Shape optimization in incompressible Navier-Stokes flows

Oliver Behm

Born 8th January 1988 in Bonn

20th February 2014

Master's Thesis Mathematics

Advisor: Prof. Dr. Michael Griebel
Institute for Numerical Simulation

Mathematisch-Naturwissenschaftliche Fakultät der
Rheinischen Friedrich-Wilhelms-Universität Bonn

Contents

Introduction

Motivation

In many engineering applications a defining aspect of the design process is iterating upon proposed solutions until they satisfy the requirements of the problem. Traditionally this is done through prototyping and experimentation, which is both time consuming and expensive. As a classical example, consider the construction of an airfoil. Barring computation, an engineer has to make educated guesses as to the airfoils shape. Then he will order the construction of a, probably scaled, prototype and analyze it in a costly wind tunnel to gather information about the relevant properties (i.e. drag, lift, etc.) which can be taken into account when preparing the next draft.

Today, advances in computing hardware and simulation techniques have made it possible to replace physical experiments by virtual prototyping. In the context of the airfoil example this could mean the design is entered into a simulation software that allows computation of the flow around the airfoil and the derived quantities of interest without having to conduct any physical experiments.

The next step is to automate the design process itself, i.e. let a computer make the modifications and evaluate the results with minimal supervision. Ideally it would only be necessary to describe the desired properties and construction dependent constraints to automatically search for a design that satisfies both, based on a model that describes the application. Shape optimization techniques are used for the automation of problems that can be framed in terms of varying an object geometry that can be appraised in a quantifiable sense, i.e. the wing profile with the least drag or the column section that bears the most load.

Shape Optimization for Incompressible Fluids

The problem considered in this thesis are geometries immersed in incompressible and transient fluid flows, the applications of which include the design of structures such as bridge piers and mooring areas in rivers, the submerged parts of deep sea structures, water pipes or ship compo-

nents. Those geometries are optimized in regard to their effect on the surrounding fluid, the aim being to minimize the dissipation of energy into heat caused by disturbances in the flow.

An essential question in tackling these problems is the way in which the fluid flow model is taken into consideration while searching for an optimal geometry. Modern approaches to shape optimization from the mathematical literature mostly propose an explicit treatment using analytical methods, but this has to be done on a case by case basis depending on the requirements and additional constraints. While this is fine when considering a specific example, in the context of an actual application that means it is impossible to add e.g. additional equations describing the heat transfer in the fluid without re-deriving the optimization procedure as well.

Black-Box Solution

The approach that is followed here is a black-box treatment of the application model, making no assumptions beyond the fact that there is an opaque implementation available that maps trial geometries to the desired measurements. After parameterizing the geometry through a finite number of parameters, the problem is then pliable to classical optimization methods as long as they are limited to using function evaluations. While this can negatively impact the performance of the optimization, it allows to add to the model in the way described above without touching the other parts of the setup, even making it possible to switch the simulation software with minimal effort.

The resulting optimization problem is typically of a high dimension, making it an imperative to take special care when choosing a method for solving it. Particularly considering the restrictions mentioned above. For this reason a sparse-grid based approach from Novak and Ritter (1996) was adapted for the constraints arising during the parameterization and applied to the shape optimization problem.

Contributions

The contributions of this thesis are the following:

- It gives an overview over current methods for approaching shape optimization problems and justifies the one chosen herein.
- A parallelized hyperbolic cross point method was implemented, adapted to linear constrained problems using a projection-penalty approach and supplemented by a subsequent local search. It was tested using several benchmark problems and compared with various non-adaptive methods and the original reference implementation.

- A mesh generator using the distmesh algorithm was implemented and combined with a pre-existing fast marching method to allow the automatic triangulation of flow geometries.
- A FEniCS based finite element discretization of the Navier-Stokes equations was used in conjunction with the automatic mesh generation to solve the parameterized flow problems.
- Together with a B-Spline parameterization the above components were used to set up a black-box shape optimization solver.
- The approach was used for several experiments modeled after practical applications.

Overview

In **chapter 1** we will give an overview over the different approaches to shape optimization problems with PDE constraints from the literature. This will serve to contextualize and justify the approach pursued in this thesis.

Chapters 2-5 explain the theoretical background and implementation of all parts involved in the setup of the solver. This starts with a discussion of the Navier-Stokes equation, the energy dissipation and the simulation software in **chapter 2**. **Chapter 3** explains the automatic mesh generation implemented for use with the Navier-Stokes solver, **chapter 4** the parameterization of the geometries that are to be optimized. In **chapter 5** we give a short introduction to the global optimization problem and elucidate the methods used here.

Numerical experiments for each separate part of the solver and the shape optimization problem as a whole are discussed in **chapter 6**, the thesis then concludes with a summary and suggestions for future work in **chapter 7**.

1 Shape Optimization

Shape optimization as a whole stands at the intersection of the disciplines of optimization, computational geometry and partial differential equations. For this reason, every effort towards solving a shape optimization problem of any kind must decide on solution strategies for the components of the problem concerning each of these disciplines. Those decisions are all interrelated and this chapter aims to give an overview over the different approaches to solving shape optimization problems, as well as their respective advantages and drawbacks. Special emphasis will be put on the black-box approach chosen in this thesis and its justification. While there seems to be a lack of general survey articles in this area, a good overview focused on flow problems can be obtained from the books by Mohammadi and Pironneau (2001) or Thévenin and Janiga (2008).

1.1 Overview

For the purposes of this discussion we will consider a general shape optimization problem of the form

$$\text{minimize } F(\omega, u) \text{ over all } (\omega, u) \text{ s.t. } c(\omega, u) = 0. \tag{1.1}$$

Here ω is the controlled geometry that is to be optimized, u is the *state variable* given by the solution of the partial differential equation $c(\omega, u) = 0$, also called the *state equation*, and the *objective function* $F(\omega, u)$ is the measurement of interest. The main parting point in all approaches towards solving (1.1) is whether the function and constraint are taken as a simple procedure mapping geometries to measurements or if one tries to extract further information, which generally means computing sensitivities of the objective with respect to the controlled variables. For ease of presentation we will rewrite the problem (1.1) into a minimization problem without explicit constraints. Consider an operator G, that maps each geometry ω to the solution u of the PDE constraint $c(\omega, u) = 0$, assuming there is such a unique solution. Then (1.1) is essentially the same as minimizing the *reduced functional*

$$J(\omega) := F(\omega, G(\omega)). \tag{1.2}$$

Shape Calculus

The prevalent way to extract sensitivity information from the fully continuous shape optimization problem is through the use of *shape calculus*, as is described in detail in Schmidt and Schulz (2010) for problems involving the stationary Navier-Stokes equation. The principal idea is to consider *shape derivatives* of the objective function: If we consider a parameter dependent deformation $\omega_t = \{x_0 + tv(x_0) : x_0 \in \omega\}$ of ω along an appropriately smooth vector field v, the derivative of J at ω applied to v is defined by

$$dJ(\omega)[v] = \lim_{t \searrow 0} \frac{J(\omega_t) - J(\omega)}{t}. \tag{1.3}$$

The fundamental result used for implementing gradient descent algorithms using shape derivatives is the *Hadamard formula*. The volume integral form of this formula says that for objective functions of the form

$$J(\omega) = \int_\omega f \, \mathrm{d}x, \tag{1.4}$$

without PDE constraint, the shape derivative is given by

$$dJ(\omega)[v] = \int_{\partial\omega} (v \cdot n) f \, \mathrm{d}\sigma(x). \tag{1.5}$$

The nature of this gradient as a normal product, allows to apply it directly to the set to be optimized as a descent direction. When PDE constraints are added though, the derivative will incorporate a linearized adjoint form of the PDE. These have to be derived from the model equation in a non-trivial fashion on a case by case basis. Accordingly there is need for software solving the adjoint problem and that software, too, has to be updated every time the model is changed. In addition, the adjoint linearized Navier-Stokes equation is a final value problem that evolves backwards in time and involves the solution to the full Navier-Stokes equation at the linearization point as data (c.f. Hinze and Kunisch (2001), which is concerned with parameter based optimal control of the Navier-Stokes equation). In particular this means that the whole solution of the Navier-Stokes equation for the geometry in question has to be held in memory at once, which leads to huge space requirements especially for explicit methods.

Sensitivites from the (semi-)Discretized Problem

Another possibility is to first discretize some or all parts of the optimization problem and only then calculate derivatives. Consider a discretization of $\omega(p)$ of the shape depending on a finite

set of parameters $p^T = (p_1, \ldots, p_n)$ and a corresponding reduced functional

$$J(p) = F(p, G(p)), \tag{1.6}$$

where we have absorbed the extra step over ω into F and G for the moment.

It is, in principle, possible to calculate partial derivatives for J in the usual way. Taking the derivative by p_i and applying the chain rule leads to

$$\frac{\mathrm{d}}{\mathrm{d}p_i} J(p) = F_{p_i}(p, G(p)) + F_u(p, G(p)) \cdot G_{p_i}(p). \tag{1.7}$$

Regardless if G is the continuous solution operator or the state equation has been discretized, the dependence of the solution on the geometry still has to be taken into account when trying to calculate analytical derivatives.

There are multiple approaches to approximate sensitivities without explicitly treating the geometry dependence. One is to simply use finite difference based gradient approximations

$$\nabla J(p)^T = \left(\frac{J(p + \epsilon e_1) - J(p)}{\epsilon}, \ldots, \frac{J(p + \epsilon e_n) - J(p)}{\epsilon} \right). \tag{1.8}$$

In many cases these approximations can be accurate enough for use even in quasi-Newton methods, but especially for simulation based optimization they also have drawbacks. Since the function whose gradient is to be approximated is itself the result of a discretization, it doesn't necessarily resemble the actual function at the scales necessary for good finite difference approximations. Furthermore, adaptivity and parallelism may add a some amount of (non-deterministic) noise. The feasibility of finite difference based gradients for flow control problems was analyzed by Burkardt et al. (2002) in the context of a two-dimensional channel-flow model problem.

Another possibility is *automatic differentiation*. The premise of this method is that every actual implementation of a simulation is composed of very basic operations such as $+, -, \times, \div$, elementary functions and programming directives such as loops and if-clauses. In principle this enables one to differentiate the function resulting from the implementation by stepping through the source code, differentiating each of those operations and applying the chain- and product-rules. Using this it is not only possible to take partial derivatives as above, but the approach can also be used to compute the adjoint equation mentioned in the previous paragraph. Anderson et al. (2001) compared all three approaches in an aerodynamics application. By design this method needs access to all source code used in the simulation of the state constraint which precludes the use of commercial software and closed source libraries. Problems also arise when combining software in multiple different programming languages, which is a common feature in large scale simulations that require the combination of several distinct programs.

1.2 Parameterized Black-Box Shape Optimization

The approach pursued by this thesis is to parameterize the geometry and interpret the PDE constraint as a fully opaque process mapping geometries to solutions, hence the name *black-box*. Therein the PDE constraint is treated as part of the functional, similar to the definition of the reduced functional above, and no attempt is made to extract any information exceeding simple function evaluations. After parameterizing the geometry space the problem is then pliable to classical optimization techniques insofar as they are restricted to using function evaluations.

We will explain the particulars of the procedure in terms of the problem considered in this thesis: Recall that the aim is to optimize a geometry ω in a fluid flow u described by the Navier-Stokes equations $c(u, \omega) = 0$. The objective function used to assess the geometry is the energy dissipation into heat $F(\omega, u)$

- First, the suitable geometries are parameterized using cubic splines. Expanding those splines into a basis expansion gives the parameters $c_1, \ldots, c_n$ describing the possible geometries and thus a map
$$c_1, \ldots, c_n \mapsto \omega(c_1, \ldots, c_n). \tag{1.9}$$
Additional constraints on ω are reformulated as linear constraints on the parameters $c_1, \ldots, c_n$.
- Next we choose a simulation software NaSt, that approximates the solution u to the flow constraint $c(u, \omega) = 0$. Combining this with the geometry parameterization gives a map
$$c_1, \ldots, c_n \mapsto \mathrm{NaSt}(\omega(c_1, \ldots, c_n)) \tag{1.10}$$
that assigns each set of geometry parameters an approximation to the flow generated by this geometry.
- Computing the energy dissipation of this approximation defines the black-box objective function
$$c_1, \ldots, c_n \mapsto F(\mathrm{NaSt}(\omega(c_1, \ldots, c_n))), \tag{1.11}$$
composed of the subsequent steps of geometry generation, simulation and post processing.
- This objective is minimized using a modified hyperbolic cross point method, adapted to handle the linear constraints arising from the parameterization.

Using this approach is is possible to exchange any of the components above with minimal effort, i.e. exchange the simulation software for a different one, try different objectives or extend the model itself.

The optimization method is to be chosen with some care. The problem resulting from this approach is characterized by the fact that function evaluations are extremely expensive, each requiring a full Navier-Stokes simulation of the problem. Further, the number of unknowns in the optimization is dictated by the number of unknowns in the parameterization, which results in a high dimensional problem for any non-trivial geometries.

As mentioned before the that fact the objective function is the result of a discretization itself has to be taken into account. If the state equation is discretized with a mesh resolution h it is to be assumed that variations in the geometry of scales smaller than h are not reflected accurately in the objective function, making sub h step lengths in local searches unfeasible.

2 Navier-Stokes Equation

2.1 Overview

The physics of transient incompressible fluid flow in d dimensions are, from an Eulerian point of view, described by a system of $d+1$ partial differential equations known as the *incompressible Navier-Stokes equations*. If $\Omega \subset \mathbb{R}^d$ is the domain of flow and $[0,T]$ the time interval of interest they are given by

$$u_t + (u \cdot \nabla)u = \nu \Delta u - \nabla p + f \text{ in } \Omega \times [0,T], \tag{2.1}$$

$$\operatorname{div} u = 0 \qquad \text{in } \Omega \times [0,T]. \tag{2.2}$$

The functions $u : \Omega \times [0,T] \to \mathbb{R}^d$ and $p : \Omega \times [0,T] \to \mathbb{R}$ are the velocity and pressure of the fluid at each point and time. The parameter ν, called *kinematic viscosity*, is specific to the fluid in question and describes its behavior under the assumption of constant density. Equation (2.1) is called *momentum equation* and has the form of a nonlinear transport equation, where $\nu \Delta u$ describes the viscous forces exerted by the fluid, $f : \Omega \times [0,T] \to \mathbb{R}$ are known body forces (for instance gravity) and ∇p are the pressure gradient forces caused by the *incompressibility condition* (2.2). The pressure p can in that sense be interpreted as a Lagrange multiplier enforcing equation (2.2).

The Laplace operator in equation (2.1) is understood to be applied component wise and thus (2.1) are, in fact, three equations. The convective term is read as

$$(u \cdot \nabla)u = \left(\sum_{j=1}^{d} u_j \partial_j \right) u = \left(\sum_{j=1}^{d} u_j (\partial_j u_i) \right)_{i=1,\dots,d}. \tag{2.3}$$

To constitute a well posed problem, the equations (2.1) and (2.2) have to be complemented by initial values $v_0 : \Omega \to \mathbb{R}^d$ and $p_0 : \Omega \to \mathbb{R}$, the velocity and pressure at time $t = 0$. The velocity also needs to be prescribed at the boundary for all times. Common boundary conditions include:

- *Dirichlet boundary conditions* for in- or outflow

$$u(x, y) = g(x, y). \tag{2.4}$$

- *No-slip condition*

$$u(x, y) = 0, \tag{2.5}$$

  which is special case of a Dirichlet condition and commonly assumed to hold at solid walls in flow regimes where the cohesive forces of the fluid do not dominate its behavior.

- *Slip condition*

$$\partial_n u(x, y) = 0, \tag{2.6}$$

  which is a homogeneous Neumann boundary condition.

- *Do-nothing condition*

$$\nu \partial_n u - p\nu = 0, \tag{2.7}$$

  which is the natural boundary condition derived from the variational form of the Navier-Stokes equation.

In all cases (x, y) is a point on the boundary $\partial\Omega$, n is the outer normal to $\partial\Omega$ and $\partial_n u = \nabla u \cdot n$ is the normal derivative of u.

Weak Solutions

In anticipation of the discussion of finite element methods in section 2.4.2, this paragraph gives a short motivation of the notion of *weak solvability* of the Navier-Stokes equation.

Consider a classical solution (u, p) to the Navier-Stokes equation on $\Omega \times [0, T]$, i.e. u and p are at least of the class C^2 and C^1 respectively and satisfy the equations (2.1) and (2.2) at every point $(x, t) \in \Omega \times [0, T]$ and assume that $\partial\Omega$ is sufficiently smooth. Multiply the momentum equations (2.1) by a smooth, compactly supported function $v : \Omega \times [0, T] \to \mathbb{R}^d$ in the sense of a scalar product and integrate over $\Omega \times [0, T]$. This leads to the relation

$$\int_0^T \int_\Omega u_t \cdot v + (u \cdot \nabla)u \cdot v - \nu \Delta u \cdot v + \nabla p \cdot v \, \mathrm{d}x\mathrm{d}t = \int_0^T \int_\Omega v \cdot f \, \mathrm{d}x\mathrm{d}t. \tag{2.8}$$

Applying Greens identity to the terms involving the Laplacian and the pressure as well as using the fact that v is zero on the boundary leads to the equality

$$\int_0^T \int_\Omega u_t \cdot v + (u \cdot \nabla)u \cdot v + \nu \nabla u : \nabla v + p \operatorname{div} v \, \mathrm{d}x\mathrm{d}t = \int_0^T \int_\Omega v \cdot f \, \mathrm{d}x\mathrm{d}t, \tag{2.9}$$

where $A : B = \sum_{i,j} A_{ij} B_{ij}$ denotes the Frobenius inner product of matrices. Taking a smooth and compactly supported function $q : \Omega \times [0,T] \to \mathbb{R}$ one can do the same to the incompressibility condition (2.2), which gives

$$\int_\Omega \operatorname{div} u \, q \, \mathrm{d}x = 0. \tag{2.10}$$

In summary: Every classical solution to the Navier-Stokes equation satisfies the variational equations (2.9) and (2.10) for all smooth and compactly supported functions v and q. On the other hand, a sufficiently smooth pair of functions u, p satisfying equations (2.9) and (2.10) for all v, q solves the classical equations by the fundamental lemma of the calculus of variations. This motivates the following definition:

Definition 2.1 (Weak Solutions)**.** *Let the function space V be defined as follows:*

$$V = \{v \in \mathrm{L}^2([0,T]; H_0^1(\Omega)^d) : \partial_t v \in \mathrm{L}^2([0,T]; H_0^1(\Omega)^d)\}. \tag{2.11}$$

Functions $u \in V$ and $p \in L^2([0,T], L^2(\Omega))$ are said to weakly solve the Navier-Stokes equation, if

$$\begin{aligned} \int_0^T \int_\Omega u_t \cdot v + (u \cdot \nabla) u \cdot v + \nu \nabla u : \nabla v + p \operatorname{div} v \, \mathrm{d}x \mathrm{d}t &= \int_0^T \int_\Omega v \cdot f \, \mathrm{d}x \mathrm{d}t \\ \int_0^T \int_\Omega \operatorname{div} u \, q \, \mathrm{d}x \mathrm{d}t &= 0 \end{aligned} \tag{2.12}$$

for all $v \in V$ and $q \in L^2([0,T], L^2(\Omega))$.

Here $\mathrm{H}_0^1(\Omega)^d$ denotes the *Sobolev space* of square integrable and one-time weakly differentiable functions from Ω to $\mathbb{R}^d$ with zero-trace, for details see Adams (1975) or Alt (2002). Note that the variational form of the Navier-Stokes equation is more commonly derived by taking the space V to only include functions with vanishing spatial divergence, eliminating the incompressibility condition as an equation as well as the pressure from the problem. Since the construction of finite-element spaces is more complicated in that case we have stuck to the saddle point form above. In two space dimensions the Navier-Stokes equation is known to have a unique weak solution–c.f. Temam (1977).

2.2 Energy Dissipation

The central quantity that makes up the objective functions used in the numerical experiments is the *energy dissipation*. For a sufficiently smooth velocity field $v : \Omega \times [0,T] \to \mathbb{R}^d$ it is defined by the H^1 semi-norm

$$D(v,t) = \int_\Omega |\nabla v(x,t)|^2 \, \mathrm{d}x = |v(\cdot,t)|^2_{H^1(\Omega.\mathbb{R}^d)} \tag{2.13}$$

at every time t. The choice of this functional is motivated by its close relation to the *kinetic energy* of the flow

$$E_{\text{kin}}(v,t) = \int_\Omega |v(t)| \,\mathrm{d}x, \tag{2.14}$$

as shall be shown in the rest of the paragraph. Let $v \in C^1([0,T], C^2(\Omega, \mathbb{R}^d))$ satisfy the homogeneous Navier-Stokes equation with Dirichlet zero boundary conditions on a bounded domain with sufficiently smooth border $\partial\Omega$:

$$v_t + (v \cdot \nabla)v - \nu\Delta v + \nabla p = 0 \text{ in } \Omega, \tag{2.15}$$

$$\operatorname{div} v = 0 \text{ in } \Omega, \tag{2.16}$$

$$v|_{\partial\Omega} = 0 \text{ on } \Omega. \tag{2.17}$$

Multiplying the momentum equation (2.15) by v and integrating over Ω, we find that

$$\begin{aligned} \frac{1}{2}\frac{\mathrm{d}}{\mathrm{d}t}\int_\Omega |v|^2 \,\mathrm{d}x &= \int_\Omega v_t \cdot v \,\mathrm{d}x \\ &= \nu\int_\Omega \Delta v \cdot v \,\mathrm{d}x - \int_\Omega \nabla p \cdot v \,\mathrm{d}x - \int_\Omega (v\cdot\nabla)v \cdot v \,\mathrm{d}x. \end{aligned} \tag{2.18}$$

Integrating by parts and using the fact that $v|_{\partial\Omega} \equiv 0$ allows the resulting boundary terms to be disregarded, the three summands of the last term can be computed as

$$\int_\Omega \nabla p \cdot v \,\mathrm{d}x = -\int_\Omega p \operatorname{div} v \,\mathrm{d}x = 0, \tag{2.19}$$

$$\nu\int_\Omega \Delta v \cdot v \,\mathrm{d}x = -\nu\int_\Omega |\nabla v| \,\mathrm{d}x, \tag{2.20}$$

and

$$\begin{aligned} \int_\Omega (v\cdot\nabla)v \cdot v \,\mathrm{d}x &= \sum_{k=1}^d \sum_{l=1}^d \int_\Omega v_k(\partial_k v_l)v_l \,\mathrm{d}x \\ &= \sum_{k=1}^d \sum_{l=1}^d \frac{1}{2}\int_\Omega v_k \partial_k v_l^2 \,\mathrm{d}x \\ &= \sum_{k=1}^d \sum_{l=1}^d -\frac{1}{2}\int_\Omega (\partial_k v_k) v_l^2 \,\mathrm{d}x \\ &= \sum_{l=1}^d -\frac{1}{2}\int_\Omega v_l^2 \operatorname{div} v \,\mathrm{d}x \\ &= 0. \end{aligned} \tag{2.21}$$

Combining these shows that

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_\Omega |v|^2 \,\mathrm{d}x = -2\nu \int_\Omega |\nabla v| \,\mathrm{d}x$$

or

$$\frac{\mathrm{d}}{\mathrm{d}t} E_{\mathrm{kin}}(v,t) = -2\nu D(v,t). \tag{2.22}$$

This means that the rate of change of the kinetic energy of the flow is inversely proportional to the energy dissipation, i.e. a small dissipation means a small loss in kinetic energy over time. Minimizing, for example, the average dissipation

$$⨍_0^T \int_\Omega |\nabla v| \mathrm{d}x\mathrm{d}t = \frac{1}{T}\int_0^T \int_\Omega |\nabla v| \mathrm{d}x\mathrm{d}t \tag{2.23}$$

is thus taken to be morally equivalent to minimizing the average loss in kinetic energy over time.

In cases where the boundary conditions are more complicated the partial integrations above generate multiple boundary terms. Since the common (no-)slip conditions are nothing else than Dirichlet or Neumann zero conditions, what is left over are the regions on the boundary where fluid enters or leaves the domain and a energy exchange takes place. In this respect the actual energy dissipation describes the losses inside of the domain.

2.3 Continuous Shape Optimization Problem

In this section, the obstacle flow also used as a numerical test case in chapter 6 is considered in a continuous setting. This is done to illustrate some of the challenges that arise when posing optimization problems that aim to modify the geometry underlying a partial differential equation.

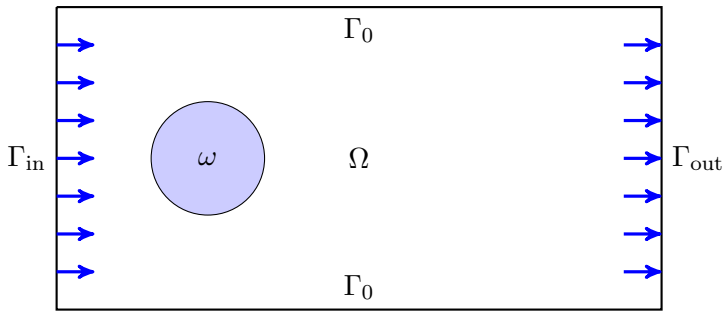

Figure 2.1: Sketch of the problem discussed in section 2.3. Fluid flows through the domain Ω from left to right, no-slip conditions hold on the boundaries Γ_0 and the surface of the obstacle $\partial\omega$.

The setting is that of an obstacle in a channel with an incompressible fluid flow from one end to the other as well as walls on either side of the obstacle. This is illustrated in figure 2.1: A fluid flows enters the domain with velocity u_{in} from the left side Γ_{in} and travels to the right, where it leaves the domain through the border marked Γ_{out}. A do-nothing boundary condition holds on that end. The obstacle is modeled by cutting a hole ω into the flow domain, i.e. the the flow is computed on $\Omega\backslash\omega$, with no-slip boundary conditions on $\partial\omega$. These also hold on the remaining walls Γ_0. The flow around ω up until time T can be described by the equations

$$\begin{aligned} u_t + (u \cdot \nabla)u &= \frac{1}{\text{Re}}\Delta u - \nabla p \text{ in } [0,T] \times \Omega\backslash\omega \\ \text{div}\, u &= 0 \text{ in } [0,T] \times \Omega\backslash\omega, \\ u &= u_{\text{in}} \text{ on } \Gamma_{\text{in}}, \\ \frac{1}{\text{Re}}\partial_\nu u - p\nu &= 0 \text{ on } \Gamma_{\text{out}}, \\ u &= 0 \text{ on } \Gamma_0, \\ u &= 0 \text{ on } \partial\omega. \end{aligned} \tag{2.24}$$

The premise of the shape optimization problem will be the following: Find an obstacle geometry ω, such that the flow given by the system (2.24) minimizes the energy dissipation among all flows belonging to some geometry ω.

We will now formulate this problem in a more concise and approachable manner. For the moment assume the map $\omega \mapsto u(\omega)$, that assigns any obstacle $\omega \subset \Omega$ the solution u of (2.24), to be well defined. Concatenating this with taking the average dissipation $u \mapsto \fint_0^T D(t,u)\mathrm{d}t$, we arrive at the function

$$\begin{aligned} &J : \mathcal{P}(\Omega) \to \mathbb{R}, \\ &J(\omega) = \fint_{[0,T]} \int_{\Omega\backslash\omega} |\nabla u(\omega)| \mathrm{d}x\mathrm{d}t \end{aligned} \tag{2.25}$$

that assigns each conceivable geometry the energy dissipation of its resulting flow profile. Note that $\mathcal{P}(\Omega)$ denotes the power set of Ω.

The shape optimization problem of finding an obstacle geometry that minimizes the dissipation may now be posed as follows: Find $\omega^* \subset \Omega$ such that

$$\omega^* = \underset{\omega \in \mathcal{P}(\Omega)}{\text{argmin}}\, J(\omega). \tag{2.26}$$

This approach has several deficiencies. For one, depending on the form of ω, the set $\Omega\backslash\omega$ may be too pathological to solve the Navier-Stokes equations. Consider, for example, $\omega = \Omega\backslash(\Omega \cap \mathbb{Q}^d)$. Then it follows that $\Omega\backslash\omega = \Omega \cap \mathbb{Q}^d$ and thus $\partial(\Omega\backslash\omega) = \Omega$, which is obviously nonsensical. Less

severe but equally undesirable choices of ω may be sets which divide the domain or $\omega = \Omega$, which is actually a solution of problem (2.26)–no flow means no dissipation. It is clear that additional constraints on ω have to be imposed. For example that it be simply connected with smooth boundary, to avoid problems with the Navier-Stokes equations. To avoid trivial solutions similar to $\omega = \Omega$ one could also impose a volume constraint of the form $|\omega| = C$ or limit the region where ω may be to a further subset of Ω.

Next, $\mathcal{P}(\Omega)$ is an unsuitable space to attempt any minimization problems. It lacks structure and compactness results compared to common function spaces, let alone Euclidean space.

Obviously there are many subtleties to be considered when analyzing shape optimization problems. The approach taken in this thesis sidesteps these problems by parameterizing the object geometry through a finite set of coefficients. This restricts the amount of possible shapes to a comparatively small set and would be unsatisfactory from an analytical point of view, but in (industrial) applications one generally has a general idea of the geometry. For example when the goal is to improve upon an existing design, or additional constraints of the application, maybe predicated by physical realities of construction, give a priori constraints on the shape.

Another property of note is that, even though the dissipation is quadratic as a function in u, this does not even necessarily imply convexity in u, since the Navier-Stokes equation is non-linear, let alone in ω. This makes descent type algorithms unsuitable when searching for globally optimal solutions.

2.4 Black Box Solvers and Discretization

This section contains a description of the Navier-Stokes discretization chosen for the numerical examples and the reasoning behind this choice.

2.4.1 NaSt3DGPF (Finite Differences)

NaSt3DGPF is a three-dimensional, parallel, finite difference based Navier-Stokes solver developed by the group of Prof. Griebel at the Institute for Numerical Simulation in Bonn. It was used for initial experiments during this thesis.

It supports explicit second order schemes for time discretization, a choice of second or first order upwind schemes or central differences on a staggered grid for spatial discretization and a Chorin projection scheme for decoupling of the incompressibility equation. The details of the implementation can be found in Croce (2002) or, for the same principle method but in two dimensions, in Griebel et al. (1998).

More complex geometries than rectangles are handled using a flag field approach, storing geometry information about each cell of the finite difference grid. This approach is binary: Every cell is either part of the fluid domain, or an obstacle cell. This means that shapes on scales larger than the mesh width can easily be approximated by flagging all cells inside the shape as obstacle cells. This gives a good approximation of the general shape as well as the volume, but since every cell is either completely part of the fluid domain or completely an obstacle cell, this leads to a very rugged approximation of the surface of the obstacle. An illustration of this is given in figure 2.2, left image. Approximations of this kind show well known pathologies. For example,

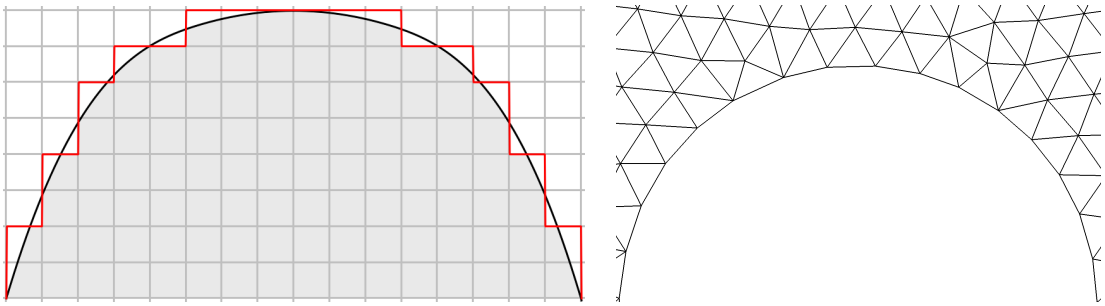

Figure 2.2: On the left is a approximation to a circular geometry through a flagfield approach, the shaded area is the obstacle and the resulting boundary is red. On the left is a similar object approximated through a triangulation.

when taking smaller mesh widths in figure 2.2 the length of the border outlined in red will not converge to the length of the black bounding curve, even though the borders itself will converge in a point wise sense. For this reason, the solver was switched to another one based on finite elements.

2.4.2 FEniCS (Finite Elements)

The second fluid solver considered was a FEniCS based finite element solver. This section gives a short introduction into the nature of FEniCS and the discretization used for the Navier-Stokes equation.

As opposed to NaSt3D, FEniCS is not a single purpose fluid dynamics solver. Rather, it is a collection of tools that allow for the automatized finite element based solution of, in principle, arbitrary partial differential equations. Its main components are the DOLFIN Python interface (see Logg and Wells (2010) and Logg et al. (2012c)), the Unified Form Language (UFL, Alnæs (2012)), and the FEniCS Form Compiler (FFC, Kirby and Logg (2006), Logg et al. (2012b), and Ølgaard and Wells (2010)). A thorough introduction to the complete package and its application to a variety of problems is given in Logg et al. (2012a), the scripts used for the computations in this thesis are based on the demo applications distributed together with the software.

The UFL is a domain specific language to describe variational forms and functionals in a way that closely follows the mathematical notation. For example, the bilinear form

$$b(u, v) = \int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}x, \tag{2.27}$$

in the weak formulation of the Laplace equation is, in Python, defined by the UFL expression `b = inner(grad(u), grad(v))*dx`. This makes it possible to transfer the mathematical description of the problem directly into code.

Having described the problem in such a way, the generation of finite element approximations can be as simple as loading a triangulation, specifying the kind of finite element spaces the functions used in the description are defined in and asking for a solution. The form compiler then generates C++ code to assemble the linear system resulting from the description and calls a linear solver. calls PETSc routines (see Balay et al. (2013b), Balay et al. (2013a)) for their solution.

In case of the Navier-Stokes equation the time variable has to be taken into account separately, since FEniCS doesn't explicitly support any time discretization. What it does though, is expose the underlying linear structure of the abstract functions and spaces to some extent. This makes it possible to pre-assemble all matrices outside of the time loop and directly interact with the PETSc interface, which exposes a variety of linear solvers (e.g. PCG, BiCGStab, GMRES, ...) and preconditioners (e.g. Jacobi, Gauß-Seidel, Algebraic Multigrid, ...). For more details on PETSc see Balay et al. (2013b), Balay et al. (2013a).

The discretization used is a method of lines ansatz which in turn uses a Chorin projection (c.f. Chorin (1968)) for time stepping as well as decoupling of the continuity equation and a FEM discretization for the spatial part of the functions. It is described in the following paragraphs.

Time Discretization and Decoupling

For the time discretization partition the interval $[0, T]$ into a grid $0 = t_0, \ldots, t_N = T$ with step size Δt and consider u^n, p^n the approximations of u, p at time t_n. The Chorin projection method is similar to an explicit Euler method in that the time derivative is replaced by a forward difference quotient

$$\partial_t u(x, t_n) \approx \frac{u^{n+1} - u^n}{\Delta t}. \tag{2.28}$$

Additionally, to decouple the momentum equation from the incompressibility condition, the pressure is simply dropped from the equation to compute a intermediary velocity field $\tilde{u}$ instead

of u^{n+1} via the explicit relation

$$\frac{\tilde{u} - u^n}{\Delta t} + (u^n \cdot \nabla)u^n - \frac{1}{\mathrm{Re}}\Delta u^n = f^n. \tag{2.29}$$

To compute the pressure and recover the actual divergence free velocity at the next time step, (2.29) is subtracted from the momentum equation with forward difference quotient. Together with the continuity condition at time t_{n+1} this results in the system

$$\begin{aligned} \frac{u^{n+1} - \tilde{u}}{\Delta t} + \nabla p^{n+1} &= 0, \\ \operatorname{div} u^{n+1} &= 0. \end{aligned} \tag{2.30}$$

Taking the divergence of the first equation and rearranging gives

$$-\Delta p^{n+1} = \frac{1}{\Delta t}\operatorname{div} \tilde{u}, \tag{2.31}$$

which can be solved for p^{n+1} and then enables u^{n+1} to be computed through

$$u^{n+1} = \tilde{u} + \Delta t \nabla p^n. \tag{2.32}$$

Equations (2.29), (2.31) and (2.32) now allow u^{n+1} to be determined from u^n through straightforward explicit computations and solving a single Poisson equation.

Spatial Discretrization

For the spatial discretization recall the discussion in section 2. The same way we arrived at the variational form of the full Navier-Stokes equation then, we can derive weak forms of the equations derived in the previous paragraph by integrating over Ω and applying Greens theorem where appropriate. This results in the variational relation of the intermediary velocity step (2.29),

$$\int_\Omega \frac{\tilde{u} - u^n}{\Delta t} \cdot v + (u^n \cdot \nabla)u^n \cdot v - \frac{1}{\mathrm{Re}}\Delta u^n \cdot v \,\mathrm{d}x = \int_\Omega f^n \cdot v \,\mathrm{d}x \quad \text{for all } v \in H_0^1(\Omega, \mathbb{R}^d), \tag{2.33}$$

the pressure computation (2.31)

$$\int_\Omega \nabla p^{n+1} \cdot \nabla q \,\mathrm{d}x = \int_\Omega \frac{1}{\Delta t}\operatorname{div} \tilde{u}\, q \,\mathrm{d}x \quad \text{for all } q \in H^1(\Omega, \mathbb{R}^d), \tag{2.34}$$

and the velocity correction (2.32)

$$\int_\Omega u^{n+1} \cdot v \, \mathrm{d}x = \int_\Omega (\tilde{u} + \Delta t \nabla p^n) \cdot v \, \mathrm{d}x \quad \text{for all } v \in H_0^1(\Omega, \mathbb{R}^d). \tag{2.35}$$

The Galerkin ansatz proposes to replace the function spaces of the weak formulation by finite dimensional subspaces $V_h \subset H_0^1(\Omega, \mathbb{R}^d)$ and $Q_h \subset H^1(\Omega, \mathbb{R}^d)$, where we already use h to signify the future dependence on some mesh parameter. I.e. instead of looking for $p^{n+1} \in H^1(\Omega, \mathbb{R}^d)$ such that (2.34) holds for all $q \in H^1(\Omega, \mathbb{R}^d)$, we look for $p_h \in Q_h$ such that (2.34) holds for all $q_h \in Q_h$. Doing this, the variational problems above reduce to finite dimensional linear equations: If $\phi_1, \ldots, \phi_M$ is some basis of Q_h, we can write p_h^{n+1} and every q_h as

$$p_h^{n+1}(x) = \sum_{i=1}^N p_{h,i}^{n+1} \phi_i(x), \; q_h(x) = \sum_{i=1}^N q_{h,i} \phi_i(x). \tag{2.36}$$

Inserting these representations allows us to rewrite the left hand side of (2.34) as

$$\int_\Omega \nabla \left(\sum_{i=1}^N p_{h,i}^{n+1} \phi_i(x) \right) \cdot \nabla \left(\sum_{j=1}^N q_{h,j} \phi_j(x) \right) \mathrm{d}x = \sum_{i=1}^N \sum_{j=1}^N p_{h,i}^{n+1} q_{h,j} \int_\Omega \nabla \phi_i \cdot \nabla \phi_j \, \mathrm{d}x \tag{2.37}$$

and the right hand side as

$$\sum_{i=1}^N q_{h,i} \int_\Omega \frac{1}{\Delta t} \mathrm{div}\, \tilde{u} \, \phi_i(x) \, \mathrm{d}x. \tag{2.38}$$

Writing $M_{h,ij} = \int_\Omega \nabla \phi_i \cdot \nabla \phi_j \, \mathrm{d}x$ for the *stiffness matrix* and $b_{h,i} = \int_\Omega \frac{1}{\Delta t} \mathrm{div}\, \tilde{u} \, \phi_i(x) \, \mathrm{d}x$ we see that (2.34) (in the finite dimensional case) is equivalent to $q_h^T M p_h^{n+1} = q_h^T b$ for all $q_h \in Q_h$ or simply

$$M_h p_h^{n+1} = b_h. \tag{2.39}$$

In this way, each of the three steps in the Chorin pressure splitting can be reduced to a system of linear equations.

What is left is to choose suitable spaces V_h and Q_h that lead to a stable and accurate method. Let $\mathcal{T}_h$ be a triangulation of Ω, i.e. $\mathcal{T}_h$ is a collection of triangles T, such that $\cup_{T \in \mathcal{T}_h} T$ is an approximation of Ω and satisfies the usual constraints asked of such a triangulation (no hanging nodes, no degenerate triangles, etc.). The spaces are then chosen to be

$$V_h = \{v \in C^1(\Omega, \mathbb{R}^d) : v|_T \text{ is a quadratic polynomial for all } T \in \mathcal{T}_h\} \tag{2.40}$$

and

$$Q_h = \{q \in C(\Omega, \mathbb{R}) : v|_T \text{ is linear for all } T \in \mathcal{T}_h\}. \tag{2.41}$$

This choice corresponds to a so-called *Taylor-Hood* finite element approach, which prescribes that the pressure discretization be of one order less than the velocity discretization. If both were taken to be quadratic, or both linear, the resulting discretization would be unstable as per the *inf-sup* condition. For a more detailed discussion of finite element spaces for saddle point problems see Braess (2007), p. 157.

The spatial accuracy is controlled via a mesh-width parameter h. A common constraining relation between the triangles in $\mathcal{T}_h$ and the mesh-width parameter is that the diameter of every triangle in $\mathcal{T}_h$ satisfies $\operatorname{diam}(T) \leq 2h$ and that the triangulation as a whole is *uniform*. That means that there is some constant $c > 0$ such that every triangle contains a circle of diameter $cd > 2h$.

Computing derived quantities

The finite element method also allows for the easy computation of some derived quantities. Let $\psi_{h,1}, \ldots, \psi_{h,M}$ be a basis of V_h and $v_h = \sum_{i=1}^{M} v_{h,i}\psi_{h,i} \in V_h$, then the energy dissipation of v_h is given by

$$\int_\Omega |\nabla u|^2 \,\mathrm{d}x = \sum_{i=1}^{M}\sum_{j=1}^{M} v_{h,i}v_{h,j} \int_\Omega \nabla\psi_{h,i} : \nabla\psi_{h,j} \,\mathrm{d}x. \tag{2.42}$$

This way, the energy dissipation of the solution of the discretized Navier-Stokes equation can be computed exactly at any given time.

3 Mesh Generation using Distmesh

An important aspect of the finite element simulation is a high quality mesh of the domain. In this case there is the added complexity of having to generate meshes for parameterized geometries in an automated and unsupervised environment. While very good meshing software is freely available–cf. Triangle, by Shewchuk (1996)–none of the packages that the author found satisfied all needs of the application. The meshing environment should deliver high quality meshes, easily run automated and be quickly adaptable to changes in the geometry input. To this end the distmesh algorithm by Persson (2004), see also Persson and Strang (2004), was implemented in Python using the NumPy package and combined with a level set generator using a fast marching algorithm.

The distmesh algorithm can be seen as an iterative procedure to improve an existing mesh with respect to element quality and desired mesh width, initialized using a Delaunaty triangulation of an initial node set. Consider a distribution of points $p^1 = (p_x^1, p_y^1), \ldots, p^N = (p_x^N, p_y^N)$ sorted into a matrix

$$P = \begin{bmatrix} p_x^1 & p_y^2 \\ \vdots & \vdots \\ p_x^N & p_y^N \end{bmatrix} \in \mathbb{R}^{2\times N} \tag{3.1}$$

together with a Delaunay triangulation $\mathcal{T}$ on those points. Now think of each edge in the triangulation as a linear spring that exerts a force on its endpoints that is proportional to the deviation of the length of the edge to the desired mesh width. That means if an edge is shorter than the desired mesh width its endpoints are pushed apart. For performance reasons contraction of an edge is not considered. A mesh is then assumed to be of high quality if it achieves an equilibrium of the spring forces

$$F(P) = 0, \tag{3.2}$$

where $F : \mathbb{R}^{2\times N} \to \mathbb{R}^{2\times N}$ denotes the accumulated forces acting on each vertex. Since the forces are determined using the Delaunay triangulation of the point set, this function is neither continuous nor necessarily well defined, since not all point distributions allow a unique Delaunay

triangulation. This is of no consequence to the algorithm but should be kept in mind, since it will not be reflected in the notation.

The exact formula used to compute the forces is

$$[F(P)]_i = \sum_{q \in E(p^i)} (h - \|p^i - q\|)^+ \frac{p^i - q}{\|p^i - q\|}, \tag{3.3}$$

where h is the desired edge length in the mesh, $a^+ := \max(a, 0)$ and $E(p^i)$ is the neighborhood of all points connected to p^i by an edge in $\mathcal{T}$. As was mentioned above, forces are only positive for edges that are too short. This achieves an expansion of the mesh structure as the iteration process goes on and allows the points to fill out the specified geometry.

To confine this expanding mesh structure to the geometry, the distmesh ansatz uses a description of the desired geometry by a *signed distance function*. If Ω is the region to be meshed, its signed distance function ϕ is defined by

$$\phi(x) = \begin{cases} -\operatorname{dist}(x, \partial\Omega) & \text{if } x \in \Omega \\ \operatorname{dist}(x, \partial\Omega) & \text{if } x \notin \Omega \end{cases} \tag{3.4}$$

with

$$\operatorname{dist}(x, \partial\Omega) = \inf_{y \in \partial\Omega} \|x - y\|_2. \tag{3.5}$$

This function can be utilized to test whether a point x is in Ω via

$$x \in \Omega \iff \phi(x) < 0 \tag{3.6}$$

and to move points outside Ω back onto it via the projection

$$x \mapsto x - \phi(x)\nabla\phi(x). \tag{3.7}$$

The latter works since the negative gradient points in the direction of steepest descent, which in this case means the direction to the near-most point of $\partial\Omega$. Since $\phi(x)$ is the distance to $\partial\Omega$ it is also the exact step length needed to arrive at $\partial\Omega$ when moving along $\nabla\phi(x)$, if $\nabla\phi(x)$ is of unit length. The latter is a well known fact that is also used as a characterization of signed distance functions, namely that they satisfy the *eikonal equation*

$$\begin{aligned} \|\nabla\phi\|_2 &= 1 \text{ in } \Omega \\ \phi &= 0 \text{ on } \partial\Omega. \end{aligned} \tag{3.8}$$

Using the geometry information supplied by the signed distance function, the force equilibrium (3.2) is now solved in the spirit of a constrained minimization problem. This is done using a projected Richardson-like iteration with artificial time stepping:

$$P_{new} = P_{old} + \delta_t F(P_{old}). \tag{3.9}$$

After each iteration, points that have left the geometry (i.e. $\phi(p^i) > 0$) are projected back as described in equation (3.7). To account for the fact that the movement of points may cause the initial triangulation $\mathcal{T}$ to have the Delaunay property, a new triangulation is calculated after the total movement of the points in relation to the positions at the time of the last triangulation exceeds a prescribed tolerance. The complete procedure is summarized in algorithm 1.

Algorithm 1 Distmesh

Input: Signed distance function ϕ, initial point distribution p, desired mesh width h_0, iteration parameters $\delta_t, \epsilon_T, \epsilon_F$.
Output: Triangulation $\mathcal{T}$ of the region described by ϕ.

```
Let p_old = p
Compute Delaunay triangulation T of p
while δ_t F(p) > ε_F do
    Update p = p + δ_t F(p)
    for p ∈ P with p > φ(p) do
        Project p = p − φ(p)∇φ(p)
    end for
    if ‖p − p_old‖_2 > ε_T then
        Recompute Delaunay triangulation T of p
        Let p_old = p
    end if
end while
```

For all computations in this thesis, the initial points were supplied by a quasi-random Halton sequence (cf. section 5.1) in $[0, l_x] \times [0, l_y]$ with $N = \lfloor l_x \cdot l_y \cdot h^{-2} \rfloor$, where $[0, l_x] \times [0, l_y]$ is a box containing Ω. N is the amount of points in a uniform grid on $[0, l_x] \times [0, l_y]$ of mesh width h and thus a logical choice for the amount of points needed for a unstructured mesh of the same fineness. Superfluous points inside the bounding box but outside Ω are deleted once before the start of the iteration.

Figure 3.1 shows an exemplary quasi-random distribution of vertices and their Delaunay triangulation as well as the converged result of the distmesh algorithm for a perforated circle geometry using an exact level set function. Figure 3.2 shows the same for an actual problem geometry as described later in this thesis. Both plots are colored by a heuristic measure for element quality

given by

$$\text{Quality(T)} := \frac{|T|}{\max(l_0, l_1, l_2)^2}, \tag{3.10}$$

where l_0, l_1 and l_2 denote the side lengths of the triangle T.

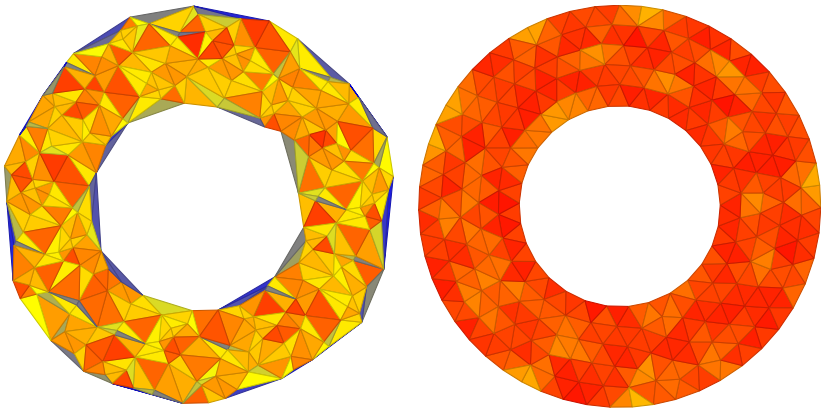

Figure 3.1: Mesh of a perforated circular geometry. Right: Initial triangulation of quasi-random points, left: Result of the algorithm.

To generate the level set function for the parameterized geometries a fast marching method was used to solve the corresponding eikonal equation on a high resolution regular grid of a region containing the geometry. Projections were computed using this approximation and bilinear interpolation.

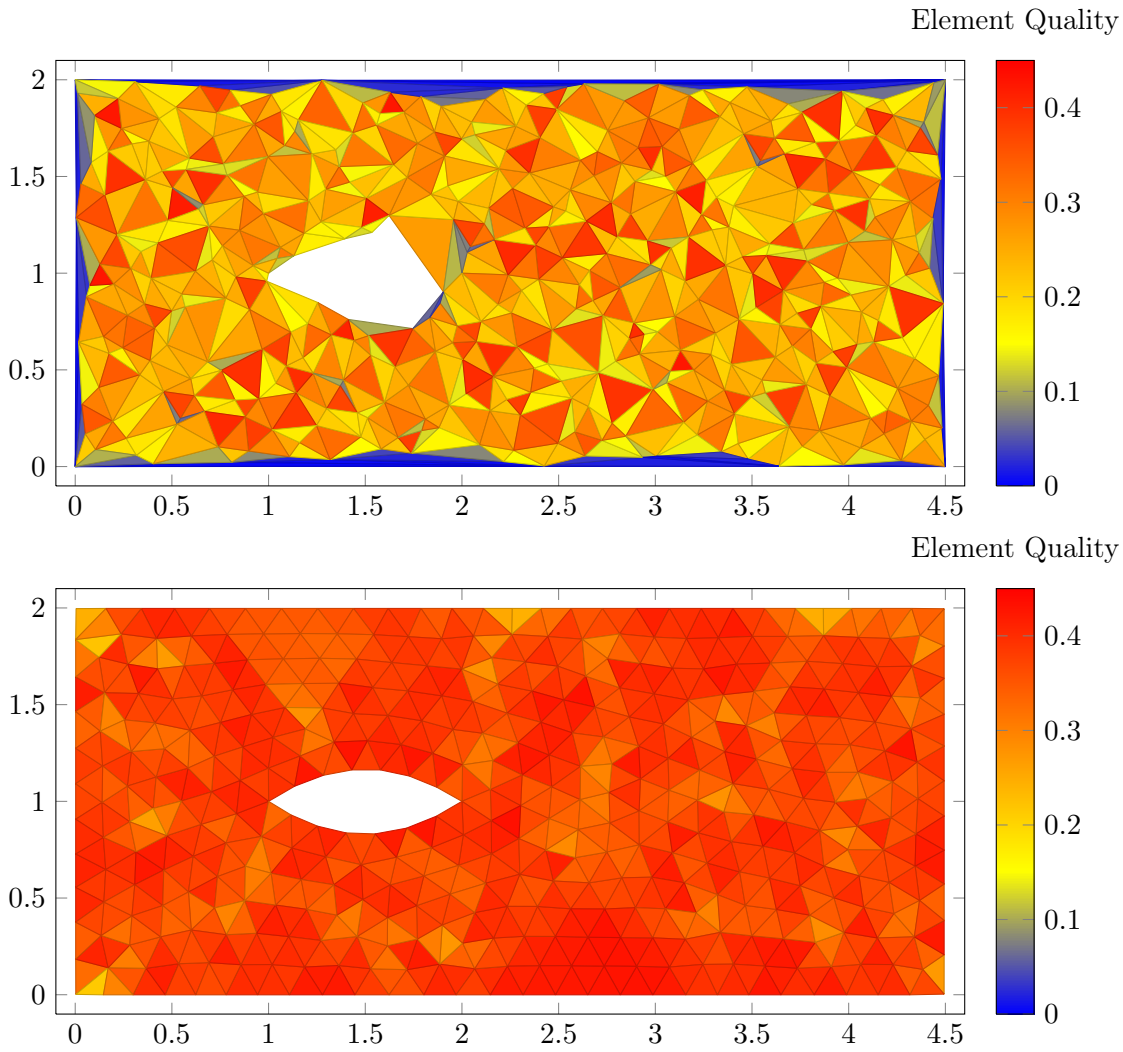

Figure 3.2: Mesh of a obstacle flow geometry. Top: Initial triangulation of quasi-random points, bottom: Result of the algorithm.

4 Parameterization

The shape space is discretized by parameterizing a range of (pre-defined) possible geometries, making the problem accessible to classical optimization techniques. This section explains the tools used for the parameterization and how they are used in the computational examples.

4.1 Basis Splines

The parameterization is conducted using B-splines, which shall be defined in this section. The notation and presentation of the subject is based on Deuflhard and Hohmann (2002), further information about spline curves and fast algorithms can be found in Piegl and Tiller (1995).

A *spline of degree* k on $[a, b]$ with respect to a partition $a = t_0 < t_1 < \ldots < t_{l+1} = b$ is a function $\phi \in C^{k-1}([a, b], \mathbb{R})$ such that ϕ restricted to $[t_{i-1}, t_i]$ is a polynomial of degree k for every $i = 1, \ldots, l + 1$. These splines form a vector space and a common choice of basis of this vector space are *B-splines*.

Definition 4.1 (B-Splines)**.** *Let* $\tau_1 \leq \ldots \leq \tau_n$ *be some sequence of knots, the* B-Splines $N_{i,k}$ *of order* k *and* $i = 1, \ldots, n - k$ *are recursively defined by*

$$\begin{aligned} N_{i,1}(t) &= \chi_{[\tau_i, \tau_{i+1})}(t), \\ N_{i,k}(t) &= \frac{t - \tau_i}{\tau_{i+k-1} - \tau_i} N_{i,k-1}(t) + \frac{\tau_{i+k} - t}{\tau_{i+k} - \tau_{i+1}} N_{i+1,k-1}(t). \end{aligned} \tag{4.1}$$

If the knots in definition 4.1 are chosen according to

$$\begin{aligned} \tau_1 = \ldots = \tau_k &= t_0 \\ \tau_{j+k} &= t_j \text{ for } j = 1, \ldots, l \\ \tau_{l+k+1} = \ldots = \tau_{l+2k} &= t_{l+1}, \end{aligned} \tag{4.2}$$

the B-splines $N_{i,k}$, $i = 1, \ldots, l+1$ form a basis of the space of splines of degree k with respect to the partition $t_0, \ldots, t_{l+1}$. That is, every spline in that space is of the form

$$\phi(s) = \sum_{j=1}^{l+k} c_j N_{jk}(s) \tag{4.3}$$

for some $c \in \mathbb{R}^{l+k}$.

4.2 Parameterization of the Geometry

As an example we will describe the parameterization of the geometry in the obstacle flow problem from sections 2.3 and 6.3.1. The other examples from chapter 6 require only minor adjustments.

Since the surrounding geometry and the boundary conditions are symmetric around the obstacle (c.f. figure 6.6), we will assume that the minimizing object will also be symmetric. This stands to reason since reflecting around the center line of the object drawn in figure 6.6 does not change the problem.

For this discussion we assume that the obstacle is situated in such a way that its leftmost part touches the y-axis and its axis of symmetry coincides with the x-axis, this can be achieved by simple translation. If it is further assumed that the border has a sufficiently smooth and simple shape, the upper part of the boundary (above the x-axis) is simply the graph $y = \gamma(x)$ of some function γ over the interval $[0, L]$. This situation is illustrated in figure 4.1.

The parameterized geometries with n degrees of freedom are now taken to be all shapes of this form, where the bounding function γ is given by a cubic spline with n degrees of freedom:

$$\gamma_n(x) = \sum_{j=1}^{n} c_j N_{j3}(x). \tag{4.4}$$

The search space is now given by the finite dimensional coefficients $c \in \mathbb{R}^n$ of the B-spline expansion, which generate the obstacles

$$\omega(c) = \{(x, y) \in \mathbb{R}^2 : |y| \leq \gamma_n(x)\}. \tag{4.5}$$

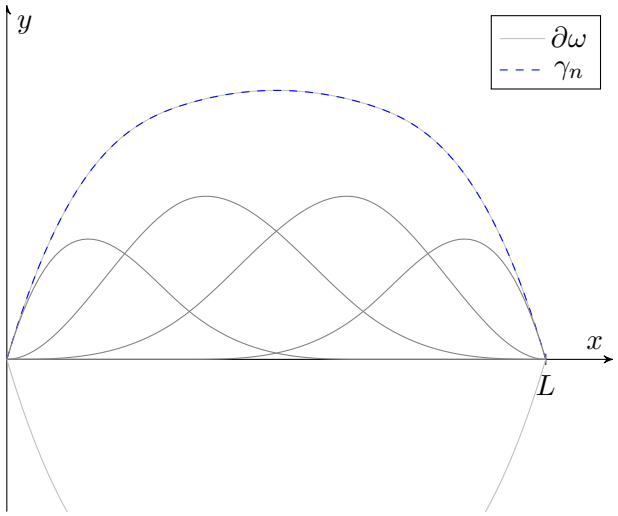

Figure 4.1: Parameterization γ_n of the upper part of the obstacle boundary $\partial\omega$. The geometry is translated, such that the left side of the obstacle touches the x_2-axis and the axis of symmetry of the object lies on the x_1-axis. Drawn in gray are the scaled basis functions of the B-spline expansion of γ_n.

Constraints

The B-spline parameterization allows for certain kinds of shape constraints on ω to be rephrased as linear inequality constraints. In the parameterization in figure 4.1 for example, it is easy to see that the volume of the parameterized obstacle is two times the area between γ_n and the x-axis. In other words:

$$|\omega| = 2\int_0^L \gamma_n(x)\,\mathrm{d}x. \tag{4.6}$$

Inserting the basis expansion of γ_n this is equivalent to

$$\frac{|\omega|}{2} = \int_0^L \sum_{i=1}^{n} c_i N_{ik}(x)\mathrm{d}x = \sum_{i=1}^{n} c_i \int_0^L N_{ik}(x)\mathrm{d}x = I^T c \tag{4.7}$$

where $I^T = (\int_0^L N_{1k}(x)\mathrm{d}x, \ldots, \int_0^L N_{nk}(x)\mathrm{d}x)$. Using this, a volume constraint of the form $V_{\min} \leq |\omega| \leq V_{\max}$ on the obstacle can be rewritten as a linear inequality constraint on the parameters

$$V_{\min} \leq I^T c \leq V_{\max}. \tag{4.8}$$

Restricting the size of the obstacle in a point wise sense can also be formulated as a linear constraint. Forcing ω to lie in the box $[0, L] \times [y_{\min}, y_{\max}]$ is done by taking control points $x_1, \ldots, x_N \in [0, L]$ corresponding to the maxima of the basis functions N_{j3} and demanding that $y_{\min} \leq \gamma_n(x_i) \leq y_{\max}$ for each $i \in \{1, \ldots, N\}$. Using, again, the basis expansion of γ_n this is

rewritten as

$$\begin{aligned} & y_{\min} \leq \gamma_n(x_i) \leq y_{\max} \text{ for all } i \in \{1, \ldots, N\} \\ \iff & y_{\min} \leq \sum_{j=1}^{n} c_j N_{jk}(x_i) \leq y_{\max} \text{ for all } i \in \{1, \ldots, N\} \\ \iff & y_{\min} \leq Ac \leq y_{\max} \text{ (component wise)} \end{aligned} \tag{4.9}$$

where

$$A = \begin{pmatrix} N_{1,3}(x_1) & N_{2,3}(x_1) & \ldots & N_{n,3}(x_1) \\ N_{1,3}(x_2) & N_{2,3}(x_2) & \ldots & N_{n,3}(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ N_{1,3}(x_m) & N_{2,3}(x_m) & \ldots & N_{n,3}(x_m) \end{pmatrix} \tag{4.10}$$

and $y_{\min}$ and $y_{\max}$ are identified with the constant vectors of the same value.

5 Optimization

Optimization algorithms aim to solve problems of the form: Find $x \in U$ such that

$$x = \operatorname*{argmin}_{y \in U} f(y) \tag{5.1}$$

for some objective function $f : U \to \mathbb{R}$ and the feasible set $U \subset \mathbb{R}^n$. An important distinction has to be made between two fundamentally different approaches to minimization problems:

- *Local search* methods iteratively generate a succession of points by performing e.g. line search along the gradient at the iterates. These methods are mostly guaranteed to converge to a stationary point of f (i.e. a point x with $\nabla f(x) = 0$) and are called globally convergent if they do so from every starting point.
- *Global search* methods, which try to find the actual minimal point across the whole of U. To work in any generality these methods have to disregard gathered information at least to some degree for enough coverage.

Depending on the function f and the feasible set this problem may be very easy or very hard to solve. For example, if f and U are a convex function and set respectively, even the global search problem is easily solved by local search methods such as gradient descent or the compass search explained later in this section. On the other hand, if f is a function with many local minima at different scales, we have to apply global optimization methods using a high number of function evaluations to have any chance at a good approximation.

Following Novak (1988), consider an abstract optimization method $x = \Phi(N, f)$ with the property that it evaluates any function to be minimized first on a fixed point x_1, then at successively chosen points $x_2(x_1, f(x_1))$, $x_3(x_1, x_2, f(x_1), f(x_2))$, ..., $x_N(x_1, \ldots, x_{N-1}, f(x_1), \ldots, f(x_{N-1}))$ and returns the point with the smallest function value. This captures derivative free minimization methods that use no a-priori information about the function f, such as Lipschitz-constants or similar, and up to N function evaluations. For such an optimization method and a set of functions F we consider

$$\sup_{f \in F} f(\Phi(N, f)) - \min_{x \in U} f(x), \tag{5.2}$$

the maximal error that occurs when applying the minimization method Φ with N evaluations to any function in F. Taking the infimum over all optimization methods Φ that can be described in the way outlined above, we can determine the worst case error of the optimization method with the best worst case behavior over the whole of the functions in F using N function evaluations:

$$E(F, N) = \inf_{\Phi \text{ as described}} \sup_{f \in F} f(\Phi(N, f)) - \min_{x \in U} f(x). \tag{5.3}$$

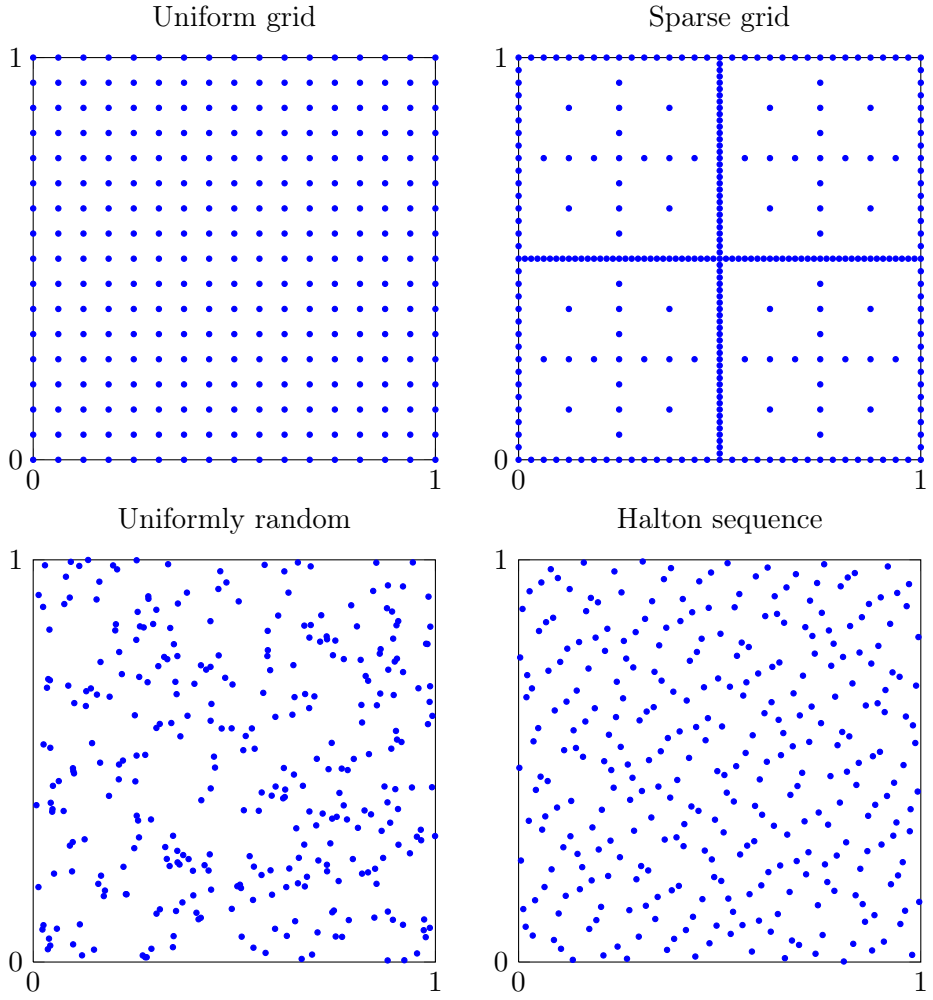

Figure 5.1: Different non-adaptive global search approaches.

If we consider the function class

$$F = \{f \in C^{k,\alpha} : \|f\|_{C^{k,\alpha}} \leq 1\} \tag{5.4}$$

of k-times differentiable, α-Hölder continuous functions with norm at most one, Novak (1988) has shown the sharp estimate

$$E(F, N) \sim N^{-\frac{k+\alpha}{n}}. \tag{5.5}$$

This means that there is no deterministic global optimization method on operating on those functions that has a better worst case behavior than the one above. In fact, it is also shown that this bound is already achieved the algorithm that simply evaluates f on the points of an appropriately fine regular grid, as illustrated in figure 5.1.

Of special note is the fact that the convergence rate deteriorates with growing dimension n. This is a well known fact in many applications and known as the *curse of dimensionality*. If, for example, the dimension of the problem is $n = 6$ (which is the case in some of the problems considered later), we would have to assume that the objective function is Lipschitz-continuous and five times continuously differentiable to guarantee a worst case error rate of at least N^{-1} using the result above.

One way to alleviate the effect of the curse of dimensionality is to introduce (quasi-) randomized methods. Monte Carlo and quasi-Monte Carlo methods are widely known as numerical quadrature methods, where they achieve convergence rates of $N^{-1/2}$ and N^{-1} respectively. Both approaches are discussed more comprehensively later in this section, but it is necessary to emphasize that both methods suffer from the fact that while the rate is dimension independent, they are also independent of smoothness. Thus even if the objective is of some smoothness, we cannot expect to gain any advantage using these methods. In figure 5.1 both methods are illustrated.

Another approach are sparse grid based methods. Going back to the work of Smolyak (1963), they have in recent time been used not only for quadrature methods but also for e.g. partial differential equations, spectral approximation or wavelet methods. Introductions are given in Gerstner and Griebel (1998) or Garcke (2013). For a basic idea (anticipating the discussion of section 5.3) consider a normal uniform grid of mesh width 2^L. All points on it are given by the formula

$$\left(\sum_{i_1=1}^{l_1} a_{1,i_1} 2^{-i_1}, \ldots, \sum_{i_n=1}^{l_n} a_{n,i_n} 2^{-i_n} \right) \tag{5.6}$$

where the multi index $l \in \mathbb{N}^n$ ranges over all l with $\|l\|_\infty \leq L$. To construct a sparse grid we restrict the range of the multi indices to those that satisfy $\|l\|_1 \leq L$, which suffices to retain most of the approximative properties of a full grid in many applications. The cost of this is a moderate increase in smoothness requirements. While the amount of points in the full grid of level L is of the order $O(2^{nL})$, the sparse grid only uses $O(2^L \log(2^L)^{n-1})$ points (Garcke, 2013). Figure 5.2 illustrates the magnitude of this difference for six dimensional problems.

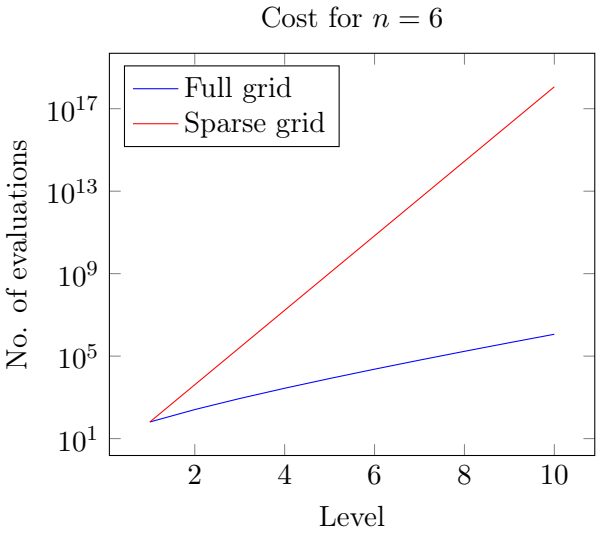

Figure 5.2: Comparison of the number of evaluations used in the full and sparse grid methods of the same level.

Novak and Ritter (1996) note that a non-adaptive method simply evaluating the objective function at sparse grid points (of ascending level) gives a bound on the worst case error $E_{\text{Sparse}}(N)$ of the first N points of

$$E_{\text{Sparse}} \sim N^{-1}(\log N)^{2(n-1)} \tag{5.7}$$

over all functions in in the mixed first order Sobolev space $W^{1,\infty}_{\text{mix}}$ with norm $\|f\|_{W^{1,\infty}_{\text{mix}}} \leq 1$. The mixed Sobolev space differs from the normal one in that all differentials of total degree 1 have to be bounded as opposed to the partial degree. I.e. $\partial_1\partial_2 f$ has to exist and be bounded for a function to lie in $W^{1,\infty}_{\text{mix}}$ but not in $W^{1,\infty}$. In essence, one trades the ∞-norm for the 1-norm in the index space of the derivatives to trade the 1-norm for the ∞-norm in the index space of the grid points. While this result gives essentially the same rate as the quasi-Monte Carlo method would give, it stands to reason that the sparse grid approaches error bounds improve with increasing smoothness (as is the case in other applications), leading to bounds of the form $E \sim N^{-s}\log(N)^{n-1}$. This means sparse grid based methods improve with increasing smoothness of the involved function while retaining their error rate with increasing dimension, modulo a logarithmic factor.

The hyperbolic cross method explained in section 5.3 aims to take advantage of these worst case estimates while introducing some degree of adaptivity and locality to improve pre-asymptotic behavior.

5.1 Quasi-Monte Carlo Method

The quasi-Monte Carlo optimization method is basically a deterministic variant of the Monte Carlo method, so the latter will be given a short introduction. A detailed discussion of both approaches can be found in Niederreiter (1992).

In its most basic form, as exemplified by optimization of a function f on $[0,1]^n$ using N samples, the Monte-Carlo method draws uniformly random samples $x_1, x_2, \ldots x_N \in [0,1]^n$ and computes an approximation to the global optimum of f on $[0,1]^n$ via

$$x_{\min} \approx \operatorname*{argmin}_{1 \le i \le N} f(x_i). \tag{5.8}$$

The quasi-Monte Carlo method replaces the random samples by a deterministic sequence of points $P = (x_1, x_2, \ldots) \subset U$ and computes the approximation to the global minimum using N samples as

$$x_N^* = \operatorname*{argmin}_{x_i \in P,\, 1 \le i \le N} f(x_i). \tag{5.9}$$

The point of this is to gain finer control over the distribution of points, which helps to avoid clustering inherent to truly random methods and leads to better error estimates. Obviously, the sequence P has to satisfy some condition to make the method converge. In case of the better known quasi-Monte Carlo quadrature, the sequences that ensure convergence are the so called *low discrepancy series*. They are chosen in such a way that their distribution approximates the measure that is to be integrated. In case of the Lebesgue measure that means

$$\sup_{B \text{ box in } U} \left| \frac{\#\{i : x_i \in B\}}{N} - |B| \right| \to 0 \text{ as } N \to \infty. \tag{5.10}$$

For optimization problems the *dispersion* of the sequence, which is defined by

$$d_N(P;U) = \sup_{x \in U} \min_{1 \le i \le N} \|x - x_i\|_2. \tag{5.11}$$

has to approach zero to achieve convergence. This is shown in the following result:

Theorem 5.1 (Niederreiter, 1992, p. 149)**.** *Let $U \in \mathbb{R}^n$ be bounded. For $P = (x_1, x_2, \ldots, x_N) \subset U$ with dispersion $d_N = d_N(P;U)$ it holds that*

$$\inf_{x \in U} f(x) - f_N^* \le \omega(f; d_N) \tag{5.12}$$

where ω is the modulus of continuity

$$\omega(f; d_N) = \sup_{x,y \in U \ s.t. \ \|x-y\|_2 \leq d_N} |f(x) - f(y)| \tag{5.13}$$

This implies if f is, for example, Lipschitz continuous with constant L_f, the approximation satisfies

$$\inf_{x \in U} f(x) - f_N^* \leq L_f d_N \to 0 \tag{5.14}$$

and the dissipation dictates the rate of convergence. Niederreiter (1992) notes that every low discrepancy series is also a low dispersion series.

Halton Sequence

For the comparative results in chapter 6 and the initialisation of the distmesh procedure a low dispersion Halton sequence was used. It was first published in Halton (1964) and is based on prime reciprocals, the full procedure is given in algorithm 2.

Algorithm 2 Halton Sequence

Input: Required dimension n and amount of samples N, n prime numbers $p^1, \ldots, p^n$ and $\epsilon > 0$.
Output: First N members of the n-dimensional Halton sequence.

```
x_0 = 0.
for k = 1, ..., N − 1 do
    for i = 1, ..., n do
        z = 1 − x_i^{k−1}
        v = p_i^{−1}
        while z < v + ε do
            v = v / p_i
        end while
        x_i^k = x_i^{k−1} + (p_i + 1)v − 1
    end for
end for
```

5.2 Asynchronous Parallel Pattern Search

The *asynchronous parallel pattern search* (APPS) is a derivative free and globally convergent local search method, that combines several properties desirable in black-box simulation based optimization. It supports linear constraints while maintaining full feasibility of all trial points and can maintain its convergence properties even if a (moderate) number of evaluations in each

iteration fail to give a result. The method is an asynchronously parallel variant of pattern search (which is also known as *generalized set search* or GSS) first published for linear constraints in Griffin et al. (2008) and was implemented as part of the HOPSPACK software (hybrid optimization software package, see Plantenga (2009)).

GSS is an advancement of what is sometimes called compass search, which we will discuss in detail in the following paragraph based on the exposition in Lewis et al. (1998). After that, the extensions to general search directions and the general ideas underlying the generalization to linear constraints and parallelization will be presented following the descriptions in the survey article Kolda et al. (2003).

Compass Search

Compass search is the most basic pattern search for optimization problems without constraints. For the time being assume that the optimization problem to be solved is of the following form: Find $x^* \in \mathbb{R}^n$ such that

$$x^* = \operatorname*{argmin}_{x \in \mathbb{R}^n} f(x). \tag{5.15}$$

The algorithm starts with an initial point x_0 and step size h_0. In each iteration, the function is sampled around the current iterate x_k along each of the coordinate axes in both negative and positive direction with a step length of h_k, i.e. the new samples are $x_k \pm h_k e_i$ for $i = 1, \ldots, n$, where $e_i = (\delta_{ij})_{j=1}^n$ is the i-th unit vector. The first sample with a lower function value is taken as the new iterate. If none of the points improves upon the function value at the current point it remains the same and the step width is halved. This corresponds to the generating set search given in algorithm 3, where the set of search directions is

$$D_k = \{e_1, \ldots, e_n, -e_1, \ldots, -e_n\} \tag{5.16}$$

for every iteration k. Figure 5.3 illustrates the procedure: For a given step length h_k the algorithm moves from point to point on grid of mesh-width h_k centered on the first point that step length occurred on until no improvement can be made. At that point the resolution is halved and the process begins anew.

We will now show that even this simple method converges to a stationary point for fairly general functions. The convergence analysis of standard line search methods depends heavily on the approximation properties of the gradient of f. Consider a point $x \in \mathbb{R}^n$, a search direction $d \in \mathbb{R}^n$ and small step lengths $h > 0$. Then, assuming f is continuously differentiable, one has

$$f(x + hd) = f(x) + h\nabla f(x) \cdot d + O(h^2). \tag{5.17}$$

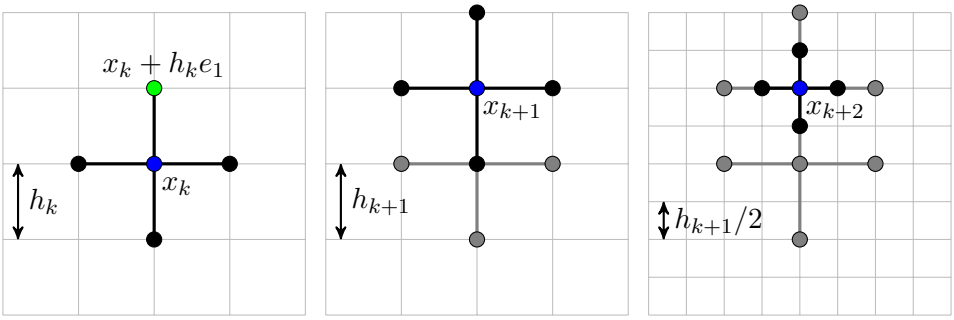

Figure 5.3: Illustration of the pattern search.

Algorithm 3 Example of a Generating Set Search

Input: Function $f : \mathbb{R}^n \to \mathbb{R}$ to minimize, starting step size h_0 and point x_0, minimal step size $h_{\min}$.

while $h > h_{\min}$ **do**
  Let D_k be some generating set for $\mathbb{R}^n$
  for $d \in D_k$ **do**
    if $f(x_k + h_k d) < f(x_k)$ **then**
      $x_{k+1} = x_k + h_k d$
      $h_{k+1} = h_k$
      break
    end if
  end for
  if no improvement was found **then**
    $x_{k+1} = x_k$
    $h_{k+1} = h_k/2$
  end if
  Set $k = k + 1$
end while

Recall that the negative gradient of f points in the direction of steepest descent. If the angle between $-\nabla f(x)$ and d is less than 90°, inserting

$$0 < \cos \angle(-\nabla f(x), d) = -\frac{\nabla f(x) \cdot d}{\|\nabla f(x)\| \, \|d\|} \tag{5.18}$$

into equation (5.17) gives

$$f(x + hd) = f(x) - h \cos \angle(-\nabla f(x), d) \, \|\nabla f(x)\| \, \|d\| + O(h^2). \tag{5.19}$$

Since the cosine is positive and the quadratic remainder approaches zero faster than h times a constant, it follows that

$$f(x + hd) < f(x) \tag{5.20}$$

for sufficiently small h. This shows that if d points in roughly the same direction as $-\nabla f(x)$, a decrease in the function value can be forced by taking a small enough step. For this reason directions d which fulfill the condition that their angle with $-\nabla f(x)$ is less than 90 degrees are called *descent directions*. Key to the convergence of all pattern search methods is that it is possible to find a descent direction in every iteration without knowing the gradient.

Theorem 5.2. *(Descent directions) Let $x \in \mathbb{R}^n$ with $\nabla f(x) \neq 0$. For some $i \in \{1, \dots, n\}$ and $\sigma \in \{-1, +1\}$ the vector σe_i is a descent direction of f with*

$$\cos \angle(\sigma e_i, -\nabla f(x)) > \frac{1}{\sqrt{n}}. \tag{5.21}$$

Proof. For any direction $\pm e_j$ note

$$\cos \angle(\pm e_j, -\nabla f(x)) = \frac{\mp e_j \cdot \nabla f(x)}{\|\nabla f(x)\|} = \frac{\mp \partial_j f(x)}{\left(\sum_{l=1}^n |\partial_l f(x)|^2\right)^{\frac{1}{2}}} \tag{5.22}$$

and take i such that $|\partial_i f(x)| = \max_{1 \leq j \leq n} |\partial_j f(x)|$, σ such that $-\sigma \partial_i f(x) = |\partial_i f(x)|$. Inserting this into (5.22) shows that

$$\cos \angle(\sigma e_i, -\nabla f(x)) > \frac{|\partial_j f(x)|}{\left(\sum_{l=1}^n |\partial_j f(x)|^2\right)^{\frac{1}{2}}} = \frac{1}{\sqrt{n}}. \tag{5.23}$$

□

This can be used to derive bounds on the gradient at all *failed* iterations of the compass search, i.e. iterations in which none of the sample points $x_k \pm h_k e_i$ have a lower function value than x_k. Assume the k-th iteration to be such an iteration and further assume that the objective function is continuously differentiable with Lipschitz continuous gradient.

The theorem above guarantees the existence of a descent direction $\sigma_k e_i$. Since the iteration failed, applying the mean value theorem along the line $t \mapsto x_k + t h_k \sigma_k e_i$ gives

$$0 \leq f(x_k + h_k \sigma_k e_i) - f(x_k) = \sigma_k h_k \partial_i f(x_k + \xi \sigma_k h_k e_i) \tag{5.24}$$

for some $\xi \in [0, h_k]$ and

$$\underbrace{-\sigma_k h_k \partial_i f(x_k)}_{= \sigma_k h_k (-\nabla f(x_k)) \cdot e_i} \leq \sigma_k h_k \left(\partial_i f(x_k + \xi \sigma_k h_k e_i) - \partial_i f(x_k)\right) \tag{5.25}$$

after subtracting $\sigma_k h_k \partial_i f(x_k)$ from both sides. Using equation (5.21) and the Lipschitz continuity of ∇f it follows that

$$\frac{\|\nabla f(x_k)\|}{\sqrt{n}} \leq \|\nabla f\|_{\text{Lip}} \xi h_k \tag{5.26}$$

or

$$\|\nabla f(x_k)\| \leq \sqrt{n} \|\nabla f\|_{\text{Lip}} h_k \tag{5.27}$$

if $h_k \leq 1$.

Since the only variable the bound depends on is the current step width, it is now obvious that the method reaches a stationary point (i.e. converges) if the steps satisfy

$$\lim_{k \to \infty} h_k = 0. \tag{5.28}$$

This can be guaranteed by different assumptions on f, one of which is that the function satisfies

$$|\{x \in \mathbb{R}^n : f(x) \leq f(y)\}| < \infty \text{ for all } y \in \mathbb{R}^n. \tag{5.29}$$

To see this, consider any step length h_k and iterate x_k. All points that can be reached by the algorithm while maintaining the same step length lie on a regular lattice of step width h_k centered on x_k as noted above. Further, any new iterate must have a smaller function value than $f(x_k)$. This limits all possible iterates using the current step size to the intersection

$$\left\{ x_k + \sum_{i=1}^{n} a_i h_k e_i : a \in \mathbb{Z} \right\} \cap \{x \in \mathbb{R}^n : f(x) \leq f(x_k)\}. \tag{5.30}$$

Since the left set is discrete and the right set is bounded the intersection must be finite, which means that the algorithm cannot continue using h_k in perpetuity. Altogether we have shown the following theorem.

Theorem 5.3. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be Lipschitz continuous and satisfy* (5.29)*, then for any $x_0 \in \mathbb{R}^n$ the iterates x_k generated by the compass search fulfill*

$$\lim_{k \to \infty} \|\nabla f(x_k)\|_2 = 0. \tag{5.31}$$

Generating Set Search

Instead of using the search directions $e_1, \ldots, e_n, -e_1, \ldots, -e_n$ as is the case in compass search, the generating set search allows the use of more general search directions that may vary from iteration to iteration. The name of method originates from the basic condition that the set of directions in every iteration must be large enough in the sense of the following definition.

Definition 5.4. *(Generating Set) The set* $D = \{v_1, \ldots, v_{l(D)}\}$ *with* $l(D) \geq n+1$ *is a* generating set *for* $\mathbb{R}^n$*, if for any vector* $x \in \mathbb{R}^n$ *there exist coefficients* $\lambda_1, \ldots, \lambda_{l(D)} \geq 0$ *such that*

$$x = \sum_{i=1}^{l(D)} \lambda_i v_i. \tag{5.32}$$

In other words: D *is a spanning set of* $\mathbb{R}^n$ *using only positive coefficients.*

Each generating set D for itself satisfies a property similar to (5.21) in that for each $x \in \mathbb{R}^n$ there exists a $d \in D$ such that

$$x \cdot d > c(D)\|x\|_2\|d\|_2 > 0. \tag{5.33}$$

Using this one can do the same analysis for the generating set search as for the pattern search to arrive at the bound

$$\|\nabla f(x_k)\| \leq c(D_k)\|\nabla f\|_{\text{Lip}} h_k \tag{5.34}$$

similar to (5.27), where the constant $c(D_k)$ was $c(D) > n^{-\frac{1}{2}}$ for the case where D_k are the coordinate directions. As opposed to the compass search, this gives no control over the gradient, since the constant can vary with each iteration and may well tend to zero if k tends to infinity. As an example of this behavior Kolda et al. (2003) mention the generating sets

$$D_k = \left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} -\frac{1}{k} \\ -1 \end{pmatrix}, \begin{pmatrix} -\frac{1}{k} \\ 1 \end{pmatrix} \right\}, \tag{5.35}$$

where $c(D_k) = 1/\sqrt{1+k^2}$.

The constant can be computed via

$$c(D) = \min_{x \in \mathbb{R}^n} \max_{d \in D} \frac{x \cdot d}{\|x\|_2\|d\|_2} \tag{5.36}$$

and is called *cosine measure* of the set D in the pattern search literature. The gradient bound can be recovered by restricting the choice of generating sets D_k in the algorithm, such that

$$c(D_k) \geq c_{\min} \tag{5.37}$$

for some $c_{\min} > 0$ and all $k > 0$ using the formula for $c(D_k)$.

While the bound on the gradient carries over easily, proving

$$\lim_{k \to \infty} h_k = 0 \tag{5.38}$$

cannot be done the same way as above, since the GSS allows that the set of the directions changes even if the step length h_k stays the same. This means that the set of all iterates reachable while stepping with h_k does not form a simple lattice anymore. Two alternatives described in Kolda et al. (2006) include adding additional constraints on the generating set to recover a lattice structure for the possible iterates, or adding a sufficient decrease condition in step 4 of the algorithm. The latter shall be explained in more detail.

The sufficient decrease condition provides that an iterate $x_k + h_k d_k$ if $f(x_k + h_k d_k) < f(x_k)$, is only accepted if

$$f(x_k + h_k d_k) < f(x_k) - \rho(h_k), \tag{5.39}$$

where ρ is continuous, monotonically increasing and satisfies $\rho(t) = o(t)$ for $t \searrow 0$. This forces the function values of new iterates to decrease by a minimum amount for each fixed h_k

Theorem 5.5. *(Kolda et al., 2006, p. 411) If f is bounded from below, the iterates from the GSS using the sufficient decrease condition* (5.39) *satisfy*

$$\lim_{k \to \infty} h_k = 0. \tag{5.40}$$

Proof. Suppose this is not the case and note that in the variant of GSS considered here, the h_k form a non-increasing sequence. Since this is the case it can be concluded that if (5.40) is not valid, it follows that $h_k \geq \epsilon > 0$ for all sufficiently large $k > k_0$. Since ρ is taken to be monotonically increasing, $h_k \geq \epsilon$ implies $\rho(h_k) \geq \rho(\epsilon) > 0$ and the sufficient decrease condition (5.39) ensures that

$$f(x_{k+1}) \leq f(x_k) - \rho(h_k) \leq f(x_k) - \rho(\epsilon) \tag{5.41}$$

at every iteration k that finds a smaller function value. This means that at every iteration past the k_0-th iteration, either no decrease is achieved and the step width is halved or the decrease achieved is at least $\rho(\epsilon)$. Since the first case can only happen a finite number of times (otherwise h_k would tend to zero) the latter must occur an infinite number of times, but that means $f(x_k) \to -\infty$ as $k \to \infty$ in contradiction to the assumption that f be bounded from below. □

Linear Constraints

The extension of the pattern search method to full linear constraints was first published in Lewis and Torczon (2000) and analyzed in Kolda et al. (2006). A very detailed discussion of the technical details for an implementation was given in Lewis et al. (2007).

From now on assume the problem to be finding $x^* \in \mathbb{R}^n$ such that

$$x^* = \underset{x \in \mathbb{R}^n,\ Ax \leq b}{\operatorname{argmin}} f(x), \tag{5.42}$$

where $A \in \mathbb{R}^{l \times n}$, $b \in \mathbb{R}^l$. Let a_i^T be the i-th row of A and $U = \{x \in \mathbb{R}^n : Ax \leq b\}$ the feasible set.

To turn the generating set search described in algorithm 3 into a method applicable to this problem while achieving the desired full feasibility, the following points will have to be addressed:

- Ensure that the initial point x_0 is feasible with respect to the constraints.
- Choose the search directions D_k and step lengths h_k restrictive enough that $x_k + h_k d$ is feasible for each $d \in D_k$.
- Choose them permissive enough to maintain convergence.

While the first point is straightforward, it is not obvious how to choose the search directions. It can easily be seen though, that the requirement that they generate the whole of $\mathbb{R}^n$ has to be relaxed. If an iterate x_k lies on the boundary of the feasible set, the search directions generating $\mathbb{R}^n$ give infeasible iterates by their very definition. For this reason the search directions have to conform to the geometry of U near x_k in every iterate. To explain how this can be done, we first discuss the geometry of U and introduce some notation from the generating set search literature.

The condition $Ax \leq b$ defines what is called a *convex polytope* in $\mathbb{R}^n$ given by the intersection of half spaces

$$U = \bigcap_{i=1}^{l} \left\{ y \in \mathbb{R}^n : a_i^T y \leq b_i \right\}. \tag{5.43}$$

From this representation it is obvious that U is bounded by plane segments and from now on we will assume that the description $Ax \leq b$ is non-degenerate in the sense that every single inequality defines a facet of U, i.e.

$$\partial U \cap \{y \in \mathbb{R}^n : a_i^T y = b_i\} \neq \emptyset \tag{5.44}$$

for every $i \in \{1, 2, \ldots, l\}$. The vector a_i is then also the normal of U on the facet defined by the intersection above. We also assume that the set U is of full dimension, which means that none of the constraints are actually equality constraints. This makes the presentation clearer, even though the linear constrained pattern search does in fact work with equality constraints as well as degenerate constraints.

The first step in generating feasible search directions is understanding the geometry surrounding the current point. For this let $\epsilon > 0$ be some tolerance parameter, $x \in U$ and define the index set $\mathcal{I}(x, \epsilon) \subset \{1, 2, \ldots, l\}$ by

$$i \in \mathcal{I}(x, \epsilon) \iff \operatorname{dist}\left(x, \{y \in \mathbb{R}^n : a_i^T y \leq b_i\}\right) \leq \epsilon. \tag{5.45}$$

This is a generalization of the notion of *active constraints*, which means those inequalities that are binding at x. Each of the constraints contained in $\mathcal{I}(x, \epsilon)$ represents a facet of U that may be crossed (or a constraint that will be violated) when taking a step of length at least ϵ in the corresponding direction. Using this set we can define the convex cone

$$K(x, \epsilon) = \left\{ \sum_{j \in \mathcal{I}(x,\epsilon)} \lambda_j a_j : \lambda_j \in \mathbb{R}_{\geq 0} \text{ for all } j \in \mathcal{I}(x, \epsilon) \right\} \tag{5.46}$$

which is generated by the normal vectors of all constraints in $\mathcal{I}(x, \epsilon)$ and generalizes the *normal cone* of directions pointing outside Ω to non-boundary points. The generators of its polar cone

$$K^\circ(x, \epsilon) = \{y \in \mathbb{R}^n : y^T x \leq 0 \text{ for all } x \in K(x, \epsilon)\} \tag{5.47}$$

can now be used as search directions. An example of this is illustrated in figure 5.4, which shows a linear constrained region and the cones as well as the search directions generated by different choices of ϵ.

Theorem 5.6. *(Kolda et al., 2006, p 949) Let $x \in U$ and $\epsilon > 0$, then $x + v$ is feasible for every $v \in K^\circ(x, \epsilon)$ with $\|v\|_2 \leq \epsilon$.*

Proof. By definition v satisfies $a_i \cdot v$ for all $i \in \mathcal{I}(x, \epsilon)$, thus $a_i \cdot (x + v) = a_i \cdot x - |a_i \cdot v| \leq b_i$ since x is feasible and thus all constraints in $\mathcal{I}(x, \epsilon)$ are satisfied. All constraints not contained in $\mathcal{I}(x, \epsilon)$ are satisfied since the distance between x and the corresponding hyperplane is larger than ϵ by definition of $\mathcal{I}(x, \epsilon)$. □

The choice of generators cannot be arbitrary and, as was the case in the unconstrained problem, one condition is that the cosine measure of the search directions be bounded from below. To ensure this, it is noted that there are in fact only finitely many cones $K(x, \epsilon)$, since there is only a finite amount of index sets $\mathcal{I}(x, \epsilon)$. If we use the same set of generators for a cone every time it comes up, that condition is trivially satisfied. Depending on the nature of the linear constraints the actual construction of search directions can be complicated, as mentioned a detailed description for this case is given in Lewis et al. (2007).

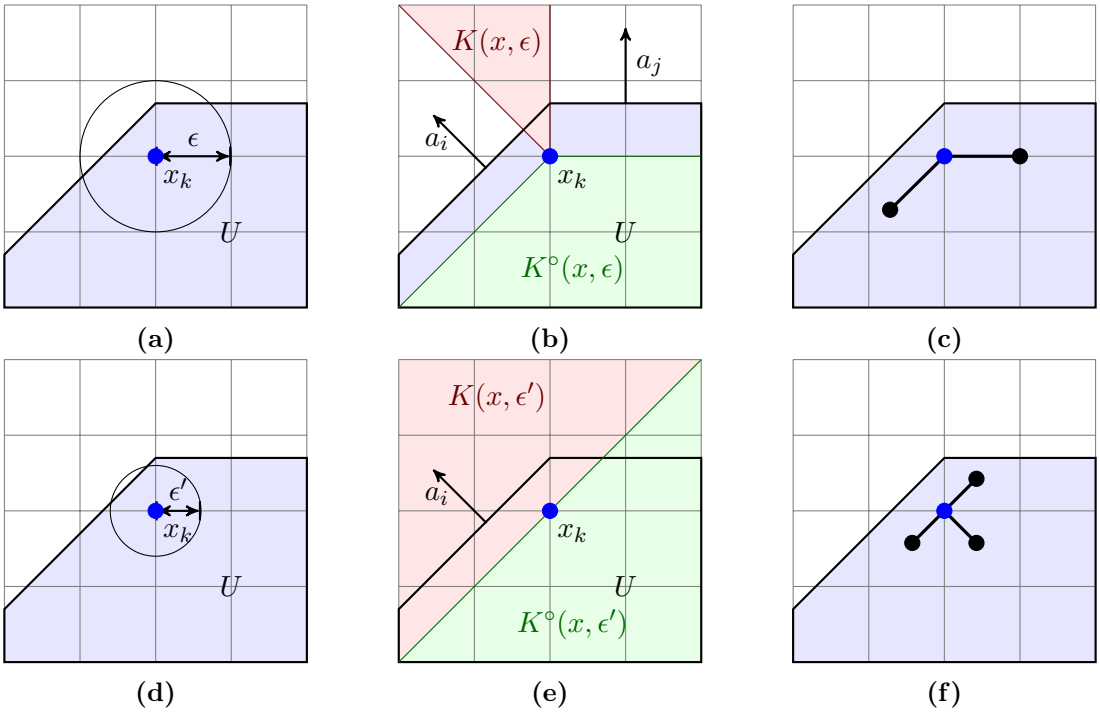

Figure 5.4: Figures (a)-(c) show the cones generated from the (ϵ-)active constraints $a_i \leq b_i$ and $a_j \leq b_j$. Figures (d)-(f) show the same situation for a smaller tolerance ϵ' with only one active constraint.

Parallelity

All function evaluations during one iteration, i.e. the trial points in all search directions, can obviously be performed in parallel. This is considered the synchronous parallel pattern search approach. During simulation based optimization however (or other problems where the duration of one function evaluation is hard to predict) it is desirable to continue freed up computing capacity when some function evaluations terminate earlier than others.

The asynchronous parallel pattern search does this by keeping score of each search direction independently. If an evaluation finishes early, a new trial point in the same direction with reduced step length is generated and queued for evaluation in its stead. The details of the convergence theory and the implementation can be found in Griffin et al. (2008). Special care has to be taken in the linear constrained case, since a reduced step size may open up more search directions than were previously available given the surrounding geometry of the feasible set.

5.3 Hyperbolic Cross Point Optimization

The hyperbolic cross point (HCP) method was first presented by Novak and Ritter (1996), their aim was to develop an adaptive global optimization method based on the non-adaptive sparse grid optimization method. As mentioned above, the non-adaptive method has good worst case estimates that, more importantly, depend on the search space dimension only in a weak way. Using adaptive ideas the HCP method tries to supplement these estimates with good average case runtime.

In this section the basic algorithm as described by Novak and Ritter (1996) as well as some modifications by Hu and Hu (2007) are explained. Then a few possible ways to extend this method to functions with linear constraints and details of the implementation are discussed.

HCP Algorithm

The basic notion is to evaluate the function at sparse grid points of increasing level as a global search approach. While doing this, the algorithm deviates in the order of evaluation and locally increases resolution to introduce an adjustable degree of local search.

For the purpose of this discussion, assume that the objective function is given on the domain

$$U = \prod_{i=1}^{n} [-0.5, 0.5]. \tag{5.48}$$

Consider a *dyadic point* $x \in U$, viz. x is of the form

$$x_i = \pm \sum_{l=1}^{k_i} a_{l,i} 2^{-l} \text{ for } i \in \{1, \ldots, n\} \tag{5.49}$$

where $a_{i,j} \in \{0, 1\}$ and it is assumed that $k_i = 1$. The latter condition assures that this representation of x_i is unique. The number k_i is called the *level* of the coordinate x_i and the level of the point x is then defined by

$$\text{level}(x) = \sum_{i=1}^{n} k_i. \tag{5.50}$$

A second dyadic point $y \in U$ is called a *neighbor of degree* $m > 0$ of x, if there is $j \in \{1, \ldots, n\}$ such that $x_i = y_i$ for $i \neq j$ and

$$y_j = x_j \pm 2^{-\text{level}(x_j)-m}. \tag{5.51}$$

By definition a dyadic point in Ω has at most $2n$ neighbors of any given degree. Any point also has at least n neighbors, since $2^{-\operatorname{level}(x_j)-m} < 2^{-\operatorname{level}(x_j)}$ implies

$$\pm \sum_{l=1}^{\operatorname{level}(x_j)} a_{j,l} 2^{-l} \mp 2^{-\operatorname{level}(x_j)-m} \in [-0.5, 0.5] \tag{5.52}$$

which simply means that between any point x and the point 0 there is a dyadic point with distance to x as in equation (5.51).

Using these preliminaries, the plain HCP algorithm works as follows: The first evaluation of the objective function is always at the point $0 \in \mathbb{R}^n$, which is the center of the search domain. Each further iteration evaluates the function on all neighbors of a certain degree of some previous evaluation point in such a way, that the algorithm maintains a balance between global and local search. Consider the algorithm in a advanced state: Let m denote the amount of completed function evaluations and $X = \{x^1, \ldots, x^m\}$ the points of evaluation. Let $\operatorname{degree}(x^i)$ denote the amount of times that a point was used to generate neighbor points in previous evaluations and let the *rank* of a point be defined by

$$\operatorname{rank}(x^i) = \#\{x^j \in X : f(x^j) < f(x^i)\}, \tag{5.53}$$

i.e. the degree of a point gets bigger the more the algorithm has searched in it's vicinity already, while the rank of a point gets smaller the smaller its function value is in comparison to the other points.

All points are now ranked according to a quality function $g : X \mapsto \mathbb{R}$,

$$g(x_i) = \alpha \ln(\operatorname{level}(x_i) + \operatorname{degree}(x_i)) + (1-\alpha) \ln(\operatorname{rank}(x_i)) \tag{5.54}$$

for some $\alpha \in [0, 1]$, where a lower value of g corresponds to a better point to base the next iteration on. Finding the point x^i with the lowest value of g now strikes a balance between finding a point with low function value (or rank) on the one hand and a region with low coverage (or degree and level) on the other hand. This balance can be adjusted on a continuous scale via the parameter α, where $\alpha = 0$ corresponds to a purely local search, while $\alpha = 1$ is a purely global sparse grid search. The influence of the parameter α on the distribution of point evaluations for one of the test-cases considered in chapter 6 is illustrated in figure 5.5.

The form (5.54) of the quality function is due to Hu and Hu (2007). The original form,

$$g(x^i) = (\operatorname{level}(x^i) + \operatorname{degree}(x^i))^{\alpha} \operatorname{rank}(x^i)^{1-\alpha}, \tag{5.55}$$

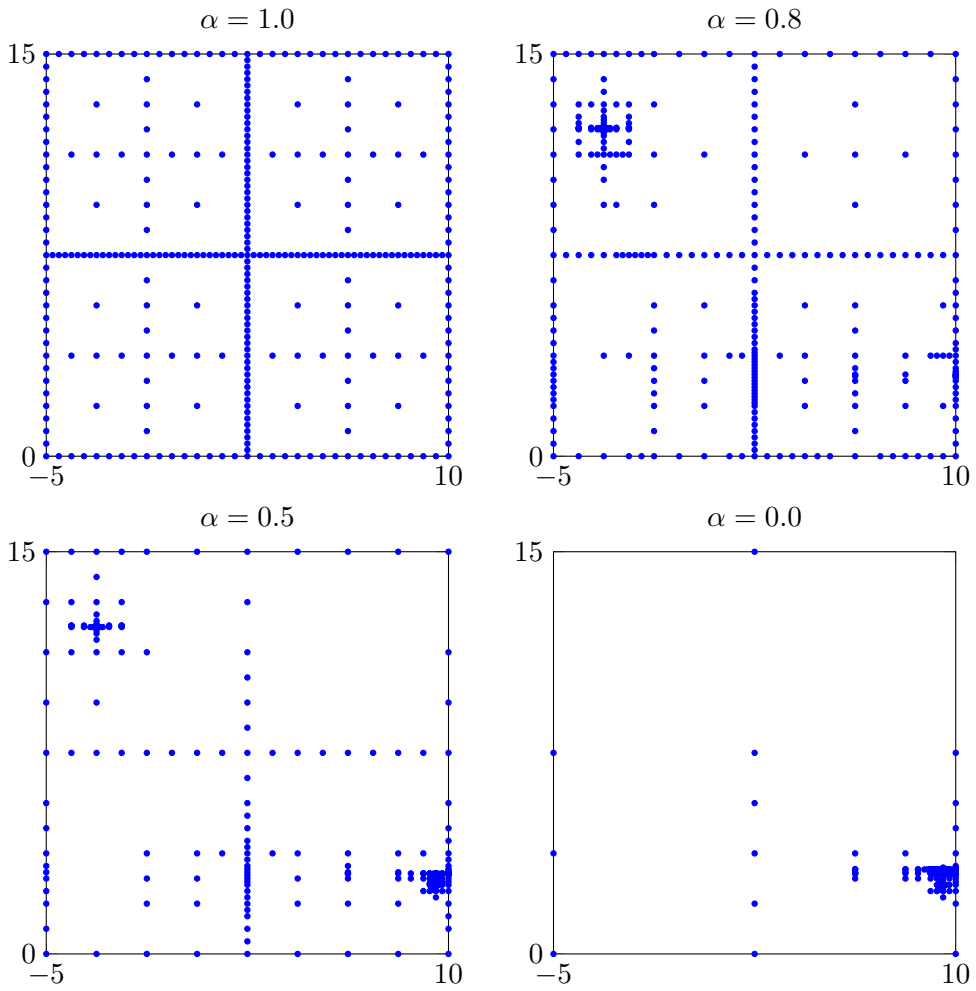

Figure 5.5: Influence of the parameter α on the box constrained HCP method without the subsequent local stage, illustrated using the function **BR** from section 6.2.

defines the same order on X, but each evaluation of a power with floating point exponent $a^b = e^{b \ln a}$ requires nested power series for the exponential and the logarithm. Using (5.54) only one series is computed. According to the authors, this saves a significant amount of computing time when evaluating at a large number of points. For completeness sake this was incorporated into the implementation for this thesis, even though the effect will be negligible for the actual shape optimization problems.

Once a point x^i is found, the objective function is evaluated at all its neighbors of degree degree(x^i) and the algorithm continues with the next iteration.

To ensure good results, the algorithm has two extra parameters that can be used to enforce more globality. First: For some $k_0 > 0$ called *fineness* only points with level$(x^j) \leq k_0$ are considered.

This places an upper bound on the maximum amount of evaluations and more importantly a lower bound on the local resolution of the the algorithm. Further, if the objective value at some point is smaller than the objective at all its neighbors of maximal fineness, that region is considered exhausted and not only are no new neighbors generated from that point, but also no new points in an ϵ box around that point are accepted. That means the base point $\tilde{x}$ of any new iteration must satisfy

$$\|\tilde{x} - y\|_\infty > \frac{\epsilon}{2} \tag{5.56}$$

for each exhausted point y.

Algorithm 4 Hyperbolic cross method

Input: Function $f : [-0.5, 0.5]^n \to \mathbb{R}$ to be minimized, number of evaluations N, parameters $\alpha \in [0, 1]$, fineness $k_0 > 0$, and $\epsilon > 0$.
Output: Estimate (x^*, f^*) of global minimum and minimal function value.

```
Let X = {0}, k = 1
while k < N do
    Find x = argmin_{x∈X} g(x) s.t. ‖x − y‖_∞ > ε/2 for all exhausted points y
    Let x_1, ..., x_m the neighbors of degree degree(x) of x
    Let X = X ∪ {x_1, ..., x_m}
    Set k = k + m.
end while
x* ← argmin_{x∈X} f(x)
f* ← f(x*)
```

Applying the HCP method to linear constrained functions

The original formulation of the HCP method admits only functions defined on rectangular sets, but the functions that arise from the parameterized shape optimization problems considered in this thesis are defined on sets given by linear constraints.

For the purposes of this discussion consider $A \in \mathbb{R}^{l \times n}$, $b \in \mathbb{R}^l$ and the objective function

$$f : U \to \mathbb{R} \tag{5.57}$$

defined on the convex polytope

$$U := \{x \in \mathbb{R}^n : Ax \leq b\}. \tag{5.58}$$

Pintér (1996) (see also Neumaier (2004), p. 24) describes a method which extends linear constrained functions from their original domain to the whole of $\mathbb{R}^n$ through a combination of a penalty term and a projection. Assume that we know a feasible point $x_0 \in U$, for $x \in \mathbb{R}^n$ and

let

$$\bar{x} = \lambda x + (1 - \lambda)x_0 \tag{5.59}$$

with $\lambda \in [0, 1]$ maximal such that $\bar{x} \in U$. Such a λ always exists, since $\lambda = 0$ gives $\bar{x} = x_0 \in U$. An extension of f can then be defined by

$$f_{\text{EX},1} := f(\bar{x}) + \|\bar{x} - x\|. \tag{5.60}$$

This extension can easily be computed, since every equation in $Ax \leq b$ adds one upper bound on λ that is independent of the others: Either $(Ax)_i \leq b$, that means x satisfies the constraints in this coordinate and λ can be taken as 1 as far as this constraint is concerned. Or $(Ax)_i > b$ and thus $(Ax_0)_i \leq (Ax)_i$, which leads to

$$\lambda(Ax)_i + (1 - \lambda)(Ax_0)_i \leq b \iff \lambda \leq \frac{b - (Ax_0)_i}{(Ax)_i - (Ax_0)_i}. \tag{5.61}$$

Using this $\bar{x}$ can be computed in n steps, adding only a small amount of overhead to the actual function evaluation. According to Neumaier (2004) this extension is (Lipschitz-) continuous if the original function f was (Lipschitz-) continuous.

This idea can be taken a step further by considering the Euclidean projection $P : \mathbb{R}^n \to U$ which is defined by

$$P(x) = \operatorname*{argmin}_{y \in U} \|x - y\|. \tag{5.62}$$

Using this one can define a second extension by

$$f_{\text{EX},2} := f(P(x)) + \|P(x) - x\|. \tag{5.63}$$

Since the Euclidean projection is known to be Lipschitz-continuous, the same continuity results as for the extension above hold here too. Rather than the simple procedure for computing (5.60) though, this requires the computation of the complete projection. Rewriting (5.62) as

$$\begin{aligned} P(x) &= \operatorname*{argmax}_{y \in U} \|x - y\| && (5.64)\\ &= \operatorname*{argmax}_{y \in U} x^T x - 2x^T y + y^T y && (5.65)\\ &= \operatorname*{argmax}_{y \in \mathbb{R}^n} y^T y - 2x^T y \quad \text{s.t. } Ay \leq b && (5.66) \end{aligned}$$

this can be seen to be equivalent to solving a quadratic program with linear constraints. In face of the simulation based function evaluations in this thesis the extra overhead is acceptable, though significant in the case of simple objective functions.

To illustrate the nature of the extension consider the function

$$f(x) = \sqrt{9x_1^2 + (3x_2 - 5)^2} - 5\exp\left(\frac{-1}{(3x_1 + 2)^2 + (3x_2 - 1)^2 + 1}\right) \tag{5.67}$$

with constraints

$$\begin{aligned} 3x_1 &\leq 4 \\ -3x_2 &\leq 2 \\ 3x_2 &\leq 5 \\ -3x_1 - 3x_2 &\leq 2 \\ -3x_1 + 3x_2 &\leq 5. \end{aligned} \tag{5.68}$$

This function is taken from Griffin et al. (2008) and also used as a computational model problem in section 6.2. Figure 5.7 shows both extensions discussed above with the original domain U outlined in black. The extension proposed by Pintér is anchored on the point $x_0 = 0$ and it can clearly be seen that the slope of the extension outside of U is skewed towards that point, while the extension using the Euclidean projection is oriented on the corresponding nearest facet of the polytope.

Figure 5.6 shows the distribution of point evaluations of the HCP method applied to f extended by formula 5.63 as well as the influence of α in this case. For higher α a certain amount of clustering is visible on the slanted sides of the feasible region, stemming from the the evaluation points are projected from a rectangle. This will be less distinct in the shape optimization applications, since the amount of evaluation points is smaller while the dimension (and thus the surface of the polytope) is higher. This can be seen in figure 6.5 of chapter 6, that shows a similar illustration for a three dimensional problem.

Terminal local search step

The reference implementation supports a concluding local search at the end of the HCP method, this is done by setting $\alpha = 0.0$ and removing the restrictions on fineness and neighborhood for the remaining point evaluations. What results is in effect a compass search, as it was discussed in the previous section, with the modification that the step length is kept independently for each axis and halved even during successful iterations. In practice this is not satisfactory as a purely local search method, since the step size decreases when a new smaller point is found, instead of being decreased when no smaller point is found (cf. the discussion of the convergence properties of the APPS method in section 5.2). For this reason the local search step is replaced by the APPS method, which is discussed further in section 5.2. Of special note is the fact that both

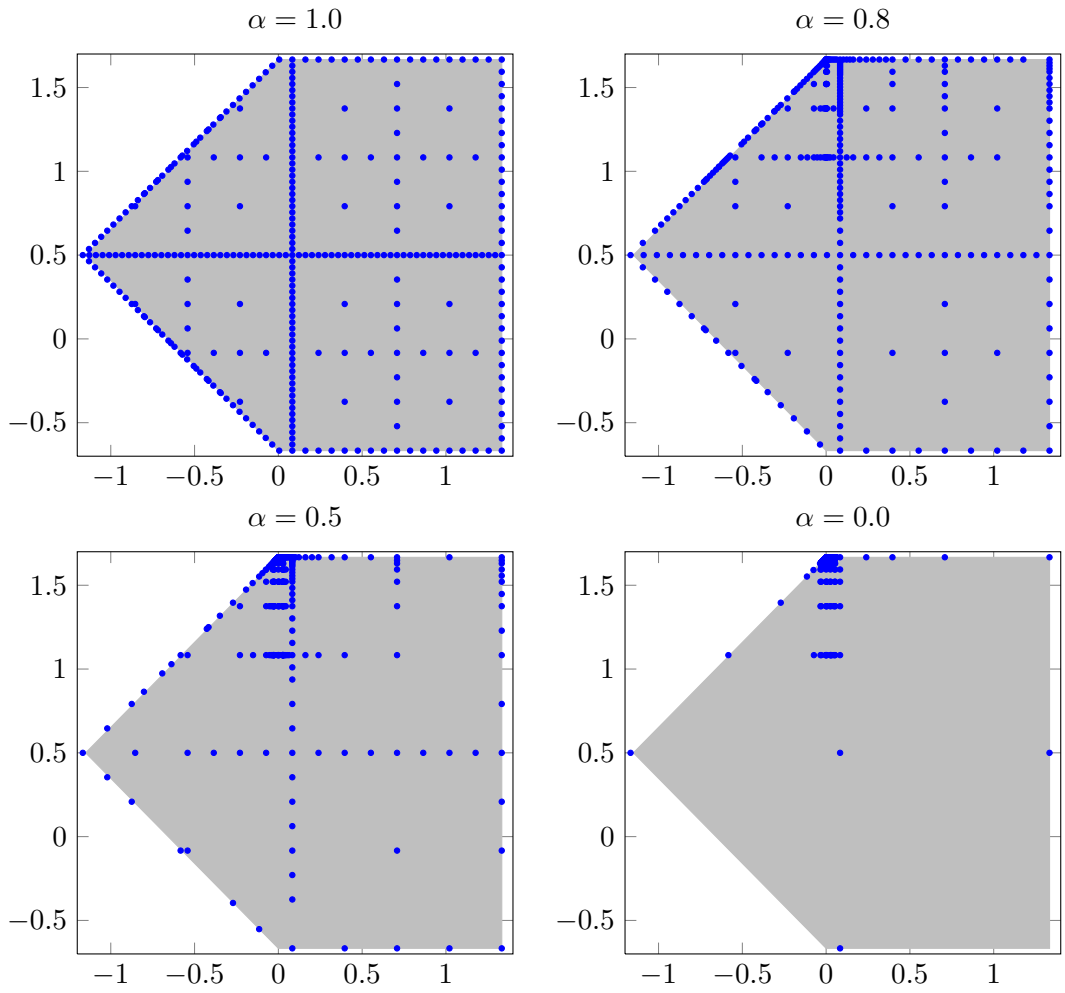

Figure 5.6: Influence of the parameter α on the linear constrained HCP method without the subsequent local stage, illustrated using the function (5.67) extended by (5.63). The original feasible region is shaded in grey.

extensions discussed in the previous section preserve the requirements for convergence posed by the APPS.

Implementation

Although, as mentioned, Novak and Ritter (1996) provide their own implementation of the algorithm, it is neither parallel nor easily extensible. For this reason a new implementation was written as an extension to the *Hybrid Optimization Parallel Search Package* (HOPSPACK), see Plantenga (2009). HOPSPACK is the C++ software framework which the reference implementation of the APPS introduced in section 5.2 is based on.

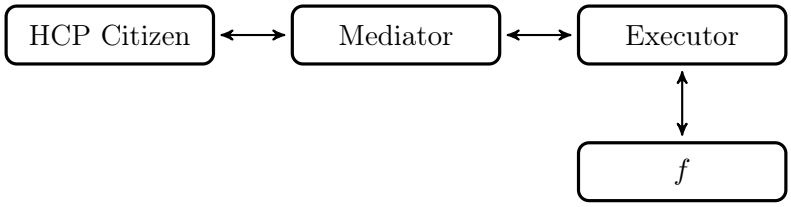

Figure 5.8: Computational model of the HOPSPACK framework. The citizen class implementing the HCP algorithm requests function evaluations from the mediator, which in turn hands them to the executor. Only the executor does any function calls and separates the algorithm from execution details.

It was developed with, possibly simulation based, robust black box optimization in mind and handles function evaluations completely by system calls, while having the capacity to work through failed calls. Its computational model (cf. figure 5.8) is such that each optimization algorithm is represented by a *Citizen* class that exchanges point evaluation requests and evaluation results with a *Mediator* class. The Mediator in turn hand all point requests to a *Executor* class, which has serial and parallel (MPI, MT) implementations and performs the system calls to the binary implementing the actual function f. This way the programming details involved with (parallel) evaluation of the error prone simulations is abstracted away from the actual algorithm.

The result is a parallel (as all evaluations in a single iteration may be performed in parallel) and easily configurable implementation. It handles failed evaluations by disregarding the point for future evaluations.

By default, if linear constraints are given the extension (5.63) is used and the Euclidean projection computed with an active set method. This method has certain requirements on the constraint matrix, if at any time the projection can not be computed the extension by Pintér is used for the immediate point and all further iterations for consistency.

The algorithm may be supplied with a bound on the global evaluations. If this is done, an APPS Citizen is spawned after the global evaluations are used up and run until local convergence or the overall amount of allowed evaluations (configured in the Mediator) is used up. This is of particular usefulness for problems with linear constraints, since this method adjusts the local boundary.

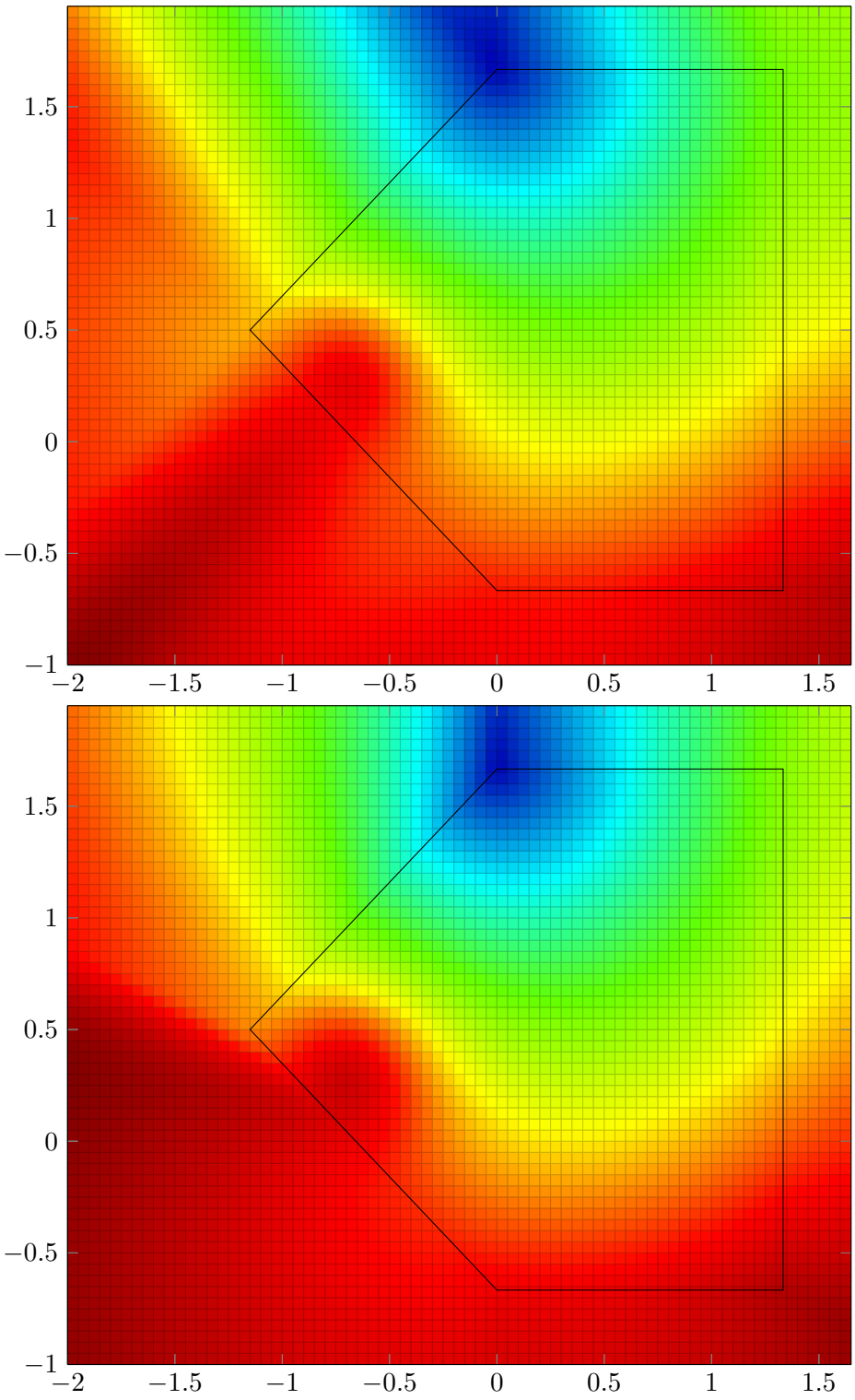

Figure 5.7: The black outline shows the region defined by the constraints (5.68) with the function (5.67) depicted inside. Outside the plot shows the extension by full projection (5.63) in the upper image and by the Pinter method (5.60) in the bottom image.

6 Numerical Experiments

6.1 Navier-Stokes Solver

To validate the Naiver-Stokes simulation, it was compared against a known analytical solution to the two-dimensional Navier-Stokes equations. This solution is the so called *Taylor-Green vortex* described in Chorin (1968).

It is given on the domain $\Omega = [-1, 1]^2$, the kinematic viscosity is $\nu = 0.01$ and underlies periodic boundary conditions

$$\begin{aligned} v(0, y) &= v(1, y) \\ v(x, 0) &= v(x, 1) \end{aligned} \tag{6.1}$$

for $x, y \in [0, 1]$. Velocity and pressure for $t \geq 0$ are given by the functions

$$\begin{aligned} v(x, y, t) &= \begin{pmatrix} \cos(\pi x) \sin(\pi y) e^{-2\nu\pi^2 t} \\ \cos(\pi y) \sin(\pi x) e^{-2\nu\pi^2 t} \end{pmatrix} \\ p(x, y, t) &= -\frac{1}{4}(\cos(2\pi x) + cos(2\pi y)) e^{-4\nu\pi^2 t} \end{aligned} \tag{6.2}$$

which also define the initial values for the simulation at $t = 0$. The time interval under consideration was taken to be $[0, 5]$, though the Taylor-Green vortex is defined for all $t \geq 0$. Figure 6.1 shows an illustration of the energy distribution and streamlines at time $t = 0$. One can a nearly rectangular vortex centered on $(0, 0)$ that is periodically copied in all directions. From the exact solution it can be seen that for times $t \geq 0$ the shape of the vortex stays but the velocities are dampened by a factor $e^{-2\nu\pi^2 t}$ with exponential decay in t.

The finite element approximation for different mesh parameters h was compared to the exact solution given by the formula above using approximations of the relative errors

$$\frac{\|v - v_h\|^2_{\mathrm{L}^2([0,T],\mathrm{L}^2(\Omega))}}{\|v\|^2_{\mathrm{L}^2([0,T],\mathrm{L}^2(\Omega))}} = \frac{\int_0^T \int_\Omega \|v - v_h\|_2 \, \mathrm{d}x\mathrm{d}t}{\int_0^T \int_\Omega \|v\|_2 \, \mathrm{d}x\mathrm{d}t} \tag{6.3}$$

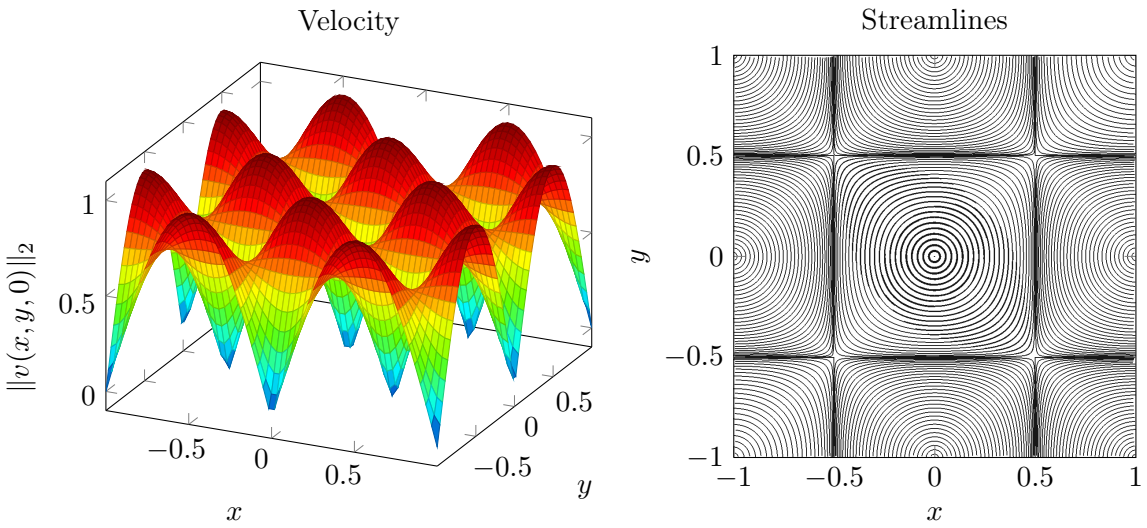

Figure 6.1: Illustration of the Taylor-Green vortex at time $t = 0$. The left image shows the magnitude of the velocity vectors, the right image shows the stream lines. In the center lies a nearly rectangular vortex that is repeated periodically in all directions. The fluid is fastest at the long outer edges of each vortex.

and

$$\frac{\|v - v_h\|^2_{\mathrm{L}^2([0,T],\mathrm{H}^1_0(\Omega))}}{\|v\|^2_{\mathrm{L}^2([0,T],\mathrm{H}^1_0(\Omega))}} = \frac{\int_0^T \int_\Omega \|\nabla v - \nabla v_h\|_2 \,\mathrm{d}x\mathrm{d}t}{\int_0^T \int_\Omega \|\nabla v\|_2 \,\mathrm{d}x\mathrm{d}t} \tag{6.4}$$

This was done using the interpolation $\mathcal{I}_h v$ of v onto the finite element space V_h, i.e. the finite element function that interpolates the values of v at the mesh nodes and the derivatives of v at the edge midpoints. This interpolation was entered into the formula above, which allows for the calculation of the integral over Ω using the mass- and stiffness matrix respectively (c.f. equation (2.42)). The time integral was approximated using a general purpose numerical quadrature routine from the QUADPACK library.

The results are shown in figure 6.2. One can see that the accumulated L^2-error converges with roughly first order, while the accumulated H^1_0-error converges with $N^{-1/2}$. This is particularly important since this is also the objective function in the shape optimization examples.

6.2 Optimization Problems

The HCP implementation was tested on a variety of classical benchmark functions for global optimization methods also used in the original paper by Novak and Ritter (1996), all of which are defined on simple box constrained regions. It was also tested with several functions featuring

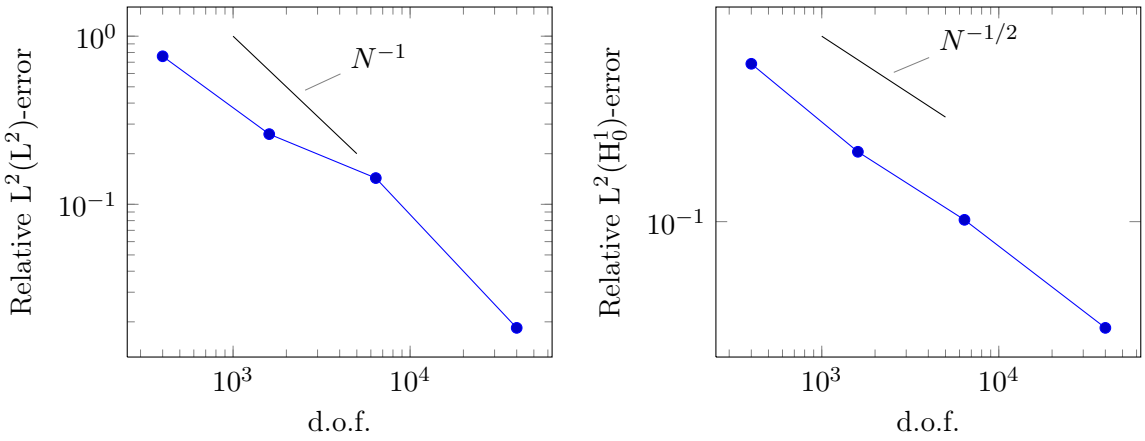

Figure 6.2: Measurements from the Taylor-Green vortex computed using the FEniCS bases Navier-Stokes solver. The error is drawn against the degrees of freedom of the triangulation N. The accumulated L^2 error seems to converge with order one, while the accumulated H_0^1-error converges with order one half.

linear constraints. This made it possible to validate the implementation against known results as well as evaluate the changes made to the original method.

Box Constrained

All test functions are featured in Törn and Žilinskas (1989) and the presentation follows the notation therein, the original sources are given below. For comparison the non-adaptive full-grid, sparse-grid and quasi-Monte Carlo methods as described in section 5 were also applied to each function. The following functions were used:

- The function **BR** from Branin and Hoo (1972):

$$f_{\mathrm{BR}}(x) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos(x_1) + 10, \tag{6.5}$$

  where

$$(x_1, x_2) \in [-5, 10] \times [0, 15]. \tag{6.6}$$

  Its global minimum 0.398 is attained at the points $(-3.142, 12.275)$, $(3.142, 2.275)$ and $(9.425, 2.425)$.

- The *Shekel functions* **S5** and **S10** from Dixon and Szegö (1978):

$$f_{\mathrm{S}m}(x) = -\sum_{i=1}^{m}\left(\|x - A(i)\|_2^2 + c_i\right)^{-1}. \tag{6.7}$$

The parameters $A(i)$ and c_i are given in table 6.1, for $x \in [0,10]^4$ it attains the approximate local minimums $-\frac{1}{c_i}$ near the points $A(i)$, in particular the global minimum is attained at $(4.0, 4.0, 4.0, 4.0)$.

Table 6.1: Parameters for the Shekel function.

i	$A(i)$	c_i
1	$(4.04.04.04.0)^T$	0.1
2	$(1.01.01.01.0)^T$	0.2
2	$(8.08.08.08.0)^T$	0.2
3	$(6.06.06.06.0)^T$	0.4
4	$(3.07.03.07.0)^T$	0.4
5	$(2.09.02.09.0)^T$	0.6
6	$(5.05.03.03.0)^T$	0.3
7	$(8.01.08.01.0)^T$	0.7
8	$(6.02.06.02.0)^T$	0.5
9	$(7.03.67.03.6)^T$	0.5

- The *Hartman function* **H6** from Dixon and Szegö (1978):

$$f_{\mathrm{H}m}(x) = -\sum_{i=1}^{4} c_i \exp\left(-\sum_{j=1}^{m} \alpha_{ij}(x_j - p_{ij})^2\right) \tag{6.8}$$

for $x \in [0,1]^6$, with $\alpha, p \in \mathbb{R}^{4\times 6}$ and $c \in \mathbb{R}^4$ given by

$$\alpha = \begin{pmatrix} 10 & 3 & 17 & 3.5 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{pmatrix}, \; c = \begin{pmatrix} 1 \\ 1.2 \\ 3 \\ 3.2 \end{pmatrix}, \tag{6.9}$$

$$p = \begin{pmatrix} 0.1312 & 0.1696 & 0.5569 & 0.0124 & 0.8283 & 0.5886 \\ 0.2329 & 0.4135 & 0.8307 & 0.3736 & 0.1004 & 0.9991 \\ 0.2348 & 0.1451 & 0.3522 & 0.2883 & 0.3047 & 0.6650 \\ 0.4047 & 0.8828 & 0.8732 & 0.5743 & 0.1091 & 0.0381 \end{pmatrix}. \tag{6.10}$$

The minimum -3.32 is attained at the point $(0.201, 0.150, 0.477, 0.275, 0.311, 0.657)^T$.

Figure 6.3 shows the relative error with respect to the minimal function value for all discussed non-adaptive methods and the global stage of the HCP implementation. For this consider x_N to be the point generated by the minimization method in question after investing N function evaluations towards solving the global minimization problem for f, then the relative error with

respect to f is defined by

$$e_{\mathrm{rel},f}(N) = \frac{|f_{\min} - f(x_N)|}{|f_{\min}|}, \tag{6.11}$$

where $f_{\min}$ is the minimum of f over the surveyed region.

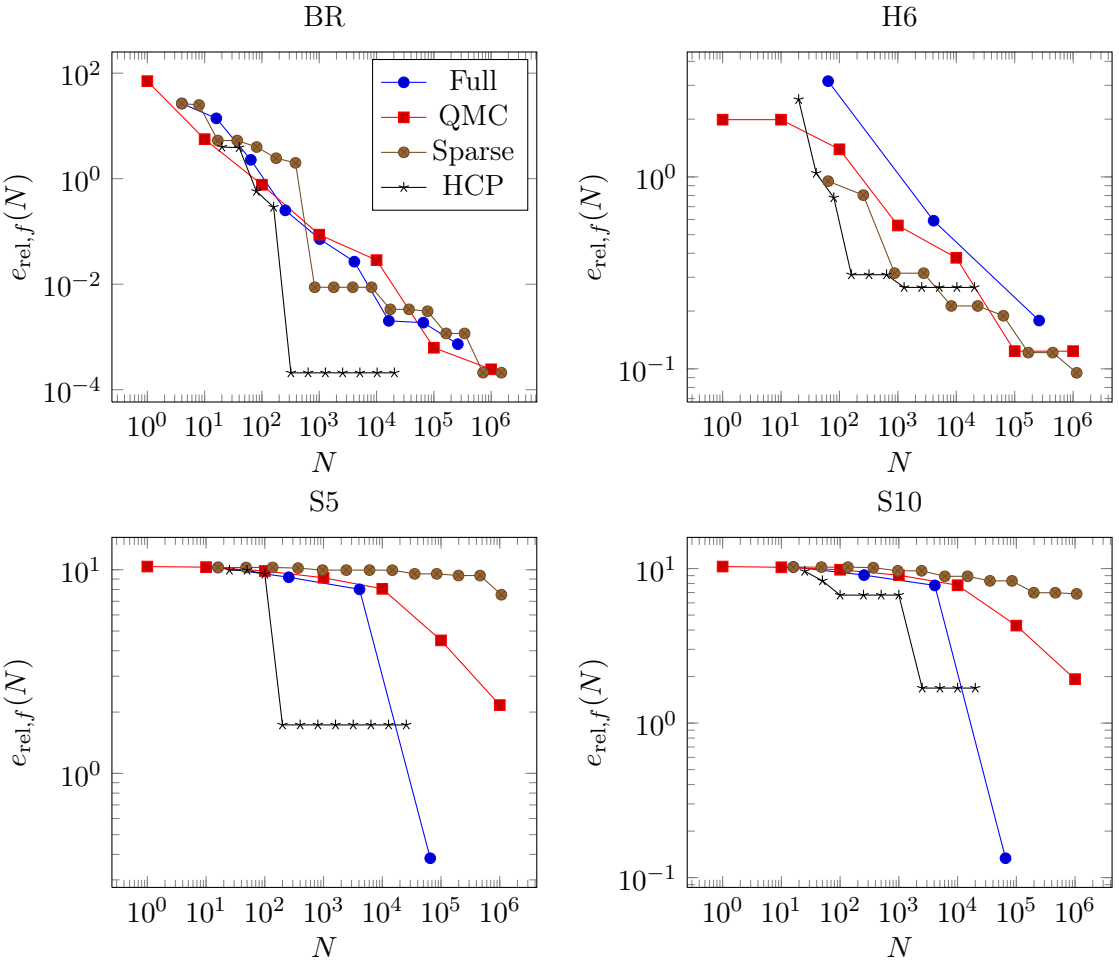

Figure 6.3: Deviation of the approximation calculated by each method from the actual minimal function value. The values for the HCP method shown here are purely from the global step of the implementation, without any subsequent APPS iterations.

Shown in figure 6.4 is the absolute spatial error of the different non-adaptive methods and the global stage of the HCP implementation. Considering x_N, f as above let X be the set of all points where f attains its minimum, then the absolute error with respect to x is defined by

$$e_{\mathrm{abs},x}(N) = \min_{x \in X} \|x - x_N\|. \tag{6.12}$$

This means $e_{\mathrm{abs},x}(N)$ denotes the distance between the calculated approximation and nearest minimal point of f. The reason for using the absolute error in this case is that the nearest minimal point may change from one data point to the next, which would mean the normalizing factor would too.

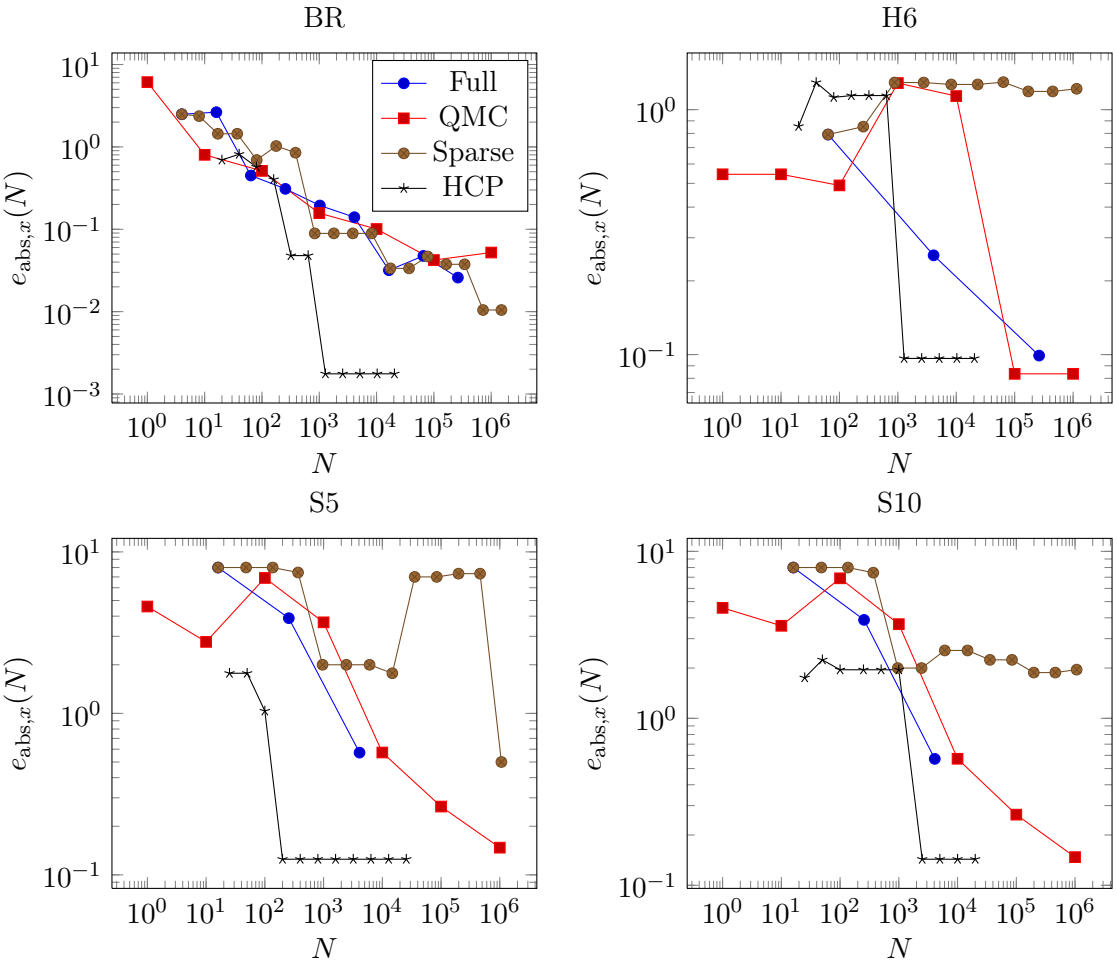

Figure 6.4: Deviation of the approximation calculated by each method from the corresponding nearest minimizing point. Since the approximation is determined by function value only, the error in this sense is not necessarily monotone in N, as is observed in the plots. The values for the HCP method shown here are purely from the global step of the implementation, without any subsequent APPS iterations.

One can see in figure 6.3 that all the non-adaptive methods show very similar behavior with respect to the minimal function value found, at least for the first few thousand evaluations. In case of the functions BR and H6 a clear convergent behavior is visible which, against expectations, is the same for all non-adaptive methods even though theory suggests that the full grid method should struggle for the 6-dimensional Hartman function. Both Shekel functions are designed in

such a way that the function itself is roughly constant away from the minima at the points $A(i)$, which is clearly reflected in the behavior seen in the plot. None of the non-adaptive methods show any significant improvement until about $N \approx 10^4$, where both the full-grid and QMC method are near a local minimum for the first time. In all cases considered here the HCP method starts to outperform the non-adaptive method at roughly the $100-1000$ evaluation mark but stagnates on that level for some time.

The error with respect to the location of the minimum depicted in figure 6.4 shows similar results in case of the function BR. In all other cases the quality of the approximation varies wildly for the non-adaptive methods, even in case of the Shekel function where all methods stagnated for a long time when considering function values. The HCP method again outperforms the other methods from an early stage, with the exception of the Harman function where it is the worst up until 1000 evaluations. Curiously the HCP method does show a monotonically decreasing error in these test cases and is not misled by local minima.

Function	n	$f(x^*)$	x^*
BR	220	0.398	$(9.422, 2.473)$
S5	121	-8.806	$(4.063, 4.063, 4.063, 4.063)$
S10	439	-9.190	$(4.063, 4.063, 4.063, 4.063)$
H6	580	-3.035	$(0.125, 0.125, 0.500, 0.250, 0.375, 0.625)$

Table 6.2: Results from Novak and Ritter (1996).

Function	n	$f(x^*)$	x^*
BR	92	0.3979	$(9.422, 2.479)$
S5	169	-10.15	$(4.000, 4.000, 4.000, 4.000)$
S10	225	-10.54	$(4.000, 4.000, 4.000, 4.000)$
H6	485	-3.202	$(0.406, 0.885, 0.843, 0.572, 0.140, 0.041)$

Table 6.3: About 12 iterations of HCP with subsequent APPS until convergence.

Tables 6.2 and 6.3 show the results given in the paper of Novak and Ritter (1996) and the results of the HCP method with subsequent APPS iterations. Replacing the HCP local stage with the APPS method gives improved results in all cases over the reference results with a similar or lower amount of function evaluations, though it is not made clear in Novak and Ritter (1996) what their termination criterion was. The number of 12 iterations was chosen as a number that stands at the lower end of the range where the global HCP stage shows best performance in the graphs above while allowing for more evaluations in higher dimensions. In all cases the HCP plus APPS combination attains very nearly the minimum if not the minimal point in a number of iterations that vastly outperforms the results in figures 6.3 and 6.4. This probably stems from the fact that the HCP best-point is near an actual minimizers early on in all examples, as seen in figure 6.4.

Linear Constrained

The extension to linear constraints described in section 5.3 was tested using three linear constrained functions from the literature. In this case the comparison with the non-adaptive methods was dispensed with and the full HCP implementation was applied using similar parameters as in the shape optimization problems. The following functions were considered:

- The function f_{L1} from Griffin et al. (2008). It is defined by

$$f_{\mathrm{L1}}(x) = \sqrt{9x_1^2 + (3x_2 - 5)^2} - 5\exp\left(\frac{-1}{(3x_1+2)^2 + (3x_2-1)^2 + 1}\right) \tag{6.13}$$

  with linear constraints

$$\begin{aligned} 3x_1 &\leq 4, \\ -3x_2 &\leq 2, \\ 3x_2 &\leq 5, \\ -3x_1 - 3x_2 &\leq 2, \\ -3x_1 + 3x_2 &\leq 5. \end{aligned} \tag{6.14}$$

  The minimum of about -4.767 is attained at the point $(0, 5/4)$.

- The function f_{L2} from Ji et al. (2007)

$$\begin{aligned} f_{\mathrm{L2}}(x) = &- \frac{3x_1 + 5x_2 + 3x_3 + 50}{3x_1 + 4x_3 + 5x_3 + 50} - \frac{3x_1 + 4x_2 + 50}{4x_1 + 3x_2 + 2x_3 + 50} \\ &- \frac{4x_1 + 2x_2 + 4x_3 + 50}{5x_1 + 4x_2 + 3x_3 + 50} \end{aligned} \tag{6.15}$$

  with linear constraints

$$\begin{aligned} 6x_1 + 4x_2 + 4x_3 &\leq 10, \\ 10x_1 + 3x_2 + 8x_3 &\leq 10, \\ x_1, x_2, x_3 &\geq 0. \end{aligned} \tag{6.16}$$

  They give an approximate minimum of -3.000042 at $(0, 0.33329, 0)$.

- The function f_{L3}, also from Ji et al. (2007), defined by

$$\begin{aligned} f_{\mathrm{L3}}(x) = &- \frac{4x_1 + 3x_2 + 3x_3 + 50}{3x_1 + 3x_3 + 5x_3 + 50} - \frac{3x_1 + 4x_3 + 50}{4x_1 + 5x_2 + 5x_3 + 50} \\ &- \frac{x_1 + 2x_2 + 5x_3 + 50}{x_1 + 5x_2 + 5x_3 + 50} - \frac{x_1 + 2x_2 + 4x_3 + 50}{5x_2 + 4x_3 + 50} \end{aligned} \tag{6.17}$$

with constraints

$$\begin{aligned} 2x_1 + x_2 + 5x_3 &\leq 10, \\ x_1 + 6x_2 + 3x_3 &\leq 10, \\ 5x_1 + 9x_2 + 2x_3 &\leq 10, \\ 9x_1 + 7x_2 + 3x_3 &\leq 10, \\ x_1, x_2, x_3 &\geq 0. \end{aligned} \tag{6.18}$$

The approximate minimizer given in the source is $(1.0715, 0, 0)$ with a value of -4.087412.

Function	n	$f(x^*)$	$e_{\text{rel},f}$	x^*	$e_{\text{abs},x}$
L1	96	-4.767	$1.22 \cdot 10^{-5}$	$(1.27 \cdot 10^{-6}, 1.667)$	$2.03 \cdot 10^{-5}$
L2	122	-3.003	-	$(2.397 \cdot 10^{-03}, 3.325, 7.772 \cdot 10^{-16})$	-
L3	122	-4.09	-	$(1.1, 0, 0)$	-

Table 6.4: Results for the linear constrained test functions. Functions L2 and L3 have no anlytical solutions available but the values found agree with those found in Ji et al. (2007).

Solutions for 12 iterations of the HCP method with subsequent APPS iterations are displayed in table 6.4. For the first test function a small relative error of the order 10^{-5} is achieved. The other two functions have no analytical solutions available but the numbers found agree with the ones given in the source. Figure 6.5 shows the distribution of evaluation points for the function L3 and different values of α, the feasible polytope is outlined in black.

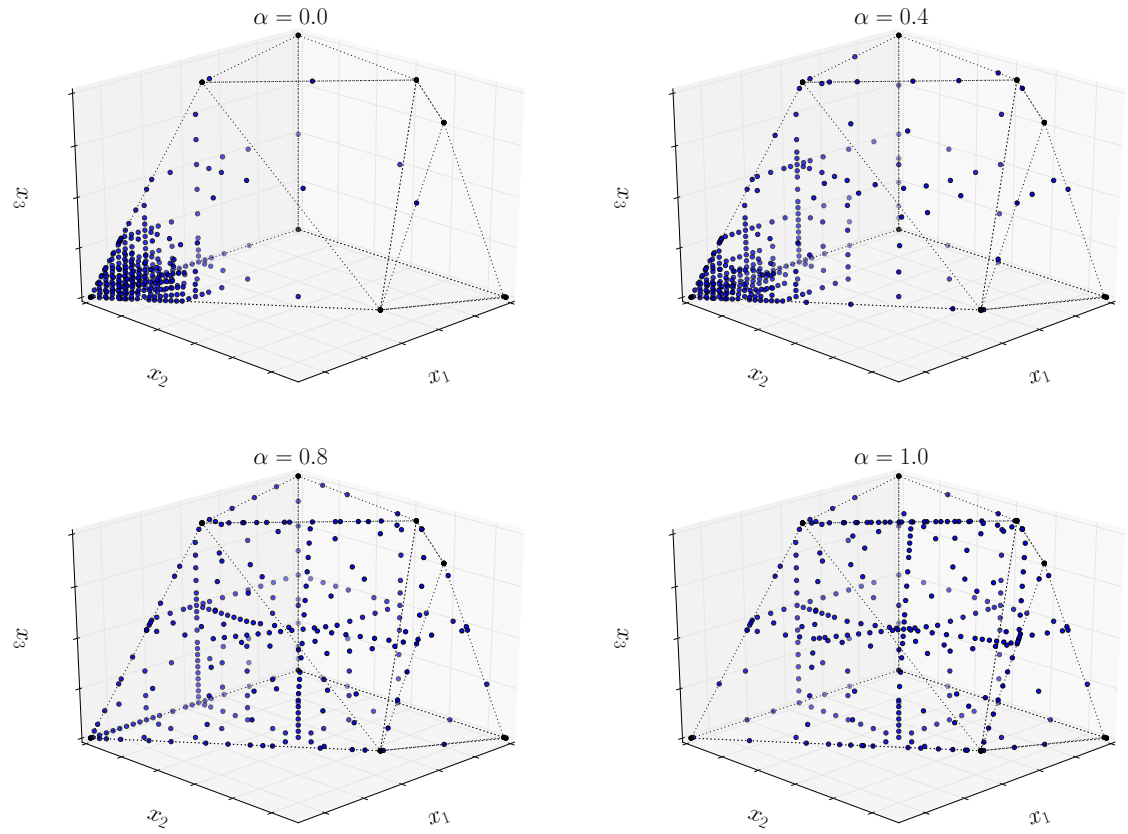

Figure 6.5: Evaluation points of the HCP method for the linearly constraind function f_{L3} after extension by a full Euclidean projection for different values of α. The black points are the vertices of the polytope generated by the linear constraints, the edges are drawn with dashed black lines.

6.3 Shape Optimization Model Problems

In this section the complete black-box solution approach is applied to several application inspired shape optimization problems involving fluid flows. Each problem is of the following general form:

$$\text{minimize} \quad F(c) = \fint_{[0,T]} \int_{\Omega\backslash\omega(c)} |\nabla u_h|^2 \, \mathrm{d}x\mathrm{d}t \tag{6.19}$$

s.t. u_h is the finite element approximation to the Navier-Stokes constraint

$$\begin{aligned} u_t + (u \cdot \nabla)u &= \nu\Delta u - \nabla p \text{ in } [0,T] \times \Omega\backslash\omega(c) \\ \operatorname{div} u &= 0 \text{ in } [0,T] \times \Omega\backslash\omega(c), \\ u &= u_{\text{in}} \text{ on } \Gamma_{\text{in}}, \\ \frac{1}{\mathrm{Re}}\partial_\nu u - p\nu &= 0 \text{ on } \Gamma_{\text{out}}, \\ u &= 0 \text{ on } \Gamma_0, \\ u &= 0 \text{ on } \partial\omega(c) \end{aligned} \tag{6.20}$$

with additional constraints on the parameterization described for each problem separately. Here $c \mapsto \omega(c)$ is a parameterization of the geometry as described in section 4.

6.3.1 Flow Past Obstacle

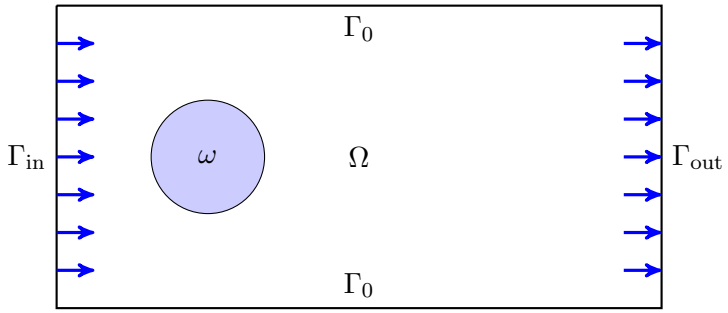

Figure 6.6: Sketch of obstacle flow problem. The geometry ω that is modified during the optimization is shown in blue, fluid flows through Ω from left to right around the obstacle.

Inspired by the flow around objects in rivers such as bridge piers, the first example is essentially the same as the problem considered in section 2.3. It is illustrated again in figure 6.6 and consists of a channel flow around an obstacle ω, which is the variable of the minimization. The inflow is $u_{\text{in}}^T = (1, 0)$ and the time interval is $[0, 5]$. Lacking initial values the simulation is instead run until the flow has had time to pass through the whole domain twice before measurements are taken. The obstacle ω is considered to be symmetric around the axis $y = 4.0$ and parameterized

as a curve above the symmetry axis. The problem aims to emulate computations arising when considering supporting columns of structures such as bridge pears, mooring places or deep sea constructions. In such cases it is of interest to minimize the degree to which the flow is disturbed to reduce the load on the obstacle.

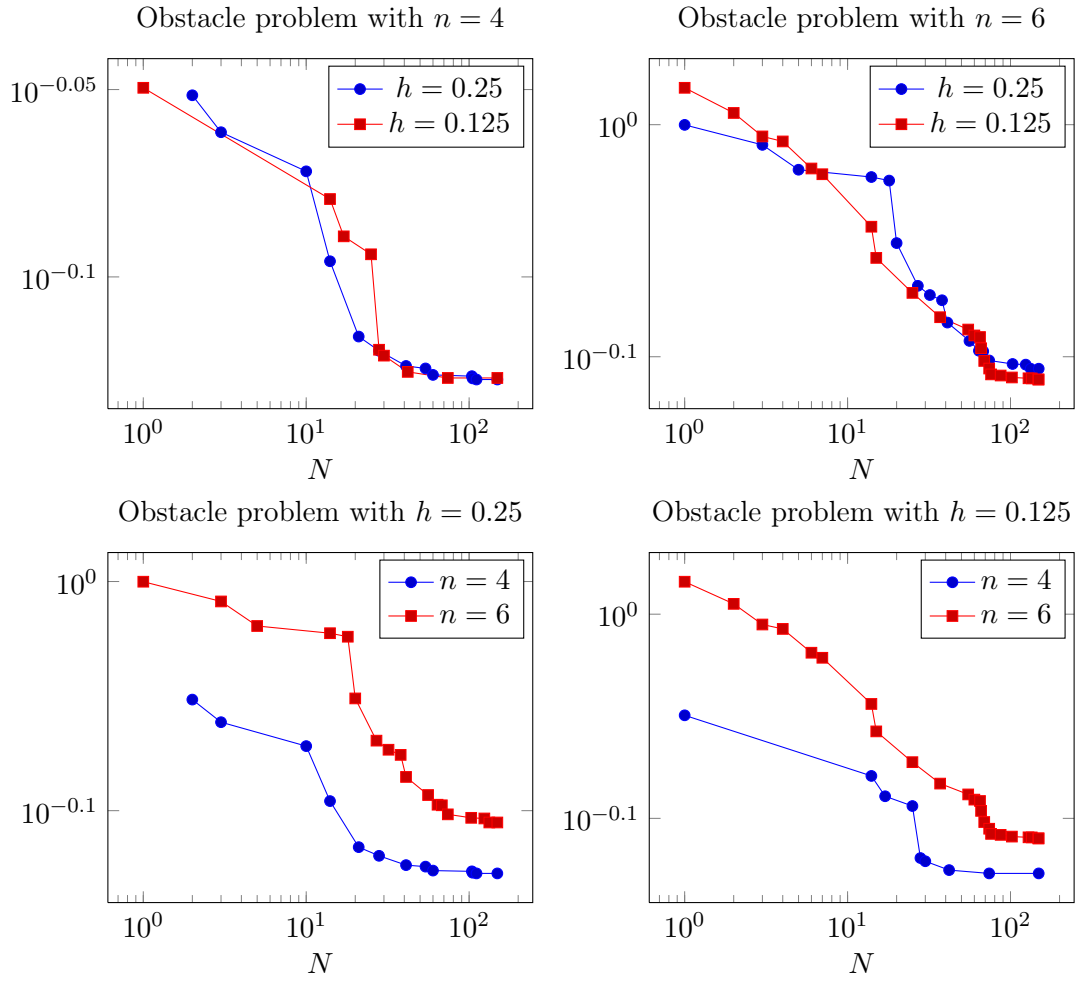

Figure 6.7: Relative improvement during the course of the optimization where N is the number of function evaluations invested. The graphs compare the results first for a fixed amount of degrees of freedom in the parameterization, then for a fixed mesh width. All values are normalized by the dissipation for a circular geometry on that mesh width.

Figure 6.8 shows a simple circular obstacle compared to the results of optimizations with $n = 4$ and $n = 6$ degrees of freedom in the parameterization and $h = 0.125$. The pictures show the velocity magnitude and streamlines of the flow at time $t = 5$. It can be seen that the unoptimized obstacle causes an unsteady flow shedding vortices in the obstacles wake while the flow around the optimized obstacles is near stationary with very little separation. Since the result with

$n = 4$ is slightly thinner the reattachment of the flow happens a bit earlier than for the obstacle computed with $n = 6$.

The progress of the optimization is shown in figure 6.7. All values are normalized by the dissipation of the circular geometry at that mesh width, i.e. show the improvement upon that geometry when compared using the same FEM parameters. For both parameterizations the optimization progress does not vary much with the mesh width and the qualitative behavior is the same across all four combinations. A moderate decrease during the beginning steps of the optimization, then a steep descent followed by many small but slowly stagnating improvements during the local APPS stage.

It is notable that while the general form of the curve is similar, the actual quantitative performance is worse for the higher dimensional parameterization, which suggests that $n = 4$ is a better trade off between possible shapes and ease of optimization in this case. Particularly in the late parts of the optimization the finer parameterization shows a high number of very small improvements, probably due to on the fact that each coefficient has smaller impact overall on the shape, while both the HCP method and the APPS vary only one coordinate at a time. The difference however is not as large as the plot might suggest, the obstacle for $n = 4$ showing about a 25% improvement while the result for $n = 6$ achieves a 22% improvement.

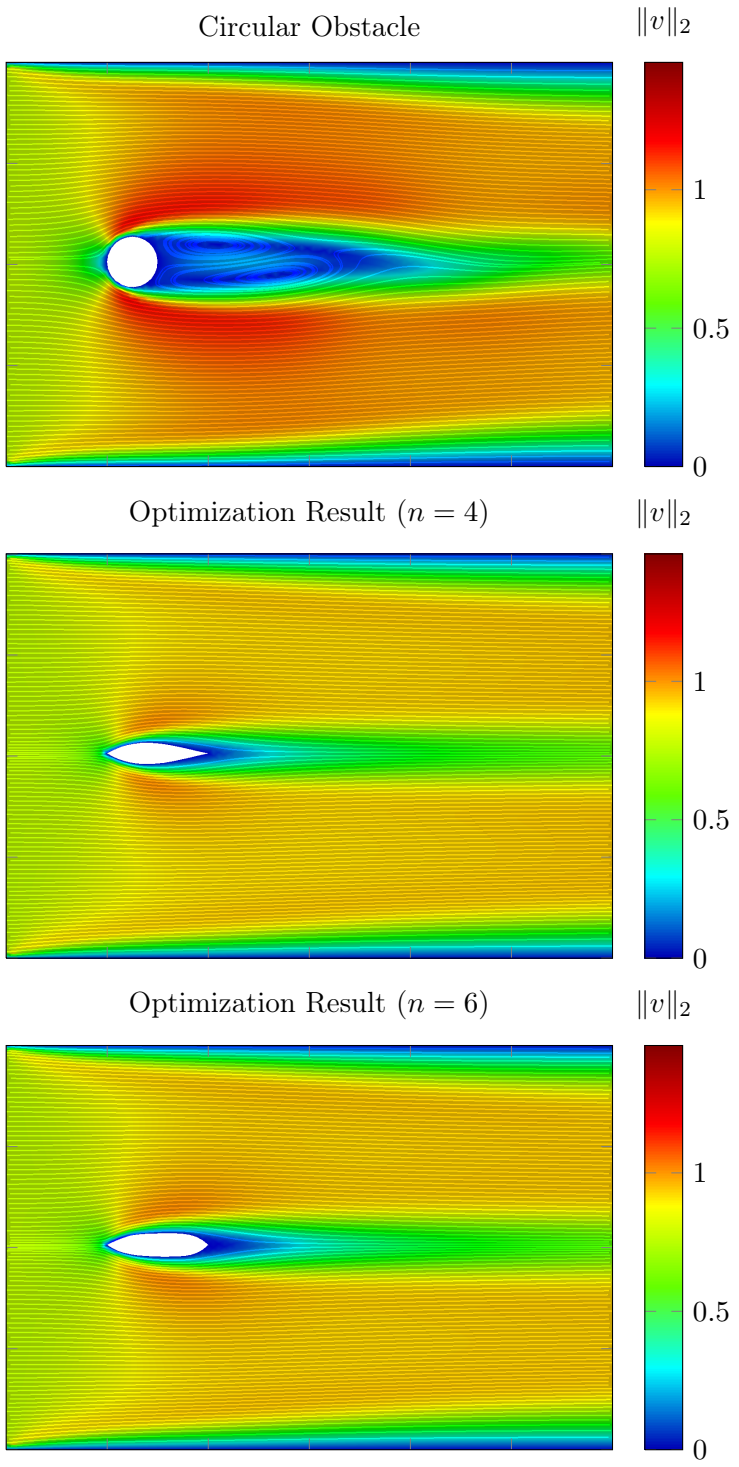

Figure 6.8: From top to bottom the figure shows a naïve circular obstacle, the optimization result for $n = 4$ and $h = 0.125$ and the result for $n = 6$ and $h = 0.125$. The result for $n = 4$ causes about 25% less energy dissipation then the circular geometry, the result for $n = 6$ around 22%. Both optimized geometries result in a near stationary flow with early reattachment and vortex shedding.

6.3.2 Nozzle Opening

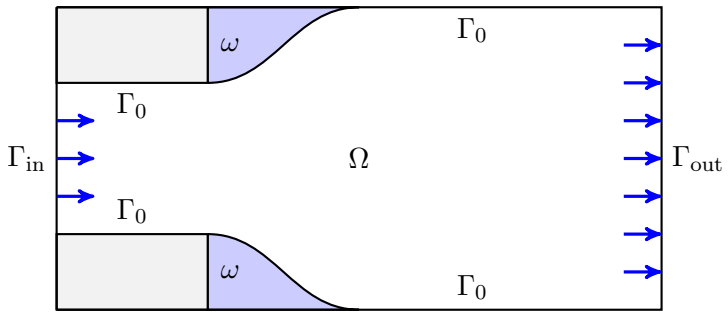

Figure 6.9: Sketch of optimized Nozzle. Fluid flows through the domain from left to right, no-slip conditions hold on the domain boundaries Γ_0 and the surface of the obstacle $\partial\omega$.

The second example is a nozzle formed flow outlet into a widening stream, illustrated in figure 6.9. Fluid enters the channel through an inlet of width 2 with a velocity of $u_{\text{in}}^T = (1, 0)$ on the left side of the domain and leaves through the far right channel wall of width 4. The upper and lower walls hold no-slip conditions. The geometry models the change between the narrow and wide parts of the domain, to ensure that the loss of energy during the transition is minimized. This models applications such as a widening of a river or artificial channel, water pipes or arteries.

The geometry is assumed to be symmetric and is parameterized in such a way that ω connects to the inlet on the left side and the wider part of the channel on the right side, point wise constraints ensure that the channel is not closed by ω. The evolution of the energy dissipation

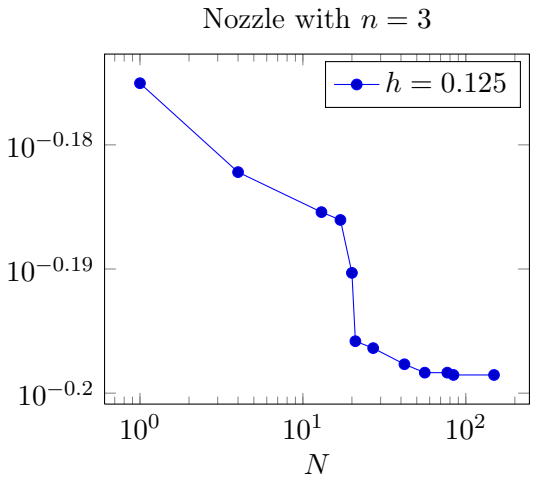

Figure 6.10: Relative improvement during the course of the nozzle optimization as compared to the non-smooth outlet.

of the then-best geometry is shown in figure 6.10. As in the last example there is a sharp decline shortly after the beginning of the optimization followed by a series of smaller adjustments leveling

out during the local stage of the optimization. Figure 6.11 shows snapshots of both the outlet without smooth transition and the optimized geometry in the beginning phases of the flow. The unoptimized geometry shows a large recirculation region with late reattachment. The contouring shows that the speed of the recirculating vortex at the far side is just as great as the outer layers of the main flow. The optimized geometry shows a much more even distribution of energy over the width of the flow with a much slower and smaller recirculation region. Reattachment starts after about 1/3 of the length it took in the unoptimized case.

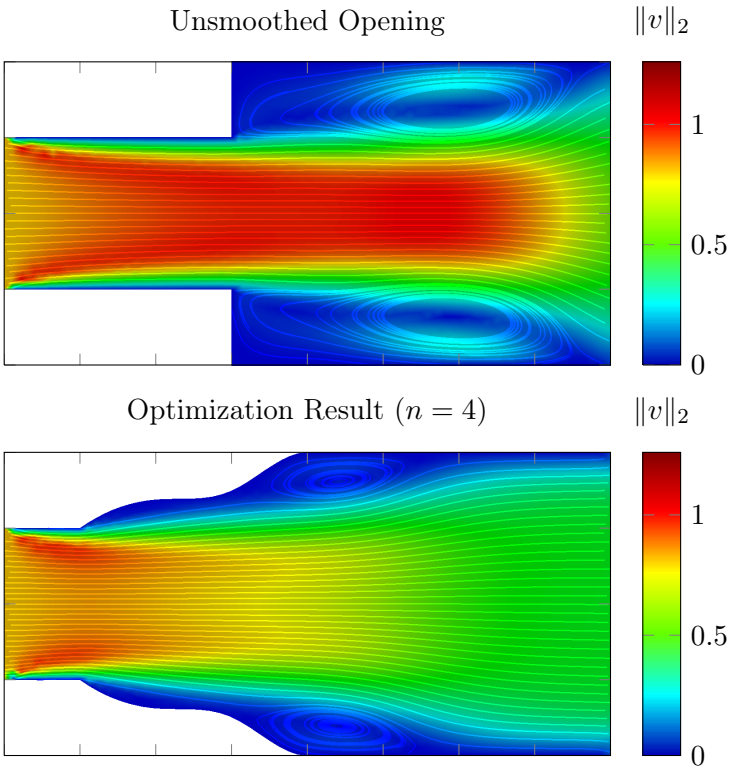

Figure 6.11: Top: Illustration of a flow outlet without any kind of transition, bottom: Flow through the outlet that resulted from the an optimization with $n = 4$ and $h = 0.125$. The optimized out lowers the energy dissipation by about 37% compared to simple one, reattachment happens much earlier with a smaller recirculation zone.

6.3.3 Branching Pipe Geometries

Last, two different kinds of pipe branchings were considered, inspired by actual pipelines that have to redirect their carried substance with minimal loss in flow rate to maximize efficiency.

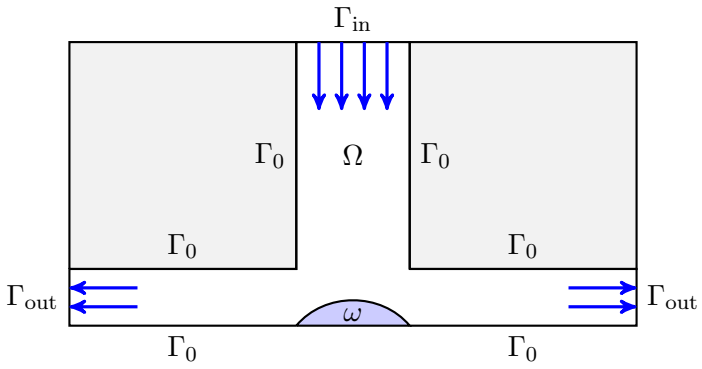

Figure 6.12: Sketch of a Tee-pipe geometry. Flow enters from the top and hits a perturbed wall splitting the stream in two directions.

Tee-pipe

The first is a Tee-pipe connection that splits a single stream of fluid into two opposing directions. A sketch is given in figure 6.12. The fluid enters the pipe from the top, hits the opposing wall that holds a parameterized perturbation and is split into two opposing streams. Figure 6.13 (bottom) shows an illustration of the flow through an unperturbed pipe while 6.13 (top) shows the same flow through a dented pipe. The optimization procedure found no improvement and returned the unperturbed tube which was found in a very early evaluation, for this reason we dispensed with a plot.

Y-connection

The second such example was a branching of one stream into two new streams of the same direction, illustrated in figure 6.14. Again, fluid enters the domain from the top and is split into two separate streams at the obstacle geometry, this time into streams that keep the same direction of flow. The geometry ω is parameterized in such a way that it connects to the supporting wall at the left and right corner the same way it is illustrated in the diagram. The optimization was done for two different resolutions of the Navier-Stokes simulation and found an improved geometry in both cases, the achieved improvements where about 6.5% for the lower resolution simulation and 7.5% for the higher one.

The development of the optimization is plotted in figure 6.15 and shows some differences to the other results. Most of the improvement is found very early in the optimization with only small improvements during the following evaluations. The higher accuracy simulation shows a greater degree of improvement during the whole simulation.

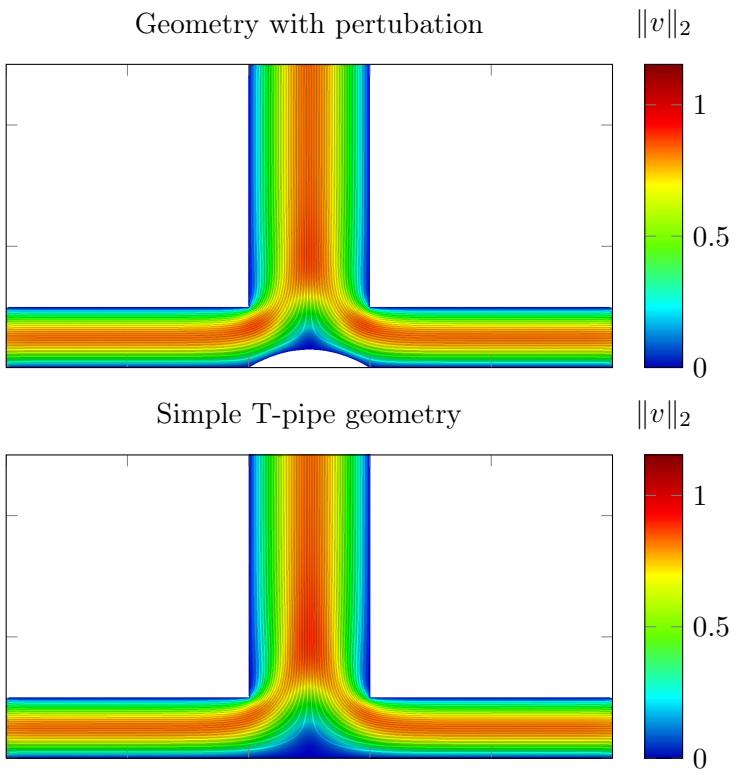

Figure 6.13: The top shows a perturbed T-pipe geometry, the bottom a normal one. The optimization did not find any improvement over the normal Tee-pipe shown in the bottom.

Figure 6.16 shows the flow through the unoptimized geometry and the optimization results of both resolutions. In this example the results show distinct variations: While both geometries feature curved elongations of the two channel boundaries left and right, the lower resolution features a dent in the center of the perturbation that creates a dead flow zone. In both cases though, it can be observed that stream entering the new pipes is less curved with respect to the channel boundaries. The vortex generated at the inside walls is smaller and the streams reattach a bit earlier.

That the dent in the perturbation does not exist in the result of the higher resolution simulation, which also gives a higher improvement, suggests that it is an artifact of the optimization. Inaccuracies in the objective function may have lead the optimization method to a false minimum.

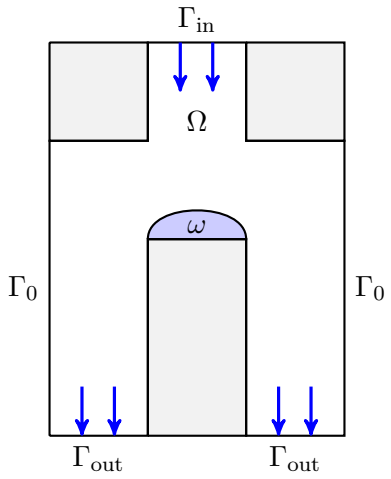

Figure 6.14: Sketch of a branching pipe. Fluid enters from above and is split into two flows of the same size. The geometry is modified where the previous stream hits the space between the two new pipes.

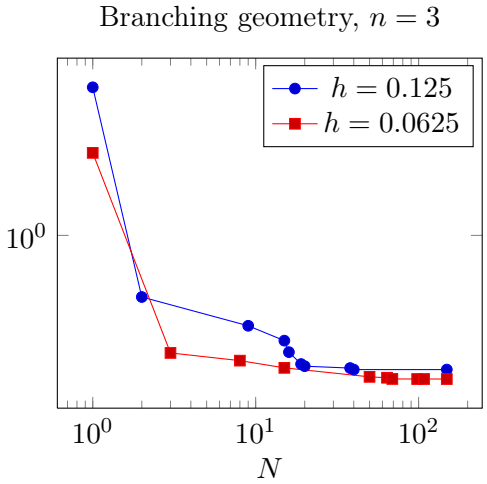

Figure 6.15: Relative improvement during the course of the branch optimization as compared to the unmodified connection.

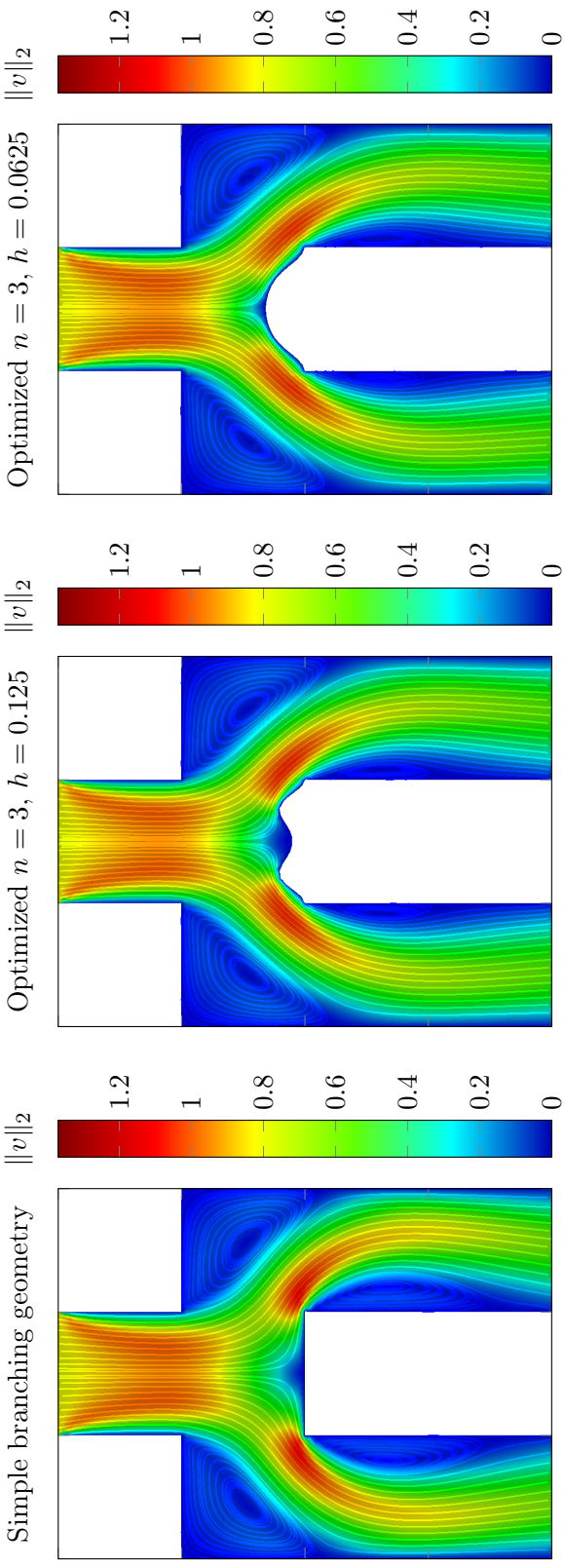

Figure 6.16: Leftmost is the unmodified geometry, center the result for mesh width $h = 0.125$ and left for $h = 0.0625$. In this example we see a clear difference between the results for different simulation resolutions. The improvement is about 6.5% for the center and 7.5% for the right geometry.

7 Summary

In this thesis a black-box approach to shape optimization problems in incompressible and transient flows was investigated. This approach was chosen for its flexibility and extensibility in a practical context and, after a discussion of its merits in comparison to other common approaches, its components where discussed in some detail.

The Navier-Stokes constraints were solved using a FEniCS based Taylor-Hood finite element discretization combined with an automatic meshing software based on the distmesh algorithm. Following the spirit of the approach this was treated as an opaque implementation. A parameterization of the feasible geometries using basis splines made the problem accessible to classical optimization techniques by reducing the search space to a finite number of dimensions and absorbing the fluid constraint into the objective function.

The resulting high-dimensional optimization problem was then treated using a parallel implementation of the hyperbolic cross point method. To accommodate the linear constraints resulting from the parameterization of the geometry constraints, this method was extended with a projection-penalty approach that allows the treatment of those constraints while maintaining feasibility of all evaluation points. As a further modification this implementation was supplemented with the option of a local generating set search to refine the result of the global search.

The finite element solution and the HCP implementation were validated separately using benchmark problems. The HCP implementation was also compared against the results given in the original publication and it was found that the addition of the local search made it possible to exceed the original performance in several cases.

Last, the complete shape optimization approach was tested using several numerical experiments inspired by real world applications. This included the geometry of an obstacle surrounded by flow, a nozzle-like flow widening and two kinds of branchings. In all cases but the Tee-pipe geometry the optimization procedure yielded geometries that significantly improved upon the energy dissipation of the simple reference geometry. The most dramatic change is seen in the first example: While the non-optimized obstacle creates an oscillating flow with detaching vortices, the optimized flow is near stationary with minimal separation. The nozzle and Y-connection

geometries feature a significant reduction in the size of the generated vortices and earlier stream reattachment.

Future perspectives

The wide scope of shape optimization as a field leaves much room for further work, the following is a short list of possibilities.

- The black-box approach should make it possible to introduce further PDE based constraints to test more involved examples. For example, the obstacle flow problem could be supplemented with an elasticity model that ensures a minimum of static stability during the optimization.
- Most actual applications cannot easily be reduced to a single quantity that is to be maximized or minimized. A good wing has not only a high lift but must find a balance between lift, drag and stability at the same time. This is known as multi-objective optimization and could possibly be approached by some modification to the HCP method.
- Combining the two points above, one could optimize criteria arising from different PDE constraints. For example modifying the obstacle problem to find a geometry that minimizes the flow disturbance while maximizing load bearing capacity.
- It would be interesting to optimize flows with multiple distinct controlled geometries, for example the upper corners in the Y-connection. This would drastically increase the problem size but may yield more improvement.

These are just a few suggestions for possible further work, the applied nature of this subject allows for many exciting ideas.

List of Figures

Bibliography

Adams, R. A. (1975). *Sobolev Spaces*. Academic Press, Inc., New York.

Alnæs, M. S. (2012). *UFL: a Finite Element Form Language*, chapter 17. Springer.

Alt, H. W. (2002). *Lineare Funktionalanalysis*. Springer, 4 edition.

Anderson, W. K., Newman, J. C., Whitfield, D. L., and Nielsen, E. J. (2001). Sensitivity analysis for Navier-Stokes equations on unstructured meshes using complex variables. *AIAA Journal*, 39(1):56–63.

Balay, S., Brown, J., , Buschelman, K., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Smith, B. F., and Zhang, H. (2013a). PETSc users manual. Technical Report ANL-95/11 - Revision 3.4, Argonne National Laboratory.

Balay, S., Brown, J., Buschelman, K., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Smith, B. F., and Zhang, H. (2013b). PETSc Web page. http://www.mcs.anl.gov/petsc.

Braess, D. (2007). *Finite Elemente*. Springer.

Branin, Jr., F. H. and Hoo, S. K. (1972). A method for finding multiple extrema of a function of n variables. In *Numerical methods for non-linear optimization (Conf., Univ. Dundee, Dundee, 1971)*, pages 231–237. Academic Press, London.

Burkardt, J., Gunzburger, M., and Peterson, J. (2002). Insensitive functionals, inconsistent gradients, spurious minima, and regularized functionals in flow optimization problems. *International Journal of Computational Fluid Dynamics*, 16(3):171–185.

Chorin, A. (1968). Numerical solution of the Navier-Stokes Equations. *Math. Comp.*, 22:745–762.

Croce, R. (2002). Ein paralleler, dreidimensionaler Navier-Stokes-Löser für inkompressible Zweiphasenströmungen mit Oberflächenspannung, Hindernissen und dynamischen Kontaktflächen. Diplomarbeit, Institut für Angewandte Mathematik, Universität Bonn, Bonn, Germany.

Deuflhard, P. and Hohmann, A. (2002). *Numerische Mathematik. I.* 3 edition.

Dixon, L. C. W. and Szegö, G. P. (1978). The global optimization problem: an introduction. In Dixon, L. C. W. and Szegö, G. P., editors, *Towards Global Optimisation 2*, pages 1–15. Amsterdam; New York : North-Holland Pub. Co. ; New York : Sole distributors for the U.S.A. and Canada, Elsevier North-Holland.

Garcke, J. (2013). Sparse grids in a nutshell. In Garcke, J. and Griebel, M., editors, *Sparse grids and applications*, volume 88 of *Lecture Notes in Computational Science and Engineering*, pages 57–80. Springer.

Gerstner, T. and Griebel, M. (1998). Numerical integration using sparse grids. *Numer. Algorithms*, 18:209–232.

Griebel, M., Dornseifer, T., and Neunhoeffer, T. (1998). *Numerical Simulation in Fluid Dynamics, a Practical Introduction*. SIAM, Philadelphia.

Griffin, J. D., Kolda, T. G., and Lewis, R. M. (2008). Asynchronous parallel generating set search for linearly constrained optimization. *SIAM J. Sci. Comput.*, 30(4):1892–1924.

Halton, J. H. (1964). Algorithm 247: Radical-inverse quasi-random point sequence. *Commun. ACM*, 7(12):701–702.

Hinze, M. and Kunisch, K. (2001). Second Order Methods for Optimal Control of Time-Dependent Fluid Flow. *SIAM J. Control Optim.*, 40(3):925–946.

Hu, Y.-K. and Hu, Y. (2007). Global optimization in clustering using hyperbolic cross points. *Pattern Recognition*, 40(6):1722 – 1733.

Ji, Y., Zhang, K.-C., and Qu, S.-J. (2007). A deterministic global optimization algorithm. *Applied Mathematics and Computation*, 185(1):382 – 387.

Kirby, R. C. and Logg, A. (2006). A compiler for variational forms. *ACM Transactions on Mathematical Software*, 32(3).

Kolda, T. G., Lewis, R. M., and Torczon, V. (2003). Optimization by direct search: new perspectives on some classical and modern methods. *SIAM Rev.*, 45(3):385–482.

Kolda, T. G., Lewis, R. M., and Torczon, V. (2006). Stationarity results for generating set search for linearly constrained optimization. *SIAM J. Optim.*, 17(4):943–968.

Lewis, R. M., Shepherd, A., and Torczon, V. (2007). Implementing generating set search methods for linearly constrained minimization. *SIAM J. Sci. Comput.*, 29(6):2507–2530.

Lewis, R. M. and Torczon, V. (2000). Pattern search methods for linearly constrained minimization. *SIAM J. Optim.*, 10(3):917–941.

Lewis, R. M., Torczon, V., and Trosset, M. W. (1998). Why pattern search works.

Logg, A., Mardal, K.-A., Wells, G. N., et al. (2012a). *Automated Solution of Differential Equations by the Finite Element Method*. Springer.

Logg, A., Ølgaard, K. B., Rognes, M. E., and Wells, G. N. (2012b). *FFC: the FEniCS Form Compiler*, chapter 11. Springer.

Logg, A. and Wells, G. N. (2010). Dolfin: Automated finite element computing. *ACM Transactions on Mathematical Software*, 37(2).

Logg, A., Wells, G. N., and Hake, J. (2012c). *DOLFIN: a C++/Python Finite Element Library*, chapter 10. Springer.

Mohammadi, B. and Pironneau, O. (2001). *Applied Shape Optimization for Fluids*. Numerical mathematics and scientific computation. Clarendon Press.

Neumaier, A. (2004). Complete Search in Continuous Global Optimization and Constraint Satisfaction. *Acta Numerica*, pages 271–369.

Niederreiter, H. (1992). *Random Number Generation and quasi-Monte Carlo Methods*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

Novak, E. (1988). *Deterministic and stochastic error bounds in numerical analysis*, volume 1349 of *Lecture Notes in Mathematics*. Springer, Berlin.

Novak, E. and Ritter, K. (1996). Global optimization using hyperbolic cross points. In *State of the art in global optimization (Princeton, NJ, 1995)*, volume 7 of *Nonconvex Optim. Appl.*, pages 19–33. Kluwer Acad. Publ., Dordrecht.

Ølgaard, K. B. and Wells, G. N. (2010). Optimisations for quadrature representations of finite element tensors through automated code generation. *ACM Transactions on Mathematical Software*, 37.

Persson, P.-O. (2004). *Mesh Generation for Implicit Geometries*. PhD thesis, Department of Mathematics, MIT.

Persson, P.-O. and Strang, G. (2004). A simple mesh generator in matlab. *SIAM Review*, 46:2004.

Piegl, L. A. and Tiller, W. (1995). *The NURBS book*. Monographs in visual communication. Springer.

Pintér, J. (1996). *Global Optimization in Action*. Kluwer.

Plantenga, T. D. (2009). Hopspack 2.0 user manual. Technical Report SAND2009-6265, Sandia National Laboratories, Albuquerque, NM and Livermore, CA.

Schmidt, S. and Schulz, V. (2010). Shape derivatives for general objective functions and the incompressible Navier-Stokes equations. *Control Cybernet.*, 39(3):677–713.

Shewchuk, J. R. (1996). Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In Lin, M. C. and Manocha, D., editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer. From the First ACM Workshop on Applied Computational Geometry.

Smolyak, S. (1963). Quadrature and interpolation formulas for tensor products of certain classes of functions. *Soviet Mathematics, Doklady*, 4:240–243.

Temam, R. (1977). *Navier-Stokes equations : theory and numerical analysis / Roger Temam*. North-Holland Pub. Co.

Thévenin, D. and Janiga, G. (2008). *Optimization and Computational Fluid Dynamics*. Optimization and Computational Fluid Dynamics. Springer.

Törn, A. and Žilinskas, A. (1989). *Global optimization*, volume 350 of *Lecture Notes in Computer Science*. Springer, Berlin.