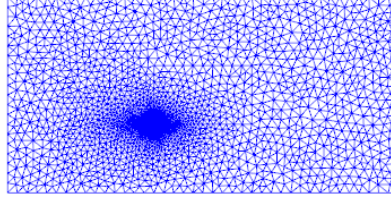


Fig. 3.5: Mesh with increased resolution around one point located on the boundary of the turbine region

3.5.2 Numerical experiments

In this section, the impact of using the ℓ^2 and L^2 Riesz representations of the functional derivative on the numerical solutions is analysed. Figure 3.6 displays the evolution of the profit functional with respect to iteration numbers for successively refined meshes.

The results for the randomly refined meshes are shown in Figs. 3.7a and 3.7b. We make the following observations: for the uniform mesh (refinement level 0), the algorithm using the ℓ^2 representation of the functional derivative converges to the optimal value in fewer iterations than with the L^2 representation. However, successively refining the mesh decreases the rate of convergence behaviour for the ℓ^2 based algorithm. Furthermore, the optimal profit functional value achieved decreases with increasing mesh refinements.

Other than for the case with the uniform mesh, the speed of the algorithm with respect to both the iteration number as well as the convergence limit for the reduced functional is larger if the functional derivative representation in L^2 is used in the optimisation. The convergence limit and speed is stable for the optimisation method using L^2 Riesz represented gradients over the range of the underlying meshes considered.

Overall, similar behaviour in the numerical results between randomly refined and point refined meshes from Figs. 3.7c and 3.7d is observed. However, the point refinement rule does not lead to convergence limits of the profit functional in the ℓ^2 case that lie significantly below the optimum achieved using L^2 Riesz represented gradients. The convergence limit of the profit functional is reached after around 30 iterations using the L^2 inner product, independent of the refinement level. Using the ℓ^2 inner product, the iteration number increases with refinement. On the finest mesh, around 80 iterations are necessary.

The gradients with respect to both inner product representations have a different geometrical structure, see Fig. 3.7. In the first iteration ($d(x) = 0$), the gradients represented with respect to L^2 demonstrate a mostly flat increase of the friction field over the entire turbine area. It is further interesting to observe the Galerkin mass matrix scaling in the point refinement case for the ℓ^2 gradient representation

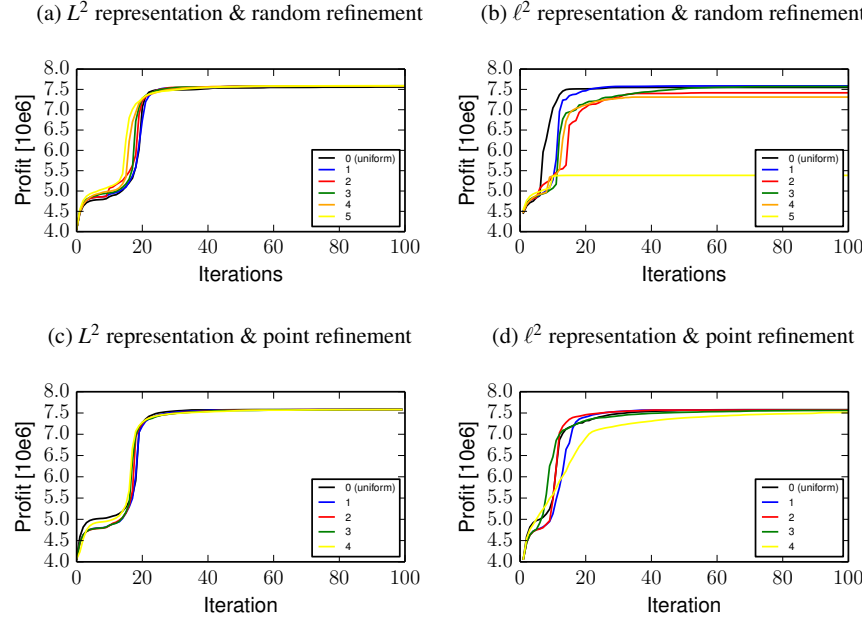


Fig. 3.6: Functional of interest given by the profit functional plotted as a function of optimisation iterations using L^2 and ℓ^2 gradient representations for successively refined meshes. For the upper figures, the random refinement rule is applied. For the lower figures, the point refinement rule is applied. The numbers in the legend state the degree of refinement of the underlying mesh

in such a clear manner: The smaller the size of the underlying element, the smaller the gradient values on this element.

Figures 3.8 and 3.9 present plots of the optimal friction field associated with the optimal turbine density function, obtained using the L^2 and ℓ^2 inner product representation for the optimisation method, respectively. The underlying meshes correspond to those shown in Fig. 3.4 (randomly refined meshes).

The reader is reminded that the boundary conditions for the shallow water equations, that are discussed in Sect. 3.4.1, describe a constant inflow from the western side of the domain towards the eastern side, and in this simple flat-bottomed channel geometry the flow remains close to west to east throughout the domain. Based on this information, the following consideration for an optimal turbine density distribution can be made: To maximise the power output (as part of the profit functional), a friction field associated with the turbine density field is formed that extracts the most possible power from the flow. Starting from $d = 0$, it makes sense if a considerable part of the increased friction field is orthogonal to the velocity. Due to the presence of increased friction in the farm area, and a consequent ‘blockage’ effect, the flow is diverted towards the Northern and Southern boundaries of the domain. To

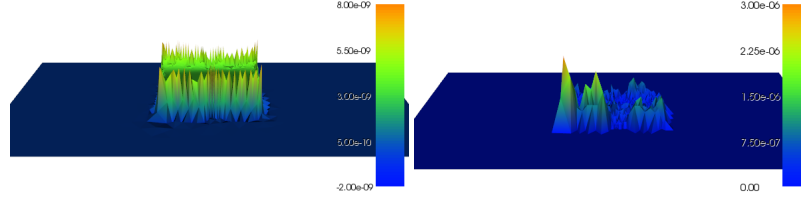
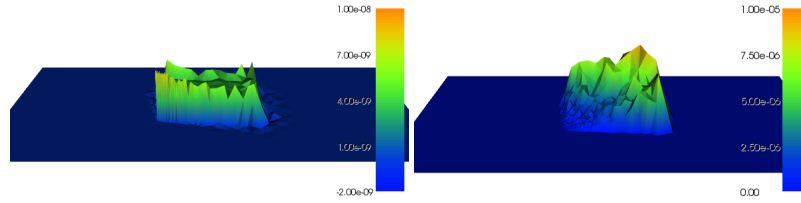
(a) L^2 representation & random refinement (b) ℓ^2 representation & random refinement(c) L^2 representation & point refinement (d) ℓ^2 representation & point refinement

Fig. 3.7: Plots of the L^2 and ℓ^2 representations of the functional derivative on the finest mesh for both refinement schemas (top: random refinement; bottom: point refinement) for $d = 0$

‘capture’ most of the flow, and hence maximise energy extraction, the friction along the Northern and Southern boundaries of the turbine region might be expected to increase. Summarizing, one might expect the optimal friction distribution to have a “U” shape with the open side to the western side of the domain. This physical intuition is confirmed by the results given in Fig. 3.8, where the L^2 inner product was used. In this case, the overall friction field distribution does not change significantly when refining the underlying mesh, which confirms the observation of mesh independence in the optimisation for the reduced functional above.

Considering the results for the optimal friction distribution displayed in Fig. 3.9 (using ℓ^2) we observe that the friction field in this case does change with mesh refinement. Distributions that contradict the physical consideration from above yield slightly (refinement levels 2 and 4) to significantly (refinement level 5) lower profits.

To confirm the dependence of the ℓ^2 based algorithm on the non-uniformity in the mesh, a series of experiments using a series of uniformly refined meshes were performed as a reference (not displayed). For these the iteration number to reach convergence is observed to not increase using the ℓ^2 (as well as L^2) based algorithm with refinement. Analogously, the final optimal friction field (and hence profit functional value) does not change significantly.

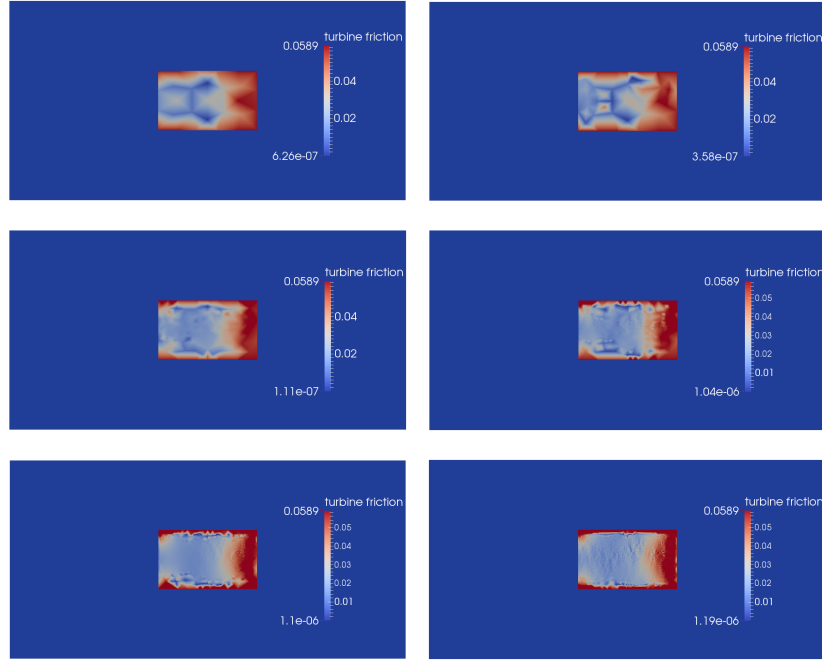


Fig. 3.8: Results of turbine density optimisation using the L^2 representation of the functional derivative for successively finer meshes corresponding to Fig. 3.4 (random refinement)

A Python script that performs the tidal turbine array optimisation with the configuration from this chapter can be found in [16]. In the optimisation step, the bounded LMVM method in the TAO library [14] is used.

3.6 Tidal resources assessment in the Pentland Firth, Scotland

In this example we apply the turbine layout optimisation to a realistic domain: the Pentland Firth between mainland Scotland and the Orkney Islands. The Pentland Firth is considered highly suitable for tidal turbine deployments [7], due to its fast tidal streams of up to 5 m/s speed in certain locations.

The aim of our experiment is to identify suitable deployment areas for tidal turbine farms. More specifically, the idea is to use the layout optimisation method as described in the previous sections, but apply it to all suitable areas across the Pentland Firth where turbines could be deployed. Since current tidal turbine technology can typically only be installed in waters with a certain depth range, we define this permissible area here as:

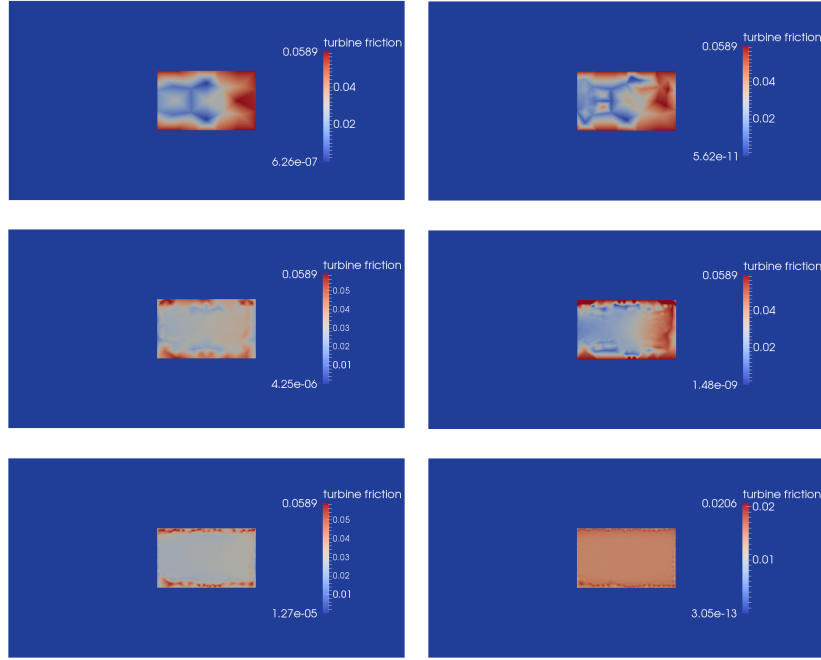


Fig. 3.9: Results of turbine density optimisation using the ℓ^2 representation of the functional derivative for successively finer meshes corresponding to Fig. 3.4 (random refinement)

$$\Omega_{\text{farm}} = \{x \in \Omega \text{ with } 25 \text{ m} \leq h(x) \leq 60 \text{ m}\}. \quad (3.41)$$

Figure 3.10a shows the domain of interest, and the sub-domain of potential turbine installations. The discretised domain is shown in Fig. 3.10b. The mesh element sizes are approximately 15 m in the proximity of the coast and vary up to 500 m in the deeper coastal ocean.

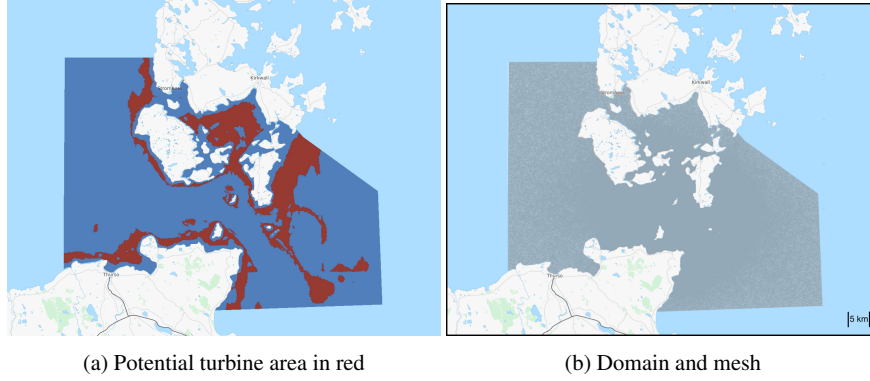


Fig. 3.10: The domain, mesh and potential turbine areas (Ω_{farm}) of the Pentland Firth example. The Pentland Firth is located in Scotland, UK and is of high interest for tidal turbine array development

The formulation of the farm optimisation problem (3.1) was extended to capture the reversing flood-ebb flow of a tidal cycle by replacing the single shallow water constraint by two subsequent, steady-state shallow water solves. Each solve corresponds to one peak (flood and ebb) flow within one tidal cycle. For our experiment, the specific date stamps were 13:55 on the 18 September 2001, yielding a east-west flow, and 20:10 on the 18th September 2001, yielding a west-east flow in the Pentland Firth.

The objective functional is defined as the sum of the power productions (using (3.10)) at the two peak flows and a regularisation term:

$$J_P(u_1, d) + J_P(u_2, d) + C \int_{\Omega_{\text{farm}}} \|\nabla d(x)\|^2 dx,$$

where $u_i, i = 1, 2$ are the horizontal velocities from the two shallow water solves, d is the farm turbine density, and $C > 0$ is the regularisation coefficient. The H_0^1 regularisation term helps to enforce smoothness in the turbine density function, and is used to avoid infeasible “checkerboard” configurations. A list of the model parameters is shown in table 3.5. Overall, the parameters were chosen more as a demonstration setup for the numerical technology rather than to provide physically realistic predictions. In particular, the viscosity needed to be chosen artificially high in order for the steady-state solvers to converge. A more accurate model would therefore use a time-dependent shallow water solver.

The problem was solved with the bounded LMVM method in the TAO library [14]. To study the mesh dependency of this problem, the solver was executed both with the ℓ^2 and the L^2 inner products for the control variable. The convergence plots for both optimisations are plotted in Fig. 3.11. In both cases, the optimisation

Table 3.5: Parameters for the shallow water equations and the tidal turbine farm used for the continuous turbine density optimisation

Parameter	Symbol	Value	Units
Water density	ρ	1000	kg/m ³
Viscosity	ν	10,000	m ² /s
Water depth	h	[4]	m
Gravity	g	9.81	m/s ²
Natural bottom friction	c_b	0.0025	
Maximum turbine friction	$c_t(1)$	10.0	
Regularisation coefficient	C	10^4	

algorithm terminates in less than 25 iterations. The optimisation with the $L^2(\Omega)$ inner product can be seen to reach an optimum with a higher functional value.

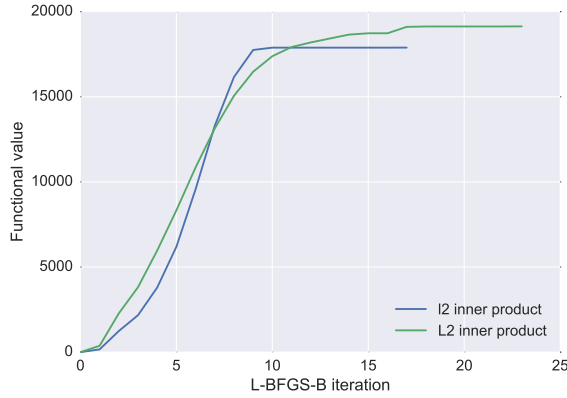


Fig. 3.11: The optimisation with the ℓ^2 inner product terminates after 17 iterations with a functional value of 17,889. The optimisation with the L^2 inner product terminates after 23 iterations with a functional value of 19,136.

The optimal turbine configuration obtained for both the ℓ^2 and the L^2 based algorithms is shown in Fig. 3.12. The difference in the two solutions is clearly visible. In particular, in the south-east region, the configuration with the L^2 inner product (3.12a) deploys significantly more turbines compared to the solution with the ℓ^2 inner product (3.12b). Figures 3.12c and 3.12d provide a close-up view to the area east of Stroma island. In these figures, the ℓ^2 -based solution shows the mesh dependent structure, while the L^2 -based solution is smoother and shows no mesh dependency.

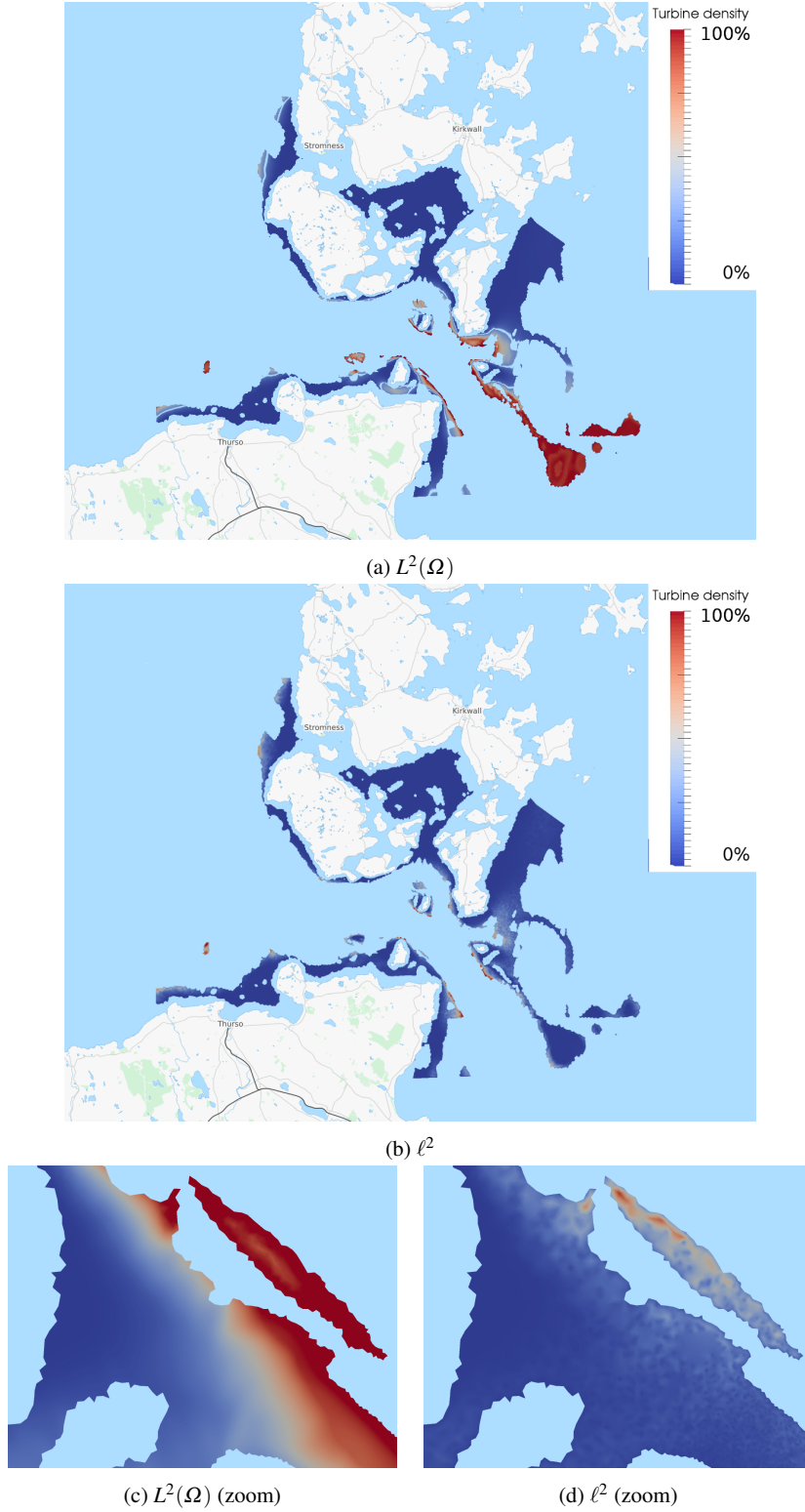


Fig. 3.12: The optimised turbine densities for the Pentland Firth example, computed with the L^2 inner product (figure a) and the ℓ^2 inner product (figure b). Figures c and d show a zoom of the region to the east of Stroma Island.

3.7 References

- [1] Mohammad Amin Abolghasemi et al. “Simulating tidal turbines with multi-scale mesh optimisation techniques”. In: *Journal of Fluids and Structures* 66 (2016), pp. 69–90. ISSN: 0889-9746. DOI: <http://dx.doi.org/10.1016/j.jfluidstructs.2016.07.007>. URL: <http://www.sciencedirect.com/science/article/pii/S0889974615301729>.
- [2] S. C. Brenner and L. R. Scott. *The Mathematical Theory of Finite Element Methods*. 3rd Edition, Springer Science+Business Media, LLC, 2008.
- [3] Didier Bresch. “Shallow-Water Equations and Related Topics”. In: *Handbook of Differential Equations: Evolutionary Equations*. Ed. by C.M. Dafermos and Milan Pokorný. Vol. 5. Handbook of Differential Equations: Evolutionary Equations. North-Holland, 2009, pp. 1–104. DOI: [http://dx.doi.org/10.1016/S1874-5717\(08\)00208-9](http://dx.doi.org/10.1016/S1874-5717(08)00208-9). URL: <http://www.sciencedirect.com/science/article/pii/S1874571708002089>.
- [4] British Oceanographic Data Centre (BODC). *GEBCO_08 bathymetry grid, version 20100927*. <http://www.gebco.net>. 2010.
- [5] D Culley. *A hierarchy of approaches for the optimal design of tidal turbine farms*. http://www.icoe2014canada.org/wp-content/uploads/2014/11/CulleyDavid_5-4.pdf. 5th International Conference on Ocean Energy. 2014.
- [6] Scott Draper et al. “An electrical analogy for the Pentland Firth tidal stream power resource”. In: *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 470.2161 (2013). ISSN: 1364-5021. DOI: [10.1098/rspa.2013.0207](https://doi.org/10.1098/rspa.2013.0207).
- [7] Crown Estate. “Wave and tidal energy in the Pentland Firth and Orkney waters: How the projects could be built”. In: *The Crown Estate, London* (2011).
- [8] P. E. Farrell et al. “Automated derivation of the adjoint of high-level transient finite element programs”. In: *SIAM Journal on Scientific Computing* (2012). accepted.
- [9] P. E. Farrell Funke S. W. and M. D. Piggott. “Tidal turbine array optimisation using the adjoint approach”. In: *Renewable Energy* 63 (2014), pp. 658–673. DOI: [10.1016/j.renene.2013.09.031](https://doi.org/10.1016/j.renene.2013.09.031).
- [10] Simon W Funke. “The Automation of PDE-constrained Optimisation and its Applications”. PhD thesis. Imperial College London, UK, 2012.
- [11] S.W. Funke, S.C. Kramer, and M.D. Piggott. “Design optimisation and resource assessment for tidal-stream renewable energy farms using a new continuous turbine approach”. In: *Renewable Energy* 99 (2016), pp. 1046–1061. ISSN: 0960-1481. DOI: <http://dx.doi.org/10.1016/j.renene.2016.07.039>. URL: <http://www.sciencedirect.com/science/article/pii/S0960148116306358>.
- [12] Stephan C. Kramer and Matthew D. Piggott. “A correction to the enhanced bottom drag parameterisation of tidal turbines”. In: *Renewable Energy* 92

- (2016), pp. 385–396. ISSN: 0960-1481. DOI: 10.1016/j.renene.2016.02.022.
- [13] Anders Logg, Kent-Andre Mardal, and Garth Wells. *Automated solution of differential equations by the finite element method: The FEniCS book*. Vol. 84. Springer Science & Business Media, 2012.
 - [14] Todd Munson et al. *TAO 2.1 Users Manual*. Tech. rep. ANL/MCS-TM-322. USA: Argonne National Laboratory, 2012.
 - [15] Brian L. Polagye and Philip C. Malte. “Far-field dynamics of tidal energy extraction in channel networks”. In: *Renewable Energy* 36.1 (2011), pp. 222–234. DOI: <http://dx.doi.org/10.1016/j.renene.2010.06.025>.
 - [16] Tobias Schwedes, David Ham, and Simon Wolfgang Funke. *Code for simulations in Chapter 3 of ResearchBrief*. Dec. 2016. DOI: 10.5281/zenodo.224251. URL: <https://doi.org/10.5281/zenodo.224251>.
 - [17] G. Sutherland, M. Foreman, and C. Garrett. “Tidal current energy assessment for Johnstone Strait, Vancouver Island”. In: *Proceedings of the Institution of Mechanical Engineers A* 221. 2007, pp. 147–157.
 - [18] R. Vennell. “Tuning turbines in a tidal channel”. In: *Journal of Fluid Mechanics* 663 (2010), pp. 253–267. DOI: 10.1017/S0022112010003502.
 - [19] Ross Vennell. “Estimating the power potential of tidal currents and the impact of power extraction on flow speeds”. In: *Renewable Energy* 36.12 (2011), pp. 3558–3565. ISSN: 0960-1481. DOI: 10.1016/j.renene.2011.05.011.
 - [20] Ross Vennell et al. “Designing large arrays of tidal turbines: A synthesis and review”. In: *Renewable and Sustainable Energy Reviews* 41.0 (2015), pp. 454–472. DOI: 10.1016/j.rser.2014.08.022.
 - [21] C. B Vreugdenhil. *Numerical Methods for Shallow-Water Flow*. Springer, 1994.
 - [22] J. Young. *Optizelle: An open source software library designed to solve general purpose nonlinear optimization problems*. www.optimojoe.com, Open source software. 2016.

Index

- adjoint equation, 33
- basis function
 - global, 13
 - local, 13
- BFGS algorithm
 - in $\mathbb{R}^n$, 43
 - in Hilbert space, 44
- boundary condition
 - Dirichlet, 4
 - Neumann, 4
- boundary value problem, 4
- Burgers' equation, 14
- Cauchy stress tensor, 86
- Cauchy-Schwarz inequality, 22
- central-difference formula, 33
- complex-step formula, 33
- condition number, 60
- continuity equation, 86
- directional derivative, 23
- dual space, 21
- Euler discretisation, backward, 16
- exact line search, 60
- finite element method, 3
- finite elements
 - P_1 , 10
 - P_2 , 10
- forward model, 29
- Fréchet derivative, 23
- Gâteaux derivative, 18
- Galerkin
 - finite element approximation, 9
 - system, 9
- Gauss' theorem, 86
- H^1 , 6
- H^2 , 6
- Hilbert space, 21
- hydrostatic balance, 88
- incompressibility, 87
- index
 - global, 12
 - local, 12
- interior point method, 47, 68
- Kantorovich's lemma, 60
- L^1_{loc} , 7
- L^2 , 5
 - inner product, 5
 - norm, 5
- Laplace operator, 4
- Limited Memory, Variable Metric (LMVM)
 - method, 68
- Line Search Newton-CG algorithm, 40
- load vector, 9
- mass matrix, 27, 58
 - local, 61
- mesh refinement
 - non-uniform, 66, 68
 - random, 70
 - uniform, 66
- momentum equation, 87
- Moré-Thuente line search, 68
- nabla operator, 5
- Navier-Stokes equations, 88

- Newton's method, 18
- Poisson equation, 3
- power functional, 85
- profit functional, 85
- reduced functional, 29
- reduced optimization problem, 29
- reference element, 13
- residual, 17
 - form, 17
- Riesz
 - representation theorem, 21
 - representer, 22
- scatter matrix, 12
- secant equation, 42
- shallow water equations, 82, 88
- shape function, 8
- singular value decomposition, 61
- steepest descent algorithm, 39
- stiffness matrix, 9, 64
 - global, 11
 - local, 12
- Stokes' stress constitutive equation, 87
- tangent linear model, 32
- Taylor's theorem, 24
- Taylor-Hood mixed finite elements P_2 - P_1 , 92
- test function, 2, 4, 24
- tidal turbine array, 79
- transformation
 - affine, 61
 - theorem, 62
- trial function, 24
- triangulation, 10
- turbine
 - drag force, 84
 - friction function, 84
- vector Laplacian, 15
- weak
 - derivative, 7
 - formulation, 7
 - solution, 5