Lectures Notes
Algorithms and Preconditioning in
PDE-Constrained Optimization

Prof. Dr. R. Herzog

held in July 2010
at the Summer School on
Analysis and Numerics
of PDE Constrained Optimization,
Lambrecht

CHEMNITZ UNIVERSITY OF TECHNOLOGY
Contents

Chapter 1. Algorithms in PDE-Constrained Optimization 5
  1  A Taxonomy of Methods 5
  2  Methods for Unconstrained Problems 7
    2.1  Black-Box Methods 8
    2.2  All-at-once Methods 12
  3  Treatment of Inequality Constraints 16
    3.1  Control Constraints: Primal-Dual Active Set Strategy 16
    3.2  Mixed Control-State Constraints 21
    3.3  State Constraints 22

Chapter 2. Preconditioning in PDE-Constrained Optimization 25
  4  Introduction 25
  5  Properties of Saddle Point Problems 26
    5.1  Saddle Point Problems Arising in Optimization 27
    5.2  Hilbert Space Setting 28
    5.3  Spectral Properties 29
    5.4  An Optimal Control Example 30
  6  Preconditioning KKT Systems 33
    6.1  Early Approaches 34
    6.2  Constraint Preconditioners 35
    6.3  Preconditioned Conjugate Gradients 35
    6.4  Implementation Details 38

Appendix A. Software 45
  7  Source Code with Comments 45

Bibliography 51
CHAPTER 1

Algorithms in PDE-Constrained Optimization

Contents

1 A Taxonomy of Methods 5
2 Methods for Unconstrained Problems 7
  2.1 Black-Box Methods 8
  2.1.1 Steepest Descent Method 8
  2.1.2 Nonlinear Conjugate Gradient Methods 10
  2.1.3 Newton’s Method 11
  2.2 All-at-once Methods 12
  2.2.1 First-Order Augmented Lagrangian Methods 12
  2.2.2 SQP Methods 14
  2.2.3 Augmented-Lagrangian SQP Methods 15
3 Treatment of Inequality Constraints 16
  3.1 Control Constraints: Primal-Dual Active Set Strategy 16
  3.1.1 Primal-Dual Active Set Strategy as an Outer Iteration 17
  3.1.2 Primal-Dual Active Set Strategy within Newton and SQP Iterations 20
  3.1.3 Primal-Dual Active Set Strategy as a Semismooth Newton Iteration 21
  3.2 Mixed Control-State Constraints 21
  3.3 State Constraints 22

§ 1 A Taxonomy of Methods

Let us consider mathematical optimization problems which may involve equality and inequality constraints:

\[
\begin{align*}
\text{Minimize} & \quad f(x) \\
\text{s.t. (subject to)} & \quad e(x) = 0 \\
& \quad g(x) \leq 0.
\end{align*}
\]

We assume throughout that \( f, e, g \) are sufficiently smooth functions. The optimization variable \( x \) can be an element from some finite or infinite dimensional space. In the latter case, and when \( e(x) = 0 \) involves a partial differential equation (PDE), we call (1.1) a PDE-constrained optimization problem.

Examples for this class of problems are:

- optimal control problems,
- parameter identification problems,
- shape optimization problems.
Chapter 1. Algorithms in PDE-Constrained Optimization

It is a particular feature in PDE-constrained optimization that the variable $x$ can be partitioned into $x = (y, u)$. This partitioning is induced by the PDE equality constraint. For well-posed problems, the state variable $y$ can be uniquely (or at least locally uniquely) determined from the PDE $e(y, u) = 0$, for any given $u$. In this context, $u$ is called the control variable or more generally, design variable. For the examples of problem classes above, $u$ is either the control variable, the parameter to be identified, or the shape variable.

This observation implies that in PDE-constrained optimization, we have a choice:

(a) whether we keep the constraint $e(y, u) = 0$ as a side constraint in our optimization problem and treat both $(y, u)$ as optimization variables,

(b) or whether we eliminate the PDE constraint by means of a solution operator (control-to-state map) $y = S(u)$ (which solves $e(S(u), u) = 0$ for us); then we replace $y$ by $S(u)$ and keep only the design variable $u$ as an optimization variable.

Note that this choice sets apart PDE-constrained optimization problems from generic nonlinear optimization problems (1.1), for which it is often not evident how to eliminate the equality constraints by solving for some of the variables. However, the choice whether to eliminate or keep the PDE constrains may not always be ours to make. For instance, we might be asked to build an optimization loop around an existing solver $y = S(u)$ for a particular equation.

Algorithms for solving PDE-constrained optimization problems can be classified along a number of different dimensions. Here is an attempt at naming at least some of these dimensions.

(a) How is the PDE constraint treated?

<table>
<thead>
<tr>
<th>Black-box methods</th>
<th>All-at-once methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>The PDE constraint is eliminated using a solver $y = S(u)$</td>
<td>The PDE constraint is kept explicitly as a side constraint. Both the state and control variables $x = (y, u)$ are optimization variables.</td>
</tr>
</tbody>
</table>

(b) What is the highest order of derivatives (or approximations thereof) used by the algorithm?

<table>
<thead>
<tr>
<th>Gradient-based methods</th>
<th>Hessian-based methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use only first-order derivatives of $f$, $e$ and $h$.</td>
<td>Use second-order derivatives of at least one of $f$, $e$ and $h$.</td>
</tr>
</tbody>
</table>

(c) What is the typical local rate of convergence of the method?

| The method may typically exhibit a q-linear rate of convergence. | Or it may converge at least q-superlinearly. |

(d) How costly is each iteration?
§ 2. Methods for Unconstrained Problems

| No solutions of auxiliary optimization problems are needed. | Every iteration requires the solution of at least one auxiliary optimization problem. |

(e) Does the method produce iterates which are feasible w.r.t. some of the constraints?

(f) Does the method provide some mechanism to improve its global convergence properties?

(g) Will the method take advantage of the fact that some of the constraints may be simple or linear?

(h) Can the method be applied in function space, or can it be applied to discretized (finite dimensional) problems only?

Of course, these dimensions are not independent of each other. For instance, in order to achieve a faster-than-linear rate of convergence, a method will typically need to use more than just gradient information, and an auxiliary optimization problem (e.g., based on a second-order Taylor expansion of the original problem) has to be solved in each iteration.

§ 2 Methods for Unconstrained Problems

We begin by briefly reviewing methods for problems without inequality constraints, which we shall call ‘unconstrained’ problems:

\[
\begin{align*}
\text{Minimize} & \quad f(y, u) \quad \text{over} \quad (y, u) \in Y \times U \\
\text{s.t.} & \quad e(y, u) = 0.
\end{align*}
\]

(2.1)

It is assumed that we have at our disposal the control-to-state map \( y = S(u) \) which solves \( e(S(u), u) = 0 \) so that we may choose to eliminate the PDE constraint \( e(\cdot) \), which maps \( Y \times U \to Z' \) with Hilbert spaces \( Y, U, Z \).

A simple and often used example is the following optimal control problem.

\[
\begin{align*}
\text{Minimize} & \quad \frac{1}{2} \|y - y_\Omega\|_{L^2(\Omega)}^2 + \frac{\nu}{2} \|u\|_{L^2(\Omega)}^2 \\
\text{s.t.} & \quad (\nabla y, \nabla v)_{L^2(\Omega)} = (u, v)_{L^2(\Omega)} \quad \text{for all} \quad v \in H^1_0(\Omega)
\end{align*}
\]

(2.2)

with state space \( Y = H^1_0(\Omega) \) and control space \( U = L^2(\Omega) \). Here \( e : Y \times U \to Z' \) represents the weak form of Poisson’s equation,

\[
\langle e(y, u), v \rangle_{Z', Z} = (\nabla y, \nabla v)_{L^2(\Omega)} = (u, v)_{L^2(\Omega)} \quad \text{for all} \quad v \in H^1_0(\Omega).
\]

Figure 2.1 shows an overview of frequently used black-box (left column) and all-at-once methods (right column). We consider all of them except the SLP (sequential linear programming) in this section. Methods with typically exhibit a linear rate of convergence are shown in blue, while higher order methods are shown in red. We emphasize that we can give here only a brief overview of classes of methods and we omit many aspects which are important to make practical methods, like issues of inexactness and globalization, for instance.
§ 2.1 Black-Box Methods

Black-box methods treat the reduced problem

\[
\text{Minimize } \hat{f}(u) := f(S(u), u) \text{ over } u \in U. \tag{2.3}
\]

§ 2.1.1 Steepest Descent Method

Literature: [Kelley, 1999, Section 3.1]

The steepest descent method, or gradient method, uses the negative gradient as a search direction in every iteration. Note that we need to carefully distinguish between the gradient and the derivative of $\hat{f}$. By definition, the derivative of $\hat{f}$ at $u$, denoted by $\hat{f}'(u)$, is an element of $U'$, the dual space of $U$. We use the Riesz isomorphism $R : U' \to U$, to define the gradient $\nabla \hat{f}(u) = R\hat{f}'(u)$ as an element of $U$. This implies that

\[
\left\langle \hat{f}'(u), \delta u \right\rangle_{U', U} = \left\langle \nabla \hat{f}(u), \delta u \right\rangle_U,
\]

holds, where $(\cdot, \cdot)_U$ is the scalar product of $U$. The distinction between $\hat{f}'$ and $\nabla \hat{f}$ becomes important as soon as we leave $\mathbb{R}^n$ with the standard inner product behind (where $\hat{f}'(u)$ and $\nabla \hat{f}(u)$ are just transposes of each other). Note that the Riesz isomorphism and thus the definition of the gradient depend on the scalar product in $U$.

The efficient evaluation of the derivative (or the gradient) uses the adjoint technique. Note that by the chain rule,

\[
\hat{f}'(u) \delta u = f_y(S(u), u) S'(u) \delta u + f_u(S(u), u) \delta u
\]

holds, or in terms of gradients,

\[
\left\langle \nabla \hat{f}(u), \delta u \right\rangle_U = \left\langle \nabla_y f(\cdot), S'(u) \delta u \right\rangle_Y + \left\langle \nabla_u f(\cdot), \delta u \right\rangle_U.
\]

Also by the chain rule, we obtain

\[
e(S(u), u) = 0 \Rightarrow e_y(\cdot) S'(u) + e_u(\cdot) = 0 \Rightarrow S'(u) = -e_y(\cdot)^{-1} e_u(\cdot).
\]
Here and in the sequel, we abbreviate $(\mathcal{S}(u), u)$ by $(\cdot)$. This implies that

$$
\langle \hat{f}'(u), \delta u \rangle_{U', U} = \langle f_y(\cdot), -e_y(\cdot)^{-1}e_u(\cdot) \delta u \rangle_{Y', Y} + \langle f_u(\cdot), \delta u \rangle_{U', U}
$$

$$
= \langle -(e_y(\cdot)^{-1}f_y(\cdot)), e_u(\cdot) \delta u \rangle_{Z', Z} + \langle f_u(\cdot), \delta u \rangle_{U', U}
$$

$$
= \langle e_u(\cdot)^*p + f_u(\cdot), \delta u \rangle_{U', U},
$$

where the adjoint state $p \in Z$ is defined as the solution of

$$
e_y(y, u)^* p = -f_y(y, u)
$$

with $y = \mathcal{S}(u)$.

In practice the formula for the adjoint equation can be found conveniently using the (formal) Lagrangian calculus, see [Tröltzsch, 2010, Section 2.10]. For our example (2.2), the adjoint state is defined as the solution of

$$
(\nabla v, \nabla p)_{L^2(\Omega)} = -(y - y_0, v)_{L^2(\Omega)} \quad \text{for all } v \in H^1_0(\Omega).
$$

This happens to be the same PDE as the state equation, with a different right hand side, since the differential operator is self-adjoint in this example. The gradient is given by

$$
\nabla \hat{f}(u) = \nu u - p.
$$

Using the negative gradient of $\hat{f}$ at the current iterate $u_n$ as a search direction, we deduce the steepest descent method, Algorithm 1.

**Algorithm 1 Steepest descent method**

**Input:** $u_0 \in U$, $n_{\text{max}}$

**Output:**

1. Set $n := 0$ and done := false
2. while not done and $n < n_{\text{max}}$ do
3. Calculate $r_n := -\nabla \hat{f}(u_n)$
4. if convergence criterion satisfied then
5. Set done := true
6. else
7. Find an appropriate step length $t_n \approx \arg \min_{t>0} \hat{f}(u_n + tr_n)$
8. Set $u_{n+1} := u_n + t_n r_n$
9. Set $n := n + 1$
10. end if
11. end while

It is sufficient to determine the step length in step 7 by the Armijo rule with an initial trial step length of $t = 1$.

The steepest descent algorithm has the advantage of being easy to implement. It usually makes good progress during the first iterations but has poor convergence properties. It is therefore usually not recommended as a stand-alone method, but can be used as a fall-back to globalize more advanced methods.
§ 2.1.2 Nonlinear Conjugate Gradient Methods

Literature: [Nocedal and Wright, 2006, Section 5.2], [Kelley, 1999, Section 3.2.4], Volkwein [2004, 2003], Hager and Zhang [2006]

The (linear) conjugate gradient (CG) method is known as an iterative method with attractive convergence properties for the solution of linear systems $Ax = b$ of equations with symmetric and positive definite coefficient matrix $A$. Since solving this equation and finding the minimizer of $(1/2)x^T Ax - b^T x$ are the same, CG can be viewed as an optimization algorithm for strictly convex quadratic objective functions. Its nonlinear versions can be employed to find minimizers of unconstrained optimization problems with more general nonlinear objectives.

A nonlinear CG method is stated as Algorithm 2. In step 4, a search procedure is needed to determine an appropriate step length which minimizes $\varphi(t) = \hat{f}(u_k + td_k)$, or finds a zero of $\varphi'(t) = \left(\nabla \hat{f}(u_k + td_k), d_k\right)_U$. In principle, Newton’s method can be used for this purpose, but this requires the repeated evaluation of the Hessian of $\hat{f}$, which is prohibitive in the context of nonlinear CG methods. As a remedy, one may resort to a secant method in which $\varphi(t)$ is approximated by $\varphi(t) \approx \varphi(0) + \varphi'(0) t + \frac{\varphi'(\sigma) - \varphi'(0)}{2\sigma} t^2$ with some $\sigma > 0$. Minimization of the right hand side then leads to

$$t = -\frac{\sigma \varphi'(0)}{\varphi'(\sigma) - \varphi'(0)} = -\frac{\sigma \left(\nabla \hat{f}(u_k), d_k\right)_U}{\left(\nabla \hat{f}(u_k + \sigma d_k), d_k\right)_U - \left(\nabla \hat{f}(u_k), d_k\right)_U}.$$ (2.4)

Typically, only few iterations of (2.4) are carried out which generate a sequence of step lengths $t_i$ and linearization points $\sigma_{i+1} = -t_i$, starting from an arbitrary initial value $\sigma_0 > 0$. Such inexact line searches, however, may lead to search directions which are not descent directions for the objective $\hat{f}$. A common solution is then to restart the method by re-setting $d$ to the negative reduced gradient whenever $(r,d)_U \leq 0$ is found. This is the purpose of step 10 in Algorithm 2. An alternative step length selection strategy in step 4 based on Wolfe conditions is given in [Nocedal and Wright, 1999, eq. (5.42)].

**Algorithm 2** Nonlinear conjugate gradient algorithm

**Input:** $u_0 \in U$, $n_{\max}$

**Output:**

1. Set $n := 0$ and done := false
2. Evaluate $d_n := r_n := -\nabla \hat{f}(u_n)$
3. while not done and $n < n_{\max}$ do
4. Calculate a step length $t_n$ satisfying $\left(\nabla \hat{f}(u_n + t_n d_n), d_n\right)_U = 0$
5. Set $u_{n+1} := u_n + t_n d_n$
6. Set $r_{n+1} := -\nabla \hat{f}(u_{n+1})$
7. Determine step length $\beta_{n+1}$ by one of the formulas below
8. Set $d_{n+1} := r_{n+1} + \beta_{n+1} d_n$ and increase $n$
9. if $(r_n, d_n)_U \leq 0$ then
10. Set $d_n := r_n$
11. end if
12. end while
Several common choices exist for the selection of the step length $\beta_{k+1}$ in step 7. Among them are the Fletcher-Reeves and the Polak-Ribière formulas

$$
\beta_{k+1}^{\text{FR}} := \frac{(r_{k+1}, r_{k+1})}{(r_k, r_k)} U(r_k, r_k), \quad \beta_{k+1}^{\text{PR}} := \frac{(r_{k+1}, r_{k+1} - r_k)}{(r_k, r_k)} U(r_k, r_k), \quad \beta_{k+1}^{\text{PR}+} := \max\{\beta_{k+1}^{\text{PR}}, 0\}.
$$

We mention that nonlinear CG methods generally outperform the steepest descent method, and refer to [Nocedal and Wright, 1999, Section 5.2] and the survey article Hager and Zhang [2006] for a comparison of the various step length selection strategies. For an application of nonlinear conjugate gradient methods for the solution of optimal control problems, we refer to, e.g., Volkwein [2004, 2003].

§ 2.1.3 Newton’s Method

**Literature:** [Ito and Kunisch, 2008, Chapter 5.2], [Kelley, 1999, Section 2]

Newton’s method attacks the necessary optimality condition $\hat{f}'(u) = 0$. Its application results in the iteration

$$
\nabla^2 \hat{f}(u_n) d_n = -\hat{f}'(u_n), \quad u_{n+1} := u_n + d_n,
$$

(2.5)

Here the Hessian $\nabla^2 \hat{f}(u)$ is understood as an element of $\mathcal{L}(U, U')$, and hence $d_n \in U$.

**Algorithm 3** Newton’s method

**Input:** $u_0 \in U$, $n_{\text{max}}$

**Output:**

1: Set $n := 0$ and done := false
2: while not done and $n < n_{\text{max}}$ do
3: Calculate $r_n := -\hat{f}'(u_n)$
4: if convergence criterion satisfied then
5: Set done := true
6: else
7: Solve $\nabla^2 \hat{f}(u_n) d_n := r_n$
8: Set $u_{n+1} := u_n + d_n$
9: Set $n := n + 1$
10: end if
11: end while

The reduced Hessian matrix is usually not formed explicitly due to the tremendous computational effort to do so. By contrast, (2.5) is solved iteratively, using a Krylov method such as MINRES or CG, which take advantage of the symmetry of $\nabla^2 \hat{f}(u_n)$. Every iteration then requires the evaluation of one matrix-vector product $\nabla^2 \hat{f}(u) \delta u$. Algorithm 4 describes how to achieve this.

Quasi-Newton methods, such as BFGS, offer an alternative to the exact evaluation of the Hessian matrix. By constrast, they store and accumulate gradient information from iteration to iteration as a substitute for second derivatives. Due to the high dimension of discretized optimal control problems, limited-memory versions such as LM-BFGS, should be employed.

The basic Newton method as stated in Algorithm 3 has good local convergence properties (at q-superlinear or even q-quadratic rates). But in order to solve truly nonlinear problems, it has to be globalized, for instance, by embedding it into a
trust-region framework, or by using a truncated version, which stops the iterative solution of (2.5) when encountering directions of negative curvature, see [Nocedal and Wright, 2006, Section 6].

Algorithm 4 Evaluation of the reduced Hessian times a vector $\nabla^2 \hat{f}(u) \delta u$

**Input:** $y = S(u), p = -e_y(y, u)^* f_y(y, u)$ (adjoint state)

**Output:** $\nabla^2 \hat{f}(u) \delta u$

1: Solve the linearized state equation $e_y(y, u) \delta y = -e_u(y, u) \delta u$ for $\delta y \in Y$

2: Calculate the right hand side $t := L_yu(y, u, p)(\cdot, \delta u) + L_yy(y, u, p)(\cdot, \delta y) \in Y'$

3: Solve the adjoint state equation $e_y(y, u)^* \delta p = -t$ for $\delta p \in Z$

4: Return $\nabla^2 \hat{f}(u) \delta u := L_{uu}(y, u, p)(\cdot, \delta u) + L_{uy}(y, u, p)(\cdot, \delta y) + e_u(y, u)^* \delta p \in U'$

§ 2.2 All-at-once Methods

As was already mentioned, all-at-once methods keep the PDE constraint $e(y, u) = 0$ as an explicit side constraint during the optimization. Both the state and control variables $x = (y, u)$ are optimization variables now. The Lagrangian associated with problem (2.1) is defined as

$$L(y, u, p) = f(y, u) + \langle p, e(y, u) \rangle_{Z, Z'}.$$  

We will often combine $x = (y, u)$ into one variable for conciseness of our notation. We briefly review three important classes of methods for the solution of problem (2.1).

§ 2.2.1 First-Order Augmented Lagrangian Methods

**Literature:** [Ito and Kunisch, 2008, Chapter 3], [Bertsekas, 1996, Chapter 2]

Augmented Lagrangian methods (aka method of multipliers) are related to Lagrangian methods as well as penalty methods. Lagrangian methods take turns in minimizing w.r.t. $x$ the Lagrangian with $p_n$ fixed

$$f(x) + \langle p_n, e(x) \rangle$$

and then updating $p_n$ so that the Lagrange dual function is increased.

An example for the class of penalty methods is the quadratic penalty approach, which—in the present context—considers a family of unconstrained problems

$$\text{Minimize } f(y, u) + \frac{c}{2} \| e(y, u) \|_{Z'}^2 \quad \text{over } (y, u) \in Y \times U.$$  

The disadvantage of these methods is that the penalty parameter $c$ needs to be driven to $\infty$ which renders the resulting problems increasingly ill-conditioned.

Combining these two ideas leads to Algorithm 5. It stands out as a feature that the parameter $c$ does not need to be taken to $\infty$.

Let us introduce the Augmented Lagrangian functional

$$L_c(x, p) := f(x) + \langle p, e(x) \rangle + \frac{c}{2} \| e(x) \|_{Z'}^2.$$  

The derivative w.r.t. $x$ of $L_c$ is given by

$$L_{c,x}(x, p) \delta x = L_x(x, p) \delta x + c \langle e(x), e_x(x) \delta x \rangle_{Z'}.$$  \hspace{1cm} (2.6)
This implies that every KKT point (satisfying \(e(x^*) = 0\) and \(L_x(x^*, z^*) = 0\)) will also be a stationary point for \(L_{c,x}\) for any \(c > 0\).

The Hessian w.r.t. \(x\) of \(L_c\) is given by
\[
L_{c,xx}(x, p)(\delta x_1, \delta x_2) = L_{xx}(x, p)(\delta x_1, \delta x_2) + c \left( e_x(x) \delta x_1, e_x(x) \delta x_2 \right)_Z, \\
+ c \left( e(x), e_{xx}(x)(\delta x_1, \delta x_2) \right)_Z. 
\] (2.7)

Suppose that \((x^*, p^*)\) is a point satisfying second-order sufficient conditions, i.e., \((x^*, z^*)\) is a KKT point and the Hessian \(L_{xx}(x^*, p^*)\) is positive definite on the nullspace of \(e_x(x^*)\). Then we infer from (2.7) that for \(c > 0\), the Hessian of \(L_c\) has better coercivity properties than the Hessian of the original Lagrangian \(L\):
\[
L_{c,xx}(x^*, p^*)(\delta x, \delta x) = L_{xx}(x^*, p^*)(\delta x, \delta x) + c \| e_x(x^*) \delta x \|^2_Z, \\
\geq L_{xx}(x^*, p^*)(\delta x, \delta x).
\]

Indeed, one can prove under appropriate assumptions that for \(c \geq c_0\), the Hessian \(L_{c,xx}(x, p)\) is positive definite on all of \(X\), uniformly for any \(x\) in some neighborhood of \(x^*\).

**Algorithm 5** First-order Augmented Lagrangian method

**Input:** \(p_0 \in Z, n_{\max}\)

**Output:**
1. Set \(n := 0\) and done := false
2. while not done and \(n < n_{\max}\) do
3. Solve for \(x_{n+1}\)
   
   Minimize \(L_{c_n}(x, p_n) := f(x) + \langle p_n, e(x) \rangle + \frac{c_n}{2} \| e(x) \|^2_Z\), over \(x \in X\)
4. Update the adjoint state
   
   \(p_{n+1} := p_n + \sigma_{n+1} e(x_{n+1})\)
5. if convergence criterion satisfied then
6.   Set done := true
7. else
8.   Set \(c_{n+1}\) and \(n := n + 1\)
9. end if
10. end while

The choice of the augmentation parameter \(c_n\) in Algorithm 5 is important in practice, yet "no general purpose techniques appear to be available" (see [Ito and Kunisch, 2008, p.77] and [Bertsekas, 1996, Chapter 2]). In the update step 4 for the adjoint state, another parameter \(\sigma_{n+1}\) has to be chosen.

Finally, we comment on why Algorithm 5 is termed a first-order algorithm. To this end, we introduce the Lagrangian dual function associated with problem (2.1), i.e.,
\[
q(p) = \inf_{x \in X} L(x, p) = \inf_{x \in X} f(x) + \langle p, e(x) \rangle_{Z, Z'}.
\]

At the minimum \(\bar{\pi}\) (suppose it exists), necessarily \(f_x(\bar{\pi}) + \langle p, e_x(\bar{\pi}) \rangle = 0\) holds, and we consider \(\bar{\pi}(\cdot)\) a function of \(p\). By the chain rule, the derivative of \(q\) is
\[
q'(p) \delta p = \langle f_x(\bar{\pi}) + e_x(\bar{\pi})^* p, \bar{\pi}(p) \delta p \rangle + \langle \delta p, e(\bar{\pi}) \rangle = \langle \delta p, e(\bar{\pi}) \rangle,
\]
since the first term is zero. Recalling that the Lagrange dual problem is

\[ \text{Maximize } q(p) \text{ over } p \in Z, \]

step 4 of Algorithm 5 is a step in the direction of steepest ascent towards the solution of the dual problem, hence the attribute 'first-order'.

Clearly, the main effort in every iteration is the solution of the primal unconstrained minimization problem in step 3. Due to ill-conditioning of the problem for large values of \( c_n \), gradient-based methods are out of the question and Newton-type methods (§ 2.1.3) should be used.

§ 2.2.2 SQP Methods

Literature: [Ito and Kunisch, 2008, Chapter 5.3], [Nocedal and Wright, 2006, Chapter 18], [Tröltzsch, 2010, Section 4.11], Alt [1994], Hinze and Kunisch [2001]

SQP (sequential quadratic programming) methods solve a sequence of QP (quadratic programming) problems built from successive second-order models of the original problem. At a given, or current, point \((x_n, p_n)\), this QP is

\[
\begin{align*}
\text{Minimize} & \quad \frac{1}{2} \mathcal{L}_{xx}(x_n, p_n)(x - x_n, x - x_n) + f_x(x_n)(x - x_n) \\
\text{subject to} & \quad \epsilon_x(x_n)(x - x_n) + \epsilon(x_n) = 0.
\end{align*}
\]

(2.8)

In the most basic form of the SQP algorithm, the solution \(x\) of (2.8) (suppose it exists) and the Lagrange multiplier \(p\) (adjoint state) associated with the linearized equality constraint are used as subsequent iterates \((x_{n+1}, p_{n+1})\).

Algorithm 6 Basic SQP algorithm

Input: \(x_0 \in X, p_0 \in Z, n_{\text{max}}\)

Output:

1: Set \(n := 0\) and done := false
2: while not done and \(n < n_{\text{max}}\) do
3: Solve the QP (2.8) for \((x_{n+1}, p_{n+1})\)
4: if convergence criterion satisfied then
5: Set done := true
6: else
7: Set \(n := n + 1\)
8: end if
9: end while

It is easy to verify that Algorithm 6 is equivalent to Newton’s method, applied to the KKT conditions \(\mathcal{L}_x(x, p) = 0\) and \(\epsilon(x) = 0\). Therefore, the SQP method is sometimes referred to as the Lagrange-Newton method. The Newton step reads

\[
\begin{pmatrix}
\mathcal{L}_{yy}(\cdot) & \mathcal{L}_{yu}(\cdot) & e_y(\cdot) \\
\mathcal{L}_{uy}(\cdot) & \mathcal{L}_{uu}(\cdot) & e_u(\cdot) \\
e_y(\cdot) & e_u(\cdot) & 0
\end{pmatrix}
\begin{pmatrix}
\delta y \\
\delta u \\
\delta p
\end{pmatrix}
= - \begin{pmatrix}
\mathcal{L}_y(\cdot) \\
\mathcal{L}_u(\cdot) \\
e(\cdot)
\end{pmatrix},
\]

(2.9)

where \((\cdot)\) stands for the current iterate \((y_n, u_n)\) or \((y_n, u_n, p_n)\) as appropriate. Indeed, there are many similarities between Algorithm 6 and Newton’s method for the
§ 2. Methods for Unconstrained Problems

reduced problem, Algorithm 3. It can be shown that the Newton direction $d_n$ in Algorithm 3 satisfies

$$
\begin{pmatrix}
L_{yy}(\cdot) & L_{yu}(\cdot) & e_y(\cdot)^* & 0 \\
L_{uy}(\cdot) & L_{uu}(\cdot) & e_u(\cdot)^* & 0 \\
e_y(\cdot) & e_u(\cdot) & 0 & 0
\end{pmatrix}
\begin{pmatrix}
\delta y \\
\delta u \\
d_n \\
\delta p
\end{pmatrix}
= -
\begin{pmatrix}
0 \\
0 \\
L_u(\cdot) \\
e_y(\cdot)
\end{pmatrix}.
$$

However, $\delta y$ and $\delta z$ are not used as updates in Algorithm 3, but they are discarded and the state equation and adjoint equation are solved exactly in every iteration.

§ 2.2.3 Augmented-Lagrangian SQP Methods

Literature: [Ito and Kunisch, 2008, Chapter 6], Volkwein [2000, 1997]

The weakest link in the first-order Augmented Lagrangian Algorithm 5 (see Section 2.2.1) is the first-order (gradient-based) update of the dual variables (the adjoint state). Augmented-Lagrangian SQP methods improve on this point and employ a second-order update formula. It will become clear that the resulting method is closely related to SQP methods, which determines the name of the algorithm.

Our starting point is again the Augmented Lagrangian functional

$$
L_c(x, p) := f(x) + \langle p, e(x) \rangle + \frac{c}{2} \|e(x)\|_Z^2,
$$

and the observation that every KKT point for (2.1) (satisfying $e(x^*) = 0$ and $L_x(x^*, p^*) = 0$) will also be a stationary point for $L_{c,x}$ for any $c > 0$. Let us apply Newton’s method to the system

$$
L_{c,x}(x, p) = 0, \quad e(x) = 0,
$$

which results in the iteration

$$
\begin{pmatrix}
L_{c,xx}(x_n, p_n) & e_x(x_n)^* \\
e_x(x_n) & 0
\end{pmatrix}
\begin{pmatrix}
x - x_n \\
p - p_n
\end{pmatrix}
= -
\begin{pmatrix}
L_{c,x}(x_n, p_n) \\
e_x(x_n)
\end{pmatrix}.
$$

A straightforward computation (see [Ito and Kunisch, 2008, p.158–159]) shows that (2.10) is equivalent to first setting $\tilde{p}_n := p_n + cRe(x_n)$, with $R : Z' \to Z$ the Riesz isomorphism, and then solving

$$
\begin{pmatrix}
L_{xx}(x_n, \tilde{p}_n) & e_x(x_n)^* \\
e_x(x_n) & 0
\end{pmatrix}
\begin{pmatrix}
x - x_n \\
p - \tilde{p}_n
\end{pmatrix}
= -
\begin{pmatrix}
L_x(x_n, \tilde{p}_n) \\
e(x_n)
\end{pmatrix},
$$

which avoids the need to actually form derivatives of $L_c$.

Altogether, we obtain Algorithm 7, which coincides with [Ito and Kunisch, 2008, Algorithm 6.3]. For the choice $c = 0$, Algorithm 7 reduces to the SQP method (Algorithm 6).

Note that, in contrast to the first-order Augmented Lagrangian Algorithm 5, the system (2.11) combines the primal and dual updates into one iteration rather than carrying them out subsequently. At some additional cost per iteration, the next iterate $x_{n+1}$ could be alternatively determined from a primal minimization w.r.t. $x$, and then (2.11) might only be used to update the dual variable, see [Ito and Kunisch, 2008, Algorithms 6.1, 6.2].
§ 3 Treatment of Inequality Constraints

Inequality constraints always add nonlinearity to the optimization problem (1.1), even if $f$, $e$, and $g$ are linear. This is due to the complementarity condition $\langle \mu, g(x) \rangle = 0$ which introduces a multiplicative coupling.

It becomes evident already in finite dimensional optimization that it is important to distinguish between simple coordinatewise constraints ($a \leq x \leq b$) and more involved constraints such as coupled linear ($a \leq Ax \leq b$) or nonlinear ones. In PDE-constrained optimization, an additional feature which separates ‘easy’ from ‘hard’ constraints is whether the inequality constraint involves the control and/or the state variables.

In a first attempt, one may try and treat the inequality constraints in an outer loop. For instance, barrier methods or penalty methods can be used to remove the inequality constraints and convert them into additional terms in the objective such as

$$-c \int_\Omega \ln(y - y_b) \, dx \quad \text{or} \quad \frac{c}{2} \| \max \{0, y - y_b\} \|_{L^2(\Omega)}^2$$

in case of a pointwise state constraint $y \leq y_b$ in $\Omega$. Then any of the approaches from § 2 can be used in the inner loop. This is illustrated in Figure 3.1.

However, more efficient strategies are usually obtained when dealing with the inequality constraints in the main optimization loop. We highlight some popular algorithms for important special cases in the subsequent sections.

§ 3.1 Control Constraints: Primal-Dual Active Set Strategy

Literature: Bergoumioux et al. [1999], [Ito and Kunisch, 2008, Chapter 7], Rösch and Kunisch [2002], Ulbrich [2003], [Tröltzsch, 2010, Section 2.12]

Let us state the following model problem:

$$\begin{align*}
\text{Minimize} \quad & f(y, u) \quad \text{over} \quad (y, u) \in Y \times U \\
\text{s.t.} \quad & e(y, u) = 0 \\
& u_a \leq u \leq u_b \quad \text{in} \quad \Omega_0.
\end{align*}$$

(3.1)

The control constraint is posed on the set $\Omega_0$ where the control is defined, e.g., a subset of $\Omega$, or part of its boundary. The case of finite dimensional controls is also
§ 3. Treatment of Inequality Constraints

Figure 3.1. Schematic of algorithms with inequalities treated in an outer loop.

included with appropriate interpretation. Under suitable assumptions, necessary optimality conditions for (3.1) are given by

\[
\begin{align*}
    f_y(y,u) \delta y + e_y(y,u) \delta y &= 0 \quad \text{for all } \delta y \in Y \\
    f_u(y,u) \delta u + e_u(y,u) \delta u + \mu &= 0 \quad \text{for all } \delta u \in U \\
    e(y,u) &= 0 \\
    u_a \leq u \leq u_b, \quad u = u_a \text{ where } \mu < 0, \quad u = u_b \text{ where } \mu > 0.
\end{align*}
\]  

Equation (3.2d) gives rise to a nonlinear relationship between the Lagrange multiplier \( \mu \) and the control variables \( u \).

§ 3.1.1 Primal-Dual Active Set Strategy as an Outer Iteration

We briefly review the primal-dual active set strategy (PDAS) which converts (3.1) into a sequence of equality-constrained problems (3.7), so that methods from § 2 can be applied. The PDAS can be motivated by starting off from the equivalent re-formulation of (3.1)

\[
\begin{align*}
    \text{Minimize} & \quad f(y,u) + I_{U_{ad}}(u) \quad \text{over } (y,u) \in Y \times U \\
    \text{s.t.} & \quad e(y,u) = 0
\end{align*}
\]  

which uses the indicator function

\[
I_{U_{ad}}(u) = \begin{cases} 
0 & \text{if } u \in U_{ad} \\
+\infty & \text{if } u \notin U_{ad}
\end{cases}
\]

of the convex admissible set

\[U_{ad} = \{ u \in U : u_a \leq u \leq u_b \leq 0 \text{ in } \Omega_0 \}.
\]

The optimality conditions for (3.3) involve an element \( \mu \in \partial I_{U_{ad}}(u) \) from the convex subdifferential of \( I_{U_{ad}} \), which is equivalent to the complementarity relation (3.2d).

In convex analysis, the generalized Moreau-Yosida approximation

\[
\varphi_c(w;\mu) = \inf_{v \in U} \{ \varphi(u - v) + (\mu, v)_U + \frac{c}{2} \| v \|^2_U \} \quad \text{for given } \mu \in U
\]
of a convex function \( \varphi : U \to \mathbb{R} \cup \{+\infty\} \) (with further properties) is a powerful tool to regularize the potentially non-smooth function \( \varphi \). In the case of \( \varphi = I_{U_{\text{ad}}} \) and \( U = L^2(\Omega_0) \), one can show

\[
I_{U_{\text{ad}},c}(u; \mu) = \int_{\Omega} i_{U_{\text{ad}},c}(u(\xi); \mu(\xi)) \, d\xi
\]

(3.4)

\[
(I_{U_{\text{ad}},c})'(u; \mu) = c \left( u + \frac{1}{c} \mu - \text{proj}_{U_{\text{ad}}}(u + \frac{1}{c} \mu) \right),
\]

(3.5)

where

\[
i_{U_{\text{ad}},c}(u; \mu) = -\frac{1}{2c} |\mu|^2 + \frac{c}{2} \max\{0, u - u_b + \frac{1}{c} \mu\}^2 + \frac{c}{2} \min\{0, u - u_a + \frac{1}{c} \mu\}^2
\]

and \( \text{proj}_{U_{\text{ad}}} \) is the pointwise projection onto \( U_{\text{ad}} \), i.e.,

\[
\text{proj}_{U_{\text{ad}}}(u) = \max\{u_a, \min\{u, u_b\}\}.
\]

An illustration of \( I_{U_{\text{ad}},c}(\cdot; \mu) \) for the case \( U = \mathbb{R} \) and various values of \( c \) and \( \mu \) is given in Figure 3.2.

\[
\mu = 0 \quad \mu = 10 \quad \mu = 20
\]

\[
c = 10
\]

\[
c = 50
\]

\[
c = 250
\]

**Figure 3.2.** An illustration of the generalized Moreau-Yosida regularization of the indicator function \( I \) of an interval in \( \mathbb{R} \).

At this point, there are several possibilities to use the Moreau-Yosida approximation:

(a) We could solve, instead of (3.3), a family of regularized problems

\[
\begin{aligned}
\text{Minimize} & \quad f(y, u) + I_{U_{\text{ad}},c}(u; \mu) \\
\text{over} & \quad (y, u) \in Y \times U \\
\text{s.t.} & \quad e(y, u) = 0,
\end{aligned}
\]

(3.6)

perhaps for some fixed choice of \( \mu \), and drive \( c \to \infty \). This is a penalty method, and \( \mu \neq 0 \) causes a shift of the threshold where the penalty kicks in, see Figure 3.2.
§ 3. Treatment of Inequality Constraints

(b) However, there is a more efficient method to treat control constraints, which uses the fact that

\[
\mu \in \partial I_{U_{ad}}(u) \iff \mu = (I_{U_{ad}})_t(u; \mu)
\]

for some (and then for all) \( c > 0 \), see [Ito and Kunisch, 2000, Theorem 2.4]. This relationship, together with (3.5), motivates the use of the multiplier rule

\[
\mu = c \left( u + \frac{1}{c} \mu - \text{proj}_{U_{ad}} \left( u + \frac{1}{c} \mu \right) \right)
\]

for some \( c > 0 \) as a prediction strategy for the sets where \( u = u_a \) or \( u = u_b \) should hold. (Recall that this is encoded in the multiplier by (3.2d).) One arrives at Algorithm 8, the primal-dual active set strategy (PDAS).

Active set strategies have long since been used in optimization to estimate which of the inequalities of a given problem will be active (satisfied with equality) at the solution. One of the distinguishing features of PDAS is that it uses both primal \((u)\) and dual \((\mu)\) variables for this estimation.

Algorithm 8 Primal-dual active set strategy

**Input:** \( u_0 \in U, \mu_0 \in \mathcal{U} \)

**Output:**
1. Set \( n := 0 \) and \( \text{done} := \text{false} \)
2. **while** not done and \( n < n_{\text{max}} \) **do**
3.   Determine the active and inactive sets
    \[
    \mathcal{A}^-_n := \left\{ x \in \Omega_0 : \mu_n - c(u_a - u_n) < 0 \right\} \\
    \mathcal{A}^+_n := \left\{ x \in \Omega_0 : \mu_n + c(u_n - u_b) > 0 \right\} \\
    \mathcal{I}_n := \Omega_0 \setminus (\mathcal{A}^+_n \cup \mathcal{A}^-_n)
    \]
4.   **if** convergence criterion satisfied **then**
5.     done := true
6.   **else**
7.     Solve the equality-constrained problem (3.7) for \( u_{n+1} \) in \( U \) and associated Lagrange multipliers \( \mu_{\pm} \)
8.     Set \( \mu_{n+1} := \mu_{+} - \mu_{-} \)
9.     Set \( n := n + 1 \)
10. **end if**
11. **end while**

In every iteration of PDAS, the following equality-constrained problem has to be solved.

\[
\begin{align*}
\text{Minimize} \quad & f(y, u) \quad \text{over} \quad (y, u) \in Y \times U \\
\text{s.t.} \quad & e(y, u) = 0 \\
& \quad \text{and} \quad u = u_a \text{ in } \mathcal{A}^-_n, \quad u = u_b \text{ in } \mathcal{A}^+_n.
\end{align*}
\]

(3.7)

This can be achieved, for instance, by restricting \( u \) to the inactive set \( \mathcal{I}_n \) and then using any of the algorithms considered in § 2, or alternatively, by adding \( u = u_a \) and \( u = u_b \) to the problem as additional equality constraints. We obtain PDAS-Newton or PDAS-SQP, for instance.
§ 3.1.2 Primal-Dual Active Set Strategy within Newton and SQP Iterations

Literature: Tröltzsch [1999], Griesse et al. [2008], [Griesse, 2007, Chapter 2]

We have just outlined in the previous section an algorithmic scheme as in Figure 3.1, where the inequalities are removed (by the primal-dual active set approach) through an outer iteration, and then Newton’s method or an SQP method is used as an inner loop.

However, Newton’s method (§ 2.1.3) and the SQP method (§ 2.2.2) can naturally be extended to handle inequality constraints, and thus the PDAS might as well be used in an inner loop, even in the case of pointwise nonlinear inequality constraints.

Despite the complementarity conditions present in problems with inequality constraints, the optimality condition can be written in a form close to an equation, namely a generalized equation, see Robinson [1980]. In the case of the reduced formulation appropriate for Newton’s method, this generalized equation takes the form

\[ 0 \in F(u, \mu) + \mathcal{N}(\mu), \]  

where \( \mathcal{N}(\mu) \) is the normal cone of the set of non-negative Lagrange multipliers

\[ K^+ = \{ \mu \in L^2(\Omega_0) : \mu \geq 0 \text{ a.e. in } \Omega_0 \}. \]

For example, in the case of problem (2.2) with the one-sided pointwise control constraint \( u \leq u_b \) in \( \Omega_0 \), one has

\[ F(u, \mu) = \left( \nu u - p(u) + \mu \right) \]

where \( p(u) \) is the adjoint state belonging to \( u \), and the normal cone is given by

\[ \mathcal{N}(\mu) = \{ 0 \} \times \begin{cases} \emptyset & \text{if } \mu \notin K^+ \\ \{ z \in L^2(\Omega) : (z, \nu - \mu)_{L^2(\Omega_0)} \leq 0 \text{ for all } \nu \in K^+ \} & \text{if } \mu \in K^+. \end{cases} \]

The Newton iteration for (3.8) reads

\[ 0 \in F(u_n, \mu_n) + F'(u_n, \mu_n)(u - u_n, \mu - \mu_n) + \mathcal{N}(\mu), \]

which is to be solved for \((u_{n+1}, \mu_{n+1})\) and which constitutes the optimality system for a QP with linearized inequality constraints, which can be solved using PDAS, for instance. This leads to Newton-PDAS.

In the same manner, SQP methods naturally handle nonlinear inequality constraints, say, \( g(u) \leq 0 \). Then the sequence of QPs will contain linearized inequality constraints (compare (2.8)):

\begin{align*}
\text{Minimize} & \quad \frac{1}{2} \mathcal{L}_{xx}(x_n, p_n)(x - x_n, x - x_n) + f_x(x_n)(x - x_n) \quad \text{over } x \in X \\
\text{s.t.} & \quad e_x(x_n)(x - x_n) + e(x_n) = 0 \\
& \quad \text{and } \quad g(u_n)(u - u_n) + g(u_n) \leq 0.
\end{align*}  

Again, (3.9) can be solved using PDAS, which results in the SQP-PDAS approach. We depict this situation in Figure 3.3.
The close relationship pointed out in § 2.2.2 between SQP and Newton’s method is maintained since (3.9), too, can be interpreted as a Newton step for a generalized equation of the form

\[ 0 \in F(y, u, p, \mu) + N(\mu), \]

which now involves the state and the adjoint state as well.

Finally, we remark that the Augmented Lagrangian-SQP approach (§ 2.2.3) can also be extended to cover inequality constraints. We refer to [Ito and Kunisch, 2008, Chapter 6.3, Algorithm 6.6] for details.

§ 3.1.3 Primal-Dual Active Set Strategy as a Semismooth Newton Iteration

Literature: Hintermüller et al. [2002], [Ito and Kunisch, 2008, Chapter 8.4], Ulbrich [2003]

There is even one further possibility which combines the linearization and PDAS into one single loop. This interpretation is based on a reformulation of the complementarity condition (3.2d) using a particular nonlinear complementarity (NCP) function. Apparently, \( \max\{a, b\} = 0 \) holds if and only if \( a \leq 0, b \leq 0 \) and \( ab = 0 \). This remains true if \( b \) is replaced by \( cb \) for any \( c > 0 \). It is easy to check that (3.2d) can be equivalently stated as

\[
\mu - \min\{0, \mu - c(u_a - u)\} - \max\{0, \mu + c(u - u_b)\} = 0 \quad \text{in } \Omega_0,
\]

and hence (3.2) becomes a system of equations, which are, however, not differentiable in the usual sense. Nevertheless, this modified system can be shown to be semismooth under certain conditions. Note that the subsets of \( \Omega_0 \) where the max or min are attained by the non-zero terms coincide with the active sets occurring in Algorithm 8. The derivative of \( \max\{0, u\} \) can be determined pointwise and it is either 0 or 1, depending on whether \( u > 0 \) or \( u \leq 0 \). Consequently, the resulting semismooth Newton iteration takes the form of an active set method.

Depending on whether we use the nonlinear complementarity condition (3.10) in the context of the black-box or the all-at-once framework, we obtain two more methods which differ from PDAS-Newton and Newton-PDAS, or from PDAS-SQP and SQP-PDAS mainly with respect to when the active sets are updated.

§ 3.2 Mixed Control-State Constraints

Literature: Meyer et al. [2005], Meyer et al. [2007], Griesse et al. [2008], Rösch and Wachsmuth [2010]
Chapter 1. Algorithms in PDE-Constrained Optimization

Mixed control-state constraints, such as \( m_a \leq y + u \leq m_b \) or nonlinear variants thereof, behave in many ways like control constraints considered in § 3.1. There is a natural extension of the primal-dual active set method in the all-at-once context. We refer to the literature above.

§ 3.3 State Constraints

Literature: Ito and Kunisch [2003], Hintermüller and Kunisch [2006], Hintermüller and Hinze [2008]

The treatment of pointwise state constraints, such as \( y_a \leq y \leq y_b \) on \( \Omega \), is more involved. The reason lies partly with the associated Lagrange multiplier \( \mu \), which is not, in general, a function, but only a Borel measure \( \mu \in C(\Omega)' \), see Casas [1986, 1993]. Consequently, the ideas which led to the PDAS method for control constraints, cannot be repeated. In particular, we fail to give a meaning to the analog of \( \mu = (I_{(u,a,c)}')^T(u;\mu) \). Neither can we pursue the alternative motivation (3.10) of PDAS because the complementarity condition does not have a pointwise interpretation in case of state constraints.

At this point we recall that Augmented Lagrangian Methods (see § 5) were capable of removing undesired constraints by way of adding a penalty term to the Lagrangian. We only need to extend this idea to inequality constraints. For simplicity we consider for a moment only an upper bound \( y - y_b \leq 0 \), which we convert into an equality constraint \( y - y_b + s = 0 \) by means of a slack variable \( s \geq 0 \), which becomes an additional optimization variable.

Applying the Augmented Lagrangian idea to our state-constrained problem with slack variable

\[
\begin{align*}
\text{Minimize} & \quad f(y, u) \quad \text{over} \quad (y, u, s) \in Y \times U \times L^2(\Omega) \\
\text{s.t.} & \quad e(y, u) = 0 \\
& \quad y - y_b + s = 0 \quad \text{in} \quad \Omega, \quad s \geq 0 \quad \text{in} \quad \Omega,
\end{align*}
\]

we arrive at the Augmented Lagrangian functional

\[
L_c(y, u, \mu, s) := f(y, u) + (\mu, y - y_b + s)_{L^2(\Omega)} + \frac{c}{2} \|y - y_b + s\|_{L^2(\Omega)}^2. \tag{3.11}
\]

Note that we made a conscious choice here by augmenting the constraint \( y - y_b + s = 0 \) with respect to the \( L^2(\Omega) \) norm. We also chose not to augment the PDE constraint \( e(y, u) = 0 \), rather we keep it explicitly as a side constraint.

Looking back at the first-order Augmented Lagrangian method (Algorithm 5, step 3) we see that we will need to find a minimizer of \( L_c \) with respect to the primal variables \( (y, u, s) \). A closer look at (3.12) shows that \( L_c \) is actually uniformly convex w.r.t. \( s \), and hence a partial minimization w.r.t. \( s \) can be carried out analytically. If it weren’t for the constraint \( s \geq 0 \), we would find the minimizer to be \( \bar{s}(y, \mu) = -(y - y_b) - \mu/c \).

With the constraint, we have

\[
\bar{s}(y, \mu) = \max \left\{ 0, -(y - y_b) - \frac{1}{c} \mu \right\}.
\]

By plugging this expression into \( L_c \) we arrive at

\[
L_c(y, u, \mu) := f(y, u) + \frac{c}{2} \left\| \max \{0, y - y_b + \frac{1}{c} \mu\} \right\|_{L^2(\Omega)}^2 - \frac{1}{2c} \|\mu\|_{L^2(\Omega)}^2. \tag{3.13}
\]
§ 3. Treatment of Inequality Constraints

Wait a minute, we have encountered terms like these before. Indeed, the Augmented Lagrangian functional and the original objective plus a generalized Moreau-Yosida penalty,
\[ L_c(y, u, \mu) = f(y, u) + I_{Y_{ad}, c}(y; \mu), \]
are the same! Here we use \( Y_{ad} = \{ y \in L^2(\Omega) : y \leq y_b \} \) and \( I_{Y_{ad}, c}(y; \mu) \) is similar to (3.4).

We can therefore proceed from here in either of two ways:

(a) We may use (3.13) within a first-order Augmented Lagrangian method, by minimizing (3.13) w.r.t. \((y, u, p)\), subject to \(e(y, u) = 0\), for some current values of \(c_n \in \mathbb{R}\) and \(\mu_n \in L^2(\Omega)\), then perform an update for the dual variable \(\mu_n\) associated with the augmented constraint. This is Algorithm 9, with the setting slightly extended to cover bilateral constraints \(y_a \leq y \leq y_b\), and the term \(\frac{1}{2c_n} \| \mu \|^2_{L^2(\Omega)}\) omitted from the objective in step 3 since it is constant from the point of view of the 'primal' variables \((y, u, p)\).

(b) Or we may use (3.13) in the spirit of a penalty method, perhaps keep \(\mu\) fixed, but drive \(c \to \infty\).

Both variants were analyzed and tested in Ito and Kunisch [2003], and the penalty approach was found to perform better in practice.

**Algorithm 9** First-order Augmented Lagrangian method for state-constrained problems

<table>
<thead>
<tr>
<th>Input: ( \mu_0 \in L^2(\Omega) ), ( n_{\text{max}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
</tr>
<tr>
<td>1. Set ( n := 0 ) and ( \text{done := false} )</td>
</tr>
<tr>
<td>2. while not done and ( n &lt; n_{\text{max}} ) do</td>
</tr>
<tr>
<td>3. Solve for ((y_{n+1}, u_{n+1}, p_{n+1}))</td>
</tr>
<tr>
<td>Minimize ( L_{c_n}(y, u, \mu_n) := f(y, u) + \frac{c_n}{2} | \max{0, y - y_b + \frac{1}{c_n} \mu_n} |^2_{L^2(\Omega)} )</td>
</tr>
<tr>
<td>( + \frac{c_n}{2} | \max{0, y - y_a - \frac{1}{c_n} \mu_n} |^2_{L^2(\Omega)} ) over ((y, u) \in Y \times U)</td>
</tr>
<tr>
<td>s.t. ( e(y, u) = 0 )</td>
</tr>
<tr>
<td>4. Update the Lagrange multiplier</td>
</tr>
<tr>
<td>( \mu_{n+1} := \max{0, \mu_n + c_n (y - y_b)} + \min{0, \mu_n - c_n (y_a - y)} )</td>
</tr>
<tr>
<td>5. if convergence criterion satisfied then</td>
</tr>
<tr>
<td>6. Set done := true</td>
</tr>
<tr>
<td>7. else</td>
</tr>
<tr>
<td>8. Set ( c_{n+1} ) and ( n := n + 1 )</td>
</tr>
<tr>
<td>9. end if</td>
</tr>
<tr>
<td>10. end while</td>
</tr>
</tbody>
</table>

We briefly mention an alternative method of regularizing state constraints, which goes back to Meyer et al. [2005, 2007]. The idea is to replace \( y_a \leq y \leq y_b \) by the mixed control-state constraint \( y_a \leq y + \varepsilon u \leq y_b \), which is benign for fixed values of
This approach is termed Lavrentiev regularization and it clearly requires that $u$ and $y$ are defined on the same domain. It does not work, for instance, with distributed constraints and boundary control. For this case, Krumbiegel and Rösch [2009] have developed the so-called virtual control concept, which has similarities to the generalized Moreau-Yosida penalty approach.
CHAPTER 2

Preconditioning in PDE-Constrained Optimization

Contents

4 Introduction 25
5 Properties of Saddle Point Problems 26
5.1 Saddle Point Problems Arising in Optimization 27
5.2 Hilbert Space Setting 28
5.3 Spectral Properties 29
5.4 An Optimal Control Example 30
6 Preconditioning KKT Systems 33
6.1 Early Approaches 34
6.2 Constraint Preconditioners 35
6.3 Preconditioned Conjugate Gradients 35
6.4 Implementation Details 38
6.4.1 PCG Solver 40
6.4.2 Scalar Product 40
6.4.3 Preconditioner 40

§ 4 Introduction

Literature: Saad [2003], Simoncini and Szyld [2007]

Despite remarkable progress in the area of sparse direct solvers for linear systems of equations, iterative solvers are most often the only feasible solution approaches for very large-scale problems. In particular, sparse factorizations of PDE problems discretized by finite element, finite difference or finite volume methods in 3D (three spatial dimensions) suffer from significant fill-in, which limits the applicability of direct solution methods.

The convergence of iterative solvers for $Ax = b$ depends on the spectral properties of $A$ like the distribution of the eigenvalues or its condition number. Unfortunately, matrices arising in most discretization approaches to PDEs are ill-conditioned. For instance, the finite element stiffness matrix $A$ of a 2nd-order elliptic PDE operator has a condition number proportional to $h^{-2}$. Therefore, iterative solvers will only be effective when combined with a preconditioner $P$. The preconditioned solver will then solve the equivalent system $P^{-1}Ax = P^{-1}b$, which hopefully has better spectral properties. It is a requirement that systems $Py = f$ are significantly easier to solve than the problem $Ax = b$ itself.

It was already mentioned in Chapter 1 that those optimization methods with a faster-than-linear rate of convergence typically need to solve, in every iteration, a simplified version of the original problem. To illustrate this point, we restrict our discussion to the SQP (§ 2.2.2) and Augmented Lagrangian SQP (§ 2.2.3) methods.
both of these methods, we have to solve in every iteration a quadratic programming (QP) subproblem, which is represented by a linear system of equations governed by the matrix
\[
\begin{pmatrix}
  L_{xx} & e_x^* \\
e_x & 0
\end{pmatrix}
\]
evaluated at the current iterate, compare (2.9) and (2.11). Matrices of this type are termed **saddle point matrices**, and the corresponding linear systems are **saddle point systems**. The name comes from the fact that solving KKT systems is (at least for convex problems) equivalent to finding a saddle point for the Lagrangian.

We conclude that the efficient solution of saddle point problems is of great importance for the efficient solution of nonlinear optimization problems. Therefore, we address in § 5 their spectral properties and in § 6 their preconditioned solution by appropriate Krylov subspace methods.

§ 5 Properties of Saddle Point Problems

We begin by explaining some concepts in finite dimensions and later switch back to the Hilbert space setting in Section 5.2. Sometimes saddle point problems are defined to be matrices partitioned in the following way:
\[
\begin{pmatrix}
  A & B_1^T \\
  B_2 & -C
\end{pmatrix}
\]
This definition is so general that actually every matrix becomes a saddle point matrix. We will here adhere to the special case
\[
\mathcal{K} = \begin{pmatrix}
  A & B^T \\
  B & 0
\end{pmatrix}
\text{ with } A = A^T
\] (5.1)
and use the term **saddle point matrix** exclusively for these. Notice that the SQP and AL-SQP methods produce subproblems of this type. This remains true also in the presence of inequality constraints, for instance, with control constraints treated using the primal-dual active set strategy, see (6.8), or with state constraints as in § 3.3, see [Herzog and Sachs, 2010, Section 3]. We mention that in some optimization methods, most notably interior point methods, the \(-C\) block is present, but these will not be considered here.

Let us suppose from now on that \(A \in \mathbb{R}^{n \times n}\) and \(B \in \mathbb{R}^{m \times n}\) with \(m \leq n\).

The first question is: Under what conditions is (5.1) invertible? This is easy to characterize in case \(A = A^\top \succeq 0\) (\(A\) is positive semidefinite). Then (5.1) is invertible if and only if \(B\) has full row rank \(m\), and \(\ker A \cap \ker B = \{0\}\). When the conditions \(A \succeq 0\) or \(A = A^\top\) are dropped, the situation becomes more difficult, see Gansterer et al. [2003].

In the sequel we shall work with the standing assumption that \(B\) has full row rank \(m \leq n\), and one of the following conditions holds:

<table>
<thead>
<tr>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A = A^\top \succ 0)</td>
<td>on (\ker B)</td>
</tr>
<tr>
<td>(A = A^\top \succ 0)</td>
<td>on (\ker B), (A \succeq 0)</td>
</tr>
<tr>
<td>(A = A^\top \succ 0)</td>
<td>on all of (\mathbb{R}^n)</td>
</tr>
</tbody>
</table>
§ 5. Properties of Saddle Point Problems

Note that (STD) ⇒ (STD•) ⇒ (STD).

It is well known that saddle point matrices are indefinite. Under assumption (STD), \( \mathcal{K} \) has exactly \( n \) positive and \( m \) negative eigenvalues, see for instance Chabrillac and Crouzeix [1984]. In other words, its inertia is \( \text{In}(\mathcal{K}) = (n, m, 0) \). If (STD•) holds, then \( A \) is invertible, and the claim can be easily seen using the factorization

\[
\begin{bmatrix}
A & B^T \\
B & 0
\end{bmatrix} = \begin{bmatrix}
I & 0 \\
BA^{-1} & I
\end{bmatrix} \begin{bmatrix}
A & 0 \\
0 & S
\end{bmatrix} \begin{bmatrix}
I & A^{-1}B^T \\
0 & I
\end{bmatrix}.
\]

Here \( S = -BA^{-1}B^T \) denotes the Schur complement, which is negative definite.

§ 5.1 Saddle Point Problems Arising in Optimization

Each of the conditions just mentioned actually has a natural meaning in optimization. To discuss this, consider the finite dimensional problem

\[
\begin{align*}
& \text{Minimize} & & \frac{1}{2}x^TAx - f^Tx \\
& \text{s.t.} & & Bx = g.
\end{align*}
\]

(5.2)

The Lagrangian for (5.2) is \( \mathcal{L}(x,p) = (1/2)x^TAx - f^Tx + p^T(Bx - g) \) and its KKT conditions are

\[
\begin{bmatrix}
A & B^T \\
B & 0
\end{bmatrix} \begin{bmatrix}
x \\
p
\end{bmatrix} = \begin{bmatrix}
f \\
g
\end{bmatrix}.
\]

(5.3)

Condition (STD) is sufficient to ensure the existence of a unique minimizer \( x^* \) of (5.2). To see this, let \( Z \) be a matrix whose columns form a basis of \( \ker B \). Choose \( x_0 \in \mathbb{R}^n \) satisfying \( Bx_0 = g \) (this works since \( B \) was assumed to be onto) and consider the equivalent reduced problem

\[
\begin{align*}
& \text{Minimize} & & \frac{1}{2}(x_0 + Zy)^TA(x_0 + Zy) - f^T(x_0 + Zy) \\
& \text{s.t.} & & Bx = g.
\end{align*}
\]

for \( y \in \mathbb{R}^{n-m} \). Due to the assumption that \( A \) is positive definite on ker \( B \) it follows that \( Z^TAZ \) is positive definite, therefore the problem above has a unique solution \( y^* \), its objective being uniformly convex. The solution of (5.2) is then \( x^* = x_0 + Zy^* \), and the unique associated \( p^* \) can be found from solving

\[
BB^Tp^* = B(f - Ax^*).
\]

This follows again from the condition that \( B \) has full row rank, which is usually called the LICQ (linear independence constraint qualification) in optimization, and which is known to entail the uniqueness of the Lagrange multiplier \( p \).

Let us discuss the necessity of condition (STD). It \( A \) were not positive definite on ker \( B \), there would exist in ker \( B \) a direction of negative curvature for \( (1/2)x^TAx - f^Tx \), and hence the reduced objective would be unbounded below on the feasible set \( \{ x \in \mathbb{R}^n : Bx = g \} \).

The stronger requirement (STD+) means that the objective is convex (but not necessarily uniformly so) on the whole space \( \mathbb{R}^n \). In this case, we say that (5.2) is a convex problem, and solving (5.3) is actually equivalent to finding the unique saddle point of the Lagrangian, characterized by

\[
\mathcal{L}(x^*, p) \leq \mathcal{L}(x^*, p^*) \leq \mathcal{L}(x, p^*) \quad \text{for all } x \in \mathbb{R}^n, \ p \in \mathbb{R}^m.
\]

Finally, (STD++) implies that the objective is uniformly convex on the whole space \( \mathbb{R}^n \).
While (5.2) was a quadratic programming problem, we are often interested in solving nonlinear optimization problems:

\[
\begin{align*}
\text{Minimize} &\quad f(x) \\
\text{s.t.} &\quad e(x) = 0.
\end{align*}
\] (5.4)

The Lagrangian associated with (5.5) is \( \mathcal{L}(x, p) = f(x) + p^T e(x) \). As was already mentioned in § 4, SQP type methods try to approach a solution by solving a sequence of problems of type (5.3), with \( A = \mathcal{L}_{xx}(x_n, p_n) \) (Hessian of the Lagrangian) and \( B = e_x(x_n) \) (linearization of the constraints). Let us relate the properties of the nonlinear problem (5.5) to those of (5.2). The key are second-order sufficient optimality conditions, which, in the case of (5.5), are:

Suppose that the first-order necessary conditions \( \mathcal{L}_x(x^*, p^*) = 0 \) and \( e(x^*) = 0 \) hold, and that \( e_x(x^*) \) has full row rank, i.e., LICQ holds at \( x^* \). Suppose in addition that \( \mathcal{L}_{xx}(x^*, p^*) \) is positive definite on \( \ker e_x(x^*) \).

Then \( x^* \) is a strict local optimal solution for (5.5).

We see that these conditions match precisely our minimal standard assumptions (STD) stated above. By a continuity argument, we can expect them to hold for all matrices

\[
\begin{pmatrix}
\mathcal{L}_{xx}(x, p) & e_x(x)^* \\
e_x(x) & 0
\end{pmatrix}
\]

at least in the neighborhood of a point \( (x^*, p^*) \) satisfying the second-order sufficient condition.

§ 5.2 Hilbert Space Setting

Literature: [Girault and Raviart, 1986, Chapter I, Section 4], [Quarteroni and Valli, 1994, Section 7]

In the Hilbert space setting, the matrices \( A \) and \( B \) have to be replaced by bilinear forms \( a(\cdot, \cdot) \) and \( b(\cdot, \cdot) \), which act on Hilbert spaces \( X \) and \( Q \). The equivalent of problem (5.3) becomes: Find \( x \in X, q \in Q \) such that

\[
\begin{align*}
a(x, z) + b(z, q) &= \langle f, z \rangle \quad \text{for all} \quad z \in X \\
b(x, r) &= \langle g, r \rangle \quad \text{for all} \quad r \in Q.
\end{align*}
\] (5.6)

We need to explore how our standard assumptions are to be modified for this infinite dimensional setting. First of all, we assumed \( B \) to have full row rank. This is equivalent to saying that the smallest singular value \( \sigma_{\min}(B) \) is still positive. Since singular values can be characterized by a min–max property, this is in turn equivalent to requiring

\[
\min_{p \in \mathbb{R}^m} \max_{x \in \mathbb{R}^n} \frac{p^T B x}{\|p\| \|x\|} > 0.
\]

This is precisely the condition one usually imposes in the infinite dimensional setting:

\[
\inf_{p \in Q} \sup_{x \in X} \frac{b(x, p)}{\|p\|_Q \|x\|_X} \geq k_0 > 0,
\]

the so-called inf–sup condition. In addition, we need to require \( b \) to be bounded, i.e., \( b(x, p) \leq \|b\| \|x\|_X \|p\|_Q \), which is automatic in finite dimensions.

The conditions on \( A \) are more straightforward to translate. We require that
• $a$ is symmetric: $a(x, z) = a(z, x)$ for all $x, z \in X$
• $a$ is bounded: $a(x, z) \leq \|a\| \|x\|_X \|z\|_X$ for all $x, z \in X$
• $a$ is coercive (positive definite) on $\ker B$: $a(x, x) \geq \alpha_0 \|x\|_X^2$ for all $x \in \ker B$.

These assumptions correspond to (STD) in the Hilbert space case. It should now be clear how (STD+) and (STD++) are to be understood.

Under assumptions (STD) one can show that (5.6) has a unique solution. Solving (5.6) is equivalent to solving the infinite dimensional QP

\[
\begin{align*}
\text{Minimize} & \quad \frac{1}{2} a(x, x) - \langle f, x \rangle \\
\text{s.t.} & \quad b(x, r) = \langle g, r \rangle \quad \text{for all } r \in Q.
\end{align*}
\]

(5.7)

We already mention at this point that the four numbers

$\|a\|, \alpha_0$ (pertaining to $a(\cdot, \cdot)$) and

$\|b\|, k_0$ (pertaining to $b(\cdot, \cdot)$)

contain a lot of information about the spectral properties of problem (5.6).

§ 5.3 Spectral Properties

**Literature:** Rusten and Winther [1992], Gould and Simoncini [2009]

We already mentioned that the spectral properties, e.g., distribution of eigenvalues and condition number, are important data to estimate the convergence behavior of iterative solutions methods. We turn again to the finite dimensional (discretized) setting, and we need to investigate in particular if and how the spectral properties depend on the mesh size $h$.

We quote from Rusten and Winther [1992] the following result. Suppose that (STD++) holds, and let us denote by

• $\mu_1 \geq \mu_2 \geq \cdots \geq \mu_n > 0$ the eigenvalues of $A$
• $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_m > 0$ the singular values of $B$.

Then the following bounds hold for the eigenvalues of $K$: Its spectrum $\sigma(K)$ is contained in the intervals $\mathcal{I}^- \cup \mathcal{I}^+$, where

\[
\begin{align*}
\mathcal{I}^- &= \left[ \frac{1}{2} \left( \mu_n - \sqrt{\mu_n^2 + 4 \sigma_1^2} \right), \quad \frac{1}{2} \left( \mu_1 - \sqrt{\mu_1^2 + 4 \sigma_m^2} \right) \right] \subset \mathbb{R}^-, \\
\mathcal{I}^+ &= \left[ \mu_m, \quad \frac{1}{2} \left( \mu_1 + \sqrt{\mu_1^2 + 4 \sigma_1^2} \right) \right] \subset \mathbb{R}^+.
\end{align*}
\]

An inspection of the proof of this result shows that it actually extends verbatim to the case (STD+), i.e., with $\mu_n \geq 0$ instead of $\mu_n > 0$.

This information can be used to estimate the spectral condition number of $K$, since

\[
\kappa(K) = \|K\| \|K^{-1}\| = \frac{\sigma_{\max}(K)}{\sigma_{\min}(K)}
\]

holds and for symmetric matrices, the singular values coincide with the absolute values of their eigenvalues, i.e., $\sigma_i(K) = |\lambda_i(K)|$. This implies that the condition
Chapter 2. Preconditioning in PDE-Constrained Optimization

number will be no larger than the absolute value of the bounds of $I^-$ and $I^+$ farthest away from zero, divided by the absolute value of the bounds of these intervals closest to zero:

$$\kappa(K) \leq \frac{\max\{\frac{1}{2}(\mu_1 + \sqrt{\mu_1^2 + 4\sigma_1^2}), -\frac{1}{2}(\mu_n - \sqrt{\mu_n^2 + 4\sigma_n^2})\}}{\min\{\mu_n, -\frac{1}{2}(\mu_1 - \sqrt{\mu_1^2 + 4\sigma_1^2})\}}. \quad (5.8)$$

Unfortunately, this estimate cannot be sharp since it yields $\infty$ if only $\text{(STD)}^+$ is satisfied but not $\text{(STD}^++)$. Then $\mu_n = 0$ holds and the denominator is zero, but we already know that $K$ is invertible (even under the much weaker condition $\text{(STD)}$), hence indeed $\kappa(K) < \infty$.

Recently, the eigenvalue bounds for $K$ have been improved by Gould and Simoncini [2009]. They even apply to the case $\text{(STD)}$ now, when nothing is known about the sign of the smallest eigenvalue $\mu_n$ of $A$. In their most general result ([Gould and Simoncini, 2009, Proposition 2.2]), the lower bound, $\mu_n$, in $I^+$ above is replaced by a number $\gamma$, which is defined as the smallest positive root of the cubic equation

$$\mu^3 - \mu^2(\tilde{\mu} + \mu_n) + \mu(\tilde{\mu} - \mu_n - \|a\|^2 - \sigma_m^2) + \tilde{\mu}\sigma_m^2. \quad (5.9)$$

It is also shown that $\gamma$ cannot be larger than the smallest eigenvalue of the reduced problem, $\tilde{\mu} := \mu_{\min}(Z^TAZ)$, where $Z$ is a matrix whose columns form a basis of $\ker B$. Since $\gamma > 0$ holds, the corresponding condition number estimate

$$\kappa(K) \leq \frac{\max\{\frac{1}{2}(\mu_1 + \sqrt{\mu_1^2 + 4\sigma_1^2}), -\frac{1}{2}(\mu_n - \sqrt{\mu_n^2 + 4\sigma_n^2})\}}{\min\{\gamma, -\frac{1}{2}(\mu_1 - \sqrt{\mu_1^2 + 4\sigma_1^2})\}} \quad (5.9)$$

will not yield $\infty$ even under our weakest assumption $\text{(STD)}$.

We calculate the extreme eigenvalues and singular values and the corresponding estimate for the condition number in the next section, using as an example a discretized optimal control problem.

§ 5.4 An Optimal Control Example

We consider the following standard test example in optimal control with $\nu > 0$:

$$\begin{align*}
\text{Minimize} & \quad \frac{1}{2}\|y - y_0\|^2_{L^2(\Omega_0)} + \frac{\nu}{2}\|u\|^2_{L^2(\Omega)} \\
\text{s.t.} & \quad (\nabla y, \nabla v)_{L^2(\Omega)} = (u, v)_{L^2(\Omega)} \quad \text{for all } v \in H^1_0(\Omega).
\end{align*} \quad (5.10)$$

The domain in this example is $\Omega = (0, 1)^2 \subset \mathbb{R}^2$. Note that the state is observed only on a subset, $\Omega_0 = (0.5, 1) \times (0, 1)$.

We discretized the problem using the standard finite element spaces

- $Y_h$ consisting of piecewise linear $(P_1)$, continuous functions,
- $U_h$ consisting of piecewise constant $(P_0)$, discontinuous functions.

The standard basis functions are denoted by $\{\varphi_i\} \subset Y_h$ and $\{\psi_i\} \subset U_h$. The discrete problem is a quadratic programming (QP) problem whose necessary (and sufficient) optimality conditions are given by

$$\begin{pmatrix} A & B \\ B & 0 \end{pmatrix} \begin{pmatrix} x \\ p \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix} \quad (5.11)$$
with building blocks
\[
A = \begin{pmatrix} M^{\chi_{\Omega_0}} & 0 \\ 0 & M_{uu} \end{pmatrix}, \quad B = \begin{pmatrix} K & -M_{yu} \end{pmatrix}.
\]

The individual blocks are

- \(M^{\chi_{\Omega_0}}\) state space mass matrix with the characteristic function of \(\Omega_0\) as a weight \(\chi_{\Omega_0}\phi_j, \phi_i)_{L^2(\Omega)}\),
- \(M_{uu}\) control space mass matrix \((\psi_j, \psi_i)_{L^2(\Omega)}\),
- \(K\) stiffness matrix \((\nabla \phi_j, \nabla \phi_i)_{L^2(\Omega)}\),
- \(M_{yu}\) mixed state/control space mass matrix \((\phi_j, \psi_i)\).

We computed the spectral estimates according to [Gould and Simoncini, 2009, Proposition 2.2] on a sequence of uniformly refined meshes for the parameter \(\nu = 1\). The results are shown in Table 5.1.

<table>
<thead>
<tr>
<th>level</th>
<th>(\mu_1(A))</th>
<th>(\mu_n(A))</th>
<th>(\tilde{\mu}(A))</th>
<th>(\sigma_1(B))</th>
<th>(\sigma_n(B))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.97e-02</td>
<td>-8.27e-24</td>
<td>3.03e-02</td>
<td>7.08e+00</td>
<td>1.55e-01</td>
</tr>
<tr>
<td>2</td>
<td>1.39e-02</td>
<td>-9.93e-24</td>
<td>7.68e-03</td>
<td>7.73e+00</td>
<td>4.58e-02</td>
</tr>
<tr>
<td>3</td>
<td>3.79e-03</td>
<td>-3.31e-24</td>
<td>1.93e-03</td>
<td>7.93e+00</td>
<td>1.25e-02</td>
</tr>
<tr>
<td>4</td>
<td>9.69e-04</td>
<td>-4.96e-24</td>
<td>4.83e-04</td>
<td>7.98e+00</td>
<td>3.26e-03</td>
</tr>
<tr>
<td>level</td>
<td>([I^-] \quad [I^+])</td>
<td>([I^-] \quad [I^+])</td>
<td>(\kappa(K))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>-----------------</td>
<td>-----------------</td>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>-7.08e+00</td>
<td>-1.36e-01</td>
<td>2.84e-02</td>
<td>7.09e+00</td>
<td>2.50e+02</td>
</tr>
<tr>
<td>2</td>
<td>-7.73e+00</td>
<td>-3.94e-02</td>
<td>7.02e-03</td>
<td>7.74e+00</td>
<td>1.10e+03</td>
</tr>
<tr>
<td>3</td>
<td>-7.93e+00</td>
<td>-1.07e-02</td>
<td>1.76e-03</td>
<td>7.93e+00</td>
<td>4.50e+03</td>
</tr>
<tr>
<td>4</td>
<td>-7.98e+00</td>
<td>-2.81e-03</td>
<td>4.43e-04</td>
<td>7.98e+00</td>
<td>1.80e+04</td>
</tr>
</tbody>
</table>

Table 5.1. Spectral bounds (upper row) for matrices \(A\) and \(B\) in problem (5.10) on various grid levels, with respect to the standard inner product. Computed eigenvalue bounds (lower row) based on [Gould and Simoncini, 2009, Proposition 2.2] and corresponding condition number estimates.

We can make the following observations:

(a) The \(A\) block is only positive semidefinite. It has a number of zero eigenvalues, \(\mu_k(A) = \mu_{k+1}(A) = \cdots = \mu_n(A) = 0\), which is due to the observation of the state being restricted to the subdomain \(\Omega_0\) in the objective of (5.10).

(b) Nevertheless, the \(A\) block is positive definite on \(\ker B\), as can be seen from \(\tilde{\mu}(A)\) (the smallest eigenvalue of the restriction of \(A\) to \(\ker B\)) remaining positive. This is in accordance with our expectation, since assumption (STD) is satisfied for our problem (5.10).

(c) The largest eigenvalue \(\mu_1(A)\) of the \(A\) block decreases like \(\sim h^2\), which is in accordance with known results for mass matrices in 2D.

(d) The largest singular value \(\sigma_1(B)\), that is the norm of \(B\), is almost constant. This is due to the fact that \(\sigma_1(B)\) is dominated by the largest eigenvalue of the stiffness matrix \(K\), which is bounded above by a constant, independent of the mesh level.
(e) The smallest singular value $\sigma_m(B)$ decreases like $\sim h^2$.

All in all, we conclude that the upper bound of $\mathcal{I}^-$, i.e., $\frac{1}{2}(\mu_1 - \sqrt{\mu_1^2 + 4\sigma_m^2})$, approaches zero from below with a rate of $\sim h^2$. At the same time, the lower bound of $\mathcal{I}^+$, i.e., $\gamma$, which is bounded above by $\tilde{\mu}$, approaches zero from above with the same rate. Consequently, the condition number estimate $\kappa(\mathcal{K})$ shown in the last column, is of order $\sim h^{-2}$, which can also be confirmed by direct verification. The same rate for $\kappa(\mathcal{K})$ also holds in the 3D case. The need to use preconditioned iterative solvers (§ 6) becomes evident at this point.

On second thought, however, one thing is curious. We noticed earlier that the spectral properties of problem (5.10) should actually be described in the four numbers $\|a\|, \alpha_0, \|b\|$ and $k_0$, which are mesh independent. Indeed, it is an easy exercise (compare [Herzog and Sachs, 2010, Lemma 3.1]) to show that for our problem

$$\|a\| = \max(1, \nu), \quad \alpha_0 = \nu/(4c_P), \quad \|b\| = 2, \quad k_0 = (1/c_P)^2$$

hold, where $c_P$ is the Poincaré constant $\|y\|_{H^1(\Omega)} \leq c_P \|y\|_{H^1(\Omega)}$.

A first attempt of an explanation might be that the discretization alters these constants. Well, this is indeed a concern. Since we are using conforming discretizations $Y_h \subset Y = H^1_0(\Omega)$ and $U_h \subset U = L^2(\Omega)$, it is easy to see that the constants $\|a\|$ and $\|b\|$ must also be valid at all discrete levels, independent of the mesh size. But this is not the case for $\alpha_0$ and $k_0$, since in general

$$\{x_h \in X_h : b(x_h, p_h) = 0 \text{ for all } p_h \in Q_h \} \not\subset \ker B$$

and, given $p_h \in Q_h$,

the sup-generating element $x \in X$ of $\frac{b(x, p_h)}{\|x\|_X \|p_h\|_Q}$ may not belong to $X_h$.

The constants valid on the discrete level indeed depend on the choice of the spaces $X_h = Y_h \times U_h$ and $Q_h = Y_h$. With our choice above, however, we do satisfy the discrete stability condition, which means that we may have to live with smaller but mesh independent constants $0 < \alpha'_0 \leq \alpha_0$ and $0 < k'_0 \leq k_0$.

Therefore, the reason for the condition number $\kappa(\mathcal{K})$ going to infinity with rate $\sim h^{-2}$ must lie elsewhere. To make the story short, in our computations above we have (implicitly) equipped the spaces $X_h$ and $Q_h$ with the scalar products of $\mathbb{R}^n$ and $\mathbb{R}^m$. However, this does not make a lot of sense for vectors which actually represent elements of function spaces. Therefore, we should have equipped the finite dimensional spaces with appropriate scalar products, so that the norms of the coefficient vectors reflect the norms of the functions represented by them. For the state and adjoint variables $y$ and $u$, appropriate scalar products are induced by the matrices $(K + M_{yy})$, while for the control variable we take $M_{uu}$. With these scalar products, our discrete coefficient vectors have norms which correctly reflect the $H^1(\Omega)$ or $L^2(\Omega)$ norms of the functions they represent.

How can we re-compute the spectral parameters of our matrices $A$ and $B$ using this knowledge? Changing the scalar products amounts to using the algorithms for generalized eigenvalue and singular value computation. To illustrate this point, $\mu$ is a generalized eigenvalue of $A$ with respect to the scalar product mentioned above if

$$\begin{pmatrix} M_{yy}^{\chi_0} & 0 \\ 0 & M_{uu} \end{pmatrix} \begin{pmatrix} y \\ u \end{pmatrix} = \mu \begin{pmatrix} K + M_{yy} & 0 \\ 0 & M_{uu} \end{pmatrix} \begin{pmatrix} y \\ u \end{pmatrix}$$
§ 6 Preconditioning KKT Systems

Literature: Benzi et al. [2005], Stoll and Wathen [2008]

The following quotes are taken from Barrett et al. [1994] and Benzi et al. [2005]:

- "The convergence rate of iterative methods depends on spectral properties of the coefficient matrix. Hence one may attempt to transform the linear system into one that is equivalent in the sense that it has the same solution, but that has more favorable spectral properties. A preconditioner is a matrix that effects such a transformation."

- "For saddle point problems, the construction of high-quality preconditioners necessitates exploiting the block structure of the problem, together with detailed knowledge about the origin and structure of the various blocks. Because the latter varies greatly from application to application, there is no such thing as the 'best' preconditioner for saddle point problems."

Using (left) preconditioning, we convert a saddle point problem of the form (5.3) into the equivalent system

\[
\hat{K}^{-1} \begin{pmatrix} A & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} x \\ p \end{pmatrix} = \hat{K}^{-1} \begin{pmatrix} f \\ g \end{pmatrix}
\]

(6.1)

with the purpose that the preconditioned matrix has better spectral properties. Since \( \hat{K}^{-1} \) has to be applied to a given vector during the iterative solution process, i.e., linear systems with \( \hat{K} \) have to be solved, the computational costs for this has to be balanced with the improvement in convergence achieved by preconditioning. Clearly, the 'best' preconditioner from the convergence point of view of the matrix itself, but solving with \( \hat{K} \) is then equivalent to solving the original system.

Table 5.2. Same information as in Table 5.1, but with respect to appropriate scalar products, i.e., using generalized eigenvalue and singular value computations.
Chapter 2. Preconditioning in PDE-Constrained Optimization

There are a number of classes of preconditioners available in the literature which respect the block structure and properties of saddle point problems. We refer to [Benzi et al., 2005, Section 10] for an overview. Many of the well established preconditioners were designed with applications other than optimal control in mind. A prominent example is the Stokes problem

\[
\begin{pmatrix}
-\mu \triangle & \nabla \\
-\text{div} & 0
\end{pmatrix}
\begin{pmatrix}
u \\
p
\end{pmatrix}
= \begin{pmatrix}
f \\
0
\end{pmatrix}
\]

with viscosity \(\mu > 0\).

A typical Hilbert space setting for this problem is \(X = H_0^1(\Omega) \times H_0^1(\Omega)\) and \(Q = L^2_0(\Omega) \times L^2_0(\Omega)\),

\[
a(u, v) = \mu (\nabla u, \nabla v)_{L^2(\Omega)^d \times d}, \quad b(u, p) = -(\text{div} u, p)_{L^2(\Omega)}.
\]

It is easy to see from here that the Stokes problem verifies assumption (STD++). By contrast, we saw in § 5.4 that optimality systems for optimal control problems typically satisfy only the much weaker condition (STD). Therefore, most of the preconditioners for the Stokes problem cannot be directly applied to KKT systems in optimal control.

In the remainder of this section, we briefly touch upon several classes of preconditioners which make do with the assumption of (STD). We use again the KKT conditions of our model problem (5.10) as an illustrative example. In continuous form, this system reads

\[
K \begin{pmatrix} y \\ u \\ p \end{pmatrix} = \begin{pmatrix} \chi_{\Omega_0} & 0 & -\triangle \\ 0 & \nu I & -I \\ -\triangle & -I & 0 \end{pmatrix} \begin{pmatrix} y \\ u \\ p \end{pmatrix} = \begin{pmatrix} \chi_{\Omega_0} y_{\Omega} \\ 0 \\ 0 \end{pmatrix}
\]

(6.2)

where \(-\triangle\) stands for the negative Laplacian with homogeneous Dirichlet boundary data.

§ 6.1 Early Approaches

Literature: Battermann and Heinkenschloss [1998], Battermann and Sachs [2001], Haber and Ascher [2000], Biros and Ghattas [2005]

The main idea behind early approaches in preconditioning KKT systems from PDE-constrained optimization problems was to exploit available preconditioners for the forward operator, i.e., the operator governing the state equation \((-\triangle\) in our example). This led to preconditioners of the class

\[
\widehat{K} = \begin{pmatrix} 0 & 0 & -\widehat{\triangle} \\ 0 & \widehat{H} & -I \\ -\widehat{\triangle} & -I & 0 \end{pmatrix}
\]

where \(H\) stands for the reduced Hessian matrix \(\nabla^2 \tilde{f}(\cdot)\), or

\[
H = \left(\triangle^{-1}\right)^* \begin{pmatrix} \chi_{\Omega_0} & 0 \\ 0 & \nu I \end{pmatrix} \left(\triangle^{-1}\right)
\]

and where the general symbol \(\widehat{M}\) represents a preconditioner for the matrix or operator \(M\).

\(L_0^2(\Omega)\) denotes the space of \(L^2\) functions on \(\Omega\) with zero average. It is needed to normalize the pressure, which is determined only up to a constant.
§ 6. Preconditioning KKT Systems

Since \( \hat{\mathcal{K}} \) is equivalent to a lower triangular matrix, every solve with \( \hat{\mathcal{K}} \) requires two solves with the preconditioners \(-\hat{\Delta}\) and one with the approximate reduced Hessian \( \hat{H} \). Often \( \hat{H} = \nu I \) was used, because the true reduced Hessian is just a compact perturbation of this. Since both \( \mathcal{K} \) and \( \hat{\mathcal{K}} \) are indefinite operators (indefinite matrices after discretization), general purpose solvers like QMR or GMRES were generally used with these preconditioners. (MINRES requires positive definite preconditioners and therefore wasn’t considered.)

§ 6.2 Constraint Preconditioners

**Literature:** Gould et al. [2001], Keller et al. [2000], Dollar and Wathen [2006]

Constraint preconditioners are of the following structure:

\[
\hat{\mathcal{K}} = \begin{pmatrix}
G & B^T \\
B & 0
\end{pmatrix}
\]

with \( G \) symmetric positive definite on \( \ker B \).

They open up the possibility of using the PPCG (preconditioned projected conjugate gradient) method for the solution of

\[
\begin{pmatrix}
A & B \\
B & 0
\end{pmatrix}
\begin{pmatrix}
x \\
p
\end{pmatrix} = \begin{pmatrix}
f \\
g
\end{pmatrix}
\]

in spite of \( \mathcal{K} \)'s indefiniteness. Indeed, the conjugate gradient iteration taking place can be shown to be equivalent to one in the reduced space \( \ker B \), see Gould et al. [2001], provided that the initial iterate is consistent with the equation \( Bx = g \).

It may appear worthwhile at first glance to apply constraint preconditioners to PDE constrained optimization problems. However, there is actually little benefit of preconditioning the benign block \( A \), consisting of mass matrices, while leaving the \( B \) block intact. Indeed, solving systems with \( \hat{\mathcal{K}} \) is as hard as solving systems with \( \mathcal{K} \) since the 'difficult' operator is our PDE operator \((-\Delta \text{ in the example})\), which is not preconditioned at all.

The simple trick of swapping rows and columns in (6.2) does not help either because the only useful constellation would be

\[
\begin{pmatrix}
\chi \Omega & -\Delta & 0 \\
-\Delta & 0 & -I \\
0 & -I & \nu I
\end{pmatrix}
\]

which is not of the form (5.1).

We return later however, to a constraint preconditioner as a building block of another preconditioner to cope with issues arising due to control inequality constraints, see § 6.4.3.

§ 6.3 Preconditioned Conjugate Gradients

**Literature:** Schöberl and Zulehner [2007], Herzog and Sachs [2010]

The conjugate gradient method is known to be a reliable iterative solver only for symmetric and positive definite matrices. Since saddle point matrices are indefinite (§ 5.3), the possibility of solving them via CG seems to be ruled out. In principle, instead of solving \( \mathcal{K} \left( \begin{pmatrix} \tilde{x} \\ \tilde{q} \end{pmatrix} \right) = \left( \begin{pmatrix} \tilde{f} \\ \tilde{g} \end{pmatrix} \right) \), one could solve \( \mathcal{K}^T \mathcal{K} \left( \begin{pmatrix} \tilde{x} \\ \tilde{q} \end{pmatrix} \right) = \mathcal{K}^T \left( \begin{pmatrix} \tilde{f} \\ \tilde{g} \end{pmatrix} \right) \) which is
Chapter 2. Preconditioning in PDE-Constrained Optimization

governed by a symmetric and positive definite matrix. However, this is not recommended due to the now squared condition number.

In their seminal paper Bramble and Pasciak [1988], to everybody’s surprise, the authors found a way of applying the preconditioned conjugate gradient iteration to saddle-point problems. This became possible through a clever combination of a preconditioner and a non-standard scalar product \( D \), which render the preconditioned matrix self-adjoint and positive definite with respect to \( D \). The conjugate gradient method with this non-standard inner product can thus be applied to the preconditioned matrix. Ever since, the Bramble-Pasciak idea and extensions thereof have risen to popular solution methods for saddle point problems arising mainly from the discretization of PDEs. It seems that this technique was long underestimated if not unknown in the optimization and optimal control communities.

We say that a matrix \( A \) is self-adjoint w.r.t. the scalar product \( D \) if

\[
(x, Ay)_D = (Ax, y)_D \quad \text{for all } x, y
\]

holds, i.e., if \( A = D^{-1}A^\top D \). We say that it is positive definite w.r.t. \( D \) if

\[
(x, Ax)_D \geq c \|x\|_D^2 \quad \text{for all } x,
\]

with a constant \( c > 0 \). Stoll and Wathen [2008] have more on self-adjointness in general inner products, in particular for saddle point problems.

In 2007, Schöberl and Zulehner [2007] gave a thorough analysis and systematic construction of a class of symmetric indefinite preconditioners, together with suitable scalar products, which effect the desired magic. The preconditioners are of the form

\[
\hat{K} = \begin{pmatrix}
\hat{A} & B^\top \\
B & B\hat{A}^{-1}B^\top - \hat{S}
\end{pmatrix} = \begin{pmatrix}
I & 0 \\
B\hat{A}^{-1} & I
\end{pmatrix} \begin{pmatrix}
\hat{A} & B^\top \\
0 & -\hat{S}
\end{pmatrix},
\]

where \( \hat{A} \) and \( \hat{S} \) are symmetric and nonsingular matrices that we define below. This factorization reveals that every application of the preconditioner requires two solves with \( \hat{A} \) and one solve with \( \hat{S} \). The interesting fact of the matter is that suitable preconditioner building blocks are very easy to construct. One may simply take properly scaled preconditioners for the scalar product matrices (let’s call them \( \mathcal{X} \) and \( \mathcal{Q} \)) in the spaces \( \mathcal{X} \) and \( \mathcal{Q} \):

\[
\hat{A} = \frac{1}{\sigma} \hat{\mathcal{X}}, \quad \hat{S} = \frac{\sigma}{\tau} \hat{\mathcal{Q}},
\]

where \( \hat{\mathcal{X}} \) and \( \hat{\mathcal{Q}} \) are suitable approximations to \( \mathcal{X} \) and \( \mathcal{Q} \), respectively.

This finding is of great practical value because excellent preconditioners for the scalar product matrices are available ‘off-the-shelf’ for very many combinations of spaces and finite element discretizations. Here are some examples:

- For the space \( X = H^1(\Omega) \), the standard scalar product matrix is the stiffness matrix associated with the weak form of the problem

\[
-\Delta y + y = 0 \quad \text{in } \Omega, \quad \partial_n y = 0 \quad \text{on } \partial \Omega.
\]

For the space \( X = H^1_0(\Omega) \), an alternative scalar product is the one associated with the weak form of

\[
-\Delta y = 0 \quad \text{in } \Omega, \quad y = 0 \quad \text{on } \partial \Omega.
\]
§ 6. Preconditioning KKT Systems

For both problems, the geometric multigrid approach is one convenient way of obtaining a suitable preconditioner $\tilde{X}$ of optimal complexity for a broad choice of discretizations.

- For the space $X = L^2(\Omega)$, the scalar product matrix is known as the mass matrix in the finite element context. Due to its mesh independent condition number, it can be preconditioned with very little effort, e.g., by diagonally preconditioned conjugate gradients, symmetric Gauss-Seidel iterations, or Chebyshev semi-iteration (see Wathen and Rees [2008] for the latter).

- For product spaces, one may simply use independent preconditioners for each factor space.

Schöberl and Zulehner [2007] found explicit bounds for $\sigma$ and $\tau$ to make suitable scaling parameters. These bounds are

$$\sigma < \frac{1}{\|a\|} \quad \text{and} \quad \tau > \frac{1}{(1 - q_X)(1 - q_Q)} \frac{1}{k_0^2}, \quad (6.4)$$

and they depend only on two of the four governing constants and also on the quality of the preconditioners, for which we make the non-restrictive assumptions of spectral equivalence

$$(1 - q_X) \tilde{X} \preceq X \preceq \tilde{X}, \quad (1 - q_Q) \tilde{Q} \preceq Q \preceq \tilde{Q}.$$ 

With $\sigma$ and $\tau$ chosen to satisfy (6.4), the matrix

$$D = \tilde{K} - K = \begin{pmatrix} \hat{A} - A & 0 \\ 0 & B\hat{A}^{-1}B^T - \hat{S} \end{pmatrix} \quad (6.5)$$

defines a scalar product, and the preconditioned matrix $\tilde{K}^{-1}K$ is self-adjoint and positive definite w.r.t. $D$. Moreover, the preconditioned condition number can be estimated by

$$\kappa(\tilde{K}^{-1}K) \sim \frac{\|a\|_{a_0}}{k_0} \left( \frac{\|b\|_{b_0}}{k_0} \right)^2. \quad (6.6)$$

Recall that the contraction rate of conjugate gradient iterations is

$$\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}. \quad (6.7)$$

Estimate (6.6) thus shows that the performance of the preconditioned conjugate gradient method will depend exclusively on properties of the infinite dimensional problem and it will not deteriorate with decreasing mesh size (under the condition of stable discretizations, see the footnote).

---

2We recall from Section 5.4 that the coercivity and inf-sup constants $\alpha_0'$ and $k_0'$ observed after discretization could be worse (smaller) than those belonging to the undiscretized problem. The mesh independence of $\alpha_0'$ and $k_0'$ imposes a discrete stability constraint on the choice of the finite element pairs which generate the finite dimensional spaces. We assume here that this condition is satisfied, as was the case in our example in Section 5.4, because this is an issue of discretization, not of preconditioning.
§ 6.4 Implementation Details

**Literature:** Herzog and Sachs [2010]

In this section we show how to implement and apply the aforementioned preconditioner to the following problem, taken from [Herzog and Sachs, 2010, Section 3.1].

\[
\text{Minimize} \quad \frac{1}{2} \| y - y_d \|_{L^2(\Omega)}^2 + \frac{\nu}{2} \| u - u_d \|_{L^2(\Omega)}^2 \\
\text{s.t.} \quad \begin{cases} 
-\Delta y + y = u & \text{in } \Omega \\
\partial_n y = 0 & \text{on } \partial \Omega 
\end{cases}
\quad \text{and } u_a \leq u \leq u_b \text{ a.e. in } \Omega. 
\]

(6.7)

Note that this closely resembles our earlier example, but control constraints are included. We point out that the source code for your own numerical experiments is available from [http://www.tu-chemnitz.de/mathematik/part_dgl/publications.php](http://www.tu-chemnitz.de/mathematik/part_dgl/publications.php).

We already know from § 3.1, Algorithm 8, that every step in the primal-dual active set strategy (or semismooth Newton method) requires the solution of the following system:

\[
\begin{pmatrix}
I & L^* & c \chi_{A_k}^+ & c \chi_{A_k}^- & c \\
\nu I & -I & \chi_{A_k} & \chi_{I_k} & 0 \\
L & -I & \chi_{A_k} & \chi_{I_k} & c \chi_{A_k}^+ u_b + c \chi_{A_k}^- u_a
\end{pmatrix}
\begin{pmatrix}
y_{k+1} \\
u_{k+1} \\
p_{k+1} \\
\xi_{k+1}
\end{pmatrix}
= \begin{pmatrix}
y_d \\
u_d \\
0 \\
\end{pmatrix},
\]

(6.8)

where \(\chi_{A_k^+}, \chi_{A_k^-}\) and \(\chi_{A_k}\) denote the characteristic functions of \(A_k^+, A_k^-\) and \(A_k = A_k^+ \cup A_k^-\), respectively, and \(L\) represents the differential operator of the PDE constraint in (6.7). In our present example, we have \(L = -\Delta + I\) with homogeneous Neumann boundary conditions in weak form, considered as an operator from \(H^1(\Omega)\) into \(H^1(\Omega)^*\).

System (6.8) changes from iteration to iteration due to changes in the active sets. Since, however, we focus here on the efficient solution of individual Newton steps, we drop the iteration index from now on. From (6.8) one infers \(\xi_{I_k} = 0\) (the restriction of \(\xi\) to the inactive set \(I\)), and we may eliminate this variable from the problem. The Newton system then attains an equivalent symmetric saddle point form:

\[
\begin{pmatrix}
I & L^* & \chi_{A_k} & 0 & 0 \\
\nu I & -I & \chi_{A_k} & \chi_{I_k} & 0 \\
L & -I & \chi_{A_k} & \chi_{I_k} & c \chi_{A_k}^+ u_b + \chi_{A_k}^- u_a
\end{pmatrix}
\begin{pmatrix}
y \\
u \\
p \\
\xi_{A}\n\end{pmatrix}
= \begin{pmatrix}
y_d \\
u_d \\
0 \\
\end{pmatrix},
\]

(6.9)

which fits into our framework with the following identifications

\[x = (y, u) \in X = H^1(\Omega) \times L^2(\Omega)\]
\[q = (p, \xi_{A}) \in Q = H^1(\Omega) \times L^2(\Omega)\]

and bilinear forms

\[a((y, u), (z, v)) := (y, z)_{L^2(\Omega)} + \nu (u, v)_{L^2(\Omega)}\]
\[b((y, u), (p, \xi_{A})) := (y, p)_{H^1(\Omega)} - (u, p)_{L^2(\Omega)} + (u, \xi_{A})_{L^2(\Omega)}.\]
It is an easy exercise (compare again [Herzog and Sachs, 2010, Lemma 3.1]) to show that our problem satisfies assumptions (STD) with constants

\[ \|a\| = \max\{1, \nu\}, \quad \alpha_0 = \nu/2, \quad \|b\| = 2, \quad k_0 = 1/2, \]

independent of the active sets.

When discretizing the problem with standard finite elements, say, piecewise linear continuous functions for both, the state and control variables, we get a discrete variant of equation (6.10b). As in the continuous setting, we infer \( \vec{\mu} = \vec{0} \) and eliminate this variable to obtain

\[
\begin{pmatrix}
M_h & L_h^T & \cdot \\
\nu M_h & -M_h & P_A^T \\
L_h & -M_h & \cdot \\
P_A & \cdot & \cdot \\
\end{pmatrix}
\begin{pmatrix}
\vec{y} \\
\vec{u} \\
\vec{p} \\
\vec{\mu}_A \\
\end{pmatrix}
= \begin{pmatrix}
M_h \vec{y}_d \\
M_h \vec{u}_d \\
0 \\
P_A^+ \vec{u}_b + P_A^- \vec{u}_a \\
\end{pmatrix}.
\tag{6.11}
\]

\( P_A \) is a rectangular matrix consisting of those rows of the diagonal 0-1-matrix \( \chi_A \) which belong to the active indices, and similarly for \( P_A^\pm \).

Some comments concerning the discrete system (6.11) are in order. The variable \( \vec{\mu} \) is the Lagrange multiplier associated to the discrete constraint \( \vec{u}_a \leq \vec{u} \leq \vec{u}_b \) imposed on the coefficient vector, and the relations \( \vec{\mu} = P_A^\top \vec{\mu}_A \) and \( \vec{\mu}_A = P_A \vec{\mu} \) hold. If we set \( \vec{\xi} = M_h^{-1} \vec{\mu} \), then \( \vec{\xi} \) is the coordinate vector of a function in \( L^2(\Omega) \) which approximates the multiplier \( \xi \) in the continuous system. This observation must be reflected by the choice of norms on the discrete level, see (6.12) below.

The settings

\[
\vec{x} = (\vec{y}, \vec{u}) \in X_h = \mathbb{R}^n \times \mathbb{R}^n \\
\vec{q} = (\vec{p}, \vec{\mu}_A) \in Q_h = \mathbb{R}^n \times \mathbb{R}^{n_A}
\]

and bilinear forms

\[
a_h((\vec{y}, \vec{u}), (\vec{z}, \vec{v})) := \vec{z}^\top M_h \vec{y} + \nu \vec{v}^\top M_h \vec{u} \\
b_h((\vec{y}, \vec{u}), (\vec{p}, \vec{\mu}_A)) := \vec{p}^\top L_h \vec{y} - \vec{p}^\top M_h \vec{u} + \vec{\mu}_A^\top (P_A \vec{u})
\]

define the setting for the discrete problem. As mentioned above, care has to be taken in choosing an appropriate norm for the discrete multiplier \( \vec{\mu}_A \). We use scalar products in the spaces \( X_h \) and \( Q_h \) represented by the following matrices:

\[
X = \begin{pmatrix} K_h & M_h \end{pmatrix} \quad \text{and} \quad Q = \begin{pmatrix} K_h & P_A^\top M_h^{-1} P_A \end{pmatrix}
\tag{6.12}
\]

with \( M_h \) and \( K_h \) defined as

\[
M_h = (\varphi_i, \varphi_j)_{L^2(\Omega)} \quad \text{mass matrix} \tag{6.13a}
\]

\[
L_h = K_h = (\nabla \varphi_i, \nabla \varphi_j)_{L^2(\Omega)} + (\varphi_i, \varphi_j)_{L^2(\Omega)} \quad \text{stiffness matrix}, \tag{6.13b}
\]

where \( \{\varphi_i\}_{i=1}^n \) is the standard basis of piecewise linear continuous basis functions.

We recall that the preconditioner has the form

\[
\hat{K} = \begin{pmatrix} I & 0 \\
AB^{-1} & I \end{pmatrix} \begin{pmatrix} A & B^\top \\
0 & -S \end{pmatrix}
\]
and that we solely require correctly scaled preconditioners for the scalar product matrices defined in (6.12). We use
\[
\hat{A} = \frac{1}{\sigma} \begin{pmatrix} \hat{K}_h \\ \hat{M}_h \end{pmatrix} \quad \text{and} \quad \hat{S} = \frac{\sigma}{\tau} \begin{pmatrix} \hat{K}_h \\ \hat{P}_A \hat{M}_h^{-1} \hat{P}_A \end{pmatrix}
\]  
(6.14)
and refer to Herzog and Sachs [2010] for details on the automatic choice of scaling parameters \(\sigma\) and \(\tau\).

The remainder of these notes is devoted to explaining some more implementation details to convince the reader that the proposed preconditioner is actually easy to apply.

§ 6.4.1 PCG Solver

We begin by showing the \texttt{pcg} method in Algorithm 10. The reader will verify that this is actually a standard implementation of the conjugate gradient method (see, e.g., Saad [2003] or Shewchuk [1994]), applied to the preconditioned matrix \(\hat{K}^{-1} K\), and with respect to the \(D\) scalar product, whose implementation is described in the next subsection.

§ 6.4.2 Scalar Product

It is to be noted that the scalar product \(\langle (\vec{t}_x, \vec{t}_q), (\vec{r}_x, \vec{r}_q) \rangle_D\), with \(D\) given in (6.5), cannot usually be evaluated for arbitrary pairs of vectors. The reason is that matrix-vector products with \(\hat{A}\) and \(\hat{S}\) are usually not available (in contrast to products with \(\hat{A}^{-1}\) and \(\hat{S}^{-1}\), which are realized by applications of the preconditioners; think of multigrid). And thus \(\hat{A} \vec{r}_x\) and \(\hat{S} \vec{r}_q\) cannot be evaluated unless \((\vec{r}_x, \vec{r}_q) = \hat{K}^{-1}(\vec{s}_x, \vec{s}_q)\) holds. That is, the evaluation of the scalar product is implementable if one of the factors is known to be the preconditioner applied to another pair of vectors. (Fortunately, this is the case during the \texttt{pcg} iterations.) We denote this situation by \(\langle (\vec{t}_x, \vec{t}_q), (\vec{r}_x, \vec{r}_q); (\vec{s}_x, \vec{s}_q) \rangle_D\). Under these circumstances the scalar product can be evaluated as follows:
\[
\langle (\vec{t}_x, \vec{t}_q), (\vec{r}_x, \vec{r}_q); (\vec{s}_x, \vec{s}_q) \rangle_D = \langle \vec{t}_x \rangle^\top (\vec{s}_x - B \vec{r}_q - A \vec{r}_x) + \langle \vec{t}_q \rangle^\top (\vec{s}_q - B \vec{r}_x).
\]
As a consequence, it is necessary to maintain the relations \(\vec{r} = \hat{K}^{-1} \vec{s}\) and \(\vec{q} = \hat{K}^{-1} \vec{e}\) throughout the iteration, which requires the storage of one extra vector compared to common conjugate gradient implementations with respect to the standard inner product.

§ 6.4.3 Preconditioner

Algorithm 11 below describes in detail the application of the preconditioner (6.14) in terms of
\[
\hat{K} \begin{pmatrix} \vec{r}_x \\ \vec{r}_q \end{pmatrix} = \begin{pmatrix} \vec{s}_x \\ \vec{s}_q \end{pmatrix}
\]  
(6.15)
where \(\vec{r}_x = (\vec{r}_y, \vec{r}_u)\), \(\vec{r}_q = (\vec{r}_p, \vec{r}_{p, A})\) and \(\vec{s}_x = (\vec{s}_y, \vec{s}_u)\), \(\vec{s}_q = (\vec{s}_p, \vec{s}_{p, A})\) hold. The building blocks of the preconditioner are as follows:

- \((\hat{K}_h)^{-1} \vec{b}\) is realized by one \texttt{multigrid} \texttt{V-cycle} applied to the linear system with the scalar product matrix \(K_h\) (representing the discrete \(H^1(\Omega)\) scalar product) and right hand side \(\vec{b}\). A number of \(\nu_{GS}\) forward and reverse
Algorithm 10 Conjugate gradient method for $\tilde{K}^{-1}K$ w.r.t. to scalar product $\mathcal{D}$.

\textbf{Input:} right hand side $(\tilde{b}_\ell, b_\ell)$ and initial iterate $(\tilde{x}_\ell, x_\ell)$

\textbf{Output:} solution $(\bar{x}_\ell, \bar{\ell})$ of $K(\tilde{x}_\ell, \tilde{x}_\ell) = (\tilde{b}_\ell, b_\ell)$

1: Set $n := 0$ and compute initial residual
   \[ (s_x, s_q) := (\tilde{b}_x - A \tilde{x}_x - B^T \tilde{x}_q) \quad \text{and} \quad (d_x, d_q) := (\tilde{r}_x, \tilde{r}_q) = \tilde{K}^{-1} (s_x, s_q) \]

2: Set $\delta_0 := \delta^+ := ((\tilde{r}_x, \tilde{r}_q), (\tilde{r}_x, \tilde{r}_q); (s_x, s_q)) \mathcal{D}$

3: while $n < n_{\text{max}}$ and $\delta^+ > \varepsilon_{\text{rel}} \delta_0$ and $\delta^+ > \varepsilon_{\text{abs}}$ do

4: Set
   \[ (\bar{e}_x, \bar{e}_q) := (A d_x + B^T d_q) \quad \text{and} \quad (\bar{q}_x, \bar{q}_q) := \tilde{K}^{-1} (\bar{e}_x, \bar{e}_q) \]

5: Set $\alpha := \delta^+ / ((\bar{d}_x, \bar{d}_q), (\bar{q}_x, \bar{q}_q); (\bar{e}_x, \bar{e}_q)) \mathcal{D}$

6: Update the solution
   \[ (\bar{x}_x, \bar{x}_q) := (\bar{x}_x, \bar{x}_q) + \alpha (\bar{d}_x, \bar{d}_q) \]

7: Update the residual
   \[ (\bar{r}_x, \bar{r}_q) := (\bar{r}_x, \bar{r}_q) - \alpha (\bar{q}_x, \bar{q}_q) \quad \text{and} \quad (\bar{s}_x, \bar{s}_q) := (\bar{s}_x, \bar{s}_q) - \alpha (\bar{e}_x, \bar{e}_q) \]

8: Set $\delta := \delta^+$ and $\delta^+ := ((\bar{r}_x, \bar{r}_q), (\bar{r}_x, \bar{r}_q); (\bar{s}_x, \bar{s}_q)) \mathcal{D}$

9: Set $\beta := \delta^+ / \delta$

10: Update the search direction
    \[ (\bar{d}_x, \bar{d}_q) := (\bar{r}_x, \bar{r}_q) + \beta (\bar{d}_x, \bar{d}_q) \]

11: Set $n := n + 1$

12: end while

13: return $(\bar{x}_x, \bar{x}_q)$

Gauss-Seidel smoothing steps are used, starting from an initial guess of $\tilde{0}$. In Algorithm 11, the evaluation of $(\tilde{K}_h)^{-1}\tilde{b}$ is denoted by multigrid$(\tilde{b})$.

- $(\tilde{M}_h)^{-1}\tilde{b}$ corresponds to $\nu_{\text{SGS}}$ symmetric Gauss-Seidel steps for the mass matrix $M_h$ (representing the scalar product in $L^2(\Omega)$) with right hand side $\tilde{b}$, and with an initial guess $\tilde{0}$. This is denoted by $\text{SGS}(\tilde{b})$ in Algorithm 11.

- Note that the evaluation of $\tilde{\mu}_A = (P_A M_h^{-1} P_A^T)^{-1} \tilde{b}_A$ is equivalent to solving the linear system
  \[ \begin{pmatrix} M_h & P_A^T \\ P_A & 0 \end{pmatrix} \begin{pmatrix} \tilde{r} \\ \tilde{\mu}_A \end{pmatrix} = - \begin{pmatrix} \tilde{0} \\ \tilde{b}_A \end{pmatrix}, \]  
  (6.16)

where $\tilde{r}$ is a dummy variable. System (6.16) can be solved efficiently by the preconditioned projected conjugate gradient (ppeg) method (see § 6.2) in
the standard scalar product, where
\[
\begin{pmatrix}
\text{diag}(M_h) & P_A^T \\
P_A & 0
\end{pmatrix}
\]
serves as a (constraint) preconditioner. In Algorithm 11, this corresponds to the call `ppcg(\vec{b}_A, A^+, A^-)`. In practice, we use a tight relative termination tolerance of $10^{-12}$ for the residual in `ppcg`, which took at most 13 steps to converge in all examples.\(^4\) Note that the projected conjugated gradient method requires an initial iterate consistent with the second equation $P_A \vec{r} = -\vec{b}_A$ in (6.16). Due to the structure of $P_A$, we may simply take $\vec{r}_A = -\vec{b}_A$, $\vec{r}_L = 0$ and $\vec{\mu}_A = 0$ as initial iterate.

**Algorithm 11** Application of the preconditioner according to (6.15)

**Input:** right hand sides $\vec{s}_x = (\vec{s}_y, \vec{s}_u)$ and $\vec{s}_q = (\vec{s}_p, \vec{s}_{\mu_A})$, scaling parameters $\sigma, \tau$, and active sets $A^+, A^-$

**Output:** solution $\vec{r}_x = (\vec{r}_y, \vec{r}_u)$ and $\vec{r}_q = (\vec{r}_p, \vec{r}_{\mu_A})$ of (6.15)

1: $\vec{r}_y' := \text{multigrid}(\sigma \vec{s}_y)$
2: $\vec{r}_u' := \text{SGS}(\sigma \vec{s}_u)$
3: \[
\begin{pmatrix}
\vec{s}_p' \\
\vec{s}_{\mu_A}'
\end{pmatrix} := B \begin{pmatrix}
\vec{r}_y' \\
\vec{r}_u'
\end{pmatrix} - \begin{pmatrix}
\vec{s}_p \\
\vec{s}_{\mu_A}
\end{pmatrix}
\]
4: $\vec{r}_p := \text{multigrid}(\tau \vec{s}_p'/\sigma)$
5: $\vec{r}_{\mu_A} := \text{ppcg}(\tau \vec{s}_{\mu_A}'/\sigma, A^+, A^-)$
6: \[
\begin{pmatrix}
\vec{s}_y' \\
\vec{s}_u'
\end{pmatrix} := \begin{pmatrix}
\vec{s}_y \\
\vec{s}_u
\end{pmatrix} - B \begin{pmatrix}
\vec{r}_p \\
\vec{\mu}_A
\end{pmatrix}
\]
7: $\vec{r}_y := \text{multigrid}(\sigma \vec{s}_y')$
8: $\vec{r}_u := \text{SGS}(\sigma \vec{s}_u')$
9: return $\vec{r}_y, \vec{r}_u, \vec{r}_p, \vec{r}_{\mu_A}$

After this, Herzog and Sachs [2010] continues with the investigation of problems with state constraints, regularized either by the Lavrentiev technique (leading to mixed control-state constraints), or by the Moreau-Yosida penalty approach, see § 3.3.

We stop here with the presentation of some numerical experiments for the control constrained case in 2D and 3D. The setup is described in [Herzog and Sachs, 2010, Example 4.2]. Our conclusion is, not surprisingly, that for moderate discretizations in 2D a sparse direct solver is not easy to beat, while for 3D problems our iterative solver wins immediately.

We point out once again that the source code for this example is available from


\(^4\) The reason for solving (6.16) practically to convergence is that intermediate iterates in conjugate gradient iterations depend nonlinearly on the right hand side, and thus early termination would effectively yield a nonlinear preconditioner $\hat{S}$ not covered by the theory.
Most of it (excluding the standard multigrid routines) is also shown in the appendix with some additional comments.

**Figure 6.1.** The plots show the average solution time per Newton step vs. the dimension of the discretized state space. We compare the pcg method to MATLAB’s sparse direct solver applied to the linearized optimality system (6.9) of problem (6.7) in 2D (left) and 3D (right). The triangle has slope 1 and it visualizes the linear complexity of the proposed pcg solver w.r.t. the number of unknowns.

**Figure 6.2.** The plots show the convergence history of the pcg residual in the 2D (left) and 3D (right) cases on the finest grid, for all Newton steps.
APPENDIX A

Software

§ 7 Source Code with Comments for the Preconditioned CG Code

We show here an actual MATLAB implementation of the preconditioned conjugate gradient algorithm for the solution of the control constrained problem (6.7) in 2D or 3D with the setup described in [Herzog and Sachs, 2010, Example 4.2]. The complete code for this example is available from http://www.tu-chemnitz.de/mathematik/part_dgl/publications.php. It is identical to the code shown below except that the latter was stripped from a number of fprintf commands. We also do not address standard multigrid routines here.

We begin by loading the problem data for the 3D version of problem (6.7). For convenience, the .mat file already contains a hierarchy of all matrices across the grid levels as well as the multigrid transfer operators. With the prepost parameter, we may override the default number of Gauss-Seidel pre- and post-smoothing steps for the multigrid V-cycle if desired. Explore also the other data in the problem structure.

```matlab
% Load the problem data
load Example_CC3D.mat;

problem.prepost = 2;

% Get matrices and sizes
M = problem.fem.M{problem.levels}; % mass matrix
L = problem.fem.L{problem.levels}; % PDE operator matrix
ny = problem.fem.np{problem.levels}; % degrees of freedom for state/control/adjoint state
```

Next we initialize our variables \((y,u,p,\mu)\) and enter the PDAS (semismooth Newton) loop, which was described in § 3.1.

```matlab
% Initialize variables
y = zeros(ny,1); u = zeros(ny,1); % same # of dofs (distributed control)
p = zeros(ny,1); mu = zeros(ny,1);

% Initialize flags and counters
iter = 0; done = 0;

% Do a simple semismooth Newton loop
while (iter < problem.maxSSN & ~done)
    % Determine active and inactive sets
    % -------------------------------------
```
Appendix A. Software

```matlab
Aplus = find(mu + problem.cSSN * (u - problem.ub) > 0);
Aminus = find(mu + problem.cSSN * (u - problem.ua) < 0);
```

Next we set up the linear system and right hand side, see (6.11). In a truly high performance code, the matrices \( A \) and \( B \) would not be formed of course, but matrix-vector products would be used instead.

```matlab
% Solve the Newton system by Bramble-Pasciak like cg
% -----------------------------------------------
% Set up the active set projector
PA = zeros(ny,1);
PA(Aplus) = 1; PA(Aminus) = 1;
PA = spdiags(PA,0,ny,ny);
PA = PA(union(Aplus,Aminus),:);
nA = length(Aplus)+length(Aminus);

% Set up some zero matrices
Z1 = sparse(ny,ny); Z2 = sparse(ny,nA);

% Set up the saddle point blocks
A = [M Z1; Z1 problem.nu*M];
B = [L -M; Z2' PA];

% Set up right hand side
bx = [M*problem.fem.yd; problem.nu*M*problem.fem.ud];
bq = zeros(ny,1);
bq(Aplus) = problem.ub;
bq(Aminus) = problem.ua;
bq = [zeros(ny,1); PA*bq];
```

We are now ready to call the pcg solver. We haven’t mentioned the safeguard strategy yet (see [Herzog and Sachs, 2010, Algorithm 2]) which detects unsuitable values of the scaling parameters and simply tries again with corrected values if they were wrong.

```matlab
% Prepare to call the Bramble-Pasciak cg solver
% with a safeguard strategy, should inappropriate
% scaling be detected.
scaling_rejected = 1;

% Call the solver
% -----------------------------------------------
while (scaling_rejected)
    [x,q,flag,pcgiter] = ...
    bpcg(A,B,[],bx,bq,...
        problem.atolpcg,problem.rtolpcg,problem.maxpcg,
        ...@Khatm1,[y;u],[p;PA*mu],...
        problem,PA,nA,B,Aplus,Aminus);

    if (flag >= 0) % scaling was ok
        scaling_rejected = 0;
    ```
Once the Newton system was successfully solved, we decompose \((x, q) = (y, u, p, \mu)\), check for convergence and finish the semismooth Newton loop.

\[
\begin{align*}
\text{else} & \\
& \text{problem.sigma} = \text{problem.sigma} / \sqrt{2}; \\
& \text{problem.tau} = \text{problem.tau} \times \sqrt{2}; \\
\text{end}
\end{align*}
\]

\[
\begin{align*}
\text{end \%while (scaling_rejected)} \\
\end{align*}
\]

Our work horse within the semismooth Newton loop is the preconditioned conjugate gradient solver (here called \texttt{bpcg} after Bramble and Pasciak), see Algorithm 10. The actual implementation of the preconditioner (Algorithm 11, the call to \texttt{Khatm1}, meaning \(\hat{K}^{-1}\)) is not shown, but it is contained in the available source code. As described in \S\ 6.4.3, it consists of standard components such as a multigrid V-cycle.

\[
\begin{align*}
\text{function \[x,q,flag,iter\] = bpcg(A,B,Bt,bx,bq,atol,rtol,maxit,} \\
\text{Khatm1,x0,q0,\ldots,\text{varargin})} \\
\text{% Initialize iteration counter and variables} \\
& \text{iter} = 0; \\
& \text{x = x0; q = q0;} \\
\text{% Compute the initial residual} \\
& \text{sx} = \text{bx} - \text{A}\times x - \text{B}^\prime \times q; \\
& \text{sq} = \text{bq} - \text{B} \times x; \\
\text{% Compute the preconditioned initial residual} \\
& \text{[rx,rq] = Khatm1(sx,sq,\text{varargin});} \\
\text{% This is also the initial search direction} \\
& \text{dx = rx; dq = rq;}
\end{align*}
\]
% Compute delta = residual norm squared w.r.t. the scalar product D
delta0 = scalar_product(A,B,rx,rq,rx,rq,sx,sq);
delta = delta0;

% Set convergence flag
converged = 0;

% Enter loop
while (iter < maxit & ~converged & delta > 0)
    % Evaluate e = K * d
    ex = A * dx + B' * dq;
eq = B * dx;

    % Compute q = Khat^{-1} * e
    [qx,qq] = Khatm1(ex,eq,varargin);

    % Compute the step length (should be non-negative)
    alpha = delta / scalar_product(A,B,dx,dq,qx,qq,ex,eq);

    % Update the solution
    x = x + alpha * dx;
    q = q + alpha * dq;

    % Update the residual and s to maintain r = Khat^{-1} * s
    % Re-evaluate the residual in every 50th iteration
    if (iter > 0 & mod(iter,50) == 0)
        sx = bx - A * x - B' * q;
sq = bq - B * x;
        [rx,rq] = Khatm1(sx,sq,varargin);
    else
        rx = rx - alpha * qx;
        rq = rq - alpha * qq;
        sx = sx - alpha * ex;
        sq = sq - alpha * eq;
    end

    % Remember old value of delta
deltaold = delta;

    % Compute new value of delta
delta = scalar_product(A,B,rx,rq,rx,rq,sx,sq);

    % Compute new value of beta
    beta = delta / deltaold;

    % Update search direction d
dx = rx + beta * dx;
dq = rq + beta * dq;
§ 7. Source Code with Comments

```matlab
% Increase iteration counteriter = iter + 1;

% Check for convergence
converged = (delta <= atol^2) | (delta <= rtol^2 * delta0);
end %while (iter < maxit & ~converged & delta > 0)

% Set return flag
if (delta <= 0)
    flag = -1; % incorrect scaling of preconditioner
elseif converged
    flag = 0; % converged to desired tolerance
else
    flag = 1; % max # of iterations reached without convergence
end
end % function bpcg

Finally, we show the implementation of the scalar product, which was described in § 6.4.2.

```
Bibliography


Jürgen Sprekels.


