VSB TECHNICAL UNIVERSITY OF OSTRAVA | FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE | DEPARTMENT OF APPLIED MATHEMATICS

Improving Quadratic Programming Algorithms

Vylepšování algoritmů kvadratického programování

Jakub Kružík

Ph.D. Thesis
Supervisor: Doc. Ing. David Horák, Ph.D.
Ostrava, 2024

Abstract

The main objective of this thesis is to present improvements in quadratic programming algorithms. These improvements speed up the solution of quadratic programming problems, with or without constraints, which are key in various fields, including, but not limited to, economics, engineering, and machine learning.

The main improvements are for solving box-constrained quadratic programming problems. The MPRGP (Modified Proportioning with Reduced Gradient Projections) algorithm is analyzed and, based on this analysis, improved. The analysis reveals that the expansion of the active set through the reduced gradient projections is the most expensive part of the algorithm. Our modification of the expansion step, using the projected conjugate gradient, proves significantly superior to the original algorithm in most cases. The presented fallback steps and criteria can be used to guarantee convergence or even ensure that the convergence rate is at least as good as that of the standard MPRGP algorithm. Another presented modification is to use the Spectral Projected Gradient (SPG) method as the expansion step. This proves to be extremely effective in certain cases, but a little less so in others. Numerical experiments showcasing the effectiveness of the proposed methods, as well as a comparison with the SPG method, are presented on a number of benchmarks.

Another improvement of MPRGP is in preconditioning, which is not straightforward to implement when the problem is constrained. Our improvement is to cheaply approximate the preconditioning in face, which must be recomputed every time the active set changes, with a preconditioner that is set up only once. The numerical experiments show speedups between 5.1 and 13.4 compared to the unpreconditioned algorithm. The previous expansion step modification is a key ingredient for an effective preconditioned algorithm. An error analysis comparing the standard and the approximate variant of the preconditioning in face is provided to complement the numerical experiments.

Further improvements include the scalability of FETI (Finite Element Tearing and Interconnecting) domain decomposition methods, which allow us to solve problems with more than one billion unknowns using tens of thousands of computational cores on large supercomputers.

Most of the presented algorithms are implemented in the PERMON library, of which the author of this thesis was the maintainer and the main contributor throughout their doctoral studies. The main aim of PERMON is to provide a scalable framework for the solution of large-scale quadratic problems. Another software contribution was the multilevel deflation preconditioner, PCDEFLATION, in the PETSc library for scientific computing.

Keywords: quadratic programming, optimization, gradient projections, conjugate gradients, preconditioning, deflation, coarse problem, penalty method, MPRGP, SPG, SMALE, FETI, BETI, hybrid FETI, PERMON, PCDEFLATION

Abstrakt

Hlavním cílem této práce je představit vylepšení algoritmů kvadratického programování. Tato vylepšení urychlují řešení problémů kvadratického programování, s omezeními i bez nich, které jsou klíčové v různých oblastech včetně ekonomie, inženýrství a strojového učení.

Hlavní vylepšení jsou zaměřena na řešení problémů kvadratického programování s omezením ve tvaru boxu. Algoritmus MPRGP (Modified Proportioning with Reduced Gradient Projections) je analyzován a na základě této analýzy vylepšen. Analýza ukazuje, že rozšíření aktivní množiny pomocí projekcí redukovaného gradientu je nejdražší částí algoritmu. Naše modifikace kroku rozšíření aktivní množiny pomocí projektovaného konjugovaného gradientu se ukazuje jako výrazně lepší než původní algoritmus ve většině případů. Představené fallback kroky a kritéria mohou být použity k zajištění konvergence po konečném počtu iterací nebo dokonce rychlosti konvergence alespoň stejně dobré jako standardní algoritmus MPRGP. Další představenou modifikací je použití metody Spektrálních Projektovaných Gradientů (SPG) jako kroku rozšíření aktivní množiny. To se ukazuje jako velmi efektivní v určitých případech, ale o něco méně v jiných. Numerické experimenty ukazující účinnost navrhovaných metod a srovnání s metodou SPG jsou prezentovány na řadě benchmarků.

Další vylepšení MPRGP spočívá v předpodmínění, které není snadné implementovat, když má problém omezení. Naše vylepšení spočívá v levné aproximaci předpodmínění v takzvané face, které musí být přepočítáno pokaždé, když se změní aktivní množina, pomocí předpodmínění, které je sestaveno pouze jednou. Numerické experimenty ukazují zrychlení mezi 5.1 a 13.4 ve srovnání s algoritmem bez předpodmínění. Předchozí modifikace kroku rozšíření aktivní množiny je klíčovou ingrediencí pro efektivní algoritmus s předpodmíněním. Numerické experimenty jsou doplněny o analýzu chyby porovnávající standardní a aproximovanou variantu předpodmínění ve face.

Další vylepšení se týkají škálovatelnosti metod doménové dekompozice typu FETI (Finite Element Tearing and Interconnecting), což nám umožňuje řešit problémy s více než jednou miliardou neznámých pomocí desítek tisíc výpočetních jader na velkých superpočítačích.

Většina představených algoritmů je implementována v knihovně PERMON, jejímž správcem a hlavním přispěvatelem byl autor této práce během doktorského studia. Hlavním cílem knihovny PERMON je poskytnout škálovatelné řešení velkých kvadratických problémů. Dalším softwarovým příspěvkem byl víceúrovňový deflační předpodmiňovač PCDEFLATION do knihovny PETSc určené pro vědecké výpočty.

Klíčová slova: kvadratické programování, optimalizace, projekce gradientu, předpodmínění, deflace, hrubý problém, metoda penalty, MPRGP, SPG, SMALE, FETI, BETI, hybrid FETI, PERMON, PCDEFLATION

I would like to thank Assoc. Prof. David Horák, the supervisor of this thesis, for his invaluable help, ideas, advice, and suggestions, which significantly improved this work. My thanks go to the entire PERMON team for all their help. I would also like to thank Prof. Reinhard Nabben, Prof. Jacek Gondzio, and Prof. Ulrich Rüde for hosting me during my research stays in their respective groups. Heartfelt thanks go to my family and friends for their continued support.

I would also like to acknowledge my employers, the Institute of Geonics of the Czech Academy of Sciences and the Department of Applied Mathematics, Faculty of Electrical Engineering and Computer Science at VSB - Technical University of Ostrava, for allowing me to work on this thesis. The research has been supported by a number of projects listed in Appendix B.

Contents

List of Abbreviations and Symbols

Methods

BEM	Boundary Element Method
FEM	Finite Element Method
FETI	Finite Element Tearing and Interconnecting
TFETI	Total Finite Element Tearing and Interconnecting
CG	Conjugate Gradient
DCG	Deflated Conjugate Gradient
PCG	Preconditioned Conjugate Gradient
PDCG	Preconditioned Deflated Conjugate Gradient
MINRES	Minimal Residual
GMRES	Generalized Minimal Residual
MPGP	Modified Proportioning with Gradient Projections
MPRGP	Modified Proportioning with Reduced Gradient Projections
MPPCG	Modified Proportioning with Projected Conjugate Gradient
SPG	Spectral Projected Gradient
MPSPG	Modified Proportioning with Spectral Projected Gradient
MPSPGf	Modified Proportioning with Spectral Projected Gradient with fused matrix-vector multiplication
SMALE	Semimonotonic Augmented Lagrangian

Preconditioners

ICC	Incomplete Cholesky Factorization
SSOR	Symmetric Successive Over-Relaxation

General Abbreviations

BLAS	Basic Linear Algebra Subprograms
CP	Coarse Problem
DOF	Degree of Freedom
KKT	Karush-Kuhn-Tucker conditions
LCQ	Linear Constraint Qualification
LICQ	Linear Independence Constraint Qualification
SCQ	Slater Constraint Qualification

MPI	Message Passing Interface
QP	Quadratic Programming
SPD	Symmetric Positive Definite (matrix)
SPS	Symmetric Positive Semidefinite (matrix)
SVM	Support Vector Machine

Symbols

$\mathbb{R}$, $\mathbb{R}^n$, $\mathbb{R}^{m\times n}$	Set of real numbers, vectors, and matrices
$\mathbb{C}$, $\mathbb{C}^n$, $\mathbb{C}^{m\times n}$	Set of complex numbers, vectors, and matrices
$\mathbb{Z}$	Set of all integers
$\mathbb{N}$	Set of all positive integers
$\boldsymbol{x}^T$	Transpose of a vector or matrix
$\boldsymbol{o}$, $\boldsymbol{O}$	Null vector, null matrix
$\boldsymbol{I}$	Identity matrix
$\boldsymbol{A}^{-1}$	Inverse of a matrix
$\boldsymbol{A}^{+}$	Generalized inverse of a matrix
$\\|\boldsymbol{x}\\|$	Euclidean norm of a vector
$\\|\boldsymbol{A}\\|$	Spectral norm of a matrix (largest singular value)
$\sigma(\boldsymbol{A})$	Spectrum (set of eigenvalues) of a matrix
$\kappa(\boldsymbol{A})$	Condition number of a matrix
$\mathrm{tr}(\boldsymbol{A})$	Trace of a matrix
$[\boldsymbol{x}_k]_i$, $[\boldsymbol{x}_k]_I$	Vector components given by set I or $\{i\}$; brackets are omitted when subscript k is not present
$\boldsymbol{A}_{I,J}$, $[\boldsymbol{A}]_{I,J}$	Submatrix with rows and columns selected by sets I and J, respectively
$\mathrm{Im}\,\boldsymbol{A}$	Image of a matrix
$\mathrm{Ker}\,\boldsymbol{A}$	Null space (kernel) of a matrix
dim	Dimension of a vector space
$\mathrm{diag}(x_1,\ldots,x_n)$	Matrix with $(x_1,\ldots,x_n)$ on its diagonal
span	Linear span
div	Divergence operator
∇	Gradient operator
$P_\Omega(\boldsymbol{x})$	Projection onto set Ω
P_k	Lagrange elements over simplices of order k
Q_1	First-order Lagrange elements over quadrilaterals or hexahedra
C^1	Class of functions with the 1st derivative continuous in its domain
$L^2(\Omega)$	Vector space of the square-integrable functions (with respect to the Lebesgue measure) on a set Ω
$\mathcal{K}_k(\boldsymbol{A},\boldsymbol{g}_0)$	Krylov subspace of dimension k generated by a matrix $\boldsymbol{A}$ and a vector $\boldsymbol{g}_0$

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Quadratic Programming (QP) represents a subclass of optimization problems dealing with the solution of

$$\arg\min_{\boldsymbol{x}} \frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} - \boldsymbol{x}^T\boldsymbol{b} \quad \text{s.t.} \quad \boldsymbol{x} \in \Omega,$$

where, in general, $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is a symmetric matrix[1] called the Hessian matrix, vector $\boldsymbol{b} \in \mathbb{R}^n$ is known as the right-hand side, and Ω is a set of constraints on the solution vector $\boldsymbol{x} \in \mathbb{R}^n$. The minimized quadratic function $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} - \boldsymbol{x}^T\boldsymbol{b}$ is known as the cost function.

This work focuses on developing algorithms and techniques to solve these problems when $\boldsymbol{A}$ is positive semidefinite, and Ω is a closed convex set that contains specific types of constraints, such as linear equality and box constraints. We note that the special case of QP with no constraints, i.e., $\Omega = \mathbb{R}^n$, corresponds to a system of linear equations.

QP problems have become nearly ubiquitous with the advancement of scientific computing. They can be found in fields including economics, engineering, machine learning, and many others. A vast variety of applications in science and industry require the solution of QP subproblems, and the efficiency of these underlying QP problems often drives the efficiency of the entire application. Furthermore, as computers become increasingly powerful, there is a growing interest in and the means to solve more complex problems.

The increased problem complexity arises from various sources, such as the increased complexity of underlying models or the pursuit of higher accuracy in simulations by increasing their resolution. An illustration of the first example is adding fractures to a hydro-mechanical model of a porous medium (e.g., modeling deep geological repositories of radioactive waste [4]). The fractures add non-penetration conditions (linear inequality constraints) to a finite element linear elasticity model (system of linear equations).

In the case of the second example, enhancing the spatial and/or temporal resolution of simulations requires efficient utilization of the available computational resources. To

[1]Such a general QP is NP-hard to solve exactly [1], and there are no local criteria to determine if the local minimizer is also a global one [2, 3].

facilitate this, most of the algorithms presented in this work have been implemented in the PERMON library [5].

PERMON (Parallel, Efficient, Robust, Modular, Object, Numerical)[2] is a C library based on PETSc [6]. PERMON provides data structures, transformations, algorithms, and supporting functions for QP problem solutions. PERMON also contains FETI-type domain decomposition [7, 8], which can significantly accelerate the solving of certain types of QP problems. Parallelism, and more specifically scalability, are among the primary development goals of PERMON. Our main objective is to solve large QP problems on modern supercomputers. As such, PERMON was benchmarked on problems containing more than one billion unknowns (see Section 7.2.1) and was run on up to 27,000 cores (see Section 4.3.9). However, PERMON can also run efficiently on standard personal computers.

The author's own results are summarized in the introduction of each chapter and in the conclusion 8.1. A list of publications by the author is available in Appendix A.

The thesis is divided into eight chapters and has the following outline:

- Chapter 2 reviews the optimization theory used in this thesis. In particular, Section 2.4.1 on optimality conditions and Section 2.5 on duality are important.

- Chapter 3 introduces the software and supercomputers used for computing numerical experiments on a number of benchmarks, which are also described in this chapter. The benchmarks are introduced in an early chapter because they are used throughout the following sections, sometimes to motivate the development of the algorithms' improvements.

- Chapter 4 discusses solution methods for unconstrained QP problems. These methods serve as building blocks for the algorithms for constrained QP problems in the subsequent chapters. Specifically, we introduce the steepest descent method and improvements consisting of either the Barzilai-Borwein step length or the conjugate gradient and deflated conjugate gradient methods. Finally, a general deflation preconditioner, which is implemented in PETSc, is discussed.

- In Chapter 5, we describe the MPRGP algorithm for the solution of QP problems with box constraints or constraints with similarly cheap projections onto the feasible set. The main part of the chapter is devoted to the improvements of the MPRGP algorithm using modifications of the active set expansion step and approximate preconditioning in face. Both improvements exhibit very large speedups.

- Chapter 6 discusses methods for the solution of equality-constrained problems. In particular, the range space method and an augmented Lagrangian method (SMALE) are used in the following chapter.

[2]The author of this thesis has been a PERMON maintainer since 2018, with contributions starting in 2015.

- In Chapter 7, the linear inequality QP problems are solved by transforming them into bound and possibly equality-constrained problems using duality. FETI-type domain decomposition methods are introduced as a way to accelerate the solution of the dual QP problems. A number of scalability improvements for the FETI method are discussed and supported by numerical experiments.
- Finally, Chapter 8 provides the conclusion, where the results of the thesis are summarized, the author's contributions are reiterated, and future plans are discussed.

Chapter 2

Optimization Overview

This chapter reviews basic facts and concepts related to optimization, with a particular focus on QP. After introducing our optimization problem, we review the convexity of functions and sets. The convexity of the feasible set ensures the existence of the projection onto the feasible set. This projection is used in many algorithms described in the subsequent sections. After describing the projection operators, we review some basic optimality conditions and introduce the first-order necessary (and sufficient) optimality tests. Then we briefly describe duality and the dual problem. Finally, we make a note on the descent direction, which is a key concept in iterative optimization algorithms.

The primary references for this section are [9–13].

2.1 Optimization Problem

A general optimization problem is

$$\arg\min_{\boldsymbol{x}} f(\boldsymbol{x}) \quad \text{s.t.} \quad \begin{cases} g_i(\boldsymbol{x}) \leq 0 & \text{for } i = 1, \dots, m, \\ h_j(\boldsymbol{x}) = 0 & \text{for } j = 1, \dots, r, \\ \quad x \in X \subset \mathbb{R}^n, \end{cases}$$

where f is the *cost function*, g_i and h_j are inequality and equality constraints, respectively, and X represents other constraints, e.g., $X = \{x \mid x_i \in \mathbb{Z}\}$ for some i. The set X is typically assumed to be closed, and the functions f, g_i, and h_j are typically assumed to be smooth (at least of class C^1) [13]. The actual *feasible set* (that is, the set of feasible solutions), which we denote Ω, is given as the intersection of the individual constraint sets.

This work deals with convex (see the next section) quadratic programming problems

$$\arg\min_{\boldsymbol{x}} \frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} - \boldsymbol{x}^T\boldsymbol{b} \quad \text{s.t.} \quad \boldsymbol{x} \in \Omega, \tag{2.1}$$

where $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is a symmetric positive semidefinite (SPS) matrix called the *Hessian* matrix, vector $\boldsymbol{b} \in \mathbb{R}^n$ is known as the *right-hand side*, and Ω is a closed and convex set of constraints on the solution vector $\boldsymbol{x} \in \mathbb{R}^n$. The minimized quadratic function $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} - \boldsymbol{x}^T\boldsymbol{b}$ is the cost function.

A visualization of a quadratic cost function $f(\boldsymbol{x})$ with a symmetric positive definite (SPD) Hessian of dimension $n = 2$ can be found in Figure 2.1.

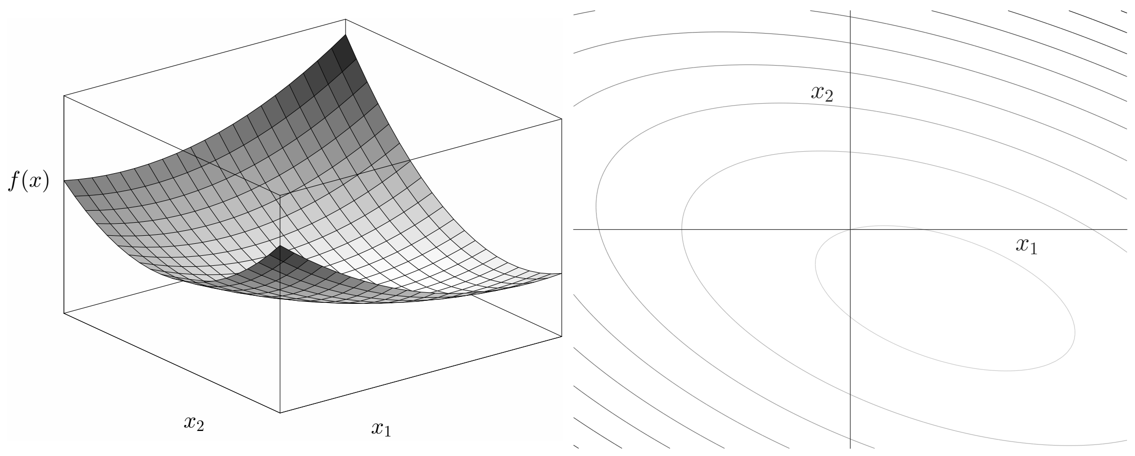

Figure 2.1: Surface and contour plot of a quadratic cost function $f(\boldsymbol{x})$ with an SPD Hessian of dimension $n = 2$. The feasible set can be visualized in the right plot. For example, $\boldsymbol{x}_2 \geq 0$ limits the feasible solutions to the $\boldsymbol{x}_1$ axis and above.

2.2 Convexity

A set Ω is *convex* if the line segment connecting any two points in Ω lies in Ω, which is formally summarized below.

Definition 2.2.1 (Convex Set)**.** *A subset $\Omega \subset \mathbb{R}^n$ is called convex if*

$$\alpha \boldsymbol{x} + (1-\alpha)\,\boldsymbol{y} \in \Omega, \qquad \forall \boldsymbol{x}, \boldsymbol{y} \in \Omega, \forall \alpha \in [0,1]\,.$$

Examples of convex sets include the empty set, sets containing a single point, real intervals, and the set of real numbers $\mathbb{R}$. On the other hand, some sets used as constraints in optimization are not convex, e.g., the set of integers $\mathbb{Z}$ or the set $\{0, 1\}$ used in binary optimization. Importantly, the intersection of closed convex sets is closed and convex [9].

Similarly to the convex set, a function f is convex when it lies below the line segment connecting any two points on its graph.

Definition 2.2.2 (Convex Function)**.** *Let a subset $\Omega \subset \mathbb{R}^n$ be convex. A function $f : \Omega \mapsto \mathbb{R}$ is convex if*

$$f\left(\alpha \boldsymbol{x} + (1-\alpha)\,\boldsymbol{y}\right) \leq \alpha f\left(\boldsymbol{x}\right) + (1-\alpha)\, f\left(\boldsymbol{y}\right) \qquad \forall \boldsymbol{x}, \boldsymbol{y} \in \Omega, \forall \alpha \in (0,1)\,.$$

The function f is called strictly convex *if the above inequality is strict.*

The restriction to QP problems with an SPS Hessian gives us the convexity of the cost function, while an SPD Hessian gives us the strict convexity [10].

Convexity is a very strong property of functions and sets. For instance, if a convex function f is minimized over a convex set, then every local optimal solution is global [9]. Moreover, convexity can give us the existence and possibly even the uniqueness of the optimum. Therefore, it is a central property in optimization in general and is widely used in the following sections.

2.3 Projection onto Convex Sets

Our requirement for the feasible set Ω to be closed and convex ensures the existence and uniqueness of the projection onto Ω. The general definition of the projection is given by the following theorem.

Theorem 2.3.1 (Projection Theorem)**.** *Let a subset $\Omega \subset \mathbb{R}^n$ be nonempty, closed, and convex. The projection $P_\Omega : \mathbb{R}^n \mapsto \Omega$ is defined as*

$$P_\Omega(\boldsymbol{x}) = \arg\min_{\boldsymbol{y}\in\Omega} ||\boldsymbol{y} - \boldsymbol{x}||.$$

Then the projection $P_\Omega(\boldsymbol{x})$ is the unique minimizer *of the Euclidean distance between $\boldsymbol{x} \in \mathbb{R}^n$ and all $\boldsymbol{y} \in \Omega$.*

Proof. See Proposition 1.1.4 in [9]. □

An equivalent formulation is to minimize the squared norm of the above distance, leading to a constrained least squares problem

$$P_\Omega(\boldsymbol{x}) = \arg\min_{\boldsymbol{y}\in\Omega} \frac{1}{2}||\boldsymbol{y} - \boldsymbol{x}||^2,$$

which is a QP problem.

A common application of the projection is to ensure that iterates in an algorithm are kept in the feasible set, which is utilized by the algorithms of Chapter 5.

2.3.1 Projection Examples

Projections onto some sets have closed forms. These closed forms are typically easier to evaluate than using an algorithm to solve the projection QP problem given in Theorem 2.3.1. Selected projection operators are provided below.

2.3.1.1 Bound Constraints

For the *lower bound* constraints defined as

$$\Omega = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{l} \leq \boldsymbol{x}\},$$

the projection is given component-wise as

$$[P_\Omega(\boldsymbol{x})]_i = \max\{\boldsymbol{l}_i, \boldsymbol{x}_i\}, \quad i \in \{1, \dots, n\}.$$

The projection for the *upper bound* is given analogously.

2.3.1.2 Box Constraints

Combining the lower and upper bound constraints gives the *box* constraints defined as

$$\Omega = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{l} \le \boldsymbol{x} \le \boldsymbol{u}\}.$$

The projection onto the box constraints is defined by subsequent projections onto the lower and upper bounds, respectively. Therefore, the projection is given component-wise as

$$[P_\Omega(\boldsymbol{x})]_i = \min\{\boldsymbol{u}_i, \max\{\boldsymbol{l}_i, \boldsymbol{x}_i\}\}, \quad i \in \{1, \dots, n\}.$$

2.3.1.3 Linear Equality Constraints

Homogeneous[1] linear equality constraints are defined as

$$\Omega = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{B}\boldsymbol{x} = \boldsymbol{o}\}, \text{ where } \boldsymbol{B} \in \mathbb{R}^{m \times n}.$$

The projection is then defined as the orthogonal projection onto the null space of $\boldsymbol{B}$. If $\boldsymbol{B}$ is full row rank, which means $\left(\boldsymbol{B}\boldsymbol{B}^T\right)^{-1}$ exists, then the projection can be defined as

$$P_\Omega(\boldsymbol{x}) = \left[\boldsymbol{I} - \boldsymbol{B}^T\left(\boldsymbol{B}\boldsymbol{B}^T\right)^{-1}\boldsymbol{B}\right]\boldsymbol{x}.$$

In any case, for a general matrix $\boldsymbol{B}$, the projection is defined as

$$P_\Omega(\boldsymbol{x}) = \left[\boldsymbol{I} - \boldsymbol{B}^+\boldsymbol{B}\right]\boldsymbol{x},$$

where $\boldsymbol{B}^+$ is the *Moore-Penrose inverse* of $\boldsymbol{B}$ [14]. We note that if $\boldsymbol{B}$ is not full row rank, there are redundant linear equality constraints that could be eliminated.

For the nonhomogeneous linear equality constraints[2]

$$\Omega = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{B}\boldsymbol{x} = \boldsymbol{c}\} \ne \emptyset, \text{ where } \boldsymbol{B} \in \mathbb{R}^{m \times n},$$

the projection can be modified. Let

$$\boldsymbol{x} = \boldsymbol{x}_{\mathrm{Im}} + \boldsymbol{x}_{\mathrm{Ker}}, \tag{2.2}$$

where $\boldsymbol{x}_{\mathrm{Im}} \in \operatorname{Im} \boldsymbol{B}^T$ and $\boldsymbol{x}_{\mathrm{Ker}} \in \operatorname{Ker} \boldsymbol{B}$. Then the solution for the image part is given as the least squares solution for the linear equality, which is

$$\boldsymbol{x}_{\mathrm{Im}} = \boldsymbol{B}^+\boldsymbol{c},$$

or equivalently, when $\boldsymbol{B}$ is full row rank,

$$\boldsymbol{x}_{\mathrm{Im}} = \boldsymbol{B}^T\left(\boldsymbol{B}\boldsymbol{B}^T\right)^{-1}\boldsymbol{c}.$$

[1] A homogeneous system of linear equations has the right-hand side equal to the zero vector.

[2] Technically, the constraint functions are affine. We take the name linear from the fact that $\boldsymbol{B}$ is a linear map. The distinction between affine and linear is used only in Section 2.4.1 about optimality conditions.

The solution for the null space is given by the projections in the above paragraph. Substituting the solutions for the image and null space parts into Equation (2.2) gives us the projection

$$P_\Omega(\boldsymbol{x}) = \left[\boldsymbol{I} - \boldsymbol{B}^+\boldsymbol{B}\right]\boldsymbol{x} + \boldsymbol{B}^+\boldsymbol{c}$$

or equivalently, when $\boldsymbol{B}$ is full row rank,

$$P_\Omega(\boldsymbol{x}) = \left[\boldsymbol{I} - \boldsymbol{B}^T\left(\boldsymbol{B}\boldsymbol{B}^T\right)^{-1}\boldsymbol{B}\right]\boldsymbol{x} + \boldsymbol{B}^T\left(\boldsymbol{B}\boldsymbol{B}^T\right)^{-1}\boldsymbol{c},$$

which we also derive in Example 2.4.2 from the optimality conditions.

The splitting (2.2) of the solution vector into two orthogonal vectors, one vector in the image of $\boldsymbol{B}^T$ and the other in the null space of $\boldsymbol{B}$, is known as the homogenization of the linear equality constraints. After homogenization, the original QP problem is reduced to a QP problem restricted to the null space of $\boldsymbol{B}$, i.e., we solve for $\boldsymbol{x}_{Ker}$ with homogeneous linear equality constraints

$$\Omega = \left\{\boldsymbol{x}_{Ker} \in \mathbb{R}^n \mid \boldsymbol{B}\boldsymbol{x}_{Ker} = \boldsymbol{o}\right\},$$

where the cost function and potentially other constraints present are shifted by substituting Equation (2.2) with known $\boldsymbol{x}_{\mathrm{Im}}$ into them.

The homogenization of the linear equality constraints is often employed to simplify more general QP problems, especially when additional constraints are involved. We use it in the FETI method derivation in Section 7.1.2.

2.4 Optimality Conditions

The following theorem provides important conditions for the existence of the optimum for fairly general functions.

Theorem 2.4.1 (Weierstrass' Theorem [9]). *Let a subset $\Omega \subset \mathbb{R}^n$ be nonempty and let the cost function $f : \Omega \mapsto \mathbb{R}$ be lower semicontinuous*[3] *at all points of Ω. Assume that one of the three conditions holds:*

1. *Ω is compact*[4]*.*

2. *Ω is closed and f is coercive*[5]*.*

3. *There exists a scalar γ such that the level set*

$$\{\boldsymbol{x} \in \Omega \mid f(\boldsymbol{x}) \leq \gamma\}$$

is nonempty and compact.

[3]Function f is *lower semicontinuous* at vector $\boldsymbol{x}$ if $f(\boldsymbol{x}) \leq \liminf_{k\to+\infty} f(\boldsymbol{x}_k)$ for every sequence $\{\boldsymbol{x}_k\} \in \Omega$ that converges to $\boldsymbol{x}$.

[4]A set is *compact* if it is closed and bounded. A subset of $\mathbb{R}^n$ is *bounded* if it is inside a ball of finite radius.

[5]Function $f(x)$ is said to be *coercive* if $f(x) \to +\infty$ as $||\boldsymbol{x}|| \to +\infty$.

Then the set of minima of f over Ω is nonempty and compact.

Proof. See Proposition A.8 in [9]. □

Applying Weierstrass' theorem to our QP problem, we observe that the QP cost function is continuous. However, the feasible set Ω may or may not be compact. While we always assume that the set is closed, it may not be bounded, e.g., partially constrained problems or problems with only a lower bound. As for the second condition, our QP cost function is coercive if and only if the Hessian is positive definite [10].

The Frank-Wolfe theorem provides the existence of a solution for quadratic functions for a certain set Ω, provided that the cost function is bounded.

Theorem 2.4.2 (Frank-Wolfe Theorem)**.** *Let f be a quadratic function that is bounded below on a convex polyhedron $\Omega \subset \mathbb{R}^n$. Then f attains its minimum on Ω.*

Proof. See statement (i) in the appendix of [15]. □

A convex polyhedron is an intersection of finitely many closed half-spaces [12], which is often written in terms of vectors $\boldsymbol{x}$ satisfying a linear inequality $\boldsymbol{Bx} \leq \boldsymbol{o}$. The restriction on the set Ω in the statement can be relaxed [16], e.g., for Ω a vector sum of a convex and compact set and a closed cone [9]. If the right-hand side is in the range of the Hessian ($\boldsymbol{b} \in \operatorname{Im} \boldsymbol{A}$) and the Hessian is SPS, then the cost function f is bounded from below [10].

It is not enough for the set Ω to be closed and convex and for a QP cost function to be bounded below on Ω for the solution to exist, as illustrated by the following example.

Example 2.4.1. Let $\Omega = \{(x, y) \in \mathbb{R}^2 \mid xy \geq 1 \wedge y \geq 0\}$ and $f(x, y) = x^2$. Then f has an infimum of 0 on Ω, which is closed and convex, but the infimum is not attained on Ω.

2.4.1 Necessary and Sufficient Optimality Conditions

In practice, it is important to be able to recognize whether a given vector can be, or even is, a solution. For simplicity, let us consider a general minimization problem with the feasible set given only by inequality and equality constraints

$$\arg\min_{\boldsymbol{x}} f(\boldsymbol{x}) \quad \text{s.t.} \quad \begin{cases} g_i(\boldsymbol{x}) \leq 0 & \text{for } i = 1, \dots, m, \\ h_j(\boldsymbol{x}) = 0 & \text{for } j = 1, \dots, r, \end{cases} \tag{2.3}$$

where the functions f, g_i, and h_j are continuously differentiable[6]. Writing the feasible set in vector notation

$$\Omega = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{o} \wedge \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{o}\},$$

where $\boldsymbol{g}(\boldsymbol{x}) = (g_1(\boldsymbol{x}), \dots, g_m(\boldsymbol{x}))^T$, $\boldsymbol{h}(\boldsymbol{x}) = (h_1(\boldsymbol{x}), \dots, h_r(\boldsymbol{x}))^T$, and the inequality is understood component-wise, we introduce the Lagrangian function

$$L(\boldsymbol{x}, \boldsymbol{\lambda}_I, \boldsymbol{\lambda}_E) = f(\boldsymbol{x}) + \boldsymbol{\lambda}_I^T \boldsymbol{g}(\boldsymbol{x}) + \boldsymbol{\lambda}_E^T \boldsymbol{h}(\boldsymbol{x}).$$

[6]The condition that the cost and constraint functions are of class C^1 can be relaxed, and constraints that are not expressed as equality or inequality can be included as well; see, e.g., [13].

The components of vectors $\boldsymbol{\lambda}_I$ and $\boldsymbol{\lambda}_E$, which we will sometimes write more compactly as $\boldsymbol{\lambda} = \left(\boldsymbol{\lambda}_I^T, \boldsymbol{\lambda}_E^T\right)^T$, are called *Lagrange multipliers*. The *first-order necessary optimality conditions* are

$$\begin{aligned}
\nabla_{\boldsymbol{x}} L(\boldsymbol{x}, \boldsymbol{\lambda}_I, \boldsymbol{\lambda}_E) &= \boldsymbol{o} \\
\nabla_{\boldsymbol{\lambda}_I} L(\boldsymbol{x}, \boldsymbol{\lambda}_I, \boldsymbol{\lambda}_E) = \boldsymbol{g}(\boldsymbol{x}) &\leq \boldsymbol{o} \\
\nabla_{\boldsymbol{\lambda}_E} L(\boldsymbol{x}, \boldsymbol{\lambda}_I, \boldsymbol{\lambda}_E) = \boldsymbol{h}(\boldsymbol{x}) &= \boldsymbol{o} \\
\boldsymbol{\lambda}_I^T \boldsymbol{g}(\boldsymbol{x}) &= 0 \\
\boldsymbol{\lambda}_I &\geq \boldsymbol{o}.
\end{aligned}$$

The first condition is the *stationarity* condition, which is analogous to $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \boldsymbol{o}$ for unconstrained optimization. The next two conditions are the *primal feasibility* conditions. Finally, the last two conditions are the *complementary slackness* and the *dual feasibility*. The complementary slackness forbids two linked inequalities ($g_i(\boldsymbol{x})$ and $[\boldsymbol{\lambda}_I]_i$) to be "slack", i.e., hold with strict inequality. These first-order necessary optimality conditions are known as *Karush-Kuhn-Tucker (KKT) conditions* [9].

In the case of *convex optimization* (functions f and g_i are convex, h_j are affine), the KKT conditions are also sufficient [11]. This means that any point $(\boldsymbol{x}, \boldsymbol{\lambda})$ that satisfies the KKT conditions is primal and dual optimal, i.e., $\boldsymbol{x}$ is the solution of problem (2.3), and $\boldsymbol{\lambda}$ is the solution to the dual problem introduced in the following section. Recall that the QP problems in which we are interested are convex.

The existence (and uniqueness) of the Lagrange multipliers (and thus the existence of the primal solution $\boldsymbol{x}$ in the convex case) can be guaranteed by the constraint qualifications. We provide three examples of constraint qualifications in Definitions 2.4.4 to 2.4.6. Many more can be found in [17].

Remark 2.4.3. *Assume that $\boldsymbol{x}^*$ is the local minimum of problem (2.3). Then if one of the following constraint qualifications is satisfied, there exists $\boldsymbol{\lambda}^*$ such that the pair $(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)$ satisfies the KKT conditions [9].*

Definition 2.4.4 (Linearity Constraint Qualification (LCQ))**.** *Functions g_i and h_j are affine.*

Definition 2.4.5 (Linear Independence Constraint Qualification (LICQ))**.** *The gradients of the equality constraints and of the active inequality constraints are linearly independent at $\boldsymbol{x}^*$. The* active inequality constraints *at $\boldsymbol{x}^*$ are those for which $g_i(\boldsymbol{x}^*) = 0$.*

Definition 2.4.6 (Slater's Constraint Qualification (SCQ))**.** *The equality constraints h_j are affine, the inequality constraints g_i are convex, and there exists a feasible vector $\overline{\boldsymbol{x}}$ satisfying*

$$g_j(\overline{\boldsymbol{x}}) < 0 \quad \textit{for each inequality constraint active at } \boldsymbol{x}^*.$$

The KKT conditions for convex problems, equipped with a constraint qualification, can be used not only to identify whether we have found a minimizer but also to solve the optimization problem. The following example provides the projection onto the set defined by nonhomogeneous linear equality constraints that we promised in Section 2.3.

Example 2.4.2. To obtain the projection $P(\boldsymbol{x})$ onto a set defined by linear equality constraints $\boldsymbol{By} = \boldsymbol{c}$, where $\boldsymbol{B}$ is full row rank, we need to solve by Theorem 2.3.1

$$P(\boldsymbol{x}) = \arg\min_{\boldsymbol{y}} \frac{1}{2}||\boldsymbol{y} - \boldsymbol{x}||^2 \quad \text{s.t.} \quad \boldsymbol{By} = \boldsymbol{c},$$

where we used the equivalent QP problem. The KKT conditions are both necessary (since the constraints are affine) and sufficient (since the cost function is convex). Writing the Lagrangian

$$\frac{1}{2}||\boldsymbol{y} - \boldsymbol{x}||^2 + \lambda^T (\boldsymbol{By} - \boldsymbol{c}),$$

and then writing the KKT conditions, we have

$$\begin{aligned} \nabla_{\boldsymbol{y}} L(\boldsymbol{x}, \boldsymbol{\lambda}) &= \boldsymbol{y} - \boldsymbol{x} + \boldsymbol{B}^T \boldsymbol{\lambda} = \boldsymbol{o}, \\ \nabla_{\boldsymbol{\lambda}} L(\boldsymbol{x}, \boldsymbol{\lambda}) &= \boldsymbol{By} - \boldsymbol{c} = \boldsymbol{o}. \end{aligned}$$

Substituting the first equation into the second, we obtain

$$\boldsymbol{Bx} - \boldsymbol{BB}^T \boldsymbol{\lambda} - \boldsymbol{c} = \boldsymbol{o}.$$

Solving this equation for the only unknown $\boldsymbol{\lambda}$, we get (since $\boldsymbol{B}$ is full rank, so that $\left(\boldsymbol{BB}^T\right)^{-1}$ exists)

$$\boldsymbol{\lambda} = \left(\boldsymbol{BB}^T\right)^{-1} (\boldsymbol{Bx} - \boldsymbol{c}).$$

Finally, we substitute the known $\boldsymbol{\lambda}$ into the first KKT condition to obtain the projection

$$P(\boldsymbol{x}) = \boldsymbol{y} = \boldsymbol{x} - \boldsymbol{B}^T \left(\boldsymbol{BB}^T\right)^{-1} (\boldsymbol{Bx} - \boldsymbol{c}).$$

2.5 Duality and Dual Problem

In the previous example, we first solved for the vector of Lagrange multipliers $\boldsymbol{\lambda}$. In fact, we found the solution of

$$\arg\max_{\boldsymbol{\lambda}} q(\boldsymbol{\lambda}) \quad \text{s.t.} \quad \boldsymbol{\lambda}_I \geq \boldsymbol{o},$$

where

$$q(\boldsymbol{\lambda}) = \inf_{\boldsymbol{x} \in \Omega} L(\boldsymbol{x}, \boldsymbol{\lambda})$$

is the *dual function*. The maximization problem above is the *dual problem* to the problem (2.3), which we analogously call the *primal problem*. The solution $\boldsymbol{\lambda}^*$ is the *dual solution* that, together with the *primal solution* $\boldsymbol{x}^*$, forms the *primal-dual solution pair*.

Defining the *optimal primal value* with the primal solution $\boldsymbol{x}^*$

$$f^* = f(\boldsymbol{x}^*),$$

and similarly for the dual problem

$$q^* = \sup_{\boldsymbol{\lambda}_I \geq \boldsymbol{o}} q(\boldsymbol{\lambda}),$$

the *optimal dual value* always underestimates the optimal primal value.

Theorem 2.5.1 (Weak Duality Theorem [9])**.** *We have*

$$q^* \leq f^*.$$

Proof. See Proposition 6.1.3 in [9]. □

The difference between f^* and q^*

$$f^* - q^*$$

is the *optimal duality gap*, which, by the weak duality theorem, is always nonnegative. If the optimal duality gap is zero, we say that *strong duality* holds. Strong duality holds if the primal problem is convex and a constraint qualification holds [9]. If strong duality holds, the *duality gap*

$$f(\boldsymbol{x}) - q(\boldsymbol{\lambda})$$

for feasible $\boldsymbol{x}$ and $\boldsymbol{\lambda}$ can be used as a stopping criterion in algorithms.

A characterization of a primal and dual solution is provided by the following theorem.

Theorem 2.5.2 (Saddle Point Theorem)**.** *A primal-dual pair* $(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)$ *is the solution to the primal and the dual problem if and only if* $\boldsymbol{x}^* \in \Omega$, $\boldsymbol{\lambda}_I^* \geq 0$, *and* $(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)$ *is the* saddle point *of the Lagrangian, in the sense that*

$$L(\boldsymbol{x}^*, \boldsymbol{\lambda}) \leq L(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) \leq L(\boldsymbol{x}, \boldsymbol{\lambda}^*), \qquad \forall \boldsymbol{x} \in \Omega, \boldsymbol{\lambda}_I \geq \boldsymbol{o}.$$

Proof. See Proposition 6.1.6 in [9]. □

We note that the optimal value for the primal problem can be expressed symmetrically to the optimal dual value as

$$p^* = \inf_{\boldsymbol{x} \in \Omega} \sup_{\boldsymbol{\lambda}_I \geq \boldsymbol{o}} L\left(\boldsymbol{x}, \boldsymbol{\lambda}\right).$$

Together with the saddle point theorem and strong duality, this min-max property of the primal and dual cost functions gives rise to zero-sum games [11, 13]. Other problems exhibit natural saddle point structures as well, e.g., Stokes flow [18], where the pressure can be interpreted as a Lagrange multiplier, and optimization algorithms for solving the saddle point (or enforcing the equality constraints on the divergence of velocity) can be used.

2.6 Descent Direction

Many iterative algorithms make steps updating the approximation $\boldsymbol{x}_k$ in the form

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha \boldsymbol{d},$$

where $\boldsymbol{d}$ is a *descent* direction.

Assuming unconstrained QP minimization

$$\arg\min_{\boldsymbol{x}} \frac{1}{2}\boldsymbol{x}^T \boldsymbol{A}\boldsymbol{x} - \boldsymbol{x}^T \boldsymbol{b},$$

a vector $\boldsymbol{d}$ is also a *decrease* direction if a step in the direction $\boldsymbol{d}$ reduces the cost function value

$$f(\boldsymbol{x} + \alpha \boldsymbol{d}) < f(\boldsymbol{x})$$

for all sufficiently small $\alpha > 0$. Using Taylor expansion

$$f(\boldsymbol{x} + \alpha \boldsymbol{d}) = f(\boldsymbol{x}) + \alpha(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})^T \boldsymbol{d} + \frac{\alpha^2}{2}\boldsymbol{d}^T \boldsymbol{A}\boldsymbol{d},$$

we have that $\boldsymbol{d}$ is a decrease direction if and only if

$$(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})^T \boldsymbol{d} < 0.$$

Some strategies for choosing the step length α will be discussed in Chapter 4 and Section 5.2.3. In the case of constrained problems, we need to ensure that the approximation $\boldsymbol{x}_{k+1}$, after taking the step, is feasible.

Chapter 3

Implementation of Quadratic Programming Algorithms and Quadratic Programming Benchmarks

The behavior and performance of the presented algorithms for QP are illustrated and supported by a number of benchmarks. The benchmarks are interspersed throughout the following chapters, and the analysis of the presented QP algorithms on these benchmarks is often used to motivate further developments and improvements of the algorithms. As the performance of the algorithms, particularly their timing, is affected by the implementation and hardware, these aspects are also described.

First, the software, then the hardware, and finally the benchmarks are described. Each benchmark description contains a short overview of the problem, which is followed by a more detailed description. The benchmark sections of this chapter may be skipped and only referred to when the benchmarks are encountered in the following chapters, either by reading the short overview or the complete description.

Most of the algorithms presented in the following chapters are implemented in the PERMON library. The new implementations and improvements to the library by the thesis's author are considered part of this thesis. The ability of the PERMON library to solve QP problems in a scalable manner has proven to be useful for several applications and libraries.

3.1 Software Implementing QP Algorithms

Most of the presented algorithms are implemented in the PERMON library. The PERMON library utilizes the PETSc library. Both libraries are described in the remainder of this section.

The software and parameters used to compile the presented software greatly affect the performance of the implementations. These are presented for individual machines in the following Hardware section 3.2.

3.1.1 PETSc-TAO

PETSc (Portable, Extensible Toolkit for Scientific Computation) [6, 19, 20] is a library for the scalable (parallel) solution of scientific applications modeled by partial differential equations. It provides vectors, matrices, discretization management and data communication, linear solvers, nonlinear solvers, time integrators, and more. In addition, TAO (Toolkit for Advanced Optimization), an optimization library, is a part of PETSc.

The parallelization of the library is primarily achieved through the row-wise distribution of vectors and matrices among the available computational cores, which is realized by MPI. The use of GPUs is also supported.

PETSc is written in C and provides Fortran and Python bindings, as well as an interface to many additional numerical libraries. From these, we mostly use external direct solver libraries MUMPS [21, 22] and SuperLU_DIST [23], which provide fast parallel implementations of the Cholesky and/or LU decompositions.

3.1.2 PERMON

PERMON [5] stands for Parallel, Efficient, Robust, Modular, Object-Oriented, and Numerical. The main PERMON module is the PermonQP library, which consists of the object for QP problem definition, QP transformations, QP solvers, and supporting functions. PermonQP provides a number of QP transformations that are used to modify a given QP problem so that the formulation can be solved by the provided solvers and/or improved so that the solution is more efficient. Each QP transformation creates a new QP problem with a function that can reconstruct the solution of the QP problem before transformation. These QP problems are placed into a chain (doubly linked list) illustrated in Figure 3.1. The FETI method described in Section 7.1 is essentially implemented in PermonQP as a chain of QP transformations: dualization, homogenization of the equality constraints, and projection enforcing the equality constraints. More information about the QP transformations and the QP transformation chain can be found in [24]. The closed development of the library started in 2011, and it was made publicly available in 2016.

The second PERMON module is called PermonSVM. It provides a library and a computer program to train linear support vector machine classifiers, which are described in Section 3.3.6. An overview of the features of PermonSVM can be found in the README in the project's repository [25], with more details available in [26]. PermonSVM depends on PermonQP for the solution of QP problems. The public development of PermonSVM started in 2017.

The PERMON libraries are written in C and utilize the PETSc library as the linear algebra backend. Moreover, the TAO optimization solvers and linear systems KSP solvers,

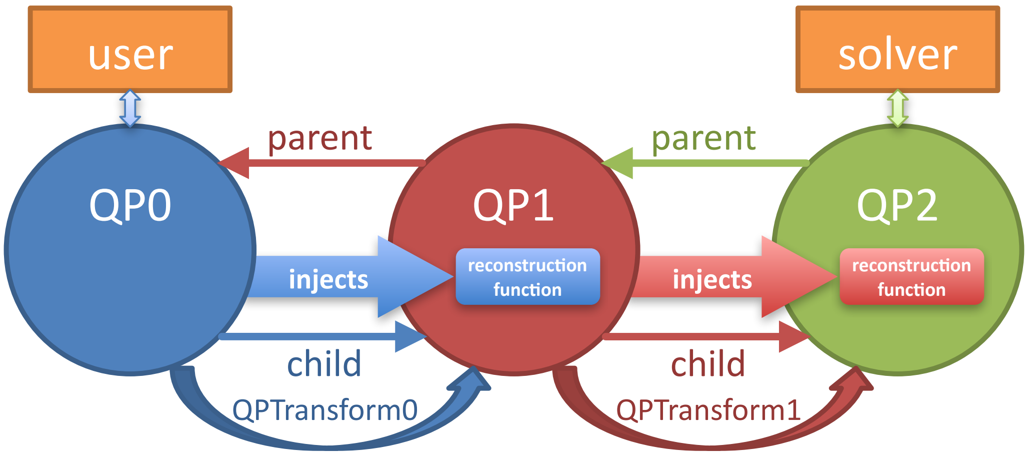

Figure 3.1: An illustration of the QP problems chain. The user provides the QP0 formulation, which is then transformed into the QP1 and finally the QP2 formulation. Each QP transformation injects a reconstruction function so that it is possible to reconstruct the solution of the original problem once QP2 is solved [24].

including their preconditioners, can be utilized directly to solve QP problems. The PERMON libraries have the same object design as PETSc, making them easy to use for users familiar with PETSc. A number of additional PETSc features are utilized to decrease the complexity of PERMON. These include, e.g., profiling, I/O routines, the build system, the test system, and documentation generation.

As in PETSc, most of the parallelization in PERMON comes from the row-wise distribution of matrices and vectors, which is realized by MPI. The scalability of PERMON was tested on up to 27,000 computational cores (Section 4.3.9) and on problems with more than a billion unknowns (Section 7.2.1).

Both modules are published under the permissive BSD-2-Clause open-source license and are available on GitHub [25].

PERMON has been interfaced by several software packages and research codes:

- The DEMSI project, solved in Los Alamos, uses PermonQP algorithms for load balancing particles in ice sheet melting simulations [27].
- The Flow123d library [28], developed at the Technical University of Liberec, utilizes FETI and QP solution infrastructure provided by PERMON to solve mechanical contact subproblems in hydro-mechanical problems.
- The HyTeG library [29], developed at the University of Erlangen-Nuremberg, providing high-performance finite element methods, can utilize PERMON to solve constrained FEM problems.
- The SIFEL library [30], developed at the Czech Technical University, can utilize the FETI method and the QP solution infrastructure provided by PERMON to solve large-scale mechanical contact problems in structural engineering.

- Wildfire detection software [26], developed by Argonne National Laboratory, the Institute of Geonics of the Czech Academy of Sciences, Oak Ridge National Laboratory, and the VSB - Technical University of Ostrava, uses support vector machines provided by PermonSVM to detect wildfires from satellite images.

The author of this thesis directly helped implement the interface or, at the very least, provided support for the codes listed above. In fact, the author has been the main contributor to PermonQP and the maintainer of PERMON since 2018. An overview of the contributed commits to PermonQP is in Figure 3.2, and the contributions can be viewed in the project's Git repository [25]. As the maintainer, the author made biannual major releases that coincided with the releases of PETSc. Altogether, there were 15 major releases from version 3.7 and a small number of minor releases. The current version, at the time of writing, is 3.21, which was released in April 2024.

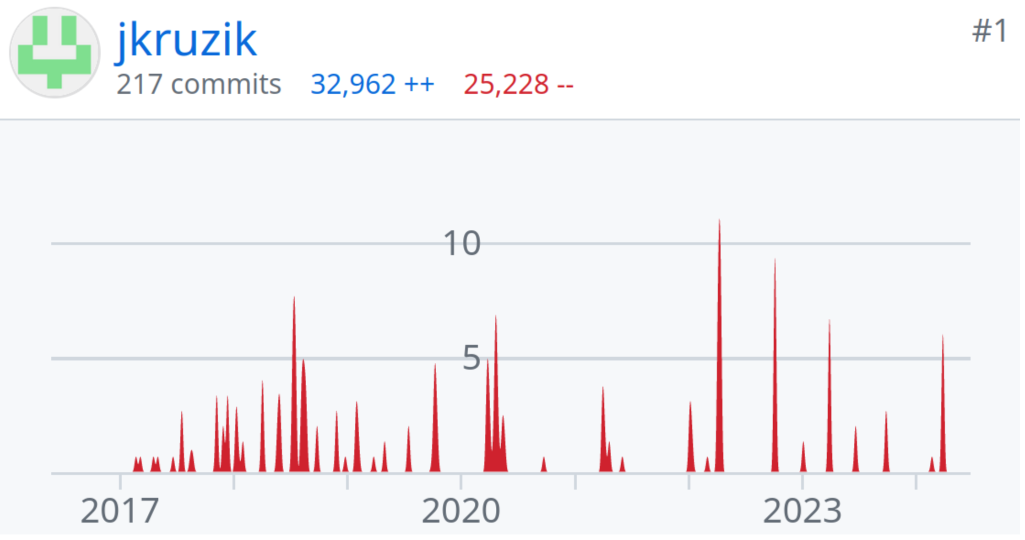

Figure 3.2: The number of commits (excluding merge commits) and line additions and deletions (nearly 60% of the repository's overall commits and lines changed) made by the author to PermonQP since the public development of PermonQP started in May 2016 until June 2024 [25].

3.2 Hardware Used to Compute Benchmarks

Large-scale benchmarks, as well as benchmarks where timings are provided, were run on one or more of the following supercomputers.

3.2.1 ARCHER

ARCHER [31] was a Cray XC30 supercomputer operated by EPCC (formerly the Edinburgh Parallel Computing Centre). It had a peak performance of 2.5 petaFLOPS, ranking at number 20 on the TOP500 list [32] when introduced. It was decommissioned in 2021.

The supercomputer contained 4, 920 nodes. Each node featured two 2.7 GHz, 12-core Intel Xeon E5-2697v2 (Ivy Bridge) processors and at least 64 GB of memory. The Cray Aries interconnect linked all compute nodes in a Dragonfly topology.

As for the software used in our benchmarks, the BLAS, LAPACK, and ScaLAPACK libraries were provided by Cray LibSci. The MPI implementation was Cray MPICH. All other libraries, including PERMON, PETSc, MUMPS, SuperLU, etc., were compiled using optimized builds (-*O3*) by the Cray Compiling Environment (CCE).

3.2.2 LUMI

LUMI [33] is an HPE Cray EX supercomputer hosted by the LUMI consortium. It has a peak performance of 531 petaFLOPS, ranking at number 3 on the TOP500 list [32] when introduced.

Our experiments were run on the LUMI-C (CPU-only) partition, which consists of 2, 048 nodes. Each node contains two 2.45 GHz, 64-core AMD EPYC 7763 (Zen 3) processors and at least 256 GB of memory. The HPE Cray Slingshot-11 200 Gbps interconnect links all nodes in a Dragonfly topology.

As for the software used in our benchmarks, the BLAS, LAPACK, and ScaLAPACK libraries were provided by Cray LibSci. The MPI implementation was Cray MPICH. All other libraries were compiled using optimized builds (-*O3*) by the HPE Cray Compiling Environment (CCE).

3.2.3 MareNostrum 3

MareNostrum 3 [34] was an IBM supercomputer operated by the Barcelona Supercomputing Center. It had a peak performance of 1 petaFLOPS, ranking at number 30 on the TOP500 list [32] when updated to the final configuration. It was decommissioned in 2017.

The supercomputer contained 3, 056 nodes. Each node featured two 2.6 GHz, 8-core Intel Xeon E5-2670 (Sandy Bridge) processors and at least 32 GB of memory. Compute nodes were interconnected by InfiniBand FDR10.

As for the software used in our benchmarks, the BLAS, LAPACK, and ScaLAPACK libraries were provided by Intel MKL. The MPI implementation was IBM POE. All other libraries were compiled with optimized builds (-*O3*) using Intel compilers.

3.2.4 Salomon

Salomon [31] was an SGI ICE X supercomputer operated by the IT4Innovations National Supercomputing Center, VSB-Technical University of Ostrava. It had a peak performance of 2 petaFLOPS, ranking at number 40 on the TOP500 list [32] when introduced. It was decommissioned in 2021.

The supercomputer contained 1, 008 nodes. Each node featured two 2.5 GHz, 12-core Intel Xeon E5-2680v3 (Haswell) processors and 128 GB of memory. The nodes were interconnected by InfiniBand FDR56 with a 7D Enhanced hypercube topology.

As for the software used in our benchmarks, the BLAS, LAPACK, and ScaLAPACK libraries were provided by Intel MKL. The MPI implementation was Intel MPI. All other libraries were compiled with optimized builds (-*O3*) using Intel compilers.

3.3 QP Benchmarks

The algorithms described in the following chapters were tested on the benchmarks presented in this section. The description of each benchmark provides a brief overview in the first paragraph, followed by a more detailed description, which should allow for the reconstruction of the benchmark. The appropriate function spaces and the weak formulations are omitted, but they can be found in the provided references. The default initial guess for the benchmarks is a null vector, unless stated otherwise.

3.3.1 Random Box-Constrained Problems

Three random box-constrained QP problems, named BQP1, BQP2, and BQP3, were generated using code available at [35], based on the methodology described in [36].

All generated problems have non-degenerate solutions. The size of the Hessian is $n = 15,000$ with a condition number $\kappa(\boldsymbol{A}) = 10^4$, and the amount of near-degeneracy is $ndeg = 1$. The number of active variables at the solution is 10%, 50%, and 90% for problems BQP1, BQP2, and BQP3, respectively. The generated problems are available under the BQP-15000 directory in [37].

3.3.2 1D Poisson's Contact Problems

The problem is a model of string displacement with a lower bound. The first variant, *ex1*, has the bound constraints over the entire domain, while *ex2* has the same constraints but only over the first half of the domain. See Figure 3.3 for the visualization of the solutions to the problems. These problems correspond to PERMON examples with the same names.

Formally, we discretize the 1D Poisson's equation

$$\begin{aligned} -u''(x) &= -15, \qquad x \in [0,1] \\ u(0) &= u(1) = 0 \end{aligned}$$

by the central finite differences and impose the following constraint on the solution

$$u(x) \geq \frac{\sin\left(4\pi x - \frac{\pi}{6}\right)}{2} - 2, \qquad x \in \Omega,$$

where $\Omega = [0,1]$ for *ex1* and $\Omega = \left[0, \frac{1}{2}\right]$ for *ex2*.

3.3.3 2D Journal Bearing Problem

The *jbearing2* benchmark is a variant of the journal bearing problem from the MINPACK-2 test problem collection [38]. The benchmark computes a pressure distribution in a thin film

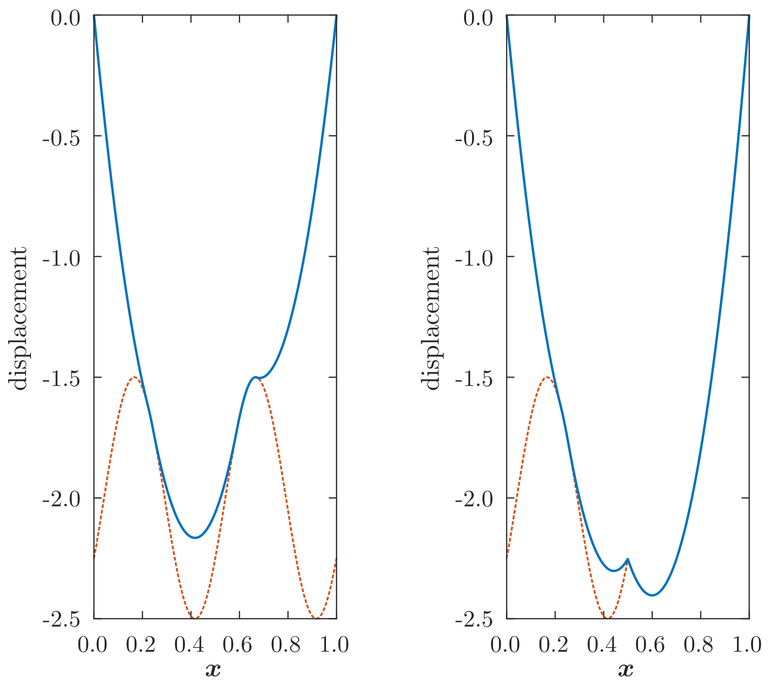

Figure 3.3: Solution for the *ex1* problem (left) and the *ex2* problem (right). The dashed line represents the values of the bound constraints.

of lubricant between a freely rotating cylindrical shaft (journal) inside a cylindrical sleeve and the aforementioned sleeve. An illustration of the solution can be found in Figure 3.4. The problem implementation can be found in PERMON as an example under the *jbearing2* name.

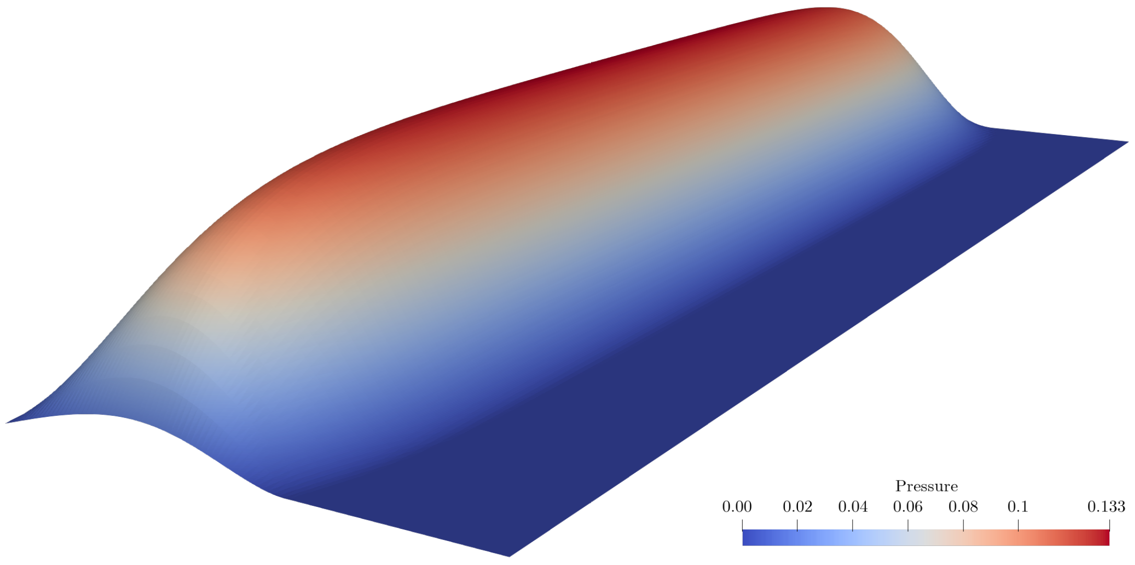

Figure 3.4: Solution for the journal bearing problem. The pressure is scaled by a factor of 20, with the legend containing the true values.

The continuous version of the problem has the form

$$\min_{v\in K} q(v) \equiv \int_{\mathcal{D}} \left(\frac{1}{2} w_q(x) \|\nabla v(x)\|^2 - w_l(x)v(x) \right) dx,$$

where $w_q(x_1, x_2) = (1 + \epsilon \cos x_1)^3$, $w_l(x_1, x_2) = \epsilon \sin x_1$ for some eccentricity constant $\epsilon \in (0, 1)$, and $\mathcal{D} = (0, 2\pi) \times (0, 2d)$, for some constant $d > 0$. Since the unknown variable is the pressure, which under normal operating conditions cannot be negative [39], we have a lower bound on the nonnegativity of the solution. Therefore, the convex feasible set is

$$K = \{v \in H_0^1(\mathcal{D})\colon v \geq 0 \text{ on } \mathcal{D}\},$$

where $H_0^1(\mathcal{D})$ is the Hilbert space of functions with compact support on $\mathcal{D}$ such that v and $\|\nabla v\|^2$ belong to $L^2(\mathcal{D})$.

We use P_1 finite elements regular discretization of this problem, which leads to a QP problem with nonnegativity lower bound constraints. For our tests, we consider the journal bearing problem with $\epsilon = 0.1$ and $d = 10$, and different discretizations characterized by the number of grid points in the x_1 and x_2 directions.

3.3.4 3D Linear Elasticity Cuboid Contact Problem

The linear elasticity cuboid is fixed at the bottom and pushed from the top. There is a rigid obstacle close to the right side, and non-penetration conditions are imposed between the cuboid and the obstacle. See Figure 3.5 for the problem illustration and solution.

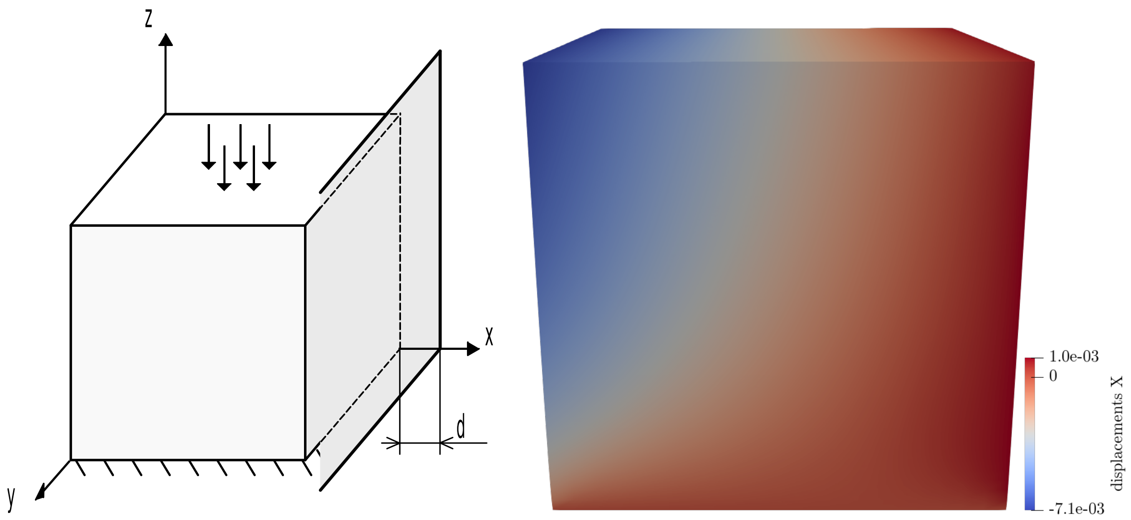

Figure 3.5: Problem setting for the 3D linear elasticity cuboid contact problem and the solution.

The model of linear elasticity [40] over a cuboid domain Ω, deforming under the application of the body force $\boldsymbol{f}$ and the surface force $\boldsymbol{g}$, can be described as

$$\begin{aligned} -\operatorname{div} \sigma &= \boldsymbol{f} && \text{in } \Omega, \\ \boldsymbol{u} &= \boldsymbol{o} && \text{on } \Gamma_D, \\ \sigma \cdot \boldsymbol{n} &= \boldsymbol{g} && \text{on } \Gamma_N, \end{aligned}$$

where the unknown $\boldsymbol{u}$ is the displacement, Γ_D is the bottom side, Γ_N is the top side, and $\boldsymbol{n}$ is the outer normal. Hooke's law, which reads

$$\sigma = \mathbf{C}\varepsilon(\boldsymbol{u}), \tag{3.1}$$

relates the Cauchy stress tensor σ to the strain tensor $\varepsilon(\boldsymbol{u})$, which is defined as

$$\varepsilon(\boldsymbol{u}) = \frac{1}{2}\left(\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^T\right),$$

by means of the symmetric positive definite elastic tensor $\mathbf{C}$. In our case, since we have an isotropic material, it holds that

$$\mathbf{C}\varepsilon(\boldsymbol{u}) = 2\mu\varepsilon(\boldsymbol{u}) + \lambda \operatorname{tr}(\varepsilon(\boldsymbol{u}))\boldsymbol{I},$$

where μ and λ are the material-dependent Lamé constants, and tr is the trace of a matrix defined as the sum of the diagonal elements. The Lamé constants are calculated from the Young's modulus E and the Poisson's ratio ν by the following relations

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)} \quad \text{and} \quad \mu = \frac{E}{2(1+\nu)}.$$

In our case, the parameters are $\boldsymbol{f} = \boldsymbol{o}$, $\boldsymbol{g} = -465$ N/mm^2, $E = 2 \cdot 10^5$ MPa, $\nu = 0.33$, the distance from the obstacle is 10^{-3} mm, and the cuboid is $1 \times 1 \times 1$ mm unless otherwise stated. The problem is regularly discretized with Q_1 finite elements.

3.3.5 3D Tunnel Excavation in Fractured Porous Medium

The hydro-mechanical model simulates the excavation of a cylindrical tunnel extending a large-profile drift and, more importantly, the subsequent changes in rock pressure resulting from this excavation. It is assumed that the highly permeable fractures affect the pressure evolution within the domain. The mechanics part is a QP problem with non-penetration (linear inequality) contact conditions on the fractures. An illustration of the geometry, including the fractures, is in Figure 3.6. A detailed view of a computational mesh with fractures is shown in Figure 3.7.

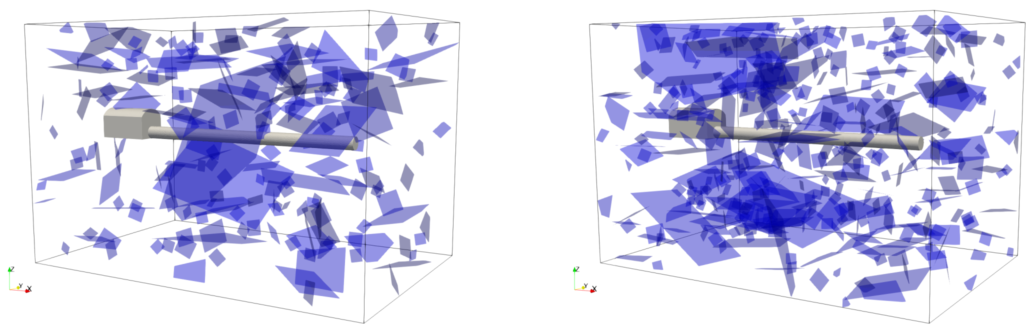

Figure 3.6: Two discrete fracture network (blue) configurations in the tunnel (grey) excavation test. Left: 200 fractures; right: 400 fractures [4].

The model is based on Biot's poroelasticity, describing the balance of mass and forces, as given by the equations

$$-\operatorname{div}\sigma = \boldsymbol{f} \qquad \text{in } I \times \Omega \tag{3.2}$$

$$\partial_t\left(Sp + (\operatorname{div}\boldsymbol{u})\right) + \operatorname{div}\boldsymbol{q} = g \qquad \text{in } I \times \Omega, \tag{3.3}$$

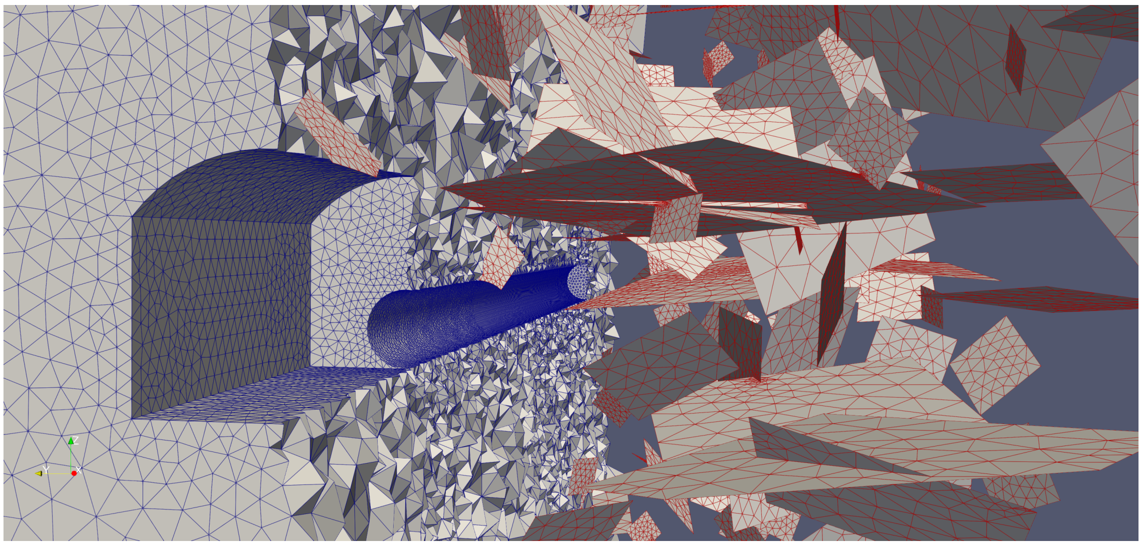

Figure 3.7: A detailed view of the computational mesh of the domain with 400 fractures (2D objects) [4].

where the unknowns $\boldsymbol{u}$ are the displacement and p is the pressure; furthermore, S is the storativity, $\boldsymbol{f}$ is the density of the body force, g is the density of the fluid source, and $I = [0, T]$ is the time interval.

The stress tensor σ is determined by Hooke's law (3.1), and the flux $\boldsymbol{q}$ is given by Darcy's law

$$\boldsymbol{q} = -\mathbf{K}\nabla p,$$

where $\mathbf{K}$ is the hydraulic conductivity tensor and ∇p is the hydraulic gradient. The problem is discretized by simplicial finite elements, where the displacement uses P_1, the pressure uses P_0, and the velocity uses the lowest-order Raviart-Thomas (RT_0) basis functions. The solution is obtained using the fixed-stress splitting method, which iterates between solving the discretized mechanics part Equation (3.2), which is a linear elasticity problem, and the discretized flow part Equation (3.3). See [4] for the boundary conditions and the precise problem setting, and see [4, 41] for the model derivation. The model is implemented in the Flow123d library [28].

3.3.6 Support Vector Machine Classification

The benchmark consists of learning the Support Vector Machine (SVM) linear classifier [42] implemented in PermonSVM [43]. The SVM datasets (Australian, Diabetes, and Ionosphere) used in the benchmark are available from the LIBSVM datasets webpage [44], and their sizes are given in Table 3.1.

The goal of the SVM classifier is to find the maximal-margin hyperplane that divides samples into two classes, with the hyperplane having the maximum distance (margin) between the two classes. Let us define a training set

$$\{(\boldsymbol{x}_1, y_1)\,,\ (\boldsymbol{x}_2, y_2)\,,\ \ldots,\ (\boldsymbol{x}_m, y_m)\},$$

Dataset	Training size	Number of features
Australian	690	14
Diabetes	768	8
Ionosphere	351	34

Table 3.1: Number of samples in the training datasets (training size) and the number of features for each dataset used in our benchmarks.

where m is the number of samples, $\boldsymbol{x}_i \in \mathbb{R}^n$ ($n \in \mathbb{N}$ represents the number of features) is the ith sample, and $y_i \in \{-1, 1\}$ denotes the label (class) of the ith sample. The goal of the soft-margin SVM is to find a separating hyperplane

$$\boldsymbol{w}^T \boldsymbol{x} - b = 0,$$

where $\boldsymbol{w}$ is the normal of the hyperplane, and $\frac{b}{||\boldsymbol{w}||}$ determines the bias from the origin. Having the separating hyperplane, we can classify any sample $\boldsymbol{x}$ by a simple rule:

$$\text{If } \boldsymbol{w}^T \boldsymbol{x} + b \geq 0, \text{ then } \boldsymbol{x} \text{ belongs to Class A, else } \boldsymbol{x} \text{ belongs to Class B.}$$

See the illustration in Figure 3.8, left.

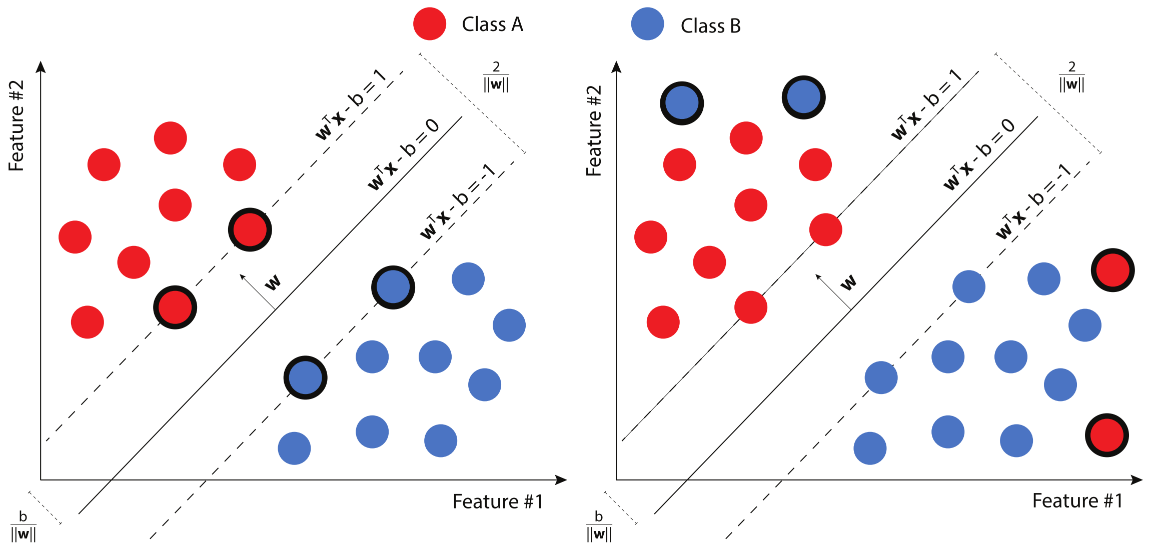

Figure 3.8: Illustration of a hyperplane separating the two classes with a maximal margin. The encircled samples are the support vectors (left) and the misclassified samples due to the penalty parameter (right).

The distance between the two classes for a given hyperplane is $\frac{2}{||w||}$. Therefore, to maximize the distance, we solve the following minimization problem

$$\underset{\boldsymbol{w}, b, \xi_i}{\arg\min} \frac{1}{2} \boldsymbol{w}^T \boldsymbol{w} + C \sum_{i=1}^{m} \xi_i \quad \text{s.t.} \quad \begin{cases} y_i \left(\boldsymbol{w}^T \boldsymbol{x}_i - b \right) \geq 1 - \xi_i, \\ \xi_i \geq 0, \end{cases}$$

where we penalize misclassified samples by the penalty parameter C and obtain a so-called soft-margin SVM primal formulation. See illustration Figure 3.8, right.

The bias $\boldsymbol{b}$ can be incorporated into $\boldsymbol{w}$ by augmenting samples by one dimension in the following way

$$\widehat{\boldsymbol{w}} = \begin{pmatrix} \boldsymbol{w} \\ b \end{pmatrix}, \quad \widehat{\boldsymbol{x}_i} = \begin{pmatrix} \boldsymbol{x}_i \\ 1 \end{pmatrix}.$$

Then dualizing (Section 2.5) the above minimization problem with the augmented variables, we obtain the (relaxed bias) dual soft-margin SVM formulation

$$\arg\min_{\boldsymbol{\alpha}} \ \frac{1}{2}\boldsymbol{\alpha}^T\boldsymbol{Y}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{Y}\boldsymbol{\alpha} - \boldsymbol{\alpha}^T\boldsymbol{e} \quad \text{s.t.} \quad \boldsymbol{o} \le \boldsymbol{\alpha} \le C\boldsymbol{e},$$

where $\boldsymbol{e} = (1, \dots, 1)^T$, $\boldsymbol{X} = (\widehat{\boldsymbol{x}_1}, \dots, \widehat{\boldsymbol{x}_m})$, and $\boldsymbol{Y} = \operatorname{diag}(y_1, \dots, y_m)$. This final formulation is a box-constrained QP with an SPS Hessian.

The normal vector is reconstructed by

$$\boldsymbol{w} = \sum_{i=1}^{m} \boldsymbol{\alpha}_i y_i \boldsymbol{x}_i,$$

while the bias is reconstructed by

$$b = \frac{1}{|I^{SV}|} \sum_{i \in I^{SV}} \left(\boldsymbol{x}_i^T \boldsymbol{w} - y_i \right),$$

where $I^{SV} = \{i \mid \alpha_i > 0,\ i = 1, 2,\ \dots, m\}$, and $\left|I^{SV}\right|$ is the cardinality of I^{SV}.

Chapter 4

Unconstrained Quadratic Programming

Minimization of an unconstrained QP problem

$$\arg\min_{\boldsymbol{x}} \frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} - \boldsymbol{x}^T\boldsymbol{b} \tag{4.1}$$

is equivalent to the solution of a *system of linear equations*

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{4.2}$$

due to the stationarity KKT condition (Section 2.4.1).

The solution of systems of linear equations plays a prominent role in many problems across various disciplines, e.g., economics and engineering. Therefore, it is not surprising that Krylov subspace methods, one of the most successful classes of methods for solving large systems of linear equations, have been named among the "Top 10 Algorithms of the 20th Century" [45].

In this chapter, we first discuss the steepest descent method, which is the basis for many algorithms introduced in the latter parts of this thesis. Then we derive the conjugate gradient method, which is perhaps the most well-known representative of the Krylov subspace methods. Finally, a deflation preconditioner for accelerating the Krylov subspace methods is discussed.

The author's main result in this section is a general multilevel deflation preconditioner known as PCDEFLATION [46], which is part of PETSc.

4.1 Steepest Descent Methods

The steepest descent method is based on the line search procedure

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k. \tag{4.3}$$

To ensure that the iterates decrease the cost function, i.e., $f(\boldsymbol{x}_{k+1}) < f(\boldsymbol{x}_k)$, we need to choose an appropriate search direction and step length. The direction of the most rapid decrease, i.e., the direction giving the *steepest descent*, is the direction of the negative gradient

$$\boldsymbol{d_k} = -\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k) = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k,$$

which, of course, verifies the condition on the decrease direction (Section 2.6). The negative gradient is commonly known as the *residual* $\boldsymbol{r}_k = \boldsymbol{d}_k = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k$. Rewriting Equation (4.3) using residuals as the descent directions yields

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{r}_k.$$

Scaling the previous equation by $-\boldsymbol{A}$ and adding $\boldsymbol{b}$ to both sides, we obtain the residual recurrence

$$\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k \boldsymbol{A}\boldsymbol{r}_k.$$

The step length α_k is obtained by minimizing the cost function

$$\arg\min_{\alpha_k} f(\boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k).$$

Using the necessary condition for extrema, we have

$$0 = \frac{\partial}{\partial \alpha_k} f(\boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k) = \boldsymbol{d}_k^T \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k) = \boldsymbol{d}_k^T \left(\boldsymbol{A}\left(\boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k\right) - \boldsymbol{b}\right),$$

therefore

$$\alpha_k = \frac{\boldsymbol{d}_k^T \boldsymbol{r}_k}{\boldsymbol{d}_k^T \boldsymbol{A}\boldsymbol{d}_k}, \tag{4.4}$$

and we have also found that the descent direction $\boldsymbol{d}_k = \boldsymbol{r}_k$ is orthogonal to the gradient $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_{k+1}) = -\boldsymbol{r}_{k+1}$. See Figure 4.1, left, for an illustration, and notice that the orthogonality of the descent directions means that we might minimize in the same direction more than once.

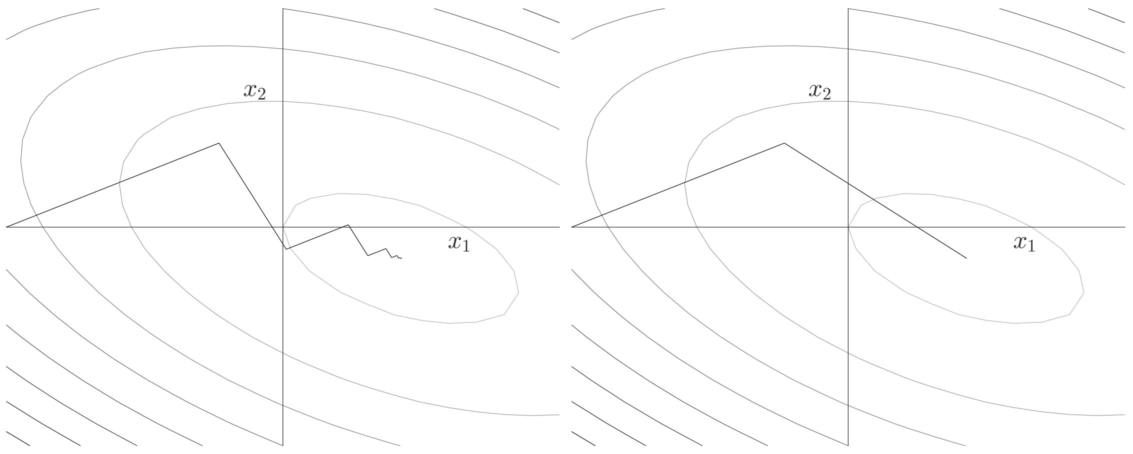

Figure 4.1: Contour plots of $f(\boldsymbol{x})$ for an SPD matrix of dimension $n = 2$, with plotted steps of the steepest descent method (left) and the CG method (right), which is introduced in the next section.

The stopping criterion is typically the reduction of the norm of the gradient

$$||\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{b}|| < \epsilon,$$

the square of which we conveniently compute as the numerator in the step length computation.

We can summarize the previous observations in Algorithm 4.1.

Algorithm 4.1: Steepest descent method

```
   Input: $\boldsymbol{A}$, $\boldsymbol{x}_0$, $\boldsymbol{b}$
1  $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$
2  for $k = 0, \cdots$:
3      $\boldsymbol{s} = \boldsymbol{A}\boldsymbol{r}_k$
4      $\alpha_k = \left(\boldsymbol{r}_k^T \boldsymbol{r}_k\right) / \left(\boldsymbol{s}^T \boldsymbol{r}_k\right)$
5      $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{r}_k$
6      $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k \boldsymbol{s}$
   Output: $\boldsymbol{x}_k$
```

4.1.1 Alternative Step Lengths

While the steepest descent step length is optimal in the sense that it yields the largest decrease in the cost function value, a shorter (or longer) step length can yield quicker convergence because it can avoid the zigzag pattern illustrated in the left part of Figure 4.1. This alternating pattern is essentially present even for an SPD Hessian with dimensions greater than two. In [47, 48], it was shown that the steepest descent method asymptotically alternates between two directions corresponding to the eigenvectors associated with the smallest and largest eigenvalues of the Hessian[1]. The alternative step lengths presented in [49] not only avoid the alternating pattern but also try to approximate the second-order information by the step length.

Writing the Taylor expansion for our QP problem

$$f(\boldsymbol{x} + \boldsymbol{d}) = f(\boldsymbol{x}) + \nabla_{\boldsymbol{x}} f(\boldsymbol{x})^T \boldsymbol{d} + \frac{1}{2}\boldsymbol{d}^T \boldsymbol{A}\boldsymbol{d},$$

the optimal $\boldsymbol{d}$ is obtained from the stationarity condition

$$\boldsymbol{o} = \frac{\partial}{\partial \boldsymbol{d}} f(\boldsymbol{x} + \boldsymbol{d}) = \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) + \boldsymbol{A}\boldsymbol{d}.$$

Then for an SPD Hessian $\boldsymbol{A}$, the optimal direction is

$$\boldsymbol{d} = -\boldsymbol{A}^{-1} \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$$

and we have found the relation for the Newton method

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{d} = \boldsymbol{x}_k - \boldsymbol{A}^{-1} \nabla_{\boldsymbol{x}} f(\boldsymbol{x}).$$

[1]When the initial gradient is not equal to a nonzero multiple of an eigenvector, in which case the steepest descent method converges in a single iteration

The Quasi-Newton method replaces the inverse of the Hessian in the above equation with some approximation. One such approximation can be obtained from the secant equation

$$\boldsymbol{A}(\boldsymbol{x}_k - \boldsymbol{x}_{k-1}) = \boldsymbol{A}(\boldsymbol{x}_k - \boldsymbol{b} - \boldsymbol{x}_{k-1} + \boldsymbol{b}) = \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k) - \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_{k-1}).$$

Approximating the Hessian $\boldsymbol{A}$ by $(\alpha_k \boldsymbol{I})^{-1}$, we can obtain α_k by minimizing the above equation either as

$$\arg\min_{\alpha_k^{\text{BB1}}} ||\alpha_k^{-1} \boldsymbol{s}_{k-1} - \boldsymbol{y}_{k-1}||$$

or

$$\arg\min_{\alpha_k^{\text{BB2}}} ||\boldsymbol{s}_{k-1} - \alpha_k \boldsymbol{y}_{k-1}||,$$

where $\boldsymbol{s}_{k-1} = \boldsymbol{x}_k - \boldsymbol{x}_{k-1}$ and $\boldsymbol{y}_{k-1} = \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k) - \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_{k-1})$. Carrying out the minimization gives, respectively, two *Barzilai-Borwein* step length rules [49]

$$\alpha_k^{\text{BB1}} = \frac{||\boldsymbol{s}_{k-1}||^2}{\boldsymbol{s}_{k-1}^T \boldsymbol{y}_{k-1}},$$

and

$$\alpha_k^{\text{BB2}} = \frac{\boldsymbol{s}_{k-1}^T \boldsymbol{y}_{k-1}}{||\boldsymbol{y}_{k-1}||^2}. \tag{4.5}$$

Further modifications of these step lengths are used in the spectral projected gradient method in Section 5.2.3.1.

4.2 Conjugate Gradient Method

4.2.1 Minimization over Subspace

Another option to escape the alternating directions in the steepest descent method is to define the step direction using the information obtained from the previous steps, thereby avoiding repeated minimization in the same direction. In order to achieve this, we will present a method that, in a single iteration, will find the minimizer over some subspace and a way to extend this subspace for the next iteration. Formally, we would like to create a nested sequence of subspaces

$$\mathcal{S}_1 \subset \mathcal{S}_2 \subset \cdots \subset \mathbb{R}^n,$$

where $\dim(\mathcal{S}_k) = k$. Now, given an initial guess $\boldsymbol{x}_0$ for each k, we will solve

$$\arg\min_{\boldsymbol{x} \in \boldsymbol{x}_0 + \mathcal{S}_k} f(\boldsymbol{x}).$$

Because the subspaces are expanding, we will obtain better and better approximations of the solution $\boldsymbol{x}$, and after n steps, the exact solution. However, we would like to obtain a good approximation far sooner than after n steps. Therefore, we need to construct subspaces $\mathcal{S}_k$ so that $f(\boldsymbol{x})$ decreases quickly.

Since the subspaces are nested, we know that $\mathcal{S}_{k+1}$ contains the previous minimizer $\boldsymbol{x}_k$. Moreover, we already know that the objective function $f(\boldsymbol{x})$ decreases most rapidly in the direction of the negative gradient. Therefore, it seems reasonable to extend $\mathcal{S}_k$ into $\mathcal{S}_{k+1}$ by the gradient $\boldsymbol{g}_k = \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k)$. This choice makes the next approximation $\boldsymbol{x}_{k+1}$ at least as good as the one that the steepest descent method would make.

It follows that $\mathcal{S}_1 = \operatorname{span}\{\boldsymbol{g}_0\}$. Now, as discussed above, we extend the space by $\boldsymbol{g}_1$, i.e., $\mathcal{S}_2 = \operatorname{span}\{\boldsymbol{g}_0, \boldsymbol{g}_1\}$. It turns out we can rewrite this slightly because

$$\boldsymbol{g}_1 = \boldsymbol{A}\boldsymbol{x}_1 - \boldsymbol{b} = \boldsymbol{A}\boldsymbol{x}_1 - \boldsymbol{A}\boldsymbol{x}_0 + \boldsymbol{g}_0 = \boldsymbol{A}(\boldsymbol{x}_0 + \alpha\boldsymbol{g}_0) - \boldsymbol{A}\boldsymbol{x}_0 + \boldsymbol{g}_0 \in \operatorname{span}\{\boldsymbol{g}_0, \boldsymbol{A}\boldsymbol{g}_0\},$$

where $\alpha \in \mathbb{R}$. We used the fact that $\boldsymbol{x}_1$ is the minimizer on $\boldsymbol{x}_0 + \operatorname{span}\{\boldsymbol{g}_0\} = \boldsymbol{x}_0 + \alpha\boldsymbol{g}_0$. Similarly, for the next space, we have

$$\begin{aligned}\boldsymbol{g}_2 &= \boldsymbol{A}\boldsymbol{x}_2 - \boldsymbol{b} = \boldsymbol{A}\boldsymbol{x}_2 - \boldsymbol{A}\boldsymbol{x}_1 + \boldsymbol{g}_1\\ &= \boldsymbol{A}(\boldsymbol{x}_0 + \alpha\boldsymbol{g}_0 + \beta\boldsymbol{A}\boldsymbol{g}_0) - \boldsymbol{A}(\boldsymbol{x}_0 + \gamma\boldsymbol{g}_0) + \boldsymbol{g}_1 \in \operatorname{span}\{\boldsymbol{g}_0, \boldsymbol{A}\boldsymbol{g}_0, \boldsymbol{A}^2\boldsymbol{g}_0\},\end{aligned}$$

where $\alpha, \beta, \gamma \in \mathbb{R}$. By repeating this process, we obtain that

$$\mathcal{S}_{k+1} = \operatorname{span}\{\boldsymbol{g}_0, \boldsymbol{g}_1, \boldsymbol{g}_2, \dots, \boldsymbol{g}_k\} = \operatorname{span}\{\boldsymbol{g}_0, \boldsymbol{A}\boldsymbol{g}_0, \boldsymbol{A}^2\boldsymbol{g}_0, \dots, \boldsymbol{A}^k\boldsymbol{g}_0\} = \mathcal{K}_{k+1}(\boldsymbol{A}, \boldsymbol{g}_0). \quad (4.6)$$

The space $\mathcal{K}_{k+1}(\boldsymbol{A}, \boldsymbol{g}_0) = \operatorname{span}\{\boldsymbol{g}_0, \boldsymbol{A}\boldsymbol{g}_0, \boldsymbol{A}^2\boldsymbol{g}_0, \dots, \boldsymbol{A}^k\boldsymbol{g}_0\}$ is called the *Krylov subspace*. Thus, we found that we can hope for a faster convergence than that of the steepest descent method by minimizing the functional (4.1) in the Krylov subspaces.

This introduction was taken from the author's Master's thesis [50] and is, in turn, based on [51].

4.2.2 Conjugate Gradient Method

This section derives the CG method using some ideas from [9] to tie in the idea of the previous section.

The CG method uses the same line search procedure as the steepest descent method

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k\boldsymbol{p}_k, \quad (4.7)$$

with the optimal line search step length given by Equation (4.4).

However, instead of taking residuals as the descent direction, we require that our directions $\boldsymbol{p}_k$ are mutually $\boldsymbol{A}$-*conjugate*, which means

$$\boldsymbol{p}_i^T\boldsymbol{A}\boldsymbol{p}_j = 0, \quad \forall\, i \le k \wedge j \le k \quad \text{s.t.} \quad i \ne j.$$

Such vectors are also known as $\boldsymbol{A}$-*orthogonal*. Writing a linear combination of a set of mutually $\boldsymbol{A}$-conjugate vectors

$$\boldsymbol{p}_k = \alpha_0\boldsymbol{p}_0 + \cdots + \alpha_{k-1}\boldsymbol{p}_{k-1},$$

then multiplying by $\boldsymbol{p}_k^T\boldsymbol{A}$ and using $\boldsymbol{A}$-orthogonality between $\boldsymbol{p}_k$ and all $\boldsymbol{p}_j$, $j=0,\dots,k-1$, we have

$$\boldsymbol{p}_k^T\boldsymbol{A}\boldsymbol{p}_k = \alpha_0\boldsymbol{p}_k^T\boldsymbol{A}\boldsymbol{p}_0 + \cdots + \alpha_{k-1}\boldsymbol{p}_k^T\boldsymbol{A}\boldsymbol{p}_{k-1} = 0.$$

Thus, a set of mutually $\boldsymbol{A}$-conjugate vectors is linearly independent.

We can build the set of $\boldsymbol{A}$-conjugate directions using the Gram-Schmidt procedure to $\boldsymbol{A}$-orthogonalize the current residual against all previous search directions to define the new search direction

$$\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \sum_{j=0}^{k}\beta_{k+1}^{j}\boldsymbol{p}_j, \tag{4.8}$$

provided that the residuals are linearly independent, which we will verify later. Multiplying the last equation by $\boldsymbol{p}_i^T\boldsymbol{A}$ for each $i=0,\dots,k$, we get from the mutual $\boldsymbol{A}$-conjugacy that

$$\boldsymbol{p}_i^T\boldsymbol{A}\boldsymbol{p}_{k+1} = \boldsymbol{p}_i^T\boldsymbol{A}\boldsymbol{r}_{k+1} + \sum_{j=0}^{k}\beta_{k+1}^{j}\boldsymbol{p}_i^T\boldsymbol{A}\boldsymbol{p}_j = 0,$$

and then

$$\beta_{k+1}^{j} = -\frac{\boldsymbol{p}_j^T\boldsymbol{A}\boldsymbol{r}_k}{\boldsymbol{p}_j^T\boldsymbol{A}\boldsymbol{p}_j} = -\frac{\boldsymbol{r}_k^T\boldsymbol{A}\boldsymbol{p}_j}{\boldsymbol{p}_j^T\boldsymbol{A}\boldsymbol{p}_j}. \tag{4.9}$$

Since α_k is chosen to minimize the cost function

$$\arg\min_{\alpha_k} f(\boldsymbol{x}_k + \alpha_k\boldsymbol{p}_k),$$

then for all i

$$\left.\frac{\partial f(\boldsymbol{x}_i + \alpha\boldsymbol{p}_i)}{\partial\alpha}\right|_{\alpha=\alpha_i} = \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_{i+1})^T\boldsymbol{p}_i = 0.$$

Moreover, for $i=0,\dots,k-1$

$$\begin{aligned}\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_{k+1})^T\boldsymbol{p}_i &= \boldsymbol{p}_i^T\left(\boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{b}\right) = \boldsymbol{p}_i^T\boldsymbol{A}\left(\boldsymbol{x}_{i+1} + \sum_{j=i+1}^{k}\alpha_j\boldsymbol{p}_j\right) - \boldsymbol{p}_i^T\boldsymbol{b}\\ &= \boldsymbol{p}_i^T\boldsymbol{A}\boldsymbol{x}_{i+1} - \boldsymbol{p}_i^T\boldsymbol{b} = \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_{i+1})^T\boldsymbol{p}_i,\end{aligned}$$

which combined with the previous equality gives

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_{k+1})^T\boldsymbol{p}_i = 0, \quad \forall i=0,\dots,k. \tag{4.10}$$

Using the previous result, we have

$$\left.\frac{\partial f(\boldsymbol{x}_0 + \gamma_0\boldsymbol{p}_0 + \cdots + \gamma_k\boldsymbol{p}_k)}{\partial\gamma_i}\right|_{\substack{\gamma_i=\alpha_j\\ j=0,\dots,k}} = 0, \quad \forall i=0,\dots,k.$$

and therefore the cost function is minimized over an ever-expanding subspace

$$\arg\min_{\boldsymbol{x}\in\boldsymbol{x}_0+\mathcal{S}_{k+1}^{*}} f(\boldsymbol{x}),$$

where $\mathcal{S}_{k+1}^* = \{\boldsymbol{p}_0, \dots, \boldsymbol{p}_k\}$. Finally, we have, by construction from Equation (4.8), that $\boldsymbol{r}_k$ is a linear combination of the directions $\boldsymbol{p}_0, \dots, \boldsymbol{p}_k$. Therefore,

$$\boldsymbol{r}_0, \dots, \boldsymbol{r}_k \text{ and } \boldsymbol{p}_0, \dots, \boldsymbol{p}_k \text{ span the same subspace.}$$

Moreover, if $\boldsymbol{r}_k = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k$ is not zero (otherwise, we have found the solution), we have from Equation (4.10) that

$$\boldsymbol{r}_k \text{ is orthogonal to } \boldsymbol{p}_0, \dots, \boldsymbol{p}_{k-1},$$

which by previous observation means

$$\boldsymbol{r}_k \text{ is orthogonal to } \boldsymbol{r}_0, \dots, \boldsymbol{r}_{k-1},$$

and therefore $\boldsymbol{r}_0, \dots, \boldsymbol{r}_k$ are linearly independent, which was the missing piece for the Gram-Schmidt process to be valid. Furthermore, the discussion shows that the subspace we minimize over $\mathcal{S}_k^*$ is precisely the Krylov subspace $\mathcal{K}_k$ from the previous section.

To give the CG method in its common format, we find alternative formulas for the step lengths α and β. First, multiplying Equation (4.8) by $\boldsymbol{A}\boldsymbol{p}_{k+1}$ and using the $\boldsymbol{A}$-conjugacy of the search directions, we obtain an identity

$$\boldsymbol{p}_{k+1}^T \boldsymbol{A}\boldsymbol{p}_{k+1} = \boldsymbol{p}_{k+1}^T \boldsymbol{A}\boldsymbol{r}_{k+1}. \tag{4.11}$$

The difference of subsequent residuals is, by the residual definition and Equation (4.7),

$$\boldsymbol{r}_{j+1} - \boldsymbol{r}_j = -\boldsymbol{A}\left(\boldsymbol{x}_{j+1} - \boldsymbol{x}_j\right) = -\alpha_j \boldsymbol{A}\boldsymbol{p}_j.$$

Multiplying by $\boldsymbol{r}_{k+1}$, we have from the orthogonality of the residuals that

$$-\alpha_j \boldsymbol{r}_{k+1}^T \boldsymbol{A}\boldsymbol{p}_j = \boldsymbol{r}_{k+1}^T \left(\boldsymbol{r}_{j+1} - \boldsymbol{r}_j\right) = \begin{cases} 0, & \text{if } j = 0, \dots, k-1 \\ \boldsymbol{r}_{k+1}^T \boldsymbol{r}_{k+1}, & \text{if } j = k. \end{cases} \tag{4.12}$$

Since the numerator in β_{k+1}^j is nonzero only for $j = k$, we simplify the notation $\beta_{k+1}^j = \beta_{k+1}$. As in Section 4.1, we can obtain the residual recurrence

$$\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k \boldsymbol{A}\boldsymbol{p}_k.$$

Multiplying the residual recurrence by $\boldsymbol{r}_k$, we get from the orthogonality of residuals and the identity (4.11) that

$$\boldsymbol{r}_k^T \boldsymbol{r}_k = \alpha_k \boldsymbol{r}_k^T \boldsymbol{A}\boldsymbol{p}_k = \alpha_k \boldsymbol{p}_k^T \boldsymbol{A}\boldsymbol{p}_k. \tag{4.13}$$

Substituting this equation, together with Equation (4.12), into Equation (4.9) allows us to compute β_{k+1} as

$$\beta_{k+1} = \frac{\boldsymbol{r}_{k+1}^T \boldsymbol{r}_{k+1}}{\boldsymbol{r}_k^T \boldsymbol{r}_k}.$$

Furthermore, Equation (4.13) gives us an alternative formulation for the line search step length

$$\alpha_k = \frac{\boldsymbol{r}_k^T \boldsymbol{r}_k}{\boldsymbol{r}_k^T \boldsymbol{A}\boldsymbol{p}_k} = \frac{\boldsymbol{r}_k^T \boldsymbol{r}_k}{\boldsymbol{p}_k^T \boldsymbol{A}\boldsymbol{p}_k}.$$

Putting the derived formulas together yields the CG method illustrated in Algorithm 4.2.

Algorithm 4.2: CG method

```
Input: $\boldsymbol{A}$, $\boldsymbol{x}_0$, $\boldsymbol{b}$
1 $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$
2 $\boldsymbol{p}_0 = \boldsymbol{r}_0$
3 for $k = 0, \cdots$:
4     $\boldsymbol{s} = \boldsymbol{A}\boldsymbol{p}_k$
5     $\alpha_k = \left(\boldsymbol{r}_k^T\boldsymbol{r}_k\right) / \left(\boldsymbol{s}^T\boldsymbol{p}_k\right)$
6     $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k\boldsymbol{p}_k$
7     $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k\boldsymbol{s}$
8     $\beta_{k+1} = \left(\boldsymbol{r}_{k+1}^T\boldsymbol{r}_{k+1}\right) / \left(\boldsymbol{r}_k^T\boldsymbol{r}_k\right)$
9     $\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_{k+1}\boldsymbol{p}_k$
Output: $\boldsymbol{x}_k$
```

4.2.3 CG Method Convergence

The CG algorithm is constructed so that in every iteration it minimizes $f(\boldsymbol{x})$ over the Krylov subspace $\boldsymbol{x}_0 + \mathcal{K}_k(\boldsymbol{A}, \boldsymbol{r}_0)$. With the *error* of approximation defined as $\boldsymbol{\epsilon}_k = \boldsymbol{x}_k - \boldsymbol{x}^*$, we have

$$\begin{aligned} f(\boldsymbol{x}_k) &= \frac{1}{2}\boldsymbol{x}_k^T\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{x}_k^T\boldsymbol{b} = \frac{1}{2}\left(\boldsymbol{x}_k + \boldsymbol{x}^* - \boldsymbol{x}^*\right)^T\boldsymbol{A}\left(\boldsymbol{x}_k + \boldsymbol{x}^* - \boldsymbol{x}^*\right) - \boldsymbol{x}_k^T\boldsymbol{b} \\ &= \frac{1}{2}\boldsymbol{\epsilon}_k^T\boldsymbol{A}\boldsymbol{\epsilon}_k + \frac{1}{2}\boldsymbol{\epsilon}_k^T\boldsymbol{A}\boldsymbol{x}^* + \frac{1}{2}(\boldsymbol{x}^*)^T\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{x}_k^T\boldsymbol{b} = \frac{1}{2}\boldsymbol{\epsilon}_k^T\boldsymbol{A}\boldsymbol{\epsilon}_k - \frac{1}{2}(\boldsymbol{x}^*)^T\boldsymbol{A}\boldsymbol{x}^*, \end{aligned}$$

and since $f(\boldsymbol{x}^*) = -(\boldsymbol{x}^*)^T\boldsymbol{A}\boldsymbol{x}^*/2$, the equation can be rewritten as

$$f(\boldsymbol{x}_k) = \frac{1}{2}||\boldsymbol{\epsilon}_k||_{\boldsymbol{A}}^2 + f(\boldsymbol{x}^*).$$

Therefore, each iteration of the CG method minimizes the error in $\boldsymbol{A}$-norm over the subspace $\boldsymbol{x}_0 + \mathcal{K}_k(\boldsymbol{A}, \boldsymbol{r}_0)$. Since

$$\boldsymbol{\epsilon}_k = \boldsymbol{x}_k - \boldsymbol{x}^* \in -\boldsymbol{x}^* + \boldsymbol{x}_0 + \operatorname{span}\{\boldsymbol{r}_0, \ldots, \boldsymbol{A}^{k-1}\boldsymbol{r}_0\} = \boldsymbol{\epsilon}_0 + \operatorname{span}\{\boldsymbol{A}\boldsymbol{\epsilon}_0, \ldots, \boldsymbol{A}^k\boldsymbol{\epsilon}_0\},$$

the error term can be written as a linear combination

$$\boldsymbol{\epsilon}_k = \left(\boldsymbol{I} + \sum_{i=1}^{k}\phi_i\boldsymbol{A}^i\right)\boldsymbol{\epsilon}_0,$$

where $\phi_i \in \mathbb{R}$ are coefficients chosen by CG so that $||\boldsymbol{\epsilon}_k||_{\boldsymbol{A}}$ is minimized. Moreover, using a polynomial $P_k(\boldsymbol{A})$ of degree k that satisfies $P_k(\boldsymbol{O}) = \boldsymbol{I}$, we can rewrite the previous equation as

$$\boldsymbol{\epsilon}_k = P_k(\boldsymbol{A})\boldsymbol{\epsilon}_0. \tag{4.14}$$

Then the CG method in the kth iteration finds a polynomial $P_k(\boldsymbol{A})$ that minimizes the error term given by (4.14) in the $\boldsymbol{A}$-norm, i.e.,

$$||\boldsymbol{\epsilon}_k||_{\boldsymbol{A}} = \min_{P_k}||P_k(\boldsymbol{A})\boldsymbol{\epsilon}_0||_{\boldsymbol{A}}.$$

Given the $\lambda_{min} = \lambda_1, \dots, \lambda_n = \lambda_{max}$ eigenvalues of the Hessian $\boldsymbol{A}$ and their corresponding normalized eigenvectors $\boldsymbol{v}_1, \dots, \boldsymbol{v}_n$, the initial error term $\boldsymbol{\epsilon}_0$ can be expressed as a linear combination

$$\boldsymbol{\epsilon}_0 = \sum_{i=1}^{n} \xi_i \boldsymbol{v}_i,$$

which allows us to write (4.14) as

$$\boldsymbol{\epsilon}_k = P_k(\boldsymbol{A}) \sum_{i=1}^{n} \xi_i \boldsymbol{v}_i = \sum_{i=1}^{n} \xi_i P_k(\lambda_i) \boldsymbol{v}_i.$$

Using the orthonormality of the eigenvectors $\boldsymbol{v}_i$, we can write the square of the $\boldsymbol{A}$-norm of the error as

$$\begin{aligned} ||\boldsymbol{\epsilon}_k||_{\boldsymbol{A}}^2 = \min_{P_k} \sum_{i=1}^{n} \xi_i^2 \left(P_k(\lambda_i)\right)^2 \lambda_i &\leq \min_{P_k} \max_{\lambda \in \sigma(\boldsymbol{A})} \left(P_k(\lambda)\right)^2 \sum_{i=1}^{n} \xi_i^2 \lambda_i \\ &= \min_{P_k} \max_{\lambda \in \sigma(\boldsymbol{A})} \left(P_k(\lambda)\right)^2 ||\boldsymbol{\epsilon}_0||_{\boldsymbol{A}}^2. \end{aligned} \tag{4.15}$$

This result provides some insight into what constitutes a favorable spectrum of the Hessian $\boldsymbol{A}$. Assuming our initial guess $\boldsymbol{x}_0$ was not a solution, the error is zero if P_k is zero for each distinct eigenvalue. Tight clusters of eigenvalues or eigenvalues with high multiplicity are (essentially) reduced by the same P_k, which, in practice, saves a number of iterations. The eigenvalues close to zero are problematic because of the restriction $P_k(0) = 1$. On the other hand, if the Hessian $\boldsymbol{A}$ is only positive semidefinite and the linear system is consistent (the right-hand side $\boldsymbol{b}$ is in the range of $\boldsymbol{A}$), we can still use CG for the solution of such a system [52]. Moreover, the zero eigenvalues, and therefore the null space of the Hessian, are ignored.

It follows from (4.15) that the relative error in the $\boldsymbol{A}$-norm can be bounded by

$$\frac{||\boldsymbol{\epsilon}_k||_{\boldsymbol{A}}}{||\boldsymbol{\epsilon}_0||_{\boldsymbol{A}}} \leq \min_{P_k} \max_{\lambda} |P_k(\lambda)| \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^k,$$

where $\kappa = \lambda_{max}/\lambda_{min}$ is called the condition number of $\boldsymbol{A}$. The last inequality is derived by bounding the value of $\max_\lambda |P_k(\lambda)|$ by the kth scaled and shifted Chebyshev polynomial on the $[\lambda_{min}, \lambda_{max}]$ interval; see, e.g., [53] for a proof. Bear in mind that the previous result is only an upper bound, i.e., having two Hessians with one being much better conditioned than the other, there is no guarantee that the CG will converge faster for the Hessian that is better conditioned. For example, the CG method applied to a badly conditioned problem with only two distinct eigenvalues will converge in at most two iterations, while if the better-conditioned Hessian has many distinct eigenvalues, the CG method may require many iterations.

4.2.4 Preconditioned CG Method

As was shown in the previous section, the speed of convergence depends on the spectral properties of the Hessian $\boldsymbol{A}$. Therefore, suitably improving the spectrum of the Hessian will improve the convergence of the CG method. The improvement is achieved by *preconditioning*, which consists of scaling the system of linear equations by the preconditioner $\boldsymbol{M}^{-1}$

$$\boldsymbol{M}^{-1}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{M}^{-1}\boldsymbol{b}, \tag{4.16}$$

where $\boldsymbol{M}$ is, in the case of the CG method, an SPD matrix [9, 51]. Note that if $\boldsymbol{M}^{-1} = \boldsymbol{A}^{-1}$, then by the previous equation $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$, and we have found the solution. Finding the inverse of $\boldsymbol{A}$ is, however, costly. Indeed, $\boldsymbol{M}^{-1}$ is often available only as an action on a vector, which may even be realized by an iterative method. Therefore, we often refer to $\boldsymbol{M}^{-1}$ as the application or action of the preconditioner on a vector or matrix. Overviews of general preconditioners can be found, e.g., in [51, 54, 55]. Of course, using specialized preconditioners for the given problem usually yields better results. Examples of these are the preconditioners for the Darcy flow problem [56] and 3×3 block matrices [57], which are co-authored by the thesis author, and the preconditioners for FETI methods described in Section 7.1.3.

The issue with the preconditioned system (4.16) is that $\boldsymbol{M}^{-1}\boldsymbol{A}$ is not generally an SPD matrix. This problem can be circumvented because, for every square SPD matrix $\boldsymbol{M}$, there exists a Cholesky factorization $\boldsymbol{M} = \boldsymbol{L}\boldsymbol{L}^T$. Using the Cholesky factorization, the system (4.2) can be transformed into

$$\boldsymbol{L}^{-1}\boldsymbol{A}\boldsymbol{L}^{-T}\widetilde{\boldsymbol{x}} = \boldsymbol{L}^{-1}\boldsymbol{b}, \quad \widetilde{\boldsymbol{x}} = \boldsymbol{L}^T\boldsymbol{x},$$

which is known as split or symmetric preconditioning [55]. Furthermore, the preconditioned Hessian $\boldsymbol{L}^{-1}\boldsymbol{A}\boldsymbol{L}^{-T}$ is SPD and has the same eigenvalues as $\boldsymbol{M}^{-1}\boldsymbol{A}$.

Denoting $\widetilde{\boldsymbol{A}} = \boldsymbol{L}^{-1}\boldsymbol{A}\boldsymbol{L}^{-T}$ and $\widetilde{\boldsymbol{b}} = \boldsymbol{L}^{-1}\boldsymbol{b}$, we can solve the system

$$\widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{x}} = \widetilde{\boldsymbol{b}}$$

with the CG method to find $\widetilde{\boldsymbol{x}}$, and then the solution $\boldsymbol{x}$ for the original problem. However, we would have to factorize the preconditioning matrix $\boldsymbol{M}$ to obtain the factor $\boldsymbol{L}$. Fortunately, it turns out that this is not necessary. Let us write the update formulas for the modified system

$$\begin{aligned}
\alpha_k &= \left(\widetilde{\boldsymbol{r}}_k^T\widetilde{\boldsymbol{r}}_k\right) / \left(\widetilde{\boldsymbol{p}}_k^T\widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{p}}_k\right), \\
\widetilde{\boldsymbol{x}}_{k+1} &= \widetilde{\boldsymbol{x}}_k + \alpha_k\widetilde{\boldsymbol{p}}_k, \\
\widetilde{\boldsymbol{r}}_{k+1} &= \widetilde{\boldsymbol{r}}_k - \alpha_k\widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{p}}_k, \\
\beta_{k+1} &= \left(\widetilde{\boldsymbol{r}}_{k+1}^T\widetilde{\boldsymbol{r}}_{k+1}\right) / \left(\widetilde{\boldsymbol{r}}_k^T\widetilde{\boldsymbol{r}}_k\right), \\
\widetilde{\boldsymbol{p}}_{k+1} &= \widetilde{\boldsymbol{r}}_{k+1} + \beta_{k+1}\widetilde{\boldsymbol{p}}_k.
\end{aligned}$$

Using $\boldsymbol{x}_k = \boldsymbol{L}^{-T}\widetilde{\boldsymbol{x}}_k$ and the definition of $\widetilde{\boldsymbol{b}}$, the residual can be rewritten as

$$\widetilde{\boldsymbol{r}}_k = \widetilde{\boldsymbol{b}} - \widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{x}}_k = \boldsymbol{L}^{-1}\left(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k\right) = \boldsymbol{L}^{-1}\boldsymbol{r}_k.$$

Substituting this into our update formulas and using the definition of $\widetilde{\boldsymbol{A}}$, we get

$$\begin{aligned}
\alpha_k &= \left(\boldsymbol{r}_k^T \boldsymbol{M}^{-1}\boldsymbol{r}_k\right) / \left(\left(\boldsymbol{L}^{-T}\widetilde{\boldsymbol{p}}_k\right)^T \boldsymbol{A}\left(\boldsymbol{L}^{-T}\widetilde{\boldsymbol{p}}_k\right)\right), \\
\boldsymbol{L}^T\boldsymbol{x}_{k+1} &= \boldsymbol{L}^T\boldsymbol{x}_k + \alpha_k\widetilde{\boldsymbol{p}}_k, \\
\boldsymbol{L}^{-1}\boldsymbol{r}_{k+1} &= \boldsymbol{L}^{-1}\boldsymbol{r}_k - \alpha_k\boldsymbol{L}^{-1}\boldsymbol{A}\boldsymbol{L}^{-T}\widetilde{\boldsymbol{p}}_k, \\
\beta_{k+1} &= \left(\boldsymbol{r}_{k+1}^T\boldsymbol{M}^{-1}\boldsymbol{r}_{k+1}\right) / \left(\boldsymbol{r}_k^T\boldsymbol{M}^{-1}\boldsymbol{r}_k\right), \\
\widetilde{\boldsymbol{p}}_{k+1} &= \boldsymbol{L}^{-1}\boldsymbol{r}_{k+1} + \beta_{k+1}\widetilde{\boldsymbol{p}}_k.
\end{aligned}$$

Finally, substituting $\boldsymbol{z}_k = \boldsymbol{M}^{-1}\boldsymbol{r}_k$ and $\boldsymbol{p}_k = \boldsymbol{L}^{-T}\widetilde{\boldsymbol{p}}_k$ into the derived formulas yields the preconditioned conjugate gradient (PCG) method illustrated in Algorithm 4.3, which only requires the action of the preconditioner $\boldsymbol{M}^{-1}$.

Algorithm 4.3: PCG method

```
   Input: $A$, $M^{-1}$, $x_0$, $b$
 1 $r_0 = b - Ax_0$
 2 $z_0 = M^{-1}r_0$
 3 $p_0 = z_0$
 4 for $k = 0, \cdots$:
 5     $s = Ap_k$
 6     $\alpha_k = \left(r_k^T z_k\right) / \left(s^T p_k\right)$
 7     $x_{k+1} = x_k + \alpha_k p_k$
 8     $r_{k+1} = r_k - \alpha_k s$
 9     $z_{k+1} = M^{-1} r_{k+1}$
10     $\beta_{k+1} = \left(r_{k+1}^T z_{k+1}\right) / \left(r_k^T z_k\right)$
11     $p_{k+1} = z_{k+1} + \beta_{k+1} p_k$
   Output: $x_k$
```

Algorithm: CG method

```
   Input: $A$, $x_0$, $b$
 1 $r_0 = b - Ax_0$
 2
 3 $p_0 = r_0$
 4 for $k = 0, \cdots$:
 5     $s = Ap_k$
 6     $\alpha_k = \left(r_k^T r_k\right) / \left(s^T p_k\right)$
 7     $x_{k+1} = x_k + \alpha_k p_k$
 8     $r_{k+1} = r_k - \alpha_k s$
 9
10     $\beta_{k+1} = \left(r_{k+1}^T r_{k+1}\right) / \left(r_k^T r_k\right)$
11     $p_{k+1} = r_{k+1} + \beta_{k+1} p_k$
   Output: $x_k$
```

4.3 Deflation Preconditioner

Deflation for the CG method, also known as CG with preconditioning by projectors, was introduced independently in [58–60]. The basic idea of deflation is to split the solution into a deflation subspace and its $\boldsymbol{A}$-conjugate complement. The solution in the deflation subspace is obtained directly, and the conjugate gradients are restricted to the $\boldsymbol{A}$-conjugate complement of the deflation subspace. The CG method can be sped up significantly by choosing the deflation space that contains the slowly converging parts of the solution.

4.3.1 Deriving Deflation Preconditioner

Let us define the *deflation matrix* as

$$\boldsymbol{W} = (\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_m) \in \mathbb{R}^{n\times m}, m < n.$$

Assuming that $\boldsymbol{W}$ is a full rank matrix and $\mathcal{W}$ is a subspace spanned by the columns of $\boldsymbol{W}$, we can denote a projector

$$\boldsymbol{P} = \boldsymbol{I} - \boldsymbol{W}\left(\boldsymbol{W}^T\boldsymbol{A}\boldsymbol{W}\right)^{-1}\boldsymbol{W}^T\boldsymbol{A} = \boldsymbol{I} - \boldsymbol{Q}\boldsymbol{A}$$

onto the $\boldsymbol{A}$-conjugate complement of $\mathcal{W}$.

Given an arbitrary initial guess $\boldsymbol{x}_{-1}$ and defining the residual $\boldsymbol{r}_{-1} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_{-1}$, we can choose $\boldsymbol{x}_0$ to be

$$\boldsymbol{x}_0 = \boldsymbol{x}_{-1} + \boldsymbol{W}\left(\boldsymbol{W}^T\boldsymbol{A}\boldsymbol{W}\right)^{-1}\boldsymbol{W}^T\boldsymbol{r}_{-1}. \tag{4.17}$$

Multiplying from the left by $\boldsymbol{W}^T\boldsymbol{A}$ yields

$$\begin{aligned}\boldsymbol{W}^T\boldsymbol{A}\boldsymbol{x}_0 &= \boldsymbol{W}^T\boldsymbol{A}\boldsymbol{x}_{-1} + \boldsymbol{W}^T\left(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_{-1}\right)\\ \boldsymbol{W}^T\boldsymbol{A}\boldsymbol{x}_0 &= \boldsymbol{W}^T\boldsymbol{b} && (4.18)\\ \boldsymbol{o} &= \boldsymbol{W}^T\boldsymbol{b} - \boldsymbol{W}^T\boldsymbol{A}\boldsymbol{x}_0 = \boldsymbol{W}^T\boldsymbol{r}_0. && (4.19)\end{aligned}$$

From (4.18), it follows that $\boldsymbol{x}_0$ is the exact solution of (4.2) in $\mathcal{W}$ and therefore (Equation (4.19)) $\boldsymbol{r}_0$ is orthogonal to $\mathcal{W}$. If we use $\boldsymbol{x}_0$ as the initial guess for CG, we obtain the InitCG method [61] illustrated in Algorithm 4.4.

Algorithm 4.4: InitCG method

```
   Input: $\boldsymbol{A}$, $\boldsymbol{x}_{-1}$, $\boldsymbol{b}$, $\boldsymbol{W}$
1  $\boldsymbol{r}_{-1} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_{-1}$
2  $\boldsymbol{x}_0 = \boldsymbol{x}_{-1} + \boldsymbol{Q}\boldsymbol{r}_{-1}$
3  $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$
4  $\boldsymbol{p}_0 = \boldsymbol{r}_0$
5  for $k = 0, \cdots$:
6      $\boldsymbol{s} = \boldsymbol{A}\boldsymbol{p}_k$
7      $\alpha_k = \left(\boldsymbol{r}_k^T\boldsymbol{r}_k\right) / \left(\boldsymbol{s}^T\boldsymbol{p}_k\right)$
8      $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k\boldsymbol{p}_k$
9      $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k\boldsymbol{s}$
10     $\beta_{k+1} = \left(\boldsymbol{r}_{k+1}^T\boldsymbol{r}_{k+1}\right) / \left(\boldsymbol{r}_k^T\boldsymbol{r}_k\right)$
11     $\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_{k+1}\boldsymbol{p}_k$
   Output: $\boldsymbol{x}_k$
```

If the columns of $\boldsymbol{W}$ are exact eigenvectors, then in exact arithmetic $\mathcal{W}$ is orthogonal to $\mathcal{K}_k(\boldsymbol{A}, \boldsymbol{r}_0)$ because the residuals will not have any components in the direction of the eigenvectors spanning $\mathcal{W}$. However, if $\boldsymbol{W}$ does not consist of the exact eigenvectors or the computations are done in finite precision, this relation does not hold, and some sort of correction needs to be employed.

The first problem is that $\boldsymbol{p}_0 = \boldsymbol{r}_0$ is not necessarily $\boldsymbol{A}$-orthogonal to $\mathcal{W}$. If this is the case, then $\boldsymbol{x}_1$ has components in $\mathcal{W}$. This is resolved by setting

$$\boldsymbol{p}_0 = \boldsymbol{P}\boldsymbol{r}_0.$$

Similarly, since $\boldsymbol{r}_{k+1} = \boldsymbol{r}_0 - \boldsymbol{A}\left(\alpha_0 \boldsymbol{p}_0 + \cdots + \alpha_k \boldsymbol{p}_k\right)$, we use the same trick as above, so that the update formula for the descent direction becomes

$$\boldsymbol{p}_{k+1} = \boldsymbol{P}\boldsymbol{r}_{k+1} + \beta_{k+1}\boldsymbol{p}_k.$$

Effectively, we are making the search direction $\boldsymbol{A}$-conjugate to $\mathcal{W}$ by projecting the components in $\mathcal{W}$ out of the residuals. This ensures that CG is not searching in $\mathcal{W}$, but only in its $\boldsymbol{A}$-conjugate complement. Thus, the required splitting of the solution is achieved. Setting the preconditioned residual of Section 4.2.4 to $\boldsymbol{z}_k = \boldsymbol{P}\boldsymbol{r}_{k+1}$, we get that our method is equivalent to the preconditioned CG method with the $\boldsymbol{A}$-conjugate projection $\boldsymbol{P}$ taken as the preconditioner [62]. This provides the values of α_k and β_k, but as shown in [50], it is not necessary to modify the values of α_k and β_k.

Modifying Algorithm 4.4 so that the search directions are explicitly $\boldsymbol{A}$-orthogonalized with respect to $\mathcal{W}$ gives us the deflated conjugate gradient (DCG) method, as shown in Algorithm 4.5.

Algorithm 4.5: DCG method

```
Input: $\boldsymbol{A}$, $\boldsymbol{x}_{-1}$, $\boldsymbol{b}$, $\boldsymbol{W}$
1  $\boldsymbol{P} = \boldsymbol{I} - \boldsymbol{Q}\boldsymbol{A}$
2  $\boldsymbol{r}_{-1} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_{-1}$
3  $\boldsymbol{x}_0 = \boldsymbol{x}_{-1} + \boldsymbol{Q}\boldsymbol{r}_{-1}$
4  $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$
5  $\boldsymbol{p}_0 = \boldsymbol{P}\boldsymbol{r}_0$
6  for $k = 0, \cdots$:
7      $\boldsymbol{s} = \boldsymbol{A}\boldsymbol{p}_k$
8      $\alpha_k = \left(\boldsymbol{r}_k^T\boldsymbol{r}_k\right) / \left(\boldsymbol{s}^T\boldsymbol{p}_k\right)$
9      $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k\boldsymbol{p}_k$
10     $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k\boldsymbol{s}$
11     $\beta_{k+1} = \left(\boldsymbol{r}_{k+1}^T\boldsymbol{r}_{k+1}\right) / \left(\boldsymbol{r}_k^T\boldsymbol{r}_k\right)$
12     $\boldsymbol{p}_{k+1} = \boldsymbol{P}\boldsymbol{r}_{k+1} + \beta_{k+1}\boldsymbol{p}_k$
Output: $\boldsymbol{x}_k$
```

Algorithm: CG method

```
Input: $\boldsymbol{A}$, $\boldsymbol{x}_0$, $\boldsymbol{b}$
1
2  $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$
3
4
5  $\boldsymbol{p}_0 = \boldsymbol{r}_0$
6  for $k = 0, \cdots$:
7      $\boldsymbol{s} = \boldsymbol{A}\boldsymbol{p}_k$
8      $\alpha_k = \left(\boldsymbol{r}_k^T\boldsymbol{r}_k\right) / \left(\boldsymbol{s}^T\boldsymbol{p}_k\right)$
9      $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k\boldsymbol{p}_k$
10     $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k\boldsymbol{s}$
11     $\beta_{k+1} = \left(\boldsymbol{r}_{k+1}^T\boldsymbol{r}_{k+1}\right) / \left(\boldsymbol{r}_k^T\boldsymbol{r}_k\right)$
12     $\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_{k+1}\boldsymbol{p}_k$
Output: $\boldsymbol{x}_k$
```

4.3.2 Preconditioned Deflated CG Method

An additional preconditioner can be incorporated into the previous algorithm. The derivation is done in the same way as in Section 4.2.4. Carrying this out yields the preconditioned DCG (PDCG) method, illustrated in Algorithm 4.6.

4.3.3 Preconditioning Effect of Deflation

In [50], it was shown that the deflated CG method is equivalent to solving a system with the Hessian $\boldsymbol{A}$ preconditioned by $\boldsymbol{P}$. Indeed, the following equivalence of spectra of operators was established:

$$\sigma(\boldsymbol{P}\boldsymbol{A}) = \sigma(\boldsymbol{P}^T\boldsymbol{A}) = \sigma(\boldsymbol{P}\boldsymbol{P}^T\boldsymbol{A}) = \sigma(\boldsymbol{P}^T\boldsymbol{A}\boldsymbol{P}).$$

Algorithm 4.6: PDCG method

```
   Input: $\boldsymbol{A}$, $\boldsymbol{x}_{-1}$, $\boldsymbol{b}$, $\boldsymbol{W}$
 1 $\boldsymbol{P} = \boldsymbol{I} - \boldsymbol{Q}\boldsymbol{A}$
 2 $\boldsymbol{r}_{-1} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_{-1}$
 3 $\boldsymbol{x}_0 = \boldsymbol{x}_{-1} + \boldsymbol{Q}\boldsymbol{r}_{-1}$
 4 $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$
 5 $\boldsymbol{z}_0 = \boldsymbol{M}^{-1}\boldsymbol{r}_0$
 6 $\boldsymbol{p}_0 = \boldsymbol{P}\boldsymbol{z}_0$
 7 for $k = 0, \cdots$:
 8     $\boldsymbol{s} = \boldsymbol{A}\boldsymbol{p}_k$
 9     $\alpha_k = \left(\boldsymbol{r}_k^T\boldsymbol{z}_k\right) / \left(\boldsymbol{s}^T\boldsymbol{p}_k\right)$
10     $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k\boldsymbol{p}_k$
11     $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k\boldsymbol{s}$
12     $\boldsymbol{z}_{k+1} = \boldsymbol{M}^{-1}\boldsymbol{r}_{k+1}$
13     $\beta_{k+1} = \left(\boldsymbol{r}_{k+1}^T\boldsymbol{z}_{k+1}\right) / \left(\boldsymbol{r}_k^T\boldsymbol{z}_k\right)$
14     $\boldsymbol{p}_{k+1} = \boldsymbol{P}\boldsymbol{z}_{k+1} + \beta_{k+1}\boldsymbol{p}_k$
   Output: $\boldsymbol{x}_k$
```

Assuming that the columns of the deflation matrix $\boldsymbol{W}$ are the exact eigenvectors of $\boldsymbol{A}$, it immediately follows that

$$\boldsymbol{P}^T\boldsymbol{A}\boldsymbol{W} = \boldsymbol{A}\boldsymbol{W} - \boldsymbol{A}\boldsymbol{W}\left(\boldsymbol{W}^T\boldsymbol{A}\boldsymbol{W}\right)^{-1}\boldsymbol{W}^T\boldsymbol{A}\boldsymbol{W} = \boldsymbol{O} = \mathrm{diag}(0,\ldots,0)\boldsymbol{W},$$

i.e., the columns of $\boldsymbol{W}$ are eigenvectors of $\boldsymbol{P}^T\boldsymbol{A}$ belonging to $\lambda = 0$ eigenvalues. Moreover, if λ and $\boldsymbol{v}$ are an eigenpair of $\boldsymbol{A}$ but $\boldsymbol{v}$ is not a column of $\boldsymbol{W}$, then we have $\boldsymbol{W}^T\boldsymbol{v} = \boldsymbol{o}$ and also $\boldsymbol{W}^T\boldsymbol{A}\boldsymbol{v} = \boldsymbol{o}$. Therefore,

$$\boldsymbol{P}^T\boldsymbol{A}\boldsymbol{v} = \boldsymbol{A}\boldsymbol{v} - \boldsymbol{A}\boldsymbol{W}\left(\boldsymbol{W}^T\boldsymbol{A}\boldsymbol{W}\right)^{-1}\boldsymbol{W}^T\boldsymbol{A}\boldsymbol{v} = \boldsymbol{A}\boldsymbol{v} = \lambda\boldsymbol{v}.$$

In other words, the DCG operator has the same spectrum as the Hessian $\boldsymbol{A}$, except that the eigenvalues belonging to eigenvectors comprising the deflation matrix $\boldsymbol{W}$ are shifted to zero. As was mentioned in Section 4.2.3, the CG method ignores the space spanned by the null space of the operator. This allows us to consider the effective condition number

$$\kappa_{eff}\left(\boldsymbol{P}^T\boldsymbol{A}\right) = \frac{\lambda_{max}}{\lambda_{min}},$$

where λ_{max} and λ_{min} are, respectively, the largest and the smallest nonzero eigenvalues of $\boldsymbol{P}^T\boldsymbol{A}$ or one of the spectrally equivalent operators.

4.3.4 Shifting the Eigenvalues

As shown in the previous section, if the deflation matrix $\boldsymbol{W}$ consists of the exact eigenvectors of the Hessian $\boldsymbol{A}$, then the associated eigenvalues are shifted to zero. However, if the deflated eigenvectors are only approximate, the associated eigenvalues might not be zeroed out completely but may instead be merely very small. The eigenvalues close to zero can significantly slow down the convergence, as mentioned in Section 4.2.3.

It was suggested in [63] that we can add a correction factor $\boldsymbol{Q}$ to the projector $\boldsymbol{P}$, leading to a so-called projector with coarse problem correction

$$\boldsymbol{P}_c = \boldsymbol{P} + c\boldsymbol{Q},$$

where $c \geq 0$. Replacing $\boldsymbol{P}$ in the DCG method with $\boldsymbol{P}_c$, we have, similarly to the above,

$$\begin{aligned}\boldsymbol{P}_s^T \boldsymbol{AW} &= \boldsymbol{AW} - \boldsymbol{AW}\left(\boldsymbol{W}^T\boldsymbol{AW}\right)^{-1}\boldsymbol{W}^T\boldsymbol{AW} + c\boldsymbol{W}\left(\boldsymbol{W}^T\boldsymbol{AW}\right)^{-1}\boldsymbol{W}^T\boldsymbol{AW}\\ &= c\boldsymbol{W} = \operatorname{diag}(c,\dots,c)\boldsymbol{W},\end{aligned}$$

and since $\boldsymbol{Q}$ is orthogonal to eigenvectors not in $\boldsymbol{W}$, we can show, in the same way as in the previous section, that the rest of the eigenvalues are not changed. Therefore, using $\boldsymbol{P}_c$ leads to the preconditioned operators having the same spectrum as the Hessian $\boldsymbol{A}$, except that the eigenvalues belonging to the columns of the deflation matrix $\boldsymbol{W}$ are shifted to the parameter c. Moreover, choosing the parameter c to coincide with an eigenvalue that is not deflated from the Hessian does not create an isolated eigenvalue in the deflated Hessian spectrum.

Since

$$\boldsymbol{P}_c = \boldsymbol{I} - \boldsymbol{QA} + \boldsymbol{Q} = \boldsymbol{I} - \boldsymbol{Q}\left(\boldsymbol{A} - \boldsymbol{I}\right),$$

the cost of applying $\boldsymbol{P}_c$ compared to $\boldsymbol{P}$ is one additional vector-vector addition.

The numerical experiments in [63] showed that $\boldsymbol{P}_c$ also has a stabilizing effect when the projections $\boldsymbol{P}$, and especially the application of the inverse in $\boldsymbol{P}$, are computed with low accuracy.

4.3.5 Deflation Coarse Problem

The inverse in the projector $\boldsymbol{P}$ is called the coarse problem (CP), while $\boldsymbol{W}^T\boldsymbol{AW}$ is called the coarse problem matrix. The coarse problem is usually solved by a direct solver.

Our implementation has a row-wise distribution of matrices. The CP matrix is assembled in parallel, and the rows of the CP matrix are distributed among the same number of cores used to solve the linear system. However, the dimension of the CP is smaller than the dimension of $\boldsymbol{A}$, quite often significantly (even just a few rows). If we tried to solve this problem by a fully parallel approach, the cost of communication, as well as the required time, could be extremely high. To solve this problem, the same strategy that was successfully used for the solution of the FETI method CP [64, 65] is employed to solve the deflation CP. First, an MPI sub-communicator is created, and then the whole CP matrix is distributed over the available MPI ranks in the sub-communicator. The CP matrix is then factorized on the sub-communicator. The forward and backward solves are performed by scattering the whole input vector into the sub-communicator. The result is then distributed among all cores in the global communicator. A heuristic chooses the size of the sub-communicator, controlling the partial parallelization.

4.3.6 Required Accuracy for the Coarse Problem Solution

While it was remarked that the CP is solved by a direct solver and, therefore, with full machine precision, it will prove useful to examine the accuracy level that is actually needed for the CP solution.

Given the relative tolerance ϵ for the outer iteration, the numerical experiments in [66] showed that to achieve the same convergence as that obtained by using a direct solver, the stopping criterion of the inner solver (CG) used for solving the CP has the form

$$||\boldsymbol{r}_k^{inner}|| \leq c\epsilon||\boldsymbol{b}^{inner}||, \qquad 0 < c \leq 1.$$

It was shown in [67], using the theory developed for the inexact Krylov subspace methods [68, 69], that this accuracy is needed only in the first few iterations of the outer solver and can be relaxed as we get closer to the solution of the original system. Their stopping criterion has the form of

$$||\boldsymbol{r}_k^{inner}|| \leq \frac{c\epsilon}{||\boldsymbol{r}_i^{outer}||}||\boldsymbol{b}^{inner}||, \qquad c > 0.$$

The constant c is guaranteed to exist. However, we do not know the upper bound, i.e., the value that would lead to the maximal relaxation of the stopping criterion while keeping the number of iterations required by the outer solver to converge the same as when the CP is solved directly. The DCG method that uses this stopping criterion is called the adaptive precision DCG method.

4.3.7 Multilevel Deflation

Given a hierarchy of the deflation matrices $\boldsymbol{W}_k$, $k \in \{1, \dots, n\}$ such that

$$\boldsymbol{W}_1^T \boldsymbol{A} \boldsymbol{W}_1 \tag{4.20}$$

is a coarse problem matrix, and

$$\boldsymbol{W}_2^T \boldsymbol{W}_1^T \boldsymbol{A} \boldsymbol{W}_1 \boldsymbol{W}_2$$

is even coarser. It is assumed that this hierarchy continues until the coarsest problem matrix reads

$$\boldsymbol{W}_n^T \cdots \boldsymbol{W}_2^T \boldsymbol{W}_1^T \boldsymbol{A} \boldsymbol{W}_1 \boldsymbol{W}_2 \cdots \boldsymbol{W}_n.$$

Assume that DCG is used to solve (4.2) with $\boldsymbol{W}_1$ being the deflation matrix. If $\boldsymbol{W}_1$ is large, then it would be very costly to factorize the coarse problem matrix (4.20). Instead, the coarse problem can be solved again by a deflated method, where the deflation matrix is $\boldsymbol{W}_2$. This nesting of DCG solvers for CP can continue until the coarsest CP is small enough to be solved easily by a direct method. The resulting method is called *multilevel* or *nested deflation*.

Numerical experiments combining the multilevel deflation with the shifted projector $\boldsymbol{P}_c$ and the adaptive precision of the previous sections can be found in [64].

4.3.8 Implementation

Our implementation of the deflation preconditioner, called PCDEFLATION [46], is available in PETSc. It supports complex systems of linear equations, i.e., $\boldsymbol{A} \in \mathbb{C}^{n \times n}$, $\boldsymbol{x} \in \mathbb{C}^n$, $\boldsymbol{b} \in \mathbb{C}^n$, as well as the complex deflation space $\boldsymbol{W} \in \mathbb{C}^{n \times m}$. Moreover, other Krylov subspace methods besides CG, e.g., MINRES and GMRES, for indefinite and possibly non-Hermitian systems of linear equations, are supported. Effectively, supporting complex numbers only requires swapping the transpose for the Hermitian transpose in the deflation operator [70].

Other features of PCDEFLATION are:

- Computing only the initial guess as in InitCG (Algorithm 4.4).
- The default deflation space is the Haar wavelet deflation introduced in [71]. Other wavelet-based deflation spaces [50] are available, as well as simple aggregation. Of course, the user can provide their own deflation space, e.g., eigenvectors obtained with SLEPc [72].
- Additional preconditioners (Section 4.3.2) are supported, and any PETSc preconditioner from the multitude available can be used.
- The coarse problem correction (Section 4.3.4) is available.
- The coarse problem is solved efficiently on the sub-communicator as described in Section 4.3.5. The default size of the sub-communicator is selected by a heuristic based on the problem size and the number of available cores.
- Nested or multilevel deflation, which is described in Section 4.3.7, is supported. It automatically selects Flexible CG or Flexible GMRES as the solvers for the nested coarse problems.
- Related to the previous point, the preconditioner supports adaptive precision across every level (Section 4.3.6).

All preconditioner options can be controlled programmatically or with command line arguments.

4.3.9 Results of the Deflated CG Method

In [64], we compared several methods for the solution of the FETI coarse problem, which is described in a later section (Section 7.1.2), for the 3D linear elasticity cube of Section 3.3.4. The FETI coarse problem is solved on the sub-communicators in the same way as described in Section 4.3.5, which is in the results denoted as strategy S1. Strategy S2 employs an LU direct solver to compute the explicit inverse of the coarse problem, with multiple sub-communicators computing a part of the inverse. The deflation uses 100 eigenvectors corresponding to the smallest eigenvalues. These were computed using the Krylov-Schur method implemented in SLEPc with the default settings and a tolerance of 10^{-4}. The

CG and DCG methods had the relative stopping criterion of 10^{-8}, which was sufficient to achieve the relative violation of the KKT conditions for the FETI primal problem (7.8) below 10^{-5}. The results computed on the ARCHER (Section 3.2.1) and Salomon (Section 3.2.4) supercomputers for different numbers of FETI coarse problem solves are presented in Figures 4.2 to 4.4.

The results showed that the deflation preconditioner applied to the CG method, i.e., the DCG method, is the best way to solve the FETI coarse problem for a moderate number of coarse problem solves. Moreover, together with the unpreconditioned CG, the DCG method exhibits the best scalability.

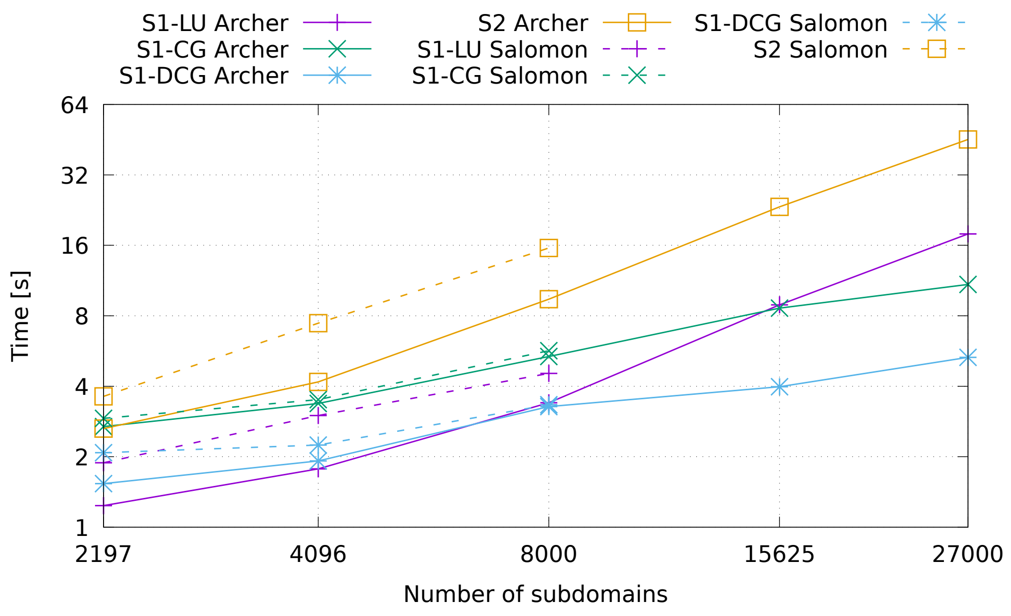

Figure 4.2: Weak scaling: Coarse problem solver setup + 100 coarse problem solves. One subdomain is assigned to one core [64].

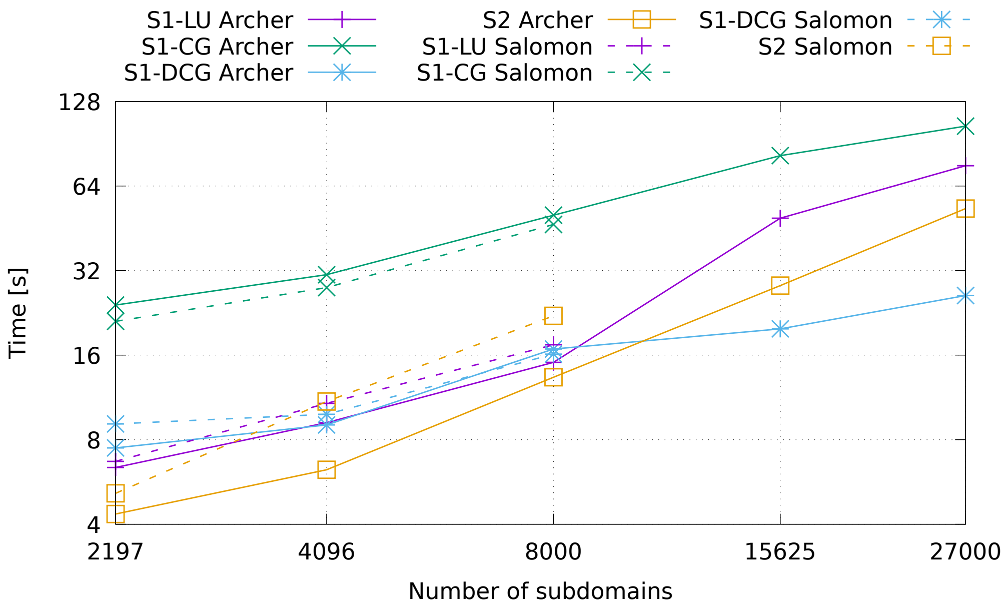

Figure 4.3: Weak scaling: Coarse problem solver setup + 1,000 coarse problem solves. One subdomain is assigned to one core [64].

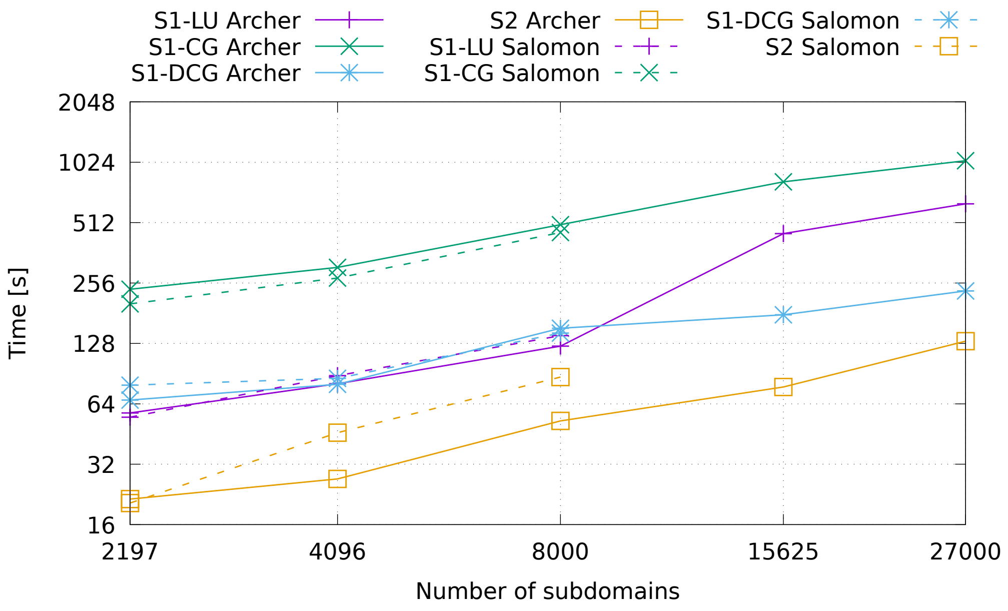

Figure 4.4: Weak scaling: Coarse problem solver setup + 10,000 coarse problem solves. One subdomain is assigned to one core [64].

Chapter 5

Projection-Based Quadratic Programming Algorithms

This chapter focuses on algorithms that use the projections of Section 2.3 to keep iterates feasible. Our exposition will focus on solving QP problems with box constraints

$$\Omega = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}\},$$

but the algorithms can be extended to other sets for which we have inexpensive exact projections. We admit partially constrained problems, i.e., we admit $\boldsymbol{l}_i = -\infty$ and $\boldsymbol{u}_j = +\infty$ for some components i and j.

The main focus of this chapter is on the MPRGP algorithm, which is described in the first section.

The second section introduces modifications to the most expensive step of the MPRGP algorithm, achieving a geometric mean of speedups of nearly 3 with one variant and over 6 with a variant combining MPRGP and the spectral projected gradient method on the problems for which the methods were designed.

The last section introduces a new variant of preconditioning for the MPRGP method. The MPRGP method equipped with the original preconditioning is sometimes even slower than the unpreconditioned MPRGP. The modifications in the previous section prove to be the key to making the new preconditioned MPRGP method extremely fast. The MPRGP equipped with the new preconditioner and the modifications achieved speedups between 5.1 and 13.4 compared to the standard unpreconditioned MPRGP.

The improvements to the MPRGP algorithm and preconditioning described in the last two sections are the own work of the author and are among the key results of this thesis. The results are supported by articles [64, 73].

5.1 MPRGP Algorithm

QP problems with box constraints can be solved using the modified proportioning with gradient projections (MPGP) or modified proportioning with reduced gradient projections

(MPRGP) algorithms [10, 74]. We will focus on the MPRGP variant, noting the differences with MPGP in the following description of the algorithm. The simplification to the feasible set with only one of the bound constraints is straightforward. It is also possible to adapt the algorithm for various other constraints, such as elliptic constraints [75, 76].

As the name of the algorithm suggests, it utilizes gradient information for minimization, placing it among the first-order optimization methods. While MPRGP does not directly work with active and free sets, the information about active and free sets is hidden in the gradient splitting. Consequently, MPRGP is considered an active set algorithm.

In each iteration, MPRGP performs one of three types of steps: unconstrained minimization, expansion, or proportioning. Since our Hessian $\boldsymbol{A}$ is SPS, the unconstrained minimization is performed by a step of the conjugate gradient method described in Section 4.2. The active set is expanded by the *expansion step*, which consists of a maximal feasible unconstrained minimization, in our case a partial CG step to the box, followed by a fixed step length gradient projection. Finally, a *proportioning step*, designed to reduce the active set, consists of a step of the steepest descent method (Section 4.1).

To properly describe the algorithm, we first need to define the *gradient splitting*. Let $\boldsymbol{g} = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}$ be the gradient of the cost function $f(\boldsymbol{x})$ and let

$$\mathcal{A} = \{i \mid \boldsymbol{x}_i = \boldsymbol{l}_i \quad \text{or} \quad \boldsymbol{x}_i = \boldsymbol{u}_i\}\,, \qquad \mathcal{F} = \{i \mid \boldsymbol{l}_i < \boldsymbol{x}_i < \boldsymbol{u}_i\}$$

be the *active* and *free set*, respectively. Then the gradient splitting is defined componentwise for $i \in \{1, 2, \ldots, n\}$ and is computed after each gradient evaluation. The *free gradient* $\boldsymbol{g}^f$ is defined as

$$\boldsymbol{g}_i^f = \begin{cases} 0 & \text{if} \quad i \in \mathcal{A}, \\ \boldsymbol{g}_i & \text{if} \quad i \in \mathcal{F}. \end{cases} \tag{5.1}$$

The *reduced free gradient* $\boldsymbol{g}^r$ is defined as

$$\boldsymbol{g}_i^r = \begin{cases} 0 & \text{if} \quad i \in \mathcal{A}, \\ \min\left(\frac{\boldsymbol{x}_i - \boldsymbol{l}_i}{\overline{\alpha}}, \boldsymbol{g}_i\right) & \text{if} \quad i \in \mathcal{F} \quad \text{and} \quad \boldsymbol{g}_i > 0, \\ \max\left(\frac{\boldsymbol{x}_i - \boldsymbol{u}_i}{\overline{\alpha}}, \boldsymbol{g}_i\right) & \text{if} \quad i \in \mathcal{F} \quad \text{and} \quad \boldsymbol{g}_i \le 0, \end{cases}$$

where $\overline{\alpha} \in (0, 2||\boldsymbol{A}||^{-1}]$ is used as a priori chosen fixed step length in the expansion step [74]. Typically, the parameter is given by

$$\overline{\alpha} = \alpha_u ||\boldsymbol{A}||^{-1},$$

where $\alpha_u \in (0, 2]$ is a user-selected constant, and $||\boldsymbol{A}||^{-1}$ is approximated by the power method. Effectively, $\boldsymbol{g}^f$ is the gradient on the free set, and $\boldsymbol{g}^r$ is the free gradient that is reduced such that a steepest descent-type step in its opposite direction, that is $-\boldsymbol{g}^r$, with the step length $\overline{\alpha}$, does not leave the feasible set Ω. A step in either of these directions can expand the active set but cannot reduce it.

The *chopped gradient* $\boldsymbol{g}^c$ is defined as

$$\boldsymbol{g}_i^c = \begin{cases} 0 & \text{if} \quad i \in \mathcal{F}, \\ \min(\boldsymbol{g}_i, 0) & \text{if} \quad \boldsymbol{x}_i = \boldsymbol{l}_i, \\ \max(\boldsymbol{g}_i, 0) & \text{if} \quad \boldsymbol{x}_i = \boldsymbol{u}_i. \end{cases}$$

A step in the direction $-\boldsymbol{g}^c$ may reduce the active set but cannot expand it.

The next ingredient is the projection onto the feasible set Ω, which by Section 2.3 is defined as

$$[P_\Omega(\boldsymbol{x})]_i = \min\left\{\boldsymbol{u}_i, \max\left\{\boldsymbol{l}_i, \boldsymbol{x}_i\right\}\right\}, \quad i \in \{1, \dots, n\}.$$

Finally, the *projected gradient* is defined as $\boldsymbol{g}^P = \boldsymbol{g}^f + \boldsymbol{g}^c$. The decrease in its norm serves as the natural stopping criterion for the algorithm since $\boldsymbol{g}^P = \boldsymbol{o}$ is equivalent to satisfying the Karush-Kuhn-Tucker conditions of Section 2.4.1 for a box-constrained QP problem.

These are all the necessary ingredients to summarize MPRGP in Algorithm 5.1, which is explained in more detail in the following text.

Algorithm 5.1: MPRGP method

```
   Input: $\boldsymbol{A}$, $\boldsymbol{x}_0 \in \Omega$, $\boldsymbol{b}$, $\Gamma > 0$, $\overline{\alpha} \in (0, 2||\boldsymbol{A}||^{-1}]$
 1 $\boldsymbol{g}_0 = \boldsymbol{A}\boldsymbol{x}_0 - \boldsymbol{b}$, $\boldsymbol{p}_0 = \boldsymbol{g}_0^f$, $k = 0$
 2 while $||\boldsymbol{g}_k^P||$ is not small:
 3     if $||\boldsymbol{g}_k^c||^2 \leq \Gamma^2 ||\boldsymbol{g}_k^f||^2$:
 4         $\alpha_k^{feas} = \max\{\alpha \mid \boldsymbol{x}_k - \alpha \boldsymbol{p}_k \in \Omega\}$
 5         $\alpha_k^{cg} = \boldsymbol{g}_k^T \boldsymbol{p}_k / \boldsymbol{p}_k^T \boldsymbol{A} \boldsymbol{p}_k$
 6         if $\alpha_k^{cg} \leq \alpha_k^{feas}$:
 7             CG step - Algorithm 5.2
 8         else:
 9             Expansion step - Algorithm 5.3;
10     else:
11         Proportioning step - Algorithm 5.4;
12     $k = k + 1$
   Output: $\boldsymbol{x}_k$
```

Algorithm 5.2: CG step

```
1 $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k^{cg} \boldsymbol{p}_k$
2 $\boldsymbol{g}_{k+1} = \boldsymbol{g}_k - \alpha_k^{cg} \boldsymbol{A} \boldsymbol{p}_k$
3 $\beta_k = \boldsymbol{p}_k^T \boldsymbol{A} \boldsymbol{g}_{k+1}^f / \boldsymbol{p}_k^T \boldsymbol{A} \boldsymbol{p}_k$
4 $\boldsymbol{p}_{k+1} = \boldsymbol{g}_{k+1}^f - \beta_k \boldsymbol{p}_k$
```

Algorithm 5.3: Expansion step

```
1 $\boldsymbol{x}_{k+\frac{1}{2}} = \boldsymbol{x}_k - \alpha_k^{feas}\boldsymbol{p}_k$
2 $\boldsymbol{g}_{k+\frac{1}{2}} = \boldsymbol{g}_k - \alpha_k^{feas}\boldsymbol{A}\boldsymbol{p}_k$
3 $\boldsymbol{x}_{k+1} = P_\Omega(\boldsymbol{x}_{k+\frac{1}{2}} - \overline{\alpha}\boldsymbol{g}_k^f)$
4 $\boldsymbol{g}_{k+1} = \boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{b}$
5 $\boldsymbol{p}_{k+1} = \boldsymbol{g}_{k+1}^f$
```

Algorithm 5.4: Proportioning step

```
1 $\alpha_k^{sd} = \boldsymbol{g}_k^T\boldsymbol{g}_k^c/(\boldsymbol{g}_k^c)^T\boldsymbol{A}\boldsymbol{g}_k^c$
2 $\alpha_k^{feas} = \max\{\alpha \mid \boldsymbol{x}_k - \alpha\boldsymbol{g}_k^c \in \Omega\}$
3 if $\alpha_k^{feas} < \alpha_k^{sd}$:
4     $\alpha_k^{sd} = \alpha_k^{feas}$
5 $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k^{sd}\boldsymbol{g}_k^c$
6 $\boldsymbol{g}_{k+1} = \boldsymbol{g}_k - \alpha_k^{sd}\boldsymbol{A}\boldsymbol{g}_k^c$
7 $\boldsymbol{p}_{k+1} = \boldsymbol{g}_{k+1}^f$
```

In each iteration, the algorithm checks that the current approximation of the solution, $\boldsymbol{x}_k$, is *proportional*[1]

$$||\boldsymbol{g}_k^c||^2 \leq \Gamma^2||\boldsymbol{g}_k^f||^2, \qquad \Gamma > 0. \tag{5.2}$$

If this inequality does not hold, the chopped gradient $\boldsymbol{g}^c$ dominates the norm of the projected gradient $\boldsymbol{g}^P$, depending on the value of the parameter Γ (typically $\Gamma = 1$). Therefore, we need to release some components from the active set by a proportioning step. This proportioning step consists of a single step of the steepest descent method in the direction $-\boldsymbol{g}^c$. In the case of the box and similar constraints, the step may need to be shortened so that it does not leave the feasible set.

On the other hand, if the current solution is proportional, i.e., (5.2) holds, then the free gradient $\boldsymbol{g}^f$ dominates the norm of the projected gradient $\boldsymbol{g}^P$, and we focus on the minimization of the free gradient $\boldsymbol{g}^f$. First, we compute α^{cg} as the optimal step length for minimization in the direction $-\boldsymbol{p}$ and α^{feas} as the *maximum feasible step length* in this direction that does not leave the feasible set. If $\alpha^{cg} \leq \alpha^{feas}$, we can perform an unconstrained minimization using a standard CG step; otherwise, we proceed with the expansion step. Note that initially, and after both the expansion and proportioning steps, the next minimization direction is set to $\boldsymbol{p} = \boldsymbol{g}^f$, while the CG steps set the next minimization direction $\boldsymbol{A}$-orthogonal to the previous one.

The expansion consists of the so-called half-step, which is a step with the maximum step length α^{feas} in the direction $-\boldsymbol{p}$. The half-step expands the active set, but typically only by one component. Then a step in the direction $-\boldsymbol{g}^f$ with a fixed step length $\overline{\alpha} \in (0, 2||\boldsymbol{A}||^{-1}]$

[1]We note that Dostál [10] employs a *strict* proportionality condition that replaces $\boldsymbol{g}^f$ with $\boldsymbol{g}^r$. This is needed for the convergence proof but reduces the proportionality cone as $||\boldsymbol{g}^r|| \leq ||\boldsymbol{g}^f||$. The MPGP algorithm uses the proportionality condition as given by Equation (5.2).

is performed[2]. Notice that using either $\boldsymbol{g}^f$ or $\boldsymbol{g}^r$ is equivalent, since we have

$$\boldsymbol{x}_{k+1} = P_\Omega(\boldsymbol{x}_{k+\frac{1}{2}} - \overline{\alpha}\boldsymbol{g}^f_{k+\frac{1}{2}}) = \boldsymbol{x}_{k+\frac{1}{2}} - \overline{\alpha}\boldsymbol{g}^r_{k+\frac{1}{2}},$$

i.e., due to $\boldsymbol{g}^r$ construction, the step with $\boldsymbol{g}^r$ does not need to be projected onto the feasible set. The active set is expanded in a component j if this component is in the free set and either

$$\boldsymbol{g}_j > 0 \quad \text{and} \quad \overline{\alpha}\boldsymbol{g}_j \geq \boldsymbol{x}_j - \boldsymbol{l}_j$$

or

$$\boldsymbol{g}_j \leq 0 \quad \text{and} \quad \overline{\alpha}\boldsymbol{g}_j \leq \boldsymbol{x}_j - \boldsymbol{u}_j.$$

Therefore, $\overline{\alpha}$ controls how large a component of the gradient (in the correct direction) has to be to expand the active set in the given component. Larger values of $\overline{\alpha}$ have the potential to expand the active set in more components. However, even with the largest possible value of the step length $\overline{\alpha} = 2||\boldsymbol{A}||^{-1}$, the active set may not be expanded at all. We refer to this expansion step variant as *Fixed* due to the use of the fixed step length.

Finally, the maximum feasible step length used in the expansion and, depending on the constraints, in the proportioning step can have a closed form, e.g.,

$$\begin{aligned}\alpha^{feas} &= \max\{\alpha \mid \boldsymbol{x} - \alpha\boldsymbol{p} \in \Omega\}\\ &= \min\left\{\left\{(\boldsymbol{x}_i - \boldsymbol{l}_i)/\boldsymbol{p}_i \,\middle|\, \begin{array}{l}\boldsymbol{p}_i > 0,\\ i \in \{1,\dots,n\}\end{array}\right\} \cup \left\{(\boldsymbol{x}_i - \boldsymbol{u}_i)/\boldsymbol{p}_i \,\middle|\, \begin{array}{l}\boldsymbol{p}_i < 0,\\ i \in \{1,\dots,n\}\end{array}\right\}\right\},\end{aligned}$$

in the case of the box constraints.

The operation count for each of the three steps is summarized in Table 5.1. We argue that, in most cases, the cost of a step primarily depends on the number of Hessian multiplications it performs.

Note that either bound may be omitted in the formulation of the algorithm. If both bounds are omitted, the algorithm is equivalent to the standard CG method. The CG method in finite precision can suffer from the loss of orthogonality of the search directions and subsequent delay in convergence [54, 77, 78]. The MPRGP method naturally limits this effect due to the explicit gradient evaluation whenever the expansion step is computed.

[2]MPGP uses the direction $-\boldsymbol{g}$ instead of $-\boldsymbol{g}^f$.

Step	Hess. mult.	Dot prod.	Vec. update	Grad. split.
CG	1	2	3	1
Expansion: Fixed	2	1	5	2
Proportioning	1	1	3	1
Expansion: Projected CG	2	1	3	1
+ Fallback 1	0/1	1/1	0/4	0/1
+ Fallback 2	0/1	1/1	0/4	1/2
Expansion: SPG	2	1+	4+	1

Table 5.1: Number of operations per MPRGP step. The bottom part of the table contains the newly proposed variants of the expansion step introduced in the following sections.

5.2 MPRGP Expansion Modifications

As discussed in the previous section, the time to solution is primarily determined by the number of Hessian multiplications. Therefore, we need to minimize the overall number of Hessian multiplications to speed up the MPRGP algorithm.

In our previous work [79], we demonstrated that MPRGP may need many expansion steps to identify the active set. This is because standard expansion steps often enlarge the active set by only one or a few components. Moreover, the expansion step is approximately twice as expensive as the other steps due to it requiring two Hessian multiplications instead of one. Our idea is to modify the expansion step to enlarge the active set faster. Such modifications should lead to a reduction in the number of expansion steps, as well as the overall number of Hessian multiplications.

In [79], the fixed step length in gradient projection was replaced with the steepest descent step length or its approximation, computed using various gradients used throughout the method. The steepest descent step length could significantly reduce the number of Hessian multiplications needed to compute the solution, despite the expansion step needing three Hessian multiplications. However, the performance was highly dependent on the scaling factor α_u of the step length. The article also introduced an algorithm (MPRGP with projected CG) in which the expansion step was performed by projecting the full unconstrained CG step. This latter scheme significantly outperformed the classic MPRGP algorithm and is the focus of the next two subsections.

The last subsection deals with a combination of the MPRGP algorithm and the spectral projected gradient method. The modification is based on our experience comparing the two algorithms in [73], where we observed that the spectral projected gradient method can converge very quickly for certain problems but only to a lower satisfaction of the KKT conditions, while MPRGP can converge to full machine precision.

5.2.1 MPRGP with Projected CG Expansion Step

Recall that the expansion consists of the half-step, followed by a fixed step length gradient projection. The half-step is a CG step with the step length reduced to ensure that the computed approximation is within the feasible set, i.e., $\boldsymbol{x}_{k+\frac{1}{2}} \in \Omega$.

Since our goal is to expand the active set faster, it seems reasonable to replace the half-step with the full CG step, followed by a subsequent projection onto the feasible set. To be more specific, our expansion step becomes

$$\boldsymbol{x}_{k+1} = P_\Omega(\boldsymbol{x}_k - \alpha_k^{cg}\boldsymbol{p}_k),$$

followed by resetting $\boldsymbol{p}_{k+1} = \boldsymbol{g}_{k+1}^f$. Note that realizing the expansion in this way simplifies the implementation, as we can always compute the CG step and then calculate the gradient using the CG recurrence when the step is feasible; otherwise, we project the solution onto the feasible set and explicitly recompute the gradient. Algorithm 5.5 illustrates the implementation. It replaces the *if...else* block on lines 6–9 in Algorithm 5.1. We call this algorithm MPPCG (Modified Proportioning with Projected Conjugate Gradient).

Algorithm 5.5: Projected CG step

```
1 $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k^{cg}\boldsymbol{p}_k$
2 if $\alpha_k^{cg} \le \alpha^{feas}$:
3     $\boldsymbol{g}_{k+1} = \boldsymbol{g}_k - \alpha_k^{cg}\boldsymbol{A}\boldsymbol{p}_k$
4     $\beta_k = \boldsymbol{p}_k^T\boldsymbol{A}\boldsymbol{g}_{k+1}^f / \boldsymbol{p}_k^T\boldsymbol{A}\boldsymbol{p}_k$
5     $\boldsymbol{p}_{k+1} = \boldsymbol{g}_{k+1}^f - \beta_k\boldsymbol{p}_k$
6 else:
7     $\boldsymbol{x}_{k+1} = P_\Omega(\boldsymbol{x}_{k+1})$
8     $\boldsymbol{g}_{k+1} = \boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{b}$
9     $\boldsymbol{p}_{k+1} = \boldsymbol{g}_{k+1}^f$
```

As shown in Table 5.1, this *projected CG* expansion variant has the same or a better operation count compared to the other expansion variants.

Table 5.2 compares the standard MPRGP and MPPCG on 3D linear elasticity with contact, as described in Section 3.3.4, and on training SVMs on three datasets, as described in Section 3.3.6. The contact problem uses TFETI, as described in Chapter 7, with regular decomposition into $1,000$ subdomains (10 in each direction) with $27,000$ elements per subdomain (30 in each direction), making a total of $81,812,703$ (undecomposed) degrees of freedom (DOFs). The problem requires an outer solver (SMALE-M of Section 6.3) with parameters $\eta = 0.1$, $\rho = 1.1||\boldsymbol{A}||$, $M = 100||\boldsymbol{A}||$, and $\beta = 10$. The results were obtained with MPRGP/MPPCG parameter $\Gamma = 1$ and with the fixed expansion step length value of $\overline{\alpha} = 1.9||\boldsymbol{A}||^{-1}$, which is the recommended value in [79]. The stopping tolerance relative to the right-hand side was 10^{-6} for the contact problem and 10^{-1} for the SVMs.

We note that the computation of the largest eigenvalue $\lambda_{max} = ||\boldsymbol{A}||$ usually employs the power method [51], which requires some additional Hessian multiplications[3]. The projected CG expansion step does not require the largest eigenvalue estimates and thus saves some additional Hessian multiplications. We do not count the number of Hessian multiplications required by the power method in the presented results.

[3]PERMON defaults for the power method are the relative tolerance of 10^{-4} with a maximum of 50 iterations. The typical number of iterations is in the low tens but can reach the maximum number of iterations.

Problem	Expansion type	Hess. mult.	CG	Exp.	Prop.
3D Contact	Fixed	323	144	83	3
	Projected CG	292	171	53	5
SVM: Australian	Fixed	195	8	92	2
	Projected CG	83	16	32	2
SVM: Diabetes	Fixed	630	2	313	1
	Projected CG	133	13	58	3
SVM: Ionosphere	Fixed	381	21	179	1
	Projected CG	125	14	54	2

Table 5.2: Comparison of the number of Hessian multiplications and the number of each step [79].

Looking at the number of Hessian multiplications, the projected CG variant, compared to the standard expansion variant, achieved a speedup of 1.10 for the contact problem and between 2.34 and 4.73 for the SVMs.

5.2.2 MPRGP with Projected CG Expansion Step and Fallback

In some cases, the projected CG expansion step can lead to an increase in the value of the cost function [10]. We illustrate this for a 2D problem in Figure 5.1. Clearly, the CG step finds the unconstrained minimizer of the cost function, but the subsequent projection onto the feasible set places us on a higher contour line, i.e., increases the value of the cost function. In this case, the standard expansion would put us closer to the solution. Yet, unless $\overline{\alpha}$ happens to be such that the fixed gradient projection finds the exact solution, which in this case it cannot, both approaches converge in the next iteration. Note that in this example, the *opt* step length with $\alpha_u = 1$ described in [79], with any combination of the allowed vectors for both the computation of the step length and the line search direction, would converge to the exact solution in a single expansion step.

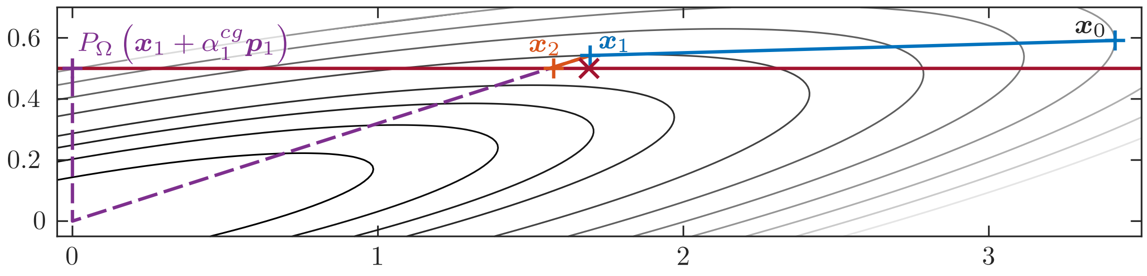

Figure 5.1: Projected CG step illustration (based on [10]). The ellipses are the cost function contour lines. The feasible set is on and above the red horizontal line. The red cross (X) on the horizontal line is the solution. The point $\boldsymbol{x}_1$ is reached by the first unconstrained CG step starting with $\boldsymbol{x}_0$. The point $\boldsymbol{x}_2$ would be reached by the expansion step with the largest admissible $\overline{\alpha}$, and the point $P_\Omega\left(\boldsymbol{x}_1 + \alpha_1^{cg}\boldsymbol{p}_1\right)$ would be reached by the projected CG expansion step.

To illustrate our previous point that the projected CG expansion step can increase the cost function, we increased the relative tolerance on the projected gradient to 10^{-4} for the

SVMs. We are aware that this tolerance is too strict and overfits the SVM model [43], but the problems are valid QP problems for our illustrative purposes. Preliminary results are reported in Table 5.3.

Dataset	Exp. type	Hess. Mult.	CG	Exp.	Prop.	Cost inc.	Fall.
Australian	Fixed	4567	1134	1704	24	423	0
Australian	Projected CG	3571	565	1455	95	127	0
Australian	Fallback 1	3298	1015	1014	36	218	218
Australian	Fallback 2	2160	747	662	32	74	56
Diabetes	Fixed	1108	124	491	1	18	0
Diabetes	Projected CG	1439	113	627	71	109	0
Diabetes	Fallback 1	292	84	96	1	14	14
Diabetes	Fallback 2	292	84	96	1	14	14
Ionosphere	Fixed	628	149	237	4	8	0
Ionosphere	Projected CG	265	104	78	4	6	0
Ionosphere	Fallback 1	320	109	89	5	27	27
Ionosphere	Fallback 2	277	104	78	5	13	11

Table 5.3: Comparison of MPRGP expansion step variants computing an intentionally overfitted SVM model with the relative tolerance of 10^{-4}. The fallback expansion is defined in the latter part of this section. The comparison includes the number of cost function increases and the number of fallback steps in the last two columns, respectively.

It can be observed that the method with the projected CG expansion outperforms the standard method for the Australian and Ionosphere datasets. However, there is a slowdown of 30% on the Diabetes dataset.

We can see a marked increase in the number of proportioning steps for the Australian dataset and especially for the Diabetes dataset. This suggests that, at some points, the expansion of the active set may be too aggressive. Moreover, when comparing the convergence curves in Figures 5.2 and 5.3 for the Diabetes dataset, the convergence of the projected CG variant is much faster in the first approximately 30 iterations. However, after that, one expansion step leads to a significant increase in the value of the cost function.

To avoid the increase in the value of the cost function (as the standard MPRGP decreases the value of the cost function monotonically), we propose falling back to the fixed gradient projection expansion when the projected conjugate gradient expansion step would increase the value of the cost function. Algorithmically, we replace Algorithm 5.5 with Algorithm 5.6, where the *fallback* condition is

$$f(\boldsymbol{x}_{k+1}) > f(\boldsymbol{x}_k). \tag{5.3}$$

We refer to this condition as *Fallback 1*. The step length for our numerical experiments is chosen to be $\overline{\alpha} = ||\boldsymbol{A}||^{-1}$, which maximizes the decrease in the value of the cost function. Moreover, enforcing expansion to not increase the value of the cost function guarantees the convergence of the algorithm by Proposition 5.12 in [10]. The effect on the convergence curve for the new variant on the Diabetes dataset can be observed in Figure 5.4.

The drawback of this fallback scheme is that we now need to estimate the largest eigenvalue of the Hessian to select the fixed step length $\overline{\alpha}$. Nevertheless, from the results

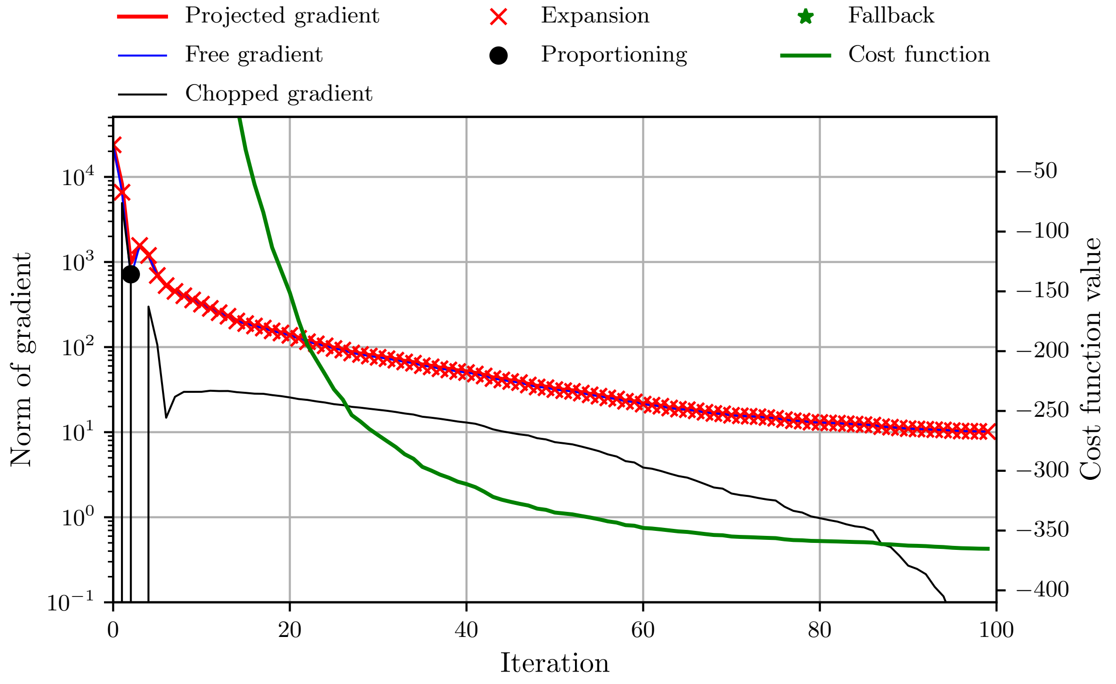

Figure 5.2: SVM - Diabetes dataset: First 100 iterations of the standard MPRGP.

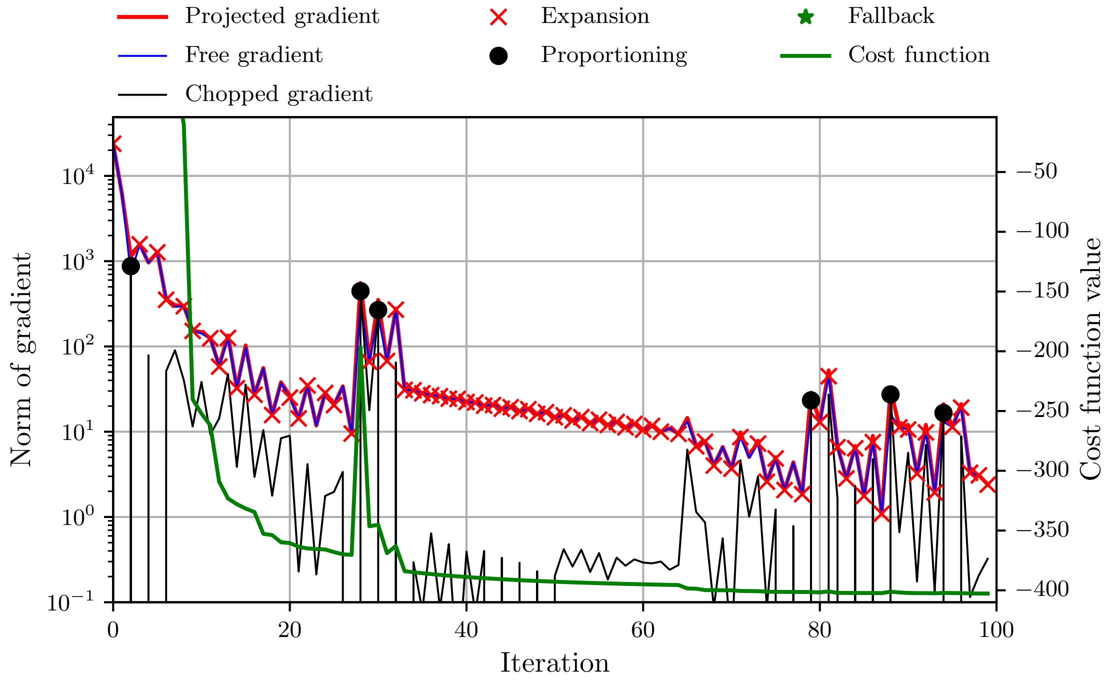

Figure 5.3: SVM - Diabetes dataset: First 100 iterations of MPPCG.

Algorithm 5.6: Projected CG step with fallback

```
1  $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k^{cg} \boldsymbol{p}_k$
2  if $\alpha_k^{cg} \leq \alpha^{feas}$:
3      $\boldsymbol{g}_{k+1} = \boldsymbol{g}_k - \alpha_k^{cg} \boldsymbol{A}\boldsymbol{p}_k$
4      $\beta_k = \boldsymbol{p}_k^T \boldsymbol{A}\boldsymbol{g}_{k+1}^f / \boldsymbol{p}_k^T \boldsymbol{A}\boldsymbol{p}_k$
5      $\boldsymbol{p}_{k+1} = \boldsymbol{g}_{k+1}^f - \beta_k \boldsymbol{p}_k$
6  else:
7      $\boldsymbol{x}_{k+1} = P_\Omega(\boldsymbol{x}_{k+1})$
8      $\boldsymbol{g}_{k+1} = \boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{b}$
9      if $fallback$:
10         $\boldsymbol{x}_{k+\frac{1}{2}} = \boldsymbol{x}_k - \alpha_k^{feas} \boldsymbol{p}_k$
11         $\boldsymbol{g}_{k+\frac{1}{2}} = \boldsymbol{g}_k - \alpha_k^{feas} \boldsymbol{A}\boldsymbol{p}_k$
12         $\boldsymbol{x}_{k+1} = P_\Omega(\boldsymbol{x}_{k+\frac{1}{2}} - \overline{\alpha}\boldsymbol{g}_{k+\frac{1}{2}}^f)$
13         $\boldsymbol{g}_{k+1} = \boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{b}$
14     $\boldsymbol{p}_{k+1} = \boldsymbol{g}_{k+1}^f$
```

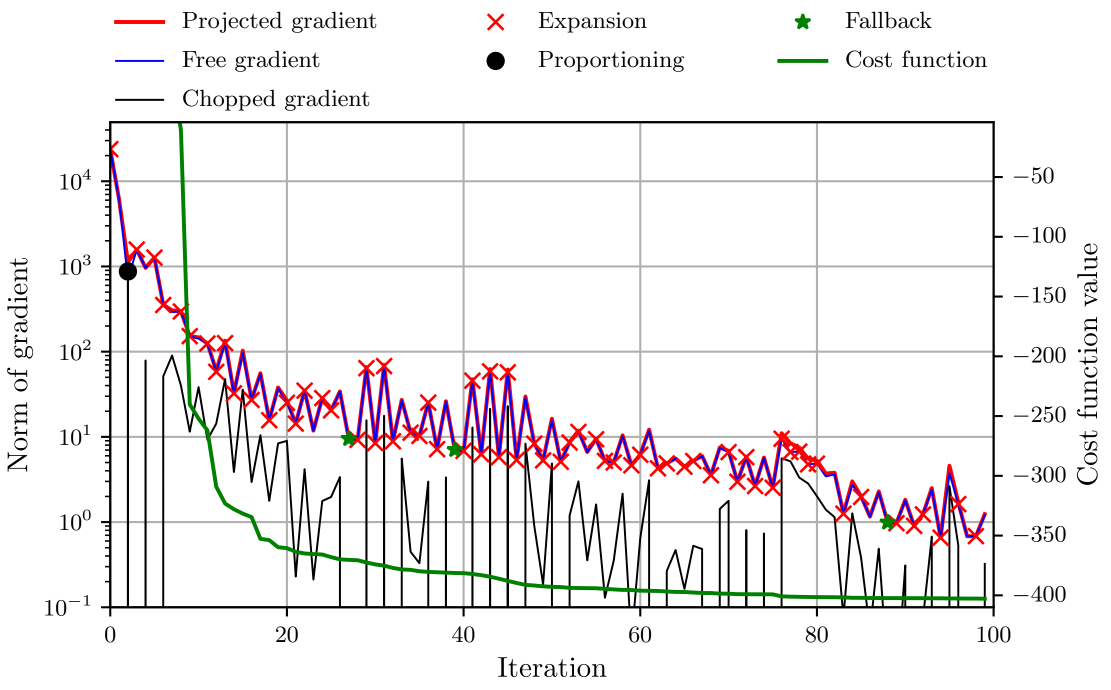

Figure 5.4: SVM - Diabetes dataset: First 100 iterations of MPPCG with Fallback 1.

in Table 5.3, MPPCG with Fallback 1 expansion step consistently outperforms MPRGP with the fixed step length expansion step. The same could be said when comparing the new variant with the projected CG variant without the fallback, except that there is a 21% increase in the number of Hessian multiplications for the Ionosphere dataset.

Upon examining the convergence of MPPCG with Fallback 1 for the Ionosphere dataset in Figure 5.5, we observe a long string of fallback steps between iterations 75 and 95.

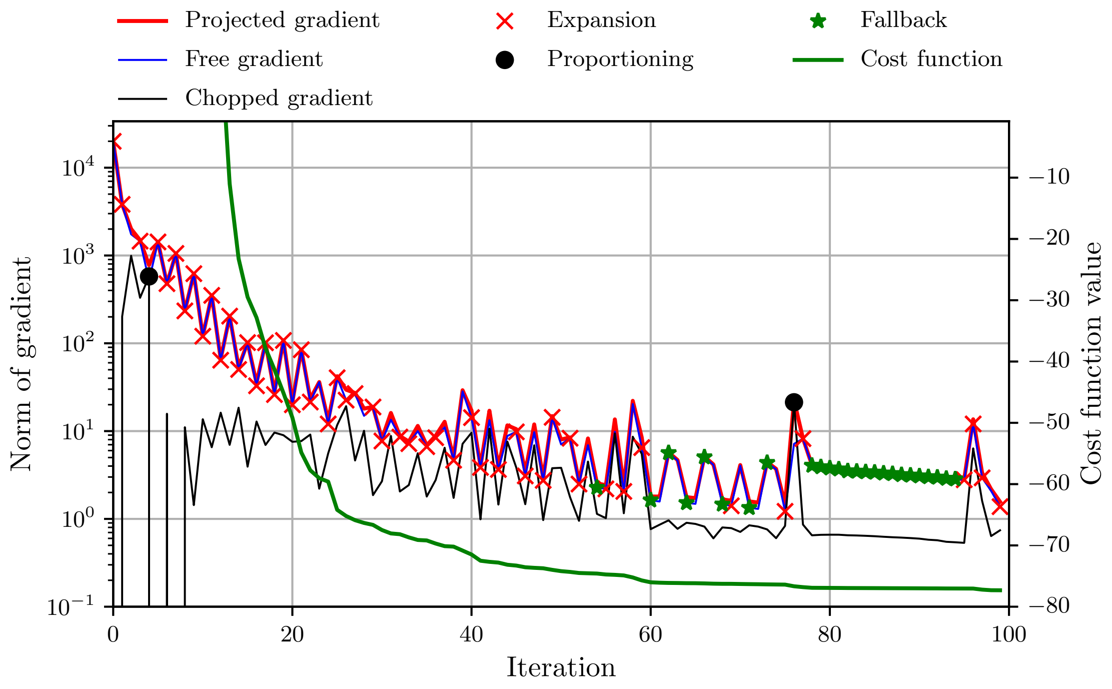

Figure 5.5: SVM - Ionosphere dataset: First 100 iterations of MPPCG with Fallback 1.

This suggests that the fixed step length expansion is not expanding the active set quickly enough. Therefore, we relax our Fallback 1 scheme, allowing an increase in the cost function value if the next step of the MPRGP algorithm is not the proportioning step. Intuitively, we do not want to excessively expand the active set, only for the next step to immediately release some of the active components. The new fallback condition, called *Fallback 2*, then reads

$$f(\boldsymbol{x}_{k+1}) > f(\boldsymbol{x}_k) \quad \wedge \quad ||\boldsymbol{g}^c_{k+1}|| > \Gamma ||\boldsymbol{g}^f_{k+1}||.$$

This latest fallback scheme once again improves convergence (see Table 5.3), and in the case of the Ionosphere dataset, it nearly recovers the performance of the non-fallback projected CG expansion variant. For comparison with Figure 5.5, the convergence curve for the Ionosphere dataset solved with the Fallback 2 variant is provided in Figure 5.6.

The number of operations added to the projected CG expansion step for both fallback variants is summarized in Table 5.1. Since the fallback criterion check is done in each expansion step, the first value corresponds to the number of operations required for evaluating the fallback condition, i.e., when the fallback condition is false. The second

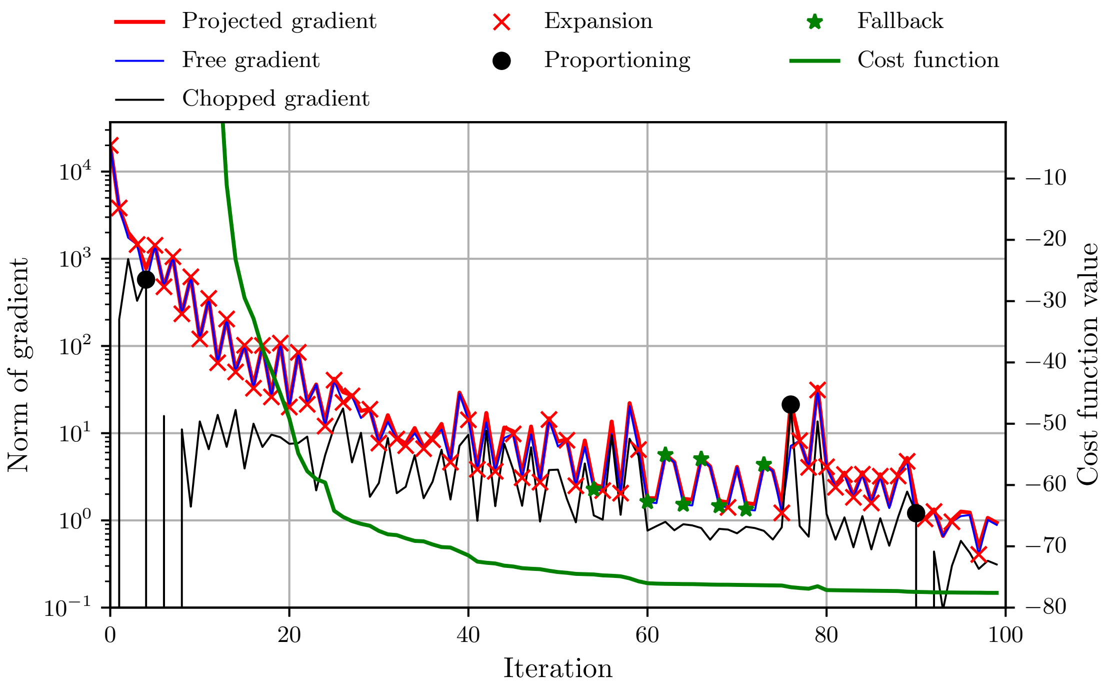

Figure 5.6: SVM - Ionosphere dataset: First 100 iterations of MPPCG with Fallback 2.

value is for both checking the fallback condition and carrying out the fallback step, i.e., when the fallback condition is true.

As mentioned earlier, the presented results are considered preliminary. This is because the comparison of the value of the cost function should take into account the tolerance of the computation. In fact, Table 5.3 reports the number of cost function increases, which are counted whenever the Fallback 1 criterion, given by Equation (5.3), is satisfied. We can see that there is a large number of cost function increases for the standard MPRGP, which is guaranteed to have a monotonic decrease. This is due to the cost function nearly stagnating at its minimum, while the projected gradient has not yet reached the stopping criterion. However, in finite precision, the cost function can increase in the last few represented decimal digits. It is possible that taking into account the computational tolerance in the fallback schemes could further improve their performance. There is also the possibility of developing additional fallback schemes, e.g., performing fallback if the Fallback 2 criterion is true, but only if some of the components that would be released by the proportioning step were added by the current expansion step.

The convergence bound of the standard MPRGP method is [10]

$$f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}^*) \leq \eta \left(f(\boldsymbol{x}_k) - f(\boldsymbol{x}^*) \right), \quad \eta \in \left[\frac{3}{4}, 1 \right),$$

where the constant η depends on the smallest and the largest eigenvalue of $\boldsymbol{A}$ and the parameters $\overline{\alpha}$ and Γ. Using the convergence bound, we can propose a fallback condition

$$f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}^*) > \eta \left(f(\boldsymbol{x}_k) - f(\boldsymbol{x}^*) \right),$$

which can be reordered as

$$f(\boldsymbol{x}_{k+1}) - \eta f(\boldsymbol{x}_k) > (1-\eta)\, f(\boldsymbol{x}^*). \tag{5.4}$$

Of course, we typically do not have the optimal value $f(\boldsymbol{x}^*)$ before we find the solution. An underestimate $f(\overline{\boldsymbol{x}})$ of the optimal cost $f(\boldsymbol{x}^*)$ could be used instead. Using Equation (5.4) as the fallback criterion in the MPPCG method would ensure that the method has at least as good a bound on convergence as the standard MPRGP method.

5.2.3 MPRGP with Efficient Gradient Projections

Our comparison in [73] of MPRGP, MPPCG, and a variant of the spectral projected gradient (SPG) [80] algorithm showed that, in cases with many active constraints, SPG methods can achieve much better performance than the MPRGP-type methods. On the other hand, the MPRGP and MPPCG algorithms are usually better when the active set is small because they benefit from long chains of CG steps for unconstrained minimization. Additionally, it was briefly remarked that the SPG algorithm can find the solution only to a limited precision.

In the following text, we will describe an SPG method, showcase the convergence of the SPG and MPRGP-type methods, and finally develop a method combining MPRGP and SPG.

5.2.3.1 Spectral Projected Gradient Method

The basic iteration of the standard gradient projection (GP) method along the feasible direction [9] is given by

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \nu_k \boldsymbol{d}_k = \boldsymbol{x}_k + \nu_k \left(P_\Omega(\boldsymbol{x}_k - \alpha_k \boldsymbol{g}_k) - \boldsymbol{x}_k \right),$$

where $\boldsymbol{d}_k$ is a feasible decrease direction for the objective function at $\boldsymbol{x}_k$, $\alpha_k \in [\alpha_{min}, \alpha_{max}]$, $0 < \alpha_{min} \leq \alpha_{max}$, is the step length parameter, and $\nu_k \in (0, 1]$ is the line search parameter determined by a backtracking procedure implementing a monotone Armijo rule [9] or its nonmonotone version [81].

The GP method is described in Algorithm 5.8. Observe that for $M = 1$ at step 3 of the algorithm, the standard Armijo rule is obtained, while for $M > 1$, a nonmonotone Grippo-Lampariello-Lucidi (GLL) line search [81] is performed. A natural stopping criterion for GP schemes is based on the decrease in the norm of the descent direction.

When the step length α_k is defined by some type of Barzilai-Borwein step length (see Section 4.1.1), the GP method is known as the spectral projected gradient method [80, 82]. The spectral prefix in the name is added because the step lengths sweep the spectrum of the Hessian [83, 84].

Our implementation of the SPG method employs the box constraints-aware step length rule BoxVABB$_{\text{min}}$ [83]. The BB2 step length defined by Equation (4.5) is modified according

Algorithm 5.7: GP method

```
Input: $\boldsymbol{x}_0 \in \Omega,\ \delta, \sigma \in (0,1),\ M \in \mathbb{N},\ 0 < \alpha_{min} \leq \alpha_{max},\ \alpha_0 \in [\alpha_{min}, \alpha_{max}]$
1 for $k = 0, 1, \ldots$:
2     $\boldsymbol{d}_k = P_\Omega(\boldsymbol{x}_k - \alpha_k \boldsymbol{g}_k) - \boldsymbol{x}_k$
3     $\nu_k = 1; \quad f_{ref} = \max\{f(\boldsymbol{x}_{k-i}),\ 0 \leq i \leq \min(k, M-1)\}$
4     while $f(\boldsymbol{x}_k + \nu_k \boldsymbol{d}_k) > f_{ref} + \sigma \nu_k \boldsymbol{g}_k^T \boldsymbol{d}_k$:
5         $\nu_k = \delta \nu_k;$
6     $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \nu_k \boldsymbol{d}_k$
7     define the step length $\alpha_{k+1} \in [\alpha_{min}, \alpha_{max}]$
Output: $\boldsymbol{x}_{k+1}$
```

to the free set in the last two iterations

$$\alpha_k^{\text{BoxBB2}} = \frac{[\boldsymbol{s}_{k-1}]_{\mathcal{I}_{k-1}}^T [\boldsymbol{y}_{k-1}]_{\mathcal{I}_{k-1}}}{||\, [\boldsymbol{y}_{k-1}]_{\mathcal{I}_{k-1}} \,||^2},$$

where $\mathcal{I}_{k-1}$ is a subset of indices defined as follows:

$$\mathcal{I}_{k-1} = \{1, 2, \ldots, n\} \setminus \mathcal{J}_{k-1},$$
$$\mathcal{J}_{k-1} = \{i \mid ([x_{k-1}]_i = l_i \wedge [x_k]_i = l_i) \vee ([x_{k-1}]_i = u_i \wedge [x_k]_i = u_i)\}.$$

The BoxVABB$_{\text{min}}$ step length then reads

$$\alpha_k^{\text{BoxVABB}_{\text{min}}} = \begin{cases} \min\left\{\alpha_j^{\text{BoxBB2}} \,\middle|\, j = \max\{1, k - m_\alpha\}, \ldots, k\right\} & \text{if } \dfrac{\alpha_k^{\text{BoxBB2}}}{\alpha_k^{BB1}} < \tau_k \\ \alpha_k^{\text{BB1}} & \text{otherwise,} \end{cases}$$

where m_α is a nonnegative integer, $\tau_1 \in (0,1)$ and the parameter τ_k is updated in accordance with the following procedure:

$$\tau_{k+1} = \begin{cases} \frac{\tau_k}{\vartheta} & \text{if } \dfrac{\alpha_k^{\text{BoxBB2}}}{\alpha_k^{\text{BB1}}} < \tau_k \\ \vartheta \tau_k & \text{otherwise} \end{cases}$$

with $\vartheta > 1$.

5.2.3.2 Convergence of MPRGP-Type and SPG Methods

We monitored the convergence of the MPRGP, MPPCG, and SPG methods on the generated box-constrained problems described in Section 3.3.1 to better understand the behavior of these methods. The initial guess was set halfway between the box constraints. The reported numbers of Hessian multiplications in the remainder of this section include the Hessian multiplications used by the power method to determine the fixed step length for MPRGP.

The MPRGP and MPPCG variants use the same parameters as those in the previous section. The SPG algorithm parameters are $\alpha_0 = 1$, $\alpha_{min} = 10^{-10}$, $\alpha_{max} = 10^6$, $M = 10$, $\sigma = 10^{-4}$, $\delta = 0.5$, and BoxVABB$_{min}$ step length rule uses $\tau_1 = 0.5$ and $m_\alpha = 2$.

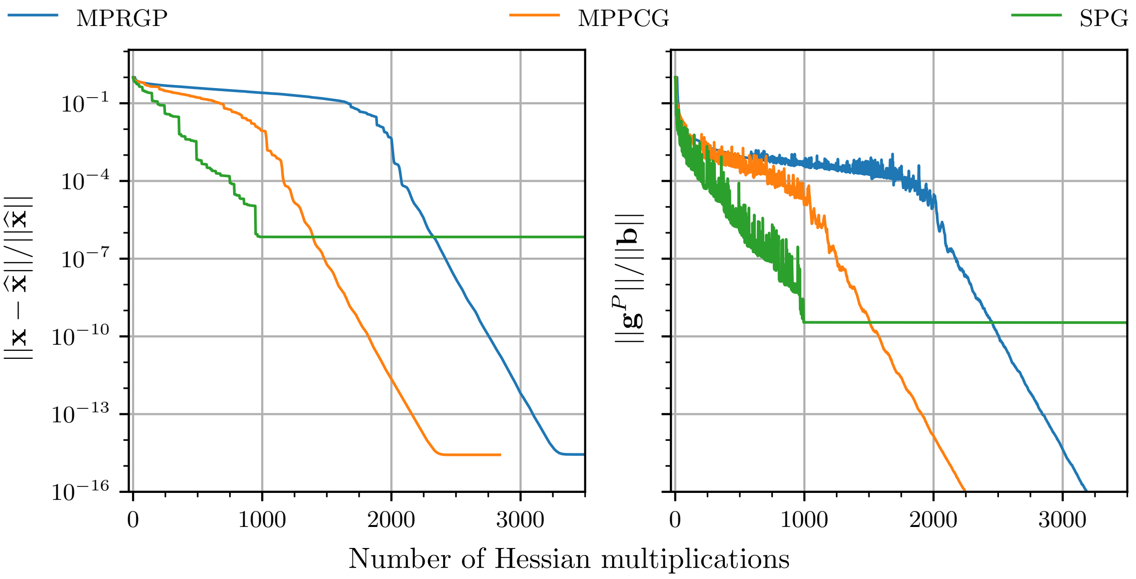

Figure 5.7: BQP1: Convergence curves for MPRGP, MPPCG, and SPG.

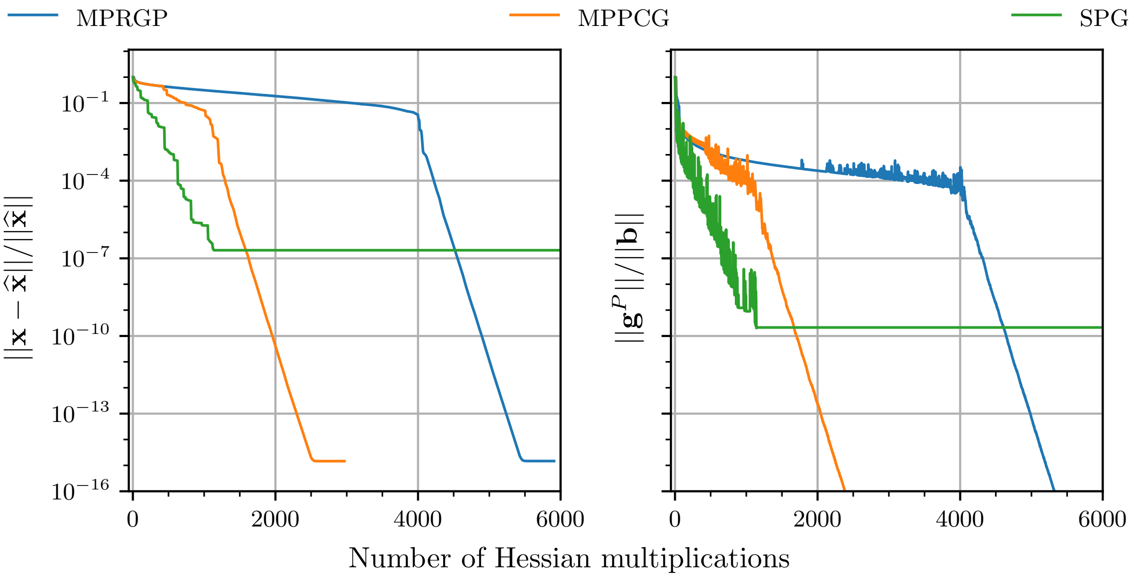

Figure 5.8: BQP2: Convergence curves for MPRGP, MPPCG, and SPG.

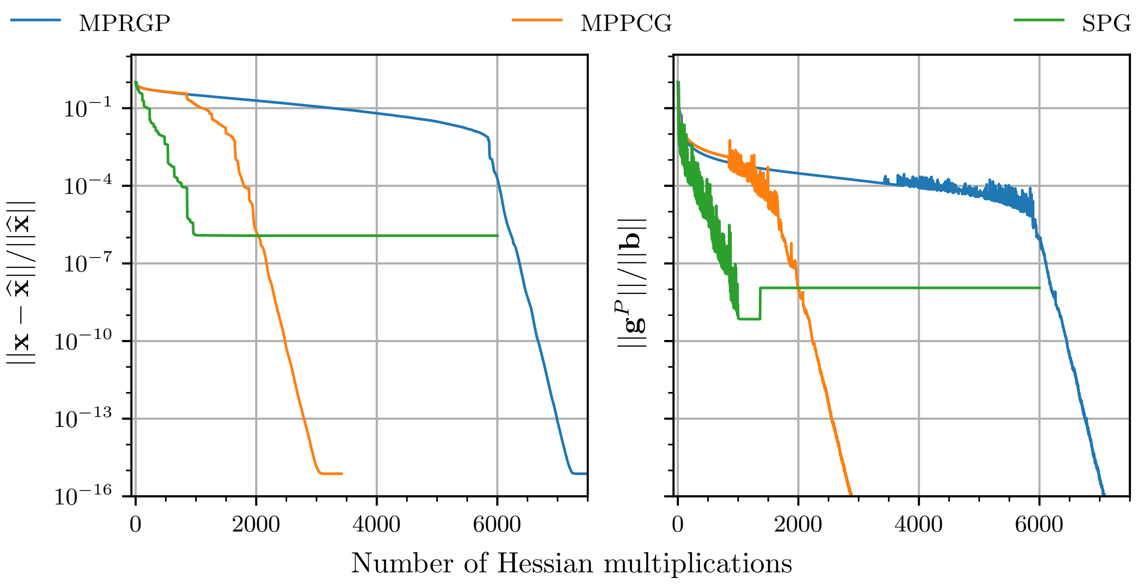

Figure 5.9: BQP3: Convergence curves for MPRGP, MPPCG, and SPG.

Figures 5.7 to 5.9 compare the convergence of each method in terms of the relative norms of the error and the projected gradient.

We can clearly see that the SPG method can achieve the norm of the relative error of only about 10^{-6} and the norm of the relative projected gradient between 10^{-8} and 10^{-10} before it stagnates, while the MPRGP-type algorithms can converge up to the computer precision (IEEE 754 double).

On the other hand, the SPG method converges to its maximum precision in approximately the same number of iterations for each problem, whereas the MPRGP-type methods require more Hessian multiplications with the increasing percentage of active set components, i.e., between problems. The increase in the number of Hessian multiplications is relatively low for the MPPCG method and quite high for the standard MPRGP.

Additionally, we plot detailed convergence curves for each method and problem in Figures 5.10 to 5.18, which include the number of correctly and incorrectly identified active set components. Additionally, we refer to the relative norms of the error and the projected gradient collectively as the relative error on the left y-axis. We note that the SPG method is by far the quickest to identify the active set, and the number of Hessian multiplications required to achieve 99.5% correct active set identification increases fairly slowly with the increasing size of the active set. The MPPCG method is much faster at identifying the active set than the standard MPRGP.

The inability of the SPG method to converge to higher precision seems to be related to the failure in the GLL line search, which is indicated when $\nu \leq eps||\boldsymbol{d}||$, where ν is the step length, eps is the machine epsilon[4], and $\boldsymbol{d}$ is the descent direction. The failure means that, given the direction $\boldsymbol{d}$, there is no meaningfully large step length ν for which there is a sufficient decrease in the cost function. In such a case, we set $\nu_k = eps$. The first failure

[4] $eps = 2^{-52} \approx 2.22\text{e}{-16}$

of the line search is indicated in the convergence figures.

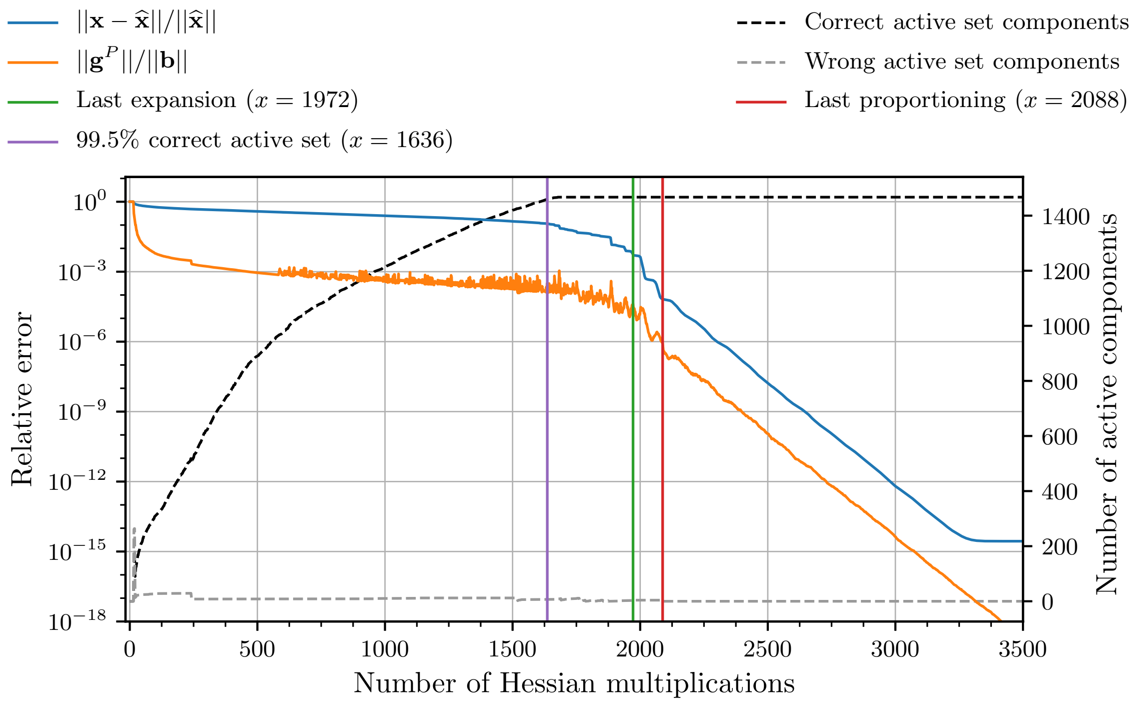

Figure 5.10: BQP1: Progress of MPRGP.

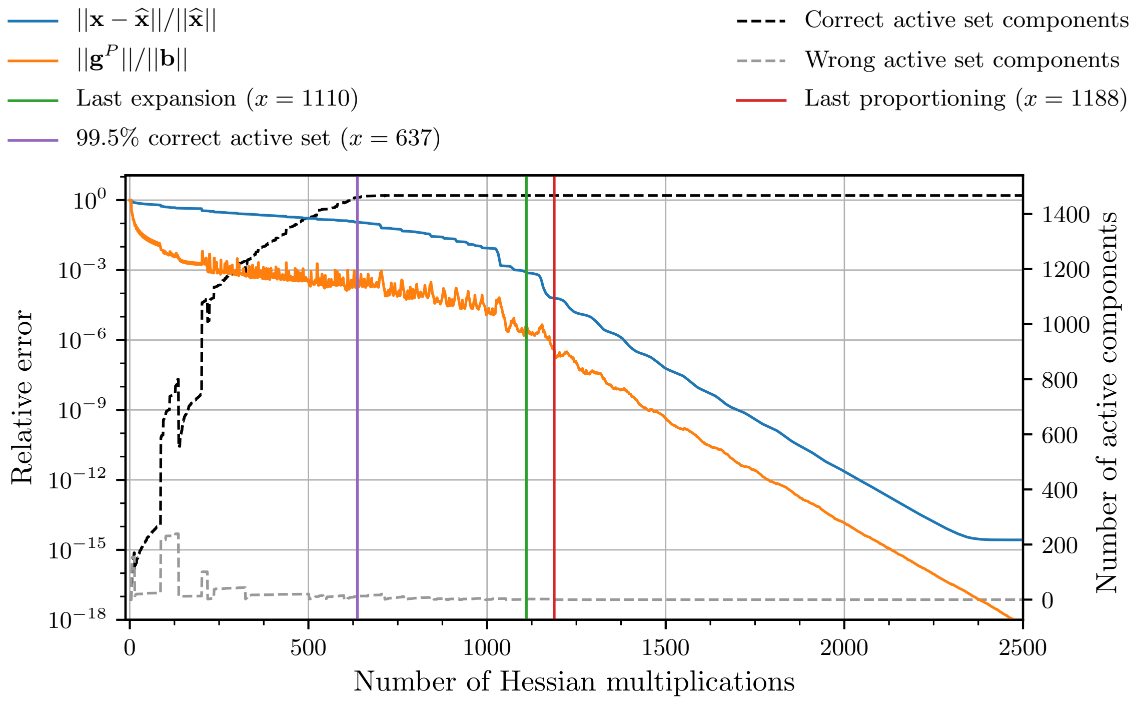

Figure 5.11: BQP1: Progress of MPPCG

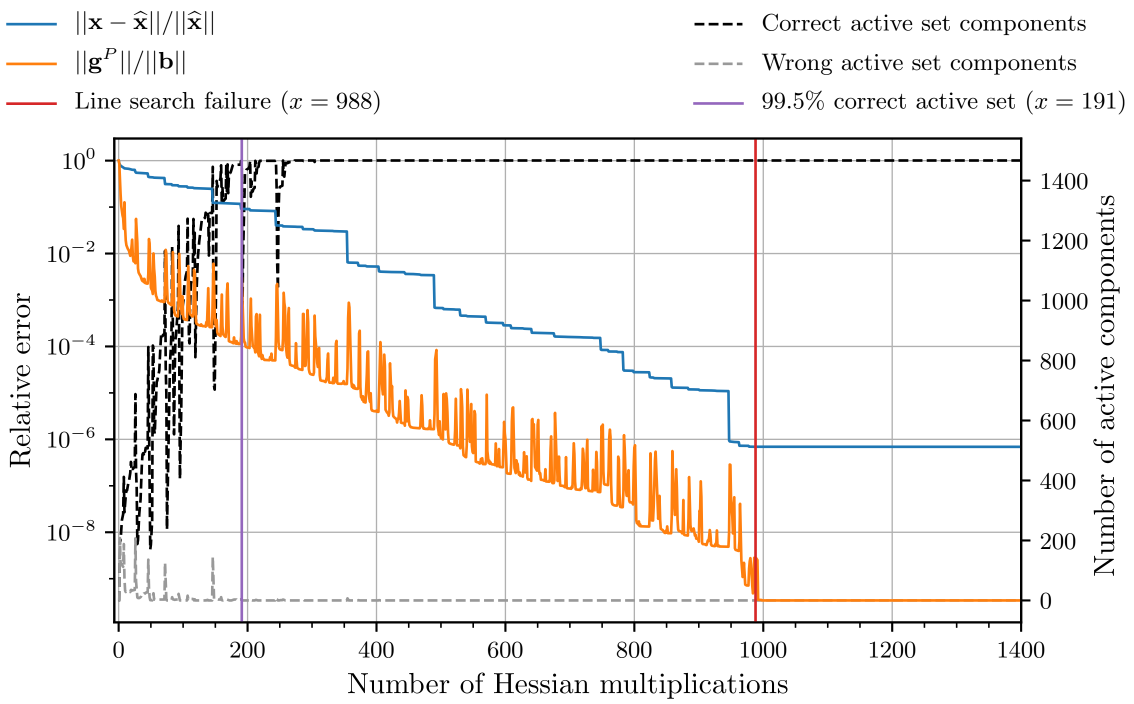

Figure 5.12: BQP1: Progress of SPG.

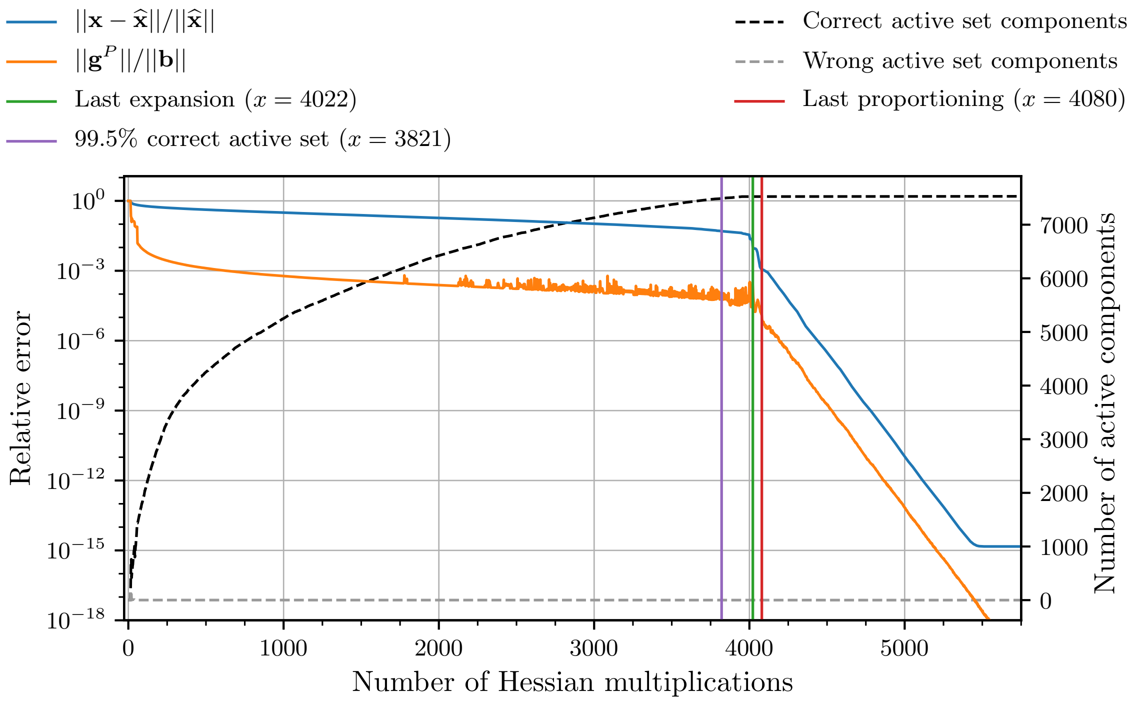

Figure 5.13: BQP2: Progress of MPRGP.

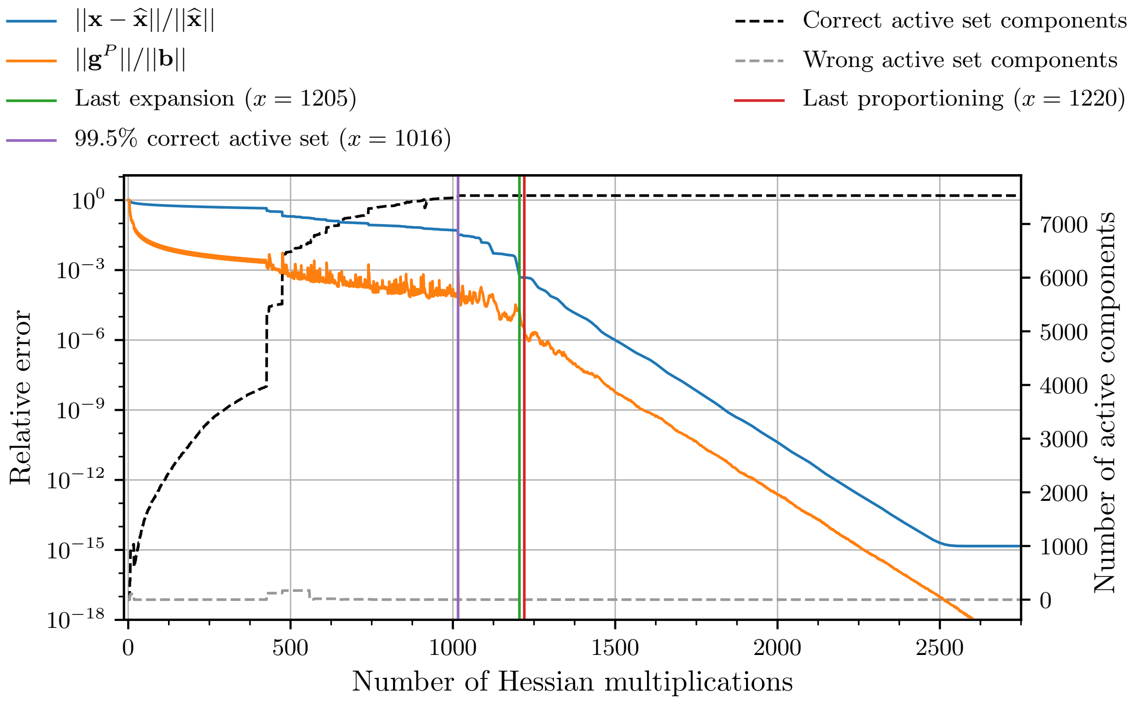

Figure 5.14: BQP2: Progress of MPPCG.

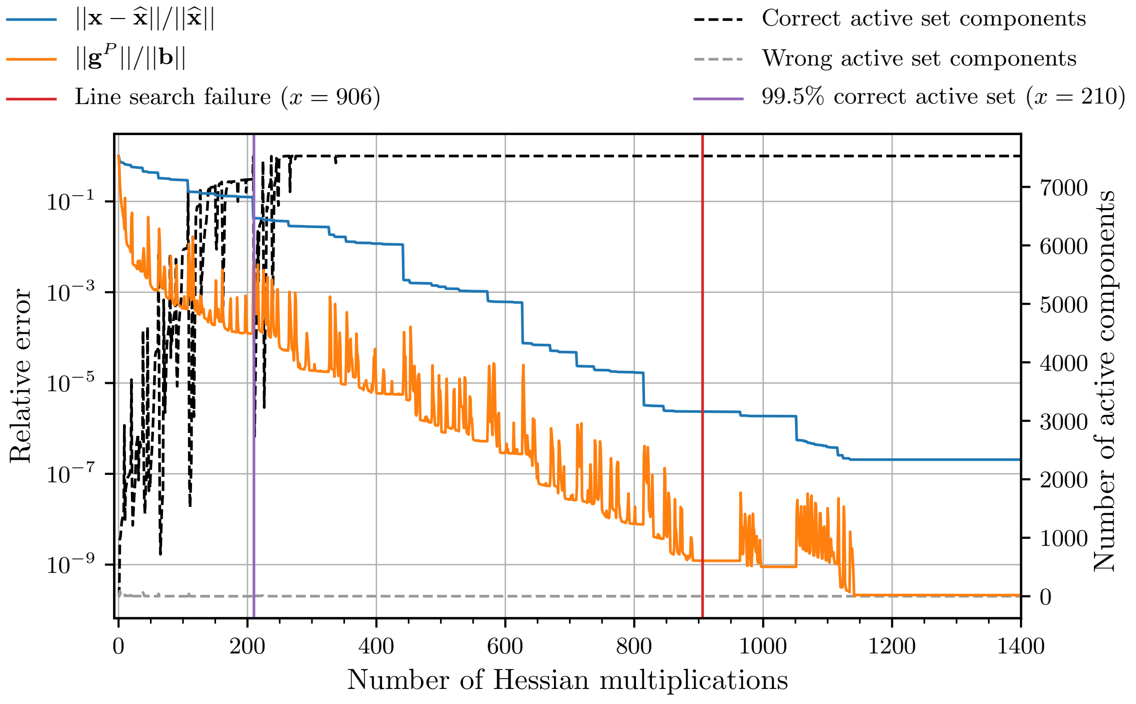

Figure 5.15: BQP2: Progress of SPG.

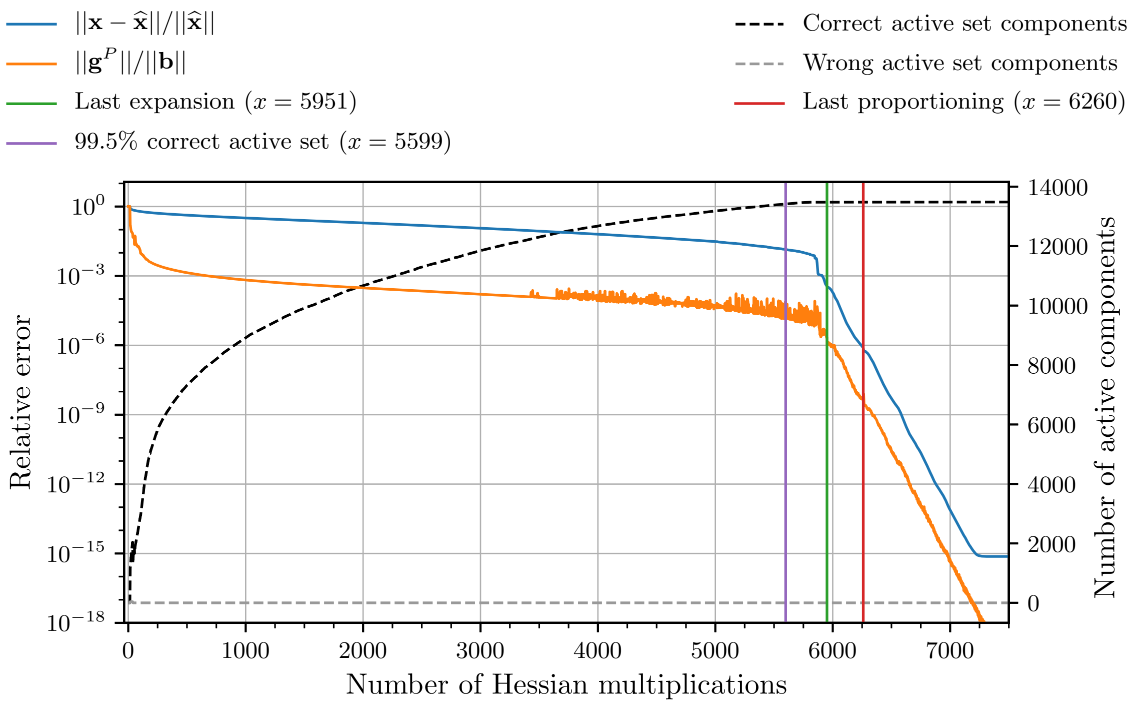

Figure 5.16: BQP3: Progress of MPRGP.

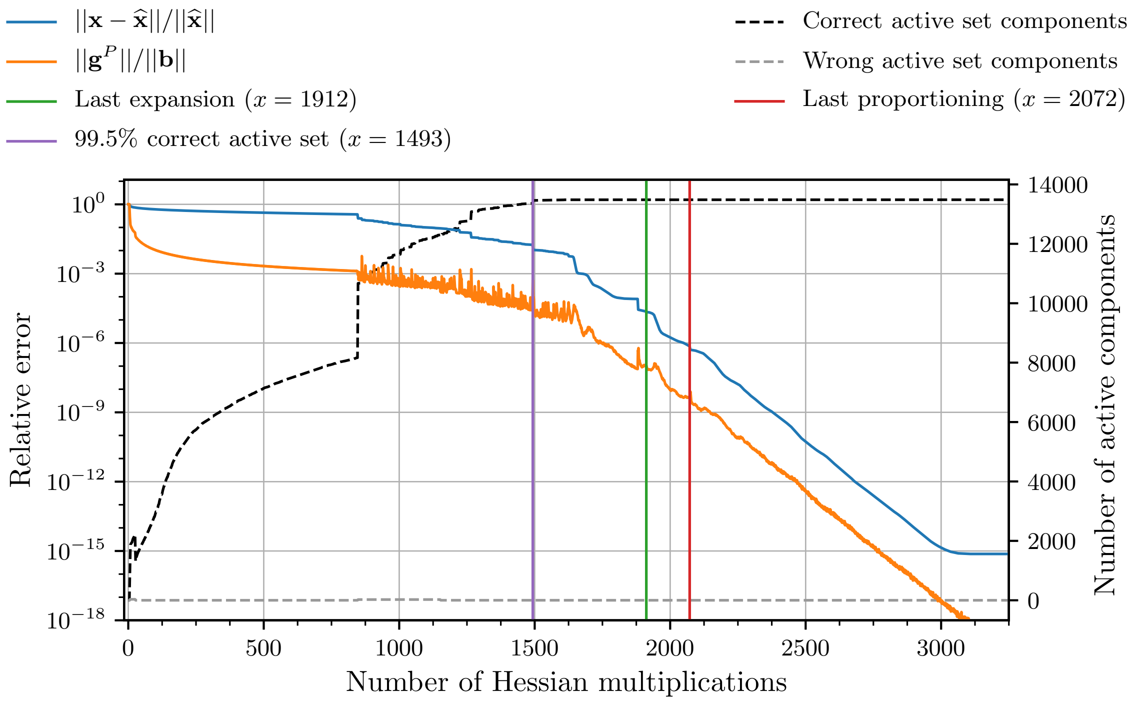

Figure 5.17: BQP3: Progress of MPPCG.

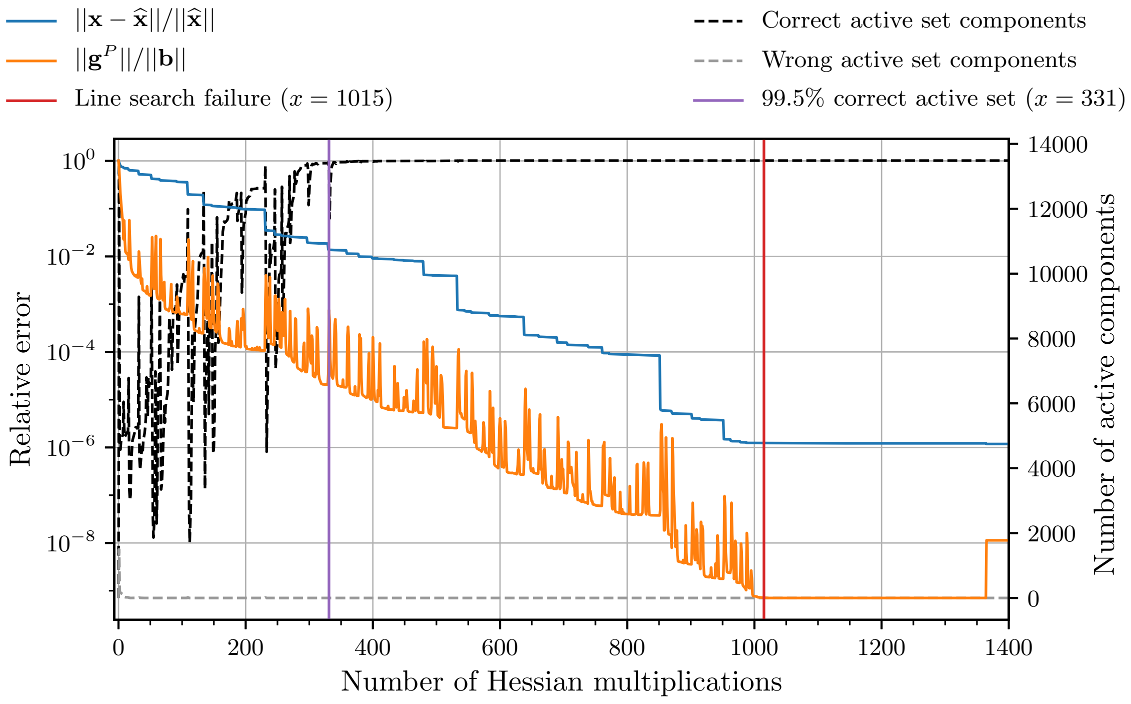

Figure 5.18: BQP3: Progress of SPG.

5.2.3.3 MPSPG Algorithm

The presented results indicate that a hybrid algorithm, leveraging the fast active set identification of the SPG method combined with the efficient unconstrained minimization of the MPRGP method and its capability to converge to high precision, would be of interest.

In principle, there are two possibilities for combining the two methods. The first one is to start with the SPG method and then switch to the MPRGP or MPPCG method. There are many heuristics for the switching criterion, such as reaching some norm of the projected gradient, the active set not changing (sufficiently) for a number of iterations, and the (infinity) norm of the descent direction $\boldsymbol{d}$ or the step length ν being below a threshold. The second option is to replace the expansion step in the MPRGP algorithm with an SPG method.

Here, we present the second variant. We integrate the SPG algorithm by replacing the expansion step in Algorithm 5.1 with Algorithm 5.8, using the same notation and parameters as in Algorithm 5.7. We call the combined method MPSPG (Modified Proportioning with Spectral Projected Gradient). Notice that the expansion step can now also release some components from the active set.

We can omit the expansion half-step in line 1 in Algorithm 5.8 and move the computation of $\boldsymbol{d}_k$ before the computation of α_k^{cg} in line 5 in Algorithm 5.1. This allows us to merge the matrix-vector multiplication of $\boldsymbol{A}\boldsymbol{p}_k$ and $\boldsymbol{A}\boldsymbol{d}_k$ into a single matrix-multiple vectors multiplication, improving the locality of matrix entries and providing additional opportunities for data reuse [85]. We call this variant with merged/fused matrix-vector multiplication MPSPGf. It was shown in [86] that, with a good algorithm, the matrix-

Algorithm 5.8: SPG expansion step

```
1  $\boldsymbol{x}_{k+\frac{1}{2}} = \boldsymbol{x}_k - \alpha_k^{feas}\boldsymbol{p}_k$
2  $\boldsymbol{g}_{k+\frac{1}{2}} = \boldsymbol{g}_k - \alpha_k^{feas}\boldsymbol{A}\boldsymbol{p}_k$
3  define the step length $\alpha_k \in [\alpha_{min}, \alpha_{max}]$ by BoxVABB$_{\min}$
4  $\boldsymbol{d}_k = P_\Omega\left(\boldsymbol{x}_{k+\frac{1}{2}} - \alpha_k \boldsymbol{g}_{k+\frac{1}{2}}\right) - \boldsymbol{x}_{k+\frac{1}{2}}$
5  $\nu_k = 1$
6  $f_{ref} = \max\{f(\boldsymbol{x}_{k+\frac{1}{2}}), f(\boldsymbol{x}_{k-i}),\ 0 \le i \le \min(k, M-1)\}$
7  while $f(\boldsymbol{x}_{k+\frac{1}{2}} + \nu_k \boldsymbol{d}_k) > f_{ref} + \sigma\nu_k(\boldsymbol{g}_{k+\frac{1}{2}})^T\boldsymbol{d}_k$:
8      $\nu_k = \delta\nu_k$
9  $\boldsymbol{x}_{k+1} = \boldsymbol{x}_{k+\frac{1}{2}} + \nu_k \boldsymbol{d}_k$
10 $\boldsymbol{g}_{k+1} = \boldsymbol{g}_{k+\frac{1}{2}} + \nu_k \boldsymbol{A}\boldsymbol{d}_k$
11 $\boldsymbol{p}_{k+1} = \boldsymbol{g}_{k+1}^f$
```

vector multiplication with two vectors achieves about 1.5 times the performance of two matrix-vector multiplications with a single vector on a single thread. In parallel, and especially with a large number of nodes, this speedup factor could be even higher. Our preliminary tests in PETSc suggest that the factor can exceed 2.

A combination of the MPGP and SPG methods was proposed earlier in [87]. However, the combination replaced the proportioning step with a variant of the SPG method. Since, in our numerical experiments, the proportioning step is the least frequent step, we did not consider any modifications to it. Analogous to our MPSPG algorithm, the MPGP with SPG proportioning can both add and release the components of the active set in the proportioning step.

5.2.3.4 Results

We use the BQP examples from the previous section and examples from the PERMON library as our benchmarks. Specifically, the PERMON examples used are the 1D Poisson's contact problems *ex1*, *ex2* (Section 3.3.2), and the journal bearing problem *jbearing2* (Section 3.3.3).

For each benchmark and solver, unless otherwise specified, we set the relative tolerance to $rtol = 10^{-4}$. All other parameters are kept the same as in the previous sections.

The results for each tested variant and benchmark are presented in Table 5.4. We compare only the number of Hessian multiplications performed by each algorithm. In the MPSPGf algorithm with the fused matrix-vector multiplication, we count the matrix times two vectors as a single Hessian multiplication and report the number of matrix-vector multiplications not counted in brackets. We leave it to the reader to assess the cost factor of the matrix multiplying two vectors for their specific setting and machine. The variant with matrix-vector multiplication counted once will be referred to as fused-best, while the variant with the cost of 1.5 of multiplying two vectors compared to multiplying a single vector will be referred to as fused-realistic. The table also contains geometric means (GM) of the speedups compared to the standard MPRGP algorithm. The geometric means for the fused algorithms are reported for the fused-best variant, and then in brackets for the fused-realistic variant.

Problem	Notes	MPRGP	MPPCG	SPG	MPSPG	MPSPGf
BQP1	rtol: 10^{-4}	1726	714	208	262	219 (217)
BQP2	rtol: 10^{-4}	3174	799	205	273	266 (265)
BQP3	rtol: 10^{-4}	3910	1268	248	401	234 (231)
BQP1	rtol: 10^{-6}	2075	1175	579	745	592 (590)
BQP2	rtol: 10^{-6}	4299	1230	522	744	726 (465)
BQP3	rtol: 10^{-6}	6081	1735	604	1224	692 (688)
GM of BQP speedups		1	2.94	9.19	6.25	8.12 (5.54)
ex1	grid: 100	177	164	249	157	167 (165)
ex1	grid: 1000	3245	3037	2709	3946	2918 (2882)
ex1	grid: 5000	31657	25673	19806	74756	22541 (22232)
ex2	grid: 100	208	200	369	158	181 (178)
ex2	grid: 1000	2825	3366	3077	2993	2361 (2348)
ex2	grid: 5000	21525	16103	24505	20227	20767 (20543)
jbearing	grid: 50x50	154	142	154	135	139 (129)
jbearing	grid: 100x100	318	335	358	346	322 (307)
jbearing	grid: 200x50	663	664	740	803	690 (659)
jbearing	grid: 400x25	1463	1559	1597	1700	1443 (1383)
GM PERMON ex. speedups		1	1.05	0.92	0.91	1.10 (0.74)
Overall GM of speedups		1	1.54	2.19	1.87	2.32 (1.57)

Table 5.4: Comparison of the number of Hessian multiplications. The note *grid* refers to the number of discretization points in 1D or 2D.

The results are divided into the BQP problems, which have high ratios of expansion steps to the overall number of steps (between 59% and 99% with an average of 80%) in the MPRGP method, and into PERMON examples, which have high ratios of CG steps (between 84% and 94% with an average of 91%).

Therefore, the SPG method and the MPRGP variants with faster expansion schemes perform exceptionally well on the BQP problems, achieving a geometric mean of speedups between 2.9 and 9.2.

On the other hand, all methods, except the fused-realistic variant, have an overall performance within 10% of the MPRGP method in the PERMON examples. Only the projected CG variant and the fused-best variant have slightly higher performance than the standard MPRGP.

Overall, due to the huge gains for the BQP problems, the SPG method and the new variants perform better, with the geometric mean of speedups between 1.5 and 2.3.

While the SPG method achieved the second-highest overall speedup, it does not converge to high precision. The new methods combining the MPRGP and SPG methods do not suffer, at least on the tested BQP benchmarks, from this phenomenon. We report the results on the BQP problems with the relative tolerance $rtol = 10^{-12}$ in Table 5.5, where the standalone SPG method does not converge.

Problem	MPRGP	MPPCG	MPSPG	MPSPGf
BQP1	2750	1787	1422	1274 (1272)
BQP2	4951	1932	1402	1249 (1248)
BQP3	6797	2463	1923	1360 (1355)
GM of BQP speedups	1	2.22	2.89	3.50 (2.33)

Table 5.5: Comparison of the number of Hessian multiplications on the BQP problems with the relative tolerance of 10^{-12}.

The results are as expected when considering the convergence curves presented in Section 5.2.3.2. Notably, the overall speedup of the new methods decreases as the ratio of CG steps to the overall number of steps needed to achieve convergence increases. Nevertheless, the presented expansion modifications maintain a large geometric mean of speedups between 2.2 and 3.5.

5.3 Preconditioning MPRGP-Type Methods

As discussed in Section 4.2.3, preconditioning can significantly accelerate the CG method. However, applying preconditioners to constrained QP problems is not straightforward. For simplicity, let us consider only a lower bound-constrained QP

$$\arg\min_{\boldsymbol{x}} \frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} - \boldsymbol{x}^T\boldsymbol{b} \quad \text{s.t.} \quad \boldsymbol{l} \leq \boldsymbol{x}.$$

Let us consider an SPD preconditioner matrix $\boldsymbol{M}$ and the application of a preconditioner with this matrix as $\boldsymbol{M}^{-1}$, as in Section 4.2.4. Using the split preconditioning to preserve the symmetry of the Hessian, the cost function is transformed into

$$\frac{1}{2}\widehat{\boldsymbol{x}}^T\boldsymbol{L}^{-1}\boldsymbol{A}\boldsymbol{L}^{-T}\widehat{\boldsymbol{x}} - \widehat{\boldsymbol{x}}^T\boldsymbol{L}^{-1}\boldsymbol{b},$$

where $\boldsymbol{M} = \boldsymbol{L}\boldsymbol{L}^T$ and $\boldsymbol{x} = \boldsymbol{L}^{-T}\widehat{\boldsymbol{x}}$. Due to the variable change, the bound constraints are transformed into general linear inequality constraints[5]

$$\boldsymbol{l} \leq \boldsymbol{L}^{-T}\widehat{\boldsymbol{x}}.$$

QP problems with linear inequality constraints are typically much more difficult to solve.

We will show in the following subsections how to incorporate preconditioning that does not change the type of the constraints into the MPRGP-type methods.

5.3.1 Preconditioning in Face

Preconditioning in face was introduced in [88] for the Polyak algorithm [89], and its use is described for the MPRGP algorithm in [10].

[5]Unless $\boldsymbol{L}^{-T}$ is diagonal, e.g., when using a diagonal scaling preconditioner.

The idea is to apply the preconditioning only on the free set. In order to achieve this, we split the preconditioner matrix according to the free set and the active set

$$\overline{\boldsymbol{M}} = \begin{pmatrix} \boldsymbol{M}_{\mathcal{F}\mathcal{F}} & \boldsymbol{M}_{\mathcal{F}\mathcal{A}} \\ \boldsymbol{M}_{\mathcal{A}\mathcal{F}} & \boldsymbol{M}_{\mathcal{A}\mathcal{A}} \end{pmatrix}.$$

Then only the free gradient is preconditioned by a preconditioner computed solely on the free set

$$\boldsymbol{z} = \begin{pmatrix} \boldsymbol{z}^f_{\mathcal{F}} \\ \boldsymbol{o} \end{pmatrix} = \boldsymbol{M}^{-1} \begin{pmatrix} \boldsymbol{g}^f_{\mathcal{F}} \\ \boldsymbol{o} \end{pmatrix} = \begin{pmatrix} \boldsymbol{M}^{-1}_{\mathcal{F}\mathcal{F}} & \boldsymbol{o} \\ \boldsymbol{o} & \boldsymbol{o} \end{pmatrix} \begin{pmatrix} \boldsymbol{g}^f_{\mathcal{F}} \\ \boldsymbol{o} \end{pmatrix}, \tag{5.5}$$

where $\boldsymbol{M}^{-1}$ is the application of the preconditioning in face, while $\boldsymbol{M}^{-1}_{\mathcal{F}\mathcal{F}}$ is an application of some standard preconditioner like incomplete Cholesky. The preconditioning in face gives a sort of preconditioned free gradient $\boldsymbol{z}^f$. Note that the vectors are usually not reordered in actual implementations.

Obviously, the major drawback is that the preconditioner must be recomputed or at least updated every time the free set changes. One way to avoid recomputing the preconditioner is to restrict the preconditioner not to the current free set, but to the set that will never be active. Then the preconditioner needs to be computed only once. For example, if only a part of the solution vector is constrained, the preconditioner can be computed and applied only to the unconstrained part. Such problems arise in, e.g., contact problems. In the particular case of the 3D cuboid contact of Section 3.3.4 with $n \times n \times n$ unknowns, only n^2 unknowns, i.e., at most $1/n$ of all unknowns, can become active. This allows us to apply preconditioning to the remaining $(n-1)\, n^2$ unknowns. In the following subsection, we develop an alternative preconditioning method that avoids the need to recompute the preconditioner without prior knowledge of the set that will never be active.

5.3.2 Approximate Preconditioning in Face

To avoid the need to recompute the preconditioner, it is possible to apply the full preconditioner, which is denoted $\overline{\boldsymbol{M}}^{-1}$, computed for the entire preconditioning matrix $\overline{\boldsymbol{M}}$, and then zero out the active set components

$$\boldsymbol{z} = \begin{pmatrix} \tilde{\boldsymbol{z}}^f_{\mathcal{F}} \\ \boldsymbol{o} \end{pmatrix} = \boldsymbol{M}^{-1} \begin{pmatrix} \boldsymbol{g}^f_{\mathcal{F}} \\ \boldsymbol{o} \end{pmatrix} = \text{gradientSplit}_{Free}(\overline{\boldsymbol{M}}^{-1} \begin{pmatrix} \boldsymbol{g}^f_{\mathcal{F}} \\ \boldsymbol{o} \end{pmatrix}),$$

where the function $\text{gradientSplit}_{Free}()$ zeros out the active set components of a given vector in the same way as computing the free gradient in Equation (5.1). The operator $\boldsymbol{M}^{-1}$ is again the application of the preconditioner, which we call the *approximate preconditioning in face* because it tries to approximate the preconditioned free gradient $\boldsymbol{z}^f$ from Equation (5.5). The operator $\overline{\boldsymbol{M}}^{-1}$ is the application of some standard preconditioner, disregarding any information about the free set.

Assuming $\overline{\boldsymbol{M}} = \boldsymbol{A}$ and the application of the preconditioner is the actual inverse, i.e., any matrix inverse notation for the rest of this subsection represents the inverse of the matrix and not an application of some preconditioner, then the approximate preconditioner corresponds to the preconditioning by the Schur complement eliminating the active set variables

$$\text{gradientSplit}_{Free}(\overline{\boldsymbol{M}}^{-1}\begin{pmatrix}\boldsymbol{g}^f_{\mathcal{F}}\\ \boldsymbol{o}\end{pmatrix}) = \begin{pmatrix}(\boldsymbol{M}_{\mathcal{F}\mathcal{F}} - \boldsymbol{M}_{\mathcal{F}\mathcal{A}}\boldsymbol{M}^{-1}_{\mathcal{A}\mathcal{A}}\boldsymbol{M}_{\mathcal{A}\mathcal{F}})^{-1}\boldsymbol{g}^f_{\mathcal{F}}\\ \boldsymbol{o}\end{pmatrix} = \begin{pmatrix}\boldsymbol{S}^{-1}\boldsymbol{g}^f_{\mathcal{F}}\\ \boldsymbol{o}\end{pmatrix}.$$

Moreover, by expanding the expression further, we obtain

$$\begin{aligned}\begin{pmatrix}\boldsymbol{S}^{-1}\boldsymbol{g}^f_{\mathcal{F}}\\ \boldsymbol{o}\end{pmatrix} &= \begin{pmatrix}(\boldsymbol{M}^{-1}_{\mathcal{F}\mathcal{F}} + \boldsymbol{M}^{-1}_{\mathcal{F}\mathcal{F}}\boldsymbol{M}_{\mathcal{F}\mathcal{A}}(\boldsymbol{M}_{\mathcal{A}\mathcal{A}} - \boldsymbol{M}_{\mathcal{A}\mathcal{F}}\boldsymbol{M}^{-1}_{\mathcal{F}\mathcal{F}}\boldsymbol{M}_{\mathcal{F}\mathcal{A}})^{-1}\boldsymbol{M}_{\mathcal{A}\mathcal{F}}\boldsymbol{M}^{-1}_{\mathcal{F}\mathcal{F}})\boldsymbol{g}^f_{\mathcal{F}}\\ \boldsymbol{o}\end{pmatrix}\\ &= \begin{pmatrix}(\boldsymbol{I} + \boldsymbol{M}^{-1}_{\mathcal{F}\mathcal{F}}\boldsymbol{M}_{\mathcal{F}\mathcal{A}}(\boldsymbol{M}_{\mathcal{A}\mathcal{A}} - \boldsymbol{M}_{\mathcal{A}\mathcal{F}}\boldsymbol{M}^{-1}_{\mathcal{F}\mathcal{F}}\boldsymbol{M}_{\mathcal{F}\mathcal{A}})^{-1}\boldsymbol{M}_{\mathcal{A}\mathcal{F}})\boldsymbol{M}^{-1}_{\mathcal{F}\mathcal{F}}\boldsymbol{g}^f_{\mathcal{F}}\\ \boldsymbol{o}\end{pmatrix}.\end{aligned}$$

Applying the preconditioner to the Hessian restricted to the free set instead of the free gradient would result in

$$\boldsymbol{S}^{-1}\boldsymbol{A}_{\mathcal{F}\mathcal{F}} = \boldsymbol{I} + \boldsymbol{M}^{-1}_{\mathcal{F}\mathcal{F}}\boldsymbol{M}_{\mathcal{F}\mathcal{A}}(\boldsymbol{M}_{\mathcal{A}\mathcal{A}} - \boldsymbol{M}_{\mathcal{A}\mathcal{F}}\boldsymbol{M}^{-1}_{\mathcal{F}\mathcal{F}}\boldsymbol{M}_{\mathcal{F}\mathcal{A}})^{-1}\boldsymbol{M}_{\mathcal{A}\mathcal{F}}.$$

Given $r = \text{rank}(\boldsymbol{M}_{\mathcal{A}\mathcal{F}})$, the eigenvalues of the preconditioned operator $\boldsymbol{S}^{-1}\boldsymbol{A}_{\mathcal{F}\mathcal{F}}$ are

$$1 = \lambda_1 = \cdots = \lambda_{n-r} \leq \cdots \leq \lambda_n.$$

The eigenvalues of the preconditioning in face would be equal to ones. Therefore, the difference between the two preconditioners is only in $\text{rank}(\boldsymbol{M}_{\mathcal{A}\mathcal{F}})$ eigenvalues and the term

$$\boldsymbol{M}^{-1}_{\mathcal{F}\mathcal{F}}\boldsymbol{M}_{\mathcal{F}\mathcal{A}}(\boldsymbol{M}_{\mathcal{A}\mathcal{A}} - \boldsymbol{M}_{\mathcal{A}\mathcal{F}}\boldsymbol{M}^{-1}_{\mathcal{F}\mathcal{F}}\boldsymbol{M}_{\mathcal{F}\mathcal{A}})^{-1}\boldsymbol{M}_{\mathcal{A}\mathcal{F}}$$

can be thought of as the error of the approximate preconditioning in face compared to the standard preconditioning in face.

To illustrate the previous result, we plotted in Figure 5.19 the eigenvalues for the journal bearing problem (Section 3.3.3) in the first iteration with the zero initial guess. The difference between the preconditioning in face and its approximate variant is precisely in the last 50 eigenvalues, since $\text{rank}(\boldsymbol{M}_{\mathcal{A}\mathcal{F}}) = 50$. We note that those last 50 eigenvalues are spaced throughout the interval starting at 1 and ending with some maximal eigenvalue.

To see how the condition number and the rank of the off-diagonal matrix $\boldsymbol{M}_{\mathcal{A}\mathcal{F}}$ change throughout the iterative process, we plotted these quantities together with the free set size for the 1D Poisson's contact problem *ex1* (Section 3.3.2) and for the journal bearing problem in Figures 5.20 and 5.21. Apart from the first iteration for the *ex1* problem, the condition number of the preconditioned operator remained constant and was always significantly lower than the condition number of the unpreconditioned operator. The off-diagonal matrix rank was at most 4 for the 1D Poisson's problem and grew moderately from 25 to the maximum of 59 for the journal bearing problem, which represented only a tiny fraction of the free set size where the preconditioning is applied.

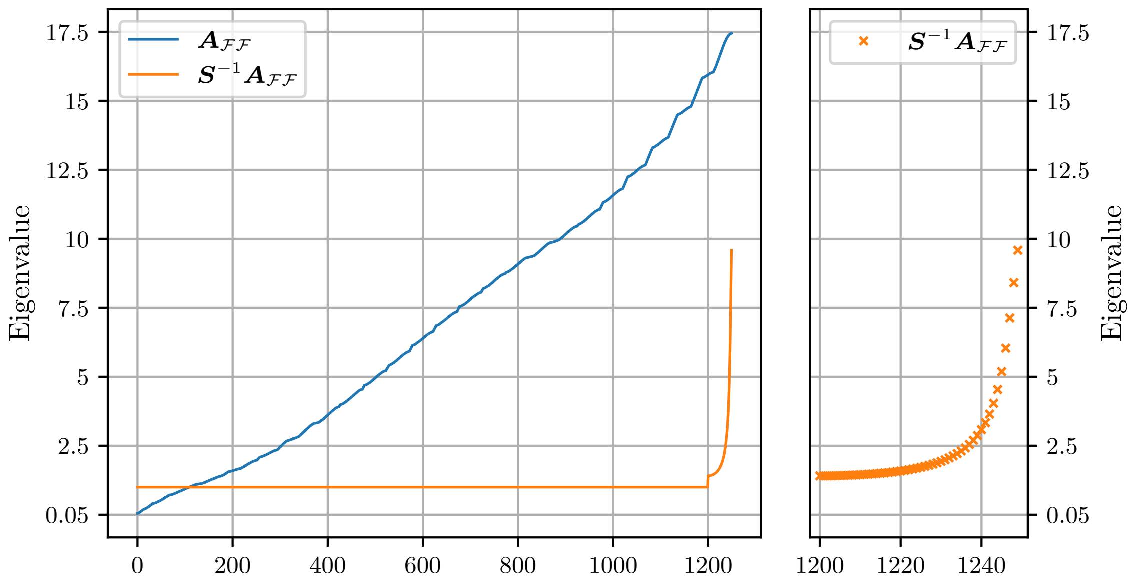

Figure 5.19: Eigenvalues of the journal bearing problem with 50x50 grid points (2,500 DOFs) preconditioned by the inverse matrix at iteration 0 (the free set size is 1,250, and the rank of the off-diagonal block is 50).

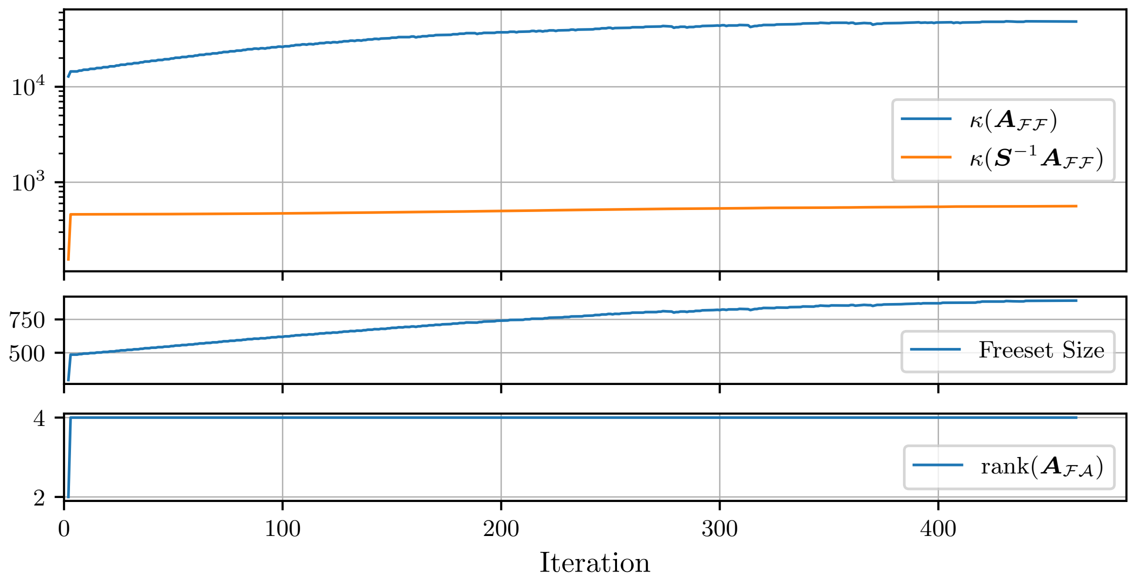

Figure 5.20: Condition number, free set size, and rank of the off-diagonal block for *ex1* with 1,000 DOFs preconditioned by the inverse matrix.

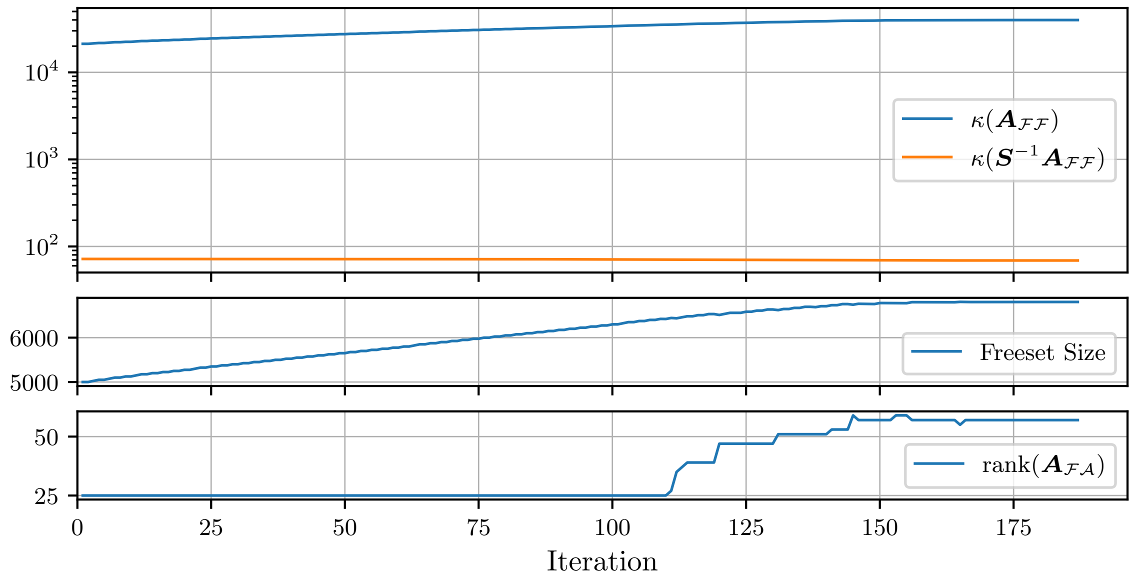

Figure 5.21: Condition number, free set size, and rank of the off-diagonal block for the journal bearing problem with 400x25 grid points (10,000 DOFs) preconditioned by the inverse matrix.

5.3.3 Preconditioned MPRGP and MPPCG Methods

The preconditioning is incorporated into the MPRGP-type methods in the same way as the preconditioning for the steepest descent and CG methods is incorporated; see Section 4.2.4. For completeness, the preconditioned MPRGP algorithm can be found in Algorithm 5.9. The preconditioned MPPCG method is obtained by replacing the *if...else* block on lines 6–9 in Algorithm 5.9 with Algorithm 5.13.

Algorithm 5.9: Preconditioned MPRGP

```
   Input: $\boldsymbol{A}$, $\boldsymbol{M}^{-1}$, $\boldsymbol{x}_0 \in \Omega$, $\boldsymbol{b}$, $\Gamma > 0$, $\overline{\alpha} \in (0, 2||\boldsymbol{A}||^{-1}]$
1  $\boldsymbol{g}_0 = \boldsymbol{A}\boldsymbol{x}_0 - \boldsymbol{b}$, $\boldsymbol{z}_0 = \boldsymbol{M}^{-1}\boldsymbol{g}_0^f$, $\boldsymbol{p}_0 = \boldsymbol{z}_0$, $k = 0$
2  while $||\boldsymbol{g}_k^P||$ is not small:
3      if $||\boldsymbol{g}_k^c||^2 \leq \Gamma^2 ||\boldsymbol{g}_k^f||^2$:
4          $\alpha_k^{feas} = \max\{\alpha \mid \boldsymbol{x}_k - \alpha \boldsymbol{p}_k \in \Omega\}$
5          $\alpha_k^{cg} = \boldsymbol{g}_k^T \boldsymbol{z}_k / \boldsymbol{p}_k^T \boldsymbol{A} \boldsymbol{p}_k$
6          if $\alpha_k^{cg} \leq \alpha_k^{feas}$:
7              Preconditioned CG step - Algorithm 5.10
8          else:
9              Preconditioned expansion step - Algorithm 5.11;
10     else:
11         Preconditioned proportioning step - Algorithm 5.12;
12     $k = k + 1$
   Output: $\boldsymbol{x}_k$
```

Algorithm 5.10: Preconditioned CG step

```
1 $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k^{cg}\boldsymbol{p}_k$
2 $\boldsymbol{g}_{k+1} = \boldsymbol{g}_k - \alpha_k^{cg}\boldsymbol{A}\boldsymbol{p}_k$
3 $\boldsymbol{z}_{k+1} = \boldsymbol{M}^{-1}\boldsymbol{g}_{k+1}^f$
4 $\beta_k = \boldsymbol{p}_k^T\boldsymbol{A}\boldsymbol{z}_{k+1}/\boldsymbol{p}_k^T\boldsymbol{A}\boldsymbol{p}_k$
5 $\boldsymbol{p}_{k+1} = \boldsymbol{z}_{k+1} - \beta_k\boldsymbol{p}_k$
```

Algorithm 5.11: Preconditioned expansion step

```
1 $\boldsymbol{x}_{k+\frac{1}{2}} = \boldsymbol{x}_k - \alpha_k^{feas}\boldsymbol{p}_k$
2 $\boldsymbol{g}_{k+\frac{1}{2}} = \boldsymbol{g}_k - \alpha_k^{feas}\boldsymbol{A}\boldsymbol{p}_k$
3 $\boldsymbol{x}_{k+1} = P_\Omega(\boldsymbol{x}_{k+\frac{1}{2}} - \overline{\alpha}\boldsymbol{g}_k^f)$
4 $\boldsymbol{g}_{k+1} = \boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{b}$
5 $\boldsymbol{z}_{k+1} = \boldsymbol{M}^{-1}\boldsymbol{g}_{k+1}^f$
6 $\boldsymbol{p}_{k+1} = \boldsymbol{z}_{k+1}$
```

Algorithm 5.12: Preconditioned proportioning step

```
1 $\alpha_k^{sd} = \boldsymbol{g}_k^T\boldsymbol{g}_k^c/(\boldsymbol{g}_k^c)^T\boldsymbol{A}\boldsymbol{g}_k^c$
2 $\alpha_k^{feas} = \max\{\alpha \mid \boldsymbol{x}_k - \alpha\boldsymbol{g}_k^c \in \Omega\}$
3 if $\alpha_k^{feas} < \alpha_k^{sd}$:
4     $\alpha_k^{sd} = \alpha_k^{feas}$
5 $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k^{sd}\boldsymbol{g}_k^c$
6 $\boldsymbol{g}_{k+1} = \boldsymbol{g}_k - \alpha_k^{sd}\boldsymbol{A}\boldsymbol{g}_k^c$
7 $\boldsymbol{z}_{k+1} = \boldsymbol{M}^{-1}\boldsymbol{g}_{k+1}^f$
8 $\boldsymbol{p}_{k+1} = \boldsymbol{z}_{k+1}$
```

Algorithm 5.13: Preconditioned projected CG step

```
1 $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k^{cg}\boldsymbol{p}_k$
2 if $\alpha_k^{cg} \leq \alpha^{feas}$:
3     $\boldsymbol{g}_{k+1} = \boldsymbol{g}_k - \alpha_k^{cg}\boldsymbol{A}\boldsymbol{p}_k$
4     $\boldsymbol{z}_{k+1} = \boldsymbol{M}^{-1}\boldsymbol{g}_{k+1}^f$
5     $\beta_k = \boldsymbol{p}_k^T\boldsymbol{A}\boldsymbol{z}_{k+1}/\boldsymbol{p}_k^T\boldsymbol{A}\boldsymbol{p}_k$
6     $\boldsymbol{p}_{k+1} = \boldsymbol{z}_{k+1} - \beta_k\boldsymbol{p}_k$
7 else:
8     $\boldsymbol{x}_{k+1} = P_\Omega(\boldsymbol{x}_{k+1})$
9     $\boldsymbol{g}_{k+1} = \boldsymbol{A}\boldsymbol{x}_{k+1} - \boldsymbol{b}$
10    $\boldsymbol{z}_{k+1} = \boldsymbol{M}^{-1}\boldsymbol{g}_{k+1}^f$
11    $\boldsymbol{p}_{k+1} = \boldsymbol{z}_{k+1}$
```

5.3.4 Results

The presented results were computed on a single core of the LUMI supercomputer (Section 3.2.2). The stopping criterion was the relative tolerance of 10^{-10}. As far as we know, these are the only results of the MPRGP-type algorithm showcasing the preconditioning in face (results for partially constrained problems using deflation can be found in [10, 90, 91]).

Standard preconditioners available in PETSc with the default options were used to compute the results. The Cholesky preconditioner is the application of the direct solver, i.e., preconditioning by the inverse of the Hessian, using MUMPS. ICC is the incomplete Cholesky factorization [92], and SSOR is the symmetric successive over-relaxation [93].

The CG method, applied to the system of linear equations preconditioned by the inverse of the Hessian, will converge in a single iteration. That is not the case for the preconditioned MPRGP-type methods because the active set needs to be identified. However, if we start with the correct active set or once the correct active set is identified, the preconditioned method converges to machine precision in a single iteration. In any case, since the inverse preconditioner is the optimal preconditioner for the preconditioning in face in the sense of preconditioning the Hessian on the free set, the number of Hessian multiplications for the Cholesky preconditioner is a very interesting metric[6].

Due to the inclusion of the preconditioner in each iteration, the number of Hessian multiplications, while still of interest, cannot be used as a metric for comparison between the methods. Therefore, timings and speedups based on the timings are provided.

The results for a number of problem sizes are presented in Tables 5.6 and 5.7 and Tables 5.8 to 5.11 for the 3D linear elasticity cube contact problem[7] described in Section 3.3.4 and for the journal bearing problem described in Section 3.3.3, respectively. Columns S_b and S_M contain speedups. S_b is computed with respect to the same unpreconditioned method, while S_M is computed with respect to the unpreconditioned MPRGP method.

First, examining the performance of the standard MPRGP with the preconditioning in face, the number of Hessian multiplications is significantly reduced compared to the unpreconditioned method. This reduction is driven by a large decrease in the CG steps, which is precisely what we would expect. The number of expansion steps appears to be proportional to the preconditioner effectiveness, being the lowest for the Cholesky preconditioner, followed by ICC, and finally SSOR. Compared to the unpreconditioned method, the number of expansions was typically lower for the journal bearing problem and higher for the elasticity problem. The number of proportioning steps follows similar trends as the expansion steps, but the change between the preconditioners is much less pronounced.

As for the new approximate preconditioning in face combined with the standard MPRGP, the effectiveness in reducing the number of CG iterations is still there. However, there is a further increase in the number of expansion steps, which is very noticeable in

[6]Although it cannot be taken as the lower bound on the number of Hessian multiplications needed by the preconditioned MPRGP-type algorithm, despite this being the case in our numerical experiments.

[7]The contact condition is represented by upper bound constraints.

journal bearing problems. The cause of the increase could be driven by the decrease in the effectiveness of the approximate preconditioning in face compared to the standard preconditioning in face. Overall, the number of Hessian multiplications usually increases compared to the standard preconditioning in face, especially for larger journal bearing problems. Despite this, the time needed by the approximate preconditioning in face is significantly lower than the preconditioning in face (except for the ICC preconditioner in Table 5.11, where the preconditioning in face is slightly faster). The variants of the preconditioners in face applying the ICC preconditioner exhibited a speedup between 1.96 and 4.66 for the preconditioning in face, and between 4.28 and 6.43 for the approximate variant on the journal bearing problem. However, they were much slower on the elasticity benchmark, where they attained a speedup of at most 0.15 and 0.86 for the preconditioning in face and its approximate variant, respectively.

Despite the preconditioning working, i.e., the number of CG steps is reduced, the growth in the number of expansion steps limits the usefulness of the preconditioning. Fortunately, we developed algorithms in Section 5.2 that are specifically designed to reduce the number of expansions. Therefore, we applied the preconditioning to the MPPCG method described in Section 5.2.1. The results show that the MPPCG method significantly limits the number of expansion steps, while the number of CG and proportioning steps is in the same ballpark, if not nearly identical, compared to the MPRGP method. Even the unpreconditioned MPPCG method exhibits some speedup over the unpreconditioned MPRGP. The preconditioning in face always performs better than the approximate variant in terms of the number of Hessian multiplications, but worse in terms of the time to solution. The approximate preconditioning in face exhibits small speedups even for the SSOR preconditioner, ranging from 1.14 to 1.94. Equipping the approximate preconditioner with ICC leads to very large speedups between 2.70 and 10.38. If the unpreconditioned MPRGP is taken as the base, then the speedups range between 5.13 and 13.46.

Method	Type	Precond.	Hess.	CG	Exp.	Prop.	Time [s]	S_b	S_M
MPRGP	None	None	2030	1262	381	5	1.81	1.00	1.00
MPRGP	Face	Cholesky	788	5	389	4	227.16	0.01	0.01
MPRGP	Approx	Cholesky	817	28	392	4	9.51	0.19	0.19
MPRGP	Face	ICC	1154	95	526	6	20.65	0.09	0.09
MPRGP	Approx	ICC	1357	99	626	5	2.17	0.84	0.84
MPRGP	Face	SSOR	1617	178	717	4	15.94	0.11	0.11
MPRGP	Approx	SSOR	1642	191	723	4	2.31	0.78	0.78
MPPCG	None	None	1054	864	92	5	0.95	1.00	1.90
MPPCG	Face	Cholesky	12	5	1	4	6.24	0.15	0.29
MPPCG	Approx	Cholesky	35	28	1	4	1.22	0.78	1.48
MPPCG	Face	ICC	117	83	14	5	3.26	0.29	0.56
MPPCG	Approx	ICC	163	127	15	5	0.35	2.70	5.13
MPPCG	Face	SSOR	257	192	30	4	3.73	0.26	0.49
MPPCG	Approx	SSOR	285	202	39	4	0.49	1.94	3.68

Table 5.6: Results for preconditioning the 3D linear elasticity cube contact problem with 10x20x40 finite elements (28,413 DOFs).

Method	Type	Precond.	Hess.	CG	Exp.	Prop.	Time [s]	S_b	S_M
MPRGP	None	None	6544	3590	1472	9	88.51	1.00	1.00
MPRGP	Face	Cholesky	1818	6	903	5	14883.00	0.01	0.01
MPRGP	Approx	Cholesky	3095	44	1522	6	439.77	0.20	0.20
MPRGP	Face	ICC	4258	209	2020	8	594.58	0.15	0.15
MPRGP	Approx	ICC	5446	350	2544	7	102.93	0.86	0.86
MPRGP	Face	SSOR	5964	371	2793	6	487.90	0.18	0.18
MPRGP	Approx	SSOR	6040	405	2814	6	152.52	0.58	0.58
MPPCG	None	None	2766	2269	244	8	37.93	1.00	2.33
MPPCG	Face	Cholesky	14	6	1	5	212.37	0.18	0.42
MPPCG	Approx	Cholesky	57	43	3	7	30.19	1.26	2.93
MPPCG	Face	ICC	344	212	60	11	72.38	0.52	1.22
MPPCG	Approx	ICC	473	297	84	7	10.38	3.65	8.53
MPPCG	Face	SSOR	696	439	125	6	82.46	0.46	1.07
MPPCG	Approx	SSOR	715	443	132	7	22.55	1.68	3.93

Table 5.7: Results for preconditioning the 3D linear elasticity cube contact problem with 20x40x80 finite elements (209,223 DOFs).

Method	Type	Precond.	Hess.	CG	Exp.	Prop.	Time [s]	S_b	S_M
MPRGP	None	None	2884	2660	69	85	0.33	1.00	1.00
MPRGP	Face	Cholesky	157	78	0	78	1.95	0.17	0.17
MPRGP	Approx	Cholesky	494	79	165	84	1.63	0.20	0.20
MPRGP	Face	ICC	179	100	0	78	0.17	1.96	1.96
MPRGP	Approx	ICC	308	79	72	84	0.05	6.43	6.43
MPRGP	Face	SSOR	999	732	93	80	0.77	0.43	0.43
MPRGP	Approx	SSOR	994	666	122	83	0.21	1.59	1.59
MPPCG	None	None	2348	2218	25	79	0.27	1.00	1.24
MPPCG	Face	Cholesky	157	78	0	78	1.95	0.14	0.17
MPPCG	Approx	Cholesky	197	97	10	79	0.91	0.29	0.36
MPPCG	Face	ICC	179	100	0	78	0.17	1.59	1.96
MPPCG	Approx	ICC	208	87	19	82	0.04	7.28	9.01
MPPCG	Face	SSOR	748	623	22	80	0.60	0.44	0.55
MPPCG	Approx	SSOR	858	699	38	82	0.19	1.42	1.76

Table 5.8: Results for preconditioning the journal bearing problem with 400x25 discretization points (10,000 DOFs).

Method	Type	Precond.	Hess.	CG	Exp.	Prop.	Time [s]	S_b	S_M
MPRGP	None	None	7789	6989	306	187	3.30	1.00	1.00
MPRGP	Face	Cholesky	309	154	0	154	35.22	0.09	0.09
MPRGP	Approx	Cholesky	856	150	274	157	10.50	0.31	0.31
MPRGP	Face	ICC	366	189	11	154	1.29	2.56	2.56
MPRGP	Approx	ICC	1092	128	379	205	0.65	5.11	5.11
MPRGP	Face	SSOR	3168	1633	667	200	8.29	0.40	0.40
MPRGP	Approx	SSOR	4928	2344	1173	237	3.71	0.89	0.89
MPPCG	None	None	7286	6578	260	187	3.03	1.00	1.09
MPPCG	Face	Cholesky	309	154	0	154	35.16	0.09	0.09
MPPCG	Approx	Cholesky	421	196	33	158	7.09	0.43	0.47
MPPCG	Face	ICC	352	191	3	154	1.26	2.40	2.61
MPPCG	Approx	ICC	454	154	64	171	0.29	10.38	11.30
MPPCG	Face	SSOR	2066	1538	159	209	6.14	0.49	0.54
MPPCG	Approx	SSOR	2823	1970	308	236	2.25	1.35	1.47

Table 5.9: Results for preconditioning the journal bearing problem with 800x50 discretization points (40,000 DOFs).

Method	Type	Precond.	Hess.	CG	Exp.	Prop.	Time [s]	S_b	S_M
MPRGP	None	None	12022	9389	1199	234	9.95	1.00	1.00
MPRGP	Face	Cholesky	309	154	0	154	75.85	0.13	0.13
MPRGP	Approx	Cholesky	1839	148	765	160	39.96	0.25	0.25
MPRGP	Face	ICC	507	222	64	156	3.32	3.00	3.00
MPRGP	Approx	ICC	1920	140	772	235	2.16	4.60	4.60
MPRGP	Face	SSOR	4634	1971	1226	210	22.85	0.44	0.44
MPRGP	Approx	SSOR	7330	2667	2185	292	10.45	0.95	0.95
MPPCG	None	None	8906	7809	440	216	7.39	1.00	1.35
MPPCG	Face	Cholesky	309	154	0	154	75.86	0.10	0.13
MPPCG	Approx	Cholesky	459	208	46	158	15.35	0.48	0.65
MPPCG	Face	ICC	457	217	38	163	3.10	2.38	3.21
MPPCG	Approx	ICC	1042	169	274	324	1.17	6.29	8.47
MPPCG	Face	SSOR	2853	1913	357	225	16.24	0.46	0.61
MPPCG	Approx	SSOR	2996	1961	409	216	4.64	1.59	2.14

Table 5.10: Results for preconditioning the journal bearing problem with 800x100 discretization points (80,000 DOFs).

Method	Type	Precond.	Hess.	CG	Exp.	Prop.	Time [s]	S_b	S_M
MPRGP	None	None	37044	28703	3844	652	60.14	1.00	1.00
MPRGP	Face	Cholesky	617	308	0	308	317.32	0.19	0.19
MPRGP	Approx	Cholesky	3612	244	1525	317	156.26	0.38	0.38
MPRGP	Face	ICC	987	357	159	311	12.91	4.66	4.66
MPRGP	Approx	ICC	6225	250	2738	498	14.06	4.28	4.28
MPRGP	Face	SSOR	14072	5986	3780	525	144.25	0.42	0.42
MPRGP	Approx	SSOR	25871	8442	8281	866	73.02	0.82	0.82
MPPCG	None	None	25166	21632	1509	515	40.40	1.00	1.49
MPPCG	Face	Cholesky	617	308	0	308	317.43	0.13	0.19
MPPCG	Approx	Cholesky	887	379	93	321	59.38	0.68	1.01
MPPCG	Face	ICC	776	368	42	323	11.15	3.62	5.40
MPPCG	Approx	ICC	1976	238	564	609	4.47	9.04	13.46
MPPCG	Face	SSOR	9609	6194	1346	722	113.18	0.36	0.53
MPPCG	Approx	SSOR	11661	6902	1982	794	35.32	1.14	1.70

Table 5.11: Results for preconditioning the journal bearing problem with 1600x100 discretization points (160,000 DOFs).

Chapter 6

Quadratic Programming with Linear Equality Constraints

This chapter presents several ways to solve QP problems with the feasible set given by a linear equality

$$\Omega = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{Bx} = \boldsymbol{c}\}\,.$$

First, we present methods based on the KKT conditions that are capable of solving QP problems with linear equality constraints exactly. Then we turn to iterative solvers that allow some violations of the constraints. Among the iterative methods, we review the penalty method and then improve upon its idea in augmented Lagrangian-type methods. The augmented Lagrangian solver presented here will be used in the following section, where we solve QP problems with linear inequality constraints.

6.1 Solution Methods Based on KKT Conditions

Assume, for simplicity, that the matrix $\boldsymbol{B}$ has full row rank, but not necessarily full column rank[1]. Writing the Lagrangian of the linear equality-constrained problem

$$L(\boldsymbol{x}, \boldsymbol{\lambda}) = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{Ax} - \boldsymbol{x}^T\boldsymbol{b} + (\boldsymbol{Bx} - \boldsymbol{c})^T\boldsymbol{\lambda}, \tag{6.1}$$

the KKT conditions of Section 2.4.1 are

$$\begin{aligned} \nabla_{\boldsymbol{x}} L(\boldsymbol{x}, \boldsymbol{\lambda}) &= \boldsymbol{Ax} - \boldsymbol{b} + \boldsymbol{B}^T\boldsymbol{\lambda} = \boldsymbol{o} \\ \nabla_{\boldsymbol{\lambda}} L(\boldsymbol{x}, \boldsymbol{\lambda}) &= \boldsymbol{Bx} - \boldsymbol{c} = \boldsymbol{o}. \end{aligned} \tag{6.2}$$

These can be written compactly as the KKT saddle point system

$$\begin{pmatrix} \boldsymbol{A} & \boldsymbol{B}^T \\ \boldsymbol{B} & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \boldsymbol{b} \\ \boldsymbol{c} \end{pmatrix}$$

[1]See [94] for when this is required.

that can be solved iteratively by the MINRES Krylov subspace method or, if the system is small, by a factorization method [94]. Alternatively, by eliminating the primal variable from the KKT system, assuming $\boldsymbol{A}$ is positive definite, we obtain

$$\boldsymbol{B}\boldsymbol{A}^{-1}\boldsymbol{B}^T\boldsymbol{\lambda} = \boldsymbol{B}\boldsymbol{A}^{-1}\boldsymbol{b} - \boldsymbol{c},$$

which can be solved by the CG method or other methods[2] of Chapter 4. This approach is known as the *range space* method [10] and is used in the following Chapter 7 in the derivation of the FETI method.

If we have available a particular solution $\widehat{\boldsymbol{x}}$,

$$\boldsymbol{B}\widehat{\boldsymbol{x}} = \boldsymbol{c},$$

and a matrix $\boldsymbol{Z}$ with columns spanning the null space of $\boldsymbol{B}$, then every feasible solution can be described as

$$\boldsymbol{x} = \boldsymbol{Z}\boldsymbol{y} + \widehat{\boldsymbol{x}}. \tag{6.3}$$

We substitute the above equation into the first KKT condition (6.2), obtaining

$$\boldsymbol{A}\boldsymbol{Z}\boldsymbol{y} + \boldsymbol{A}\widehat{\boldsymbol{x}} - \boldsymbol{b} + \boldsymbol{B}^T\boldsymbol{\lambda} = \boldsymbol{o}.$$

Pre-multiplying by $\boldsymbol{Z}^T$, noting $\boldsymbol{Z}^T\boldsymbol{B}^T = \boldsymbol{O}$, and reordering, we obtain

$$\boldsymbol{Z}^T\boldsymbol{A}\boldsymbol{Z}\boldsymbol{y} = \boldsymbol{Z}^T\left(\boldsymbol{b} - \boldsymbol{A}\widehat{\boldsymbol{x}}\right).$$

Solving for $\boldsymbol{y}$ (again by the methods of Chapter 4) and substituting back into Equation (6.3), we obtain the primal solution. Notice that we only had to solve a system reduced to the null space of $\boldsymbol{B}$, which gives the approach its name, the *null space* method [94]. This approach can be adapted by employing the projectors onto the null space of $\boldsymbol{B}$ instead of using the null space basis $\boldsymbol{Z}$ directly, as shown in Section 7.1.2.

6.2 Penalty Method

A simple way to enforce the equality constraints is to enhance the cost function in such a way that the violation of the constraints is penalized. In the *quadratic penalty* method, we minimize the unconstrained penalized cost function

$$f_\rho(\boldsymbol{x}) = f(\boldsymbol{x}) + \frac{\rho}{2}||\boldsymbol{B}\boldsymbol{x} - \boldsymbol{c}||^2,$$

where $\rho \geq 0$ is the *penalty parameter* and $||\boldsymbol{B}\boldsymbol{x} - \boldsymbol{c}||^2$ is the *penalty function* [10]. If the penalty parameter ρ were infinite, then the minimizer of the penalized cost function would be the exact solution to the equality-constrained problem. In practice, a large enough parameter is chosen so that the *violation of the linear equality constraints*

$$||\boldsymbol{B}\boldsymbol{x} - \boldsymbol{c}||$$

[2] An iterative method is typically preferable to avoid the need to assemble the Hessian $\boldsymbol{B}\boldsymbol{A}^{-1}\boldsymbol{B}^T$.

is small. If the violation of the constraints is not sufficiently small, then a larger value of the penalty needs to be chosen, and the optimization must be repeated. However, this leads to a balancing act, as large penalty values typically slow down the convergence of the unconstrained solver.

6.3 Augmented Lagrangian Methods

Instead of estimating and possibly needing to update the penalty parameter, we can explicitly include the Lagrange multipliers in our unconstrained minimization. We denote the *augmented Lagrangian* function

$$L\left(\boldsymbol{x},\boldsymbol{\lambda},\rho\right)=L\left(\boldsymbol{x},\boldsymbol{\lambda}\right)+\frac{\rho}{2}||\boldsymbol{Bx}-\boldsymbol{c}||^2=f(x)+\boldsymbol{\lambda}^T\left(\boldsymbol{Bx}-\boldsymbol{c}\right)+\frac{\rho}{2}||\boldsymbol{Bx}-\boldsymbol{c}||^2,$$

where we augmented the Lagrangian (6.1) with the quadratic penalty from the previous section.

The iterative process of minimizing the augmented Lagrangian and updating the estimate of the Lagrange multiplier

$$\boldsymbol{\lambda}_{k+1}=\boldsymbol{\lambda}_k+\rho\left(\boldsymbol{Bx}_k-\boldsymbol{c}\right)$$

is known as the *method of multipliers* [95] or a variant of the *exact augmented Lagrangian* algorithm [10]. It can be shown that the Lagrange multipliers converge to the dual solution [10], and thus $\boldsymbol{x}^k$ converges to the primal solution (as we have a convex problem under linear constraints qualification; see Section 2.4.1).

We can solve the unconstrained minimization of the augmented Lagrangian inexactly by some iterative algorithm of Chapter 4. Therefore, we have an outer loop improving the approximation of the Lagrange multipliers and an inner solver for the unconstrained problem. Moreover, there is no need to solve the inner problem to high precision at the beginning of the iterative process when the Lagrange multipliers are far from the solution. Therefore, we link the precision of the inner solver to the violation of the linear equality constraints. Increasing the value of the penalty ρ will speed up the outer loop but, similarly to the penalty method, it can slow down the inner solver. Therefore, we update the penalty only if sufficient progress has not been made by the method. It turns out that sufficient progress can be linked to a sufficient increase in the value of the augmented Lagrangian. If the inequality

$$L(\boldsymbol{x}_{k+1},\boldsymbol{\lambda}_{k+1},\rho_k)<L(\boldsymbol{x}_k,\boldsymbol{\lambda}_k,\rho_{k-1})+\frac{\rho_k}{2}||\boldsymbol{Bx}_{k+1}-\boldsymbol{c}||^2$$

holds, we say that the progress is not sufficient [10]. This variant is known as the semimonotonic augmented Lagrangian (SMALE) method, or more specifically as SMALE-ρ, which is summarized in Algorithm 6.1.

Instead of increasing the penalty value ρ to achieve a sufficient increase in the augmented Lagrangian, we could solve the inner problem to higher precision by decreasing the value

Algorithm 6.1: SMALE method

```
  Initialize: $\boldsymbol{x}_0$, $\boldsymbol{\lambda}_0$, $M_0 > 0$, $\rho_0 > 0$, $\beta > 1$, $\eta > 0$, $k = 0$
1 while $||\boldsymbol{g}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k, \rho_k)|| > \epsilon||\boldsymbol{b}|| \vee ||\boldsymbol{B}\boldsymbol{x}_k - \boldsymbol{c}|| > \epsilon||\boldsymbol{b}||$:
2     $\boldsymbol{\lambda}_{k+1} = \boldsymbol{\lambda}_k + \rho_k (\boldsymbol{B}\boldsymbol{x}_k - \boldsymbol{c})$
3     find $\boldsymbol{x}_{k+1}$ such that $||\boldsymbol{g}(\boldsymbol{x}_{k+1}, \boldsymbol{\lambda}_{k+1}, \rho_k)|| \leq \min(M_k||\boldsymbol{B}\boldsymbol{x}_{k+1} - \boldsymbol{c}||, \eta)$
4     if $L(\boldsymbol{x}_{k+1}, \boldsymbol{\lambda}_{k+1}, \rho_k) < L(\boldsymbol{x}_k, \boldsymbol{\lambda}_k, \rho_{k-1}) + \frac{\rho_k}{2}||\boldsymbol{B}\boldsymbol{x}_{k+1} - \boldsymbol{c}||^2$:
5         $\rho_{k+1} = \beta\rho_k$
6     else:
7         $\rho_{k+1} = \rho_k$
8         $M_{k+1} = M_k$
9     $k = k + 1$
```

of M_k, i.e., replacing line 5 in Algorithm 6.1 with

$$M_{k+1} = \frac{M_k}{\beta}.$$

This variant is known as SMALE-M and is used for the solution of QP problems with linear equality constraints in this thesis.

The SMALE method is quite flexible in the sense that it takes care of the equality constraints but does not affect other constraints, if present. For example, having the feasible set include constraints such as the bound constraints, i.e.,

$$\Omega = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{l} \leq \boldsymbol{x} \wedge \boldsymbol{B}\boldsymbol{x} = \boldsymbol{c}\},$$

we can swap the inner solver for one of the solvers of Chapter 5 that is capable of solving bound-constrained problems, and we update the stopping criteria to those appropriate for the new inner solver, i.e., replace gradients on lines 1 and 3 of Algorithm 6.1 with the projected gradient. In this case, SMALE using MPRGP as the inner solver is also known as SMALBE [10]. The SMALBE method is used in the next chapter to solve problems with linear inequality constraints.

Chapter 7

Quadratic Programming with Linear Inequality Constraints

QP problems with linear inequality constraints

$$\Omega = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{Bx} \leq \boldsymbol{c}\}$$

can be quite challenging to solve. In our research, we prefer to solve these problems using dualization. The drawback of dualizing linear inequality-constrained QP problems is that a solve with the Hessian matrix features in the formulation. If this dual formulation is plugged into an iterative solver, then there can be a large number of solves with the Hessian matrix. In order to accelerate these solves, we prefer to use FETI-type (finite element tearing and interconnecting) methods [96]. While originally the FETI method featured dualization only as a means to the end of solving large-scale problems arising from the finite element method [7], we prefer to consider FETI the other way around, i.e., as a method to minimize the cost of the solves with the Hessian matrix in the dual formulation. Furthermore, FETI methods can be used to naturally treat discontinuities, as described for a slope stability problem in [97].

We will derive FETI in the following section and briefly note some of our improvements to the method. The second section of this chapter presents several results using FETI or standalone dualization.

The contributions of the author in this chapter are in improving the scalability and energy efficiency of the FETI methods, and in applying QP methods to solve contact problems using FETI or a standalone dualization. Most of the contributions are in software implementation, QP solution strategies, and numerical experiments. The results are supported by [4, 64, 65, 82, 96, 98–101] articles and [97, 102–107] conference proceedings.

7.1 Finite Element Tearing and Interconnecting

In this section, we will follow the descriptions of the Total FETI (TFETI) method as seen in [8, 96, 108, 109]. For ease of cross-referencing, we temporarily adopt the notation used in these references. The solution vector $\boldsymbol{x}$ is $\boldsymbol{u}$, the right-hand side $\boldsymbol{b}$ is $\boldsymbol{f}$, the Hessian $\boldsymbol{A}$ is denoted $\boldsymbol{K}$, and Ω is the problem domain. Since the problems in the descriptions of FETI originate from mechanics, e.g., linear elasticity problems, the Hessian $\boldsymbol{K}$ is often referred to as the stiffness matrix, the right-hand side $\boldsymbol{f}$ is the load vector, and the solution vector $\boldsymbol{u}$ is the displacement vector.

Broadly speaking, the FETI method decomposes (tears) the problem into a number of subproblems assembled on non-overlapping subdomains (the unknowns are duplicated at the interfaces). The subproblems can essentially be solved independently, but the continuity of the solution across the subdomains (interconnecting) has to be enforced. Lagrange multipliers are introduced to enforce the continuity of the solution. Dirichlet boundary conditions can also be enforced by Lagrange multipliers, which make all of the subdomains floating, i.e., the stiffness matrix of each subproblem has a non-empty null space. This variant of the FETI method is known as Total FETI [8]. See Figure 7.1 for an illustration of the TFETI domain decomposition.

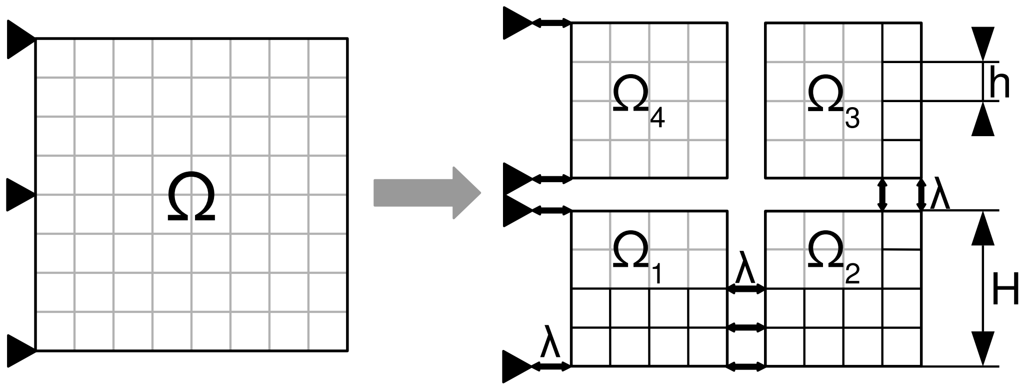

Figure 7.1: TFETI domain decomposition with outlined discretization.

Given a domain Ω decomposed into N_s non-overlapping subdomains Ω_s, $s \in \{1, \dots, N_s\}$, let each subdomain have a stiffness matrix $\boldsymbol{K}_s$, a prescribed nodal load vector $\boldsymbol{f}_s$, and a displacement vector $\boldsymbol{u}_s$. Furthermore, let the matrix $\boldsymbol{B}_E$ have entries -1, 0, and 1, describing the subdomain interconnectivity. Specifically, for an unknown that was duplicated on the interface, a row of $\boldsymbol{B}_E$ contains -1 in the column index corresponding to the unknown in one subdomain and $+1$ in the column index corresponding to the unknown in the other subdomain. In the case of using $\boldsymbol{B}_E$ to enforce a Dirichlet boundary condition, an additional row containing a single 1 is added. Then the problem of finding the reaction (displacement) vector $\boldsymbol{u}$ caused by the force $\boldsymbol{f}$ exerted on the domain Ω is

$$\arg\min_{\boldsymbol{u}} \frac{1}{2}\boldsymbol{u}^T\boldsymbol{K}\boldsymbol{u} - \boldsymbol{f}^T\boldsymbol{u} \qquad \text{s.t.} \qquad \begin{matrix} \boldsymbol{B}_I\boldsymbol{u} \le \boldsymbol{c}_I \\ \boldsymbol{B}_E\boldsymbol{u} = \boldsymbol{c}_E \end{matrix} \tag{7.1}$$

where

$$\boldsymbol{K} = \begin{pmatrix} \boldsymbol{K}_1 & & \\ & \ddots & \\ & & \boldsymbol{K}_{N_s} \end{pmatrix}, \quad \boldsymbol{f} = \begin{pmatrix} \boldsymbol{f}_1 \\ \vdots \\ \boldsymbol{f}_{N_s} \end{pmatrix}, \quad \boldsymbol{u} = \begin{pmatrix} \boldsymbol{u}_1 \\ \vdots \\ \boldsymbol{u}_{N_s} \end{pmatrix},$$

where the inequality constraints represent the non-penetration conditions for contact.

7.1.1 Dualization

Let us introduce the Lagrangian function associated with the problem (7.1)

$$L(\boldsymbol{u},\boldsymbol{\lambda}) = \frac{1}{2}\boldsymbol{u}^T\boldsymbol{K}\boldsymbol{u} - \boldsymbol{u}^T\boldsymbol{f} + (\boldsymbol{B}\boldsymbol{u}-\boldsymbol{c})^T\boldsymbol{\lambda},$$

where

$$\boldsymbol{B} = \left[\boldsymbol{B}_E^T, \boldsymbol{B}_I^T\right]^T, \qquad \boldsymbol{c} = \left[\boldsymbol{c}_E^T, \boldsymbol{c}_I^T\right]^T, \qquad \boldsymbol{\lambda} = \left[\boldsymbol{\lambda}_E^T, \boldsymbol{\lambda}_I^T\right]^T.$$

The dimension of the matrix $\boldsymbol{B}$ is $N_d \times N_p$, where N_p is the primal dimension and N_d is the number of Lagrange multipliers – dual dimension. We will assume that $\boldsymbol{B}$ is full row rank, which can be achieved, e.g., by limiting that an unknown is present in at most one inequality [109]. The KKT conditions are by Section 2.4.1

$$\nabla_{\boldsymbol{u}} L(\boldsymbol{u},\boldsymbol{\lambda}) = \boldsymbol{K}\boldsymbol{u} - \boldsymbol{f} + \boldsymbol{B}^T\boldsymbol{\lambda} = \boldsymbol{o}, \tag{7.2}$$

$$\nabla_{\boldsymbol{\lambda}_E} L(\boldsymbol{u},\boldsymbol{\lambda}) = \boldsymbol{B}_E\boldsymbol{u} - \boldsymbol{c}_E = \boldsymbol{o}, \tag{7.3}$$

$$\nabla_{\boldsymbol{\lambda}_I} L(\boldsymbol{u},\boldsymbol{\lambda}) = \boldsymbol{B}_I\boldsymbol{u} - \boldsymbol{c}_I \leq \boldsymbol{o}, \tag{7.4}$$

$$\boldsymbol{\lambda}_I^T(\boldsymbol{B}_I\boldsymbol{u} - \boldsymbol{c}_I) = \boldsymbol{o}, \tag{7.5}$$

$$\boldsymbol{\lambda}_I \geq \boldsymbol{o}. \tag{7.6}$$

The first equation (7.2) is solvable if and only if

$$(\boldsymbol{f} - \boldsymbol{B}^T\boldsymbol{\lambda}) \in \operatorname{Im}\boldsymbol{K} \Leftrightarrow \boldsymbol{R}^T(\boldsymbol{f} - \boldsymbol{B}^T\boldsymbol{\lambda}) = \boldsymbol{o}, \tag{7.7}$$

where $\boldsymbol{R} \in \mathbb{R}^{N_p \times N_k}$ columns are the null space vectors of $\boldsymbol{K}$, and $N_k = N_p - \operatorname{rank}(\boldsymbol{K})$. Multiplying the last equation by $\boldsymbol{K}$ and writing the zero right-hand side as $\boldsymbol{K}\boldsymbol{R}\boldsymbol{\alpha} = \boldsymbol{o}$, which is true for any $\boldsymbol{\alpha} \in \mathbb{R}^{N_k}$, gives

$$\boldsymbol{K}\boldsymbol{u} - \boldsymbol{f} + \boldsymbol{B}^T\boldsymbol{\lambda} = \boldsymbol{K}\boldsymbol{R}\boldsymbol{\alpha}.$$

Multiplying the last equation by a left generalized inverse $\boldsymbol{K}^+$, i.e., $\boldsymbol{K}\boldsymbol{K}^+\boldsymbol{K} = \boldsymbol{K}$, and rearranging yields the solution $\boldsymbol{u}$

$$\boldsymbol{u} = \boldsymbol{K}^+(\boldsymbol{f} - \boldsymbol{B}^T\boldsymbol{\lambda}) + \boldsymbol{R}\boldsymbol{\alpha}, \tag{7.8}$$

with unknowns $\boldsymbol{\lambda}$ and $\boldsymbol{\alpha}$. The generalized inverse is typically realized by a direct solver, which is MUMPS in our case, using either the fixing nodes [110] or projectors onto the range of the stiffness matrix $\boldsymbol{K}$ [104]. We note that sparse graph reordering algorithms can greatly improve the factorization of the stiffness matrix in terms of factorization speed, solve time, and memory requirements, as we demonstrated in [102].

Substituting $\boldsymbol{u}$ into the second to fourth KKT conditions (7.3) to (7.5) yields

$$[\nabla_{\boldsymbol{\lambda}} L(\boldsymbol{u},\boldsymbol{\lambda})]_E = \left[-\boldsymbol{B}\boldsymbol{K}^+\boldsymbol{B}^T\boldsymbol{\lambda} + (\boldsymbol{B}\boldsymbol{K}^+\boldsymbol{f} - \boldsymbol{c}) + \boldsymbol{B}\boldsymbol{R}\boldsymbol{\alpha}\right]_E = \boldsymbol{o}, \tag{7.9}$$

$$[\nabla_{\boldsymbol{\lambda}} L(\boldsymbol{u},\boldsymbol{\lambda})]_I = \left[-\boldsymbol{B}\boldsymbol{K}^+\boldsymbol{B}^T\boldsymbol{\lambda} + (\boldsymbol{B}\boldsymbol{K}^+\boldsymbol{f} - \boldsymbol{c}) + \boldsymbol{B}\boldsymbol{R}\boldsymbol{\alpha}\right]_I \leq \boldsymbol{o}, \tag{7.10}$$

$$\boldsymbol{\lambda}_I^T\left[-\boldsymbol{B}\boldsymbol{K}^+\boldsymbol{B}^T\boldsymbol{\lambda} + (\boldsymbol{B}\boldsymbol{K}^+\boldsymbol{f} - \boldsymbol{c}) + \boldsymbol{B}\boldsymbol{R}\boldsymbol{\alpha}\right]_I = \boldsymbol{o}, \tag{7.11}$$

$$\tag{7.12}$$

and together with the fifth KKT condition $\boldsymbol{\lambda}_I \geq \boldsymbol{o}$ and with the solvability of the first KKT condition $\boldsymbol{R}^T(\boldsymbol{f} - \boldsymbol{B}^T\boldsymbol{\lambda}) = \boldsymbol{o}$ (Eqs. (7.6) and (7.7)) represent the KKT conditions for

$$\arg\max_{\boldsymbol{\lambda}} -\frac{1}{2}\boldsymbol{\lambda}^T\boldsymbol{B}\boldsymbol{K}^+\boldsymbol{B}^T\boldsymbol{\lambda}+\boldsymbol{\lambda}^T(\boldsymbol{B}\boldsymbol{K}^+\boldsymbol{f}-\boldsymbol{c})-\frac{1}{2}\boldsymbol{f}^T\boldsymbol{K}^+\boldsymbol{f} \quad \text{s.t.} \quad \begin{array}{c}\boldsymbol{\lambda}_I \geq \boldsymbol{o},\\ \boldsymbol{R}^T\boldsymbol{B}^T\boldsymbol{\lambda} = \boldsymbol{R}^T\boldsymbol{f}.\end{array} \tag{7.13}$$

Denoting

$$\begin{aligned} \boldsymbol{F} &= \boldsymbol{B}\boldsymbol{K}^+\boldsymbol{B}^T, & \boldsymbol{G} &= \boldsymbol{R}^T\boldsymbol{B}^T,\\ \boldsymbol{e} &= \boldsymbol{R}^T\boldsymbol{f}, & \boldsymbol{d} &= \boldsymbol{B}\boldsymbol{K}^+\boldsymbol{f} - \boldsymbol{c}, \end{aligned}$$

omitting the constant term and changing the sign in Equation (7.13), we obtain the dual formulation

$$\arg\min_{\boldsymbol{\lambda}} \frac{1}{2}\boldsymbol{\lambda}^T\boldsymbol{F}\boldsymbol{\lambda} - \boldsymbol{\lambda}^T\boldsymbol{d} \qquad \text{s.t.} \qquad \begin{array}{c}\boldsymbol{\lambda}_I \geq 0\\ \boldsymbol{G}\boldsymbol{\lambda} = \boldsymbol{e}.\end{array} \tag{7.14}$$

Note that if the Hessian matrix of the primal problem $\boldsymbol{K}$ is nonsingular, e.g., if the problem is not decomposed into subdomains and the Dirichlet boundary conditions are incorporated into $\boldsymbol{K}$, then the dual formulation is only bound-constrained with no equality constraints.

7.1.2 Improving Dual Formulation

To improve the dual formulation, we first homogenize the equality constraints in the same way as in Equation (2.2). Splitting $\boldsymbol{\lambda}$ into $\boldsymbol{\lambda}_{\mathrm{Im}} \in \operatorname{Im}\boldsymbol{G}^T$ and $\boldsymbol{\lambda}_{\mathrm{Ker}} \in \operatorname{Ker}\boldsymbol{G}$ so that

$$\boldsymbol{\lambda} = \boldsymbol{\lambda}_{\mathrm{Im}} + \boldsymbol{\lambda}_{\mathrm{Ker}}. \tag{7.15}$$

The solution on the image of $\boldsymbol{G}^T$ is the least squares solution

$$\boldsymbol{\lambda}_{\mathrm{Im}} = \boldsymbol{G}^T(\boldsymbol{G}\boldsymbol{G}^T)^{-1}\boldsymbol{e}.$$

Since $\boldsymbol{\lambda}_{\mathrm{Im}}$ is known, rewriting the dual formulation (7.14) in the unknown $\boldsymbol{\lambda}_{\mathrm{Ker}}$ yields the homogenized dual formulation

$$\arg\min_{\boldsymbol{\lambda}_{\mathrm{Ker}}} \frac{1}{2}\boldsymbol{\lambda}_{\mathrm{Ker}}^T\boldsymbol{F}\boldsymbol{\lambda}_{\mathrm{Ker}} - \boldsymbol{\lambda}_{\mathrm{Ker}}^T(\boldsymbol{d} - \boldsymbol{F}\boldsymbol{\lambda}_{\mathrm{Im}}) \qquad \text{s.t.} \qquad \begin{array}{l}[\boldsymbol{\lambda}_{\mathrm{Ker}}]_I \geq -[\boldsymbol{\lambda}_{\mathrm{Im}}]_I\\ \boldsymbol{G}\boldsymbol{\lambda}_{\mathrm{Ker}} = \boldsymbol{o}.\end{array} \tag{7.16}$$

In the FETI methods, to improve the convergence of iterative solvers applied to the dual problem formulation, a projector onto the natural coarse space [111] is introduced. This projector accelerates convergence by propagating information throughout the problem globally.

The projector is onto the null space of $\boldsymbol{G}$ and is defined by

$$\boldsymbol{P} = \boldsymbol{P}^T = \boldsymbol{I} - \boldsymbol{G}^T(\boldsymbol{G}\boldsymbol{G}^T)^{-1}\boldsymbol{G} = \boldsymbol{I} - \boldsymbol{Q}.$$

Applying the projector symmetrically to the homogenized dual formulation (7.16) yields the final dual formulation for the FETI method

$$\arg\min_{\boldsymbol{\lambda}_{\text{Ker}}} \frac{1}{2}\boldsymbol{\lambda}_{\text{Ker}}^T \boldsymbol{PFP}\boldsymbol{\lambda}_{\text{Ker}} - \boldsymbol{\lambda}_{\text{Ker}}^T \boldsymbol{P}\left(\boldsymbol{d} - \boldsymbol{F}\boldsymbol{\lambda}_{\text{Im}}\right) \quad \text{s.t.} \quad \begin{array}{l} [\boldsymbol{\lambda}_{\text{Ker}}]_I \geq -[\boldsymbol{\lambda}_{\text{Im}}]_I \\ \boldsymbol{G}\boldsymbol{\lambda}_{\text{Ker}} = \boldsymbol{o}. \end{array}$$

We note that the projector would be sufficient on its own to enforce the equality constraints if there were no bound constraints in the dual formulation, i.e., if there were no linear inequality constraints in the primal formulation. If the linear inequality is not present, the dual formulation is solved by the conjugate gradient method[1]. Otherwise, we use the SMALE method of Section 6.3. The SMALE inner problem reads

$$\arg\min_{\boldsymbol{\lambda}_{\text{Ker}}} \frac{1}{2}\boldsymbol{\lambda}_{\text{Ker}}^T \left(\boldsymbol{PFP} + \rho\boldsymbol{G}^T\boldsymbol{G}\right)\boldsymbol{\lambda}_{\text{Ker}} - \boldsymbol{\lambda}_{\text{Ker}}^T \boldsymbol{P}\left(\boldsymbol{d} - \boldsymbol{F}\boldsymbol{\lambda}_{\text{Im}}\right) \quad \text{s.t.} \quad [\boldsymbol{\lambda}_{\text{Ker}}]_I \geq -[\boldsymbol{\lambda}_{\text{Im}}]_I\,, \tag{7.17}$$

which is solved by an MPRGP-type algorithm of Chapter 5.

If the matrix $\boldsymbol{GG}^T$ is decomposed using Cholesky factorization

$$\boldsymbol{GG}^T = \boldsymbol{LL}^T,$$

then scaling the equality constraints by $\boldsymbol{L}^{-1}$, we have

$$\boldsymbol{L}^{-1}\boldsymbol{G}\boldsymbol{\lambda}_{\text{Ker}} = \boldsymbol{o},$$

and the term added by the quadratic penalty is

$$\begin{aligned} \frac{\rho}{2}\|\boldsymbol{L}^{-1}\boldsymbol{G}\boldsymbol{\lambda}_{\text{Ker}}\|^2 &= \frac{\rho}{2}\boldsymbol{\lambda}_{\text{Ker}}^T\boldsymbol{G}^T\left(\boldsymbol{L}^{-1}\right)^T\boldsymbol{L}^{-1}\boldsymbol{G}\boldsymbol{\lambda}_{\text{Ker}} \\ &= \frac{\rho}{2}\boldsymbol{\lambda}_{\text{Ker}}^T\boldsymbol{G}^T\left(\boldsymbol{GG}^T\right)^{-1}\boldsymbol{G}\boldsymbol{\lambda}_{\text{Ker}} = \frac{\rho}{2}\boldsymbol{\lambda}_{\text{Ker}}^T\boldsymbol{Q}\boldsymbol{\lambda}_{\text{Ker}}. \end{aligned}$$

Then our final formulation of the SMALE inner problem can be written as

$$\arg\min_{\boldsymbol{\lambda}_{\text{Ker}}} \frac{1}{2}\boldsymbol{\lambda}_{\text{Ker}}^T \left(\boldsymbol{PFP} + \rho\boldsymbol{Q}\right)\boldsymbol{\lambda}_{\text{Ker}} - \boldsymbol{\lambda}_{\text{Ker}}^T \boldsymbol{P}\left(\boldsymbol{d} - \boldsymbol{F}\boldsymbol{\lambda}_{\text{Im}}\right) \quad \text{s.t.} \quad [\boldsymbol{\lambda}_{\text{Ker}}]_I \geq -[\boldsymbol{\lambda}_{\text{Im}}]_I\,.$$

The approach is known as an implicit orthogonalization of the equality constraints [24]. The Hessian of this final formulation has a better spectrum than the Hessian of formulation (7.17) [108], unless $\boldsymbol{G}$ has orthogonal rows, in which case the formulations are equivalent. Indeed, the final Hessian eigenvalues are those of $\boldsymbol{PFP}$, except for the zero eigenvalues, which are shifted to ρ. This is due to the fact that $\boldsymbol{P}$ is the orthogonal projector onto $\operatorname{Ker}\boldsymbol{G}$, and $\boldsymbol{Q}$ is the orthogonal projector onto $\operatorname{Im}\boldsymbol{G}^T$. Formally, let $\boldsymbol{PFP}\boldsymbol{v}_{\text{Ker}} = \mu_{\text{Ker}}\boldsymbol{v}_{\text{Ker}}$, then

$$\left(\boldsymbol{PFP} + \rho\boldsymbol{Q}\right)\boldsymbol{v}_{\text{Ker}} = \mu_{\text{Ker}}\boldsymbol{v}_{\text{Ker}}$$

and note that $\boldsymbol{F}$ is positive definite on $\operatorname{Ker}\boldsymbol{G}$ [109], so that zero eigenvalues come from the null space of the projector, which is $\operatorname{Im}\boldsymbol{G}^T$. Moreover, for any $\boldsymbol{v}_{\text{Im}} \in \operatorname{Im}\boldsymbol{G}^T$

$$\left(\boldsymbol{PFP} + \rho\boldsymbol{Q}\right)\boldsymbol{v}_{\text{Im}} = \rho\boldsymbol{v}_{\text{Im}}.$$

[1]Only one application of the project $\boldsymbol{P}$ is required in this case.

Therefore, there are $\dim(\operatorname{Ker}\boldsymbol{G})$ and $\dim\left(\operatorname{Im}\boldsymbol{G}^T\right)$ nonzero eigenvalues, which are equal to the nonzero eigenvalues of $\boldsymbol{PFP}$ and ρ, respectively.

After obtaining $\boldsymbol{\lambda}$, we find $\boldsymbol{\alpha}$ from the KKT conditions (7.9) to (7.11). By the KKT condition (7.11), the KKT condition (7.10) is zero only if $\boldsymbol{\lambda}_I > 0$. Denoting the active set

$$\mathcal{A} = E \cup \{i \mid \lambda_i > 0\},$$

we have from the KKT conditions (7.9) and (7.10)

$$\left[-\boldsymbol{BK}^+\boldsymbol{B}^T\boldsymbol{\lambda} + (\boldsymbol{BK}^+\boldsymbol{f} - \boldsymbol{c}) + \boldsymbol{BR\alpha}\right]_{\mathcal{A}} = \boldsymbol{o}.$$

Substituting $\widehat{\boldsymbol{F}} = \boldsymbol{B}_{\mathcal{A}}\boldsymbol{K}^+(\boldsymbol{B}_{\mathcal{A}})^T$, $\widehat{\boldsymbol{d}} = \boldsymbol{B}_{\mathcal{A}}\boldsymbol{K}^+\boldsymbol{f} - \boldsymbol{c}_{\mathcal{A}}$, and $\widehat{\boldsymbol{G}} = \boldsymbol{R}^T(\boldsymbol{B}_{\mathcal{A}})^T$ into the above equation, we have

$$\begin{aligned}
-\widehat{\boldsymbol{F}}\boldsymbol{\lambda}_{\mathcal{A}} + \widehat{\boldsymbol{d}} + \widehat{\boldsymbol{G}}^T\boldsymbol{\alpha} &= \boldsymbol{o},\\
\widehat{\boldsymbol{G}}^T\boldsymbol{\alpha} &= \widehat{\boldsymbol{F}}\boldsymbol{\lambda}_{\mathcal{A}} - \widehat{\boldsymbol{d}} \qquad / \left(\widehat{\boldsymbol{G}}\widehat{\boldsymbol{G}}^T\right)^{-1}\widehat{\boldsymbol{G}}\cdot\\
\boldsymbol{\alpha} &= \left(\widehat{\boldsymbol{G}}\widehat{\boldsymbol{G}}^T\right)^{-1}\widehat{\boldsymbol{G}}\left(\widehat{\boldsymbol{F}}\boldsymbol{\lambda}_{\mathcal{A}} - \widehat{\boldsymbol{d}}\right).
\end{aligned}$$

With both $\boldsymbol{\lambda}$ and α known, the primal solution $\boldsymbol{u}$ is obtained from Equation (7.8).

7.1.3 FETI Condition Number and Preconditioning

Suppose that the domain Ω of a Poisson or elasticity problem (without the linear inequality constraints) is a regularly discretized square or cube with the subdomain size H and discretization parameter h (as in Figure 7.1). Then we have the following estimate of the (T)FETI method condition number [8, 111]:

$$\kappa\left(\boldsymbol{PF}|\operatorname{Im}\boldsymbol{P}\right) \le C\frac{H}{h}. \tag{7.18}$$

The FETI preconditioners, introduced in [111], can further improve convergence.

Splitting the indices of DOFs into internal i and boundary b (all DOFs that have associated Lagrange multipliers) index sets, the stiffness matrix $\boldsymbol{K}$ can then be divided into blocks

$$\boldsymbol{K}^s = \begin{bmatrix}\boldsymbol{K}^s_{ii} & \boldsymbol{K}^s_{ib}\\ \boldsymbol{K}^s_{bi} & \boldsymbol{K}^s_{bb}\end{bmatrix}.$$

Decomposing the matrix $\boldsymbol{B}$ into subdomain constraint matrices $\boldsymbol{B}^s$ so that $\boldsymbol{B} = \left[\boldsymbol{B}^1 \cdots \boldsymbol{B}^{N_s}\right]$, the subdomain constraint matrix $\boldsymbol{B}^s$ can be rewritten as

$$\boldsymbol{B}^s = \begin{bmatrix}\boldsymbol{O} & \boldsymbol{B}^s_b\end{bmatrix}$$

where $\boldsymbol{B}^s_b$ is formed by the columns corresponding to the interface DOFs, while the remaining columns are all zero and correspond to the internal DOFs, as the constraint matrix acts only on the interface.

The theoretically optimal Dirichlet preconditioner reads

$$\boldsymbol{F}_D^{-1} = \sum_{s=1}^{N_s} \boldsymbol{B}^s \begin{bmatrix} \boldsymbol{O} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{S}_{bb}^s \end{bmatrix} (\boldsymbol{B}^s)^T = \sum_{s=1}^{N_s} \boldsymbol{B}_b^s \boldsymbol{S}_{bb}^s (\boldsymbol{B}_b^s)^T = \boldsymbol{B}\boldsymbol{S}\boldsymbol{B}^T,$$

where $\boldsymbol{S}_{bb}^s$ is the Schur complement eliminating the block $\boldsymbol{K}_{ii}^s$ related to internal DOFs with respect to the subdomain stiffness matrix $\boldsymbol{K}^s$,

$$\boldsymbol{S}_{bb}^s = \boldsymbol{K}_{bb}^s - (\boldsymbol{K}_{ib}^s)^T (\boldsymbol{K}_{ii}^s)^{-1} \boldsymbol{K}_{ib}^s,$$

and

$$\boldsymbol{S} = \operatorname{diag}\left(\begin{bmatrix} \boldsymbol{O} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{S}_{bb}^1 \end{bmatrix}, \dots, \begin{bmatrix} \boldsymbol{O} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{S}_{bb}^{N_S} \end{bmatrix} \right).$$

A cheaper variant is the lumped preconditioner

$$\boldsymbol{F}_L^{-1} = \sum_{s=1}^{N_s} \boldsymbol{B}^s \begin{bmatrix} \boldsymbol{O} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{K}_{bb}^s \end{bmatrix} (\boldsymbol{B}^s)^T = \sum_{s=1}^{N_s} \boldsymbol{B}_b^s \boldsymbol{K}_{bb}^s (\boldsymbol{B}_b^s)^T = \boldsymbol{B}\boldsymbol{K}\boldsymbol{B}^T.$$

The lumped preconditioner is not numerically scalable; however, it is more economical than the Dirichlet preconditioner.

Using $\boldsymbol{F}_D^{-1}$, the original condition number estimate (7.18), with some reasonable restrictions on the constraint matrix $\boldsymbol{B}$, improves as [112]

$$\kappa(\boldsymbol{P}\boldsymbol{F}_D^{-1}\boldsymbol{P}\boldsymbol{F} | \operatorname{Im} \boldsymbol{P}) \leq C \left(1 + \log \frac{H}{h}\right)^2.$$

No such theoretical improvement has been proven with the lumped preconditioner $\boldsymbol{F}_L^{-1}$. On the other hand, the lumped preconditioner is much cheaper to compute and potentially apply than the Dirichlet preconditioner $\boldsymbol{F}_D^{-1}$.

7.1.4 FETI: Further Improvements and Beyond Finite Elements

7.1.4.1 FETI Coarse Problem

The solution of the coarse problem $\left(\boldsymbol{G}\boldsymbol{G}^T\right)^{-1}$ involved in the projector $\boldsymbol{P}$ becomes a bottleneck when the number of subdomains is large. For problems with linear inequality constraints, or when using the preconditioners from the previous subsection, this bottleneck is magnified, as these problems require two CP solutions in each iteration of the solver.

In [104], we have shown that the coarse space projectors can be avoided when using the Moore-Penrose inverse instead of only the left generalized inverse as $\boldsymbol{K}^+$. Moreover, the article demonstrates how to easily obtain the Moore-Penrose inverse from a left generalized inverse. For problems with linear inequality constraints, one coarse problem still remains when using an augmented Lagrangian method. However, this coarse problem can be solved inexactly. The modification is known as the projector-avoiding FETI method.

Hybrid FETI methods [113] were introduced to further reduce the cost of the coarse problem. They group subdomains into a number of clusters at the primal level. Since the size of the coarse problem in TFETI is equal to the number of subdomains times the defect of any subdomain (e.g., six rigid body modes in 3D elasticity), connecting $m \times m \times m$ subdomains into clusters reduces the coarse problem size by a factor of m^3 in 3D[2]. The subdomains in clusters can be connected by edge averages [99, 103]. However, the hybrid methods exhibit slightly worse convergence.

In any case, efficient strategies to solve the coarse problem are required [64]. The deflated conjugate gradient method, together with these strategies, has been shown to be very effective in solving this problem in Section 4.3.9.

7.1.4.2 Energy Efficiency

Energy efficiency is a hot topic in supercomputing. It turns out that the energy efficiency of algorithms can be improved by dynamically changing the operating parameters of the hardware depending on the part of the application being executed, e.g., stiffness matrix factorization. The author's own work on the topic was a contribution to energy measurement software and methodology development, and experiments with the FETI method and BLAS routines used in FETI, consisting of changing the processor core frequency, as detailed in [101, 105–107].

7.1.4.3 Boundary Element Method

The FETI scheme can, in principle, be applied to any problem for which non-overlapping subproblems can be constructed, and the continuity of the global solution can be enforced by linear equality constraints. An easy extension is to use the boundary element method instead of the finite element method. It was shown in [98] that the condition number of a 2D model scalar problem is essentially better for the boundary element tearing and interconnecting (BETI) method

$$\frac{\kappa\,(\mathrm{FETI})}{\kappa\,(\mathrm{BETI})} \to \frac{32}{17} \approx 1.88 \quad \text{as} \quad h \to 0.$$

The drawback of the BETI method is that it is usually more expensive in the operator assembly phase but can be less expensive in the solve phase.

7.2 Results

7.2.1 Solution of Large Scale Contact Problems of Mechanics

The Figures 7.2 and 7.3 illustrate the performance of the projector-avoiding TFETI method on the 3D linear elasticity (Section 3.3.4) computed on Salomon (Section 3.2.4) with the relative tolerance of 10^{-6}. The SMALE-M parameters were $\eta = 0.1$, $\rho = 1.1||\boldsymbol{A}||$, $M = 100||\boldsymbol{A}||$, and $\beta = 10$. The MPRGP parameters were $\Gamma = 1$ and $\overline{\alpha} = 1.9||\boldsymbol{A}||^{-1}$. The scalability is demonstrated up to $15,625$ cores and 1.2 billion unknowns. The size of the

[2]In 2D, with clusters consisting of $m \times m$ subdomains, the reduction factor would be m^2.

subdomains is kept constant (30^3 elements for $73,167$ DOFs). We can observe some growth in the number of Hessian multiplications and the time to solution because the size of the contact interface increases. Overall, the projector-avoiding TFETI method exhibits better scalability and achieves a speedup of 1.7 on the largest problem compared to the standard TFETI method.

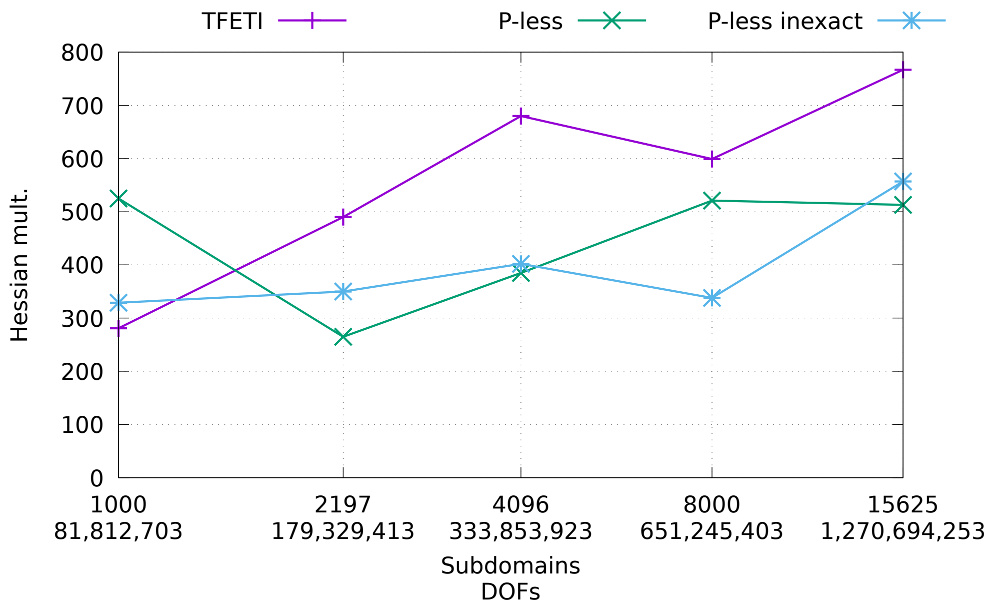

Figure 7.2: Numerical scalability comparison of the TFETI method and its projector-avoiding (P-less) variants on the 3D linear elasticity cube contact problem. The inexact method solves the one remaining coarse problem inexactly with the CG method with the relative tolerance of 10^{-2}. One subdomain is assigned to one core [104].

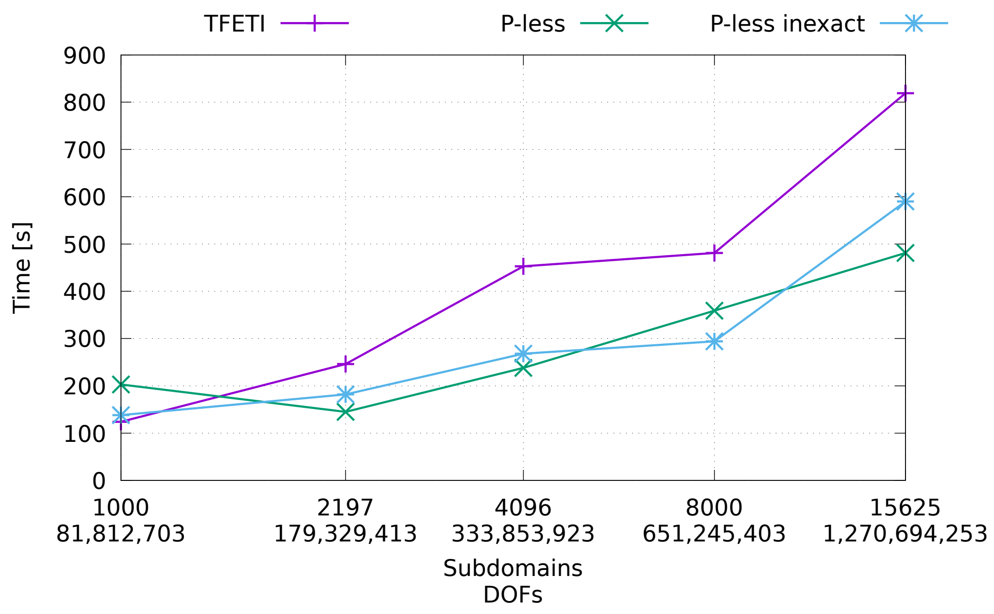

Figure 7.3: Scalability comparison of the TFETI method and its projector-avoiding (P-less) variants on the 3D linear elasticity cube contact problem. The inexact method solves the one remaining coarse problem inexactly with the CG method with the relative tolerance of 10^{-2}. One subdomain is assigned to one core [104].

7.2.2 Preconditioned FETI for Elastoplasticity

In [100], we compared the FETI preconditioners on a 3D elastoplastic $4 \times 2 \times 1$ mm cuboid, similar to the linear elasticity cuboid problem of Section 3.3.4, but without contact. The model parameters are the same as in Section 3.3.4 except for the Young's modulus $E = 200$ GPa and additional parameters $\sigma_y = 450$ MPa and $H_m = 100$ GPa, which are the initial yield stress and the hardening modulus, respectively.

The problem consists of a solution for a single time step, with the nonlinear operator linearized by a semismooth Newton method. See [100] for a complete problem and algorithm description. In each Newton iteration, a linear elasticity problem is solved with FETI to the relative tolerance of 10^{-6}. The semismooth Newton method required 5 iterations to satisfy the relative stopping criterion of 10^{-4}.

Figures 7.4 and 7.5 illustrate the performance of the FETI preconditioners on $8 \times 4 \times 2 = 64$ to $32 \times 16 \times 8 = 4096$ subdomains, each discretized by 24^3 finite elements ($46,875$ DOFs) for a total of $830,115$ to $171,421,635$ DOFs. The results were computed on MareNostrum 3, which is described in Section 3.2.3.

The choice of the rectangular cuboid domain ensures that the subdomains are quite badly conditioned. Therefore, the Dirichlet preconditioner works very well, achieving a speedup compared to the unpreconditioned variant of about 3 and 2 in terms of the Hessian multiplication and the time to solution, respectively. The time to solution may actually be worse than the unpreconditioned FETI for well-conditioned subdomains. In any case, the lumped preconditioner always achieved a small (at most 1.2) speedup in our tests. More numerical experiments with differently conditioned subdomains can be found in [100].

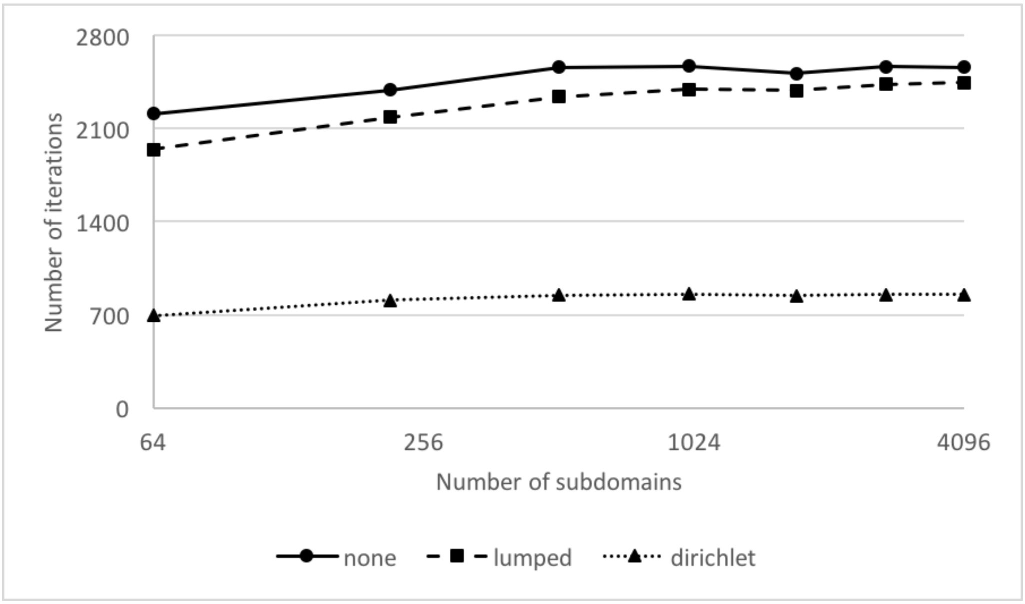

Figure 7.4: Comparison of the number of CG iterations for unpreconditioned TFETI and TFETI equipped with the lumped or Dirichlet preconditioners on the 3D elastoplastic cuboid problem [100].

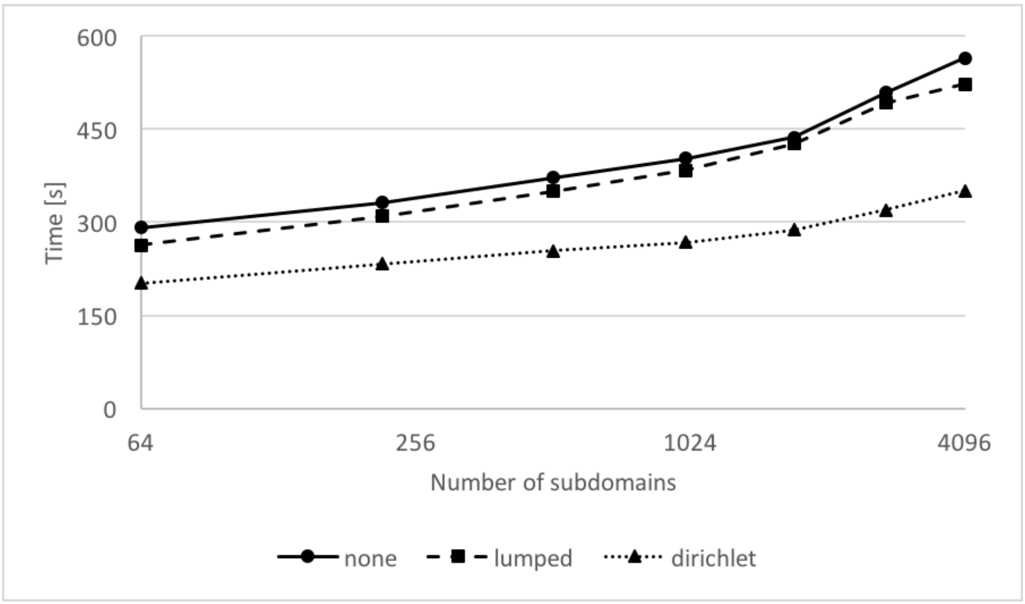

Figure 7.5: Scalability comparison of unpreconditioned TFETI and TFETI equipped with the lumped or Dirichlet preconditioners on the 3D elastoplastic cuboid problem. One subdomain is assigned to one core [100].

7.2.3 Solving Contact Problems without FETI

In [4], we presented a numerical method for the solution of hydro-mechanical problems with fracture networks and contact conditions. The main benchmark of rock relaxation during tunnel excavation in a fractured porous medium is briefly described in Section 3.3.5. A robust iterative splitting was used to alternately solve the hydrological and mechanical subproblems with the relative stopping tolerance of 10^{-4}. The hydrological subproblem was solved using the CG method with the BoomerAMG algebraic multigrid preconditioner. The mechanical subproblem with contact on fractures was solved using PERMON, employing dualization without using FETI and the MPRGP solver. The relative tolerance for the hydrological and mechanical subproblems was 10^{-8} and 10^{-6}, respectively. The MPRGP parameters were $\Gamma = 1$ and $\overline{\alpha} = 1.9||\boldsymbol{A}||^{-1}$. The simulation was run using the Flow123d software.

Performances of the solvers on a single node of the LUMI supercomputer (Section 3.2.2) employing 64 cores are summarized in Table 7.1 and Figures 7.6 and 7.7. A key ingredient for the performance of the scheme is warm starting the solvers with the last available solution. In the case of the mechanical subproblem, this means warm starting the MPRGP solver with the last dual solution. Overall, the warm starting reduced the number of iterations needed by the solvers by more than a factor of 2.

Problem	Primal	Dual	Hydr. iter.	Mech. iter.	Time [min]
306k (200)	213,945	18,632	2,108	365	6.4
502k (200)	322,953	19,591	2,587	544	10.3
1052k (200)	629,352	28,128	1,869	625	20.2
989k (400)	651,861	42,759	1,894	570	19.7

Table 7.1: Numerical scalability for each test case. The problem name consists of the number of elements in thousands and the number of fractures in brackets. We report the primal and dual dimensions of the mechanical subproblem, the number of hydraulic subproblem iterations, the number of Hessian multiplications needed by the mechanical subproblem solver, and the approximate runtime in minutes. In all cases, 71 time steps were performed [4].

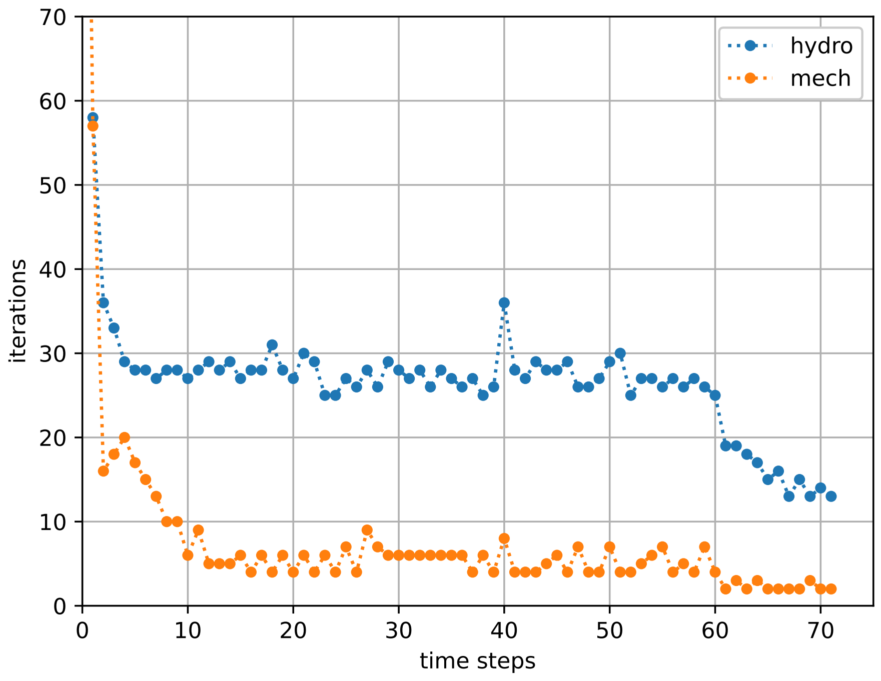

Figure 7.6: Cumulative number of iterations for both solvers at each time step. Time step iteration zero represents the steady-state initial solution, which took 148 iterations [4].

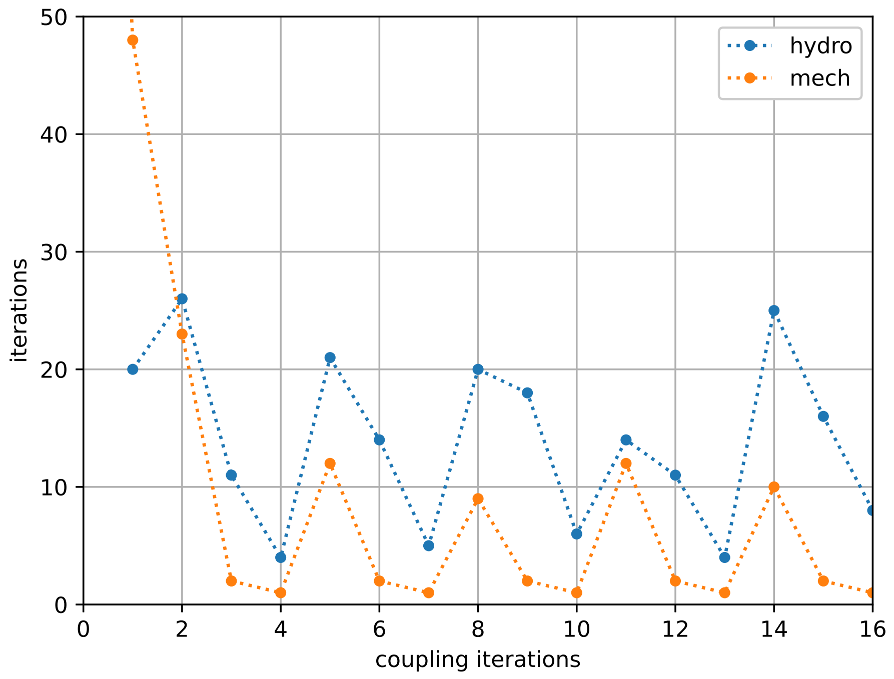

Figure 7.7: The number of iterations for each coupling iteration in the first 5 time steps (right) for the problem with one million elements and 200 fractures. Coupling iteration zero represents the steady-state initial solution, which took 148 iterations [4].

Chapter 8

Conclusion

The thesis explores a number of improvements to selected QP algorithms. The improvements are in terms of convergence, numerical scalability, time to solution, and general scalability on large supercomputers.

Chapter 2 reviews basic facts and concepts in optimization, including the Karush-Kuhn-Tucker conditions and duality.

Chapter 3 introduces the main software used throughout the thesis – the PERMON library for quadratic programming, of which the author has been the main developer and maintainer during his Ph.D. studies. Moreover, several benchmarks used in the thesis, as well as the supercomputers employed to compute these benchmarks, are described.

Chapter 4 reviews the steepest descent method and the CG method for the solution of unconstrained QP problems. Preconditioning of the CG method is discussed, with particular focus on the deflation preconditioner. It is shown that the deflated CG method is the fastest and most scalable method for the solution of the FETI coarse problem for up to a medium number of coarse problem solves with a large number of cores. The time needed for the coarse problem solution is more than halved compared to the standard direct solver solution at $27,000$ cores with a better scalability slope.

Chapter 5 reviews MPRGP and SPG methods using projection onto the feasible set. Then new modifications of MPRGP using the projected CG method (MPPCG) or the SPG method (MPSPG) instead of a fixed step length projection for the active set expansion are developed. These modifications exhibit a large geometric mean of speedups in terms of the Hessian multiplications on suitable benchmarks of 2.9 for MPPCG and 6.25 for MPSPG. Furthermore, preconditioning for MPRGP-type methods is discussed, and a new approximate preconditioning in face is developed. This new preconditioner, combined with MPPCG, achieves solution time speedups between 5.1 and 13.4 compared to the unpreconditioned version and consistently outperforms the standard preconditioning in face.

Chapter 6 describes solution methods for QP problems with linear equality constraints. These include the solutions of saddle point formulations, the penalty method, and augmented Lagrangian methods.

Dualization is used in Chapter 7 to solve QP problems with linear inequality constraints. The FETI method is reviewed as a way to accelerate the solution. A number of modifications to the FETI method (projector-avoiding FETI, hybrid FETI, and BETI) are mentioned. The projector-avoiding FETI is shown to have superior scalability compared to the standard FETI method, achieving a speedup of 1.7 on a simple 3D linear elasticity contact problem with 1.2 billion unknowns. Additional results include preconditioned FETI for 3D elastoplasticity and the solution of a 3D hydro-mechanical problem in a porous medium with individually modeled fractures with contact conditions.

8.1 Own Results and Contributions by the Author

The main results of the author are:

- The maintenance and contribution to the development of the PERMON library (Section 3.1.2). The maintenance work includes bug fixes and biannual major version releases that keep the library in sync with the development of PETSc. The contributions can be reviewed on the project's GitHub [25]. A number of projects listed in Section 3.1.2 can now utilize PERMON. Of course, while the author has contributed the most to PERMON in recent years, these contributions are based on the work of others, with special mention of Václav Hapla and David Horák for the initial PERMON development, and the PETSc team for the PETSc library.

- PCDEFLATION (Section 4.3.8) is a multilevel deflation preconditioner that the author contributed to PETSc. The deflation preconditioner is based on previous work [50, 71]. It was developed while the author was on a research stay at the Institute of Mathematics, TU Berlin, hosted by Prof. Reinhard Nabben.

- The modifications of the MPRGP expansion step (Section 5.2). The first modification is the MPPCG algorithm [79], which uses a projected CG step for the expansion of the active set. Additional analyses led to the introduction of several fallback schemes that can do one or more of the following: improve convergence, guarantee convergence, and recover the convergence bound. In [73], the author suggested that a combination of MPRGP, which excels in cost function minimization once the correct active set is identified, and SPG, which converges very quickly but to a limited satisfaction of the KKT conditions, would be of interest. Such a combination of MPRGP using SPG for the active set expansion (MPSPG) has been developed, as well as a potentially improved variant with fused matrix-vector multiplications (MPSPGf). The MPPCG method often exhibits much faster convergence and, at worst, is about equal to the standard MPRGP. The MPSPG variants have even better convergence than MPPCG on some problems but are slightly slower than the standard MPRGP on other problems.

- Approximate preconditioning in face for MPRGP-type methods (Section 5.3). A modification of the preconditioning in face, requiring only one setup of the preconditioner instead of each time the active set changes, has been developed. An error analysis between the approximate and the standard preconditioner is included. The standard preconditioning in face often does not work; while it improves convergence, it is too expensive, resulting in a slowdown. In contrast, numerical experiments show a significant speedup achieved by the new approximate preconditioning in face.

The author made a number of contributions related to the thesis topic that he considers minor[1]. These typically consist of methodology, software, data creation, numerical experiments, and writing part of the accompanying article. Only a few of these results were shown in the thesis. The contributions can be broadly categorized into the following groups:

- Improvements and applications of FETI-type methods that include projector-avoiding FETI [104], scalable strategies for FETI coarse problem solutions [64, 65], hybrid FETI/BETI methods [98, 99, 103], FETI preconditioners for elastoplasticity [100], FETI for slope stability [97], node renumbering for FETI stiffness matrix factorization [102], and energy efficiency of FETI and BLAS routines used in FETI implementations [101, 105–107].

- Solution of QP problems that include contact problems in hydro-mechanics [4], contact problems with friction in mechanics [82], and linear support vector machine classifiers [43].

- Unconstrained QP and preconditioning, including Schwarz domain decomposition preconditioners for Darcy flow [56], and inner product free methods and preconditioners for 3×3 block matrices [57].

8.2 Future Work

Future work includes publishing the new results related to the MPPCG fallback, MPSPG variants, and approximate preconditioning in face presented in this thesis.

As for future research, using the approximate preconditioning in face with the known preconditioners for FETI methods could lead to an extremely fast solver for contact problems in mechanics. Other topics include improvements to the SMALE algorithm and the use of interior point methods for the solution of contact problems employing the FETI method.

[1]Minor in the sense of the thesis author's own contribution and not the result as a whole. In these cases, the thesis author's contribution to the resulting article is typically less than 30%.

Bibliography

[1] P. M. Pardalos and S. A. Vavasis, „Quadratic programming with one negative eigenvalue is NP-hard“, *Journal of Global Optimization*, vol. 1, no. 1, pp. 15–22, 1991. DOI: 10.1007/bf00120662.

[2] P. M. Pardalos and J. B. Rosen, „Methods for global concave minimization: A bibliographic survey“, *SIAM Review*, vol. 28, no. 3, pp. 367–379, 1986. DOI: 10.1137/1028106.

[3] P. M. Pardalos and J. B. Rosen, Eds., *Constrained Global Optimization: Algorithms and Applications*. Springer-Verlag, 1987. DOI: 10.1007/bfb0000035.

[4] J. Stebel, J. Kružík, D. Horák, J. Březina and M. Béreš, „On the parallel solution of hydro-mechanical problems with fracture networks and contact conditions“, *Computers & Structures*, vol. 298, p. 107 339, 2024, ISSN: 0045-7949. DOI: 10.1016/j.compstruc.2024.107339.

[5] *PERMON web page*, http://permon.vsb.cz, 2016. (visited on 31/10/2023).

[6] S. Balay, S. Abhyankar, M. F. Adams, S. Benson, J. Brown, P. Brune, K. Buschelman, E. M. Constantinescu, L. Dalcin, A. Dener, V. Eijkhout, J. Faibussowitsch, W. D. Gropp, V. Hapla, T. Isaac, P. Jolivet, D. Karpeev, D. Kaushik, M. G. Knepley, F. Kong, S. Kruger, D. A. May, L. C. McInnes, R. T. Mills, L. Mitchell, T. Munson, J. E. Roman, K. Rupp, P. Sanan, J. Sarich, B. F. Smith, S. Zampini, H. Zhang, H. Zhang and J. Zhang, *PETSc web page*, https://petsc.org/, 2024. (visited on 24/07/2024).

[7] C. Farhat and F.-X. Roux, „A method of finite element tearing and interconnecting and its parallel solution algorithm“, *International Journal for Numerical Methods in Engineering*, vol. 32, no. 6, pp. 1205–1227, 1991, ISSN: 1097-0207. DOI: 10.1002/nme.1620320604.

[8] Z. Dostál, D. Horák and R. Kučera, „Total FETI – an easier implementable variant of the FETI method for numerical solution of elliptic PDE“, *Communications in Numerical Methods in Engineering*, vol. 22, no. 12, pp. 1155–1162, 2006. DOI: 10.1002/cnm.881.

[9] D. P. Bertsekas, *Nonlinear Programming*. Athena Scientific, Belmont, 1999, ISBN: 978-1-886529-05-2.

[10] Z. Dostál, *Optimal Quadratic Programming Algorithms, with Applications to Variational Inequalities*. SOIA, Springer, New York, US, 2009, vol. 23, ISBN: 0387848053.

[11] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, England: Cambridge University Press, 2004.

[12] R. T. Rockafellar, *Convex Analysis*. Princeton University Press, 1970, ISBN: 9781400873173. DOI: 10.1515/9781400873173.

[13] R. T. Rockafellar, „Lagrange multipliers and optimality“, *SIAM Review*, vol. 35, no. 2, pp. 183–238, 1993, ISSN: 1095-7200. DOI: 10.1137/1035044.

[14] A. Ben-Israel and T. N. E. Greville, *Generalized Inverses* (CMS Books in Mathematics), en, 2nd ed. New York, NY: Springer, 2003. DOI: 10.1007/b97366.

[15] M. Frank and P. Wolfe, „An algorithm for quadratic programming“, *Naval Research Logistics Quarterly*, vol. 3, no. 1–2, pp. 95–110, 1956, ISSN: 1931-9193. DOI: 10.1002/nav.3800030109.

[16] J. E. Martínez-Legaz, D. Noll and W. Sosa, „Non-polyhedral extensions of the Frank and Wolfe theorem“, in *Splitting Algorithms, Modern Operator Theory, and Applications*. Springer International Publishing, 2019, pp. 309–329, ISBN: 9783030259396. DOI: 10.1007/978-3-030-25939-6_12.

[17] D. W. Peterson, „A review of constraint qualifications in finite-dimensional spaces“, *SIAM Review*, vol. 15, no. 3, pp. 639–654, 1973, ISSN: 1095-7200. DOI: 10.1137/1015075.

[18] G. Strang, *Computational Science and Engineering*. Wellesley-Cambridge Press, 2007, ISBN: 9780961408817.

[19] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang and H. Zhang, „PETSc users manual“, Argonne National Laboratory, Tech. Rep. ANL-95/11 - Revision 3.10, 2018.

[20] S. Balay, W. D. Gropp, L. C. McInnes and B. F. Smith, „Efficient management of parallelism in object oriented numerical software libraries“, in *Modern Software Tools in Scientific Computing*, E. Arge, A. M. Bruaset and H. P. Langtangen, Eds., Birkhäuser Press, 1997, pp. 163–202.

[21] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent and J. Koster, „A fully asynchronous multifrontal solver using distributed dynamic scheduling“, *SIAM Journal on Matrix Analysis and Applications*, vol. 23, no. 1, pp. 15–41, 2001, ISSN: 1095-7162. DOI: 10.1137/s0895479899358194.

[22] P. R. Amestoy, A. Buttari, J.-Y. L'Excellent and T. Mary, „Performance and scalability of the block low-rank multifrontal factorization on multicore architectures", *ACM Transactions on Mathematical Software*, vol. 45, no. 1, pp. 1–26, 2019, ISSN: 1557-7295. DOI: 10.1145/3242094.

[23] X. S. Li, „An overview of SuperLU: Algorithms, implementation, and user interface", *ACM Transactions on Mathematical Software*, vol. 31, no. 3, pp. 302–325, 2005, ISSN: 1557-7295. DOI: 10.1145/1089014.1089017.

[24] V. Hapla, „Massively parallel quadratic programming solvers with applications in mechanics", Available at http://hdl.handle.net/10084/112271, PhD thesis, VSB - Technical University of Ostrava, 2016.

[25] *PERMON project repository*, https://github.com/permon. (visited on 24/07/2024).

[26] M. Pecha, „Solvers and their implementations for machine learning problems and applications", PhD thesis, VSB - Technical University of Ostrava, 2024.

[27] A. K. Turner, K. J. Peterson and D. Bolintineanu, „Geometric remapping of particle distributions in the discrete element model for sea ice (DEMSI v0.0)", *Geoscientific Model Development*, vol. 15, no. 5, pp. 1953–1970, 2022, ISSN: 1991-9603. DOI: 10.5194/gmd-15-1953-2022.

[28] *Flow123d web page*, http://flow123d.github.io. (visited on 31/10/2023).

[29] *HyTeG repository*, https://i10git.cs.fau.de/hyteg/hyteg. (visited on 31/10/2023).

[30] *SIFEL web page*, http://mech.fsv.cvut.cz/~sifel. (visited on 31/10/2023).

[31] *ARCHER web page*, http://archer.ac.uk. (visited on 31/10/2023).

[32] *Top500 list web page*, https://www.top500.org/lists/top500/. (visited on 17/07/2024).

[33] *LUMI web page*, https://lumi-supercomputer.eu. (visited on 31/10/2023).

[34] *MareNostrum 3 web page*, https://www.bsc.es/marenostrum/marenostrum/mn3. (visited on 31/10/2023).

[35] *SLBQPgen code*, https://github.com/diserafi/P2GP, 2018. (visited on 31/10/2023).

[36] D. di Serafino, G. Toraldo, M. Viola and J. Barlow, „A two-phase gradient method for quadratic programming problems with a single linear constraint and bounds on the variables", *SIAM Journal on Optimization*, vol. 28, no. 4, pp. 2809–2838, 2018. DOI: 10.1137/17m1128538.

[37] *BQP benchmarks*, https://github.com/jkruzik/qp_benchmarks. (visited on 24/07/2024).

[38] B. Averick, R. Carter, G.-L. Xue and J. More, „The MINPACK-2 test problem collection", Office of Scientific and Technical Information (OSTI), Tech. Rep., 1992. DOI: 10.2172/79972.

[39] G. Cimatti and O. Menchi, „On the numerical solution of a variational inequality connected with the hydrodynamic lubrication of a complete journal bearing“, *Calcolo*, vol. 15, no. 3, pp. 249–258, 1978, ISSN: 1126-5434. DOI: 10.1007/bf02575916.

[40] J. Nečas and I. Hlaváček, *Mathematical Theory of Elastic and Elasto-Plastic Bodies: An Introduction* (Studies in applied mechanics 3). Elsevier, 1981, ISBN: 0444997547.

[41] J. Březina and J. Stebel, „Discrete fracture-matrix model of poroelasticity“, *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 104, no. 4, 2024, ISSN: 1521-4001. DOI: 10.1002/zamm.202200469.

[42] C. Cortes and V. Vapnik, „Support-vector networks“, *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995, ISSN: 0885-6125. DOI: 10.1023/A:1022627411411.

[43] J. Kružík, M. Pecha, V. Hapla, D. Horák and M. Čermák, „Investigating convergence of linear SVM implemented in PermonSVM employing MPRGP algorithm“, in *High Performance Computing in Science and Engineering*, T. Kozubek, M. Čermák, P. Tichý, R. Blaheta, J. Šístek, D. Lukáš and J. Jaroš, Eds., Cham: Springer International Publishing, 2018, pp. 115–129, ISBN: 978-3-319-97136-0. DOI: 10.1007/978-3-319-97136-0_9.

[44] *Libsvm data: Classification (binary class)*, https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html. (visited on 31/10/2023).

[45] J. Dongarra and F. Sullivan, „Guest editors introduction to the top 10 algorithms“, *Computing in Science Engineering*, vol. 2, no. 1, pp. 22–23, 2000, ISSN: 1521-9615. DOI: 10.1109/MCISE.2000.814652.

[46] J. Kružík, *PCDEFLATION - deflation preconditioner in PETSc*, https://petsc.org/main/manualpages/PC/PCDEFLATION/, 2019. (visited on 31/10/2023).

[47] H. Akaike, „On a successive transformation of probability distribution and its application to the analysis of the optimum gradient method“, *Annals of the Institute of Statistical Mathematics*, vol. 11, no. 1, pp. 1–16, 1959, ISSN: 1572-9052. DOI: 10.1007/bf01831719.

[48] Y. Huang, Y.-H. Dai, X.-W. Liu and H. Zhang, „On the asymptotic convergence and acceleration of gradient methods“, *Journal of Scientific Computing*, vol. 90, no. 1, 2021, ISSN: 1573-7691. DOI: 10.1007/s10915-021-01685-8.

[49] J. BARZILAI and J. M. BORWEIN, „Two-point step size gradient methods“, *IMA Journal of Numerical Analysis*, vol. 8, no. 1, pp. 141–148, 1988, ISSN: 1464-3642. DOI: 10.1093/imanum/8.1.141.

[50] J. Kružík, „Implementation of the deflated variants of the conjugate gradient method“, Available at http://hdl.handle.net/10084/130303, Master's thesis, VSB - Technical University of Ostrava, 2018.

[51] G. H. Golub and C. F. van Loan, *Matrix Computations*, 4th. JHU Press, 2013, ISBN: 1421407949.

[52] A. van der Sluis and H. A. van der Vorst, „The rate of convergence of conjugate gradients“, *Numerische Mathematik*, vol. 48, no. 5, pp. 543–560, 1986, ISSN: 0945-3245. DOI: 10.1007/BF01389450.

[53] A. Greenbaum, *Iterative Methods for Solving Linear Systems*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1997, ISBN: 0-89871-396-X.

[54] A. Greenbaum, „Estimating the attainable accuracy of recursively computed residual methods“, *SIAM Journal on Matrix Analysis and Applications*, vol. 18, no. 3, pp. 535–551, 1997. DOI: 10.1137/S0895479895284944.

[55] Y. Saad, *Iterative Methods for Sparse Linear Systems*, Second. Society for Industrial and Applied Mathematics, 2003, ISBN: 0898715342. DOI: 10.1137/1.9780898718003.

[56] R. Blaheta, T. Luber and J. Kružík, „Schur complement-Schwarz DD preconditioners for non-stationary Darcy flow problems“, in *High Performance Computing in Science and Engineering*, T. Kozubek, M. Čermák, P. Tichý, R. Blaheta, J. Šístek, D. Lukáš and J. Jaroš, Eds., Cham: Springer International Publishing, 2018, pp. 59–72, ISBN: 9783319971360. DOI: 10.1007/978-3-319-97136-0_5.

[57] O. Axelsson, Z.-Z. Liang, J. Kruzik and D. Horak, „Inner product free iterative solution and elimination methods for linear systems of a three-by-three block matrix form“, *Journal of Computational and Applied Mathematics*, vol. 383, p. 113 117, 2021, ISSN: 0377-0427. DOI: 10.1016/j.cam.2020.113117.

[58] R. A. Nicolaides, „Deflation of conjugate gradients with applications to boundary value problems“, *SIAM Journal on Numerical Analysis*, vol. 24, no. 2, pp. 355–365, 1987. DOI: 10.1137/0724027.

[59] G. I. Marchuk and Y. A. Kuznetsov, „Theory and applications of the generalized conjugate gradient method“, *Advances in Mathematics. Supplementary Studies*, vol. 10, pp. 153–167, 1986.

[60] Z. Dostal, „Conjugate gradient method with preconditioning by projector“, *International Journal of Computer Mathematics*, vol. 23, no. 3-4, pp. 315–323, 1988. DOI: 10.1080/00207168808803625.

[61] J. Erhel and F. Guyomarc'h, „An augmented conjugate gradient method for solving consecutive symmetric positive definite linear systems“, *SIAM Journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1279–1299, 2000.

[62] Y. Saad, M. Yeung, J. Erhel and F. Guyomarc'h, „A deflated version of the conjugate gradient algorithm“, *SIAM Journal on Scientific Computing*, vol. 21, no. 5, pp. 1909–1926, 2000. DOI: 10.1137/S1064829598339761.

[63] J. M. Tang, R. Nabben, C. Vuik and Y. A. Erlangga, „Comparison of two-level preconditioners derived from deflation, domain decomposition and multigrid methods“, *Journal of scientific computing*, vol. 39, no. 3, pp. 340–370, 2009.

[64] J. Kruzik, D. Horak, V. Hapla and M. Cermak, „Comparison of selected FETI coarse space projector implementation strategies“, *Parallel Computing*, vol. 93, p. 102 608, 2020. DOI: 10.1016/j.parco.2020.102608.

[65] A. Vašatová, J. Tomčala, R. Sojka, M. Pecha, J. Kružík, D. Horák, V. Hapla and M. Čermák, „Parallel strategies for solving the FETI coarse problem in the PERMON toolbox“, *Programs and Algorithms of Numerical Mathematics*, pp. 154–163, 2017.

[66] R. Nabben and C. Vuik, „A comparison of deflation and the balancing preconditioner“, *SIAM Journal on Scientific Computing*, vol. 27, no. 5, pp. 1742–1759, 2006.

[67] K. Kahl and H. Rittich, „The deflated conjugate gradient method: Convergence, perturbation and accuracy“, *Linear Algebra and its Applications*, vol. 515, pp. 111–129, 2017, ISSN: 0024-3795. DOI: 10.1016/j.laa.2016.10.027.

[68] V. Simoncini and D. B. Szyld, „Theory of inexact Krylov subspace methods and applications to scientific computing“, *SIAM Journal on Scientific Computing*, vol. 25, no. 2, pp. 454–477, 2003.

[69] J. Van Den Eshof and G. L. Sleijpen, „Inexact Krylov subspace methods for linear systems“, *SIAM Journal on Matrix Analysis and Applications*, vol. 26, no. 1, pp. 125–153, 2004.

[70] L. García Ramos, R. Kehl and R. Nabben, „Projections, deflation, and multigrid for nonsymmetric matrices“, *SIAM Journal on Matrix Analysis and Applications*, vol. 41, no. 1, pp. 83–105, 2020, ISSN: 1095-7162. DOI: 10.1137/18m1180268.

[71] J. Kruzik and D. Horak, „Wavelet based deflation of conjugate gradient method“, in *Proceedings of the Fifth International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering*, Civil-Comp Press, Stirlingshire, UK, 2017. DOI: 10.4203/ccp.111.9.

[72] J. E. Roman, C. Campos, E. Romero and A. Tomas, „SLEPc users manual“, D. Sistemes Informàtics i Computació, Universitat Politècnica de València, Tech. Rep. DSIC-II/24/02 - Revision 3.9, 2018.

[73] S. Crisci, J. Kružík, M. Pecha and D. Horák, „Comparison of active-set and gradient projection-based algorithms for box-constrained quadratic programming“, *Soft Computing*, vol. 24, no. 23, pp. 17 761–17 770, 2020. DOI: 10.1007/s00500-020-05304-w.

[74] Z. Dostál and J. Schöberl, „Minimizing quadratic functions subject to bound constraints with the rate of convergence and finite termination“, *Computational Optimization and Applications*, vol. 30, no. 1, pp. 23–43, 2005. DOI: 10.1007/s10589-005-4557-7.

[75] J. Bouchala, Z. Dostál, T. Kozubek, L. Pospíšil and P. Vodstrčil, „On the solution of convex QPQC problems with elliptic and other separable constraints with strong curvature“, *Applied Mathematics and Computation*, vol. 247, pp. 848–864, 2014. DOI: 10.1016/j.amc.2014.09.044.

[76] L. Pospíšil, „Development of algorithms for solving minimizing problems with convex quadratic function on special convex sets and applications“, Available at http://hdl.handle.net/10084/110918, PhD thesis, VSB - Technical University of Ostrava, 2015.

[77] H. A. van der Vorst and Q. Ye, „Residual replacement strategies for Krylov subspace iterative methods for the convergence of true residuals“, *SIAM Journal on Scientific Computing*, vol. 22, no. 3, pp. 835–852, 2000. DOI: 10.1137/S1064827599353865.

[78] Z. Strakoš and P. Tichý, „On error estimation in the conjugate gradient method and why it works in finite precision computations.“, *ETNA. Electronic Transactions on Numerical Analysis*, vol. 13, pp. 56–80, 2002.

[79] J. Kružík, D. Horák, M. Čermák, L. Pospíšil and M. Pecha, „Active set expansion strategies in MPRGP algorithm“, *Advances in Engineering Software*, vol. 149, 2020, ISSN: 0965-9978. DOI: 10.1016/j.advengsoft.2020.102895.

[80] E. G. Birgin, J. M. Martínez and M. Raydan, „Nonmonotone spectral projected gradient methods on convex sets“, *SIAM Journal on Optimization*, vol. 10, no. 4, pp. 1196–1211, 2000. DOI: 10.1137/s1052623497330963.

[81] L. Grippo, F. Lampariello and S. Lucidi, „A nonmonotone line search technique for newton's method“, *SIAM Journal on Numerical Analysis*, vol. 23, no. 4, pp. 707–716, 1986. DOI: 10.1137/0723046.

[82] L. Pospíšil, M. Čermák, D. Horák and J. Kružík, „Non-monotone projected gradient method in linear elasticity contact problems with given friction“, *Sustainability*, vol. 12, no. 20, p. 8674, 2020, ISSN: 2071-1050. DOI: 10.3390/su12208674.

[83] S. Crisci, V. Ruggiero and L. Zanni, „Steplength selection in gradient projection methods for box-constrained quadratic programs“, *Applied Mathematics and Computation*, vol. 356, pp. 312–327, 2019. DOI: 10.1016/j.amc.2019.03.039.

[84] S. Crisci, F. Porta, V. Ruggiero and L. Zanni, „Spectral properties of Barzilai–Borwein rules in solving singly linearly constrained optimization problems subject to lower and upper bounds“, *SIAM Journal on Optimization*, vol. 30, no. 2, pp. 1300–1326, 2020, ISSN: 1095-7189. DOI: 10.1137/19m1268641.

[85] G. Goumas, K. Kourtis, N. Anastopoulos, V. Karakasis and N. Koziris, „Understanding the performance of sparse matrix-vector multiplication“, in *16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2008)*, 2008, pp. 283–292. DOI: 10.1109/PDP.2008.41.

[86] R. Kannan, „Efficient sparse matrix multiple-vector multiplication using a bitmapped format“, in *20th Annual International Conference on High Performance Computing*, 2013, pp. 286–294. DOI: 10.1109/HiPC.2013.6799135.

[87] L. Pospíšil, „An optimal algorithm with Barzilai–Borwein steplength and superrelaxation for QPQC problem“, in *Programs and Algorithms of Numerical Mathematics*, J. Chleboun, K. Segeth, J. Šístek and T. Vejchodský, Eds., vol. Proceedings of Seminar. Dolní Maxov, June 3-8, 2012, Prague: Institute of Mathematics AS CR, 2013, pp. 155–161.

[88] D. P. O'Leary, „A generalized conjugate gradient algorithm for solving a class of quadratic programming problems“, *Linear Algebra and its Applications*, vol. 34, pp. 371–399, 1980. DOI: 10.1016/0024-3795(80)90173-1.

[89] B. Polyak, „The conjugate gradient method in extremal problems“, *USSR Computational Mathematics and Mathematical Physics*, vol. 9, no. 4, pp. 94–112, 1969. DOI: 10.1016/0041-5553(69)90035-4.

[90] M. Domorádová and Z. Dostál, „Projector preconditioning for partially bound-constrained quadratic optimization“, *Numerical Linear Algebra with Applications*, vol. 14, no. 10, pp. 791–806, 2007. DOI: 10.1002/nla.555.

[91] M. Jarošová, A. Klawonn and O. Rheinbach, „Projector preconditioning and transformation of basis in FETI-DP algorithms for contact problems“, *Mathematics and Computers in Simulation*, vol. 82, no. 10, pp. 1894–1907, 2012. DOI: 10.1016/j.matcom.2010.10.031.

[92] T. F. Chan and H. A. Van der Vorst, „Approximate and incomplete factorizations“, in *Parallel Numerical Algorithms*, D. E. Keyes, A. Sameh and V. Venkatakrishnan, Eds. Dordrecht: Springer Netherlands, 1997, pp. 167–202, ISBN: 978-94-011-5412-3. DOI: 10.1007/978-94-011-5412-3_6.

[93] D. M. Young, *Iterative Solution of Large Linear Systems*. Academic Press, 1971, ISBN: 9780127730509.

[94] M. Benzi, G. H. Golub and J. Liesen, „Numerical solution of saddle point problems“, *Acta Numerica*, vol. 14, pp. 1–137, 2005, ISSN: 1474-0508. DOI: 10.1017/s0962492904000212.

[95] M. R. Hestenes, „Multiplier and gradient methods“, *Journal of Optimization Theory and Applications*, vol. 4, no. 5, pp. 303–320, 1969, ISSN: 1573-2878. DOI: 10.1007/bf00927673.

[96] D. Horak, V. Hapla, J. Kruzik, R. Sojka, M. Cermak, J. Tomcala, M. Pecha and Z. Dostal, „A note on massively parallel implementation of FETI for the solution of contact problems“, *Advances in Electrical and Electronic Engineering*, vol. 15, no. 2, 2017, ISSN: 1336-1376. DOI: 10.15598/aeee.v15i2.2321.

[97] D. Horák, J. Kružík, J. Kruis and T. Koudelka, „Slip condition in slope stability analysis solved by FETI method“, in *International Conference of Numerical Analysis and Applied Mathematics ICNAAM 2021*, AIP Publishing, 2023. DOI: 10.1063/5.0162232.

[98] P. Vodstrčil, D. Lukáš, Z. Dostál, M. Sadowská, D. Horák, O. Vlach, J. Bouchala and J. Kružík, „On favorable bounds on the spectrum of discretized Steklov–Poincaré operator and applications to domain decomposition methods in 2d“, *Computers & Mathematics with Applications*, vol. 167, pp. 12–20, 2024, ISSN: 0898-1221. DOI: 10.1016/j.camwa.2024.04.033.

[99] Z. Dostál, D. Horák, J. Kružík, T. Brzobohatý and O. Vlach, „Highly scalable hybrid domain decomposition method for the solution of huge scalar variational inequalities“, *Numerical Algorithms*, vol. 91, no. 2, pp. 773–801, 2022, ISSN: 1572-9265. DOI: 10.1007/s11075-022-01281-3.

[100] M. Čermák, V. Hapla, J. Kružík, A. Markopoulos and A. Vašatová, „Comparison of different FETI preconditioners for elastoplasticity“, *Computers & Mathematics with Applications*, vol. 74, no. 1, pp. 96–109, 2017, ISSN: 0898-1221. DOI: 10.1016/j.camwa.2017.01.003.

[101] J. Schuchart, M. Gerndt, P. G. Kjeldsberg, M. Lysaght, D. Horák, L. Říha, A. Gocht, M. Sourouri, M. Kumaraswamy, A. Chowdhury, M. Jahre, K. Diethelm, O. Bouizi, U. S. Mian, J. Kružík, R. Sojka, M. Beseda, V. Kannan, Z. Bendifallah, D. Hackenberg and W. E. Nagel, „The READEX formalism for automatic tuning for energy efficiency“, *Computing*, vol. 99, no. 8, pp. 727–745, 2017, ISSN: 1436-5057. DOI: 10.1007/s00607-016-0532-7.

[102] D. Hrbáč, J. Kružík, D. Horák and J. Kruis, „Node renumbering strategies for efficient direct methods in selected problems of soil mechanics“, in *International Conference of Numerical Analysis and Applied Mathematics ICNAAM 2021*, AIP Publishing, 2023. DOI: 10.1063/5.0163768.

[103] Z. Dostál, T. Brzobohatý, D. Horák, J. Kružík and O. Vlach, „Scalable hybrid TFETI-DP methods for large boundary variational inequalities“, in *Domain Decomposition Methods in Science and Engineering XXVI*, S. C. Brenner, E. Chung, A. Klawonn, F. Kwok, J. Xu and J. Zou, Eds., Cham: Springer International Publishing, 2022, pp. 29–40, ISBN: 978-3-030-95025-5. DOI: 10.1007/978-3-030-95025-5_3.

[104] D. Horak, Z. Dostal, V. Hapla, J. Kruzik, R. Sojka and M. Cermak, „Projectorless TFETI for contact problems: Preliminary results“, in *Proceedings of the Fifth International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering*, ser. PARENG 2017, Civil-Comp Press. DOI: 10.4203/ccp.111.8.

[105] D. Horak, L. Riha, R. Sojka, J. Kruzik, M. Beseda, M. Cermak and J. Schuchart, „Energy consumption optimization of the total-FETI solver by changing the cpu frequency“, in *AIP Conference Proceedings*, AIP Publishing, 2017. DOI: 10.1063/1.4992511.

[106] R. Sojka, L. Riha, D. Horak, J. Kruzik, M. Beseda and M. Cermak, „The energy consumption optimization of the BLAS routines“, in *AIP Conference Proceedings*, AIP Publishing, 2017. DOI: 10.1063/1.4992522.

[107] D. Horak, L. Riha, R. Sojka, J. Kruzik and M. Beseda, „Energy consumption optimization of the Total-FETI solver and BLAS routines by changing the cpu frequency“, in *2016 International Conference on High Performance Computing & Simulation (HPCS)*, IEEE, 2016. DOI: 10.1109/hpcsim.2016.7568453.

[108] Z. Dostál, F. A. Gomes Neto and S. A. Santos, „Solution of contact problems by FETI domain decomposition with natural coarse space projections“, *Computer Methods in Applied Mechanics and Engineering*, vol. 190, no. 13–14, pp. 1611–1627, 2000, ISSN: 0045-7825. DOI: 10.1016/s0045-7825(00)00180-8.

[109] Z. Dostál, T. Kozubek, M. Sadowská and V. Vondrák, *Scalable Algorithms for Contact Problems*. Springer New York, 2016, ISBN: 9781493968343. DOI: 10.1007/978-1-4939-6834-3.

[110] Z. Dostál, T. Kozubek, A. Markopoulos and M. Menšík, „Cholesky decomposition of a positive semidefinite matrix with known kernel“, *Applied Mathematics and Computation*, vol. 217, no. 13, pp. 6067–6077, 2011, ISSN: 0096-3003. DOI: 10.1016/j.amc.2010.12.069.

[111] C. Farhat, J. Mandel and F. X. Roux, „Optimal convergence properties of the FETI domain decomposition method“, *Computer Methods in Applied Mechanics and Engineering*, vol. 115, no. 3–4, pp. 365–385, 1994, ISSN: 0045-7825. DOI: 10.1016/0045-7825(94)90068-x.

[112] J. Mandel and R. Tezaur, „Convergence of a substructuring method with Lagrange multipliers“, *Numerische Mathematik*, vol. 73, no. 4, pp. 473–487, 1996, ISSN: 0945-3245. DOI: 10.1007/s002110050201.

[113] A. Klawonn and O. Rheinbach, „A hybrid approach to 3-level FETI“, *PAMM*, vol. 8, no. 1, pp. 10 841–10 843, 2008, ISSN: 1617-7061. DOI: 10.1002/pamm.200810841.

Appendix A

List of Author's Publications

The following is a list of the author's publications indexed in Web of Science or Scopus.

Articles Related to the Thesis

[4] J. Stebel, **J. Kružík**, D. Horák, J. Březina and M. Béreš, „On the parallel solution of hydro-mechanical problems with fracture networks and contact conditions", *Computers & Structures*, vol. 298, p. 107 339, 2024, ISSN: 0045-7949. DOI: 10.1016/j.compstruc.2024.107339.

[57] O. Axelsson, Z.-Z. Liang, **J. Kruzik** and D. Horak, „Inner product free iterative solution and elimination methods for linear systems of a three-by-three block matrix form", *Journal of Computational and Applied Mathematics*, vol. 383, p. 113 117, 2021, ISSN: 0377-0427. DOI: 10.1016/j.cam.2020.113117.

[64] **J. Kruzik**, D. Horak, V. Hapla and M. Cermak, „Comparison of selected FETI coarse space projector implementation strategies", *Parallel Computing*, vol. 93, p. 102 608, 2020. DOI: 10.1016/j.parco.2020.102608.

[65] A. Vašatová, J. Tomčala, R. Sojka, M. Pecha, **J. Kružík**, D. Horák, V. Hapla and M. Čermák, „Parallel strategies for solving the FETI coarse problem in the PERMON toolbox", *Programs and Algorithms of Numerical Mathematics*, pp. 154–163, 2017.

[73] S. Crisci, **J. Kružík**, M. Pecha and D. Horák, „Comparison of active-set and gradient projection-based algorithms for box-constrained quadratic programming", *Soft Computing*, vol. 24, no. 23, pp. 17 761–17 770, 2020. DOI: 10.1007/s00500-020-05304-w.

[79] **J. Kružík**, D. Horák, M. Čermák, L. Pospíšil and M. Pecha, „Active set expansion strategies in MPRGP algorithm", *Advances in Engineering Software*, vol. 149, 2020, ISSN: 0965-9978. DOI: 10.1016/j.advengsoft.2020.102895.

[82] L. Pospíšil, M. Čermák, D. Horák and **J. Kružík**, „Non-monotone projected gradient method in linear elasticity contact problems with given friction", *Sustainability*, vol. 12, no. 20, p. 8674, 2020, ISSN: 2071-1050. DOI: 10.3390/su12208674.

[96] D. Horak, V. Hapla, **J. Kruzik**, R. Sojka, M. Cermak, J. Tomcala, M. Pecha and Z. Dostal, „A note on massively parallel implementation of FETI for the solution of contact problems“, *Advances in Electrical and Electronic Engineering*, vol. 15, no. 2, 2017, ISSN: 1336-1376. DOI: 10.15598/aeee.v15i2.2321.

[98] P. Vodstrčil, D. Lukáš, Z. Dostál, M. Sadowská, D. Horák, O. Vlach, J. Bouchala and **J. Kružík**, „On favorable bounds on the spectrum of discretized Steklov–Poincaré operator and applications to domain decomposition methods in 2d“, *Computers & Mathematics with Applications*, vol. 167, pp. 12–20, 2024, ISSN: 0898-1221. DOI: 10.1016/j.camwa.2024.04.033.

[99] Z. Dostál, D. Horák, **J. Kružík**, T. Brzobohatý and O. Vlach, „Highly scalable hybrid domain decomposition method for the solution of huge scalar variational inequalities“, *Numerical Algorithms*, vol. 91, no. 2, pp. 773–801, 2022, ISSN: 1572-9265. DOI: 10.1007/s11075-022-01281-3.

[100] M. Čermák, V. Hapla, **J. Kružík**, A. Markopoulos and A. Vašatová, „Comparison of different FETI preconditioners for elastoplasticity“, *Computers & Mathematics with Applications*, vol. 74, no. 1, pp. 96–109, 2017, ISSN: 0898-1221. DOI: 10.1016/j.camwa.2017.01.003.

[101] J. Schuchart, M. Gerndt, P. G. Kjeldsberg, M. Lysaght, D. Horák, L. Říha, A. Gocht, M. Sourouri, M. Kumaraswamy, A. Chowdhury, M. Jahre, K. Diethelm, O. Bouizi, U. S. Mian, **J. Kružík**, R. Sojka, M. Beseda, V. Kannan, Z. Bendifallah, D. Hackenberg and W. E. Nagel, „The READEX formalism for automatic tuning for energy efficiency“, *Computing*, vol. 99, no. 8, pp. 727–745, 2017, ISSN: 1436-5057. DOI: 10.1007/s00607-016-0532-7.

Conference Proceedings Related to the Thesis

[43] **J. Kružík**, M. Pecha, V. Hapla, D. Horák and M. Čermák, „Investigating convergence of linear SVM implemented in PermonSVM employing MPRGP algorithm“, in *High Performance Computing in Science and Engineering*, T. Kozubek, M. Čermák, P. Tichý, R. Blaheta, J. Šístek, D. Lukáš and J. Jaroš, Eds., Cham: Springer International Publishing, 2018, pp. 115–129, ISBN: 978-3-319-97136-0. DOI: 10.1007/978-3-319-97136-0_9.

[56] R. Blaheta, T. Luber and **J. Kružík**, „Schur complement-Schwarz DD preconditioners for non-stationary Darcy flow problems“, in *High Performance Computing in Science and Engineering*, T. Kozubek, M. Čermák, P. Tichý, R. Blaheta, J. Šístek, D. Lukáš and J. Jaroš, Eds., Cham: Springer International Publishing, 2018, pp. 59–72, ISBN: 9783319971360. DOI: 10.1007/978-3-319-97136-0_5.

[71] **J. Kruzik** and D. Horak, „Wavelet based deflation of conjugate gradient method“, in *Proceedings of the Fifth International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering*, Civil-Comp Press, Stirlingshire, UK, 2017. DOI: 10.4203/ccp.111.9.

[97] D. Horák, **J. Kružík**, J. Kruis and T. Koudelka, „Slip condition in slope stability analysis solved by FETI method“, in *International Conference of Numerical Analysis and Applied Mathematics ICNAAM 2021*, AIP Publishing, 2023. DOI: 10.1063/5.0162232.

[102] D. Hrbáč, **J. Kružík**, D. Horák and J. Kruis, „Node renumbering strategies for efficient direct methods in selected problems of soil mechanics“, in *International Conference of Numerical Analysis and Applied Mathematics ICNAAM 2021*, AIP Publishing, 2023. DOI: 10.1063/5.0163768.

[103] Z. Dostál, T. Brzobohatý, D. Horák, **J. Kružík** and O. Vlach, „Scalable hybrid TFETI-DP methods for large boundary variational inequalities“, in *Domain Decomposition Methods in Science and Engineering XXVI*, S. C. Brenner, E. Chung, A. Klawonn, F. Kwok, J. Xu and J. Zou, Eds., Cham: Springer International Publishing, 2022, pp. 29–40, ISBN: 978-3-030-95025-5. DOI: 10.1007/978-3-030-95025-5_3.

[104] D. Horak, Z. Dostal, V. Hapla, **J. Kruzik**, R. Sojka and M. Cermak, „Projectorless TFETI for contact problems: Preliminary results“, in *Proceedings of the Fifth International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering*, ser. PARENG 2017, Civil-Comp Press. DOI: 10.4203/ccp.111.8.

[105] D. Horak, L. Riha, R. Sojka, **J. Kruzik**, M. Beseda, M. Cermak and J. Schuchart, „Energy consumption optimization of the total-FETI solver by changing the cpu frequency“, in *AIP Conference Proceedings*, AIP Publishing, 2017. DOI: 10.1063/1.4992511.

[106] R. Sojka, L. Riha, D. Horak, **J. Kruzik**, M. Beseda and M. Cermak, „The energy consumption optimization of the BLAS routines“, in *AIP Conference Proceedings*, AIP Publishing, 2017. DOI: 10.1063/1.4992522.

[107] D. Horak, L. Riha, R. Sojka, **J. Kruzik** and M. Beseda, „Energy consumption optimization of the Total-FETI solver and BLAS routines by changing the cpu frequency“, in *2016 International Conference on High Performance Computing & Simulation (HPCS)*, IEEE, 2016. DOI: 10.1109/hpcsim.2016.7568453.

Publications Unrelated to the Thesis

[114] J. Papuga, R. Halama, M. Fusek, J. Rojíček, F. Fojtík, D. Horák, M. Pecha, J. Tomčala, M. Čermák, V. Hapla, R. Sojka and **J. Kružík**, „Efficient lifetime estimation techniques for general multiaxial loading“, in *AIP Conference Proceedings*, AIP Publishing, 2017. DOI: 10.1063/1.4992518.

Appendix B

List of Projects

The following is a list of projects in which the author of this thesis participated.

- REFRESH - Research Excellence For REgion Sustainability and High-tech Industries
  Grant No. CZ.10.03.01/00/22_003/0000048 (Ministry of the Environment of the Czech Republic)
- International mobility of researchers of IGN II
  Grant No. CZ.02.2.69/0.0/0.0/18_053/0016978 (Ministry of Education, Youth and Sport of the Czech Republic)
- Development of iterative algorithms for solving contact problems emerging in the analysis of steel structures bolt connections
  Grant No. GA22-13220S (Czech Science Foundation (GACR))
- HPC-EUROPA3
  Grant No. INFRAIA-2016-1-730897 (EU Horizon 2020 Research and Innovation Programme)
- Efficient and reliable computational techniques for limit analysis and incremental methods in geotechnical stability
  Grant No. GA19-11441S (Czech Science Foundation (GACR))
- Prediction of EDZ properties with impact on safety and reliability of deep radioactive waste repository (ENDORSE)
  Grant No. TK02010118 (Technology Agency of the Czech Republic)
- European Joint Programme on Radioactive Waste Management (EURAD)
  Grant No. 847593 (EU Horizon 2020 Research and Innovation Programme)
  Grant No. SO2020-017 (Czech Radioactive Waste Repository Authority (SÚRAO))
- Support for Science and Research in the Moravia–Silesia Region 2019
  Grant No. RRC/10/2019 (Moravian-Silesian Region)
- IT4Innovations excellence in science
  Grant No. LQ1602 (Ministry of Education, Youth and Sport of the Czech Republic)
- Runtime Exploitation of Application Dynamism for Energy-efficient eXascale computing

(READEX)
Grant No. 671657 (EU Horizon 2020 Research and Innovation Programme)

- Efficient lifetime estimation techniques for general multiaxial loading
Grant No. GA15-18274S (Czech Science Foundation (GACR))

Student Grant Competition Projects of VSB-TU Ostrava:

- Matematické modelování a vývoj algoritmů pro výpočetně náročné inženýrské úlohy X (Mathematical modeling and algorithm development for computationally intensive engineering problems X)
Grant No. SP2024/067
- Matematické modelování a vývoj algoritmů pro výpočetně náročné inženýrské úlohy IX
Grant No. SP2023/067
- Matematické modelování a vývoj algoritmů pro výpočetně náročné inženýrské úlohy VIII
Grant No. SP2022/42
- Matematické modelování a vývoj algoritmů pro výpočetně náročné inženýrské úlohy VII
Grant No. SP2021/103
- Matematické modelování a vývoj algoritmů pro výpočetně náročné inženýrské úlohy VI
Grant No. SP2020/114
- Matematické modelování a vývoj algoritmů pro výpočetně náročné inženýrské úlohy V
Grant No. SP2019/84
- Matematické modelování a vývoj algoritmů pro výpočetně náročné inženýrské úlohy II
Grant No. SP2016/178
- PERMON toolbox development IV
Grant No. SP2018/169
- PERMON toolbox development III
Grant No. SP2017/169
- PERMON toolbox development II
Grant No. SP2016/178