M.Sc. Thesis
Master of Science in Engineering

DTU Compute
Department of Applied Mathematics and Computer Science

# Preconditioners in PDE-Constrained Optimization

Bjørn Christian Skov Jensen (s113208)

Kongens Lyngby 2016

**DTU Compute**

**Department of Applied Mathematics and Computer Science**

**Technical University of Denmark**

Matematiktorvet

Building 303B

2800 Kongens Lyngby, Denmark

Phone +45 4525 3031

compute@compute.dtu.dk

www.compute.dtu.dk

# Summary

When working with optimization problems it is often the case that the operator is ill-conditioned due to the spread of the eigenvalues, which produces poor convergence properties for iterative methods for solving optimization. Preconditioners are introduced to counteract this effect and produce faster convergence. In this thesis we consider the link between preconditioners for a PDE-constrained optimization problem and the structure of the Hilbert space in which we seek the solution.

# Preface

This Master thesis was prepared at the department of Applied Mathematics and Computer Science at the Technical University of Denmark in fulfillment of the requirements for acquiring a Master of Science degree in Engineering.

Kongens Lyngby, July 21, 2016

Bjørn Christian Skov Jensen (s113208)

# Acknowledgements

My deepest gratitute go to my supervisor Associate Professor Kim Knudsen who have been great pillars of support for me while working on this thesis. Without him this thesis would not have been possible.

Furthermore, I extend my gratitute to my second supervisor Post Doc Bolaji James Adesokan, who have been a great inspiration throughout this project and worked tirelessly with me during the last month of the project, when Kim was not available.

I would also like to express my thanks to my family, my parents and siblings, who have supported me through all these years of education. Even after they politely asked me to find my own place to stay.

I express my thanks to Professor Per Christian Hansen, who went out of his way to help me find the theory I needed for completely understanding the MINRES algorithm, even after his vacation had begun.

# Contents

**Summary** i

**Preface** iii

**Acknowledgements** v

**Contents** vii

**1 Introduction** 1

1.1 Problem statement . . . 3
1.2 Outline . . . 3
1.3 Notation . . . 4

**2 Theory** 7

2.1 Background . . . 8
2.1.1 Distributed control problem . . . 10
2.2 Function spaces . . . 11
2.2.1 Sobolev spaces . . . 11
2.2.2 Gâteaux and Fréchet differentiability . . . 13
2.3 Lagrangian and optimality conditions . . . 15
2.3.1 Lagrangian over Banach spaces . . . 16
2.3.2 First order optimality conditions . . . 17
2.4 Discretization . . . 20
2.4.1 The basics of the finite element method . . . 21
2.4.2 Discretizing the problem . . . 23
2.4.3 Applying Dirichlet boundary conditions . . . 26
2.4.4 Zero boundary conditions . . . 28
2.5 Matrix Theory . . . 30
2.5.1 Saddle point systems . . . 30

2.5.2 The Schur complement . . . 31
2.6 Preconditioning . . . 33
2.6.1 Condition number . . . 33
2.6.2 Riesz isomorphism . . . 34
2.6.3 Finite dimensional problems . . . 38
2.6.4 Schur complement approximations . . . 42
2.6.5 Preconditioner for infinite dimensional case . . . 43

**3 Practical implementation 53**
3.1 Mesh and basis functions . . . 54
3.2 Matrix assembling . . . 55
3.3 Preconditioner . . . 59
3.4 MINRES . . . 60
3.5 Visualization . . . 63

**4 Numerical results 67**
4.1 Core setup . . . 69
4.2 Different preconditioner Type I . . . 69
4.3 Different Preconditioner Type II . . . 73
4.4 Different Boundary . . . 75
4.5 Reduced system . . . 75

**5 Conclusion 79**
5.1 Future outlook . . . 81
5.1.1 Schur complement generalizations . . . 81
5.1.2 Other model problems . . . 82

**A Problem statement 83**
A.1 Learning objectives . . . 84
A.2 Time schedule . . . 85
A.3 Reflections . . . 85
A.3.1 Time schedule . . . 86
A.3.2 FEniCS (Python) vs. MATLAB . . . 86
A.3.3 Electrical Impedance Tomography (EIT) . . . 87

**B Various mathematical results 89**
B.1 Miscellaneous . . . 89
B.1.1 Finite element mass and stiffness matrices . . . 89

B.1.2 Operators: eigenvalues and orderings . . . . . . . . . . . . . . . 90
B.1.3 Fundamental Lemma of Calculus of Variations . . . . . . . . . 91
B.2 Krylov subspace optimization . . . . . . . . . . . . . . . . . . . . . . . . . 92
B.2.1 An orthogonal basis . . . . . . . . . . . . . . . . . . . . . . . . . 93
B.2.2 MINRES . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 94
B.2.3 Code: Preconditioned MINRES algorithm . . . . . . . . . . . . 97

**C Additional results** **99**

**D G-bar framework** **115**
D.1 gbar.py . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 115
D.2 Personalized connection and job . . . . . . . . . . . . . . . . . . . . 128

**Bibliography** **131**

# CHAPTER 1

# Introduction

Optimization is the task of finding the best object with respect to a set of existing criteria. We typically refer to the task as *the problem* we try to solve and the best object *the solution* to the problem. The topic of optimization has been a popular topic in mathematics due to the natural occurence of problems like this in our world. Examples of problems could be optimal heating, optimal flow control or least squares parameter estimation[De 15].

In particular over the last decades with the extensive growth in the processing power of computers it has become feasible to tackle greater and more complex problems of this sort.

In mathematical notation a general optimization problem takes the form

$$\begin{cases} \min\limits_{x \in X} & f(x), \\ \text{subj. to} & g(x) \leq 0 \end{cases}$$

where $f : X \to \mathbb{R}$ is a function to be minimized over some normed vector space $X$ and $g$ defines the criteria $x$ should fulfil. Some times one considers the problem where $g(x) \leq 0$ is replaced by an equality constraint $e(x) = 0$. We note that this is not actually another problem as $e(x) = 0$ can be written in the above form by chosen

$$g(x) = \begin{bmatrix} e(x) \\ -e(x) \end{bmatrix} \leq \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

This is what we will be doing in this thesis and we will thus use $e$ for the constraints from here on.

Solving problems like these can in general be very hard with various complications. The difficulty of the problem depend on both the function $f$, the constraints $e$ and the search space $X$. Luckily for us, it is often the case that these have particular structures that we may exploit. In our case we will consider the situation where $e$ involves a *partial differential equation* (PDE), hence our task in this thesis is labeled *PDE-constrained optimization.*

When solving optimization problems there are traditionally two different approaches; *direct* and *iterative* methods. Direct methods sometimes invloves finding a closed formula expression for the solution, but might also be a fixed algorithmic process producing the solution. Obviously iterative methods also produce the solution, however, they do it by taking may smaller steps towards the solution, and we may stop them somewhere along the way when we think we are "close enough" to the actual solution. This is often useful as while it might take hundreds or thousands of steps to reach the solution, the result we get after just fifty steps might actually be close enough for the purpose. Another significant difference of the two methods is that direct solvers often need significantly more memory than iterative solvers. This is not to say that direct solvers are bad choices, the problems are simply outgrowing them in size.

When using iterative methods for computing approximate solutions to an optimization problem it is important to know how well, the computed solution approximate the actual solution to the problem (even if we don't know the actual solution). In other words we want to have an upper bound for the error we are committing by using the approximation. The decrease in the error will depend on the *condition number* for the problem, which relates to the eigenvalues of the problem. In order for the iterative methods to have fast convergence we require the condition number to be small (by construction this means close to 1).

Often, however, problems will not have a small condition number because of the spread of the eigenvalues. In these cases we want to transform our problem into a new problem with better properties. We call such a transforming operator a *preconditioner* for the problem. As it turns out, there is a relation between preconditioners and the metric structure of our search space $X$.

This is a highly theoretical result, but with direct practical implications. We will in this thesis explore this using the distributed control problem with mixed boundary conditions:

$$\begin{aligned} \min_{(y,u)\in Y\times U} \quad & J(y,u) = \frac{1}{2}\int_\Omega (y-y_d)^2\,dx + \frac{\alpha}{2}\int_\Omega u^2\,dx \\ \text{subj. to} \quad & -\Delta y = u \quad \text{in } \Omega \\ & y = f \quad \text{on } \partial\Omega_D \\ & \tfrac{\partial y}{\partial n} = g \quad \text{on } \partial\Omega_N. \end{aligned} \tag{1.1}$$

This is the distributed control problem with mixed boundary conditions. Here $\Omega$ is our domain, $y$ is the state variable and $u$ is the control variable. $y_d$ is the desired state, $f$ and $g$ are the Dirichlet and the Neumann boundary functions respectively.

$\alpha$ is a positive scalar, often refered to as the *regularization parameter*. $Y$ and $U$ are function spaces and our search space $X$ is their direct product: $X = Y \times U$.

The Poisson equation, $-\Delta y = u$, arises in various physical settings. An example of such a setting could be heat distribution in an object, say a room, defined by the domain $\Omega$. Here $y$ is the temperature state and $u$ is a heat source term, say a radiator. The Dirichlet boundary condition corresponds to an ideal cooler/heater with infinite heat conductivity and the Neumann condition corresponds to a certain heat flux. If the Neumann condition is zero, $g = 0$, then it corresponds to ideal insulation. Then $y_d$ will be a desired heat distribution.

While we can typically not fine-control heat sources everywhere in the interior of a room (radiators are commonly placed along walls as we want to be able to move about the room) distributed control could occur in solid conducting materials where heat might be induced by currents generated from an exteriorly controlled magnetic field.

## 1.1 Problem statement

As already noted we wish in this thesis to explore the relation between preconditioners for an optimization problem, here Problem (1.1), and the underlying metric structure of the search space $X = Y \times U$. We summarize this in the following question.

> How does the underlying structure of the search space $X$ influence performance with and without preconditioners in PDE-constrained optimization?

## 1.2 Outline

This thesis is written with following structure: In the first chapter we have the introduction, which you might have read if you are here now. In this we built up the basis for our problem. We introduce a number of definitions, facts and concepts without necessarily talking much about their origin in order to reach the actual question. We also put forth our model problem which we will work with repeatedly thoughout the thesis and relate the problem to a more graspable physical scenario.

The second chapter covers the theory relevant for the thesis. We first discuss some background on optimization and PDEs, the general problem and our particular

model problem. Following that we recall the concepts of differentiability of operators in function spaces in order to introduce the Lagrangian and the first order optimality conditions for our problem.

We briefly outline the basic concepts we need from finite element theory and describe the descretization of our problem for a given finite element basis. We then discuss how boundary conditions are applied in finite element discretization and in particular how this relates to our problem, which will turn out to be a system of matrix equations.

We then talk a little about matrix theory, touching upon saddle point systems and the Schur complement. Following this up by discussion on preconditioning of saddle point systems in general, in the end relating this to our model problem.

In the third chapter we discuss the practical implementation of our problem in Python. We will go over the different step in setting up the problem and how these were done in practice.

We present the results in chapter four. Here we present our numerical results with plots and tables for different regularization values, grid sizes, and preconditioners.

In chapter five we present our conclusion along with an outlook to possible extensions.

In the end of the thesis we have several appendix with various information related to the project. Appendix A will be the problem statement as formulated in the beginning of the project. The problem statement will be following by section with reflections discussion obstacles encountered during the project.

Appendix B will contain supplementary theory and results not directly related, but still needed in the thesis. Here we will also summarize some of the basic theory learned in courses, which is relevant and might not be common knowledge to every reader.

Appendix C contains additional results in the form of several pages with plots.

In Appendix D we present a python framework set up to send and execute Python code on the university servers and automatically fetch the resulting data again.

## 1.3 Notation

For the readers convenience we here introduce the notation common used though-out the thesis, unless otherwise speccified.

| | |
|---|---|
| $A, B, C, D, \dots$ | Banach space/matrix operators in infinite/finite dimensions. |
| $I$ | The identity operator. |
| $M,\ K$ | For finite element $M$ is allways the mass matrix and $K$ is always the stiffness matrix. |
| $Q, U, V, W, X, Y$ | Normed vector spaces. |
| $a, b, c, s$ | Bilinear forms, though $c$ can be a scalar. |
| $f, g$ | Functionals. |
| $u, v, w, x, y, h$ | Elements in normed vector spaces. $h$ may also be a scalar relating to grid refinement in finite elements. |
| $p, \lambda$ | Lagrange multiplier. |
| $t, \alpha, \beta$ | Real scalars. |
| $d, i, j, k, n, m$ | Integers. |
| $\square_h$ | Subscripting by $h$ denotes the finite element subspace approximation of $\square$. |
| $\mathbf{u}, \mathbf{y}, \mathbf{p}, \mathbf{g}, \mathbf{f}$ | Vectors in $\mathbb{R}^d$. |
| $J, e$ | $J$ is always a functional and $e$ is always an operator relating to constraints. |
| $\Omega$ | Denotes a subset of $\mathbb{R}^d$. |
| $\phi, \psi, \chi$ | Finite element basis functions. $\chi_\Omega$ may also denote the indicator map on a particular set. |
| $H$ | Hilbert space. |
| $(\cdot,\cdot)_H$ | Inner product on $H$. The subscript may be omitted. |
| $\square^*$ | If $\square$ is a Banach space: Dual space of $\square$. If $\square$ is a Banch space operator: Adjoint of $\square$. |
| $\langle\cdot,\cdot\rangle_{X^*,X}$ | Dual pairing for the Banach space $X$. Subscripts may be omitted. |
| $A \leq B$ | If $A, B : H \to H$ this means $(Ax, x)_H \leq (Bx, x)_H$ for all $x \in H$. If $A, B : H \to H^*$ this means $\langle Ax, x\rangle_{H^*,H} \leq \langle Bx, x\rangle_{H^*,H}$. |
| $A \lesssim B$ | There is $c > 0$ such that $A \leq cB$. |
| $A \sim B$ | Means $A \lesssim B$ and $B \lesssim A$. |
| $\mathcal{I}_H, \mathcal{R}_H$ | Riesz isomorphism on $H$, $\mathbb{R}_H = \mathcal{I}_H^{-1}$, $\langle \mathcal{I}_H x, y\rangle = (x, y)_H$ for $x, y \in H$. |

# CHAPTER 2

# Theory

In this chapter we set up the theory for PDE-constrained optimization. We will include both aspects from optimization theory and functional analysis in order to cover the theory in detail. It is our hope that this section will thus be accessible and provide enough in depth explanations that other students with a background in introductory optimization and functional analysis, who might be interested in the topic, can read and understand this without significant further outside reading.

We will start out by briefly revisiting the topics of optimization and partial differential equations, before formulating their – for us more interesting – combination, the PDE-constrained optimization problems[De 15]. We will then proceed to define and work through the different tools we will need to tackle these problems.

We briefly cover Sobolev spaces[Gru08; Eva08] and proceed to Gâteaux and Fréchet differentiability[De 15]. We give the definition of the Lagrangian and go over how it helps us handle optimization problem[Zei95] by leading us to the KKT-conditions[De 15], which we derive for our model problem.

We next cover the discretization using the finite element method[Eng09] and discretize our model problem using it, finally discussing how boundary conditions are applied to our matrix problem. We follow this up with some matrix theory briefly touching saddle point systems[BGL05] and then discuss the Schur complement[Zha05].

Finally we consider preconditioning[Zul11] where we talk about the condition number, and relates the inner product to choices of preconditioners. Following Zulehner we derive a preconditioner for our discretized system. We discuss briefly approximations to the Schur complement[Pea13; RDW10] and end on a note about considerations for preconditioning in the operator setting.

## 2.1 Background

The study of optimization in general concerns itself with – as the name implies – finding optimal solution to a certain problem. The traditional problem considered is that of finding the minimum or the maximum of a particular function, however, since these two problems are equivalent ($x^*$ being the minimum of $f(x)$ if and only if $x^*$ is the maximum of $-f(x)$), one often simply studies how to find minima.

In general the problem considered can be formulated as: Given $f : X \to \mathbb{R}$ and $g : X \to Y$, find the $x \in X$ by solving

$$\begin{aligned} \min_{x \in X} \quad & f(x), \\ \text{subj. to} \quad & g(x) \leq 0. \end{aligned} \tag{2.1}$$

Here $X$ and $Y$ are normed vector spaces of finite dimension, i.e. $\mathbb{R}^n$. Here $g$ defines constraints on our problem and $\leq$ is some partial ordering on $Y$. For the finite dimensional case $\leq$ could for instance be a coordinate-wise comparison.

In this project we will primarily work with equality constraints, which are special cases of inequality constraints. In particular $g(x) = 0$ is the same as

$$\begin{bmatrix} g(x) \\ -g(x) \end{bmatrix} \leq 0.$$

In general problem (2.1) can be quite complicated to solve, however, often $f$ will have a particular structure that can be exploited. Many different algorithms have been developed and refined to handle these different possible structures and a variety of large scale toolboxes are available online. Commercial numerical software such as MATLAB offer additional optimization toolboxes, but free options are available as well. The free toolbox CVX for instance is in its early stages of support on the free numerical computation software Octave.

In this thesis, however, $X$ and $Y$ will in general not be the traditional choice of $\mathbb{R}^n$ but spaces of functions, since we are interested in working with partial differential equations and the unknowns here are functions.

The solution of partial differential equations (from here on abbreviated as PDE) and differential equations in general has been a particularly popular topic of study due to their ability to capture and model physical phenomena. A PDE is formulated over a domain $\Omega \subset \mathbb{R}^d$ and typically with a condition on the behavior on the boundary of the domain, $\partial\Omega$. The Dirichlet boundary condition is a common boundary conditions

and describes how the restriction of the solution to the boundary should behave. The Dirichlet boundary value problem is formulated as follows

$$\begin{aligned} \mathcal{L}y &= f, &\quad& \text{in } \Omega, \\ y &= g, && \text{on } \partial\Omega, \end{aligned} \tag{2.2}$$

where $y$ is the unknown function we wish to solve for, the right hand side $f$ is a known function on the domain $\Omega$ and $g$ is a known function on the boundary $\partial\Omega$. $\mathcal{L}$ is here the differential operator acting on $y$.

$\mathcal{L}$ may take on many different forms, depending on the problem considered. An example could be the Poisson problem

$$\begin{aligned} -\Delta y &= f, &\quad& \text{in } \Omega, \\ y &= g, && \text{on } \partial\Omega, \end{aligned} \tag{2.3}$$

where $\mathcal{L} = -\Delta = -\sum_i \frac{\partial^2}{\partial x_i^2}$ is the *Laplace operator* or simply *Laplacian.*

Other classes of PDE-problems often encountered is the Neumann boundary value problems. Instead of a fixed boundary value these problems are characterized by the fixture of the outgoing directional derivative on the boundary. These problems have the formulation

$$\begin{aligned} \mathcal{L}y &= f, &\quad& \text{in } \Omega, \\ \tfrac{\partial y}{\partial n} &= g, && \text{on } \partial\Omega, \end{aligned} \tag{2.4}$$

where $n$ denotes the outwards normal vector on the boundary of $\Omega$, and $\frac{\partial y}{\partial n} := \nabla y \cdot n$.

Analytical solutions to PDE problems are often very difficult to obtain and working with particular class of them might warrant a thesis in its own right. However, PDEs will not be our main focus in this thesis and the one we consider will be given as part of another problem we wish to solve, the problem of *PDE-constrained optimization.*

The PDE-constrained optimization problem takes a general form not unlike the optimization problem (2.1), with a function we wish to minimize and a set of constraints. The problem is often formulated as follows

$$\begin{aligned} \min_{y,u} &\quad J(y,u), \\ \text{subj. to} &\quad e(y,u) = 0, \end{aligned} \tag{2.5}$$

where $(y,u) \in Y \times U = X$, and $Y$ and $U$ are Banach spaces (often Hilbert spaces). Here $J$ is the functional on $X$ which we wish to minimize and $e : X \to W$ is the constraint. As the name PDE-constrained optimization implies $e$ here contains one

or more PDEs linking $y$ and $u$. The unknown $y$ is typically called the state, whereas $u$ is referred to as the control for the problem. In case of a heating problem in a domain $\Omega$, $y$ would describe the heat distribution in $\Omega$ and $u$ would describe how and where we add heat or energy into the system. Typically the domain $\Omega$ is a bounded domain.

We note that in literature the problem in its reduced form $f(u)$ is often considered – see for example [De 15] and [Her10] – as the implicit function theorem applied to the equation $e(y, u) = 0$ under certain conditions introduce a map $u \mapsto y(u)$. This map is sometimes called a *solution map*. Thus $f(u) = J(y(u), u)$.

Since $y$ and $u$ are functions obviously $Y$ and $U$ must be function spaces. We typically assume $Y$ and $U$ to be some subspaces of the Lebesgue space $L^2(\Omega)$, but we will return to the topic of function spaces in Section 2.2.

Now that we have established an abstract formulation of the PDE-constrained optimization problem, it might be time to consider a few examples of concrete problems.

### 2.1.1 Distributed control problem

The name *distributed control problem* refers to problems where the control parameter $u$ is spread across the entire domain of the problem, $\Omega$. We present here the Poisson distributed control problem

$$\begin{aligned} \min_{y,u} \quad & J(y,u) = \tfrac{1}{2}\|y - y_d\|^2_{L^2(\Omega)} + \tfrac{\alpha}{2}\|u\|^2_{L^2(\Omega)} \\ \text{subj. to} \quad & -\Delta y = u, \quad \text{in } \Omega, \\ & y = f, \quad \text{on } \partial\Omega_D, \\ & \tfrac{\partial y}{\partial n} = g, \quad \text{on } \partial\Omega_N. \end{aligned} \tag{2.6}$$

Here $\alpha > 0$ is a fixed constant, $J$ is a goal functional, $\Omega$ the domain of the state $y$ and control $u$ and $\partial\Omega_D$ and $\partial\Omega_N$ the Dirichlet and Neumann boundary respectively. These are disjoint sets with union $\partial\Omega$ and they define the part of the boundary with Dirichlet respectively Neumann boundary conditions.

The goal functional $J$ has here been defined and it consists of two terms. The first term measures the difference between our obtained state and some quantity $y_d$, and the second puts a bound on the control variable and keeps it small. The quantity $y_d$ is here our desired state, the state we hope $y$ to become, by tuning our control $u$, though it is often an idealized and perhaps physically unattainable state.

A particularly interesting observation is that it is here clear how the solution map $u \mapsto y(u)$ mentioned in the previous section arise as a solution to the PDE part of the

problem. The existence and well-definedness of the solution map is thus tied directly to the existence and uniqueness of the PDE problem.

Some other things that might stick out are the parameter $\alpha$ and the choice of $L^2(\Omega)$-norms in $J$. $\alpha$ is called the *regularization parameter* and denotes how much we penalize our control variable. It controls how "wild" our controls $u$ can behave. Adding a term like $\frac{\alpha}{2}\|u\|^2_{L^2(\Omega)}$ limiting our variable is sometimes called a *Tikhonov regularization* and thus one might at time see $\alpha$ referred to as a *Tikhonov parameter*.

As noted, the $L^2(\Omega)$-norm is used $J$ and is essentially the result of the underlying assumptions that $y$ and $u$ belong to some subspace of $L^2(\Omega)$. In the situations where this subspace has other possible norms, these are of course also valid choices here. But there is no denying that the well-understood structure and properties of $L^2(\Omega)$ makes a sound argument for exactly this choice. A natural choice for $y$ is $H^1(\Omega)$, which loosely stated is the set of one time differentiable $L^2(\Omega)$ functions.

We will return to Equation (2.6) throughout the Thesis, as it is our model problem of choice.

## 2.2 Function spaces

In this section we will introduce the Sobolev spaces $W^{k,p}$. Following this we will introduce Fréchet differentiability, which generalizes differentiation to Banach space operators.

We expect the reader to already be familiar with the Lebesgue spaces $L^p$. We simply note here that we will in this thesis primarily be interested in the Hilbert space case, $p = 2$, and only the real function spaces.

### 2.2.1 Sobolev spaces

Finding solutions to partial differential equations can, as previously stated, be very hard in general. To any given differential equation there might not even be a solution. To mitigate this fact, we search to a class of functions slightly broader than the class of differential functions in the classical sense.

To do this we expand the notion of differentiation from $C^k(\Omega)$-functions to the set of distributions $\mathcal{D}'(\Omega)$. This set is the set of all linear functionals on $C_0^\infty(\Omega)$. By extending to this set we will be able to differentiate a much broader range of objects than we can traditionally. For instance every locally integrable function $f \in L^1_{\text{loc}}(\Omega)$

constitutes a distribution by acting on $\phi \in C_0^\infty(\Omega)$ as follows:

$$\langle f, \phi \rangle = \int_\Omega f\phi \, dx.$$

Thus $L^1_{\text{loc}}(\Omega) \subset \mathcal{D}'(\Omega)$. Furthermore the set of Radon measures on $\Omega$, $\mathcal{M}(\Omega)$, is contained in $\mathcal{D}'(\Omega)$ by

$$\langle \mu, \phi \rangle = \int \phi \, d\mu,$$

for $\mu \in \mathcal{M}(\Omega)$ and $\phi \in C_0^\infty(\Omega)$.

In the theory of distributions differentiation is extended by defining $u' \in \mathcal{D}'(\Omega)$ as follows

$$\langle u', \phi \rangle := \langle u, -\phi' \rangle.$$

As a special case of this we obtain the *weak derivative.*

Let $y \in L^p(\Omega)$ be a function, we say that $w \in L^p(\Omega)$ is the *weak derivative* of $y$ if for all functions $\phi \in C_0^\infty(\Omega)$ the following equality holds

$$\int_\Omega y\phi' \, dx = \int_\Omega w\phi \, dx. \tag{2.7}$$

The more general definition follow here

**Definition 1.** Let $y \in L^p(\Omega)$ be a function and $\alpha = (\alpha_1, \ldots, \alpha_k)$ a multi-index. We say that $w \in L^p(\Omega)$ is the *weak derivate* of $y$ associated with $\alpha$ if

$$\int_\Omega y D^\alpha \phi \, dx = (-1)^{|\alpha|} \int_\Omega w\phi \, dx, \tag{2.8}$$

for all $\phi \in C_0^\infty(\Omega)$. We then write $w = D^\alpha y$.

**Remark 2.** We recall that $C_0^\infty(\Omega)$ is the set of smooth functions compactly supported on $\Omega$, and

$$D^\alpha := \frac{\partial^{|\alpha|}}{\partial^{\alpha_1}\partial^{\alpha_2}\ldots\partial^{\alpha_k}},$$

and $|\alpha| = k$.

Using this even discontinuous functions may be considered differentiable (in the weak sense). Take for instance the function $x \mapsto |x|$ on $[a, b]$, $a < 0 < b$. Clearly this function is in $L^p([a, b])$ and one can easily show by splitting the integral to the continuous parts of the function that

$$x \mapsto \begin{cases} 1, & \text{for } x > 0, \\ -1, & \text{for } x < 0, \end{cases}$$

is a weak derivative for $x \mapsto |x|$. Note that we write "a" and not "the" weak derivate, this is because there can be more than one weak derivative for a function.

Furthermore the definition allows us to define the Sobolev spaces, $W^{k,p}(\Omega)$, which are subspaces of $L^p(\Omega)$ consisting of $k$ times weakly differentiable functions. They come with a special norm as well, and for the Hilbert space case an inner product.

**Definition 3.** Let $1 \leq p < \infty$ and $k \in \mathbb{N}$. We then define the Sobolev space

$$W^{k,p}(\Omega) := \{y \in L^p(\Omega) \mid w = D^\alpha y \text{ exist and belong to } L^p(\Omega) \text{ for all } |\alpha| \leq k\},$$

with the norm

$$\|y\|_{W^{k,p}(\Omega)} = \left( \sum_{|\alpha| \leq k} \int_\Omega |D^\alpha y|^p \, dx \right)^{\frac{1}{p}}.$$

When $L^p(\Omega)$ is a Hilbert space, i.e. $p = 2$, the corresponding Sobolev spaces are Hilbert spaces as well. Typically one writes $H^k(\Omega)$ for $W^{k,2}(\Omega)$, and the inner product is given by

$$(u, v)_{H^k(\Omega)} = \sum_{|\alpha| \leq k} \int_\Omega D^\alpha u D^\alpha v \, dx.$$

### 2.2.2 Gâteaux and Fréchet differentiability

Working in Banach spaces we need a notion of differentiability for operators between such spaces. In this section we introduce the Fréchet derivative which generalized differentiation to Banach spaces. We do this following the definition in [De 15]. To do this we will also need to define the directional derivative for operators as well as the Gâteaux derivative.

In the following set of definitions we assume $U$ and $V$ to be Banach spaces.

**Definition 4.** Let $F : U \to V$ be a map, $u, h \in U$, if the limit

$$\lim_{t \to 0} \frac{1}{t} \left( F(u + th) - F(u) \right) = \partial F(u)h \tag{2.9}$$

exists, then we say that $\partial F(u)h$ is the *directional derivative* at $u$ in direction $h$. If the limit exists for all $h \in U$, then we say that $F$ is *directionally differentiable* at $u$.

**Definition 5.** Let $F : U \to V$ be directionally differentiable at $u \in U$ and the operator $\partial F(u) : U \to V$ from equation (2.9) be a continuous linear map, then $F$ is called *Gâteaux differentiable* at $u$ and we call $F'(u) := \partial F(u)$ the *Gâteaux derivative* of $F$ at $u$.

**Definition 6.** Let $F : U \to V$ be Gâteaux differentiable at $u \in U$ and satisfy

$$\frac{\|F(u+h) - F(u) - F'(u)h\|_V}{\|h\|_U} \to 0 \quad \text{as } \|h\|_U \to 0,$$

then we say that $F$ is *Fréchet differentiable* at $u$ and the function $F'(u)$ is called the *Fréchet derivative* of $F$ at $u$.

If for every $u \in W \subseteq U$ $F$ is Fréchet differentiable, then we naturally say that $F$ is Fréchet differentiable in $W$ and when $W = U$ we simply say that $F$ Fréchet differentiable.

We will now compute a few examples of Fréchet derivatives, some of which will prove useful when we will work on our optimization problem.

**Example 7.** Let $U = L^2(\Omega)$. The Fréchet derivative of $F(u) = \|u\|^2 = (u, u) = \int u^2\, dx$, where $u \in L^2$, is $F'(u)h = 2(u, h) = 2\int uh\, dx$.

We see that

$$\begin{aligned}
\left|F(u+h) - F(u) - 2\int uh\, dx\right| &= \left|\int (u+h)^2\, dx - \int u^2\, dx - 2\int uh\, dx\right| \\
&= \left|\int u^2\, dx + \int h^2\, dx + 2\int uh\, dx - \int u^2\, dx - 2\int uh\, dx\right| \\
&= \left|\int h^2\, dx\right| = \|h\|_{L^2}^2.
\end{aligned}$$

Hence

$$\frac{\left|F(u+h) - F(u) - 2\int uh\, dx\right|}{\|h\|_{L^2}} = \|h\|_{L^2} \to 0 \quad \text{as } \|h\|_{L^2} \to 0,$$

and $F'(u)h = 2(u, h)$. △

So the squared $L^2$-norm is everywhere Fréchet differentiable.

We cover here two more examples of Fréchet derivatives.

**Lemma 8.** *Let $U$ and $V$ be Banach spaces and $F : U \to V$ be a bounded linear operator, then for all $u \in U$, $F'(u)h = F(h)$.*

*Proof.* By linearity $(F(u + th) - F(u))/t = F(th)/t = F(h)$, thus $F(h)$ is the directional derivative of $F$ at $u$. As $F$ is bounded, $F$ is the Gâteaux derivative of $F$. By $F(u + h) - F(u) - F(h) = 0$, $F$ is also the Fréchet derivative. □

**Example 9.** Let $U = H^1(\Omega)$ and for a fixed $v \in U$ define $F(u) = \int \nabla u \cdot \nabla v\, dx$, then $F$ is a linear bounded operator from $U$ to $\mathbb{R}$ and by Lemma 8 $F$ is Fréchet differentiable with Fréchet derivative $F'(u)h = \int \nabla h \cdot \nabla v\, dx$. △

**Example 10.** Let $F : U \to V$ be Fréchet differentiable at $u \in U$ and let $L : \mathcal{D}(L) \to W$ be a bounded linear operator, then $L(F(\cdot))$ is Fréchet differentiable at $u \in U$ with Fréchet derivative $L(F'(u)h)$.

Consider the following derivation

$$\begin{aligned}\frac{\|L(F(u+h)) - L(F(u)) - L(F'(u)h)\|_W}{\|h\|_U} &= \frac{\|L(F(u+h) - F(u) - F'(u)h)\|_W}{\|h\|_U} \\ &\leq \|L\| \frac{\|F(u+h) - F(u) - F'(u)h\|_V}{\|h\|_U}.\end{aligned}$$

The last expression clearly goes to 0 as $\|h\|_U \to 0$ since $F$ was assumed Fréchet differentiable. △

## 2.3 Lagrangian and optimality conditions

In optimization theory, when one considers optimization problems with equality constraints, a powerful method for solving these problems is to consider an alternate problem incorporating the constraint. We consider here the general problem (2.1), but with the equality constraint $g(x) = 0$ in place of the inequality one.

$$\begin{aligned} \min_{x \in X} \quad & f(x), \\ \text{subj. to} \quad & g(x) = 0. \end{aligned} \tag{2.10}$$

For a problem like this we define the Lagrangian.

**Definition 11.** For the optimization problem (2.10) with $g : X \to Y$ we define the *Lagrangian* map $\mathcal{L} : X \times Y^* \to \mathbb{R}$ as

$$\mathcal{L}(x, \lambda) = f(x) - \langle \lambda, g(x) \rangle_{Y^*,Y}, \tag{2.11}$$

with $\lambda$ called the *Lagrange multiplier*.

In general $\lambda$ is an element of the dual space of $Y$. In the finite dimensional case, $Y = \mathbb{R}^n$, this simply makes $\langle \lambda, g(x) \rangle = \boldsymbol{\lambda}^T \mathbf{g}_x$ into a dot-product where $\mathbf{g}_x = g(x) \in \mathbb{R}^n$ and $\boldsymbol{\lambda}$ is the element corresponding to $\lambda$ in $\mathbb{R}^n$.

If we consider our general PDE-constrained problem in (2.5), the Langrangian naturally takes the form

$$\mathcal{L}(y, u, p) = J(y, u) - \langle p, e(y, u) \rangle_{W^*,W}, \tag{2.12}$$

where $p$ is our Lagrange multiplier.

The Lagrangian is a quite powerful tool as it gives us a single equation to optimize without considering any additional constraints. Indeed, a solution for (2.5) will be a stationary point for the Lagrangian; Appendix E in [Bis06] gives a detailed and very intuitive derivation for the finite dimensional case.

### 2.3.1 Lagrangian over Banach spaces

For the infinite dimensional case Chapter 4.14 in [Zei95] gives us the result we need.

**Lemma 12.** *Let $X$ and $W$ be real Banach spaces and $J : X \to \mathbb{R}$ and $e : X \to W$ Fréchet differentiable maps. Suppose $u \in e^{-1}(0) \subset X$ minimizes $J$ and $e'(u) : X \to W$ is surjective. Then there exists a functional $p \in W^*$ such that*

$$\mathcal{L}(u,p) = J'(u) - \langle p, e'(u)\rangle = 0.$$

**Remark 13.** For our problem $X = Y \times U$.

*Proof.* Following the steps in [Zei95]: Let $u \in e^{-1}(0) \subset X$ minimize $J$. Clearly $e(u) = 0$, we wish to show that $e'(u)h = 0$ implies $J'(u)h = 0$.

Let $h \in X$ be given such that $e'(u)h = 0$ and let

$$F(\varepsilon, v) := e(u + \varepsilon h + v),$$

for $(\varepsilon, v)$ in an open ball round $(0,0) \in \mathbb{R} \times X$. We note that

$$F(\varepsilon, v) - F(\varepsilon, 0) = e(u + \varepsilon h + v) - e(u + \varepsilon h) \to e'(u + \varepsilon h)v \quad \text{as } \|v\|_X \to 0,$$

since $e$ was assumed Fréchet differentiable, hence $F_v(\varepsilon, 0)v = e'(u+\varepsilon h)v$. As $F(0,0) = 0$ and $e'(u) = F_v(0,0)$ was assumed surjective, the implicit function theorem[Zei95, Section 4.13] tells us that there is an open neighborhood around $0 \in \mathbb{R}$ where $F(\varepsilon, v(\varepsilon)) = 0$ and thus $e(u + \varepsilon h + v(\varepsilon)) = 0$. By definition of the Fréchet derivative

$$e(u + k) = e'(u)k + \mathcal{O}(\|k\|),$$

thus setting $k = \varepsilon h + v(\varepsilon)$

$$\begin{aligned} 0 &= e(u + \varepsilon h + v(\varepsilon)) = \varepsilon e'(u)h + e'(u)v(\varepsilon) + \mathcal{O}(\|\varepsilon h + v(\varepsilon)) \\ 0 &= e'(u)v(\varepsilon) + \mathcal{O}(\|\varepsilon h + v(\varepsilon)), \end{aligned}$$

as $h$ was chosen such that $e'(u)h = 0$.

As $u$ was a minimizer, we have

$$J(u + \varepsilon h + v(\varepsilon)) \geq f(u).$$

Thus

$$J'(u)(\varepsilon h + v(\varepsilon)) + \mathcal{O}(\|\varepsilon h + v(\varepsilon\|)) \geq 0.$$

Letting $\varepsilon \to 0$ we get $v(\varepsilon) \to 0$ and thus $J'(u)h = 0$.

As we now have that $e'(u)h = 0$ with $h \in X$ implies $J'(u)h = 0$ we have

$$J'(u) \in N(e'(u))^{\perp}.$$

As $e(u)$ was Fréchet differentiable, $e'(u)$ is a bounded operator and since the domain of $e'(u)$ is all of $X$ it is closed by [Kre89, Theorem 4.13-5(a)]. By the closed range theorem[Yos80, p.205] $J'(u) \in R(e'(u)^*) = N(e'(u))^{\perp}$, thus there is a functional $p \in W^*$ such that $J'(u) = e'(u)p$, thus

$$\langle J'(u), h\rangle = \langle e'(u)^* p, h\rangle = \langle p, e'(u)h\rangle, \quad \text{for all } h \in X.$$

Thus $J'(u) - \langle p, e'(u)\rangle = 0$. □

This is a very useful property, as we may now instead seek the stationary points of the Lagrangian, which might in certain situations be much easier than solving our original problem.

### 2.3.2 First order optimality conditions

For a point $(\overline{y}, \overline{u}, \overline{p}) \in Y \times U \times W^*$ to be a stationary point of $\mathcal{L}(y, u, p)$ we require that the derivative with respect to each of the variables is zero.

$$\mathcal{L}'_y(\overline{y}, \overline{u}, \overline{p})h = J'_y(\overline{y}, \overline{u})h - \langle \overline{p}, e'_y(\overline{y}, \overline{u})h\rangle = 0 \qquad \forall h \in Y \tag{2.13}$$

$$\mathcal{L}'_u(\overline{y}, \overline{u}, \overline{p})w = J'_u(\overline{y}, \overline{u})w - \langle \overline{p}, e'_u(\overline{y}, \overline{u})w\rangle = 0 \qquad \forall w \in U \tag{2.14}$$

$$\mathcal{L}'_p(\overline{y}, \overline{u}, \overline{p}) = e(\overline{y}, \overline{u}) = 0. \tag{2.15}$$

We note that the last condition is simply the constraint for our problem.

These are called the *first order optimality conditions*. At times we also refer to them as the *Karush-Kuhn-Tucker conditions*, KKT-conditions. We derive the KKT-conditions for our model problem.

**Example 14.** We consider here the distributed control problem with mixed Dirichlet and Neumann boundary conditions where $Y = H^1(\Omega)$ and $U = L^2(\Omega)$.

$$\begin{aligned} \min_{(y,u)\in Y\times U} \quad & J(y,u) = \frac{1}{2}\int_\Omega (y-y_d)^2\,dx + \frac{\alpha}{2}\int_\Omega u^2\,dx \\ \text{subj. to} \quad & -\Delta y = u \quad \text{in } \Omega \\ & y = f \quad \text{on } \partial\Omega_D \\ & \tfrac{\partial y}{\partial n} = g \quad \text{on } \partial\Omega_N, \end{aligned} \tag{2.16}$$

From this we form the Lagrangian for the problem

$$\begin{aligned} \mathcal{L}(y,u,p) = & \frac{1}{2}\int_\Omega (y-y_d)^2\,dx + \frac{\alpha}{2}\int_\Omega u^2\,dx \\ & - \int_\Omega (-\Delta y - u)p_1\,dx - \int_{\partial\Omega_D} (y-f)p_2\,ds - \int_{\partial\Omega_N} \left(\frac{\partial y}{\partial n} - g\right) p_3\,ds. \end{aligned}$$

By the KKT-conditions (2.13)-(2.15) we derive a new set of equations. First, using (2.13) we derive the following: Let $h \in Y = H^1(\Omega)$ then

$$\begin{aligned} \mathcal{L}'_y(y,u,p)h &= \int_\Omega (y-y_d)h\,dx - \int_\Omega -\Delta h p_1\,dx - \int_{\partial\Omega_D} hp_2\,ds - \int_{\partial\Omega_N} \frac{\partial h}{\partial n}p_3\,ds \\ &= \int_\Omega (y-y_d)h\,dx + \int_\Omega h\Delta p_1\,dx - \int_{\partial\Omega} \frac{\partial h}{\partial n}p_1\,ds \\ &\quad + \int_{\partial\Omega} h\frac{\partial p_1}{\partial n}\,ds - \int_{\partial\Omega_D} hp_2\,ds - \int_{\partial\Omega_N} \frac{\partial h}{\partial n}p_3\,ds. \end{aligned}$$

Now, as $\mathcal{L}'_y(y,u,p) : Y \to \mathbb{R}$ we may pick $h \in Y$. If we first pick $h \in C_0^\infty(\Omega) \subseteq Y$ then the above expression along with $\mathcal{L}'_y(y,u,p) = 0$ yields

$$\int_\Omega (y-y_d)h\,dx + \int_\Omega h\Delta p_1\,dx = \int_\Omega \left((y-y_d) + \Delta p_1\right) h\,dx = 0,$$

which by the fundamental lemma of the calculus of variation (Lemma 34 in Appendix B) this gives us the following PDE-problem.

$$-\Delta p_1 = y - y_d \quad \text{in } \Omega. \tag{2.17}$$

By picking $h \in H_0^1(\Omega) \subseteq Y$

$$\int_{\partial\Omega} \frac{\partial h}{\partial n} p_1 \, ds + \int_{\partial\Omega_N} \frac{\partial h}{\partial n} p_3 \, ds = \int_{\partial\Omega} \frac{\partial h}{\partial n} (p_1 + \chi_{\partial\Omega_N} p_3) \, ds = 0,$$

thus

$$p_1|_{\partial\Omega_D} = 0 \quad \text{and} \quad p_1|_{\partial\Omega_N} = -p_3. \tag{2.18}$$

And from the remaining part (setting no restriction on how $h$ behaves on the boundary)

$$\int_{\partial\Omega} h \frac{\partial p_1}{\partial n} \, ds - \int_{\partial\Omega_D} h p_2 \, ds = \int_{\partial\Omega} h \left( \frac{\partial p_1}{\partial n} - \chi_{\partial\Omega_D} p_2 \right) ds = 0,$$

we get

$$\frac{\partial p_1}{\partial n}|_{\partial\Omega_N} = 0 \quad \text{and} \quad \frac{\partial p_1}{\partial n}|_{\partial\Omega_D} = p_2. \tag{2.19}$$

Clearly there is a direct relation between our Lagrange multiplier $p_1$, $p_2$ and $p_3$, hence we will simply consider only one multiplier $p = p_1$ and use the relations for $p_2$ and $p_3$ as necessary.

Equations (2.17)-(2.19) now combine into the adjoint equation with mixed boundary conditions

$$\begin{aligned} -\Delta p &= y - y_d \quad \text{in } \Omega \\ p &= 0 \quad \text{on } \partial\Omega_D \\ \frac{\partial p}{\partial n} &= 0 \quad \text{on } \partial\Omega_N. \end{aligned} \tag{2.20}$$

From here we move on to the second KKT-condition (2.14), which for $w \in U$ reads

$$\begin{aligned} \mathcal{L}'_u(y, u, p) w &= \alpha \int_\Omega uw \, dx - \int_\Omega -wp \, dx \\ &= \int_\Omega (\alpha u + p) w \, dx. \end{aligned}$$

By the fundamental lemma of the calculus of variation, $\mathcal{L}'_u(y,u,p) = 0$ yields that almost everywhere

$$\alpha u + p = 0. \tag{2.21}$$

As it was stated just after the KKT-conditions, the third and last of the KKT-conditions, is simply that our constraint $e(y,u) = 0$ has to be satisfied. Thus that the constraints in (2.16) is satisfied.

Summarizing, the KKT-conditions are realized by solving the following set of problems:

$$\begin{aligned}
&\textbf{State equation}: & -\Delta y &= u && \text{in } \Omega \\
& & y &= f && \text{on } \partial\Omega_D \\
& & \frac{\partial y}{\partial n} &= g && \text{on } \partial\Omega_N, \\
&\textbf{Adjoint equation}: & -\Delta p &= y - y_d && \text{in } \Omega \\
& & p &= 0 && \text{on } \partial\Omega_D \\
& & \frac{\partial p}{\partial n} &= 0 && \text{on } \partial\Omega_N, \\
&u\text{-}p\ \textbf{relation}: & \alpha u + p &= 0
\end{aligned}$$

△

## 2.4 Discretization

When we wish to solve optimization problems like the ones we have described so far the most common approach involves computers, and since computers operate in discrete domains, what we do is to discretize our problem to a finite dimensional setting, which the computer can handle. Obviously we don't discretize blindly, but rather do it in a fashion that ensures our discretized solution will converge towards the solution of our original problem as the discretization is refined.

Several techniques for discretization exist, but in this thesis we will use the *finite element method* (FEM) in particular for our discretization. This method is well-known and very commonly used today when people want to solve partial differential equations [Eng09]. Since we will essentially be doing this every time we wish to take a step in a direction for a better solution, this seems like a sound idea.

### 2.4.1 The basics of the finite element method

In the finite element method the domain, $\Omega$, of our problem is triangulated. Of course if $\Omega$ does not have a boundary stitched together of straight edges, this procedure will result in a new domain $\Omega_h$ which will roughly resemble $\Omega$ depending on the size of the triangles. It is quite common to refine the triangles profoundly near curved borders to better match the shapes.

For most model problem, however, the domain will be a square and can thus be directly triangulated without loss of detail. In this case $\Omega_h = \Omega$.

A triangulation like this will, obviously, yield a number of triangles, say $N$. Each of these sports three edges and vertices shared by one or more triangles. We label each vertex in the triangulation $\{x_1, x_2, \dots, x_M\}$, $M$ being the total number of vertices.

From here we pick a set of basis functions from the function space relevant to our problem. For example, if we seek to approximate $y \in H^1(\Omega)$ we pick our basis functions $\{\phi_1, \phi_2, \dots, \phi_M\} \subset H^1(\Omega)$. We choose each $\phi_i$ such that

$$\phi_i(x_j) = \begin{cases} 1 & i = j \\ 0 & i \neq j, \end{cases}$$

where $x_j$ is the $j$'th vertex. Often additional conditions are layered on top as different aspects in a PDE-problem may be captured directly in the basis functions.

**Example 15.** A choice of basis functions could be first order polynomial functions, $P_1 = \{p \,|\, p(x) = a_0 + a_1 x_1 + a_2 x_2\}$. Consider the domain $\Omega = ]-1,1[\, \times \,]-1,1[$ with a triangulation as follows. Split the domain into 4 squares separated by the axes, and divide each square into two triangles using lower left to upper right diagonals as illustrated in Figure 2.1. Labeling the vertex left to right, row by row, starting with the bottom row, we consider now $\phi_5$, the central basis function. This one will be defined on all the triangles sharing the vertex $x_5 = (0,0)$. Each triangle will have its own basis function, for example

$$\phi_5(x,y) = 1 - x, \quad (x,y) \in \Delta(x_5, x_8, x_9),$$

where $\Delta(x_5, x_8, x_9)$ defines the triangle with corners $x_5$, $x_8$ and $x_9$. For $\Delta(x_4, x_5, x_8)$ we have

$$\phi_5(x,y) = 1 - x + y, \quad (x,y) \in \Delta(x_4, x_5, x_8).$$

The entire basis function can be seen in Figure 2.2.

Another class of basis functions that should be mentioned is the polynomials of partial first order, $Q_1 = \{p \,|\, p(x) = a_0 + a_1 x_1 + a_2 x_2 + a_3 x_1 x_2\}$. These are of

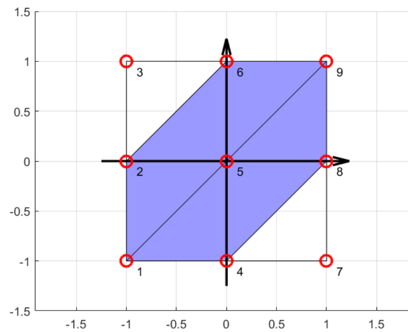


**Figure 2.1:** Simple FEM grid.

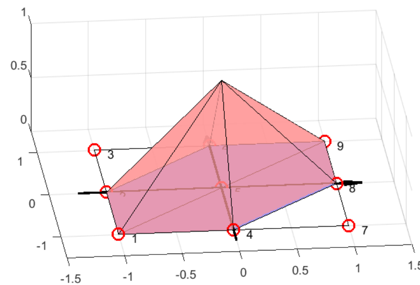


**Figure 2.2:** $P_1$ FEM basis function.

relevance when using rectangular finite elements. We have above described finite elements as triangulation, and while that is the general approach, we sometimes have the choice of separating our domain completely into rectangles instead of triangles. This is obviously not convenient for domains with curvy borders, but for rectangles, L-shapes and generally "block-*ish*" domains it will work just fine.

A $Q_1$ basis function on our previously described domain (albeit without the diagonals) is illustrated in Figure 2.3. Note how this function is only affine parallel to the axes.

△

The space of all linear combinations of the basis functions $Y_h = \text{span}\{\phi_1, \phi_2, \dots, \phi_M\}$ is a finite dimensional subspace of our original space $H^1(\Omega)$. We may consider func-

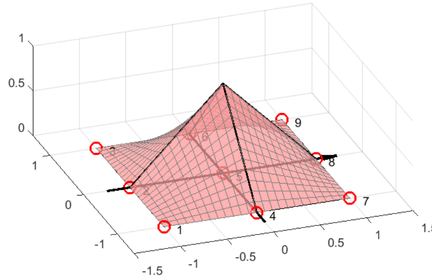


**Figure 2.3:** $Q_1$ FEM basis function.

tions in this space $y_h = \sum_{i=1}^{M} Y_i \phi_i \in Y_h$ and separately we may consider the coefficient vector $\mathbf{y} = (Y_1, Y_2, \ldots, Y_M) \in \mathbb{R}^M$.

Note that while we didn't do this here, sometimes people label first the inner nodes and then the boundary nodes afterwards. Letting $n$ be the number of interior nodes and $n_\partial$ be the number of boundary nodes. Then we write

$$\{x_1, x_2, \ldots, x_n, x_{n+1}, \ldots, x_{n+n_\partial}\} \text{ and } \{\phi_1, \phi_2, \ldots, \phi_n, \phi_{n+1}, \ldots, \phi_{n+n_\partial}\}.$$

Given the problem of solving some PDE-problem $\mathcal{L}y = f$, our problem now shifts from finding $y \in Y$, which is very hard in general, to finding $\mathbf{y} \in \mathbb{R}^M$ such that $y_h$ solves the problem or at least closely approximates a solution. This might simply seem like a crude restatement of the problem, but being reduced to finite dimensions allows us to express the problem as a problem in linear algebra which is very well understood compared to the function space setting.

### 2.4.2 Discretizing the problem

We consider here again the distributed control problem with mixed boundary conditions (2.16) which we introduced in Example 14. We restate it here: Let $Y = H^1(\Omega)$ and $U = L^2(\Omega)$, then the problem is

$$
\begin{aligned}
\min_{(y,u)\in Y\times U} \quad & J(y,u) = \frac{1}{2}\int_\Omega (y-y_d)^2\,dx + \frac{\alpha}{2}\int_\Omega u^2\,dx \\
\text{subj. to} \quad & -\Delta y = u \quad \text{in } \Omega \\
& y = f \quad \text{on } \partial\Omega_D \\
& \tfrac{\partial y}{\partial n} = g \quad \text{on } \partial\Omega_N.
\end{aligned}
$$

Holding off on the boundary conditions for now; in the example we derived the following set of equations from the KKT-conditions,

$$
\begin{aligned}
-\Delta y &= u \\
-\Delta p &= y_d - y \\
\alpha u - p &= 0.
\end{aligned}
$$

*Note that we have switched the sign of p here compared to the example.* This will not affect the solution, except for an obvious change in sign, however, it will help produce a symmetric matrix system in the end.

The first step will be to write up the weak formulation of each of these problems. We recall that

$$
\int_\Omega (-\Delta y)v\,dx = \int_\Omega \nabla y \nabla v\,dx - \int_{\partial\Omega} \frac{\partial y}{\partial n} v\,ds \tag{2.22}
$$

and thus for $v \in C_0^\infty(\Omega)$ we easily get

$$
\int_\Omega \nabla y \nabla v\,dx = \int_\Omega uv\,dx \tag{2.23}
$$

$$
\int_\Omega \nabla p \nabla v\,dx + \int_\Omega yv\,dx = \int_\Omega y_d v\,dx, \tag{2.24}
$$

and similarly for the last condition,

$$
\alpha \int_\Omega uv\,dx - \int_\Omega pv\,dx = 0. \tag{2.25}
$$

Let $\{\phi_1, \phi_2, \dots, \phi_n, \phi_{n+1}, \dots, \phi_{n+n_\partial}\}$ be our FEM basis, with $i = 1, \dots, n$ denoting the basis functions on interior nodes and $i = n+1, \dots, n+n_\partial$ the basis functions on boundary nodes. We then get the following approximations

$$y_h = \sum_{i=1}^{n} y_i \phi_i + \sum_{i=n+1}^{n+n_\partial} y_i \phi_i, \tag{2.26}$$

$$u_h = \sum_{i=1}^{n} u_i \phi_i + \sum_{i=n+1}^{n+n_\partial} u_i \phi_i, \tag{2.27}$$

and

$$p_h = \sum_{i=1}^{n} p_i \phi_i + \sum_{i=n+1}^{n+n_\partial} p_i \phi_i. \tag{2.28}$$

It should be noted that while we use the same basis functions for the approximation of both our state and control as well as Lagrange multiplier, there could be situations where it would make sense to pick different basis for each. For solvability reasons this is beyond our scope, though.

We now substitute (2.26)-(2.28) into (2.23)-(2.25). Before we go further and write out the result, we will also make a choice for $v$. Picking $v = \phi_i$ for $i \in \{1, \ldots, n\}$ we get a sequence of equations while satisfying the boundary conditions for $v$. While $\phi_i$ is not necessarily $C^\infty$ – for instance neither $P_1$ nor $Q_1$ elements are – the problem only occur on a set of measure zero, so we will disregard that. From this we obtain

$$\int_\Omega \nabla y_h \nabla \phi_j \, dx - \int_\Omega u_h \phi_j \, dx = 0, \tag{2.29}$$

$$\int_\Omega \nabla p_h \nabla \phi_j \, dx + \int_\Omega y_h \phi_j \, dx = \int_\Omega y_d \phi_j \, dx, \tag{2.30}$$

$$\alpha \int_\Omega u_h \phi_j \, dx - \int_\Omega p_h \phi_j \, dx = 0, \tag{2.31}$$

for $j = 1, \ldots, n$. By expanding $y_h$, $u_h$ and $p_h$ we get the following systems of equations

$$\sum_{i=1}^{n+n_\partial} y_i \int_\Omega \nabla \phi_i \nabla \phi_j \, dx - \sum_{i=1}^{n+n_\partial} u_i \int_\Omega \phi_i \phi_j \, dx = 0, \tag{2.32}$$

$$\sum_{i=1}^{n+n_\partial} p_i \int_\Omega \nabla \phi_i \nabla \phi_j \, dx + \sum_{i=1}^{n+n_\partial} y_i \int_\Omega \phi_i \phi_j \, dx = \int_\Omega y_d \phi_j \, dx, \tag{2.33}$$

$$\alpha \sum_{i=1}^{n+n_\partial} u_i \int_\Omega \phi_i \phi_j \, dx - \sum_{i=1}^{n+n_\partial} p_i \int_\Omega \phi_i \phi_j \, dx = 0, \tag{2.34}$$

for $j = 1, \ldots, n$. Defining the mass matrix $M$ and stiffness matrix $K$ by

$$M_{ij} = \int_\Omega \phi_i \phi_j \, dx \qquad \text{and} \qquad K_{ij} = \int_\Omega \nabla\phi_i \nabla\phi_j \, dx, \tag{2.35}$$

we may write the above system of equation as matrix equations.

$$\begin{aligned} K\mathbf{y} - M\mathbf{u} &= 0, \\ K\mathbf{p} + M\mathbf{y} &= \mathbf{y_d}, \\ \alpha M\mathbf{u} - M\mathbf{p} &= 0, \end{aligned}$$

where $\mathbf{y} = (y_1, \ldots, y_{n+n_\partial})$, $\mathbf{u} = (u_1, \ldots, u_{n+n_\partial})$, $\mathbf{p} = (p_1, \ldots, p_{n+n_\partial})$ and $(\mathbf{y_d})_j = \int_\Omega y_d \phi_j \, dx$.

Now, there is a slight problem here. We would like $M$ and $K$ to be symmetric matrices, but right now they are both $n \times (n+n\partial)$-matrices. An easy fix to this would be to substitute in $\phi_{n+1}, \ldots, \phi_{n+n_\partial}$ on $v$'s spot in the weak formulations, however, we don't have non-zero boundary anymore. By integration by parts (2.22) we get an additional contribution from the Neumann condition. Setting

$$(\mathbf{g})_i = \int_{\partial\Omega_N} \frac{\partial y}{\partial n} \phi_i \, dx = \int_{\partial\Omega_N} g\phi_i \, dx$$

and for now assuming $\frac{\partial y}{\partial n}|_{\partial\Omega_D} = 0$, we extend our matrix system to $(n+n_\partial) \times (n+n_\partial)$-matrices in the obvious way thereby obtaining the following system

$$K\mathbf{y} - M\mathbf{u} = \mathbf{g}, \tag{2.36}$$

$$K\mathbf{p} + M\mathbf{y} = \mathbf{y_d}, \tag{2.37}$$

$$\alpha M\mathbf{u} - M\mathbf{p} = 0. \tag{2.38}$$

Obviously $\mathbf{y_d}$ has been extended to the boundary nodes as well here.

### 2.4.3 Applying Dirichlet boundary conditions

In finite element theory there are algorithms for applying Dirichlet boundary conditions. What is technically done in a discretized PDE-equation $K\mathbf{y} = \mathbf{f}$ the diagonal elements in $K$ corresponding to boundary nodes are modified to 1 and all other elements in those rows are set to 0. Furthermore, the values in the vector $\mathbf{f}$ corresponding to the boundary nodes are set to the known boundary value. Then for symmetry reasons the boundary terms are all moves to the right hand side of the equation. This

is done by modifying $\mathbf{f}$ and setting the corresponding element in $K$ to 0. While this explanation my be slightly confusing, consider the following example demonstrating the modifications.

**Example 16.** Let us for example assume that $x_2$ is the boundary node in a simple 4-node system and $b$ is the known value at $x_2$. Then given the FEM matrix equation

$$\begin{bmatrix} k_{1,1} & k_{1,2} & k_{1,3} & k_{1,4} \\ k_{2,1} & k_{2,2} & k_{2,3} & k_{2,4} \\ k_{3,1} & k_{3,2} & k_{3,3} & k_{3,4} \\ k_{4,1} & k_{4,2} & k_{4,3} & k_{4,4} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix},$$

the modification would be

$$\begin{bmatrix} k_{1,1} & 0 & k_{1,3} & k_{1,4} \\ 0 & 1 & 0 & 0 \\ k_{3,1} & 0 & k_{3,3} & k_{3,4} \\ k_{4,1} & 0 & k_{4,3} & k_{4,4} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} f_1 - k_{1,2}b \\ b \\ f_3 - k_{3,2}b \\ f_4 - k_{4,2}b \end{bmatrix}.$$

△

A good question here is how we apply this to our situation. Our problem is not a simple matrix equation, but rather a system of equations involving several matrices. However, our approach will be very similar. In the example above we considered a matrix and right hand side $(K, f)$ being modified into a new matrix and new right hand side $(\widehat{K}, \widehat{f})$. Let us denote this operation by $\mathcal{BC}$.

We first notice that $p|_{\partial\Omega_D} = 0$ and $y|_{\partial\Omega_D} = f$. Define $(\mathbf{f})_i = \int_{\partial\Omega_D} f\phi_i \, dx$, then clearly for each boundary node $x_i \in \partial\Omega_D$ we must have $p_i = 0$ and $y_i = (\mathbf{f})_i$. Furthermore, for a non-boundary node $x_i \notin \partial\Omega_D$ the matrix equation (2.37) yields

$$\begin{aligned} &\sum_j M_{ij}y_j + \sum_j K_{ij}p_j = (\mathbf{y_d})_i \\ &\sum_{x_j \notin \partial\Omega_D} M_{ij}y_j + \sum_j K_{ij}p_j = (\mathbf{y_d})_i - \sum_{x_j \in \partial\Omega_D} M_{ij}y_j = (\mathbf{y_d})_i - \sum_{x_j \in \partial\Omega_D} M_{ij}(\mathbf{f})_j. \end{aligned}$$

We note that we could have limited the sum of $K_{ij}p_j$ as well, but we already established that $p_j$ already was zero in those positions.

From this observation, it seems reasonable that we should use $(\widehat{M}, \widehat{\mathbf{y_d}}) = \mathcal{BC}(M, \mathbf{y_d})$ and moreover that we should be applying boundary conditions to $K$ in some way as well.

Considering first matrix equation (2.38), then since we apply boundary conditions to $M$ we obviously find that no a boundary node $x_i \in \partial\Omega_D$ $\alpha u_i = p_i = 0$, thus $u|_{\partial\Omega_D} = 0$. Proceeding to equation (2.36) we get for a non-boundary node $x_i \notin \partial\Omega_D$

$$\sum_j K_{ij} y_j - \sum_j M_{ij} u_j = (\mathbf{g})_i$$
$$\sum_{x_j \notin \partial\Omega_D} K_{ij} y_j - \sum_j M_{ij} u_j = (\mathbf{g})_i - \sum_{x_j \in \partial\Omega_D} K_{ij} y_j = (\mathbf{g})_i - \sum_{x_j \in \partial\Omega_D} K_{ij} (\mathbf{f})_j.$$

Again, here we need not limit the sum over the $u$'s as we just established the relevant entries to be zero as with the $p$'s. From this it seems like we apply the boundary condition to the matrix $K$ and vector $\mathbf{g}$. However, as $\mathbf{g}$ represent our Neumann boundary conditions, it is nice to not mix the two. Moreover, $\mathbf{g}$ is already zero in every position we would overwrite, since we overwrite on nodes in $\partial\Omega_D$ and $\mathbf{g}$ is non-zero only on $\partial\Omega_N$. So we may simply apply our boundary conditions $(\widehat{K}, \mathbf{d}) = \mathcal{BC}(K, \mathbf{0})$, and just add the vector $\mathbf{d}$ to $\mathbf{g}$. Note how this on the boundary nodes $x_i \in \partial\Omega_D$ simply reestablishes $y_i - u_i = y_i - 0 = (\mathbf{g})_i + (\mathbf{f})_i = 0 + (\mathbf{f})_i$, thus the equations $y_i = (\mathbf{f})_i$ actually appear twice in our system.

Futhermore, we will simply redefine $M := \widehat{M}$, $K = \widehat{K}$ and $\mathbf{y_d} = \widehat{\mathbf{y_d}}$, since what we want to solve is the problem *with* the boundary conditions.

We can combine the matrix equations (2.36)-(2.38) with the boundary conditions now

$$\begin{bmatrix} M & 0 & K \\ 0 & \alpha M & -M \\ K & -M & 0 \end{bmatrix} \begin{bmatrix} \mathbf{y} \\ \mathbf{u} \\ \mathbf{p} \end{bmatrix} = \begin{bmatrix} \mathbf{y_d} \\ \mathbf{0} \\ \mathbf{g} + \mathbf{d} \end{bmatrix}. \tag{2.39}$$

### 2.4.4 Zero boundary conditions

Another strategy for handling boundary conditions is the idea of incorporating them into the space we pick the basis functions from, for instance the space $V_h$ defined by

$$V_h := \operatorname{span}\{\phi_1, \phi_2, \ldots, \phi_n\}$$

corresponds to zero Dirichlet boundary conditions on all of $\partial\Omega$. However, this only really work for zero boundary conditions as superposition of two functions $y_1$ and $y_2$ both satisfying a Dirichlet condition $y_1(x) = y_2(x) = \alpha$ at a point $x \in \partial\Omega$ will obviously satisfy $y_1(x) + y_2(x) = 2\alpha$ at $x$, which is not the same boundary condition unless $\alpha = 0$. A similar observation can be made for the Neumann boundary condition.

So to use this technique requires both the Dirichlet and Neumann boundary conditions to be zero. Luckily we can reformulate a problem into a zero-boundary condition problem. Consider the PDE-problem from our model problem

$$\begin{aligned} -\Delta y &= u && \text{in } \Omega \\ y &= f && \text{on } \partial\Omega_D \\ \frac{\partial y}{\partial n} &= g && \text{on } \partial\Omega_N, \end{aligned} \tag{2.40}$$

and let $\overline{y}$ be the solution to the following PDE-problem with boundary conditions

$$\begin{aligned} -\Delta \overline{y} &= 0 && \text{in } \Omega \\ \overline{y} &= -f && \text{on } \partial\Omega_D \\ \frac{\partial \overline{y}}{\partial n} &= -g && \text{on } \partial\Omega_N. \end{aligned}$$

Let $y$ solve the PDE-problem (2.40) but with zero boundary conditions, then $\widehat{y} = y - \overline{y}$ solves (2.40). We note that since only $y$ depend on $u$, we need only solve the zero-boundary condition version of (2.40) to find $\widehat{y}$ for a new value of $u$.

Considering this in relation to our model problem, we will be minimizing the functional $J(\widehat{y}, u)$, however,

$$\begin{aligned} J(\widehat{y}, u) &= \frac{1}{2} \int_\Omega (\widehat{y} - y_d)^2 \, dx + \frac{\alpha}{2} \int_\Omega u^2 \, dx \\ &= \frac{1}{2} \int_\Omega (y - (\overline{y} + y_d))^2 \, dx + \frac{\alpha}{2} \int_\Omega u^2 \, dx = \widehat{J}(y, u). \end{aligned}$$

The difference between $J$ and $\widehat{J}$ is simply the desired state. Thus defining $\widehat{y}_d = \overline{y} + y_d$ as our new desired state, we obtain a formulation of our model problem where the PDE part of the problem has zero boundary conditions.

Since we have seen here how we may reformulate our model problem as a variant with 0 boundary condition, we will consider only zero boundary conditions for the remainder of the thesis.

This does not render the previous sections on boundary conditions obsolete though. By applying the zero Dirichlet boundary conditions as discussed in the previous section, we are in fact making sure our functions $y_h, u_h$ and $p_h$ come from a vector space with the appropriate boundary conditions. Let

$$\widehat{V}_h := \text{span}\{\phi_1, \phi_2, \dots, \phi_n, \phi_{n+1}, \dots, \phi_{n+n_\partial}\},$$

then applying the Dirichlet conditions is equivalent to making sure our functions are from the space

$$V_h' := \left\{ f = \sum_{i=1}^{n+n_\partial} f_i \phi_i \,\middle|\, f_i = 0 \text{ when the node } x_i \in \partial\Omega_D \right\}.$$

This space satisfies $V_h \subseteq V_h' \subseteq \widehat{V}_h$.

## 2.5 Matrix Theory

The previous section ended with the discretization finalizing into a system of matrix equations. This system has structural properties that we will take advantage of. As it turns out it belongs to a particular class of matrix equations, the *saddle point systems*, which has been studied extensively in their general form [Her10; Zul11; BGL05].

In this section we briefly touch upon saddle point systems and follow up with some theory on the Schur complement, which we will need in the following sections.

### 2.5.1 Saddle point systems

A saddle point system is a matrix problem where the system matrix has the particular block structure.

$$\begin{bmatrix} A & B_1^T \\ B_2 & -C \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \mathbf{g} \end{bmatrix}. \tag{2.41}$$

Here $A \in \mathbb{R}^{n\times n}$, $B_1, B_2 \in \mathbb{R}^{n\times m}$ and $C \in \mathbb{R}^{m\times m}$ are all matrices and $\mathbf{x}, \mathbf{f} \in \mathbb{R}^n$ and $\mathbf{y}, \mathbf{g} \in \mathbb{R}^m$ vectors. Obviously, many matrix systems can be split into a block partitioning like this. Thus we require something more before we say (2.41) is a saddle point problem.

[BGL05] requires first of all $A$ non-zero or both of $B_1$ and $B_2$ to be non-zero. Additionally, five properties are listed of which one or more should be true. We list them here for convenience

(P1) $A = A^T$, ($A$ is symmetric),

(P2) $H = \frac{1}{2}(A + A^T) \geq 0$, ($H$ is positive semidefinite),

(P3) $B_1 = B_2 = B$,

(P4) $C = C^T \geq 0$, ($C$ is symmetric positive semidefinite),

(P5) $C = 0$, ($C$ is the zero matrix).

Considering our matrix problem (2.39) from earlier, we group the system matrix as follows

$$A = \begin{bmatrix} M & 0 \\ 0 & \alpha M \end{bmatrix}, \quad B_1 = B_2 = \begin{bmatrix} K & -M \end{bmatrix}, \quad \text{and} \quad C = 0. \tag{2.42}$$

Clearly (P3) and (P5) are satisfied, but in fact all five properties are satisfied. The mass matrix $M_{ij} = \int_\Omega \phi_i \phi_j \, dx$, so clearly $M$ is symmetric, making $A$ symmetric (P1). This means $H = A$ and since $M$ is positive definite by Lemma 29 in Appendix B and $\alpha > 0$, so is $A$ and thus $H$ (P3). (P4) is actually as obvious as (P5) since clearly $0 = 0^T \geq 0$.

### 2.5.2 The Schur complement

Here we will discuss some of the theory regarding the Schur complement. Consider the following block matrix

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix},$$

with $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{m\times n}$ and $D \in \mathbb{R}^{m\times m}$, and note that when $D$ is invertible it allows for the following LDU-decomposition (*lower-diagonal-upper* decomposition)

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} I_n & BD^{-1} \\ 0 & I_m \end{bmatrix} \begin{bmatrix} A - BD^{-1}C & 0 \\ 0 & D \end{bmatrix} \begin{bmatrix} I_n & 0 \\ D^{-1}C & I_m \end{bmatrix}.$$

We say that $S_D := A - BD^{-1}C$ is the *Schur complement* of $D$. When $A$ is invertible straight forward calculation yields the analogous decomposition

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} I_n & 0 \\ CA^{-1} & I_m \end{bmatrix} \begin{bmatrix} A & 0 \\ 0 & D - CA^{-1}B \end{bmatrix} \begin{bmatrix} I_n & A^{-1}B \\ 0 & I_m \end{bmatrix}.$$

We denote the Schur complement of $A$ by $S_A := D - CA^{-1}B$. One really nice property of this decomposition is that lower and upper triangular matrices inverts easily.

**Lemma 17.** *Let*

$$\begin{bmatrix} I_n & X \\ 0 & I_m \end{bmatrix}$$

*be an upper triangular matrix. Then its inverse is*

$$\begin{bmatrix} I_n & -X \\ 0 & I_m \end{bmatrix}.$$

*For a lower triangular matrix the result is analogous.*

*Proof.* Consider the following computation and comparison

$$\begin{bmatrix} I_n & X \\ 0 & I_m \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} E + XG & F + XH \\ G & H \end{bmatrix} = \begin{bmatrix} I_n & 0 \\ 0 & I_m \end{bmatrix}.$$

This implies $H = I_m$ and $G = 0$, which in turn yields $E = I_n$ and $F = -X$. The result for lower triangles follows from the simple observation

$$\begin{bmatrix} I_n & 0 \\ X & I_m \end{bmatrix} = P \begin{bmatrix} I_n & X \\ 0 & I_m \end{bmatrix} P, \qquad \text{where} \quad P = \begin{bmatrix} 0 & I_n \\ I_m & 0 \end{bmatrix}.$$

Note that $P^2 = I$. □

#### 2.5.2.1 Positive definiteness

For a Hermitian matrix $M$ (in our case that means $C = B^T$) by [Zha05, Theorem 1.6] there is a connection between the eigenvalues of $M$ and the eigenvalues of $A$ and $S_A$. What the theorem says is that the number of positive eigenvalues of $M$, $p(M)$ is equal to the total number of positive eigenvalues in $A$ and $S_A$ together, i.e. $p(A) + p(S_A)$. Likewise for negative eigenvalues and eigenvalues zero.

An immediate consequence of this is that if $M$ is positive definite then so must both $A$ and $S_A$ be. Similar for positive semi-definite, though we must require $A$ positive definite to have $S_A$. This result is summarized without proof in [Zha05] in Theorem 1.12. We state the result here and write out the proof using [Zha05, Theorem 1.6].

**Lemma 18.** *Let*

$$M = \begin{bmatrix} A & B \\ B^* & C \end{bmatrix}$$

*be a block Hermitian matrix with $A$ square and invertible. Then*

*(i)* $M > 0$ *if and only if* $A > 0$ *and* $S_A > 0$.

*(ii)* $M \geq 0$ *if and only if* $A > 0$ *and* $S_A \geq 0$.

**Remark 19.** Note that $B^*$ here denotes the Hermitian transpose, that is conjugate transpose $B^* = \overline{B}^T$.

*Proof.* Let $p(M)$ denote the number of positive eigenvalues of $M$, $q(M)$ the number of negative eigenvalues of $M$ and $z(M)$ the number of zero eigenvalues of $M$. By [Zha05, Theorem 1.6] we have $p(M) = p(A) + p(S_A)$, $q(M) = q(A) + q(S_A)$ and $z(M) = z(A) + z(S_A)$.

(i) If $M > 0$, then $q(M) = z(M) = 0$, thus $q(A) = q(S_A) = 0$ and $z(A)+z(S_A) = 0$ , hence $A > 0$ and $S_A > 0$. From this the reverse statement is trivial.

(ii) If $M \geq 0$, then $q(M) = 0$, thus $q(A) = q(S_A) = 0$. Since $A$ was assumed invertible we have $z(A) = 0$ and therefore $A > 0$. Moreover $z(M) = z(S_A)$, thus $S_A \geq 0$. Again the reverse statement follows trivially from the same considerations. □

## 2.6 Preconditioning

In this section, we introduce the concept of preconditioners. We first define the basic ingredients before we formally define what a preconditioner is.

### 2.6.1 Condition number

When solving linear system such as

$$\mathcal{A}x = f, \tag{2.43}$$

where $\mathcal{A} : X \to Y$ is some operator, $f$ is the known and $x$ is our unknown, using iterative methods a key quantity of interest is the *condition number* of $\mathcal{A}$. We often denote the condition number of $\mathcal{A}$ by $\kappa(\mathcal{A})$.

For a matrix $A$ the condition number $\kappa(A)$ is defined by the matrix norm $\kappa(A) = \|A\|\|A^{-1}\|$, where $\|A\| = \sup_{\mathbf{x}\neq\mathbf{0}} \|A\mathbf{x}\|/\|\mathbf{x}\|$. In analogous fashion we extend the condition number to more general operators $\mathcal{A} : X \to Y$, i.e. $\kappa(\mathcal{A}) = \|\mathcal{A}\|\|\mathcal{A}^{-1}\|$, with the operator norm $\|\mathcal{A}\| = \sup_{x\neq 0} \|\mathcal{A}x\|_Y/\|x\|_X$.

The condition number is interesting because it figures in the error estimate when iteratively solving problems such as (2.43). For a matrix system we can always compute the condition number as they are always bounded by their largest eigenvalue, however, general operators might not be bounded.

What we are interested in is to have a conditioning number as close to 1 as possible, but since we were probably given our problem, we can't simply ask for a new operator with better spectral properties.

Given a problem such as (2.43) we say that $\mathcal{P} : X \to Y$ is a *preconditioner* for $\mathcal{A}$ if $\mathcal{P}^{-1}\mathcal{A}$ has better spectral properties than $\mathcal{A}$. The way we measure this is by the condition number.

### 2.6.2 Riesz isomorphism

In certain cases, such as for our PDE-constrained optimization problem, one might consider operators $\mathcal{A}$ as going from a space $H$ to its dual space $H^*$. If $H$ is a Hilbert space, then the map $\mathcal{I}_H : H \to H^*$ defined by

$$\langle \mathcal{I}_H x, y \rangle = (x, y)_H,$$

could be a possible choice as a preconditioner.

$\mathcal{I}_H$ is the inverse of the Riesz isomorphism, $\mathcal{R}_H = \mathcal{I}_H^{-1}$. That is the famous map coming from the Riesz representation theorem[Kre89, Theorem 3.8-1], linking each element of the dual space $x^* \in H^*$ to an element $x$ in the Hilbert space $H$ bijectively, such that $\langle x^*, y \rangle_{H^*,H} = (x, y)_H$.

In [GHS14] it is noted that the Riesz representation takes on the role as a preconditioner, which Zulehner explores in more detail in [Zul11]. We will here summarize some of the content Zulehner presents in his article.

First off, Zulehner considers the saddle point problem in a more general form: Find $(u, p) \in X = V \times Q$ such that

$$\begin{aligned} a(u, v) + b(v, p) &= f(v), \quad \text{for all } v \in V, \\ b(u, q) - c(p, q) &= g(q), \quad \text{for all } q \in Q, \end{aligned} \tag{2.44}$$

where $V$ and $Q$ are Hilbert spaces, $a : V \times V \to \mathbb{C}$, $b : V \times Q \to \mathbb{C}$ and $c : Q \times Q \to \mathbb{C}$ are bounded bilinear forms, and $f, g$ are bounded linear functionals. Moreover, $a$ and $c$ are assumed symmetric and nonnegative.

Now, a bilinear form $s : H \times H \to \mathbb{C}$ always comes with a associated operator $S : H \to H^*$ defined by the relation $s(x, y) = \langle Sx, y \rangle$ and vice versa. For $a$, $b$ and $c$ we denote their associated operators $A : V \to V^*$, $B : V \to Q^*$ and $C : Q \to Q^*$. Each of these operators are bounded since $a$, $b$ and $c$ were.

Using these (2.44) may be restated as a problem of equality in the dual space $X^*$:

$$\begin{aligned} Au + B^*p &= f, \\ Bu - Cp &= g, \end{aligned}$$

where $B^*$ is the adjoint of $B$ defined by $\langle B^*r, v\rangle = \langle Bv, r\rangle$ for $r \in Q^*$ and $v \in V$.

The problem (2.44) may be restated in the following short hand form: Find $x = (u, p) \in X$ such that

$$\mathcal{B}(x, y) = \mathcal{F}(y) \tag{2.45}$$

for all $y = (v, q) \in X$, with

$$\mathcal{B}(x, y) = a(u, v) + b(v, p) + b(u, q) - c(p, q) \quad \text{and} \quad \mathcal{F}(y) = f(v) + g(q).$$

Like we did with the bilinear forms $a$, $b$ and $c$, we may consider the operator $\mathcal{A} : X \to X^*$ associated to $\mathcal{B}$, such that $\langle \mathcal{A}x, y\rangle = \mathcal{B}(x, y)$. Then the problem rephrases to finding $x \in X$ such that $\mathcal{A}x = \mathcal{F}$.

As each of the bilinear forms were bounded it is trivial to see that there is some constant $\overline{c}_x > 0$ such that

$$\sup_{0 \neq z \in X} \sup_{0 \neq y \in X} \frac{\mathcal{B}(z, y)}{\|z\|_X \|y\|_X} \leq \overline{c}_x < \infty.$$

By the inf-sub condition by Ladyzhenskaya, Babuška and Brezzi[Zul11] the problem (2.45) is well-posed if and only if

$$\inf_{0 \neq z \in X} \sup_{0 \neq y \in X} \frac{\mathcal{B}(z, y)}{\|z\|_X \|y\|_X} \geq \underline{c}_x > 0.$$

From these conditions, if $0 \neq x \in X$ is a solution to (2.45) we have

$$\underline{c}_x \leq \frac{\|\mathcal{F}\|_{X^*}}{\|x\|_X} = \sup_{0 \neq y \in X} \frac{|\mathcal{F}(y)|}{\|x\|_X \|y\|_X} = \sup_{0 \neq y \in X} \frac{|\mathcal{B}(x, y)|}{\|x\|_X \|y\|_X} \leq \overline{c}_x.$$

Thus we obtain the condition

$$\overline{c}_x^{-1} \|\mathcal{F}\|_{X^*} \leq \|x\|_X \leq \underline{c}_x^{-1} \|\mathcal{F}\|_{X^*}.$$

In terms of our operator $\mathcal{A}$ this condition paraphrases into

$$\underline{c}_x \|z\|_X \leq \|\mathcal{A}z\|_{X^*} \leq \overline{c}_x \|z\|_X \quad \text{for all } z \in X.$$

As it turns out these bounds $\underline{c}_x$ and $\overline{c}_x$ on our operator $\mathcal{A}$ has a direct relation to the afore mentioned condition number for $\kappa(\mathcal{A})$. Note that as for $\|\mathcal{F}\|_{X^*}$ we have $\|\mathcal{A}z\|_{X^*} = \sup_{0\neq y\in X} |\langle \mathcal{A}z, y\rangle|/\|y\|_X$, thus

$$\|\mathcal{A}\| = \sup_{0\neq z\in X} \frac{\|\mathcal{A}z\|_{X^*}}{\|z\|_X} = \sup_{0\neq z\in X} \sup_{0\neq y\in X} \frac{|\langle \mathcal{A}z, y\rangle|}{\|z\|_X\|y\|_X} = \sup_{0\neq z\in X} \sup_{0\neq y\in X} \frac{|\mathcal{B}(z,y)|}{\|z\|_X\|y\|_X} \leq \overline{c}_x.$$

Furthermore,

$$\begin{aligned}\|\mathcal{A}^{-1}\| &= \sup_{0\neq r\in \mathrm{Range}(\mathcal{A})\subseteq X^*} \frac{\|\mathcal{A}^{-1}r\|_X}{\|r\|_{X^*}} = \sup_{0\neq z\in X} \frac{\|z\|_X}{\|\mathcal{A}z\|_{X^*}} = \left(\inf_{0\neq z\in X} \frac{\|Az\|_{X^*}}{\|z\|_X}\right)^{-1} \\ &= \left(\inf_{0\neq z\in X} \sup_{0\neq y\in X} \frac{|\langle \mathcal{A}z, y\rangle|}{\|z\|_X\|y\|_X}\right)^{-1} = \left(\inf_{0\neq z\in X} \sup_{0\neq y\in X} \frac{|\mathcal{B}(z,y)|}{\|z\|_X\|y\|_X}\right)^{-1} \leq \underline{c}_x^{-1}.\end{aligned}$$

Thus the bounds on $\mathcal{A}$ yields the following bound on the condition number of $\mathcal{A}$:

$$\kappa(\mathcal{A}) = \|\mathcal{A}\|\|\mathcal{A}^{-1}\| \leq \frac{\overline{c}_x}{\underline{c}_x}.$$

Note that during the entire process nothing has been said explicitly about the inner products on $V$ and $Q$. The aim in [Zul11] is to find inner products for $V$ and $Q$ such that one might obtain bounds $\underline{c}_x$ and $\overline{c}_x$ for the operator $\mathcal{A}$. Good bounds like that will lead to nice condition numbers and thus good convergence properties for iterative methods.

The main theorem in [Zul11] then states the following

**Theorem 20** (Theorem 2.6 in [Zul11])**.** *If there are constants* $\underline{\gamma}_v, \overline{\gamma}_v, \underline{\gamma}_q, \overline{\gamma}_q > 0$ *such that*

$$\underline{\gamma}_v\|w\|_V^2 \leq a(w,w) + \|Bw\|_{Q^*}^2 \leq \overline{\gamma}_v\|w\|_V^2 \quad \textit{for all } w\in V$$

*and*

$$\underline{\gamma}_q\|r\|_Q^2 \leq c(r,r) + \|B^*r\|_{V^*}^2 \leq \overline{\gamma}_q\|r\|_Q^2 \quad \textit{for all } r\in Q,$$

*then*

$$\underline{c}_x\|z\|_X \leq \|\mathcal{A}z\|_{X^*} \leq \overline{c}_x\|z\|_X \quad \textit{for all } z\in X$$

*is satisfied with constants* $\underline{c}_x, \overline{c}_x > 0$ *that depend only on* $\underline{\gamma}_v, \overline{\gamma}_v, \underline{\gamma}_q$ *and* $\overline{\gamma}_q$*. The reverse conclusion holds true as well.*

**Remark 21.** By reading the proofs we get the following link between the constants: $\overline{c}_x = \sqrt{2}\max(\overline{c}_v, \overline{c}_q)$, with $\overline{c}_v^2 = \max(\overline{\gamma}_v, 1)\overline{\gamma}_v$ and $\overline{c}_q$ following the same pattern.

For the other,

$$\underline{c}_x = \frac{3-\sqrt{5}}{4}\frac{\underline{c}_v^2 + \underline{c}_q^2}{\max(\overline{c}_v, \overline{c}_q)},$$

with $\underline{c}_v = \min(\underline{\gamma}_v, 1)\underline{\gamma}_v$ and $\underline{c}_q$ analogously determined.

We note that these are just particular choices from the proofs, not necessarily the the best bounds.

Rewriting the two conditions in the theorem one obtains

$$\underline{\gamma}_v\langle \mathcal{I}_V w, w\rangle \leq \langle (A + B^*\mathcal{I}_Q^{-1}B)w, w\rangle \leq \overline{\gamma}_v\langle \mathcal{I}_V w, w\rangle, \quad \text{for all } w \in V,$$

and

$$\underline{\gamma}_q\langle \mathcal{I}_Q r, r\rangle \leq \langle (C + B\mathcal{I}_V^{-1}B^*)r, r\rangle \leq \overline{\gamma}_q\langle \mathcal{I}_Q r, r\rangle, \quad \text{for all } r \in Q,$$

which shortens to

$$I_V \sim A + B^*\mathcal{I}_Q^{-1}B \quad \text{and} \quad I_Q \sim C + B\mathcal{I}_V^{-1}B^*. \tag{2.46}$$

**Lemma 22.** *Equation* (2.46) *is equivalent to both of the following statements*

*(i)* $\mathcal{I}_V \sim A + B^*(C + B\mathcal{I}_V^{-1}B^*)^{-1}B$ *and* $\mathcal{I}_Q \sim C + B\mathcal{I}_V^{-1}B^*$,

*(ii)* $\mathcal{I}_Q \sim C + B(A + B^*\mathcal{I}_Q^{-1}B)^{-1}B^*$ *and* $I_V \sim A + B^*\mathcal{I}_Q^{-1}B$.

*Proof.* We will only prove (i) here as (ii) follows by analogous considerations.

(i) By equation (2.46) we get the following two inequalities:

$$\mathcal{I}_Q \leq c_1(C + B\mathcal{I}_V^{-1}B^*) \quad \text{and} \quad C + B\mathcal{I}_V^{-1}B^* \leq c_2\mathcal{I}_Q.$$

Note that $\mathcal{I}_V, \mathcal{I}_Q > 0$, thus $(C + B\mathcal{I}_V^{-1}B^*) > 0$ and it is invertible. Moreove, both $c_1, c_2 > 0$. By Lemma 33 the two inequalities yields

$$\mathcal{I}_Q^{-1} \geq c_1^{-1}(C + B\mathcal{I}_V^{-1}B^*)^{-1} \quad \text{and} \quad c_2(C + B\mathcal{I}_V^{-1}B^*)^{-1} \geq \mathcal{I}_Q^{-1}.$$

From these

$$\begin{aligned}\mathcal{I}_V &\le c_3(A + B^*\mathcal{I}_Q^{-1}B)\\ &\le c_3(A + c_2 B^*(C + B\mathcal{I}_V^{-1}B^*)^{-1}B)\\ &\le c_2 \max(1, c_2)(A + B^*(C + B\mathcal{I}_V^{-1}B^*)^{-1}B),\end{aligned}$$

and

$$\begin{aligned}\mathcal{I}_V &\ge c_4^{-1}(A + B^*\mathcal{I}_Q^{-1}B)\\ &\ge c_4^{-1}(A + c_1^{-1}B^*(C + B\mathcal{I}_V^{-1}B^*)^{-1}B)\\ &\ge c_4^{-1}\min(1, c_1^{-1})(A + B^*(C + B\mathcal{I}_V^{-1}B^*)^{-1}B).\end{aligned}$$

Hence

$$\mathcal{I}_V \sim A + B^*(C + B\mathcal{I}_V^{-1}B^*)^{-1}B.$$

The other way follows from similar derivations. □

### 2.6.3 Finite dimensional problems

Consider now the two statements in Lemma 22. If we take only the first part of either statement, say (i), then the only unknown is $\mathcal{I}_V$ and if we can find this we pretty much have an expression for $\mathcal{I}_Q$. Thus our problem has taken the following form: Find an operator $\mathcal{I}_V : V \to V^*$ which satisfy the first operator relation in Lemma 22(i). Then define $\mathcal{I}_Q$ using the second operator relation in Lemma 22(i).

Zulehner then considers the special case when $V$ and $Q$ are finite vector spaces. In this case the saddle point system is a matrix equation

$$\begin{bmatrix} A & B^T \\ B & -C \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \mathbf{g} \end{bmatrix}$$

and the the inner product is defined by the matrix

$$\mathcal{I}_X = \begin{bmatrix} \mathcal{I}_V & 0 \\ 0 & \mathcal{I}_Q \end{bmatrix}.$$

For the case where $A$ and the negative Schur complement of $A$, $S = C + BA^{-1}B^T$, are non-singular it turns out the choices $\mathcal{I}_V = A$ and $\mathcal{I}_Q = S = C + BA^{-1}B^T$ solves the operator relations in Lemma 22(i) as we shall see. Analogously, if $C$ and $S' = A + B^T C^{-1} B$ are non-singular similar considerations show that they solve the relation in Lemma 22(ii).

We note first how the equation for $\mathcal{I}_Q$ is trivially satisfied with $\underline{\gamma}_q = \overline{\gamma}_q = 1$. To see the over relation holds we first consider the matrix

$$M = \begin{bmatrix} S & B \\ B^T & A \end{bmatrix},$$

in which the Schur complement of $A$ is $S - BA^{-1}B^T = C$. Since $C$ is positive semi-definite and $A$ is positive definite by Lemma 18, $M$ must be positive semi-definite. Applying the lemma again, since $S$ is positive definite and $M$ is positive semi-definite the Schur complement of $S$ in $M$, call it $\widehat{S}$, must be positive semi-definite. By definite $\widehat{S} = A - B^T S^{-1} B \geq 0$, hence

$$A \geq B^T S^{-1} B = B^T (C + BA^{-1}B^T)^{-1} B.$$

Using this

$$\mathcal{I}_V = A \leq A + B^T I_Q^{-1} B = A + B^T (C + BA^{-1}B^T)^{-1} B \leq A + A = 2A = 2\mathcal{I}_V,$$

thus the operator relation is satisfied with $\underline{\gamma}_v = 1$ and $\overline{\gamma}_v = 2$. Hence we have robust estimates for our problem.

**Example 23.** We consider now our problem from Equation (2.39) with the matrix blocks as defined in Equation (2.42). Then $A$ block is positive definite as the mass matrix $M$ is positive definite by Lemma 29.

The negative Schur complement is

$$S = 0 + \begin{bmatrix} K & -M \end{bmatrix} \begin{bmatrix} M & 0 \\ 0 & \alpha M \end{bmatrix}^{-1} \begin{bmatrix} K \\ -M \end{bmatrix} = KM^{-1}K + \alpha^{-1} M.$$

We already established that $M$ was positive definite, thus all eigenvalues $\lambda_i$ are positive. Let $\mathbf{v}_i$ be the eigenvector for $M$ corresponding to $\lambda_i$, then

$$M\mathbf{v}_i = \lambda_i \mathbf{v}_i \quad \Longleftrightarrow \quad \frac{1}{\lambda_i}\mathbf{v}_i = M^{-1}\mathbf{v}_i,$$

hence $\lambda_i^{-1}$ is the eigenvalue for $M^{-1}$ corresponding to eigenvector $\mathbf{v}_i$. Thus $M$ and $M^{-1}$ share the same eigenvectors but with reciprocal eigenvalues. But if $\lambda_i > 0$, then $\lambda_i^{-1} > 0$ as well, hence $M^{-1}$ is also positive definite.

As $K$ is symmetric we find that $\mathbf{v}^T K M^{-1} K \mathbf{v} = (K\mathbf{v})^T M^{-1} (K\mathbf{v}) \geq 0$, thus $KM^{-1}K$ is positive semi-definite. Ergo we have $KM^{-1}K \geq 0$ and $\alpha^{-1}M > 0$, and

thus $S = KM^{-1}K + \alpha^{-1}M > 0$. So $S$ is positive definite as well, hence we may here use the preconditioner considered above:

$$P = \begin{bmatrix} M & 0 & 0 \\ 0 & \alpha M & 0 \\ 0 & 0 & S \end{bmatrix},$$

where $S = KM^{-1}K + \alpha^{-1}M$. These blocks corresponds to inner products induced on our function space $V_h$. Clearly the state, control and Lagrange multiplier receive different inner products. Let $Y_h$, $U_h$ and $P_h$ each be the function space $V_h$ but with the different inner products. Let us explore what their inner products are.

In the following $u$ and $v$ will be the finite element approximated functions and $\mathbf{u}$ and $\mathbf{v}$ their corresponding coefficient vectors. For $Y_h$ we get $(u,v)_{Y_h} = \mathbf{v}^T M\mathbf{u}$ and for $U_h$ we get $(u,v)_{U_h} = \alpha\mathbf{v}^T M\mathbf{u}$. As $M$ corresponds to the $L^2$ inner product $\mathcal{I}_{L^2}$, we have

$$\begin{aligned} (u,v)_{Y_h} &= (u,v)_{L^2} \quad \text{and} \\ (u,v)_{U_h} &= \alpha(u,v)_{L^2}. \end{aligned}$$

For $P_h$ it is slightly more complicated.

$$\begin{aligned} (u,v)_{P_h} &= \mathbf{v}^T S\mathbf{u} = \mathbf{v}^T(KM^{-1}K + \alpha^{-1}M)\mathbf{u} \\ &= \mathbf{v}^T KM^{-1}K\mathbf{u} + \alpha^{-1}\mathbf{v}^T M\mathbf{u} = (K\mathbf{v})^T M^{-1}(K\mathbf{u}) + \alpha^{-1}\mathbf{v}^T M\mathbf{u}. \end{aligned}$$

As $M$ corresponds to $\mathcal{I}_{L^2} : L^2 \to (L^2)^*$, $M^{-1}$ must correspond to $\mathcal{R}_{L^2} : (L^2)^* \to L^2$, and as $K$ corresponds to $-\Delta : H^1 \to (H^1)^*$ we may, in a sense, view $(K\mathbf{v})^T M^{-1}(K\mathbf{u})$ as the induced inner product of $\Delta u$ and $\Delta v$ in $(L^2)^*$. That is

$$(u,v)_{P_h} = \alpha^{-1}(u,v)_{L^2} + (\Delta u, \Delta v)_{(L^2)^*}.$$

△

Zulehner proceeds to show that for the case where both blocks $A$ and $C$ are positive definite, we have for each $\theta \in [0,1]$ solutions to the operator relations

$$\mathcal{I}_V = A + [A, B^T C^{-1} B]_\theta \quad \text{and} \quad \mathcal{I}_Q = C + [C, BA^{-1}B^T]_{1-\theta}, \tag{2.47}$$

where

$$[M,N]_\theta = M^{1/2}(M^{-1/2}NM^{-1/2})^\theta M^{1/2}.$$

Note that for $\theta = 0$ and $\theta = 1$ we obtain the previously mentioned solutions up to a scaling with a factor of 2.

**Example 24.** We consider again our problem given in Equation (2.39). The second matrix equation derives from the equation $\alpha u - p = 0$ which ultimately came from the KKT-conditions derived in Example 14. Rearranging this we get $u = \alpha^{-1}p$ and substituting $\alpha^{-1}p$ for $u$ everywhere we reduce the system to two coupled partial differential equations in the unknowns $y$ and $p$, here listed without their boundary conditions

$$\begin{aligned} -\Delta y &= \alpha^{-1}p, \text{and} \\ -\Delta p &= y_d - y. \end{aligned}$$

Discretization of this yields the matrix system

$$\begin{bmatrix} M & K \\ K & -\alpha^{-1}M \end{bmatrix} \begin{bmatrix} \mathbf{y} \\ \mathbf{p} \end{bmatrix} = \begin{bmatrix} \mathbf{y_d} \\ \mathbf{0} \end{bmatrix}.$$

Again, not taking boundary conditions into account here. They are applied to this system in a fashion analogous to what was seen earlier. We won't consider them further here.

Selecting the matrix blocks in the obvious way

$$A = M, \quad B = K \quad \text{and} \quad C = \alpha^{-1}M,$$

satisfy the condition that $A$ and $C$ are positive definite matrices. Zulehner's preconditioner stated in Equation (2.47) are then

$$\begin{aligned} \mathcal{I}_V &= M + [M, K(\alpha^{-1}M)^{-1}K]_\theta = M + \alpha^\theta [M, KM^{-1}K]_\theta, \quad \text{and} \\ \mathcal{I}_Q &= \alpha^{-1}M + [\alpha^{-1}M, KM^{-1}K]_{1-\theta} = \alpha^{-1}M + \alpha^{-\theta}[M, KM^{-1}K]_{1-\theta}. \end{aligned}$$

Notably,

$$\begin{aligned} [M, KM^{-1}K]_\theta &= M^{1/2}(M^{-1/2}KM^{-1}KM^{-1/2})^\theta M^{1/2} \\ &= M^{1/2}(M^{-1/2}KM^{-1/2})^{2\theta} M^{1/2}. \end{aligned}$$

Thus for $\theta = 1 - \theta = \frac{1}{2}$ we have $[M, KM^{-1}K]_{1/2} = K$ and may obtain the following simple forms for the preconditions, which Zulehner also shows.

$$\begin{aligned} \mathcal{I}_V &= M + [M, K(\alpha^{-1}M)^{-1}K]_{1/2} = M + \alpha^{1/2}K, \quad \text{and} \\ \mathcal{I}_Q &= \alpha^{-1}M + [\alpha^{-1}M, KM^{-1}K]_{1/2} = \alpha^{-1}M + \alpha^{-1/2}K. \end{aligned}$$

These are particularly nice as when $M$ and $K$ are very sparse matrices, the sum will most often also be sparse, which is a useful property. Furthermore, neither of them require inversion of a matrix, which means the quantities can easily be computed.

As in the previous example they correspond to the inner products on $Y_h$ and $P_h$

$$(u,v)_{Y_h} = (u,v)_{L^2} + \alpha^{1/2}(\nabla u, \nabla v)_{L^2}, \quad \text{and,}$$
$$(u,v)_{P_h} = \alpha^{-1}(u,v)_{L^2} + \alpha^{-1/2}(\nabla u, \nabla v)_{L^2}.$$

△

### 2.6.4 Schur complement approximations

The examples in the previous section showed us possible choices as preconditioners for our problem. The first example involved the negative Schur complement $S = C + BA^{-1}B^T$, which there takes the form $S = KM^{-1}K + \alpha^{-1}M$.

It is relevant in computations with large matrix systems to consider the efficiency of operations in speed but also in memory usage. As a finite element mass matrix $M$ will generally be quite sparse, which is great for memory. Also, often the structure of a finite element matrix like this may be exploited for fast computations. Alas, though $M$ is sparse, there is no guarantee $M^{-1}$ will be sparse. Thus $S$ will not necessarily be sparse or easily inverted.

Recall that while we have considered preconditioners $P$, what we actually use it for is to consider $P^{-1}$ applied to our system matrix. Thus we prefer $P$ to be easily invertible. Being a block diagonal matrix $P$ inverts by inverting each of the blocks diagonal elements. This require us to invert $S$, which as mentioned might be computationally very expensive.

As such approximations $\widehat{S}$ of $S$ are usually considered instead. In [RDW10] the following approximation to $S = KM^{-1}K + \alpha^{-1}M$ is proposed

$$\widehat{S}_1 = KM^{-1}K.$$

We simply drop the $\alpha^{-1}M$ term. The argument here is that for all but very small values of $\alpha$ the $KM^{-1}K$ term will be the dominating one. This approximation is nice as it inverts quite easily. It comes down to the problem of solving $K\mathbf{x} = \mathbf{b}$ for $\mathbf{x}$ two times and $M\mathbf{x} = \mathbf{b}$ for $\mathbf{b}$ once, which are both sparse problems. It is worth noting here that this approximation $\widehat{S}_1$ has no $\alpha$ dependence anymore. The logical conclusion should be that the upper and lower bounds previously established must depend on $\alpha$ now, which the numerical results will also verify. Also, $\widehat{S}_1$ is not necessarily positive definite.

By rewriting $S$ we find that

$$\begin{aligned} S &= KM^{-1}K + \alpha^{-1}M = KM^{-1}K + (\sqrt{\alpha}^{-1}M)M^{-1}(\sqrt{\alpha}^{-1}M) \\ &= KM^{-1}K + (\sqrt{\alpha}^{-1}M)M^{-1}(\sqrt{\alpha}^{-1}M) + 2\sqrt{\alpha^{-1}}K - 2\sqrt{\alpha}^{-1}K \\ &= (K + \sqrt{\alpha}^{-1}M)M^{-1}(K + \sqrt{\alpha}^{-1}M) - 2\sqrt{\alpha^{-1}}K. \end{aligned}$$

In [Pea13] this is used for proposing another approximation to the Schur complement. The proposed approximation is

$$\widehat{S}_2 = (K + \sqrt{\alpha}^{-1}M)M^{-1}(K + \sqrt{\alpha}^{-1}M).$$

That is, we drop the $-2\sqrt{\alpha}^{-1}K$ term. Again the argument is that the term will be dominated by $\widehat{S}_2$ for most values of $\alpha$. Furthermore, where the term removed for $\widehat{S}_1$ was $\mathcal{O}(\alpha^{-1})$ here we have $\mathcal{O}(\sqrt{\alpha}^{-1})$, so the error committed grows much slower.

$\widehat{S}_2$ luckily have the same conveniences as $\widehat{S}_1$ had. Defining $L = K + \sqrt{\alpha}^{-1}M$, inverting $\widehat{S}_2$ is equivalent to solving $L\mathbf{x} = \mathbf{b}$ for $\mathbf{x}$ two times and $M\mathbf{x} = \mathbf{b}$ for $\mathbf{b}$ once. Also, $M$ and $K$ being sparse means $L$ is often a sparse matrix. As an added bonus $\widehat{S}_2$ does maintain its $\alpha$-dependency, and as such the bounds should be expected to remain fairly good.

We finally note here that using the approximations $\widehat{S}_1$ and $\widehat{S}_2$ of course signifies usage of slightly different inner products as well. The correspond to the inner products

$$(u, v)_{\widehat{S}_1} = (\Delta u, \Delta v)_{(L^2)^*}$$

and

$$(u, v)_{\widehat{S}_2} = \alpha(u, v)_{L^2} + (\Delta u, \Delta v)_{(L^2)^*} + 2\sqrt{\alpha}^{-1}(\nabla u, \nabla v)_{L^2}$$

respectively.

### 2.6.5 Preconditioner for infinite dimensional case

A reasonable question coming from the abstract setting of Zulehner's article is: Can we consider our problem in the non-discretized operator setting and establish a preconditioner operator? Also, will this preconditioner operator discretize to the same preconditioner we found for the discretized system?

In this section we lay the foundation for a possible extension of the study in previous sections to preconditioners in the infinite dimensional space.

We have a notion that a generalization of the Schur-complement to operator matrices on Hilbert spaces might make derivations analogous to the finite dimensional setting from the theory possible. We present here initial thoughts on this process.

#### 2.6.5.1 Schur complement for operator matrices

Let $X$ and $Y$ be Hilbert spaces and $A : X \to X$, $B : Y \to X$, $C : X \to Y$ and $D : Y \to Y$ be bounded Hilbert space operators. Let $I_H : H \to H$ be the identity operator on $H$. We will consider the Hilbert space operator $M : H \to H$, where $H = X \times Y$, defined by

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix},$$

We begin by noting that upper triangular operator matrices behave a lot like regular matrices. They are invertible in the same way.

$$\begin{bmatrix} I_X & B \\ 0 & I_Y \end{bmatrix} \begin{bmatrix} I_X & -B \\ 0 & I_Y \end{bmatrix} = \begin{bmatrix} I_X I_X + B0 & I_X(-B) + BI_Y \\ 0I_X + I_Y 0 & 0(-B) + I_Y I_Y \end{bmatrix} = \begin{bmatrix} I_X & 0 \\ 0 & I_Y \end{bmatrix}.$$

Note how for any operator $B : X \to Y$ the upper triangular operator matrix is easily inverted by a change in sign.

Let $D$ be invertible, then the following computation can be done. Let $L, R : H \to H$ be triangular operator matrices, defined as follows

$$L = \begin{bmatrix} I_X & -BD^{-1} \\ 0 & I_Y \end{bmatrix} \quad \text{and} \quad R = \begin{bmatrix} I_X & 0 \\ -D^{-1}C & I_Y \end{bmatrix}$$

We then compute

$$\begin{aligned} LMR &= \begin{bmatrix} I_X & -BD^{-1} \\ 0 & I_Y \end{bmatrix} \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} I_X & 0 \\ -D^{-1}C & I_Y \end{bmatrix} \\ &= \begin{bmatrix} I_X A + (-BD^{-1})C & I_X B + (-BD^{-1})D \\ 0A + I_Y C & 0B + I_Y D \end{bmatrix} \begin{bmatrix} I_X & 0 \\ -D^{-1}C & I_Y \end{bmatrix} \\ &= \begin{bmatrix} A - BD^{-1}C & 0 \\ C & D \end{bmatrix} \begin{bmatrix} I_X & 0 \\ -D^{-1}C & I_Y \end{bmatrix} \\ &= \begin{bmatrix} (A - BD^{-1}C)I_X + 0(-D^{-1})C & (A - BD^{-1}C)0 + 0I_Y \\ CI_X + D(-D^{-1}C) & C0 + DI_Y \end{bmatrix} \\ &= \begin{bmatrix} A - BD^{-1}C & 0 \\ 0 & D \end{bmatrix}. \end{aligned}$$

Here the operator $S = A - BD^{-1}C : X \to X$ is the Schur complement operator for the opeator $D$ in $M$.

Sylvester's Law of Inertia[Zha05, Theorem 1.5] states that for matrices $A$ and $B$ there is a non-singular matrix $G$ such that $A = G^*BG$ if and only if $A$ and $B$ have the same number of positive, negative and zero eigenvalues.

In Section 2.5.2 we saw how positive definiteness of the Schur complement can be linked to the same property for the matrix blocks in the finite dimensional setting. In [Zha05] this is done using Sylvesters's Law of Inertia. By [Bun88; Cai80] the intertia law also holds for operators in infinite dimensional Hilbert spaces. In infinite dimensions the comparison of eigenvalues takes on a more complicated form involving dimensions of eigenspaces, see [Cai80, p. 225], which is beyond the scope here.

By the same approach as in [Zha05, Theorem 1.6], but using the generalized Hilbert space version of the Law of Inertia we can obtain a relation between the between the positive, negative and zero eigenvalues of $M$, $D$ and $S$, the Schur complement of $D$ in $M$.

Let $M$ be a Hermitian operator, i.e. $M = M^*$, where $M^*$ is the adjoint operator. Then

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix}^* = \begin{bmatrix} A^* & C^* \\ B^* & D^* \end{bmatrix} = M^*,$$

and thus $A = A^*$, $D = D^*$ and $B = C^*$. Relabeling, we from here write

$$M = \begin{bmatrix} A & B \\ B^* & D \end{bmatrix}.$$

Note that now $L$ and $R$ are adjoints to each other.

$$L^* = \begin{bmatrix} I_X & -BD^{-1} \\ 0 & I_Y \end{bmatrix}^* = \begin{bmatrix} I_X^* & 0 \\ (-BD^{-1})^* & I_Y^* \end{bmatrix} = \begin{bmatrix} I_X & 0 \\ -D^{-1}B^* & I_Y \end{bmatrix} = R$$

Recall that $M > 0$ if $(Mh, h) > 0$ for all $h \in H$. If a negative eigenvalue $\lambda$ with corresponding eigenvector $h'$ exists, then $(Mh', h') = \lambda(h', h') = \lambda\|h'\|^2 < 0$, thus $M \not> 0$. Hence if $M > 0$, $M$ can have no negative eigenvalues and the dimension of the negative signed eigenspace must be 0. Likewise for the 0 eigenspace.

Since

$$(LMRh, h) = (MRh, L^*h) = (MRh, Rh) > 0$$

we find that $LMR > 0$. Consider now any $x \in X$ and construct $h = (x, 0) \in H$, then $(LMRh, h) = (Sx, x) > 0$, where $S$ was the Schur complement, hence $S > 0$. Constructing $h = (0, y)$ for $y \in Y$ analogously shows $D > 0$.

The other way is obvious for $D > 0$ and $S > 0$ we have for $h \in H$

$$(Mh, h) = (L^{-1}\Lambda R^{-1}h, h) = (\Lambda R^{-1}h, R^{-1}h) = (Sx, x) + (Dy, y) > 0,$$

where $\Lambda = LMR$ and $R^{-1}h = (x, y)$. Thus $M > 0$.

We conclude that we may obtain a generalized Lemma 18 for Hilbert space operators.

**Lemma 25.** *Let $X, Y$ and $H = X \times Y$ be Hilbet spaces and $M : H \to H$ a Hermitian operator defined by*

$$M = \begin{bmatrix} A & B \\ B^* & C \end{bmatrix},$$

*where $A : X \to X$, $B : Y \to X$ and $C : Y \to Y$ are Hilbert space operators. Furthermore, assume $C$ to be invertible. Then $M > 0$ if and only if $C > 0$ and $S_C = A + BC^{-1}B^* > 0$, where $S_C$ is the Schur complement.*

#### 2.6.5.2 Continuous problem

In this section we consider, based on the theory from the previous section, the saddle point problem

$$M \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} v \\ w \end{bmatrix}, \quad \text{with, } M = \begin{bmatrix} A & B \\ B^* & -C \end{bmatrix}.$$

Here $A : X \to X$, $B : Y \to X$ and $C : Y \to Y$. Assuming $A$ invertible and $S = C + B^*A^{-1}B$ invertible, then following the derivation for the finite dimensional case seen in the Theory chapter we may obtain a robust preconditioner of the form

$$P = \begin{bmatrix} P_1 & 0 \\ 0 & P_2 \end{bmatrix},$$

with bounds $\underline{\gamma}_x = 1$, $\overline{\gamma}_x = 2$ and $\underline{\gamma}_y = \overline{\gamma}_y = 1$ by setting $P_1 = A$ and $P_2 = S = C + B^*A^{-1}B$.

Now the real problem is determining if the following operator

$$M = \begin{bmatrix} I_{L^2} & 0 & -\Delta \\ 0 & \alpha I_{L^2} & I_{L^2} \\ -\Delta & I_{L^2} & 0 \end{bmatrix},$$

where $\Delta$ is the Laplacian, $I_{L^2}$ is the indentity operator on $L^2$ and $\alpha > 0$, is a Hermitian operator. Here we have the following break-down of sub-operators $A$, $B$ and $C$,

$$A = \begin{bmatrix} I_{L^2} & 0 \\ 0 & \alpha I_{L^2} \end{bmatrix}, \quad B = \begin{bmatrix} -\Delta \\ I_{L^2} \end{bmatrix}, \quad C = \begin{bmatrix} 0 \end{bmatrix}.$$

For $M$ to be Hermitian we saw earlier that $A$ and $C$ should be Hermition and for $B$ we require

$$B = \begin{bmatrix} -\Delta \\ I_{L^2} \end{bmatrix}^* = \begin{bmatrix} -\Delta^* & I_{L^2}^* \end{bmatrix} = \begin{bmatrix} -\Delta & I_{L^2} \end{bmatrix}.$$

Clearly both $A$ and $C$ are Hermitian operators. $C$ is trivial and $A$ is easy because the identity operator is Hermitian on $L^2$, which is not altered by scaling with a real constant. However, $B$ is troublesome. We need $-\Delta$ to be Hermitian, i.e. self-adjoint.

However, $-\Delta$ is not just self-adjoint. This depends a lot on our space, so let us briefly outline where the problem came from. We are working over a domain $\Omega$ with boundary $\partial\Omega = \partial\Omega_D \cup \partial\Omega_N$, where $\partial\Omega_D \cap \partial\Omega_N = \emptyset$. Furthermore our functions are all in $L^2(\Omega)$, though there might be restrictions to a subspace where certain levels of derivatives exist. We will now consider realizations of $-\Delta$.

For in depth theory of realizations see [Gru08, Chp. 4]. Suffices here to say that realizations of an differential operator $A$ are particular extensions of $A_{\min} := \overline{A|_{C_0^\infty(\Omega)}}^{\|\cdot\|_G}$, where $\overline{\phantom{A}}^{\|\cdot\|_G}$ means the closure of the operator in the Graph norm. We seek here a self-adjoint realization; that is, a self-adjoint extension. Note that there is a maximal realization $A_{\max}$ which acts as $A$ but in the sense of distributions. All realizations $\tilde{A}$ of $A$ satisfy $A_{\min} \subset \tilde{A} \subset A_{\max}$.

There are many self-adjoint realizations of $-\Delta$, typically corresponding to different boundary conditions. In our problem we consider the boundary conditions $u = 0$ on $\partial\Omega_D$ and $\frac{\partial u}{\partial n} = 0$ on $\partial\Omega_N$.

Let $\gamma_0 : H^1(\Omega) \to L^2(\partial\Omega)$ be the trace operator and define $\gamma_D : H^1(\Omega) \to L^2(\partial\Omega_D)$ by $\gamma_D(u) = \gamma_0(u)\chi_{\partial\Omega_D}$. Let

$$V := \{u \in H^1(\Omega) \,|\, \gamma_D u = 0\},$$

then $V$ is a closed subspace of $H^1(\Omega)$ and thus a Hilbert space itself as we shall see next.

Consider a Cauchy-sequence $(u_n) \subset V$, then exists $u \in H^1(\Omega)$ such that $u_n \to u$ in $H^1(\Omega)$. By [Eva08, Chp. 5.5] the trace operator $\gamma_0$ is bounded, and so is $\gamma_D$:

$$C\|u\|_{H^1(\Omega)} \geq \|\gamma_0(u)\|_{L^2(\partial\Omega)} \geq \|\gamma_D(u)\|_{L^2(\partial\Omega_D)}.$$

As $\gamma_D$ is bounded it is continuous and thus $u_n \to u$ in $H^1(\Omega)$ implies $\gamma_D(u_n) \to \gamma_D(u)$ in $L^2(\partial\Omega_D)$. Since $\gamma_D(u_n) = 0$ by definition we have $0 \to \gamma_D(u)$ in $L^2(\partial\Omega_D)$, hence $\gamma_D(u) = 0$ meaning $u \in V$, which is what we wanted.

Let $s(u, v) = \sum_{k=1}^{n} (\partial_k u, \partial_k v)$ be a biliniar form on $V$.

Note, $s$ is $V$-coercive:

$$s(v,v) = \sum_{k=1}^{n} (\partial_k v, \partial_k v) = \sum_{k=1}^{n} \|\partial_k v\|^2 \geq 0,$$

thus

$$\mathrm{Re}(s(v,v)) + (v,v) \geq (v,v), \quad \text{for all } v \in V.$$

Consider the triplet $(L^2(\Omega), V, s)$, this satisfies the conditions for [Gru08, Corollary 12.19] which we state as a lemma here.

**Lemma 26** (Corollary 12.19 in [Gru08])**.** *Let $(H, V, a)$ be a triple where $H$ and $V$ are complex Hilbert spaces with $V \subset H$ algebraically, topologically and densely, and where $a$ is a bounded sesquilinear form on $V$ with $D(a) = V$. Let $A$ be the operator associated with $a$ in $H$.*

*When $a$ is $V$-coercive, then $A$ is a closed operator with $D(A)$ dense in $H$ and in $V$. Moreover, the operator associated with $a^*$ in $H$ equals $A^*$ which has the same properties as listed for $A$. In particular, if $a$ is symmetricm $A$ is selfadjoint.*

**Remark 27.** $V \subset H$ algebraically, topologically and densely means simply that the norm of $V$ is stronger than the norm of $H$ in the sense that

$$\|v\|_V \geq c\|v\|_H \quad \text{for all } v \in V,$$

where $c > 0$ is some constant. This property is easily observed to be satisfied with constant $c = 1$ for our triplet since $(v,v)_V = (v,v)_H + (\nabla v, \nabla v)_H \geq (v,v)_H$.

**Remark 28.** The corollary states some further properties about lower bounds for $A$, which we will not bother with here.

Let $T$ be the associated operator of $s$ as defined by Lemma 26 ([Gru08, Corollary 12.19]). Since $s$ is symmetric $T$ is self-adjoint. Using integration by parts

$$s(u,v) = \sum_{k=1}^{n} (\partial_k u \partial_k v) = \left( -\sum_{k=1}^{n} \partial_k^2 u, v \right), \quad \text{for } u \in C_0^\infty(\Omega), v \in V,$$

thus $T$ extends the operator $-\Delta|_{C_0^\infty(\Omega)}$. As $T$ is a self-adjoint extension of $A_{\min}$ it acts as $-\Delta$ everywhere. The domain of $T$, denoted by $D(T)$ is defined as

$$D(T) := \{u \in V \cap D(A_{\max}) \,|\, (-\Delta u, v) = s(u,v) \text{ for all } v \in V\}.$$

Considering elements of $D(T)$ belonging to $C^2(\overline{\Omega})$ we find that for $u \in V \cap C^2(\overline{\Omega})$ and $v \in V \cap C^1(\overline{\Omega})$

$$\begin{aligned}(-\Delta u, v) - s(u,v) &= \sum_{k=1}^{n} (\partial_k u, \partial_k v) + \int_{\partial\Omega} \frac{\partial u}{\partial n} \overline{v}\, d\sigma - s(u,v) \\ &= \int_{\partial\Omega} \frac{\partial u}{\partial n} \overline{v}\, d\sigma = \int_{\partial\Omega_N} \frac{\partial u}{\partial n} \overline{v}\, d\sigma.\end{aligned}$$

If $u \in D(T)$ then $(-\Delta u, v) - s(u,v) = 0$ by construction, thus $\int_{\partial\Omega_N} \frac{\partial u}{\partial n} \overline{v}\, d\sigma = 0$ and $\frac{\partial u}{\partial n} = 0$ on $\partial\Omega_N$. Likewise, if the latter is satisfied, then working backwards we get $u \in D(T)$.

In conclusion in a generalized sense $T$ represents a mixed zero Dirichlet-Neumann boundary condition corresponding to our problem. By Lemma 26 $T$ is a closed operator and $D(T)$ is dense in both $L^2(\Omega)$ and $V$.

Returning to our problem we had $A : X \to X$, $B : Y \to X$ and $C : Y \to Y$. Clearly by the matrix shape of $A$ and $B$, we have $X = X_1 \times X_2$. Let $X_1 = Y = D(T)$ and $X_2 = L^2(\Omega)$, then $-\Delta$ is self-adjoint as we wanted and we can obtain the preconditioner $P$ with

$$P_1 = A = \begin{bmatrix} I_{L^2} & 0 \\ 0 & \alpha I_{L^2} \end{bmatrix},$$

and

$$P_2 = S = \begin{bmatrix} -\Delta & I_{L^2} \end{bmatrix} \begin{bmatrix} I_{L^2}^{-1} & 0 \\ 0 & \alpha I_{L^2}^{-1} \end{bmatrix} \begin{bmatrix} -\Delta \\ I_{L^2} \end{bmatrix} = \Delta I_{L^2}^{-1} \Delta + \alpha I_{L^2}.$$

Notably, this differs slightly from the problem described in the theory where the range are dual spaces $A : X \to X^*$, $B : Y \to X^*$ and $C : Y \to Y^*$, i.e. $M : H \to H^*$, whereas here we have considered $M$ as an automorphism.

Also, there are some issues as we here have $B : Y \to X$ we are saying that for $y \in Y$, $By \in X$, that is $-\Delta y \in X_1$ and $I_{L^2} y \in X_2$. The latter is fine, but having chosen $X_1 = Y$ we require that $-\Delta$ is an automorphism, which is generally not the case.

In the future a further study in generalization of the Schur complement for operators between dual spaces could be interesting.

#### 2.6.5.3 Dual space Schur complement

Consider Hilbert spaces $X$ and $Y$ and let $A : X \to X^*$, $B : Y \to X^*$ and $C : Y \to Y^*$. Define now $M : H \to H^*$, where $H = X \times Y$, by

$$M = \begin{bmatrix} A & B \\ B^* & C \end{bmatrix}.$$

Assuming $C$ is invertible and self-adjoint, consider the following two operator matrices

$$L = \begin{bmatrix} I_{X^*} & -BC^{-1} \\ 0 & I_{Y^*} \end{bmatrix}, \quad \text{and} \quad R = \begin{bmatrix} I_X & 0 \\ -C^{-1}B^* & I_Y \end{bmatrix}.$$

Clearly $L : H^* \to H^*$ and $R : H \to H$. We note that the adjoint of the identity operator, $I_X^*$, satisfy $I_X^*(x^*)(x) = x^*(I_X x) = x^*(x)$, thus $I_X^* = I_{X^*}$. Now

$$L^* = \begin{bmatrix} I_{X^*} & -BC^{-1} \\ 0 & I_{Y^*} \end{bmatrix}^* = \begin{bmatrix} I_{X^*}^* & 0 \\ (-BC^{-1})^* & I_{Y^*}^* \end{bmatrix} = \begin{bmatrix} I_X & 0 \\ -C^{-1}B^* & I_Y \end{bmatrix} = R.$$

Thus $L$ and $R$ are adjoints of one another. We find that

$$\begin{aligned} LMR &= \begin{bmatrix} I_{X^*} & -BC^{-1} \\ 0 & I_{Y^*} \end{bmatrix} \begin{bmatrix} A & B \\ B^* & C \end{bmatrix} \begin{bmatrix} I_X & 0 \\ -C^{-1}B^* & I_Y \end{bmatrix} \\ &= \begin{bmatrix} I_{X^*}A + (-BC^{-1})B^* & I_{X^*}B + (-BC^{-1})C \\ 0A + I_{Y^*}B^* & 0B + I_{Y^*}C \end{bmatrix} \begin{bmatrix} I_X & 0 \\ -C^{-1}B^* & I_Y \end{bmatrix} \\ &= \begin{bmatrix} A - BC^{-1}B^* & 0 \\ B^* & C \end{bmatrix} \begin{bmatrix} I_X & 0 \\ -C^{-1}B^* & I_Y \end{bmatrix} \\ &= \begin{bmatrix} (A - BC^{-1}B^*)I_X + 0(-C^{-1})B^* & (A - BC^{-1}B^*)0 + 0I_Y \\ B^*I_X + C(-C^{-1}B^*) & B^*0 + CI_Y \end{bmatrix} \\ &= \begin{bmatrix} A - BC^{-1}B^* & 0 \\ 0 & C \end{bmatrix}. \end{aligned}$$

Thus $LMR = R^*MR : H \to H^*$, and we have an operator $S = A - BC^{-1}B^*$ very similar to the Schur complement seen earlier.

Going from here there is an issue though, as [Bun88] and [Cai80] considers only intertia for bounded linear automorphism on $H$. Also, eigenvalues cannot be defined as traditionally for an operator $H \to H^*$.

So a more in depth study involving generalization of eigenvalues to operators between a Hilbert space and its dual space along with definition of generalized inertia

might be interesting. If results similar to Lemma 18 could be obtained through this we would immediately obtain the robust continuous preconditioner for our model problem. This is an open question for further future investigation.

# CHAPTER 3

# Practical implementation

In this chapter we will go through the practical implementation of the theory. In this first part we will briefly outline the resources used. After that we will go over the finite element implementation: setting up the mesh, basis functions and assembling matrices. Then we make a brief note on how the preconditioner is set up, and then cover the optimization algorithm used.

This was done using the programming language Python, initially chosen due to the availability of the open source tool FEniCS. FEniCS in particularly excels when one wants to solve Partial Differential Equations.

Apart from FEniCS, later in the project through John Pearson[1] we were introduced to IFISS. IFISS, which is an open source MATLAB package, has amoung other things tools for Optimal Control Problems, and we have initially graciously borrowed a MINRES algorithm implementation used in one of the examples coming with the toolbox. However, later we rewrote the optimization algorithm using the pseudo-code in [GHS14] as a basis.

It should be noted that while FEniCS is a Python library, the latest versions are not made available on the Windows platform. Thus initial implementations has been in a virtual machine, with which follows certain limitations. Later in the project a framework was built to directly push FEniCS-code jobs to the university server nodes, through which we have access to much greater computation power. This framework is presented further in Appendix D.

We note that in the following Python code these libraries have been importet:

```
1 # NumPy
2 import numpy as np
```

[1] https://www.kent.ac.uk/smsas/our-people/profiles/pearson_john.html

```
3
4 # SciPy
5 import scipy.sparse as sps
6 import scipy.sparse.linalg as spslinalg
7
8 # MatPlotLib
9 from matplotlib.mlab import griddata
10
11 # FEniCS Dolfin
12 from dolfin import *
13
14 # MINRES
15 from minres_sparse import minres
```

.

## 3.1 Mesh and basis functions

The first step in the implementation is setting up the mesh and basis functions. This step is one of the major reasons for using the FEniCS library as it comes with a number of functions for automatic mesh generation and several posible choices of basis functions. As FEniCS always generates triangular meshes we have been using $P_1$ elements, in FEniCS called "Continuous Galerkin", abbreviated "CG", or "Lagrange" elements of order 1. The code snippet is seen below and further explained afterwards.

```
1 # Mesh and basis fucntions
2 parameters['reorder_dofs_serial'] = False
3 mesh = UnitSquareMesh(m,m)
4 V = FunctionSpace(mesh, 'CG', 1)
```

.

As the first part of setting up the mesh and basis functions we choose a numbering scheme for the nodes and basis elements. Two different schemes are the typical ones, as we don't know them to have specific names we here refer to them as the *standard* scheme and the *serial* scheme. The standard scheme labels each node row by row starting in the lower left corner while the serial scheme takes a zig-zag path. Both schemes are pictured in Figure 3.1. FEniCS uses the alternate scheme by default, but we have chosen to use the standard one here.

Then we choose our mesh, in our model problem with a unit square domain. We pick a number $m$ defining the level of refinement of our mesh. From here we set up

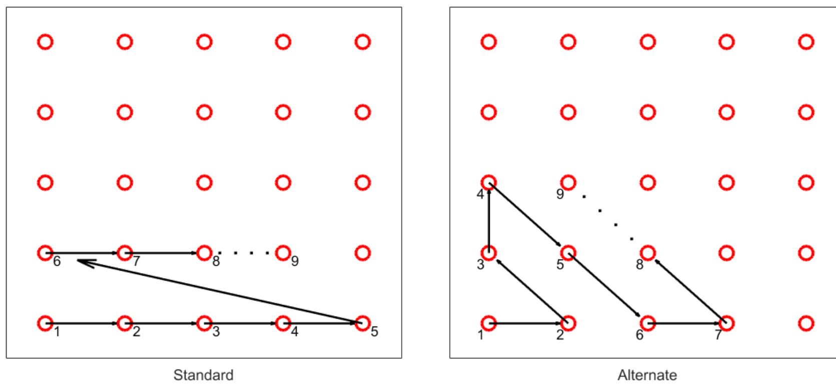


**Figure 3.1:** Standard and serial node ordering. Standard node ordering is shown in the left diagram and serial in the right.

several indicator functions to tell us if a particular point $\mathbf{x}$ is on the boundary of our mesh, and where that boundary is a Dirichlet or Neumann boundary. This allows us to use FEniCS to define a measure $ds$ on the boundary which we can use to integrate functions over the boundary.

Finally the function space, i.e. the basis functions, are chosen as $P_1$ functions as we mentioned in the beginning.

## 3.2 Matrix assembling

For the matrix assembly FEniCS again comes in handy. With the finite element function space defined we can construct trial and test functions directly using FEniCS functionality. We write up functions for the desired state, Neumann boundary and Dirichlet boundary conditions. This is done using the code snippet seen below.

```
1  # Boundary detection and boundary functions
2  def isBoundary(x):
3      return near(0,x[0]) or near(1,x[0]) or \
4             near(0,x[1]) or near(1,x[1])
5  def isDirichletBoundary(x):
6      return near(1,x[0]) or near(1,x[1])
7  def isNeumannBoundary(x):
8      return isBoundary(x) and not isDirichletBoundary(x)
9
10 class Boundary(SubDomain):
```

```
    def inside(self, x, on_boundary):
        return isBoundary(x)
class DirichletBoundary(SubDomain):
    def inside(self, x, on_boundary):
        return isDirichletBoundary(x)
class NeumannBoundary(SubDomain):
    def inside(self, x, on_boundary):
        return isNeumannBoundary(x)

class desiredState(Expression):
    def eval(self, value, x):
        if x[0] < 0.5 and x[1] < 0.5:
            value[0] = 1
        else:
            value[0] = 0

class NeumannFunction(Expression):
    def eval(self, value, x):
        value[0] = 0

Dirichlet = Constant(0.0)
g = Expression("C",C=Dirichlet,domain=mesh)
f = NeumannFunction()
yd = desiredState()

# Trial and test functions
y = TrialFunction(V)
u = TrialFunction(V)
v = TestFunction(V)
```

.

We set up the bilinear form for the weak formulation for the PDE part of our problem along with the functional on the right hand side. We also write up weak forms for the Neumann boundary condition and Dirichlet boundary condition and the desired state. With all the forms constructed FEniCS functionality allows us to assemble the matrices and vectors directly from the forms. This is seen below.

Note that the first paragraph of code sets up the measure corresponding to the boundary integral for respectively the Dirichlet and the Neumann parts of the boundary.

```
# Boundary measures
markers = FacetFunctionSizet(mesh, 0)
ds = ds[markers]
DirichletBoundary().mark(markers, 1)
```

```
NeumannBoundary().mark(markers, 2)

# Weak forms
a = inner(grad(y), grad(v))*dx
L = u*v*dx
Q = f*v*ds(2)
G = g*v*ds(1)
Yh = yd*v*dx

# Assemble matrices
K = assemble(a)
M = assemble(L)
Q = assemble(Q)
Gm = assemble(G)
Ym = assemble(Yh)
```

.

From here we move away from FEniCS objects. We cast the constructed matrices and vectors to Numpy arrays. We apply boundary conditions to the matrices and vectors as described in the theory: First to the mass matrix $M$ and the vector formulation of the desired state $\mathbf{y_d}$. Then to the stiffness matrix $K$ and a zero vector. The algorithm for applying the boundary conditions was written following the finite element course notes [Eng09] from the finite element course at DTU.

```
# NumPy matrices and vectors
K = K.array()
M = M.array()
q = Q.array()
gv = Gm.array()
z = Ym.array()

# Boundary conditions
def BoundaryConditions(A,b,f,mesh):
    # f must be an Expression with mesh-domain
    coords = mesh.coordinates().tolist()

    for i in range(0, len(coords)):
        if not isDirichletBoundary(coords[i]):
            continue

        A[i,i] = 1
        b[i] = f[i]
        for j in range(0, A.shape[0]):
            if i == j:
```

```
                continue

            A[i,j] = 0
            if not isDirichletBoundary(coords[j]):
                b[j] = b[j] - A[j,i]*f[i]
                A[j,i] = 0
    return (A,b)

# Applying boundary conditions
(K,d) = BoundaryConditions(K,np.zeros(K.shape[0]),gv,mesh)
(M,z) = BoundaryConditions(M,z,gv,mesh)
```

.

Then the matrices and vectors are made into sparse matrices and assembled into our complete system matrix and right hand side.

```
# Sparse matrices
K = sps.csr_matrix(K)
M = sps.csr_matrix(M)

# System matrix and RHS
def SystemMatrix(M,K,alpha):
    (n,_) = M.shape
    A = sps.bmat([[M,    sps.csr_matrix((n,n)),     K ],
                  [sps.csr_matrix((n,n)), alpha*M, -M ],
                  [K,    -M,     sps.csr_matrix((n,n))]])
    return A

def RightHandSide(z,g):
    (n,) = z.shape
    b = np.bmat([[z],[np.zeros(n)],[g]])
    b = np.asarray(b).squeeze()
    return b

# Build system matrix and RHS
A = SystemMatrix(M,K,alpha)
b = RightHandSide(z,d+q)
```

.

## 3.3 Preconditioner

We set up the preconditioner $P$ as an object, which you can feed the basic matrix blocks to and then set the type of preconditioner you want it to be. It then has a solve function, which computes the action of the inverse on a vector. For computational reasons the inversion is done on each block level separately, so that the preconditioner matrix is never assembled completely.

```
class Preconditioner:
    def __init__(self,M,K,alpha):
        self.type = []
        self.M = M
        self.alpha = alpha
        self.K = K

        self.D = []
        self.L = []

    def setType(self,t):
        self.type = t
        if self.type == 1: # Diagonal
            self.D = sps.diags(self.K.diagonal()**2/self.M.diagonal() + self
                .K.diagonal()/self.alpha)
        elif self.type == 2: # Rees Schur approximation
            self.L = self.K
        elif self.type == 3: # Pearson Schur approximation
            self.L = self.K+self.M/np.sqrt(self.alpha)

    def solve(self,x_it):
        n = self.M.shape[0]
        if self.type == 1:
            r1 = x_it[0:n]
            r2 = x_it[n:2*n]
            z1 = spslinalg.spsolve(sps.diags(self.M.diagonal()),r1)
            z2 = spslinalg.spsolve(sps.diags(self.M.diagonal()),r2)/self.
                alpha

            r3 = x_it[2*n:3*n]
            z3 = spslinalg.spsolve(sps.diags(self.D.diagonal()),r3)
        elif self.type == 2 or self.type == 3:
            r1 = x_it[0:n]
            r2 = x_it[n:2*n]
            z1 = spslinalg.spsolve(self.M,r1)
            z2 = spslinalg.spsolve(self.M,r2)/self.alpha

```

```
36              r3 = x_it[2*n:3*n]
37              tmp = spslinalg.spsolve(self.L,r3)
38              tmp = self.M.dot(tmp)
39              z3 = spslinalg.spsolve(self.L,tmp)
40          else:
41              z1 = x_it[0:n]
42              z2 = x_it[n:2*n]
43              z3 = x_it[2*n:3*n]
44
45          return np.hstack((z1,z2,z3))
```

.

When the preconditioner object is initialized it is fed the mass matrix, stiffness matrix and the regularization parameter alpha. Upon selecting the preconditioner type, the composite matrices required for that particular type is then computed from the given ones. From here the object can be fed to the optimization algorithm, which will use the solve command to solve the inversion problem $\mathbf{w} = P^{-1}\mathbf{v}$.

The setType method is used for selecting the preconditioner type. The primary difference is which Schur complement approximation is used. The method is easily modified for other variations of preconditioners.

```
1 # Setup preconditioner
2 P = Preconditioner(M,K,alpha)
3 P.setType(3)
```

.

## 3.4 MINRES

In Appendix B the theory behind the standard MINRES algorithm is covered in some detail. It solves the problem $Ax = b$, where the knowns are $A$ is the system matrix and $b$ the right hand side, by successively searching for the minimal residual $r = b - Ax$ in a increasing sequence of Krylov subspaces

$$\mathcal{K}_1(A, b) \subseteq \mathcal{K}_2(A, b) \subseteq \cdots \subseteq \mathcal{K}_n(A, b),$$

where $n = \dim b$.

The optimization algorithm used in the code, however, is the preconditioned MINRES algorithm which minimizes the $P^{-1}$ norm, $\|r\|_{P^{-1}} = \sqrt{(r, P^{-1}r)}$, of the residual in the Krylov subspaces $A\mathcal{K}_k(P^{-1}A, P^{-1}b)$ [GHS14].

Note that as $\mathcal{K}_k(A,b) = \text{span}_{0\le j<k}\{A^j b\}$ we have

$$\begin{aligned} A\mathcal{K}_k(P^{-1}A, P^{-1}b) &= A \operatorname*{span}_{0\le j<k}\{(P^{-1}A)^j P^{-1}b\} \\ &= \operatorname*{span}_{0\le j<k}\{(AP^{-1})^{j+1}b\} \\ &= AP^{-1} \operatorname*{span}_{0\le j<k}\{(AP^{-1})^j b\} \qquad = AP^{-1}\mathcal{K}_k(AP^{-1}, b). \end{aligned}$$

Note that by associativity

$$\begin{aligned} A(P^{-1}A)^j P^{-1} &= A(P^{-1}A)(P^{-1}A)\cdots(P^{-1}A)P^{-1} \\ &= (AP^{-1})(AP^{-1})(A\cdots P^{-1})(AP^{-1}), \end{aligned}$$

which encompasses the second equality.

Following the derivation in Appendix B but with the new choice of Krylov space we obtain the following expression for the new basis element:

$$\tilde{v}_{t+1} = AP^{-1}v_t - \sum_{j=1}^{t} \text{proj}_{v_j}(AP^{-1}v_j).$$

Here $\text{proj}_u(v) = (v,u)_{P^{-1}}\widehat{u}$, where $\widehat{u} = u/\|u\|_{P^{-1}}$, so the projection is with respect to the $P^{-1}$-norm. Recalling that $v_j$ is already normalized (here in the $P^{-1}$-norm) we have

$$\begin{aligned} \tilde{v}_{t+1} &= AP^{-1}v_t - \sum_{j=1}^{t}(AP^{-1}v_t, v_j)_{P^{-1}}v_j \\ &= Az_t - \sum_{j=1}^{t}(Az_t, P^{-1}v_j)v_j \\ &= Az_t - \sum_{j=1}^{t}(Az_t, z_j)v_j \\ &= Az_t - \sum_{j=1}^{t}(z_j^T Az_t)v_j. \end{aligned}$$

Thus again going by the derivation in the appendix

$$AP^{-1}V_t = V_{t+1}H_t, \quad H_t = \begin{bmatrix} z_1^T Az_1 & z_1^T Az_2 & z_1^T Az_3 & z_1^T Az_4 & \dots \\ \|\tilde{v}_2\|_{P^{-1}} & z_2^T Az_2 & z_2^T Az_3 & z_2^T Az_4 & \dots \\ 0 & \|\tilde{v}_3\|_{P^{-1}} & z_3^T Az_3 & z_3^T Az_4 & \dots \\ 0 & 0 & \|\tilde{v_4}\|_{P^{-1}} & z_4^T Az_4 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \in \mathbb{R}^{(t+1)\times t},$$

and $H_t$ is symmetric, since $P = P^T$ symmetric implies $P^{-1} = P^{-T}$ symmetric and thus as in the appendix

$$(P^{-1}V_t)^T AP^{-1}V_t = (P^{-1}V_t)^T V_{t+1}H_t = V_t^T P^{-1}V_{t+1}H_t = [I_t|\mathbf{0}]H_t = T_t,$$

where for $\delta_i = z_i^T Az_i$ and $\gamma_i = z_{i-1}^T Az_i = \|\tilde{v}_i\|_{P^{-1}}$ we have

$$T_t = \begin{bmatrix} \delta_1 & \gamma_2 & 0 & \dots & 0 \\ \gamma_2 & \delta_2 & \gamma_3 & & 0 \\ 0 & \gamma_3 & \delta_3 & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \gamma_t \\ 0 & 0 & \dots & \gamma_t & \delta_t \end{bmatrix}.$$

This covers how the basis differs for the preconditioned MINRES algorithm. The residual minimized here is, as stated initially, minimized in the $P^{-1}$-norm. We start with the initial guess $x_0 = 0$ and thus have $r_0 = b - Ax_0 = b$. Now $r_t = b - AP^{-1}x_t$, thus for $K_t = [b, AP^{-1}b, \dots, (AP^{-1})^{t-1}b]$ we have

$$\begin{aligned} \min_{x\in\mathcal{K}_t} \|b - AP^{-1}x\|_{P^{-1}} &= \min_{y\in\mathbb{R}^t} \|b - AP^{-1}K_t y\|_{P^{-1}} \\ &= \min_{z\in\mathbb{R}^t} \|b - AP^{-1}V_t z\|_{P^{-1}} \\ &= \min_{z\in\mathbb{R}^t} \|b - V_{t+1}H_t z\|_{P^{-1}} \\ &= \min_{z\in\mathbb{R}^t} \|\|b\|_{P^{-1}}v_1 - V_{t+1}H_t z\|_{P^{-1}} \\ &= \min_{z\in\mathbb{R}^t} \|V_{t+1}(\|b\|_{P^{-1}}V_{t+1}^{-1}v_1 - H_t z)\|_{P^{-1}}. \end{aligned}$$

Now, note that since $V_{t+1}$ contains the orthogonal basis with respect to the $P^{-1}$-inner product, we have $V_{t+1}^T P^{-1}V_{t+1} = I$, hence $V_{t+1}^{-1} = V_{t+1}^T P^{-1}$ and thus

$$\begin{aligned} \min_{z\in\mathbb{R}^t} \|V_{t+1}(\|b\|_{P^{-1}}V_{t+1}^{-1}v_1 - H_t z)\|_{P^{-1}} &= \min_{z\in\mathbb{R}^t} \|V_{t+1}(\|b\|_{P^{-1}}V_{t+1}^T P^{-1}v_1 - H_t z)\|_{P^{-1}} \\ &= \min_{z\in\mathbb{R}^t} \|V_{t+1}(\|b\|_{P^{-1}}e_1 - H_t z)\|_{P^{-1}}, \end{aligned}$$

where $e_1 = (1, 0, 0, \dots, 0) \in \mathbb{R}^{t+1}$. Furthermore, consider $\|V_k x\|_{P^{-1}}^2$, then

$$\begin{aligned} \|V_k x\|_{P^{-1}}^2 &= (V_k x, V_k x)_{P^{-1}} = (V_k x, P^{-1}V_k x)_2 \\ &= (P^{-1}V_k x)^T V_k x = x^T V_k^T P^{-1}V_k x = x^T x = (x, x)_2 = \|x\|_2^2. \end{aligned}$$

Because of this

$$\min_{z\in\mathbb{R}^t} \|V_{t+1}(\|b\|_{P^{-1}}e_1 - H_t z)\|_{P^{-1}} = \min_{z\in\mathbb{R}^t} \|\|b\|_{P^{-1}}e_1 - H_t z\|_2.$$

This covers how the preconditioner enters the MINRES algorithm. From here computations follow the same procedure as the regular MINRES algorithm.

The preconditionned MINRES algorithm in [GHS14] (Algorithm 3.1) has been implemented as `minres.py`, which can be found in Appendix B.

The function is called by

```
# Included in the beginning: from minres import minres
(x,iters,conv,resvec) = minres(A,b,P,maxit = 1000,tol = 1e-6)
```

.

## 3.5 Visualization

Visualizations has been done using MATLAB. As the computations were executed on the university servers and the result was downloaded from them, there was not much of an insentive to bring it into Python again over MATLAB. The latter also features much simpler and easier to use plotting tools. That and the fact that we have years of actual work experience in using MATLAB made MATLAB the plotting tool of choice.

The result was collected and stored as a collection of variables via the following python code snippet.

```
def ExtractSolution(x):
    n = int(x.shape[0]/3)
    return (x[0:n],x[n:2*n],x[2*n:3*n])

def Vector2Function(V,v):
    f = Function(V)
    f.vector()[:] = v
    return f

def Function2Griddata(f,n=None):
    z = f.vector().array()
    xy = f.function_space().mesh().coordinates()
    x = xy[:,0]
    y = xy[:,1]
    if not (n and isinstance(n, int)):
        n = np.round(np.sqrt(len(x)))
    xx = np.linspace(min(x),max(x),n)
    yy = np.linspace(min(y),max(y),n)

    XX,YY = np.meshgrid(xx,yy)
```

```
    ZZ = griddata(x,y,z,XX,YY,interp='linear')
    return (XX,YY,ZZ)

(yv,uv,pv) = ExtractSolution(x)

y = Vector2Function(V,yv)
u = Vector2Function(V,uv)
p = Vector2Function(V,pv)

(Ux,Uy,Uz) = Function2Griddata(u)
(Yx,Yy,Yz) = Function2Griddata(y)
(Px,Py,Pz) = Function2Griddata(p)

import scipy.io as scio
scio.savemat(outputfilename, {
    'Ux': Ux, 'Uy': Uy, 'Uz': Uz,
    'Yx': Yx, 'Yy': Yy, 'Yz': Yz,
    'Px': Px, 'Py': Py, 'Pz': Pz,
    'alpha': alpha, 'h': 1.0/m, 'iters': iters, 'resvec':resvec}, oned_as='
        row')
```

.

The stored variables are:

**State $y$:** Yx, Yy, Yz,

**Control $u$:** Ux, Uy, Uz,

**Lagrange multiplier $p$:** Px, Py, Pz.

**Iterations:** iter,

**Regularization parameter $\alpha$:** alpha,

**Mesh size $h$:** h,

**Residuals:** resvec.

The following MATLAB code was then used to generate the plots.

```
d = load(datafilelocation);

iplot = 1;
for prmtr = {'Y','U'};
  x = [0,1];
  y = [0,1];
```

```
7     margin = [-0.25,0.25];
8
9     figure(iplot);
10    hold on
11    % Axes
12    props = {'k-','LineWidth',2,'ShowArrowHead','Off'}
13    axX = quiver3(-0.1,0,0,1.3,0,0,props{:});
14    axY = quiver3(0,-0.1,0,0,1.3,0,props{:});
15    axZ = quiver3(0,0,-0.1,0,0,1.3,props{:});
16    % Surface
17    surf(d.([prmtr{1},'x']),d.([prmtr{1},'y']),d.([prmtr{1},'z']));
18    scale = 0.25;
19    xlim(x+scale*margin)
20    ylim(y+scale*margin)
21    z = zlim;
22    set(axZ,'ZData',z(1),'WData',(z(2)-z(1))*1.1);
23    zlim(z)
24    view(55,35)
25    grid on
26    xlabel('x')
27    ylabel('y')
28    iplot = iplot+1;
29  end
30
31  % Residuals
32  figure(iplot);
33  semilogy(d.resvec);
34  grid on
35  xlabel('iterations')
36  ylabel('residual')
```

.

CHAPTER 4

# Numerical results

In this chapter we will present the numerical results based on the theory discussed in Chapter 2. The results are based on different setups for our model problem. Forclarity we will now explain the core setup. In the following page we will preceed the results with an explanation of the alteration to the core setup used in each case.

The results will consist of solution plots for the state $y$ and the control $u$ along with a plot of the normalized residual $\|r_k\|/\|r_0\|$ at each iteration $k$. The residual plot also contains a slope value $a$ for the “trend-line” which has the following expression: $y = \exp(ak)$, which tells us about how fast our residuals approach zero:

$$\frac{\|r_k\|}{\|r_0\|} \approx \exp(ak).$$

The core setup is the following: The domain is chosen as the unit square, $\Omega := [0, 1]^2$. The boundary is separated into a Dirichlet part $\partial\Omega_D$ and a Neumann part $\partial\Omega_N$ defined as

$$\begin{aligned}
\partial\Omega_D &:= \{(x_1, x_2) \in \partial\Omega \,|\, x_1 = 1 \vee x_2 = 1\} \\
\partial\Omega_N &:= \partial\Omega \backslash \partial\Omega_D = \{(x_1, x_2) \in \partial\Omega \,|\, x_1 = 0 \vee x_2 = 0\} \backslash \{(0, 1), (1, 0)\}.
\end{aligned}$$

For the boundary conditions $y = f$ on $\partial\Omega_D$ and $\frac{\partial y}{\partial n} = g$ on $\partial\Omega_N$ we consider the simplest case $f = g = 0$.

The desired state $y_d$ can be seen in Figure 4.1 and is defined as $y_d := \chi_{\Omega_1}$ with $\Omega_1 = [0, \frac{1}{2}]^2$.

This yields the following matrix system

$$\begin{bmatrix} M & 0 & K \\ 0 & \alpha M & -M \\ K & -M & 0 \end{bmatrix} \begin{bmatrix} \mathbf{y} \\ \mathbf{u} \\ \mathbf{p} \end{bmatrix} = \begin{bmatrix} \mathbf{y_d} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}.$$

As a preconditioner here we use the one from Zulehner [Zul11] discussed in the Theory section when the block-matrices $A$ and negative Schur complement $S = C +$

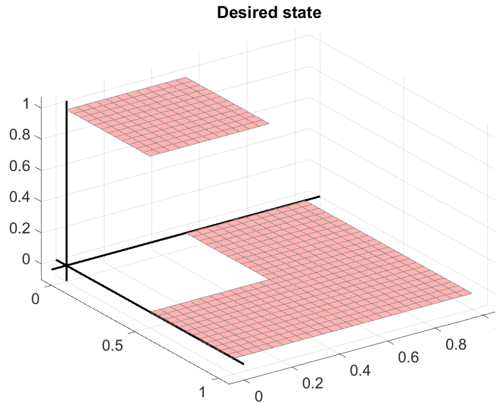


**Figure 4.1:** Desired state.

$BA^{-1}B^T$ are non-singular. Here

$$A = \begin{bmatrix} M & 0 \\ 0 & \alpha M \end{bmatrix}$$

and

$$S = 0 + \begin{bmatrix} K & -M \end{bmatrix} \begin{bmatrix} M^{-1} & 0 \\ 0 & \alpha^{-1}M^{-1} \end{bmatrix} \begin{bmatrix} K \\ -M \end{bmatrix} = KM^{-1}K + \alpha^{-1}M.$$

As an approximation to the Schur complement we use the approximation suggested by Pearson [Pea13] $S \approx \widehat{S}_2 = (K + \sqrt{\alpha}^{-1}M)M^{-1}(K + \sqrt{\alpha}^{-1}M)$.

In the following different setups we will make small changes to observe the convergence properties under these conditions. The first results will be for our core setup. In the second and third setups we will change the preconditioner in some way and see how the convergence properties are affected. In the forth and fifth setup we change the boundary conditions and in the sixth setup we again consider different preconditioner.

The MINRES algorithm was given the following parameters: "tolerance level": $10^{-9}$ and "maximum iterations": 1500. A raw setup with no preconditioning was also computed like the ones below here. However, it maxed out the iteration count in every $(h, \alpha)$ pair and thus does not really varant enough interest for a dedicated section.

More visualzations can be found in Appendix C.

## 4.1 Core setup

This setup corresponds to the inner products

$$(u,v)_{Y_h} = (u,v)_{L^2},$$
$$(u,v)_{U_h} = \alpha(u,v)_{L^2}$$
$$(u,v)_{P_h} = \alpha(u,v)_{L^2} + (\Delta u, \Delta v)_{(L^2)^*} + 2\sqrt{\alpha}^{-1}(\nabla u, \nabla v)_{L^2}.$$

The setup here is as desribed in the first part of the chapter. In Table 4.1 are the iterations counts for different values of $h$ and $\alpha$. As can be seen the iteration count is very stable with respect to variations in $h$ and $\alpha$, as wanted.

In Figure 4.2 the resulting solution to the state $y$ and constrol $u$ is visualized. The result is from the case $\alpha = 10^{-4}$ and $h = 2^{-5}$. Variations in $h$ does not change the solution significantly, however, for very small values of $h$ the gridlines will condense to a point where it can be hard to see details in the plot. This played a part in this choice for $h$. Variations in $\alpha$ change the solution significantly, larger values of alpha forces more smooth solutions as the control is penalized more, while very small values of alpha lets the state to better approximate the discontinuity in our desired state, but the control go more wild with large rapid fluctuations which is not desirable.

We also see the normalized residual plot in Figure 4.2. The normalized residuals follow the line very closely with a slope of about -0.9, which is close to -1. This is quite good and we see how the tolerance is achieved after simply 23 iterations.

| | | $\alpha$ | | | | | |
|---|---|---|---|---|---|---|---|
| | | 1e-3 | 1e-4 | 1e-5 | 1e-6 | 1e-7 | 1e-8 |
| $h$ | $2^{-4}$ | 23 | 23 | 21 | 21 | 21 | 19 |
| | $2^{-5}$ | 23 | 23 | 23 | 23 | 21 | 21 |
| | $2^{-6}$ | 23 | 23 | 23 | 23 | 23 | 21 |
| | $2^{-7}$ | 23 | 23 | 23 | 23 | 23 | 21 |

**Table 4.1:** The iteration counts for the core setup for different values of $\alpha$ and $h$.

## 4.2 Different preconditioner Type I

In this setup we change the center block of the preconditioner from $\alpha M$ to $M$. As $\alpha$ is just a constant we will still have upper and lower bounds such that the new preconditioner satisfy the relations established by Zulehner in [Zul11], however, as

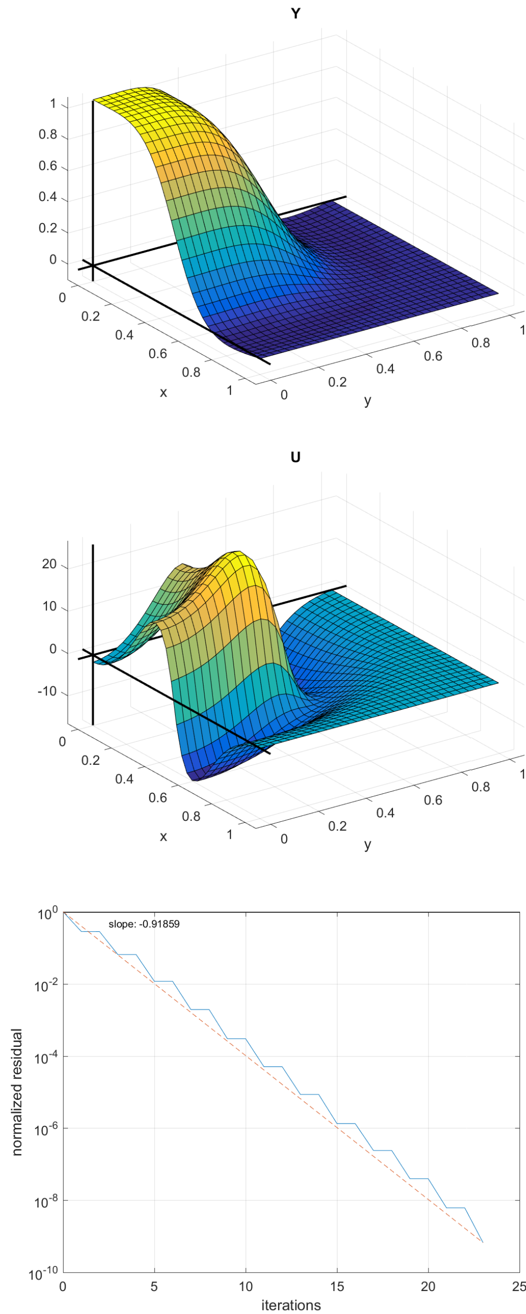


**Figure 4.2:** Solutions to the core setup. We see the state $y$ and control $u$ along with the normalized residual at each iteration. The slope is $-0.91859$. This solution is for the case $\alpha = 10^{-4}$ and $h = 2^{-5}$.

the preconditioner does not accomodate as much for $\alpha$ anymore it should be expected that the bounds now depend on $\alpha$. The resulting precondtioner is

$$P = \begin{bmatrix} M & 0 & 0 \\ 0 & M & 0 \\ 0 & 0 & S \end{bmatrix},$$

where we still use the Pearson approximation for the Schur complement, that is

$$S \approx \widehat{S}_2 = (K + \frac{1}{\sqrt{\alpha}}M)M^{-1}(K + \frac{1}{\sqrt{\alpha}}M).$$

This corresponds to the inner products

$$\begin{aligned}(u,v)_{Y_h} &= (u,v)_{L^2},\\ (u,v)_{U_h} &= (u,v)_{L^2}\\ (u,v)_{P_h} &= \alpha(u,v)_{L^2} + (\Delta u, \Delta v)_{(L^2)^*}.\end{aligned}$$

The resulting iteration counts can be seen in Table 4.2. We observe as expected that the iteration count changes dramatically with $\alpha$. Note that the maximum iterations for the MINRES algorithm was set to 1500, which is why the table simply shows 1500+ for the last results.

The visualization of the solution for choices $\alpha = 10^{-4}$ and $h = 2^{-5}$ can be seen in Figure 4.3. The change in the precondtioner should not affect the resulting solution, which we also see here, however, the number of iterations required to reach the tolerance level skyrocketed. Looking at the normalized risidual plot we see that though the normalized risiduals kind of follow the straight line, the slope is only about -0.09, so the the steps are quite poor.

| | | $\alpha$ | | | | | |
|---|---|---|---|---|---|---|---|
| | | 1e-3 | 1e-4 | 1e-5 | 1e-6 | 1e-7 | 1e-8 |
| $h$ | $2^{-4}$ | 85 | 193 | 670 | 1500+ | 1500+ | 1500+ |
| | $2^{-5}$ | 87 | 215 | 758 | 1500+ | 1500+ | 1500+ |
| | $2^{-6}$ | 89 | 232 | 783 | 1500+ | 1500+ | 1500+ |
| | $2^{-7}$ | 89 | 236 | 815 | 1500+ | 1500+ | 1500+ |

**Table 4.2:** The iteration counts for different values of $\alpha$ and $h$ for the setup with the center block of the preconditioner changed.

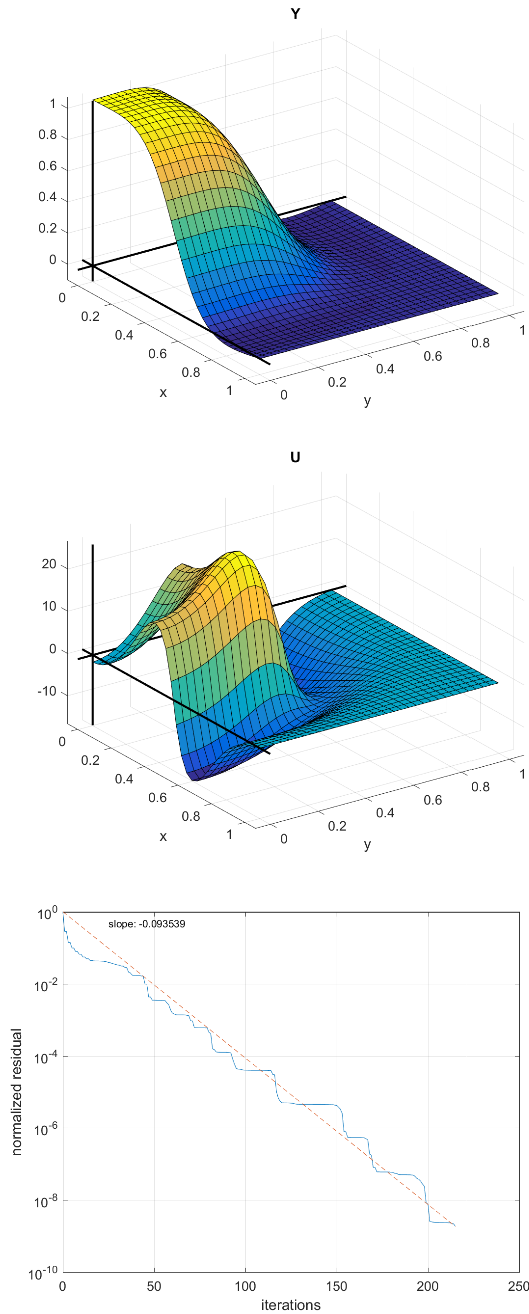


**Figure 4.3:** Solutions to the setup with the center block of the preconditioner changed. The state $y$ and the control $u$, along with the normalized residual evolution. The slope is $-0.093539$. This solution is for the case $\alpha = 10^{-4}$ and $h = 2^{-5}$.

## 4.3 Different Preconditioner Type II

In this setup we change the Schur complement approximation used in the preconditioner from the Pearson approximation to

$$S \approx \widehat{S}_1 = KM^{-1}K,$$

that is: simply dropping the $\alpha^{-1}M$ term, as stated in Section 2.6.4.

Clearly this approximation removes the $\alpha$ dependence from the Schur complement in our preconditioner and we expect the results to be in a vein similar to what we saw in the previous case.

This setup corresponds to the inner product

$$\begin{aligned}(u,v)_{Y_h} &= (u,v)_{L^2},\\ (u,v)_{U_h} &= \alpha(u,v)_{L^2}\\ (u,v)_{P_h} &= (\Delta u, \Delta v)_{(L^2)^*}.\end{aligned}$$

As we see in Table 4.3 the iteration count vary in $\alpha$ again, though not nearly as severely as we saw it in the previous setup, Section 4.2.

Again the plot of the solution for the state and control can be seen in Figure 4.4, along with the normalized residuals. The residuals show that this setup exhibits somewhat poor convergence initially, but appear to get progressivley better. The slope is about -0.64, which while not close to -1 is exceedingly better than the -0.09 from before. If we disregarded the first 10-15 iterations the slope is like closer to -0.7 or better.

| | | $\alpha$ | | | | | |
|---|---|---|---|---|---|---|---|
| | | 1e-3 | 1e-4 | 1e-5 | 1e-6 | 1e-7 | 1e-8 |
| $h$ | $2^{-4}$ | 21 | 35 | 69 | 153 | 381 | 693 |
| | $2^{-5}$ | 21 | 33 | 69 | 159 | 454 | 1313 |
| | $2^{-6}$ | 21 | 33 | 71 | 169 | 476 | 1441 |
| | $2^{-7}$ | 21 | 33 | 71 | 170 | 472 | 1438 |

**Table 4.3:** The iteration counts for different values of $\alpha$ and $h$. This is for the setup where the approximation to the Schur complement was changed.

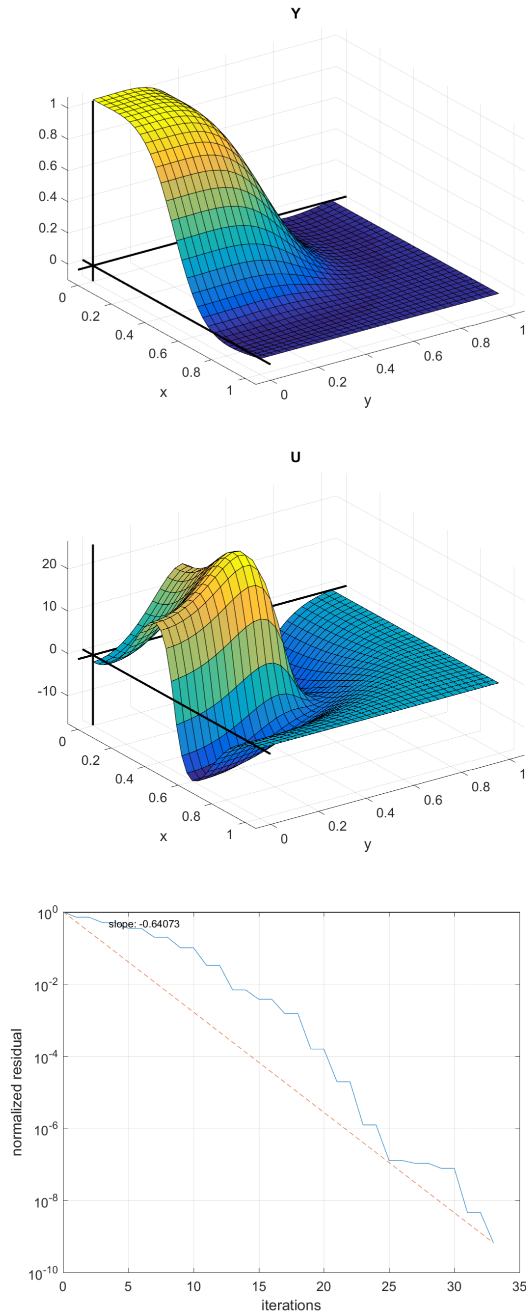


**Figure 4.4:** Solutions for the setup with the Rees Schur complement approximation. We see the state $y$ and control $u$ and normalized residual evolution. The slope is $-0.64073$. This solution is for the case $\alpha = 10^{-4}$ and $h = 2^{-5}$.

## 4.4 Different Boundary

Here we will consider a setup where we change the Dirichlet and Neumann boundary sets. The new boudnary sets will be defined as

$$\partial\Omega_D := \{(x_1, x_2) \in \partial\Omega \,|\, x_1 = 1 \vee x_2 = 0\}$$
$$\partial\Omega_N := \partial\Omega \backslash \partial\Omega_D = \{(x_1, x_2) \in \partial\Omega \,|\, x_1 = 0 \vee x_2 = 1\} \backslash \{(0,0), (1,1)\}.$$

This will force a discontinuity at the boundary because of the desired state, which could affect convergence.

The preconditioner and thus inner product here is the same as the one for the core setup in Section 4.1.

Looking at the iteration counts in Table 4.4 there is a slight change at larger values of $\alpha$, however, change is not significant. The visualized solution can be viewed in Figure 4.5 for the values $\alpha = 10^{-4}$ and $h = 2^{-5}$. The normalized residuals show the same tendency as for the core setup with a slope at about -0.9.

| | | $\alpha$ | | | | | |
|---|---|---|---|---|---|---|---|
| | | 1e-3 | 1e-4 | 1e-5 | 1e-6 | 1e-7 | 1e-8 |
| $h$ | $2^{-4}$ | 23 | 23 | 23 | 23 | 21 | 19 |
| | $2^{-5}$ | 23 | 23 | 23 | 23 | 21 | 21 |
| | $2^{-6}$ | 23 | 23 | 23 | 23 | 23 | 21 |
| | $2^{-7}$ | 23 | 23 | 23 | 23 | 23 | 23 |

**Table 4.4:** The iteration counts for different values of $\alpha$ and $h$ for the setup where the boundary sets has been changed.

## 4.5 Reduced system

In this setup we consider the reduced system for the opportunity to try out the preconditioner suggested by Zulehner in [Zul11] for the case where both block $A$ and block $C$ in the system matrix are positive definite.

We reduce the first order optimality system by using the relation $u = \alpha^{-1}p$. This gives us the reduced problem

$$-\Delta y = \alpha^{-1}p$$
$$-\Delta p = y_d - y,$$

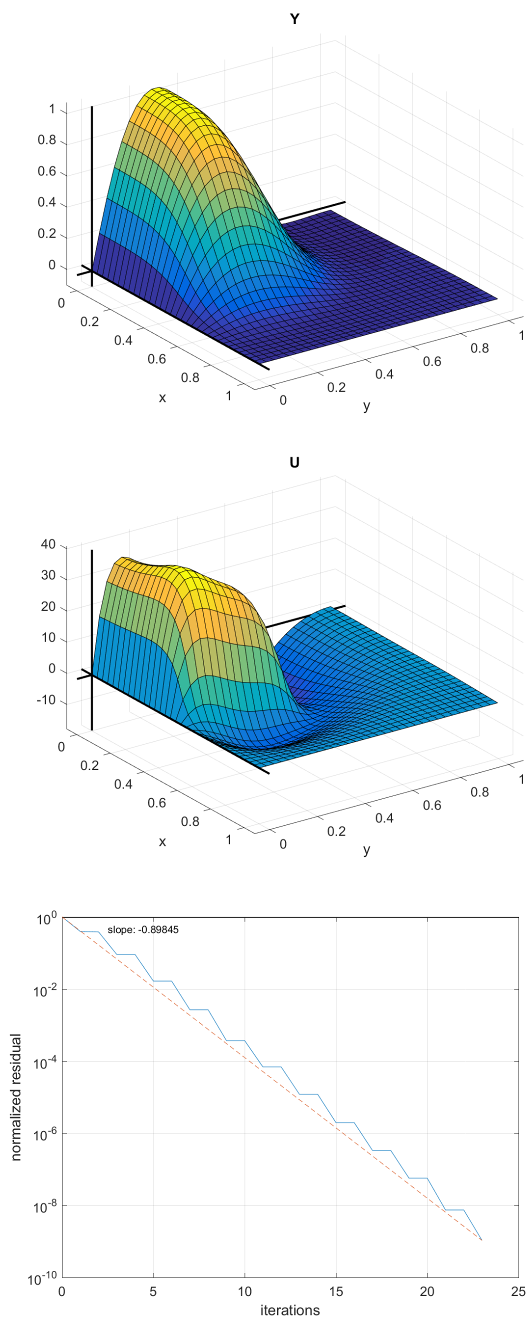


**Figure 4.5:** Solutions for the setup with changed boundary sets. Depicted are the state $y$ and control $u$ with the normalized residual evolution. The slope is $-0.89645$. The result is for $\alpha = 10^{-4}$ and $h = 2^{-5}$.

which discretizes to the following saddle point system

$$\begin{bmatrix} M & K \\ K & -\alpha^{-1}M \end{bmatrix} \begin{bmatrix} \mathbf{y} \\ \mathbf{p} \end{bmatrix} = \begin{bmatrix} \mathbf{y_d} \\ 0 \end{bmatrix}.$$

Here both block $A = M$ and block $C = \alpha^{-1}M$ are positive definite. Thus we may use the preconditioner from [Zul11]

$$P = \begin{bmatrix} M + \sqrt{\alpha}K & 0 \\ 0 & \alpha^{-1}M + \sqrt{\alpha}^{-1}K \end{bmatrix}.$$

This preconditioner has the advantage that it requires no approximation of the matrices. And since both $M$ and $K$ are sparse matrices with nonzero entries only close to the diagonal, the resulting diagonal blocks will both be sparse matrices.

This corresponds to the inner products

$$\begin{aligned} (u, v)_{Y_h} &= (u, v)_{L^2} + \sqrt{\alpha}(\nabla u, \nabla v)_{L^2} \\ (u, v)_{P_h} &= \alpha^{-1}(u, v)_{L^2} + \sqrt{\alpha}^{-1}(\nabla u, \nabla v)_{L^2}. \end{aligned}$$

As seen in Table 4.5 iteration count is very stable in both $\alpha$ and $h$. What exactly gives rise to the slight variations in number of iterations is hard to say.

The solution state and control can be seen in Figure 4.6. Compared to the earlier setups this one exhibits the best slope at about -0.97. Also, compared to the two other setups with slope about -0.9, this one seem to follow the line a bit on both sides, where the others tend to stay above. It is likely the factor that no approximation was performed in the preconditioner that makes this one slightly better.

| | | $\alpha$ | | | | | |
|---|---|---|---|---|---|---|---|
| | | 1e-3 | 1e-4 | 1e-5 | 1e-6 | 1e-7 | 1e-8 |
| $h$ | $2^{-4}$ | 20 | 20 | 20 | 21 | 21 | 19 |
| | $2^{-5}$ | 22 | 22 | 22 | 21 | 21 | 21 |
| | $2^{-6}$ | 22 | 22 | 22 | 22 | 22 | 21 |
| | $2^{-7}$ | 22 | 22 | 22 | 22 | 22 | 21 |

**Table 4.5:** The iteration counts for different values of $\alpha$ and $h$ for the reduced system setup with Zulehners preconditioner.

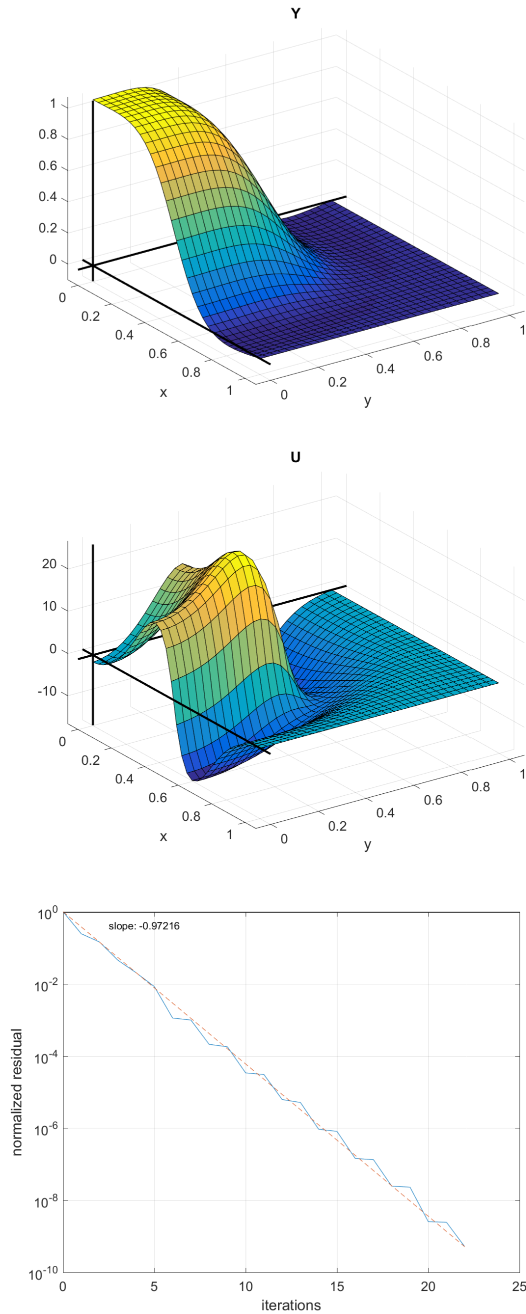


**Figure 4.6:** Solutions for the reduced system setup with Zulehners preconditioner. We see the state $y$ and control $u$ together with the normlized residuals at each step. The slope is $-0.97234$. The result is for $\alpha = 10^{-4}$ and $h = 2^{-5}$.

# CHAPTER 5

# Conclusion

We have considere PDE-constrained Optimization in this thesis. The main contribution of the thesis is the attempt to obtain a robust preconditioner for the operator setting by generalization of the Schur complement to operator matrices.

For our problem we focused on solving a distributed control problem with mixed Dirichlet and Neumann boundary conditions. In the first part of the theory chapter we developed the theoretical tools to show how to combine the sub elements of the distributed control problem into a single equation using the Lagrange multipliers. We then work through the KKT-conditions which gives rise to a saddle point system. We note here that the KKT-conditions result in a system of three equations two of which are PDE problems with certain boundary conditions. We finally solve the three equations together. In summary, the equations produced a matrix operator acting on the direct sum of the state space $Y$, control space $U$ and adjoint space $P$.

In the subsequent section of the theory we obtain a finite dimensional system by discretization of the matrix operator system using the finite element method. This produce a system of matrix equations involving mass and stiffness matrices. We investigate the application of boundary conditions to the mass and stiffness matrices, a process not normally seen reported in previous work on PDE-constrained optimization. The final system of matrix equations after applying boundary conditions also form a saddle point system.

In the last part of the theory, we focus on the goal of our thesis and investigate preconditioners for the saddle point problem. Here we followed the work by Zulhener in [Zul11] and explored how the normed spaces $Y$, $U$, and $P$ affect the condition number of the derived operator problem. That is, the saddle point problem in the (in)finite dimensional space. We observed how the normed spaces play a signigicant role on the convergence rate of the solution. The main result here stems from finding inner product operators $\mathcal{I}_V$ and $\mathcal{I}_Q$ satisfying the equivalence relations (2.46). Satisfying the relation with good constants (close upper and lower bounding constants) correpsonds to finding inner products in which the operator has a good condition

number and thus produce robust preconditioners.

In the final part on preconditioners we consider preconditioning in the operator setting. We generalize the Schur complement to operator matrices $M : H \to H$ and extend the result on positive definiteness in Lemma 18 to the operator setting in Lemma 25. As far as we know the has not been done before.

We proceed by considering a saddle point problem with such a matrix operator $M$ with a preconditioner analogous to the one in the discrete setting. In this setting we use the theory of realizations for differential operators on Sobolev spaces to derive the necessary search space for $-\Delta$ to be self-adjoint as necessary.

We end with a note that while this seemed to work, this problem differed from the problems discussed previously in the theory as the range og $M$ here is $H^*$, that is, the dual space. Furthermore, we observe some discrepancies about our spaces occuring because $-\Delta$ is not an automirphism in general.

We attempt to get around this in the last section, where we generalize the Schur complement further to operator matrices $M : H \to H^*$. However, we ultimately conclude that to establish a new generalization of Lemma 18 for this setting, we lack a generalization of the intertia law for such operators.

In chapter 4 we apply the results from the theory to practical application using the Python setup with FEniCS described in chapter 3, where we also covered the theoretical derivation for the preconditioned MINRES to demonstrate why this is an efficient choice of algorithm for our optimization problem. We present several numerical results for different variants of preconditioners:

$$P_1 = \begin{bmatrix} M & 0 & 0 \\ 0 & \alpha M & 0 \\ 0 & 0 & \widehat{S}_2 \end{bmatrix}, \quad P_2 = \begin{bmatrix} M & 0 & 0 \\ 0 & M & 0 \\ 0 & 0 & \widehat{S}_2 \end{bmatrix},$$

$$P_3 = \begin{bmatrix} M & 0 & 0 \\ 0 & \alpha M & 0 \\ 0 & 0 & \widehat{S}_1 \end{bmatrix}, \quad P_4 = \begin{bmatrix} M + \sqrt{\alpha} K & 0 \\ 0 & \alpha^{-1} M + \sqrt{\alpha}^{-1} K \end{bmatrix},$$

where $\widehat{S}_1 = KM^{-1}K$ and $\widehat{S}_2 = (K + \sqrt{\alpha}^{-1} M)M^{-1}(K + \sqrt{\alpha}^{-1} M)$ are Schur complement approximations.

The results show that the best preconditioners are $P_1$ and $P_4$, which corresponds

to the inner products

$$
\begin{aligned}
(u,v)_{Y_h} &= (u,v)_{L^2},\\
(u,v)_{U_h} &= \alpha(u,v)_{L^2}\\
(u,v)_{P_h} &= \alpha(u,v)_{L^2} + (\Delta u, \Delta v)_{(L^2)^*} + 2\sqrt{\alpha}^{-1}(\nabla u, \nabla v)_{L^2}
\end{aligned}
$$

and

$$
\begin{aligned}
(u,v)_{Y_h} &= (u,v)_{L^2} + \sqrt{\alpha}(\nabla u, \nabla v)_{L^2}\\
(u,v)_{P_h} &= \alpha^{-1}(u,v)_{L^2} + \sqrt{\alpha}^{-1}(\nabla u, \nabla v)_{L^2},
\end{aligned}
$$

respectively. In the case of $P_4$ the reduction of the system means we only require two inner products. The preconditioner $P_4$ fared marginally better than $P_1$ probably because $P_4$ requires no approximation of a Schur complement.

We observe in particular in sections 4.2 and 4.3 that the inner products

$$
\begin{aligned}
(u,v)_{Y_h} &= (u,v)_{L^2},\\
(u,v)_{U_h} &= (u,v)_{L^2}\\
(u,v)_{P_h} &= \alpha(u,v)_{L^2} + (\Delta u, \Delta v)_{(L^2)^*}
\end{aligned}
$$

and

$$
\begin{aligned}
(u,v)_{Y_h} &= (u,v)_{L^2},\\
(u,v)_{U_h} &= \alpha(u,v)_{L^2}\\
(u,v)_{P_h} &= (\Delta u, \Delta v)_{(L^2)^*}
\end{aligned}
$$

for $P_2$ and $P_3$ respectively with their lack of $\alpha$ compensation have severe effect in the rate of convergence for the optimization algorithm.

Finally we observe in section 4.4 how steep discontinuities does not visibly affect the convergence rate for the problem with a proper preconditioner.

## 5.1 Future outlook

Here we list some future works that could possible be extended from this.

### 5.1.1 Schur complement generalizations

As briefly mentioned in the conclusions above and seen in the last parts of the theory chapter we initalized a study into generalizations of the Schur complement in a pursuit to obtain a preconditioner for the operator setting analogous to the one in the

discretized setting. While we did not succeed with the extension, we did take some first steps that could be considered as a basis for further work.

A generalization of the Schur complement to Hilbert space operator matrices $M : H \to H^*$ was established in section 2.6.5.3, however, a generalization of Lemma 18 was not obtained here. We recon a more in depth study involving generalization of eigenvalues to operators $M : H \to H^*$ might lead to a definition of generalized intertia similar to the one in [Cai80]. With this established generalized results in vein with Lemma 18 could possibly be obtained, which would allow further process in what have been started here.

### 5.1.2 Other model problems

In this thesis the focus has been on one particular model problem, however, many other model problems could be considered. Futher study could involve exchanging the PDE in the model problem for

$$-\Delta y + y = u,$$

could for instance be an option. In fact, the differential operator $-\Delta + 1$ has very nice spectral properties that could be taken advantage of.

Moreover one could study the effectiveness of these preconditioners under pertubation by non-linear terms. Say,

$$-\Delta y + \beta F(y) = u,$$

where $\beta$ is a small tunable constant and $F$ is a non-linear map.

Another option could be to move away from the distributed control problem and consider a boundary control problem where the PDE could be

$$\begin{aligned} Dy &= 0 \quad \text{in } \Omega \\ y &= u \quad \text{on } \partial\Omega \end{aligned}$$

or

$$\begin{aligned} Dy &= 0 \quad \text{in } \Omega \\ \frac{\partial y}{\partial n} &= u \quad \text{on } \partial\Omega, \end{aligned}$$

where $D$ is some differential operator.

There are a number of options available for further study in this topic.

# APPENDIX A

# Problem statement

The study of optimization has been a popular topic in mathematics and with the growth of the processing power of computers in the last two decades ever more complex problems have been practically available. In general optimization concerns itself with solving problems of the form

$$\begin{cases} \min\limits_{x \in X} & f(x), \\ \text{subj. to} & g(x) \leq 0 \end{cases} \tag{A.1}$$

where $f : X \to \mathbb{R}$ is the function we wish to minimize over some normed vector space $X$ and $g$ defines constraints on $x$. Traditionally $X$ have been chosen to be of finite dimension. However, in recent years the interest in solving problems over spaces of infinite dimension such as $L^p$-spaces and more generally Sobolev spaces is of growing popularity. In practice solving is done through a discretization of the problem, a projection from $X$ down to a finite dimensional subspace $\tilde{X}$, where our prior know-how applies.

In this thesis we will consider the problem of optimization as in (A.1), however adding an equality constraint $e(x) = 0$ involving a partial differential equation (PDE). The complete problem then takes the form

$$\begin{cases} \min\limits_{x \in X} & f(x), \\ \text{subj. to} & e(x) = 0, \\ & g(x) \leq 0. \end{cases} \tag{A.2}$$

We note that in simpler problems the inequality constraint $g$ is often ommited. When $e(x)$ involves a PDE in $x$ as we stated above, we say that (A.2) is a *PDE-constrained optimization problem.*

A problem in PDE-constrained optimization is the the matrices arising from discretization of the problems in general happen to be severely ill-conditioned, the condition numbers skyrocketing as the mesh is refined over the domain. This is an issue

as iterative solvers require nice spectral properties of $A$ to guarantee convergence when trying to solve the system $Ax = b$. Direct solvers, the alternative option, are unfortunately too memory intensive when working with large problems. To combat this problem a proposed solution is to find a matrix $P$ such that the problem $P^{-1}Ax = P^{-1}b$, has better spectral properties than the original problem.[Her10] We call such a matrix $P$ a *preconditioner* for the system.

It turns out that for PDE-constrained problems a choice of preconditioner might be influenced on the inner product of $X$, when $X$ is a Hilbert space.[GHS14; Zul11] What we would like to explore in this thesis is the significance of the innert product influenced preconditioners.

> How does the underlying structure of the search space $X$ influence performance with and without preconditioners in PDE-constrained optimization? – and what about practical applications i.e. electrical impedance tomography?

## A.1 Learning objectives

As part of a master thesis a number of generic learning objectives are imposed by DTU[DTU16]. Throughout this thesis, those overarching objectives will be pursued as part of the following personal objectives.

When done with this thesis the author should – in no particular order – be able to:

- write and understand code in the programming language Python.
- use the Python software package FEniCS for solving PDE-problems.
- implement optimization algorithms for PDE-constrained optimization using FEniCS.
- show understanding of optimization theory and in particular the theory of PDE-constrained optimization.
- understand and explain the merits of using preconditioners in optimization.
- apply theory to solve PDE-constrained optimization problems.
- find litterature relevant for PDE-constrained optimization and paint a coherent overview with a basis in the following articles:

– “A note on preconditioners and scalar products in Krylov subspace methods for self-adjoint problems in Hilbert space” by Günnel, Herzog, and Sachs. [GHS14]

– “A Globally Convergent Algorithm for a PDE-Constrained Optimization Problem Arising in Electrical Impedance Tomography” by Carrillo and Gómez. [CG15]

## A.2 Time schedule

The following is the initial time schedule for the work to be done throughout this thesis. It should be noted that in February and March a 7,5 ECTS course in Differential Operators and Function spaces (DifFun) will be followed at the University of Copenhagen (KU), and thus the expected time dedicated to the thesis is considerably lower during this time.

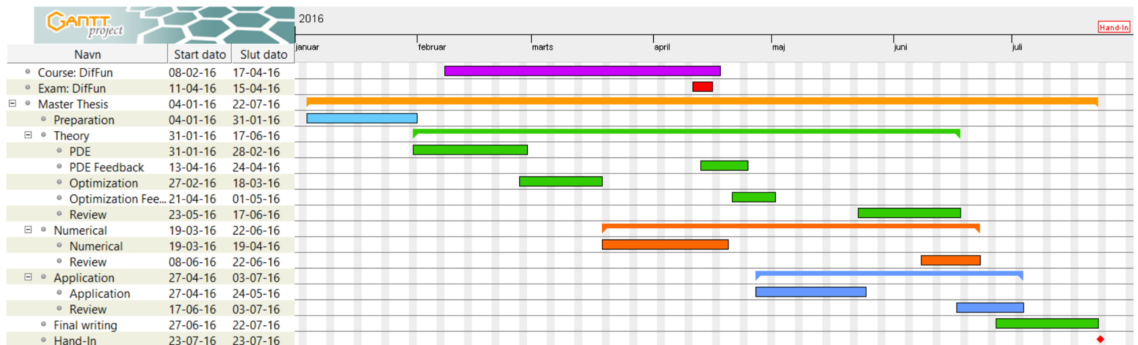


**Figure A.1:** Time table.

## A.3 Reflections

Here we will reflect on how the project overall. What we had to change, and what we might have liked to change in retrospect.

### A.3.1 Time schedule

The time table proposed here in the problem statement was an initial guess at how everything was supposed to go. We like to think that the schedule actually held up reasonably well, though there were some difinite shifts towards the end where writing and numerical application became more of a simlutaneous process. Also, the still not completely solved problem of preconditioners in infinite dimensions pressed the theory works to go on until early July, shifting the final writing to first really start in July.

Keeping up the workflow during the time during when we followed the DifFun course at the University of Copenhagen was significantly harder than expected. In retrospect it is too easy to let a sideactivity like this consume your time, but on the other hand we feel that the break-off from the thesis work probably was a good thing overall. The course also helped a lot for understanding theoretical aspects of the thesis better, so even now we would definitely not have cut it out of the process.

Repeating from the first paragraph, we like to think the schedule actually held up rather well in practice. Better than we had dared to hope initially, being an idealization.

### A.3.2 FEniCS (Python) vs. MATLAB

One of the most time consuming tasks in this project was to get a general understanding about how to use FEniCS and more importantly, when FEniCS is no longer useful. While FEniCS is a great tool for solving PDE-problems, that is really all it does. It was an initial hope that FEniCS could be used for most of the computations in this project and thus let everything be done within the FEniCS framework. This turned out to not really be possible, however.

While documentation is overall fairly good for FEniCS when you just want to solve a PDE, it should be noted this finding very specific things can be complicated. Also, due to a complete rewrite of certain functionality in later versions of FEniCS, some parts of the documentation has simply been outdated while this project was in progress. Thus, while FEniCS is certainly a powerful tool, it is definitely reasonable to ask if any time was saved by using FEniCS, or if we should simply have written our own code.

In the end FEniCS was abbandoned for the optimization part of the project and solely used for mesh and basis function generation, as well as matrix assembly. This the package handles very well. However, had we known of the IFISS library for

MATLAB from the beginning of the project we would likely have prefered MATLAB over FEniCS.

As advice for a future student taking on a topic in PDE-constrained optimization, we would suggest the student to work with MATLAB using the IFISS library as a basis if the student is more familiar with MATLAB. The demos on the topic packaged with the library are also very informative. We would not suggest anyone completely new to FEniCS to use over IFISS if the topic is PDE-constrained optimization.

### A.3.3 Electrical Impedance Tomography (EIT)

Something a reader will probably notice, when reading the problem statement and the rest of the thesis, is the distinct lack of anything relating to EIT in the main body of the thesis though it was part of the problem formulation to apply the theory to this particular problem.

Up until past midway through the project we still expected to apply the theory on EIT. We later realized though, that EIT really is a quite different problem compared to the distributed control model problem we started out with. More different than we expected.

The in EIT we assume to know the Dirichlet-to-Neumann map $\Lambda_\sigma : f \to g$ for a domain $\Omega$ and using this we wish to find the unknown quantity $\sigma$. In physical terms we have the PDE

$$\nabla \cdot (\sigma \nabla u) = 0$$

governing the behaviour of electricity in a body $\Omega$, where $u : \Omega \to \mathbb{R}$ describes the electrostatic potential and $\sigma : \Omega \to \mathbb{R}$ the conductivity. We have then “conducted experiments” by applying voltages $f : \partial\Omega \to \mathbb{R}$ along the boundary $\partial\Omega$ and measured the currents $g : \partial\Omega \to \mathbb{R}$ at the boundary. $f$ should satisfy $\int_{\partial\Omega} f \, dx = 0$.

The problem we encountered was that this did not transfer well to a setup similar to our model problem. We don’t immediately have a goal functional, and being creative an coming up with one dependent on $\sigma$ will generate some rather nasty non-liniearities in the Lagrangian and KKT-system. It was considered to fix $\sigma$ at some value, solve the problem and update iteratively, which squelches the non-liniearities. However, that would not be very relatable to the theory we had spend time working with.

In the end the EIT application was abandoned for these reasons and to let the focus remain on the core problem about preconditioners and function space structure.

# APPENDIX B
# Various mathematical results

In this appendix we will list various mathematical results which are needed in the thesis, but are not important in themselves and does not really have a natural home anywhere else.

The appendix is in two parts. The first part is simply an assortment of results. The second part covers the basics for the MINRES algorithm.

## B.1 Miscellaneous

### B.1.1 Finite element mass and stiffness matrices

**Lemma 29.** *Let $\{\phi_i\}$ be a finite element basis over the domain $\Omega$ with $V = \operatorname{span}\{\phi_i\}$, then following holds:*

*(i) The mass matrix $M_{ij} = \int_\Omega \phi_i \phi_j \, dx$ is positive definite.*

*(ii) The stiffness matrix $K_{ij} = \int_\Omega \nabla\phi_i \nabla\phi_j \, dx$ is positive semi-definite.*

*Proof.* Let $\mathbf{v} = (v_i)$ be a non-zero vector over $V$ and $v_h = \sum_i v_i \phi_i$ the corresponding function, then (i)

$$\begin{aligned}
\mathbf{v}^T M \mathbf{v} &= \sum_i \sum_j v_i M_{ij} v_j = \sum_i \sum_j v_i \left( \int_\Omega \phi_i \phi_j \, dx \right) v_j \\
&= \int_\Omega \sum_i v_i \phi_i \sum_j v_i \phi_j \, dx = \int_\Omega \left( \sum_i v_i \phi_i \right)^2 dx = \int_\Omega |v_h|^2 \, dx > 0,
\end{aligned}$$

and likewise (ii)

$$\begin{aligned}\mathbf{v}^T K \mathbf{v} &= \sum_i \sum_j v_i K_{ij} v_j = \sum_i \sum_j v_i \left( \int_\Omega \nabla\phi_i \nabla\phi_j \, dx \right) v_j \\ &= \int_\Omega \sum_i v_i \nabla\phi_i \sum_j v_i \nabla\phi_j \, dx = \int_\Omega \left( \sum_i v_i \nabla\phi_i \right)^2 dx = \int_\Omega |\nabla v_h|^2 \, dx \geq 0.\end{aligned}$$

□

**Remark 30.** The stiffness matrix is positive definite on the subspace of vectors resulting from Dirichlet boundary conditions restricting our space.

### B.1.2 Operators: eigenvalues and orderings

**Lemma 31.** *Let $A : X \to Y$ and $B : Y \to X$ be bounded operators. Then $AB : Y \to Y$ and $BA : X \to X$ has the same eigenvalues.*

*Proof.* Let $\lambda_i \in \mathbb{C}$ be an eigenvalue of $AB$ and $v_i \in Y$ be a corresponding eigenvector. Let $w_i = Bv_i \in X$,

$$ABv_i = Aw_i = \lambda_i v_i.$$

Then $BAw_i == B\lambda_i v_i = \lambda_i B v_i = \lambda_i w_i$, hence $\lambda_i$ is an eigenvalue og $BA$. The other way follows trivially. □

**Lemma 32.** *Let $A, B : H \to H$ be two bounded positive self-adjoint and invertible operators. Assume that $A - B \geq 0$, then $B^{-1} - A^{-1} \geq 0$.*

*Proof.* For positive self-adjoint operators $A$ and $B$ we have, by Theorem 9.4-2 in [Kre89], a unique positive Hermitian square roots $A^{1/2}$ and $B^{1/2}$. Now, consider $A - B \geq 0$ and multiply the equation by $B^{-1/2}$ from left and right. Then we are left with the expression $B^{-1/2}AB^{-1/2} - I \geq 0$. Hence $\sigma(B^{-1/2}AB^{-1/2}) \subset \{x \in \mathbb{R} \,|\, x \geq 1\}$.

Note that $B^{-1/2}AB^{-1/2} = (B^{-1/2}A^{1/2})(A^{1/2}B^{-1/2})$ and by Lemma 31 this operator has the same eigenvalues as $(A^{1/2}B^{-1/2})(B^{-1/2}A^{1/2}) = A^{1/2}B^{-1}A^{1/2}$, thus $A^{1/2}B^{-1}A^{1/2} - I \geq 0$ and multiplying by $A^{-1/2}$ from left and right yields $B^{-1} - A^{-1} \geq 0$ as we wanted. □

**Lemma 33.** *Let $A, B : H \to H^*$ be operators, such that $R_H A$ and $R_H B$ are satisfy the conditions of Lemma 32, where $R_H : H^* \to H$ is the Riesz isomorphism. If $A \leq B$ then $B^{-1} \leq A^{-1}$.*

*Proof.* First a note in notation, as we will need orderings of two different types of operators here we will use $A \preceq B$ for operator $A, B : H \to H$, and $A \leq B$ for operators $A, B : H \to H^*$.

Secondly, note that for operators $A, B : H^* \to H$ we have $A \leq B$ when $\langle x^*, Ax^* \rangle \leq \langle x^*, Bx^* \rangle$ for all $x^* \in H^*$.

Denote by $I_H = R_H^{-1}$, by definition

$$\begin{aligned} A &\leq B \\ \langle Ax, x \rangle &\leq \langle Bx, x \rangle \quad \text{for all } x \in H \\ (R_H Ax, x) &\leq (R_H Bx, x) \quad \text{for all } x \in H \\ R_H A &\leq_* R_H B \end{aligned}$$

By Lemma 32 this implies

$$(R_H B)^{-1} \preceq (R_H A)^{-1}.$$

As $(R_H B)^{-1} = B^{-1} R_H^{-1} = B^{-1} I_H$, by definition this is

$$(B^{-1} I_H x, x) \leq (A^{-1} I_H x, x) \quad \text{for all } x \in H.$$

Now by symmetry of the inner products, let $x^* = I_H x$, then

$$\begin{aligned} (x, B^{-1} I_H x) &\leq (x, A^{-1} I_H x) \quad \text{for all } x \in H. \\ (R_H x^*, B^{-1} x^*) &\leq (R_H x^*, A^{-1} x^*) \quad \forall x^* \in H^* \\ \langle x^*, B^{-1} x^* \rangle &\leq \langle x^*, A^{-1} x^* \rangle \quad \text{for all } x \in H \\ B^{-1} &\leq A^{-1}, \end{aligned}$$

which is what we wanted. □

### B.1.3 Fundamental Lemma of Calculus of Variations

**Lemma 34** (Fundamental Lemma of Calculus of Variations)**.** *Let $\Omega \subseteq \mathbb{R}^n$ and $f$ be a continuous function such that*

$$\int_\Omega f h \, dx = 0$$

*for all $h \in C_0^\infty(\Omega)$, then $f = 0$.*

*Proof.* Assume the contrary, that there is $x \in \Omega$ such that $f(x) \neq 0$, say $f(x) = \alpha > 0$, then as $f$ is continuous there exists an open ball about $x$, $B_\delta(x) \subset \Omega$, $\delta > 0$, such that $\inf f(B_\delta(x)) = \beta > 0$.

For an open ball such as $B_\delta(x)$ we may find a non-negative function $h \in C_0^\infty(B_\delta(x)) \subset C_0^\infty(\Omega)$, then

$$\int_\Omega fh\,dx = \int_{B_\delta(x)} fh\,dx \geq \beta \int_{B_\delta(x)} h\,dx > 0,$$

which contradict our assumption on $f$. □

## B.2 Krylov subspace optimization

This section follows the process described in the lecture notes

`http://persson.berkeley.edu/18.335/toledo_krylov.pdf`[1],

though with a number of our own considerations and corrections of errors found in the document.

First off, a Krylov subspace is defined as follows.

**Definition 35.** Given an optimization problem $Ax = b$ the *Krylov subspace* $\mathcal{K}_t(A, b)$ is defined by

$$\mathcal{K}_t(A, b) := \operatorname{span}\{b, Ab, A^2b, \ldots, A^{t-1}b\}$$

Why does it make sense to search in Krylov subspaces? The following Lemma shows why it makes sense to search in a sequence of increasing Krylov subspaces.

**Lemma 36.** *Suppose the set of vectors $V := \{b, Ab, A^2b, \ldots, A^kb\}$ are not linearly independent vectors. Then the solution $x$ to the problem $Ax = b$ is in $V \backslash \{A^kb\}$.*

*Proof.* If the set of vectors $V$ are not linearly independent, then there are $\alpha_j \in \mathbb{R}$ such that

$$b = \sum_{j=1}^{k} \alpha_j A^j b = A \left( \sum_{j=0}^{k-1} \alpha_{j+1} A^j b \right).$$

Hence $x = \sum_{j=0}^{k-1} \alpha_{j+1} A^j b$ solves $Ax = b$. □

[1] MIT Course 18.335: Introduction to Numerical Methods (Fall 2007) `http://persson.berkeley.edu/18.335/`

### B.2.1 An orthogonal basis

Now consider the orthonormal basis $v_1, v_2, \ldots, v_t$ for the $t$'th Krylov subspace $\mathcal{K}_t(A, b)$. If $t = 1$, then clearly $v_1 = b/\|b\|$. More generally, let $K_t = [b, Ab, A^2b, \ldots, A^{t-1}b]$ be the matrix defining our $t$'th Krylov subspace, and let $V_t = [v_1, v_2, \ldots, v_t]$ be an orthonormal basis for $\mathcal{K}_t(A, b)$. We will now inductively build up our basis.

There is a matrix $R_t$ such that $K_t = V_t R_t$ since the columns of $V_t$ and $K_t$ spans the same subspace. We may thus write

$$A^{t-1}b = \sum_{j=1}^{t} r_{j,t} v_j.$$

Hence by isolating $v_t$ we get

$$\begin{aligned} v_t &= r_{t,t}^{-1} A^{t-1} b - r_{t,t}^{-1} \sum_{j=1}^{t-1} r_{j,t} v_j \\ A v_t &= r_{t,t}^{-1} A^{t} b - r_{t,t}^{-1} \sum_{j=1}^{t-1} r_{j,t} A v_j. \end{aligned}$$

By Lemma 36 $Av_t \in \mathcal{K}_t(A, b)$ only if the solution $x$ for $Ax = b$ is in $\mathcal{K}_t(A, b)$. If this was the case we would have found the solution and been done already, thus we may assume $x \notin \mathcal{K}_t(A, b)$ and thus $Av_t \notin \mathcal{K}_t(A, b)$. Hence we can orthogonalize $Av_t$ with the existing basis $V_t$ using Gram-Schmidt procedure and get a basis for the new space.

As $V_t$ was already an orthogonal basis we only need to apply the algorithm to the new vector $Av_t$. By Gram-Schmidt new basis element $v_{t+1}$ is

$$\tilde{v}_{t+1} = Av_t - \sum_{j=1}^{t} \mathrm{proj}_{v_j}(Av_t),$$

where $\mathrm{proj}_u(v) = (v, u)\widehat{u}$ with $\widehat{u}$ denoting the normalization of $u$. Thus $\mathrm{proj}_{v_j}(Av_t) = (Av_t, v_j)v_j = (v_j^T A v_t)v_j$ and therefore

$$\begin{aligned} \tilde{v}_{t+1} &= Av_t - \sum_{j=1}^{t} (v_j^T A v_t) v_j \\ v_{t+1} &= \tilde{v}_{t+1} / \|\tilde{v}_{t+1}\|. \end{aligned}$$

Because of this

$$\begin{aligned} Av_t &= \|\tilde{v}_{t+1}\| v_{t+1} + \sum_{j=1}^{t} (v_j^T A v_t) v_j \\ &= h_{t+1,t} v_{t+1} + \sum_{j=1}^{t} h_{j,t} v_j, \end{aligned}$$

with $h_{t+1,t} = \|\tilde{v}_{t+1}\|$ and $h_{j,t} = v_j^T A v_t$ for $1 \leq j \leq t$. Hence

$$AV_t = V_{t+1} H_t, \quad H_t = \begin{bmatrix} v_1^T A v_1 & v_1^T A v_2 & v_1^T A v_3 & v_1^T A v_4 & \dots \\ \|\tilde{v}_2\| & v_2^T A v_2 & v_2^T A v_3 & v_2^T A v_4 & \dots \\ 0 & \|\tilde{v}_3\| & v_3^T A v_3 & v_3^T A v_4 & \dots \\ 0 & 0 & \|\tilde{v_4}\| & v_4^T A v_4 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \in \mathbb{R}^{(t+1)\times t}.$$

Assuming $A$ was symmetric, clearly $V_t^T A V_t$ is symmetric as well, thus

$$V_t^T A V_t = V_t^T V_{t+1} H_t = T_t$$

is symmetric. Now $V_t^T V_{t+1} = [I_t | \mathbf{0}]$, where $\mathbf{0} \in \mathbb{R}^t$, thus $T_t = (h_{i,j})_{1 \leq i,j \leq t}$. Let $\delta_j = v_j^T A v_j$ and $\gamma_j = v_{j-1}^T A v_j$. Clearly by symmetry we also have $\gamma_j = \|\tilde{v}_j\|$. Thus

$$T_t = \begin{bmatrix} \delta_1 & \gamma_2 & 0 & \dots & 0 \\ \gamma_2 & \delta_2 & \gamma_3 & & 0 \\ 0 & \gamma_3 & \delta_3 & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \gamma_t \\ 0 & 0 & \dots & \gamma_t & \delta_t \end{bmatrix}.$$

In conclusion we find that we need only remember a few of the vectors for each iteration:

$$v_{t+1} = Av_t - \delta_t v_t - \gamma_{t-1} v_{t-1}.$$

### B.2.2 MINRES

The MINRES algorithm seeks to solve the problem $Ax = b$ where $A$ and $b$ are the known quantities. This is done by searching iteratively for the solution $x_k$ in the shifted Krylov subspace $b + A\mathcal{K}_k(A, b)$ with minimal residual $r_k$. In general the

actual space is $r_0 + A\mathcal{K}_k(A, r_0)$, $r_0$ being the first risidual, but taking the initial guess $x_0 = 0$ sets $r_0 = b - Ax_0 = b$. We will work with this initial guess.

We note that $r_k = b - Ax_k$. Let $K_k = [b, Ab, \dots, A^{k-1}b]$ then

$$\begin{aligned}\min_{x\in\mathcal{K}_k} \|b - Ax\|_2 &= \min_{y\in\mathbb{R}^k} \|b - AK_k y\|_2 \\ &= \min_{z\in\mathbb{R}^k} \|b - AV_k z\|_2,\end{aligned}$$

where $z = R_k y$ using the equation $K_k = V_k R_k$ from the previous section. Furthermore by $AV_k = V_{k+1}H_k$

$$= \min_{z\in\mathbb{R}^k} \|b - V_{k+1}H_k z\|_2.$$

As $V_{k+1}$ is unitary, i.e. $V_{k+1}^T V_{k+1} = I_{k+1}$, we have for $x \in \mathbb{R}^{k+1}$

$$\|x\|_2^2 = x^T x = x^T I_{k+1} x = x^T V_{k+1}^T V_{k+1} x = (V_{k+1}x)^T (V_{k+1}x) = \|V_{k+1}x\|_2^2,$$

Thus

$$\begin{aligned}\min_{z\in\mathbb{R}^k} \|b - V_{k+1}H_k z\|_2 &= \min_{z\in\mathbb{R}^k} \|V_{k+1}^T b - H_k z\|_2 \\ &= \min_{z\in\mathbb{R}^k} \|\|b\|_2 e_1 - H_k z\|_2,\end{aligned}$$

since $v_1 = b/\|b\|_2$, and therefore $V_{k+1}b = \|b\|_2 V_{k+1} v_1 = \|b\|_2 e_1$.

As $H_k = [T_k|\gamma_{k+1}e_k]^T$ (since $T_k$ is symmetric) $H_k$ is a band matrix with only entries in the diagonal and first upper and lower diagonals. A matrix like this can be QR factorized using Givens rotations. A Givens rotation is a rotation such that a vector $(a, b)$ is rotated into $(r, 0)$, with $r = \sqrt{a^2 + b^2}$, that is

$$G\begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}, \quad \text{where } G = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}.$$

This extend naturally to higher dimensions

$$G\begin{bmatrix} \mathbf{0}_{\ell_1} \\ a \\ b \\ \mathbf{0}_{\ell_2} \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{\ell_1} \\ r \\ 0 \\ \mathbf{0}_{\ell_2} \end{bmatrix}, \quad \text{where } G = \begin{bmatrix} I_{\ell_1} & 0_{\ell_1,2} & 0_{\ell_1,\ell_2} \\ 0_{2,\ell_1} & \begin{matrix} c & s \\ -s & c \end{matrix} & 0_{2,\ell_2} \\ 0_{\ell_2,\ell_1} & 0_{\ell_2,2} & I_{\ell_2} \end{bmatrix}.$$

Here $\mathbf{0}_\ell \in \mathbb{R}^\ell$ is a zero vector and $0_{\ell,k} \in \mathbb{R}^{\ell\times k}$ a zero matrix. We note that there is a more general difinition of Givens rotations, but if we apply the rotations in the correct order, we won't need it here.

By composition of several Givens rotations $G_1, G_2, \ldots, G_\ell$ we may obtain a QR factorization for $H_k$. Thus $G_1 G_2 \ldots G_\ell H_k = R_k$ where $U_k$ is upper triangular. Now set $Q_k^T = G_1 G_2 \ldots G_\ell$, then $H_k = Q_k U_k$. We note that $Q_k \in \mathbb{R}^{(k+1)\times(k+1)}$ and $U_k \in \mathbb{R}^{(k+1)\times k}$, however, as $U_k$ is upper triangular we have

$$H_k = Q_k U_k = \begin{bmatrix} \widehat{Q}_k & q_{k+1} \end{bmatrix} \begin{bmatrix} \widehat{U}_k \\ \mathbf{0}_k^T \end{bmatrix} = \widehat{Q}_k \widehat{U}_k,$$

where $\widehat{Q}_k \in \mathbb{R}^{(k+1)\times k}$ and $\widehat{U}_k \in \mathbb{R}^{k\times k}$ and $q_{k+1}$ is the $(k+1)$'th column in $Q_k$. We note that $\widehat{U_k}$ is upper triangular like $U_k$ and $\widehat{Q}_k^T \widehat{Q}_k = I_k$.

From here we redefine $Q_k := \widehat{Q}_k$ and $U_k := \widehat{U_k}$, then

$$\begin{aligned}\min_{z\in\mathbb{R}^k} \| \|b\|_2 e_1 - H_k z\|_2 &= \min_{z\in\mathbb{R}^k} \| \|b\|_2 e_1 - Q_k U_k z\|_2 \\ &= \min_{z\in\mathbb{R}^k} \| \|b\|_2 Q_k^T e_1 - U_k z\|_2,\end{aligned}$$

and we may solve $U_k z = \|b\|_2 Q_k^T e_1$ for $z$.

While it is possible to solve for $z$ like this and then $x = V_k z$, this would require storing all columns of $V_k$ in memory or recomputing them at each step. This is not very efficient, thus the following computation is usually used instead. Note first that

$$x = V_k z = V_k U_k^{-1} U_k z = M_k w, \quad \text{where } M_k = V_k U_k^{-1} \text{ and } w = U_k z = \|b\|_2 Q_k^T e_1.$$

Furthermore, since $H_k$ was tridiagonal we need only remove the lower diagonal element in each column, hence reacalling that $H_k \in \mathbb{R}^{k+1} \times k$ we have $\ell = k$. Moreover, let $G_i$ be the Givens rotation matrix removing the lower diagonal element from the $i$'th column. Then $G_i$ affects only rows $i$ and $i+1$ in $H_k$, hence $R_k$ can at most be tridiagonal, the nonzero diagonals being the regular diagonal and the first and second upper diagonals.

Then $M_k U_k = V_k$. Let $m_i$ be the $i$'th column in $M_k$, then since the $i$'th column of $U_k$ is $(\mathbf{0}_{i-3}, u_{i-2,i}, u_{i-1,i}, u_{i,i}, \mathbf{0}_{k-i})^T$ we have

$$u_{i,i} m_i = v_i - u_{i-1,i} m_{i-1} - u_{i-2,i} m_{i-2}.$$

where negative and zero subscripted elements are taken to be 0. Thus we may efficiently built $M_k$ column by column. We note that by the structure of the matrices each iteration steps only changes very little, mostly extending the matrices following the same patterns.

The biggest change is that where $Q_k = G_1 G_2 \ldots G_k$ before, now $Q_{k+1} = G_1 G_2 \ldots G_{k+1} = \tilde{Q}_k G_{k+1}$, where $\tilde{Q}_k$ is $Q_k$ with an added column and row having 1 i the diagonal element. This is because $G_{k+1}$ will affect 2 rows in $\tilde{Q}_k$.

Hence we can compute all the matrices column by column rotation by rotation one in each step and we need only store 3 columns each step of the way.

### B.2.3 Code: Preconditioned MINRES algorithm

Python implementation of Algorithm 3.1 in [GHS14].

```
import numpy as np

# Scipy sparse implementation of preconditioned MINRES
def minres(A, b, P, maxit=500, tol=1e-6):
    if b.ndim > 1:
        print("b is not a 1-d vector")
        return

    n = A.shape[0]
    xk = np.zeros(n)
    xkm1 = xk.copy()
    vk = b - A.dot(xk)
    zk = P.solve(vk)
    gamk = np.sqrt(np.dot(vk,zk))

    zk = zk/gamk
    vk = vk/gamk

    # init more
    erro = gamk
    k = 1

    vkm1 = 0
    ck = 1
    ckm1 = 1
    sk = 0
    skm1 = 0
    wk = 0
    wkm1 = 0
    nukm1 = gamk

    resvec = np.zeros(maxit + 1)
    resvec[0] = erro

    while k-1 < maxit and erro > tol:
        Azk = A.dot(zk)
        deltak = np.dot(Azk,zk)

```

```
39          vkp1 = Azk - deltak*vk - gamk*vkm1
40
41          zkp1 = P.solve(vkp1)
42          gamkp1 = np.sqrt(np.dot(vkp1,zkp1))
43          zkp1 = zkp1/gamkp1
44          vkp1 = vkp1/gamkp1
45
46          alph0 = ck * deltak - ckm1 * sk * gamk
47          alph2 = sk * deltak + ckm1 * ck * gamk
48          alph3 = skm1*gamk
49          alph1 = np.sqrt(alph0 ** 2 + gamkp1 ** 2)
50
51          ckp1 = alph0/alph1
52          skp1 = gamkp1/alph1
53
54          wkp1 = (1/alph1)*(zk-alph3*wkm1 - alph2*wk)
55
56          xk = xkm1 + ckp1*nukm1*wkp1
57          nuk = -skp1*nukm1
58
59          k = k + 1
60          ckm1 = ck
61          ck = ckp1
62          skm1 = sk
63          sk = skp1
64          gamk = gamkp1
65          wkm1 = wk
66          wk = wkp1
67          nukm1 = nuk
68          xkm1 = xk.copy()
69          vkm1 = vk.copy()
70          vk = vkp1.copy()
71          zk = zkp1.copy()
72          erro = abs(nuk)
73          resvec[k-1] = erro
74
75      if erro > tol:
76          flag = 1
77          itr = k-1
78      else:
79          flag = 0
80          itr = k-1
81
82      return (xk, itr, flag, resvec[0:k])
```

minres_sparse.py.

APPENDIX C

# Additional results

In this appendix we will show solution and residual plots for other values of $\alpha$ and $h$. We won't show plots for all the runs as $5 \cdot 24 = 120$ is a bit many, but we will try to get a diverse selection going.

All the plots will be for $h = 2^{-6}$, and then varying $\alpha$-values. The plots are best viewed in the PDF-version of the Thesis, where zoom is available, as they have been retained small to not take up unnecessarily large amounts of space.

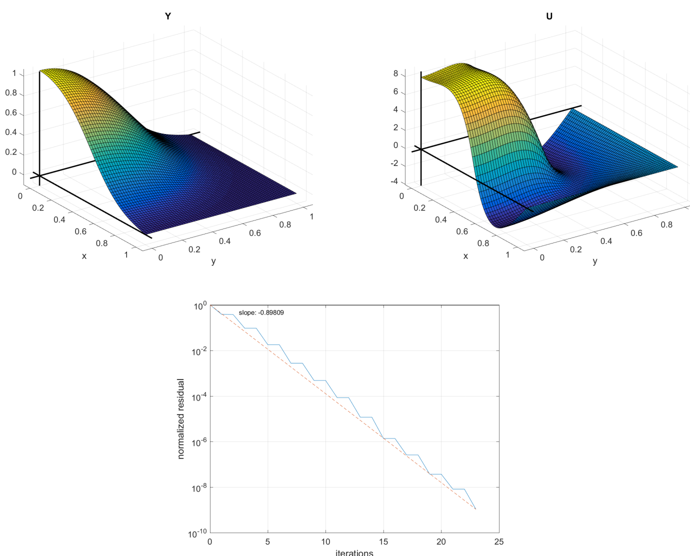


**Figure C.1:** Solutions to the core setup, state $y$ to the left, control $u$ to the right. This solution is for the case $\alpha = 10^{-3}$ and $h = 2^{-6}$.

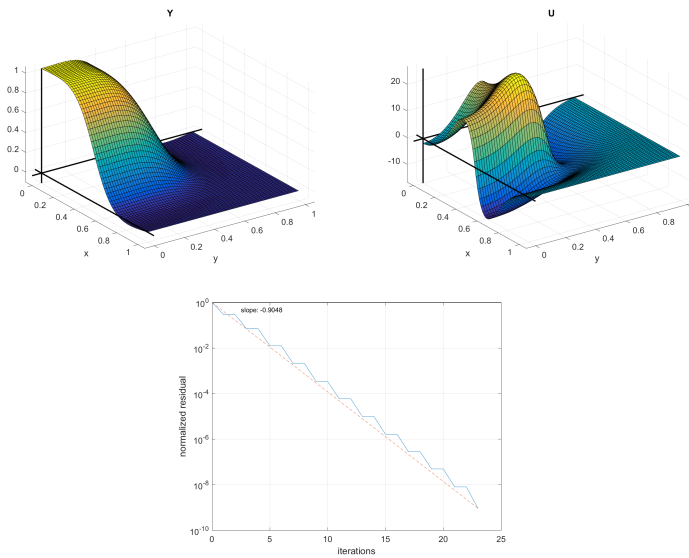


**Figure C.2:** Solutions to the core setup, state $y$ to the left, control $u$ to the right. This solution is for the case $\alpha = 10^{-4}$ and $h = 2^{-6}$.

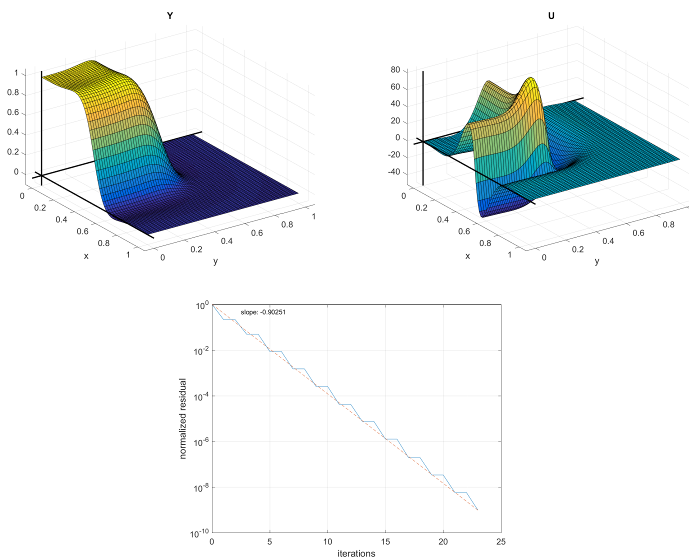


**Figure C.3:** Solutions to the core setup, state $y$ to the left, control $u$ to the right. This solution is for the case $\alpha = 10^{-5}$ and $h = 2^{-6}$.

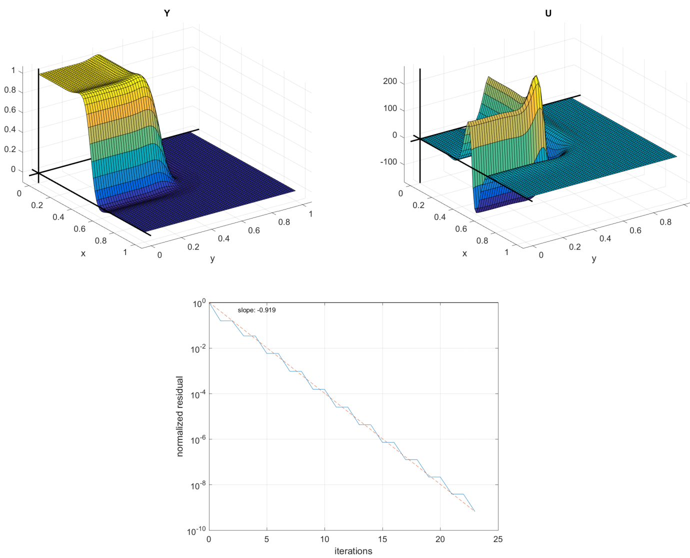


**Figure C.4:** Solutions to the core setup, state $y$ to the left, control $u$ to the right. This solution is for the case $\alpha = 10^{-6}$ and $h = 2^{-6}$.

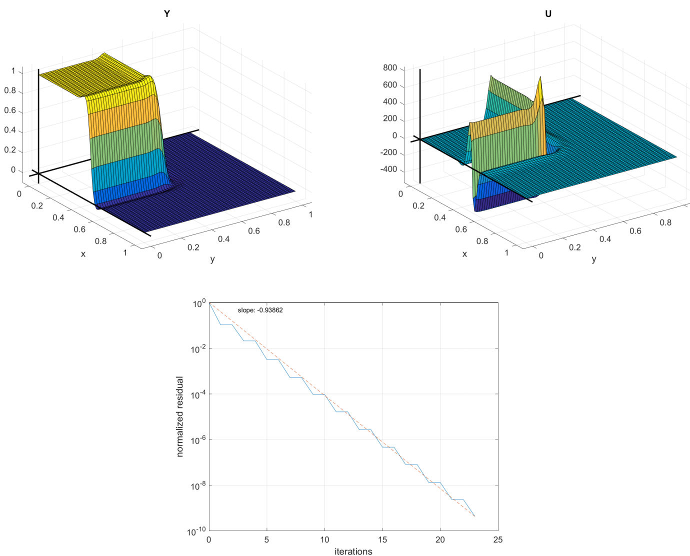


**Figure C.5:** Solutions to the core setup, state $y$ to the left, control $u$ to the right. This solution is for the case $\alpha = 10^{-7}$ and $h = 2^{-6}$.

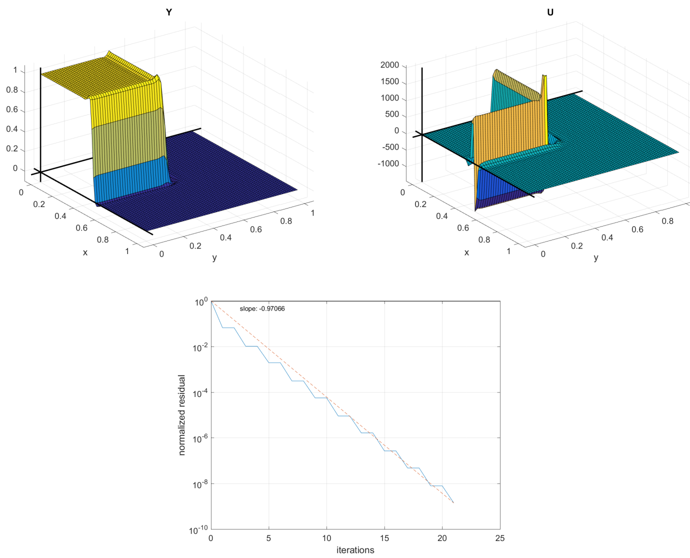


**Figure C.6:** Solutions to the core setup, state $y$ to the left, control $u$ to the right. This solution is for the case $\alpha = 10^{-8}$ and $h = 2^{-6}$.

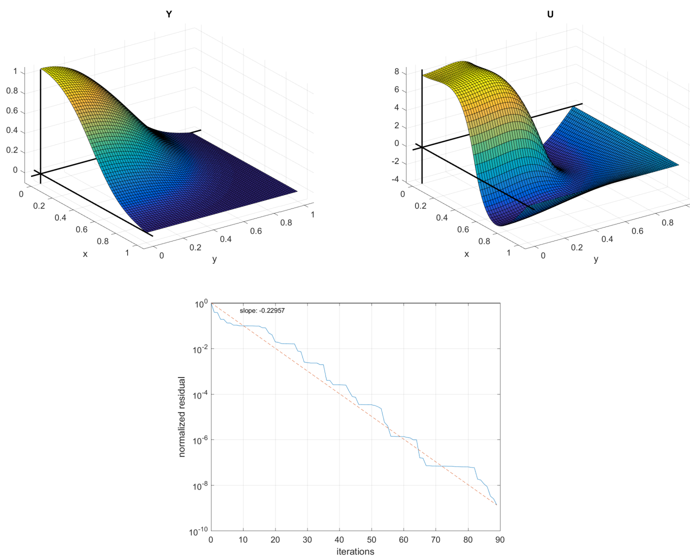


**Figure C.7:** Solutions to the setup in Section 4.2, state $y$ to the left, control $u$ to the right. This solution is for the case $\alpha = 10^{-3}$ and $h = 2^{-6}$.

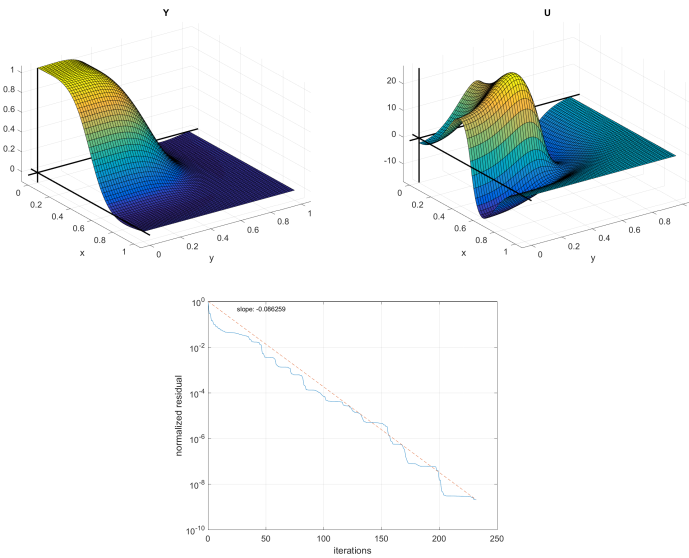


**Figure C.8:** Solutions to the setup in Section 4.2, state $y$ to the left, control $u$ to the right. This solution is for the case $\alpha = 10^{-4}$ and $h = 2^{-6}$.

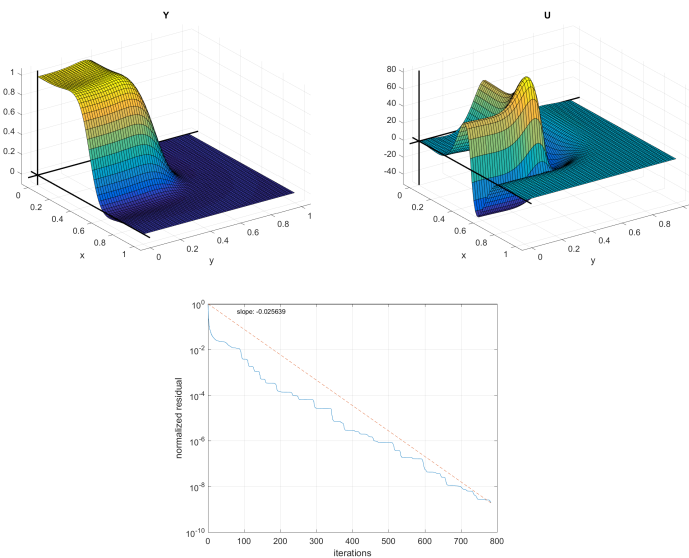


**Figure C.9:** Solutions to the setup in Section 4.2, state $y$ to the left, control $u$ to the right. This solution is for the case $\alpha = 10^{-5}$ and $h = 2^{-6}$.

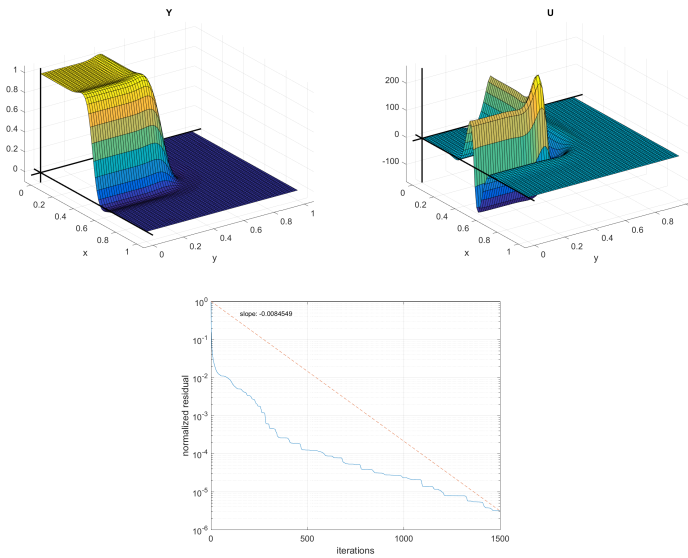


**Figure C.10:** Solutions to the setup in Section 4.2, state $y$ to the left, control $u$ to the right. This solution is for the case $\alpha = 10^{-6}$ and $h = 2^{-6}$.

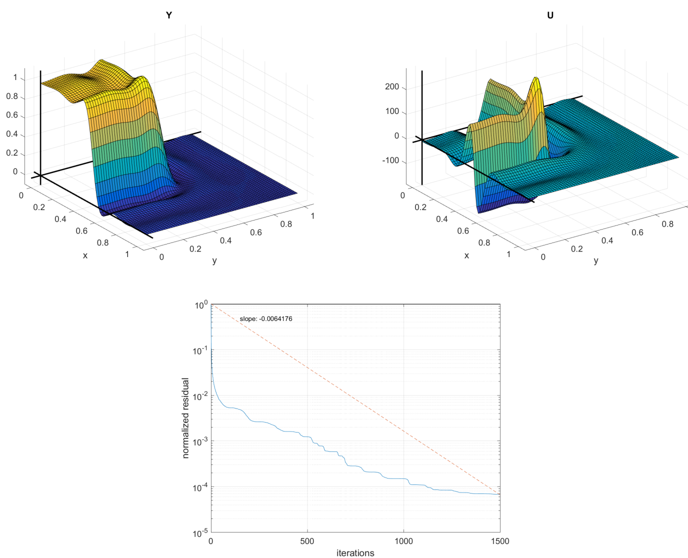


**Figure C.11:** Solutions to the setup in Section 4.2, state $y$ to the left, control $u$ to the right. This solution is for the case $\alpha = 10^{-7}$ and $h = 2^{-6}$.

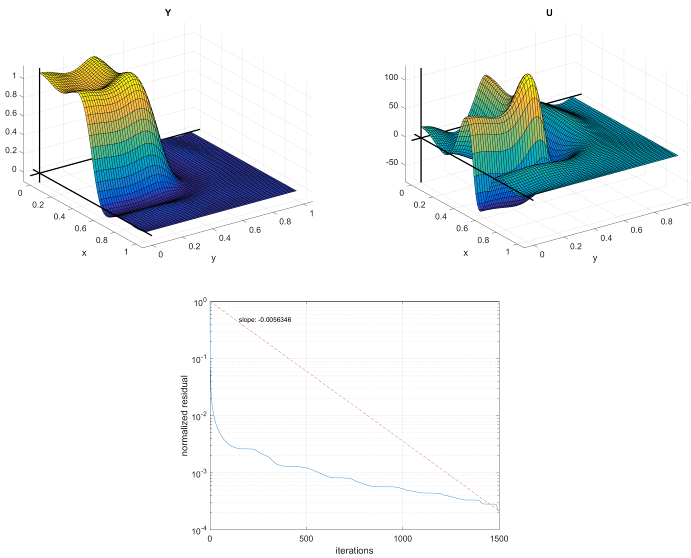


**Figure C.12:** Solutions to the setup in Section 4.2, state $y$ to the left, control $u$ to the right. This solution is for the case $\alpha = 10^{-8}$ and $h = 2^{-6}$.

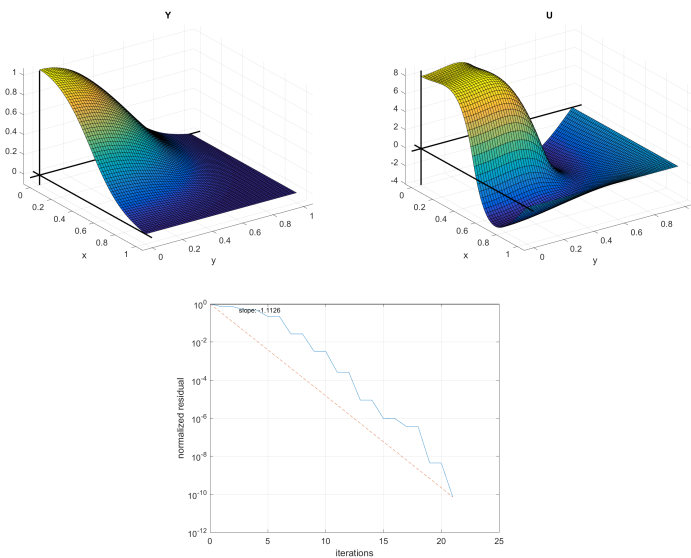


**Figure C.13:** Solutions to the setup in Section 4.3, state $y$ to the left, control $u$ to the right. This solution is for the case $\alpha = 10^{-3}$ and $h = 2^{-6}$.

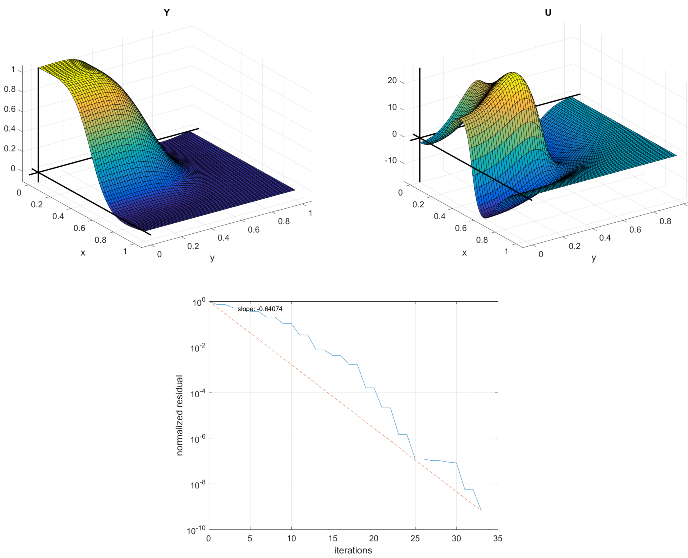


**Figure C.14:** Solutions to the setup in Section 4.3, state $y$ to the left, control $u$ to the right. This solution is for the case $\alpha = 10^{-4}$ and $h = 2^{-6}$.

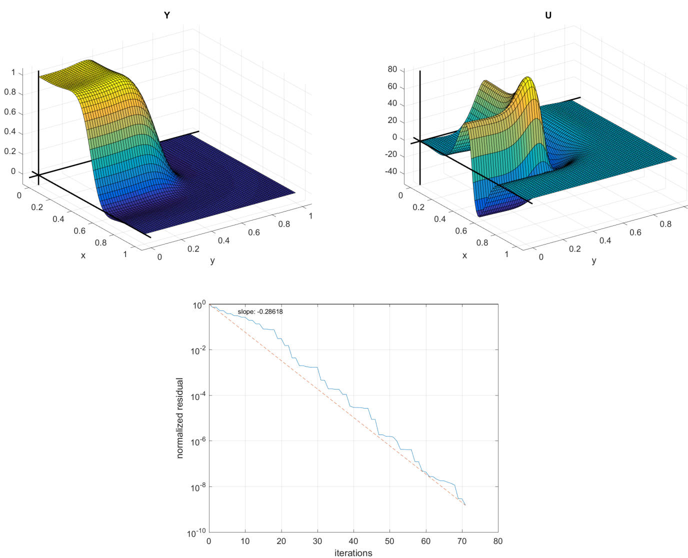


**Figure C.15:** Solutions to the setup in Section 4.3, state $y$ to the left, control $u$ to the right. This solution is for the case $\alpha = 10^{-5}$ and $h = 2^{-6}$.

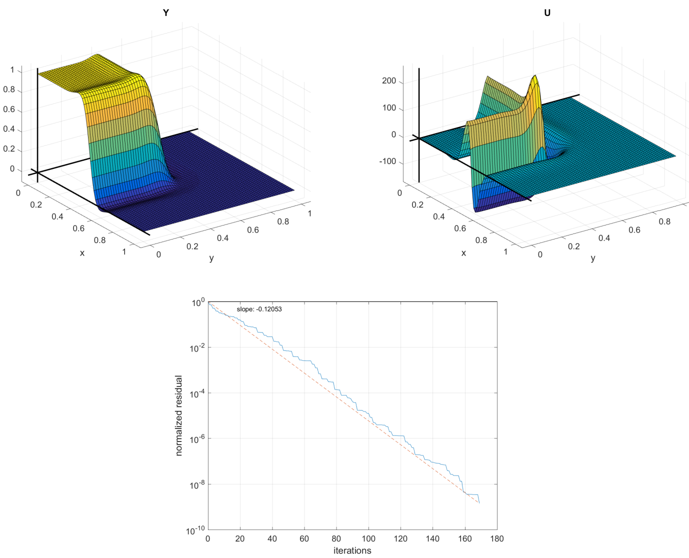


**Figure C.16:** Solutions to the setup in Section 4.3, state $y$ to the left, control $u$ to the right. This solution is for the case $\alpha = 10^{-6}$ and $h = 2^{-6}$.

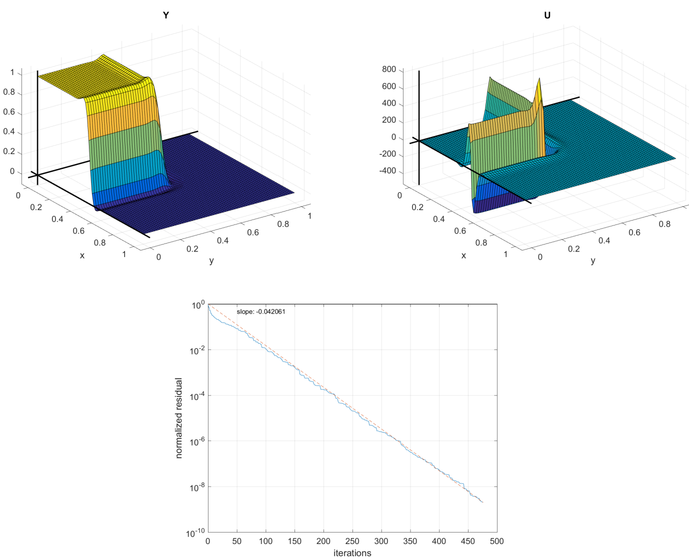


**Figure C.17:** Solutions to the setup in Section 4.3, state $y$ to the left, control $u$ to the right. This solution is for the case $\alpha = 10^{-7}$ and $h = 2^{-6}$.

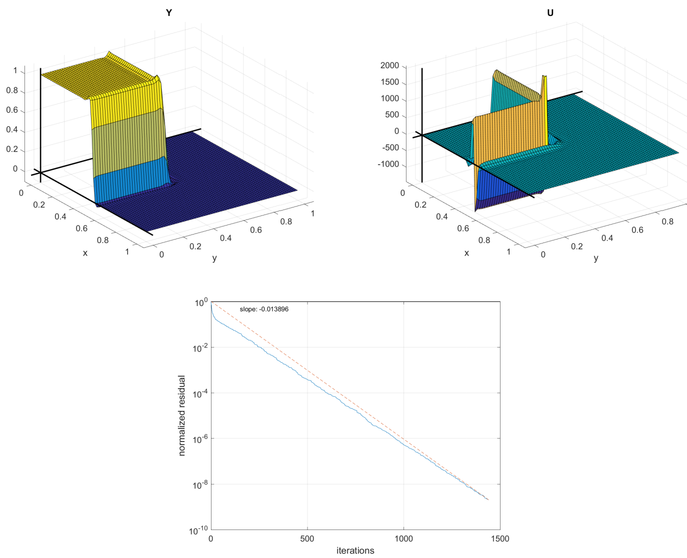


**Figure C.18:** Solutions to the setup in Section 4.3, state $y$ to the left, control $u$ to the right. This solution is for the case $\alpha = 10^{-8}$ and $h = 2^{-6}$.

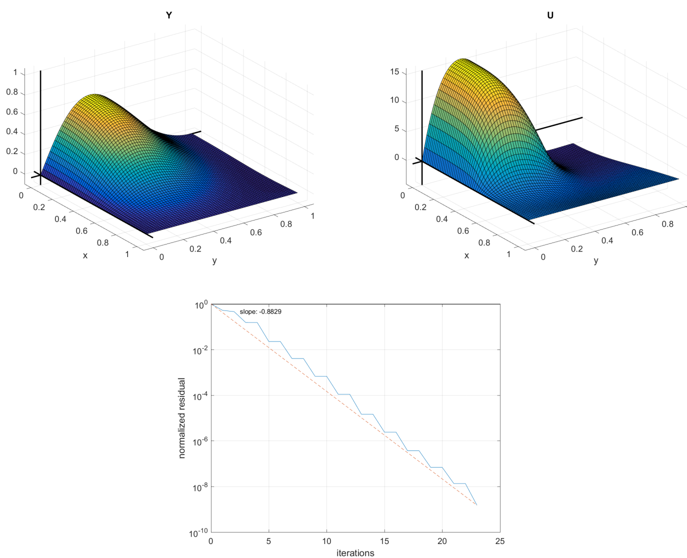


**Figure C.19:** Solutions to the setup in Section 4.4, state $y$ to the left, control $u$ to the right. This solution is for the case $\alpha = 10^{-3}$ and $h = 2^{-6}$.

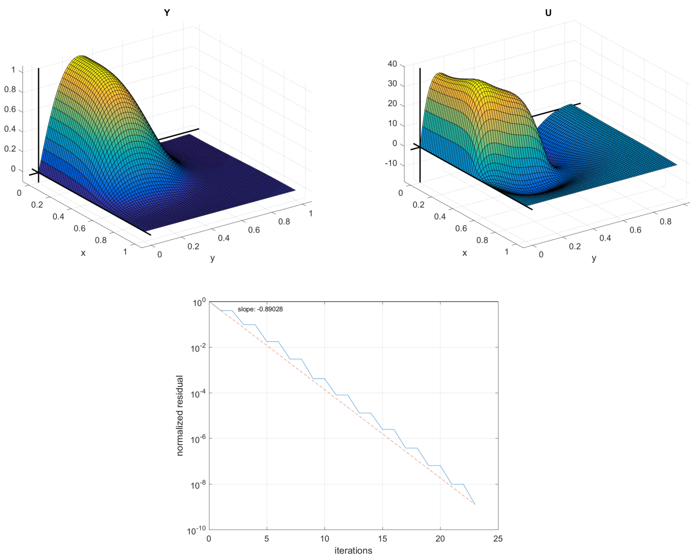


**Figure C.20:** Solutions to the setup in Section 4.4, state $y$ to the left, control $u$ to the right. This solution is for the case $\alpha = 10^{-4}$ and $h = 2^{-6}$.

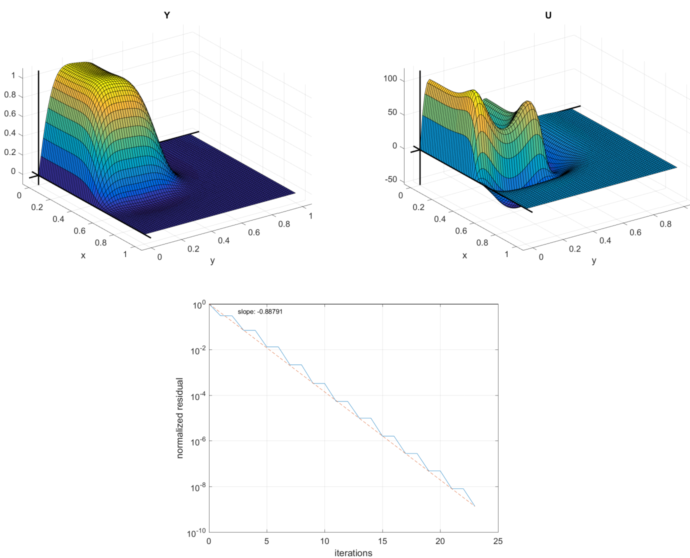


**Figure C.21:** Solutions to the setup in Section 4.4, state $y$ to the left, control $u$ to the right. This solution is for the case $\alpha = 10^{-5}$ and $h = 2^{-6}$.

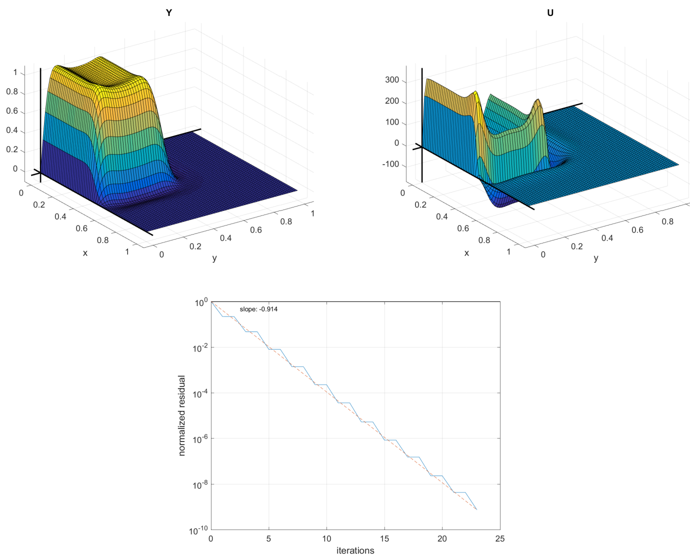


**Figure C.22:** Solutions to the setup in Section 4.4, state $y$ to the left, control $u$ to the right. This solution is for the case $\alpha = 10^{-6}$ and $h = 2^{-6}$.

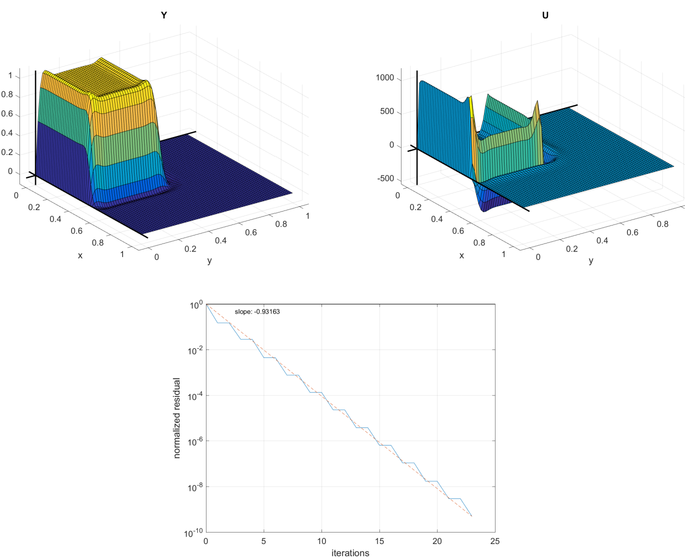


**Figure C.23:** Solutions to the setup in Section 4.4, state $y$ to the left, control $u$ to the right. This solution is for the case $\alpha = 10^{-7}$ and $h = 2^{-6}$.

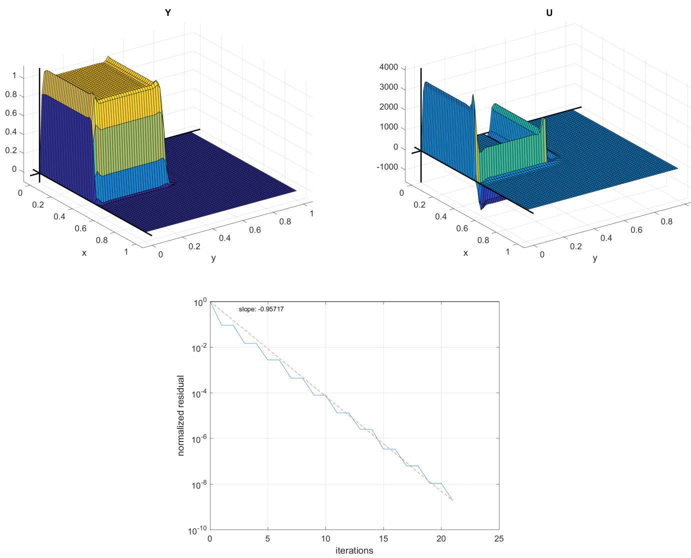


**Figure C.24:** Solutions to the setup in Section 4.4, state $y$ to the left, control $u$ to the right. This solution is for the case $\alpha = 10^{-8}$ and $h = 2^{-6}$.

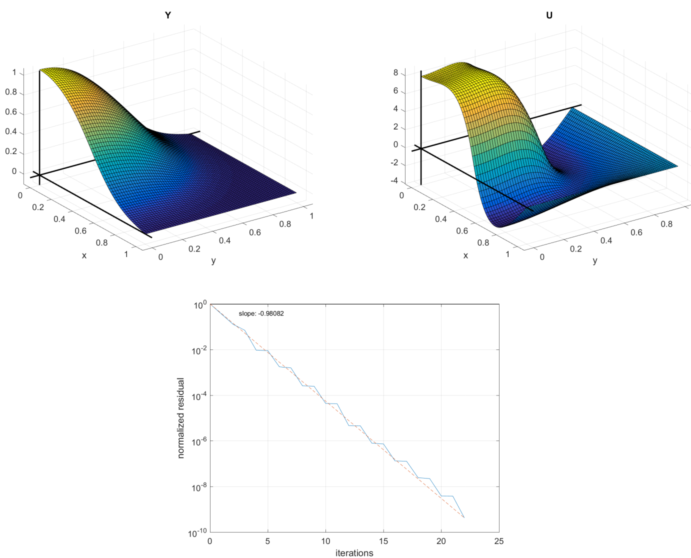


**Figure C.25:** Solutions to the setup in Section 4.5, state $y$ to the left, control $u$ to the right. This solution is for the case $\alpha = 10^{-3}$ and $h = 2^{-6}$.

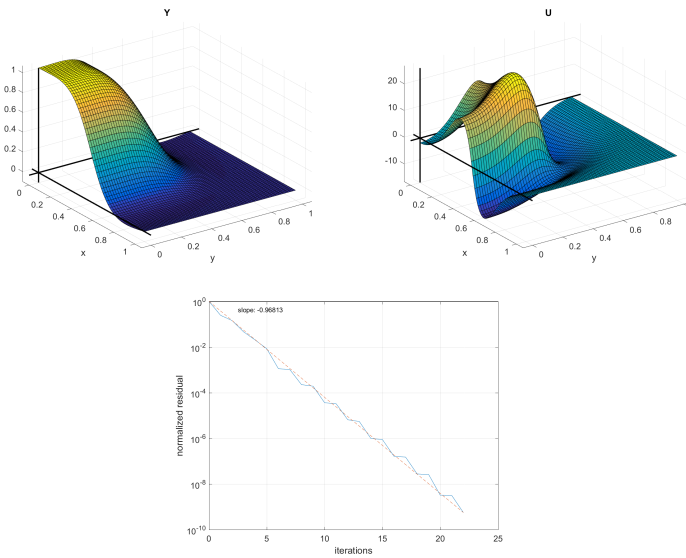


**Figure C.26:** Solutions to the setup in Section 4.5, state $y$ to the left, control $u$ to the right. This solution is for the case $\alpha = 10^{-4}$ and $h = 2^{-6}$.

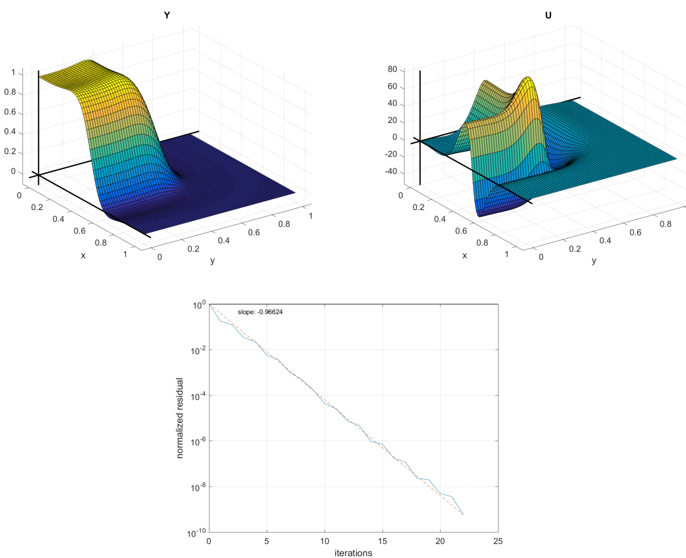


**Figure C.27:** Solutions to the setup in Section 4.5, state $y$ to the left, control $u$ to the right. This solution is for the case $\alpha = 10^{-5}$ and $h = 2^{-6}$.

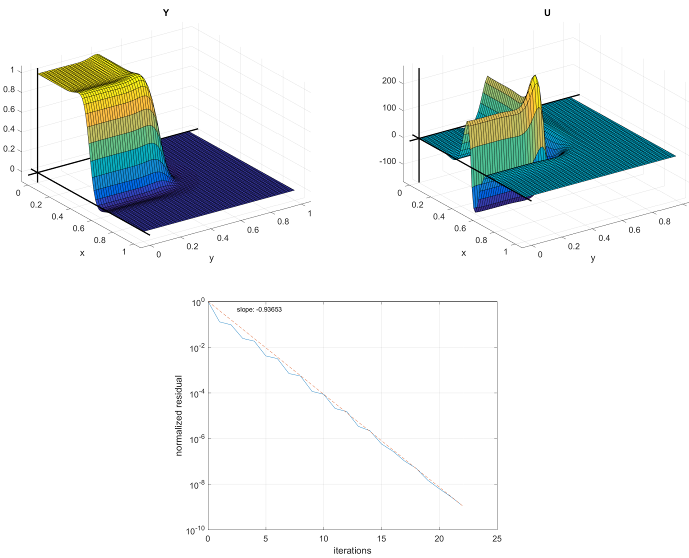


**Figure C.28:** Solutions to the setup in Section 4.5, state $y$ to the left, control $u$ to the right. This solution is for the case $\alpha = 10^{-6}$ and $h = 2^{-6}$.

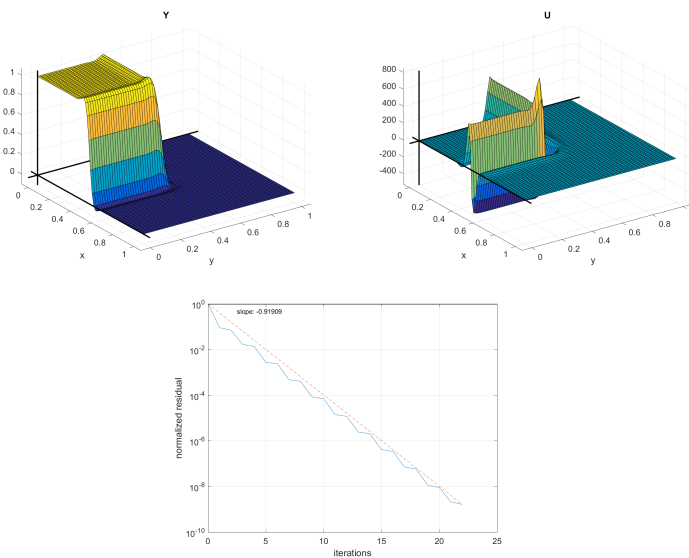


**Figure C.29:** Solutions to the setup in Section 4.5, state $y$ to the left, control $u$ to the right. This solution is for the case $\alpha = 10^{-7}$ and $h = 2^{-6}$.

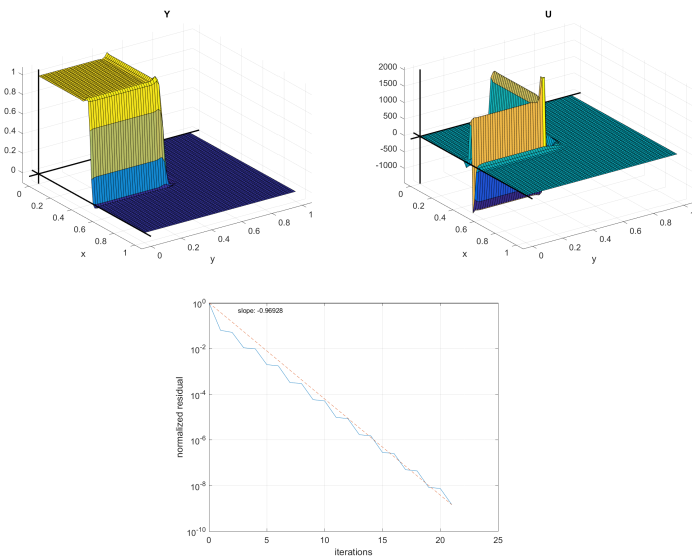


**Figure C.30:** Solutions to the setup in Section 4.5, state $y$ to the left, control $u$ to the right. This solution is for the case $\alpha = 10^{-8}$ and $h = 2^{-6}$.

# APPENDIX D

# G-bar framework

As some Python libraries such as FEniCS are not available on the Windows platform, it was necessariry to execute jobs on the university servers, using the General Databar (G-bar) service provided at DTU.

Jobs can be executed on the server directly using SSH or by connecting using Thinlinc. However, while Thinlinc provides a desktop environment it is not as smooth as using a personal computer and one might not be used to the development tools provided on the servers. Working from ones personal computer is often prefered.

Jobs can also be sent to the server cluster using the command `qsub` command. The simple syntax is `qsub <bash script file>`, however, many parameters can be specified for the command for more specialized usage. Sending jobs still requires a connection to the server, e.g. using SSH. Using `qstat` gives a list of current jobs.

Furthermore, either approach still requires all the relevant files for the job to be present on the server. This might be logical, but the logistics of moving relevant job files and results back and forth between working on and updating them are cumbersome.

In order to circumvent this hurdle we wrote a Python package for interfacing with the DTU G-bar directly in Python. This appendix covers the usage of this framework.

## D.1 gbar.py

The framework is a number of objects handling interaction with the G-bar using SSH and FTP via the existing `paramiko` package available in Python. The following objects are defined in `gbar.py`:

**Connection** is the "root"-object for the connections. It facilitates setting a hostname, a username and a password. It does not actually set up any connection to the host.

**SSH** derives from Connection. It sets up an SSH client for connection to the host address and has functions for sending commands to the host after a connection. The results can be either displayed to the terminal or captured as strings.

**FTP** derives from Connection. It sets up an FTP client for connection to the host address and has methods for exploring directories and checking for existence of files and directories on the server. It also sports functionality to upload, download and delete files, as well as open files for reading.

**gbar** derives from SSH. It sets the host name to `login.gbar.dtu.dk` and connects using SSH. Internally it also sets up and connects an FTP object to the host for handling file manipulation. There are methods for calling `qsub` and qstat.

**hpc** derives from gbar. Sets the host to `login.hpc.dtu.dk`, but is otherwise the same as gbar.

**job** is "root"-object for jobs. It allows several different settings and takes lists of files relevant to the job, i.e. the main files to be executed, the dependecies it might have and the relevant output files. The object takes a gbar-connection object and uses it to move all relevant files to the server. It generates the relevant bash script file and moves it to the server as well. It runs the job on the server and has methods for checking if the job has been completed yet. It also downloads the relevant result files after the job has run and removes everything from the server again unless asked not to.

**FEniCSjob** derives from job. It makes sure the bash script loads the FEniCS modules on the server before executing the job.

The idea is that from this foundation is is fairly easy to set up your own specialized connection and job. The following is the Python code for the package.

```
1  import paramiko
2  import getpass
3  import time
4  import os
5  import ntpath
6  
7  class Connection(object):
8      def __init__(self, host, username=None, password=None):
9          self.host = host
10         self.username = username
11         self.password = password
```

```
12
13      def get_username(self):
14          if not self.username:
15              return raw_input('Username: ')
16          return self.username
17
18      def get_password(self):
19          if not self.password:
20              return getpass.getpass('Password: ')
21          return self.password
22
23  class SSH(Connection):
24      def __init__(self, host, username=None, password=None):
25          super(SSH, self).__init__(host, username, password)
26
27          self.ssh = paramiko.SSHClient()
28          self.ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
29
30      def connect(self):
31          username = self.get_username()
32          password = self.get_password()
33          self.ssh.connect(self.host, username=username, password=password)
34
35      def close(self):
36          self.ssh.close()
37
38      def iexec_command(self, cmd, getErr=False):
39          stdin, stdout, stderr = self.ssh.exec_command(cmd)
40          if getErr:
41              for line in stderr.read().splitlines():
42                  print " | %s" % line
43          else:
44              for line in stdout.read().splitlines():
45                  print " | %s" % line
46
47      def exec_command(self, cmd, NoPrint=False):
48          stdin, stdout, stderr = self.ssh.exec_command(cmd)
49          string = stdout.read()
50          if NoPrint:
51              return string
52          else:
53              print(string)
54
55
56  class FTP(Connection):
57      def __init__(self, host, username=None, password=None):
```

```
58          super(FTP, self).__init__(host, username, password)
59          self.port = 22
60
61          self.transport = paramiko.Transport((self.host, self.port))
62          self.sftp = None
63          self.path = "./"
64
65      def connect(self):
66          # Connect to remote host
67          username = self.get_username()
68          password = self.get_password()
69          self.transport.connect(username=username, password=password)
70          self.sftp = paramiko.SFTPClient.from_transport(self.transport)
71
72      def close(self):
73          self.sftp.close()
74
75      def set_path(self, path):
76          # Sets working directory
77          # Path can be absolute or relative to homedir
78          if not path[-1] == "/":
79              raise Exception
80          if not self.exist(path):
81              raise Exception
82          self.path = path
83
84      def is_absolute_path(self, path):
85          if path[0] == "/" or path[0:2] == "./" or path[0:2] == "~/":
86              return True
87          else:
88              return False
89
90      def listdir(self, path, hidden=False):
91          # Write out the content of the path
92          if not self.is_absolute_path(path):
93              path = self.path + path
94          list = sorted([e for e in map(str, self.sftp.listdir(path)) if e[0]
                <> "." or hidden])
95          for line in list:
96              print " | %s" % line
97
98      def exist(self, path):
99          # Check if path exists on the remote host
100         if not self.is_absolute_path(path):
101             path = self.path + path
102         try:
```

```
103                 self.sftp.stat(path)
104             except IOError, e:
105                 if e[0] == 2:
106                     return False
107                 raise
108             else:
109                 return True
110 
111     def download(self, remotepath, localpath):
112         # Download file
113         if not self.is_absolute_path(remotepath):
114             remotepath = self.path + remotepath
115         self.sftp.get(remotepath, localpath)
116 
117     def upload(self, localpath, remotepath):
118         # Upload file
119         if not self.is_absolute_path(remotepath):
120             remotepath = self.path + remotepath
121         self.sftp.put(localpath, remotepath)
122 
123     def open(self, path, mode="r"):
124         # Open file to read
125         if not self.is_absolute_path(path):
126             path = self.path + path
127         return self.sftp.open(path, mode)
128 
129     def delete(self, path):
130         # Delete remote file
131         if not self.is_absolute_path(path):
132             path = self.path + path
133         self.sftp.remove(path)
134 
135 class gbar(SSH):
136     def __init__(self, username=None, password=None):
137         super(gbar, self).__init__('login.gbar.dtu.dk', username, password)
138         self.path = None
139         self.output_file = None
140         self.error_file = None
141         self.ftp = FTP('transfer.gbar.dtu.dk', username, password)
142 
143         self.connected = False
144 
145     def connect(self):
146         super(gbar, self).connect()
147         self.ftp.connect()
148         self.connected = True
```

```
149
150     def close(self):
151         super(gbar, self).close()
152         self.ftp.close()
153         self.connected = False
154
155     def is_connected(self):
156         return self.connected
157
158     # Allows for usage such as:
159     # g = gbar(...)
160     # g("ls -l")
161     # g("pwd")
162     def __call__(self, command, NoPrint=False):
163         assert isinstance(command, str)
164         return self.exec_command(command, NoPrint)
165
166     # Allows for usage such as:
167     # g = gbar(...)
168     # g > "ls -l"
169     # g > "pwd"
170     def __gt__(self, command):
171         assert isinstance(command, str)
172         return self.exec_command(command, False)
173
174     # Defining Job
175     def set_path(self, path):
176         # Sets working directory
177         # Path can be absolute or relative to homedir
178         if not self.is_connected():
179             raise Exception
180         if not path[-1] == "/":
181             raise Exception
182         if not self.ftp.exist(path):
183             raise Exception
184         self.path = path
185         self.ftp.path = path
186
187     def exec_command(self, command, NoPrint=False):
188         if not self.is_connected():
189             raise Exception
190         command = self.path_cmd() + command
191         return super(gbar, self).exec_command(command, NoPrint)
192
193     def path_cmd(self):
194         if self.path:
```

```
195                return "cd " + self.path + ";"
196            else:
197                return ""
198
199        def qsub_cmd(self, args):
200            qsub_location = "/opt/torque4/bin/qsub"
201            return qsub_location + " " + args + ";"
202
203        def qstat(self, args=None, job_name=None, output=False):
204            qstat = "/opt/torque4/bin/qstat"
205            command = qstat
206            if args:
207                command = command + " " + args
208            x = self.exec_command(command, True)
209            if not x:
210                return ""
211            x = x.split("\n")
212            l = []
213            for i in range(2,len(x)):
214                y = filter(None, x[i].split(" "))
215                if not len(y) > 0:
216                    break
217                if job_name and not (job_name == y[1]):
218                    continue
219                l.append(y)
220            if output:
221                return l
222            for i in range(0,len(l)):
223                print("{0:<15} | {1:<20} | {2:<10} | {3:<10} | {4:^3} | {5:<8}".
                        format(*l[i]))
224
225    class hpc(gbar):
226        def __init__(self, username=None, password=None):
227            super(hpc, self).__init__(username, password)
228            self.host = 'login.hpc.dtu.dk'
229
230    class job(object):
231        def __init__(self):
232            self.job_id = None
233
234            # Input / Output
235            self.main = None
236            self.dependencies = None
237            self.output = None
238
239            # Job parameters
```

```
240         self.walltime = None
241         self.localpath = None
242         self.remotepath = None
243         self.stdout_file = None
244         self.stderr_file = None
245 
246         self.conn = None
247 
248         self.started = False
249         self.completed = False
250         self.silent = False
251         self.clean_up = True
252 
253     def set_job_id(self):
254         self.job_id = self.generate_job_id()
255 
256     def set_main(self, script):
257         self.main = script
258 
259     def set_dependencies(self, scripts):
260         self.dependencies = scripts
261 
262     def set_output(self, output):
263         self.output = output
264 
265     def set_walltime(self, walltime):
266         self.walltime = walltime
267 
268     def set_silence(self, silence):
269         self.silent = silence
270 
271     def set_self_cleaning(self, clean):
272         self.clean_up = clean
273 
274     def set_localpath(self, localpath):
275         self.localpath = localpath
276 
277     def set_remotepath(self, remotepath):
278         self.remotepath = remotepath
279 
280     def set_path(self, localpath, remotepath):
281         self.set_localpath(localpath)
282         self.set_remotepath(remotepath)
283 
284     def set_stdout_file(self, stdout_file):
285         if stdout_file[0] == "/" or stdout_file[0] == ".":
```

```
286             cpath = stdout_file
287         else:
288             assert isinstance(self.remotepath, str)
289             cpath = self.remotepath + stdout_file
290 
291         path_dir = "/".join(cpath.split("/")[0:-1])
292         assert isinstance(self.conn, gbar)
293         assert self.conn.ftp.exist(path_dir)
294 
295         self.stdout_file = stdout_file
296 
297     def set_stderr_file(self, stderr_file):
298         if stderr_file[0] == "/" or stderr_file[0] == ".":
299             cpath = stderr_file
300         else:
301             assert isinstance(self.remotepath, str)
302             cpath = self.remotepath + stderr_file
303 
304         path_dir = "/".join(cpath.split("/")[0:-1])
305         assert isinstance(self.conn, gbar)
306         assert self.conn.ftp.exist(path_dir)
307 
308         self.stderr_file = stderr_file
309 
310     def set_stdout_stderr_files(self, stdout_file, stderr_file):
311         self.set_stdout_file(stdout_file)
312         self.set_stderr_file(stderr_file)
313 
314     def set_connection(self, connection):
315         self.conn = connection
316 
317     def get_filepart(self, file, part = None):
318         if part == "file":
319             return ntpath.basename(file)
320         elif part == "path":
321             return ntpath.dirname(file) + "/"
322         else:
323             return file
324 
325     def validate(self):
326         if not isinstance(self.main, str):
327             raise Exception
328         if not isinstance(self.dependencies, list):
329             if isinstance(self.dependencies, str):
330                 self.dependencies = [self.dependencies]
331             else:
```

```
332                     raise Exception
333             if not isinstance(self.output, list):
334                 if isinstance(self.output, str):
335                     self.output = [self.output]
336                 else:
337                     raise Exception
338             if not isinstance(self.localpath, str):
339                 raise Exception
340             if not isinstance(self.remotepath, str):
341                 raise Exception
342             if not isinstance(self.stdout_file, str):
343                 raise Exception
344             if not isinstance(self.stderr_file, str):
345                 raise Exception
346             if not isinstance(self.walltime, str):
347                 raise Exception
348             if not isinstance(self.conn, gbar):
349                 raise Exception
350             if not self.job_id:
351                 self.set_job_id()
352
353         def generate_job_id(self):
354             return "job01" + ("%13.2f" % time.time()).replace(".", "")
355
356         def id2time(self, job_id):
357             s = job_id[3:-2] + "." + job_id[-2:]
358             return float(s)
359
360         def walltime2sec(self):
361             l = self.walltime.split(":")
362             t = 0
363             mul = [1,60,3600]
364             for i in range(0,len(l)):
365                 t += int(l.pop())*mul[i]
366             return t
367
368         def generate_bash(self, commands):
369             # Check that path is set
370             self.validate()
371
372             script_name = self.job_id + ".sh"
373
374             # File content
375             initialization = ["#!/bin/sh",
376                               "",
377                               "#PBS -N " + self.job_id,
```

```
378                             "#PBS -l walltime=" + self.walltime,
379                             "#PBS -o " + self.stdout_file,
380                             "#PBS -e " + self.stderr_file,
381                             "cd $PBS_O_WORKDIR"]
382         header = ["",
383                   "echo",
384                   "echo ================================",
385                   "echo Running job: " + self.job_id,
386                   "echo ================================",
387                   "echo Execution time: `date`"]
388 
389         clean_up = ["rm " + script_name]
390         end_statement = ["",
391                          "echo ================================",
392                          "echo End time: `date`",
393                          "echo End: " + self.job_id]
394 
395         # Write file
396         with open(script_name, "wb") as file:
397             file.write("\n".join(initialization))
398             file.write("\n")
399             file.write("\n".join(header))
400             file.write("\n")
401             file.write("\n".join(commands))
402             file.write("\n")
403             if self.clean_up:
404                 file.write("\n".join(clean_up))
405                 file.write("\n")
406             file.write("\n".join(end_statement))
407 
408         # Upload file
409         self.conn.ftp.upload(script_name, self.remotepath + script_name)
410         os.remove(script_name)
411 
412     def is_done(self):
413         if self.completed:
414             return True
415         if not self.started:
416             return False
417 
418         if not self.conn.ftp.exist(self.stdout_file):
419             return
420 
421         f = self.conn.ftp.open(self.stdout_file)
422         lines = f.readlines()
423         if len(lines) == 0:
```

```
424                 return False
425             s = str(lines[-1])
426             f.close()
427             if s[0:4] <> "End:":
428                 return False
429             s = s[5:-1]
430             t1 = self.id2time(s)
431             t2 = self.id2time(self.job_id)
432             if t1 < t2:
433                 return False
434             self.completed = True
435             return True
436 
437         def wait_for(self):
438             if not self.started:
439                 raise Exception
440             queue_time = 300  # worst case guess(?)
441             wait_times = [3, 3, 4, 10, 10, 30, 30, 30, 60, 60, 120, 300]
442             wait_times.reverse()
443             job_done = True
444             t = 0
445             while not self.is_done():
446                 if len(wait_times) > 1:
447                     w = wait_times.pop()
448                 if not self.silent:
449                     print "(%d sec) Waiting %d seconds more... " % (t, w)
450                 time.sleep(w)
451                 t = t + w
452                 if t > queue_time + self.walltime2sec():
453                     job_done = False
454                     break
455             if job_done:
456                 print "job01 is done!"
457             else:
458                 print "job01 is not done yet, consider checking 'qstat' through
                        putty"
459 
460         def generate_command(self):
461             return ["python " + self.get_filepart(self.main, "file")]
462 
463         def run(self):
464             self.validate()
465             self.started = True
466 
467             # Upload
468             self.conn.ftp.upload(self.localpath + self.main, self.get_filepart(
```

```
                self.main, "file"))
469         for dependency in self.dependencies:
470             self.conn.ftp.upload(self.localpath + dependency, self.
                    get_filepart(dependency, "file"))
471
472         # Run
473         command = self.generate_command()
474
475         self.generate_bash(command)
476         bash_script = self.job_id + ".sh"
477
478         command = self.conn.path_cmd() + self.conn.qsub_cmd(bash_script)
479         if not self.silent:
480             print "Executing command: \n %s" % command
481         self.conn.ssh.exec_command(command)
482
483     def get_data(self):
484         if not self.completed:
485             raise Exception
486         for i in range(0, len(self.output)):
487             self.conn.ftp.download(self.output[i], self.localpath + self.
                    output[i])
488
489     def get_stdout_file(self, location=None):
490         if location:
491             self.conn.ftp.download(self.stdout_file, location)
492         else:
493             f = self.conn.ftp.open(self.stdout_file)
494             l = f.readlines()
495             for s in l:
496                 print(s.strip())
497
498     def get_stderr_file(self, location=None):
499         if location:
500             self.conn.ftp.download(self.stderr_file, location)
501         else:
502             f = self.conn.ftp.open(self.stderr_file)
503             l = f.readlines()
504             for s in l:
505                 print(s.strip())
506
507
508     def clean(self):
509         self.conn.ftp.delete(self.get_filepart(self.main, "file"))
510         for dependency in self.dependencies:
511             self.conn.ftp.delete(self.get_filepart(dependency, "file"))
```

```
512             self.conn.ftp.delete(self.get_filepart(dependency, "file") + "c"
                    )
513         for i in range(0, len(self.output)):
514             self.conn.ftp.delete(self.output[i])
515 
516     def run_wait_and_get_data(self):
517         self.run()
518         self.wait_for()
519         self.get_data()
520         if self.clean_up:
521             self.clean()
522 
523 class FEniCSjob(job):
524     def __init__(self):
525         super(FEniCSjob, self).__init__()
526 
527     def load_FEniCS_cmd(self):
528         cmd = ["module load gcc",
529                "module load FEniCS/1.5.0",
530                "module load openblas"]
531         return cmd
532 
533     def generate_command(self):
534         fenics_cmd = self.load_FEniCS_cmd()
535         python_cmd = ["python " + self.get_filepart(self.main, "file")]
536         return fenics_cmd + python_cmd
```

.

## D.2 Personalized connection and job

As an example we will show here how our connection and job was set up more specifically. `bcsj_gbar` derived from `gbar` is set to automatically connect using the username and password directly given in the object. Futhermore it directs the object to work out of a specified folder on the server. A bit of printet output to the terminal was added for feedback.

```
1 class bcsj_gbar(gbar):
2     def __init__(self, usr="<studentId>", pwd="<password>"):
3         print("--------------------------")
4         print(" Logging in to Gbar...")
5         print("--------------------------")
6         print(" > Username: " + usr)
```

```
        print(" > Password: ********")
        print("--------------------------")
        super(bcsj_gbar, self).__init__(usr, pwd)

        self.connect()
        print(" ")
        if not self.ftp.exist("./ssh_python/"):
            self.ftp.sftp.mkdir("./ssh_python/")

        self.set_path("./ssh_python/")

    def connect(self):
        super(bcsj_gbar, self).connect()
        print(" Connected...")

    def close(self):
        super(bcsj_gbar, self).close()
        print(" Connection closed...")
```

.

As seen in the results a lot of otherwise identical jobs were run using different values of $\alpha$ and $h$. In order to batch these jobs together as one an extension to the FEniCSjob object was made. This added a new property `args` to the job-object listing a number of different arguments the job should be executed with. To handle this argument the method for generating part of the bash script needed to be overloaded.

```
class bcsj_job(FEniCSjob):
    def __init__(self, connection=None):
        super(bcsj_job, self).__init__()

        self.set_connection(connection)

        self.set_remotepath("./ssh_python/")
        self.set_stdout_stderr_files(
            stdout_file="output/stdout.txt",
            stderr_file="output/stderr.txt"
        )
        self.args = None

    def set_args(self, args):
        self.args = args

    def generate_command(self):
        fenics_cmd = self.load_FEniCS_cmd()
        if self.args:
```

```
20            if isinstance(self.args, list):
21                python_cmd = []
22                i = 0
23                for arg in self.args:
24                    i += 1
25                    python_cmd += ["python " + self.get_filepart(self.main,
                          "file") + " " + arg]
26                    python_cmd += ["echo subjob {0} / {1} done!".format(i,
                          len(self.args))]
27            else:
28                python_cmd = ["python " + self.get_filepart(self.main, "file
                      ") + " " + self.args]
29        else:
30            python_cmd = ["python " + self.get_filepart(self.main, "file")]
31        return fenics_cmd + python_cmd
```

.

# Bibliography


[BGL05] Michele Benzi, Gene H. Golub, and Jörg Liesen. “Numerical solution of saddle point problems”. In: *ACTA NUMERICA* 14 (2005), pages 1–137.

[Bis06] Christopher M Bishop. “Pattern Recognition”. In: *Machine Learning* (2006).

[Bun88] John W Bunce. “Inertia and controllability in infinite dimensions”. In: *Journal of mathematical analysis and applications* 129.2 (1988), pages 569–580.

[Cai80] Bryan E Cain. “Inertia theory”. In: *Linear Algebra and its Applications* 30 (1980), pages 211–240.

[CG15] Mauricio Carrillo and Juan Alfredo Gómez. “A Globally Convergent Algorithm for a PDE-Constrained Optimization Problem Arising in Electrical Impedance Tomography”. In: *Numerical Functional Analysis and Optimization* just-accepted (2015).

[De 15] Juan Carlos De los Reyes. *Numerical PDE-constrained optimization*. Springer, 2015.

[DTU16] DTU. *Study database: MSc in Mathematical Modelling and Computation*. 2016. URL: http://sdb.dtu.dk/2015/20/378#Master_thesis.

[Eng09] Allan Peter Engsig-Karup. *The Spectral/hp-Finite Element Method for Partial Differential Equations*. 2009.

[Eva08] Lawrence C. Evans. *Partial Differential Equations*. Volume 19. American Mathematical Society, 2008.

[GHS14] Andreas Günnel, Roland Herzog, and Ekkehard Sachs. “A note on preconditioners and scalar products in Krylov subspace methods for self-adjoint problems in Hilbert space”. In: *Electronic Transactions on Numerical Analysis* 41 (2014), pages 13–20.

[Gru08] Gerd Grubb. *Distributions and operators*. Volume 252. Springer Science & Business Media, 2008.

[Her10] R. Herzog. *Lectures Notes Algorithms and Preconditioning in PDE-Constrained Optimization*. 2010. URL: `https://www.tu-chemnitz.de/mathematik/part_dgl/talks/Lecture_Notes_Algorithms_and_Preconditioning.pdf`.

[Kre89] Erwin Kreyszig. *Introductory functional analysis with applications*. Volume 81. wiley New York, 1989.

[Pea13] John W Pearson. “Fast iterative solvers for PDE-constrained optimization problems”. PhD thesis. University of Oxford, 2013.

[RDW10] Tyrone Rees, H Sue Dollar, and Andrew J Wathen. “Optimal solvers for PDE-constrained optimization”. In: *SIAM Journal on Scientific Computing* 32.1 (2010), pages 271–298.

[Yos80] K Yosida. *Functional analysis*. 1980.

[Zei95] Eberhard Zeidler. *Applied Fuctional Analysis: Main Principles and Their Applications*. Volume 109. 1995.

[Zha05] Fuzhen Zhang. *The Schur Complement and Its Applications*. Volume 4. Springer Science & Business Media, 2005.

[Zul11] Walter Zulehner. “Nonstandard norms and robust estimates for saddle point problems”. In: *SIAM Journal on Matrix Analysis and Applications* 32.2 (2011), pages 536–560.