Massively Parallel Quadratic Programming Solvers with Applications in Mechanics

Ing. Václav Hapla

Doctoral Thesis

Department of Applied Mathematics
Faculty of Electrical Engineering and Computer Science
VŠB – Technical University of Ostrava
Ostrava 2016

Abstract

This thesis focuses on practical solution of large-scale contact problems of structure mechanics by means of a derived quadratic programming (QP) formulation. An approach proposed by professor Dostál, combining a FETI-type non-overlapping domain decomposition method, the SMALBE algorithm based on augmented Lagrangians, and the MPRGP algorithm belonging to active set methods, has been adopted. This approach enjoys theoretically supported numerical scalability and a favourable potential for parallel scalability. The thesis consists of two parts: Background and Implementation.

Background is devoted to rather theoretical aspects of QP and FETI, although tightly connected to practical implementation. Original topics include QP transforms, implicit orthonormalization of equality constraints, and a minor modification of SMALBE shortening its termination phase considerably.

Second part, Implementation, deals with the massively parallel implementation of the aforementioned approach within PERMON, a new set of software libraries established by the author. The most important part, PERMON Solver Core, is formed mainly by the general-purpose QP solver PermonQP, and its extension PermonFLLOP providing support for domain decomposition. These libraries make use of and extend PETSc, an open source software framework for numerical computing. Performance of PERMON is demonstrated on several numerical experiments.

Keywords

quadratic programming, QP, domain decomposition methods, FETI, augmented Lagrangian, SMALBE, MPRGP, contact problems, structure mechanics, PERMON, PermonQP, PermonFLLOP, PETSc

Abstrakt

Tato dizertační práce se zaměřuje na praktické řešení rozsáhlých kontaktních úloh strukturální mechaniky přes odvozenou úlohu kvadratického programování (QP). Přebírá přístup navržený prof. Dostálem, kombinující nepřekrývající metodu rozložení oblasti typu FETI, algoritmus SMALBE založený na rozšířených Lagrangiánech a algoritmus MPRGP náležící k metodám aktivních množin. Tento přístup se těší teoreticky podložené numerické škálovatelnosti a příznivému potenciálu pro paralelní škálovatelnost. Tato práce se skládá ze dvou částí: Teoretické pozadí a Implementace.

Teoretické pozadí se věnuje spíše teoretickým stránkám QP a FETI, i když těsně spjatými s praktickou implementací. K originálním tématům patří QP transformace, implicitní ortonormalizace rovnostních omezení a drobná modifikace SMALBE podstatně zkracující jeho ukončovací fázi.

Druhá část, Implementace, je věnována masivně paralelní implementaci výše uvedeného přístupu v rámci PERMONu, nové sady softwarových knihoven založené autorem. Nejdůležitější částí PERMONu, tzv. Řešičové jádro (PERMON Solver Core), je tvořeno zejména QP řešičem pro obecné použití PermonQP a jeho extenzí PermonFLLOP, poskytující podporu pro rozložení oblasti. Tyto knihovny využívají a rozšiřují PETSc, softwarový rámec pro numerické výpočty s otevřeným kódem. Výkonnost PERMONu je demonstrována na několika numerických experimentech.

Klíčová slova

kvadratické programování, QP, metody rozložení oblasti, FETI, rozšířený Lagrangián, SMALBE, MPRGP, kontaktní úlohy, strukturální mechanika, PERMON, PermonQP, PermonFLLOP, PETSc

Acknowledgement

I would like to thank my supervisor and boss Tomáš Kozubek for his leadership and support; David Horák and Martin Čermák for their friendship, collaboration, help and comments on this thesis; Lukáš Pospíšil who is currently on a long stay but had contributed significantly to PERMON Solver Core and implemented PermonMembrane; Alexandros Markopoulos for PermonCube and joint idea of implicit orthonormalization; and also all other fellows, especially the members of the PERMON team, for their support and help.

I express my gratitude to Department of Applied Mathematics, Faculty of Electrical Engineering and Computer Science and VSB-Technical University of Ostrava for the opportunity to study there the most interesting field of Computational and Applied Mathematics. My thanks go also to IT4Innovations for supporting PERMON development and allowing me to write this thesis in tight connection with my work there.

Last but not least, I would like to thank my wife and my mother for their support and patience during my doctoral studies and finishing this work. And thank you both my little daughters for cheering me up!

In memory of my dear dad.

Author:
Ing. Václav Hapla
vaclav.hapla@vsb.cz

Thesis supervisor:
prof. Ing. Tomáš Kozubek, Ph.D.
tomas.kozubek@vsb.cz

Department of Applied Mathematics
Faculty of Electrical Engineering and Computer Science
VŠB – Technical University of Ostrava
17. listopadu 15, 708 33 Ostrava–Poruba
Czech Republic
http://www.am.vsb.cz
http://fei.vsb.cz
http://www.vsb.cz

Contents

List of Tables

List of Figures

Abbreviations

CP	coarse problem
DDM	domain decomposition method
DOF	degree of freedom
FE	finite element
FEM	Finite Element Method
FETI	Finite Element Tearing and Interconnecting
HPC	high performance computing
KKT	Karush–Kuhn–Tucker conditions
PDE	partial differential equation
QP	quadratic programming, quadratic programming problem, quadratic program
RHS	right hand side vector
SPD	symmetric positive definite
SPS	symmetric positive semidefinite

Symbols

$\mathbb{R}$	field of all real numbers
$\mathbb{R}^+$	set of positive real numbers
$\mathbb{R}_0^+$	set of non-negative real numbers
$\mathbb{R}^n$	linear space of all real-valued column vectors with n entries
$\mathbb{R}^{m\times n}$	linear space of all real-valued matrices with m rows and n columns
$\mathbf{A} = [\,\vdots\,],\ \mathbf{b} = [\,\vdots\,]$	matrix, vector, given entry-wise
$\square^{(0,n)},\ \square^{(n,0)}$	empty matrix with no rows and no columns, respectively
$\square$	the same with obvious or unimportant size
$\mathbf{O}^{(m,n)}, \mathbf{o}^{(m)}$	zero matrix, zero column vector with specified size
$\mathbf{O}, \mathbf{o}$	the same with obvious or unimportant size
$\mathbf{I}^{(m,m)}$	identity matrix with specified number of rows
$\mathbf{I}$	the same with obvious or unimportant size
$\boldsymbol{\infty}^{(m,n)}$	matrix with all values equal to plus infinity, with specified size
$\boldsymbol{\infty}$	the same with obvious or unimportant size
$\mathbf{A}^T$	transpose of matrix $\mathbf{A}$
$\mathbf{A}^{-1}$	inverse of matrix $\mathbf{A}$, i.e. matrix satisfying $\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$
$\mathbf{A}^\dagger$	generalized inverse of matrix $\mathbf{A}$, i.e. matrix satisfying $\mathbf{A}\mathbf{A}^\dagger\mathbf{A} = \mathbf{A}$
$\mathbf{A}^+$	Moore-Penrose pseudoinverse, i.e. matrix satisfying $\mathbf{A}\mathbf{A}^+\mathbf{A} = \mathbf{A}$, $\mathbf{A}^+\mathbf{A}\mathbf{A}^+ = \mathbf{A}^+$, $(\mathbf{A}\mathbf{A}^+)^T = \mathbf{A}\mathbf{A}^+$, $(\mathbf{A}^+\mathbf{A})^T = \mathbf{A}^+\mathbf{A}$
$\mathbf{A}^L$	left inverse of full column rank matrix $\mathbf{A}$, $\mathbf{A}^L = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$, $\mathbf{A}^L\mathbf{A} = \mathbf{I}^{(n,n)}$
$\mathbf{A}^R$	right inverse of full row rank matrix $\mathbf{A}$, $\mathbf{A}^R = \mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}$, $\mathbf{A}\mathbf{A}^R = \mathbf{I}^{(m,m)}$

$\mathcal{I} = (a, \ldots, z)$	index set (integer tuple) $\mathcal{I}$ containing all integers from $[a, z]$
$\lvert\mathcal{I}\rvert$	number of elements in index set $\mathcal{I}$
$[\mathbf{v}]_i$, v_i	i-th element of vector $\mathbf{v}$
$[\mathbf{v}]_\mathcal{I}$, $\mathbf{v}_\mathcal{I}$	subvector of $\mathbf{v}$ given by index set $\mathcal{I}$
$[\mathbf{A}]_{i,j}$, $a_{i,j}$	j-th element of i-th row of matrix $\mathbf{A}$
$[\mathbf{A}]_{\mathcal{I},*}$	submatrix of $\mathbf{A}$ with rows $\mathcal{I}$ and all columns
$[\mathbf{A}]_{*,\mathcal{J}}$	submatrix of $\mathbf{A}$ with all rows and columns $\mathcal{J}$
$[\mathbf{A}]_{\mathcal{I},\mathcal{J}}$, $\mathbf{A}_{\mathcal{I},\mathcal{J}}$	submatrix of $\mathbf{A}$ given by row index set $\mathcal{I}$ and column index set $\mathcal{J}$, i.e. $\mathbf{A}_{\mathcal{I},\mathcal{J}} = [[\mathbf{A}]_{\mathcal{I},*}]_{*,\mathcal{J}}$
$\mathbf{z} = \max(\mathbf{x}, \mathbf{y})$	element-wise maximum of vectors $\mathbf{x}$ and $\mathbf{y}$, i.e. vector with entries $z_i = \max\{x_i, y_i\}$
$\operatorname{Ker}\mathbf{A}$	kernel (nullspace) of matrix $\mathbf{A}$, $\operatorname{Ker}\mathbf{A} = \{\mathbf{x} : \mathbf{A}\mathbf{x} = \mathbf{o}\}$
$\operatorname{Im}\mathbf{A}$	image (range) of matrix $\mathbf{A}$, $\operatorname{Im}\mathbf{A} = \{\mathbf{y} : \exists\mathbf{x}, \mathbf{A}\mathbf{x} = \mathbf{y}\}$
$\sigma_{max}(\mathbf{A})$	maximal singular value of matrix $\mathbf{A}$
$\sigma_{min}(\mathbf{A})$	minimal singular value of matrix $\mathbf{A}$
$\overline{\sigma_{min}}(\mathbf{A})$	minimal *nonzero* singular value of matrix $\mathbf{A}$
$\kappa(\mathbf{A})$	(effective) condition number of matrix $\mathbf{A}$, $\kappa(\mathbf{A}) = \frac{\sigma_{max}(\mathbf{A})}{\overline{\sigma_{min}}(\mathbf{A})}$
$\mathcal{U} \oplus \mathcal{V} = \mathcal{W}$	direct sum of subspaces, $\mathcal{U} + \mathcal{V} = \mathcal{W} \ \wedge \ \mathcal{U} \cap \mathcal{V} = \{\mathbf{o}\}$ where $\mathbf{x}_k \in \mathbb{R}^{n_k}$, $1 \leq k \leq N$
$\nabla_{\mathbf{x}_k} F(\mathbf{x}_1, \ldots, \mathbf{x}_N)$	gradient of $F : (\mathbb{R}^{\sum_{k=1}^{N} n_k}) \mapsto \mathbb{R}$ with respect to $\mathbf{x}_k$,

Introduction

Most problems of mechanics may be described by partial differential equations (PDEs). To be solved with computers, they have to be discretized, e.g. with popular Finite Element Method (FEM). We typically get large sparse linear systems of equations. However, constrained problems such as contact problems of mechanics result in nonlinear optimization problems. Using a natural way of adding the constraints additively to the linear system, quadratic programming problems (QPs) arise. In presence of friction, a generalized formulation of QP has to be used. QPs also arise in other disciplines like least-squares regression, data fitting, data mining, support vector machines, control systems and many others.

FEM simulations with a large enough number of variables are not solvable on usual personal computers due to memory and computational limits. Such problems can be solved only on parallel computers. Suitable numerical methods are needed for that such as domain decomposition methods (DDM) or multigrid. DDM are natural from the user's perspective as they solve the original problem by decomposing ("tearing") into smaller independent subdomain problems which can be solved in parallel. A DDM basically constitutes a mathematical framework how to get a correct solution, continuous across subdomain interfaces, in presence of such tearing.

Finite Element Tearing and Interconnecting (FETI) methods form a successful subclass of DDM. They belong to non-overlapping methods and combine iterative and direct solvers. The FETI methods allow highly accurate computations scaling up to tens of thousands of processors and billions of unknowns. In FETI, subdomain stiffness matrices are assembled, factorized and solved independently whereas conditions of continuity of the solution across subdomain interfaces (gluing conditions) form separate linear equality constraints. In the specific flavour of FETI we typically use, called Total FETI (TFETI), Dirichlet boundary conditions are enforced by means of equality constraints, too. Hence, even in case of linear elasticity, it is advantageous to handle the resulting problem as a special case of QP (equality constrained).

Due to limitations of commercial packages, problems often have to be adapted to be solvable. This is an expensive process and results reflect less accurately physical phenomena. Moreover, it takes a long time before the most recent numerical methods needed for High Performance Computing (HPC) are implemented into such packages. These issues lead the author to establish

the PERMON (Parallel, Efficient, Robust, Modular, Object-oriented, Numerical) toolbox. It is a collection of software libraries, uniquely combining QP algorithms and DDM. PERMON is built on top of the well-known PETSc framework for numerical computations. Among the main applications are contact problems of mechanics. Our PermonFLLOP package is focused on non-overlapping DDM of the FETI type, allowing efficient and robust utilization of contemporary parallel computers for problems with billions of unknowns. Any FEM software can be used to generate mesh and assemble the stiffness matrices and load vectors per each subdomain independently. Additionally, a mapping from the local to the global numbering of degrees of freedom is needed, and non-penetration information in case of contact problems. All these data are then passed to PermonFLLOP, which prepares auxiliary data needed in the DDM. PermonQP is then called in the back-end to solve the resulting equality constrained problem with additional inequality constraints in case of contact problems.

This thesis is driven by practical implementation of QP algorithms and DDM in PERMON, particularly for massively parallel solution of contact problems of mechanics. Its theoretical part limits itself mainly to topics needed for understanding the implementation aspects, with no ambitions to cover all important theoretical aspects. An interested reader can find much more complete theoretical background in the book by prof. Dostál [10] which is the most important reference, introducing the crucial algorithms implemented in PERMON and their solid theoretical support.

Thesis structure

Part I (Background) is devoted mainly to rather theoretical aspects, although driven by implementation. It consists of these chapters:

Chapter 1 (Quadratic programming) briefly presents the field of mathematical optimization and gets quickly to its subfield of QP with special attention paid to practical computation of Lagrange multipliers and evaluation of Karush–Kuhn–Tucker (KKT) optimality conditions.

Chapter 2 (QP transforms) introduces a non-standard notion of QP transform, attempting to formalize transformation of a QP problem to ease its solution. After defining the concept, several concrete instances are presented as well as several examples of their usage. This topic appears for the first time here.

Chapter 3 (QP algorithms) discusses two particular QP solvers implemented in PERMON, MPRGP and SMALBE-M by Dostál. A new modified version of SMALBE-M is then presented which shortens the termination phase of SMALBE-M.

Chapter 4 (TFETI DDM) introduces TFETI for linear problems, extension to contact problems, and its reformulation by means of QP transforms.

Chapter 5 (Implicit orhonormalization) is devoted to the special topic of making equality constraints orthonormal "for free", a simple yet powerful new ingredient for massively parallel TFETI for contact problems, firstly presented in this thesis.

Part II (Implementation) contains topics directly connected with the practical implementation of the concepts of Part I in PERMON.

Chapter 6 (Open source software) presents an overview of pre-existing open source libraries, related by focus to our own PERMON software or even directly used by PERMON.

Chapter 7 (PERMON toolbox) introduces in detail the PERMON libraries relevant for solution of contact problems which form a practical part of the thesis.

Chapter 8 (Numerical experiments with PERMON) concludes Part II by description and evaluation of several numerical experiments with PERMON.

Part I

Background

CHAPTER I

Quadratic programming

Most engineering problems may be described by PDEs. To be solved with computers, they have to be discretized, e.g. with FEM. By these means, we typically get large sparse linear systems of equations. However, constrained problems describable by elliptic variational inequalities such as contact problems of mechanics, describing the equilibrium of elastic bodies in mutual contact, lead naturally to more general *quadratic programming* (QP) problems which will be defined in this chapter. QP problems also arise in other disciplines like least-squares regression, data fitting, data mining, support vector machines, control systems, and many others.

This chapter briefly introduces the field of mathematical optimization in Section 1.1 and gets quickly to its QP subfield in Section 1.2. Section 1.4 presents Karush–Kuhn–Tucker (KKT) optimality conditions and several useful theorems concerning existence and uniqueness of solution. Facts in Sections 1.1, 1.2 and 1.4 are generally known and loosely taken from [10]. Section 1.3 introduces zero-sized matrices so that special cases of QP are handled uniformly in Section 1.4. Section 1.5 formulates partially bound QP in order to simplify prescription of box constraints only on a subset of variables, and shows how the optimality conditions are modified. Sections 1.6 and 1.7 present original discussions on evaluation of KKT conditions and computation of Lagrange multipliers, respectively.

1.1 General discrete optimisation problems

A general discrete optimization problem reads

$$\text{find} \qquad \bar{\mathbf{x}} = \arg\min f(\mathbf{x}) \tag{1.1a}$$

$$\text{subject to} \quad h_E(\mathbf{x}) = \mathbf{o}, \tag{1.1b}$$

$$h_I(\mathbf{x}) \leq \mathbf{o}. \tag{1.1c}$$

Note the 'less than or equal' sign ($\leq$) is meant *component-wise* here and further on. The function f in (1.1a) is called an *objective* function (aka *cost*, *loss* function), h_E in (1.1b) is the *equality constraint function*, and h_I in (1.1c) is the *inequality constraint function*. The variable vector $\mathbf{x} \in \mathbb{R}^n$ is called the *optimization variable* of the problem. The *feasible set* Ω consists of all *feasible vectors*, i.e. the ones satisfying the prescribed constraints (1.1),

$$\Omega = \{\mathbf{x} \in \mathbb{R}^n : h_E(\mathbf{x}) = \mathbf{o} \wedge h_I(\mathbf{x}) \leq \mathbf{o}\}.$$

The vector $\bar{\mathbf{x}}$ is called a *solution* of the problem (1.1), if it yields the smallest *objective value* $f(\bar{\mathbf{x}})$ among all feasible vectors,

$$\bar{\mathbf{x}}: \quad f(\bar{\mathbf{x}}) = \min\{f(\bar{\mathbf{x}}) : \mathbf{x} \in \Omega\}.$$

1.2 QP problems

A *quadratic programming problem* or *quadratic program* (QP) can be seen as a generalization of a problem of solving an SPS linear system, where additional *equality constraints* and/or *inequality constraints* may be prescribed that must be satisfied by the solution vector – the solution must be feasible. Apparently, it may happen that no feasible vector solves the linear system on its own, and we instead search its nearest (least-square) solution from the feasible set.

QP is at the same time a special case of the general optimization problem (1.1) where the functions f, h_E, h_I are specified as follows:

- f is a quadratic convex function of the form

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x},$$

  where $\mathbf{A} \in \mathbb{R}^{n\times n}$ is a symmetric *Hessian* matrix of the cost function f, and $\mathbf{b} \in \mathbb{R}^n$ is the *objective right hand side* (RHS),

- h_E is an affine function of the form

$$h_E(\mathbf{x}) = \mathbf{B}_E\mathbf{x} - \mathbf{c}_E,$$

where $\mathbf{B}_E \in \mathbb{R}^{m_E \times n}$ is the *equality constraint matrix* (EM), and $\mathbf{c}_E \in \mathbb{R}^{m_E}$ is the *equality constraint right hand side* (ERHS),

- h_I is an affine function of the form

$$h_I(\mathbf{x}) = \mathbf{B}_I\mathbf{x} - \mathbf{c}_I,$$

where $\mathbf{B}_I \in \mathbb{R}^{m_I \times n}$ is the *inequality constraint matrix* (IM), and $\mathbf{c}_E \in \mathbb{R}^{m_I}$ is the *inequality constraint right hand side* (IRHS).

From here on, we will use the abbreviation QP for both quadratic programming (optimization discipline) and quadratic programs (optimization problems).

An important special case of the inequality constraints $\mathbf{B}_I\mathbf{x} \leq \mathbf{c}_I$ are *box constraints*

$$\boldsymbol{\ell} \leq \mathbf{x} \leq \mathbf{u}, \quad \boldsymbol{\ell} \in (\{-\infty\} \cup \mathbb{R})^n, \quad \mathbf{u} \in (\mathbb{R} \cup \{\infty\})^n.$$

The vectors $\boldsymbol{\ell}$ and $\mathbf{u}$ are called a *lower bound* (LB) and *upper bound* (UB), respectively. If only $\boldsymbol{\ell}$ is prescribed, we use the term *lower bound constraints* or just *bound constraints*. They can be rewritten in terms of the *lower bound constraint function* h_ℓ and the *upper bound constraint function* h_u,

$$h_\ell(\mathbf{x}) = \boldsymbol{\ell} - \mathbf{x} \leq \mathbf{o} \tag{1.2}$$

$$h_u(\mathbf{x}) = \mathbf{x} - \mathbf{u} \leq \mathbf{o}. \tag{1.3}$$

The box constraints can be merged with the general inequality constraints:

$$\mathbf{B}_I\mathbf{x} \leq \mathbf{c}_I \wedge \boldsymbol{\ell} \leq \mathbf{x} \leq \mathbf{u} \;\Leftrightarrow\; \begin{bmatrix} \mathbf{B}_I\mathbf{x} \\ -\mathbf{x} \\ \mathbf{x} \end{bmatrix} \leq \begin{bmatrix} \mathbf{c}_I \\ -\boldsymbol{\ell} \\ \mathbf{u} \end{bmatrix} \;\Leftrightarrow\; \widetilde{\mathbf{B}}_I\mathbf{x} = \begin{bmatrix} \mathbf{B}_I \\ -\mathbf{I} \\ \mathbf{I} \end{bmatrix} \mathbf{x} \;\leq\; \begin{bmatrix} \mathbf{c}_I \\ -\boldsymbol{\ell} \\ \mathbf{u} \end{bmatrix} = \widetilde{\mathbf{c}}_I. \tag{1.4}$$

Nevertheless, they are usually implemented separately for convenience. Finally note that box constraints also belong to a class called *separable convex constraints*; more specifically, they are one-dimensional case of *spherical constraints* (see [14]).

To sum up, the general optimization problem (1.1) can be in a special case of QP rewritten as

$$\text{find} \qquad \bar{\mathbf{x}} = \arg\min_{\mathbf{x}} \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x} \tag{1.5a}$$

$$\text{subject to} \quad \mathbf{B}_E\mathbf{x} = \mathbf{c}_E, \tag{1.5b}$$

$$\mathbf{B}_I\mathbf{x} \leq \mathbf{c}_I, \tag{1.5c}$$

$$\boldsymbol{\ell} \leq \mathbf{x} \leq \mathbf{u}. \tag{1.5d}$$

The conditions (1.5b)–(1.5d) may be expressed inline in terms of the *feasibility set*

$$\mathbf{x} \in \Omega = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{B}_E\mathbf{x} = \mathbf{c}_E \wedge \mathbf{B}_I\mathbf{x} \leq \mathbf{c}_I \wedge \boldsymbol{\ell} \leq \mathbf{x} \leq \mathbf{u}\}\,. \tag{1.6}$$

This is the type of problems this work deals with. We will implicitly assume that $\mathbf{A}$ is SPS, $m = m_E + m_I < n$, and that both $\mathbf{B}_E$ and $\mathbf{B}_I$ have full row rank.

1.3 Handling unprescribed constraints

It is convenient to presume that all types of constraints (1.5b)–(1.5d) are always prescribed. Absence of the lower bound $\boldsymbol{\ell}$ or the upper bound $\mathbf{u}$ in (1.5d) can be replaced by having $\boldsymbol{\ell}$ or $\mathbf{u}$ with all entries equal to $-\infty$ or ∞, respectively.

The situation when the linear constraints (1.5b) or (1.5c) are not present may be modelled conveniently by allowing $\mathbf{B}_E$ or $\mathbf{B}_I$ having the number of rows equal to zero, respectively. We will call such generalized matrices *empty matrices*.

Empty matrices help dealing with maps involving the zero vector space. We will denote an empty matrix as $\square^{(m,n)}$ where m and n, $mn = 0$, is the number of rows and columns, respectively. The size subscript may be omitted ($\square$) in case the size of the empty matrix is not important or is obvious from context. Matrix-matrix multiplication with empty matrices is defined in the following way. Let $m, n \geq 0$, $\mathbf{A} \in \mathbb{R}^{m\times n}$ be an arbitrary matrix, $\mathbf{O}^{(m,n)} \in \mathbb{R}^{m\times n}$ be a zero matrix, and $\alpha, \beta \in \mathbb{R}$. Then following operations are made valid

$$\mathbf{A}\square^{(n,0)} = \square^{(m,0)}, \qquad \square^{(0,m)}\mathbf{A} = \square^{(0,n)}, \tag{1.7}$$

$$\square^{(0,n)}\square^{(n,0)} = \square^{(0,0)}, \qquad \square^{(m,0)}\square^{(0,n)} = \mathbf{O}^{(m,n)}, \tag{1.8}$$

$$\alpha\square^{(m,n)} + \beta\square^{(m,n)} = \square^{(m,n)}, \tag{1.9}$$

A vector is considered here a special case of a matrix with the size of at least one dimension equal to one. Finally, let us define another few properties:

$$\square^{(m,n)} = \square^{(q,s)} \quad \Leftrightarrow \quad m = q \wedge n = s, \tag{1.10}$$

$$(\square^{(0,0)})^{-1} = \square^{(0,0)}, \tag{1.11}$$

$$||\square|| = 0, \qquad \det\square = 1, \tag{1.12}$$

$$\operatorname{Ker}\square^{(0,n)} = \mathbb{R}^n, \qquad \operatorname{Im}\square^{(0,n)} = \{\mathbf{o}^{(n)}\}. \tag{1.13}$$

1.4 Optimality conditions, solution existence and uniqueness

Here we remind a result that gives us necessary and sufficient conditions to be met by the QP solution, two theorems concerning existence of solution for particular QP classes, and finally an

existence and uniqueness theorem for general QP. These results are taken from [10] where also their proofs can be found.

A useful tool for the analysis of QP problems is the *Lagrangian* function $L : \mathbb{R}^{n+m} \to \mathbb{R}, m = m_E + m_I + 2n$, defined by

$$\begin{aligned} &L(\mathbf{x}, \boldsymbol{\lambda}_E, \boldsymbol{\lambda}_I, \boldsymbol{\lambda}_\ell, \boldsymbol{\lambda}_u) = \\ &= f(\mathbf{x}) \qquad\qquad + \boldsymbol{\lambda}_E^T h_E(\mathbf{x}) \qquad + \boldsymbol{\lambda}_I^T h_I(\mathbf{x}) \qquad + \boldsymbol{\lambda}_\ell^T h_\ell(\mathbf{x}) \quad + \boldsymbol{\lambda}_u^T h_u(\mathbf{x}) = \\ &= \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x} + \boldsymbol{\lambda}_E^T(\mathbf{B}_E\mathbf{x} - \mathbf{c}_E) + \boldsymbol{\lambda}_I^T(\mathbf{B}_I\mathbf{x} - \mathbf{c}_I) + \boldsymbol{\lambda}_\ell^T(\boldsymbol{\ell} - \mathbf{x}) + \boldsymbol{\lambda}_u^T(\mathbf{x} - \mathbf{u}). \end{aligned} \tag{1.14}$$

The vectors $\boldsymbol{\lambda}_E \in \mathbb{R}^{m_E}$, $\boldsymbol{\lambda}_I \in \mathbb{R}^{m_I}$, $\boldsymbol{\lambda}_\ell \in \mathbb{R}^n$, and $\boldsymbol{\lambda}_u \in \mathbb{R}^n$ are called *Lagrange multipliers* or just *multipliers* corresponding to equality, inequality, lower bound, and upper bound constraints, respectively. The constraints and multipliers may be concatenated to simplify notation, $h : \mathbb{R}^n \mapsto \mathbb{R}^m$, $h(\mathbf{x}) = \left[h_E^T(\mathbf{x}), h_I^T(\mathbf{x}), h_\ell^T(\mathbf{x}), h_u^T(\mathbf{x})\right]^T$, $\boldsymbol{\lambda} = \left[\boldsymbol{\lambda}_E^T, \boldsymbol{\lambda}_I^T, \boldsymbol{\lambda}_\ell^T, \boldsymbol{\lambda}_u^T\right]^T \in \mathbb{R}^m$. Lagrangian may be rewritten as $L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T h(\mathbf{x})$.

Theorem 1.1 *Let* $\mathbf{A}$, $\mathbf{b}$, $\mathbf{B}_E$, $\mathbf{c}_E$, $\mathbf{B}_I$, $\mathbf{c}_I$, $\boldsymbol{\ell}$, $\mathbf{u}$ *be the data defining the problem* (1.5)*, admitting rank-deficient* $\mathbf{B}_E$ *and* $\mathbf{B}_I$*. The vector* $\bar{\mathbf{x}}$ *is the solution of the problem* (1.5) *if and only if there exist Lagrange multipliers* $\bar{\boldsymbol{\lambda}}$ *such that*

$$\nabla_{\mathbf{x}}\ L(\bar{\mathbf{x}}, \bar{\boldsymbol{\lambda}}) = \mathbf{A}\bar{\mathbf{x}} - \mathbf{b} \quad + \quad \mathbf{B}_E^T\bar{\boldsymbol{\lambda}}_E \quad + \quad \mathbf{B}_I^T\bar{\boldsymbol{\lambda}}_I \quad - \quad \bar{\boldsymbol{\lambda}}_\ell \quad + \quad \bar{\boldsymbol{\lambda}}_u \quad = \mathbf{o}, \tag{1.15a}$$

$$\nabla_{\boldsymbol{\lambda}_E} L(\bar{\mathbf{x}}, \bar{\boldsymbol{\lambda}}) = \mathbf{B}_E\bar{\mathbf{x}} - \mathbf{c}_E = \mathbf{o}, \tag{1.15b}$$

$$\nabla_{\boldsymbol{\lambda}_I}\ L(\bar{\mathbf{x}}, \bar{\boldsymbol{\lambda}}) = \mathbf{B}_I\bar{\mathbf{x}} - \mathbf{c}_I \ \le \mathbf{o}, \qquad \bar{\boldsymbol{\lambda}}_I \ge \mathbf{o}, \qquad \bar{\boldsymbol{\lambda}}_I^T(\mathbf{B}_I\bar{\mathbf{x}} - \mathbf{c}_I) = 0, \tag{1.15c}$$

$$\nabla_{\boldsymbol{\lambda}_\ell}\ L(\bar{\mathbf{x}}, \bar{\boldsymbol{\lambda}}) = \boldsymbol{\ell} - \bar{\mathbf{x}} \quad \le \mathbf{o}, \qquad \bar{\boldsymbol{\lambda}}_\ell \ge \mathbf{o}, \qquad \bar{\boldsymbol{\lambda}}_\ell^T(\boldsymbol{\ell} - \bar{\mathbf{x}}) \quad = 0, \tag{1.15d}$$

$$\nabla_{\boldsymbol{\lambda}_u}\ L(\bar{\mathbf{x}}, \bar{\boldsymbol{\lambda}}) = \bar{\mathbf{x}} - \mathbf{u} \quad \le \mathbf{o}, \qquad \bar{\boldsymbol{\lambda}}_u \ge \mathbf{o}, \qquad \bar{\boldsymbol{\lambda}}_u^T(\bar{\mathbf{x}} - \mathbf{u}) \quad = 0. \tag{1.15e}$$

Note that thanks to introducing zero-sized matrices in Section 1.3, conditions corresponding to unprescribed constraints trivially hold. If the linear equality constraints are not prescribed, $m_E = 0$ and $\boldsymbol{\lambda}_E$ is an empty vector. Likewise, in case of the linear inequality constraints not being prescribed, $m_I = 0$ and $\boldsymbol{\lambda}_I$ is an empty vector. If the lower bounds $\boldsymbol{\ell}$ or the upper bounds $\mathbf{u}$ are not prescribed, $\boldsymbol{\lambda}_\ell$ or $\boldsymbol{\lambda}_u$ are zero vectors from $\mathbb{R}^n$, respectively.

The conditions (1.15) are called *Karush–Kuhn–Tucker (KKT) conditions* for the solution of the QP (1.5). If $\bar{\mathbf{x}}$ and $\bar{\boldsymbol{\lambda}}$ satisfy (1.15), then $(\bar{\mathbf{x}}, \bar{\boldsymbol{\lambda}})$ is called a *KKT pair* of the QP (1.5). The first KKT condition (1.15a) is called *stationarity condition*. In (1.15b)–(1.15e), the first column contains *primal feasibility conditions* which can be expressed also as $\mathbf{x} \in \Omega$, second *dual feasibility conditions*, and third *complementary slackness conditions*.

Next two theorems formulate conditions of existence of the solution for particular subclasses of the general QP (1.5). They are taken from propositions 2.1 and 2.10 in [10].

Theorem 1.2 *Let us assume an unconstrained problem* (1.5)*, i.e. with none of constraints* (1.5b)–(1.5d) *prescribed. Then this problem has a solution* $\bar{\mathbf{x}}$ *if and only if* $\mathbf{b} \in Im\,\mathbf{A}$*. Moreover, the problem has a unique solution* $\bar{\mathbf{x}}$ *if and only if* $\mathbf{A}$ *is SPD.*

Theorem 1.3 *Assume an equality constrained problem* (1.5)*, i.e. with no inequality constraints* (1.5c)–(1.5d) *presribed. Let* $\mathbf{R}$ *be a matrix such that* $Im\,\mathbf{R} = Ker\,\mathbf{A}$ *and let* $\mathbf{A}|Ker\,\mathbf{B}_E$ *be SPS. Then the problem has a solution* $\bar{\mathbf{x}}$ *if and only if* $\mathbf{R}^T\mathbf{b} \in Im(\mathbf{R}^T\mathbf{B}_E^T)$.

The last theorem presents a sufficient condition for existence and uniqueness of the solution of QP problem (1.5). In [10], it is a part of propositions 2.1, 2.10, 2.16 and 2.20 for particular QP subclasses.

Theorem 1.4 *Assume a problem* (1.5) *with any or none of constraints* (1.5b)–(1.5d) *prescribed. If equality constraints* (1.5b) *are prescribed, let* $\mathbf{A}|Ker\,\mathbf{B}_E$ *be SPD, else let* $\mathbf{A}$ *be SPD. Then the problem* (1.5) *has a unique solution* $\bar{\mathbf{x}}$.

1.5 Partially bound constrained problems

It is common that the box constraints with finite values are prescribed only on an index subset $\mathcal{I} \subset (1, \dots n)$. The size of this subset may be much lower than n. Hence, it can be computationally advantageous to work only with this subset rather than deal with the whole set of indices. Moreover, implementation of infinite values is specific for a machine and language; sometimes it can be modelled only by using "sufficiently large" floating point values. Finally, sometimes it is more convenient also in mathematical considerations to refer to the subset $\mathcal{I}$ explicitly, and almost inevitable if we generalize box constraints to separable convex constraints [14].

These reasons lead us to introduce an alternative form of the problem (1.5)

$$\begin{aligned} &\text{find} && \bar{\mathbf{x}} = \arg\min_{\mathbf{x}} \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x} && \text{(1.16a)}\\ &\text{subject to} && \mathbf{B}_E\mathbf{x} = \mathbf{c}_E, && \text{(1.16b)}\\ &&& \mathbf{B}_I\mathbf{x} \le \mathbf{c}_I, && \text{(1.16c)}\\ &&& \boldsymbol{\ell}_{\mathcal{I}} \le \mathbf{x}_{\mathcal{I}} \le \mathbf{u}_{\mathcal{I}}, && \text{(1.16d)} \end{aligned}$$

where $\mathbf{v}_{\mathcal{I}}$ means the restriction of the vector $\mathbf{v}$ to the indices $\mathcal{I}$, i.e.

$$[\mathbf{v}_{\mathcal{I}}]_i = [\mathbf{v}]_{\mathcal{I}_i}, i = 1, \dots, |\mathcal{I}|.$$

We can without loss of generality assume that $\mathcal{I} = (1, \dots, p)$ for a given upper index p, because the general case can be expressed using a proper index permutation.

The Lagrangian (1.14) and KKT conditions (1.15) in Theorem 1.1 then read

$$\begin{aligned} L : \mathbb{R}^{n+m} \to \mathbb{R}, \quad & m = m_E + m_I + 2|\mathcal{I}|,\\ L(\mathbf{x}, \boldsymbol{\lambda}_E, \boldsymbol{\lambda}_I, \boldsymbol{\lambda}_{\ell\mathcal{I}}, \boldsymbol{\lambda}_{u\mathcal{I}}) = & \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x} + \boldsymbol{\lambda}_E^T(\mathbf{B}_E\mathbf{x} - \mathbf{c}_E)\\ & + \boldsymbol{\lambda}_I^T(\mathbf{B}_I\mathbf{x} - \mathbf{c}_I) + \boldsymbol{\lambda}_{\ell\mathcal{I}}^T(\boldsymbol{\ell}_{\mathcal{I}} - \mathbf{x}_{\mathcal{I}}) + \boldsymbol{\lambda}_{u\mathcal{I}}^T(\mathbf{x}_{\mathcal{I}} - \mathbf{u}_{\mathcal{I}}). \end{aligned} \tag{1.17a}$$

$$\nabla_{\mathbf{x}}\ L(\bar{\mathbf{x}},\bar{\boldsymbol{\lambda}}) = \mathbf{A}\bar{\mathbf{x}} - \mathbf{b} + \mathbf{B}_E^T\bar{\boldsymbol{\lambda}}_E + \mathbf{B}_I^T\bar{\boldsymbol{\lambda}}_I + \begin{bmatrix} -\mathbf{I}^{(p,p)} \\ \mathbf{O}^{(n-p,p)} \end{bmatrix} \bar{\boldsymbol{\lambda}}_{\ell\mathcal{I}} + \begin{bmatrix} \mathbf{I}^{(p,p)} \\ \mathbf{O}^{(n-p,p)} \end{bmatrix} \bar{\boldsymbol{\lambda}}_{u\mathcal{I}} = \mathbf{o}, \tag{1.17b}$$

$$\nabla_{\boldsymbol{\lambda}_E} L(\bar{\mathbf{x}},\bar{\boldsymbol{\lambda}}) = \mathbf{B}_E\bar{\mathbf{x}} - \mathbf{c}_E = \mathbf{o}, \tag{1.17c}$$

$$\nabla_{\boldsymbol{\lambda}_I}\ L(\bar{\mathbf{x}},\bar{\boldsymbol{\lambda}}) = \mathbf{B}_I\bar{\mathbf{x}} - \mathbf{c}_I\ \leq \mathbf{o}, \qquad \bar{\boldsymbol{\lambda}}_I \geq \mathbf{o}, \qquad \bar{\boldsymbol{\lambda}}_I^T(\mathbf{B}_I\bar{\mathbf{x}} - \mathbf{c}_I)\ = 0, \tag{1.17d}$$

$$\nabla_{\boldsymbol{\lambda}_\ell}\ L(\bar{\mathbf{x}},\bar{\boldsymbol{\lambda}}) = \boldsymbol{\ell}_{\mathcal{I}} - \bar{\mathbf{x}}_{\mathcal{I}}\ \leq \mathbf{o}, \qquad \bar{\boldsymbol{\lambda}}_{\ell\mathcal{I}} \geq \mathbf{o}, \qquad \bar{\boldsymbol{\lambda}}_{\ell\mathcal{I}}^T(\boldsymbol{\ell}_{\mathcal{I}} - \bar{\mathbf{x}}_{\mathcal{I}})\ = 0, \tag{1.17e}$$

$$\nabla_{\boldsymbol{\lambda}_u}\ L(\bar{\mathbf{x}},\bar{\boldsymbol{\lambda}}) = \bar{\mathbf{x}}_{\mathcal{I}} - \mathbf{u}_{\mathcal{I}}\ \leq \mathbf{o}, \qquad \bar{\boldsymbol{\lambda}}_{u\mathcal{I}} \geq \mathbf{o}, \qquad \bar{\boldsymbol{\lambda}}_{u\mathcal{I}}^T(\bar{\mathbf{x}}_{\mathcal{I}} - \mathbf{u}_{\mathcal{I}})\ = 0. \tag{1.17f}$$

Notice that it obviously suffices to store only $\boldsymbol{\ell}_{\mathcal{I}}$, $\mathbf{u}_{\mathcal{I}}$, $\boldsymbol{\lambda}_{\ell\mathcal{I}}$, $\boldsymbol{\lambda}_{u\mathcal{I}}$ in an implementation because the rest of $\boldsymbol{\ell}$, $\mathbf{u}$, $\boldsymbol{\lambda}_\ell$, $\boldsymbol{\lambda}_u$ is in fact populated with constant values $-\infty$, ∞, 0, 0, respectively. Hence, we will further write $\boldsymbol{\ell}$, $\mathbf{u}$, $\boldsymbol{\lambda}_\ell$, $\boldsymbol{\lambda}_u$ instead of $\boldsymbol{\ell}_{\mathcal{I}}$, $\mathbf{u}_{\mathcal{I}}$, $\boldsymbol{\lambda}_{\ell\mathcal{I}}$, $\boldsymbol{\lambda}_{u\mathcal{I}}$.

1.6 KKT conditions practically

There are good reasons to compute not only the solution $\bar{\mathbf{x}}$ but also the corresponding multipliers $\bar{\boldsymbol{\lambda}}$ as they can have some physical meaning. But even in case we do not need them in our simulation, they are an important tool for the "solver self-check", giving us information whether an error of the solution found by the solver is below the given tolerance. As stated in Theorem 1.1, if we show there are some multipliers $\bar{\boldsymbol{\lambda}}$ such that $(\bar{\mathbf{x}},\bar{\boldsymbol{\lambda}})$ is a KKT pair of the given QP, we can be sure $\bar{\mathbf{x}}$ really solves this QP. Hence, if the results are erroneous but there is $\bar{\boldsymbol{\lambda}}$ meeting the KKT conditions (1.17) within the prescribed tolerance, we can be suspicious rather of the procedure which generates the QP data then the QP solver itself.

In case of unconstrained minimization, all multipliers are zero vectors, and only the stationarity condition remains (1.17b) in the form

$$\bar{\mathbf{g}} = \mathbf{A}\bar{\mathbf{x}} - \mathbf{b} = \mathbf{o}, \tag{1.18}$$

where $\mathbf{g} = \nabla_{\mathbf{x}} L(\mathbf{x},\boldsymbol{\lambda}) = \nabla_{\mathbf{x}} f(\mathbf{x})$ is the gradient of the objective function. With iterative methods and limited accuracy of computer arithmetic, we almost never get the vector of exact zeros. We rather interpret $\mathbf{g}$ as the opposite of the residual $\mathbf{r} = \mathbf{b} - \mathbf{A}\bar{\mathbf{x}}$ of the given approximation of $\bar{\mathbf{x}}$, and check that the residual norm $||\mathbf{r}|| = ||\mathbf{g}|| = ||\mathbf{A}\mathbf{x} - \mathbf{b}||$ is sufficiently small, i.e. lower than the demanded tolerance. It is actually better to evaluate the relative residual norm $||\mathbf{g}||/||\mathbf{b}||$ which takes into account the order of magnitude of the input data. The gradient $\mathbf{g}$ is relatively easy to compute and it is additionally often needed in the iterative method itself. Thus the typical stopping criterion reads

$$||\mathbf{g}||/||\mathbf{b}|| < \varepsilon, \tag{1.19}$$

where ε is the demanded tolerance, e.g. 10^{-6}. We have got an approximated form of the stationarity condition (1.18) and we can express *quantitatively* to what extent is this original condition met.

In case of constrained minimization, some of the multipliers are non-zero and the topic becomes more interesting. In order to avoid computation of the Lagrange multipliers and subsequent evaluation of the KKT conditions in each iteration, which may be relatively costly, alternative stopping criteria are developed. For example, the norm of the projected gradient $P(\mathbf{g}) \in \Omega$ is checked, where P is a projection to the feasible set Ω. However, the projection itself is a possible source of error – it may even be an optimization problem on its own. This leads us to evaluate the KKT satisfaction at least at the end of the solution phase for validation of the produced result.

Similarly to the unconstrained case, we again need the KKT conditions to be expressed in a form suitable for quantitative evaluation. The gradient of the Lagrangian is more complicated,

$$\mathbf{g} = \nabla_{\mathbf{x}} L(\bar{\mathbf{x}}, \bar{\boldsymbol{\lambda}}) = \mathbf{A}\bar{\mathbf{x}} - \mathbf{b} + \mathbf{B}_E^T \bar{\boldsymbol{\lambda}}_E + \mathbf{B}_I^T \bar{\boldsymbol{\lambda}}_I + \begin{bmatrix} -\mathbf{I}^{(p,p)} \\ \mathbf{O}^{(n-p,p)} \end{bmatrix} \bar{\boldsymbol{\lambda}}_\ell + \begin{bmatrix} \mathbf{I}^{(p,p)} \\ \mathbf{O}^{(n-p,p)} \end{bmatrix} \bar{\boldsymbol{\lambda}}_u,$$

but the approximate stationarity condition (1.17b) again reads

$$||\mathbf{g}||/||\mathbf{b}|| < \varepsilon. \tag{1.20}$$

The KKT conditions related to inequality constraints (1.17d) can be reformulated as

$$\mathbf{B}_I\bar{\mathbf{x}} - \mathbf{c}_I \leq \mathbf{o} \quad \longrightarrow \quad ||\max(\mathbf{B}_I\mathbf{x} - \mathbf{c}_I, \mathbf{o})|| < \varepsilon \tag{1.21}$$

$$\bar{\boldsymbol{\lambda}}_I \geq \mathbf{o} \quad \longrightarrow \quad ||\min(\boldsymbol{\lambda}_I, \mathbf{o})|| < \varepsilon \tag{1.22}$$

$$\bar{\boldsymbol{\lambda}}_I^T(\mathbf{B}_I\bar{\mathbf{x}} - \mathbf{c}_I) = 0 \quad \longrightarrow \quad |\boldsymbol{\lambda}_I^T(\mathbf{B}_I\mathbf{x} - \mathbf{c}_I)| < \varepsilon. \tag{1.23}$$

where max(.,.) denotes the element-wise maximum of two vectors (see Symbols). Conditions for lower bounds (1.17e) and upper bounds (1.17f) are proceeded analogously.

1.7 Computation of Lagrange multipliers

Let us now discuss how the Lagrange multipliers can be computed for certain special configurations of constraints. We discuss here only few cases which are useful further.

1.7.1 Computing $\boldsymbol{\lambda}_\ell$ of bound and equality constrained problems

Let us first focus on bound and equality constrained problems, i.e. with $(\mathbf{B}_I, \mathbf{c}_I) = (\square, \square)$ and $\mathbf{u} = \boldsymbol{\infty}^{(|\mathcal{I}|,1)}$. In this case, from the KKT conditions (1.17) only (1.17b) and (1.17e) are effective

as $\bar{\boldsymbol{\lambda}}_I = \bar{\boldsymbol{\lambda}}_{u\mathcal{I}} = \mathbf{o}$. This type of QP commonly arises by applying dualization, discussed later. Let us denote

$$\mathbf{g} = \nabla_{\mathbf{x}} L_E(\mathbf{x}, \boldsymbol{\lambda}_E) = \mathbf{A}\mathbf{x} - \mathbf{b} + \mathbf{B}_E^T \boldsymbol{\lambda}_E. \tag{1.24}$$

The stationarity condition (1.17b) reads

$$\nabla_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}) = \mathbf{g} - \begin{bmatrix} \mathbf{I}^{(p,p)} \\ \mathbf{O}^{(n-p,p)} \end{bmatrix} \boldsymbol{\lambda}_\ell = \mathbf{o}. \tag{1.25}$$

Hence, the KKT conditions (1.17) can be simplified into the form

$$\mathbf{g}_{\mathcal{I}} \geq \mathbf{o}, \quad \mathbf{g}_{\mathcal{R}} = \mathbf{o}, \quad \text{and} \quad \mathbf{g}_{\mathcal{I}}^T (\mathbf{x}_{\mathcal{I}} - \boldsymbol{\ell}) = 0, \tag{1.26}$$

where $\mathcal{R} = (p+1, \ldots, n)$. Moreover, for the optimal $\bar{\mathbf{g}}$ satisfying (1.26), it holds that

$$\bar{\boldsymbol{\lambda}}_\ell = \bar{\mathbf{g}}_{\mathcal{I}}. \tag{1.27}$$

Notice that the KKT conditions (1.17) can be conveniently expressed solely by means of the gradient of the Lagrangian for equality constrained problems, and the bound constraint multiplier $\bar{\boldsymbol{\lambda}}_\ell$ is fully determined by the pair $(\bar{\mathbf{x}}, \bar{\boldsymbol{\lambda}}_E)$. Thus we can call this pair the *KKT pair of the bound and equality constrained problem*. Nevertheless, it must not be confused with the KKT pair of the corresponding equality constrained problem, arisen by removing the bound constraints.

Note also that we allow here purely bound constrained problems with $(\mathbf{B}_E, \mathbf{c}_E) = (\square, \square)$, for which $\boldsymbol{\lambda}_E = \square$ and the term $\mathbf{B}_E^T \boldsymbol{\lambda}_E$ vanishes.

1.7.2 Computing $\boldsymbol{\lambda}_\ell$ and $\boldsymbol{\lambda}_u$ of box and equality constrained QPs

In this case $\bar{\boldsymbol{\lambda}}_I = \mathbf{o}$, so (1.17d) trivially holds. Using $\mathbf{g}$ from (1.24), the stationarity condition (1.17b) reads

$$\nabla_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}) = \mathbf{g} - \begin{bmatrix} \mathbf{I}^{(p,p)} \\ \mathbf{O}^{(n-p,p)} \end{bmatrix} \boldsymbol{\lambda}_\ell + \begin{bmatrix} \mathbf{I}^{(p,p)} \\ \mathbf{O}^{(n-p,p)} \end{bmatrix} \boldsymbol{\lambda}_u = \mathbf{o}. \tag{1.28}$$

$$\tag{1.29}$$

From here and the dual feasibility conditions in the middle column of (1.17e) and (1.17f), we get

$$\boldsymbol{\lambda}_u \geq \mathbf{o} \quad \wedge \quad \boldsymbol{\lambda}_\ell = \mathbf{g}_{\mathcal{I}} + \boldsymbol{\lambda}_u \geq \mathbf{o}. \tag{1.30}$$

Thus there are infinitely many choices of $\boldsymbol{\lambda}_u$ and $\boldsymbol{\lambda}_\ell$, and they take the form

$$\boldsymbol{\lambda}_u \geq \max(-\mathbf{g}_{\mathcal{I}}, \mathbf{o}), \quad \boldsymbol{\lambda}_\ell = \mathbf{g}_{\mathcal{I}} + \boldsymbol{\lambda}_u.$$

However, there is arguably no point in choosing other then the minimal option

$$\boldsymbol{\lambda}_u = \max(-\mathbf{g}_{\mathcal{I}}, \mathbf{o}), \quad \boldsymbol{\lambda}_\ell = \max(\mathbf{o}, \mathbf{g}_{\mathcal{I}}).$$

Indeed, it contains many zero entries simplifying evaluation of the complementarity conditions in the last column of (1.17e) and (1.17f), and if the conditions are not met, then adding some non-negative vector to $\boldsymbol{\lambda}_u$ cannot fix that.

Again, the discussion above holds also for purely box constrained problems. Note also that for $\mathbf{u} = \boldsymbol{\infty}$ we get

$$\boldsymbol{\lambda}_u = \max(-\mathbf{g}_{\mathcal{I}}, \mathbf{o}) = \mathbf{o}, \quad \boldsymbol{\lambda}_\ell = \max(\mathbf{o}, \mathbf{g}_{\mathcal{I}}) = \mathbf{g}_{\mathcal{I}},$$

hence Section 1.7.1 discusses a special case.

1.7.3 Computing $\boldsymbol{\lambda}_E$

Assume QP with prescribed equality constraints, with or without linear inequality and/or box constraints, and that the solution $\bar{\mathbf{x}}$ and the optimal Lagrange multipliers $\bar{\boldsymbol{\lambda}}_I$, $\bar{\boldsymbol{\lambda}}_\ell$, $\bar{\boldsymbol{\lambda}}_u$, satisfying (1.15c)–(1.15e) are known. Let us seek $\boldsymbol{\lambda}_E$ such that the remaining KKT conditions (1.15a)–(1.15b) are also met.

Denoting

$$\mathbf{g} = \mathbf{A}\bar{\mathbf{x}} - \mathbf{b} + \mathbf{B}_I^T\bar{\boldsymbol{\lambda}}_I - \bar{\boldsymbol{\lambda}}_\ell + \bar{\boldsymbol{\lambda}}_u, \tag{1.31}$$

the stationarity condition (1.15a) can be equivalently expressed as an over-determined linear system

$$\mathbf{B}_E^T\boldsymbol{\lambda}_E = -\mathbf{g}, \tag{1.32}$$

and the optimal equality constraint multiplier $\bar{\boldsymbol{\lambda}}_E$ must solve this system.

Let us remind our implicit presumption that $\mathbf{B}_E$ has full row rank; in that case $\mathbf{B}_E^T$ has full column rank and the equation above has at most one solution. However, the least-square solution solving the corresponding normal equation

$$\mathbf{B}_E\mathbf{B}_E^T\boldsymbol{\lambda}_E = -\mathbf{B}_E\mathbf{g}, \tag{1.33}$$

exists always and can be explicitly expressed using the left inverse of $\mathbf{B}_E^T$ as

$$\widetilde{\boldsymbol{\lambda}}_E = -(\mathbf{B}_E\mathbf{B}_E^T)^{-1}\mathbf{B}_E\mathbf{g} = -(\mathbf{B}_E^T)^L\mathbf{g}.$$

If the classical solution exists, it equals the least-square solution. Moreover, the classical solution exists if and only if

$$\mathbf{g} \in \operatorname{Im}\mathbf{B}_E^T,$$

which is thanks to Theorem 1.1 automatically true if we presume optimality of $\bar{\mathbf{x}}$, $\bar{\boldsymbol{\lambda}}_I$, $\bar{\boldsymbol{\lambda}}_\ell$ and $\bar{\boldsymbol{\lambda}}_u$. Hence, we can express the sought $\bar{\boldsymbol{\lambda}}_E$ as

$$\bar{\boldsymbol{\lambda}}_E = -(\mathbf{B}_E^T)^L\mathbf{g}. \tag{1.34}$$

We can convince ourselves we have found the proper $\bar{\boldsymbol{\lambda}}_E$ by substituting it back into (1.32), yielding an equality

$$-\mathbf{B}_E^T(\mathbf{B}_E^T)^L\mathbf{g} = -\mathbf{g}.$$

Indeed, this equality holds as $\mathbf{B}_E^T(\mathbf{B}_E^T)^L$ is a projector onto $\operatorname{Im}\mathbf{B}_E^T$ and $\mathbf{g} \in \operatorname{Im}\mathbf{B}_E^T$.

CHAPTER II

QP transforms

This chapter deals with the novel notion of a *QP transform*. It can be described as a mapping reformulating an original QP into the new one, provided the solution of the new QP can be transformed back into the solution of the original one. Performing a QP transform is typically motivated by the fact that the derived QP is somehow simpler to solve – e.g. its Hessian has smaller dimension, is better conditioned, some constraints are eliminated, have more favourable structure etc. QP transforms turn out to be a useful tool which simplifies implementation of the QP solvers, and they form one of the key design concepts in the PermonQP library (Section 7.4).

Section 2.1 establishes a special "inline" notation of QP problems. The notion of QP transform is introduced in Section 2.2. Section 2.3 presents several particular QP tranforms. Section 2.4 shows sample uses of QP transforms.

2.1 Inline QP notation

Let us introduce a convenient notation of QP problems which makes description of QP transforms more clear. Particular instances of the generic QP problem scheme (1.16) will be labelled $\mathsf{QP}_k,\ k = 1, 2, \ldots$. We will describe the instance QP_1 with any of the constraints (1.16b)–(1.16d) prescribed by an ordered set of its defining objects,

$$\mathsf{QP}_1 = \mathsf{QP}[\ \mathbf{A}, \mathbf{b} \mid \mathbf{B}_E, \mathbf{c}_E \mid \mathbf{B}_I, \mathbf{c}_I \mid \boldsymbol{\ell}, \mathbf{u}, \mathcal{I}\ ].$$

This notation will be called the *signature* of the QP. For sake of simple notation, we introduce following rules for the last triple corresponding to box constraints. The last object $\mathcal{I}$ is optional

and denotes the index set on which the box constraints are prescribed. If $\mathcal{I}$ is not present, we assume $\mathcal{I} = (1, \ldots, n)$, i.e. the fully constrained problem (1.5). If only $\boldsymbol{\ell}$ is prescribed and $\mathbf{u}$ is not, we will write either $(\boldsymbol{\ell}, \square)$ or $(\boldsymbol{\ell}, \square, \mathcal{I})$, preserving the previous rule. If neither $\boldsymbol{\ell}$ nor $\mathbf{u}$ are prescribed, we will write $(\square, \square)$ and $\mathcal{I}$ is implicitly an empty index set.

The (primal) solution of the given QP will be denoted $\bar{\mathbf{x}}$, and the optimal Lagrange multipliers corresponding to the linear equality, linear inequality, lower bound, upper bound constraints will be marked as $\bar{\boldsymbol{\lambda}}_E$, $\bar{\boldsymbol{\lambda}}_I$, $\bar{\boldsymbol{\lambda}}_\ell$, $\bar{\boldsymbol{\lambda}}_u$, respectively. The *solution signature* of QP_1 reads

$$\overline{\mathsf{QP}}_1 = \overline{\mathsf{QP}}[\, \bar{\mathbf{x}} \mid \bar{\boldsymbol{\lambda}}_E \mid \bar{\boldsymbol{\lambda}}_I \mid \bar{\boldsymbol{\lambda}}_\ell, \bar{\boldsymbol{\lambda}}_u, \mathcal{I}\,].$$

We apply to the last triple the same rules as to $(\boldsymbol{\ell}, \mathbf{u}, \mathcal{I})$ in previous paragraph.

2.2 QP transform definition

A *QP transform* T is a mapping which maps the given QP problem QP_1 to the new QP problem QP_2,

$$T(\mathsf{QP}_1) = \mathsf{QP}_2.$$

The solution $\overline{\mathsf{QP}}_1$ of the original problem QP_1 must be recoverable from the solution $\overline{\mathsf{QP}}_2$ of the derived problem QP_2 by means of the corresponding *reconstruction function* $\bar{T}$ such that

$$\overline{\mathsf{QP}}_1 = \bar{T}(\overline{\mathsf{QP}}_2).$$

Figure 2.1 shows a graphical example of a QP transform and corresponding reconstruction function. Some important QP transforms are introduced in the following section.

2.3 Concrete QP transforms

This section presents several concrete QP transforms. The selection surely does not include many potentially useful transforms; it is motivated mainly by modular design of PERMON solvers (Chapter 7) for solving large-scale variational inequalities using FETI DDM (Chapter 4). Each of the subsections is devoted to one QP transform, starting with a synopsis of the transform, followed by a detailed discussion.

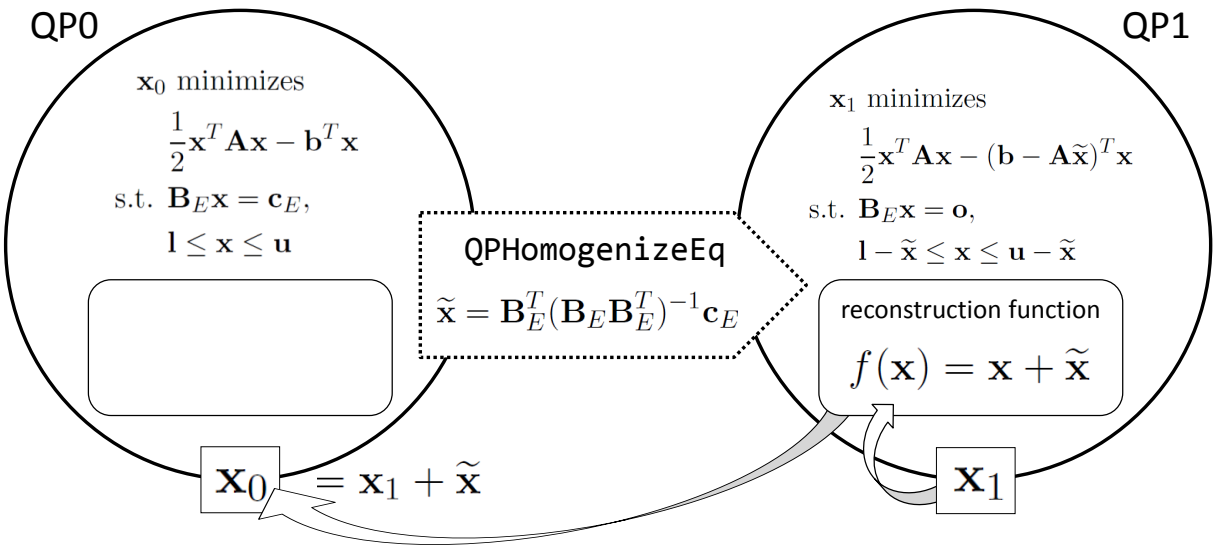


Figure 2.1: Example of a QP transform and reconstruction function – homogenization of the equality constraints (Section 2.3.3).

2.3.1 Box constraints elimination

Synopsis `EliminateBox` derives a QP without box constraints by embedding them into linear inequality constraints

$$\texttt{EliminateBox}(\mathsf{QP}_1) = \mathsf{QP}_2 \quad : \quad [\,\boldsymbol{\ell} \mid \mathbf{u}\,] \to [\,\square \mid \square\,]$$

$$\begin{aligned}
\mathsf{QP}_1 &= \mathsf{QP}[\ \mathbf{A}, \mathbf{b} \mid \mathbf{B}_E, \mathbf{c}_E \mid \mathbf{B}_I, \mathbf{c}_I \mid \boldsymbol{\ell}, \mathbf{u}, \mathcal{I} \quad] \\
\mathsf{QP}_2 &= \mathsf{QP}[\ \mathbf{A}, \mathbf{b} \mid \mathbf{B}_E, \mathbf{c}_E \mid \widetilde{\mathbf{B}}_I, \widetilde{\mathbf{c}}_I \mid \square, \square \quad] \qquad (2.1) \\
\overline{\mathsf{QP}}_2 &= \overline{\mathsf{QP}}[\ \bar{\mathbf{x}} \quad \mid \bar{\boldsymbol{\lambda}}_E \quad \mid \widetilde{\boldsymbol{\lambda}}_I \quad \mid \square, \square \quad] \\
\overline{\mathsf{QP}}_1 &= \overline{\mathsf{QP}}[\ \bar{\mathbf{x}} \quad \mid \bar{\boldsymbol{\lambda}}_E \quad \mid \bar{\boldsymbol{\lambda}}_I \quad \mid \bar{\boldsymbol{\lambda}}_\ell, \bar{\boldsymbol{\lambda}}_u, \mathcal{I}\,] \qquad (2.2)
\end{aligned}$$

$$\mathcal{I} = (1, \ldots, p), \quad 0 \le p \le n,$$

$$\widetilde{\mathbf{B}}_I = \begin{bmatrix} \mathbf{B}_I \\ -\mathbf{I}^{(p,p)} \quad \mathbf{O}^{(p,n-p)} \\ \mathbf{I}^{(p,p)} \quad \mathbf{O}^{(p,n-p)} \end{bmatrix}, \quad \widetilde{\mathbf{c}}_I = \begin{bmatrix} \mathbf{c}_I \\ -\boldsymbol{\ell}_\mathcal{I} \\ \mathbf{u}_\mathcal{I} \end{bmatrix}, \quad \widetilde{\boldsymbol{\lambda}}_I = \begin{bmatrix} \bar{\boldsymbol{\lambda}}_I \\ \bar{\boldsymbol{\lambda}}_{\ell\mathcal{I}} \\ \bar{\boldsymbol{\lambda}}_{u\mathcal{I}} \end{bmatrix} \qquad (2.3)$$

We start with this transform to show the reader a simple example of the QP transform concept. In presence of linear inequality constraints, it may be practical to eliminate box

constraints in order to simplify implementation of solvers. They can then assume that either box or linear inequality constraints are present. If both types are present, the box constraints will be merged with the linear inequality ones according to (2.3).

2.3.2 Affine space shift

Synopsis `Shift` shifts the feasible set along the given vector $\mathbf{s}$

$$\texttt{Shift}(\mathsf{QP}_1, \mathbf{s}) = \mathsf{QP}_2$$

$$\begin{array}{lllll} \mathsf{QP}_1 = \mathsf{QP}[\ \mathbf{A}, \mathbf{b} & |\ \mathbf{B}_E, \mathbf{c}_E & |\ \mathbf{B}_I, \mathbf{c}_I & |\ \boldsymbol{\ell}, \mathbf{u} &] \\ \mathsf{QP}_2 = \mathsf{QP}[\ \mathbf{A}, \mathbf{b} + \mathbf{A}\mathbf{s} & |\ \mathbf{B}_E, \mathbf{c}_E + \mathbf{B}_E\mathbf{s} & |\ \mathbf{B}_I, \mathbf{c}_I + \mathbf{B}_I\mathbf{s} & |\ \boldsymbol{\ell} + \mathbf{s}, \mathbf{u} + \mathbf{s} &] \qquad (2.4) \\ \overline{\mathsf{QP}}_2 = \overline{\mathsf{QP}}[\ \bar{\mathbf{x}} & |\ \bar{\boldsymbol{\lambda}}_E & |\ \bar{\boldsymbol{\lambda}}_I & |\ \bar{\boldsymbol{\lambda}}_\ell, \bar{\boldsymbol{\lambda}}_u &] \\ \overline{\mathsf{QP}}_1 = \overline{\mathsf{QP}}[\ \bar{\mathbf{x}} - \mathbf{s} & |\ \bar{\boldsymbol{\lambda}}_E & |\ \bar{\boldsymbol{\lambda}}_I & |\ \bar{\boldsymbol{\lambda}}_\ell, \bar{\boldsymbol{\lambda}}_u &] \qquad (2.5) \end{array}$$

Solving QP_1 means finding

$$\bar{\mathbf{x}}_1 = \arg\min_{\mathbf{x} \in \Omega_1} f(\mathbf{x}),$$

where

$$\Omega_1 = \{\mathbf{x} \in \mathbb{R}^n \ :\ \mathbf{B}_E\mathbf{x} = \mathbf{c}_E,\ \mathbf{B}_I\mathbf{x} \leq \mathbf{c}_I,\ \boldsymbol{\ell} \leq \mathbf{x} \leq \mathbf{u}\}.$$

We may introduce a substitution

$$\bar{\mathbf{x}}_1 = \bar{\mathbf{x}}_2 - \mathbf{s}$$

and minimize a new objective with the shifted RHS $\mathbf{b} + \mathbf{A}\mathbf{s}$ on the shifted feasible set

$$\Omega_2 = \Omega_1 + \mathbf{s}.$$

We get the corresponding new formulation QP_2 by shifting all right hand sides as shown in (2.4), and $\bar{\mathbf{x}}_1$ using (2.5).

It is intuitively clear that the Lagrange multipliers remain unchanged. This can be easily proven by substituting $(\bar{\mathbf{x}} - \mathbf{s},\ \mathbf{c}_E + \mathbf{B}_E\mathbf{s},\ \mathbf{c}_I + \mathbf{B}_I\mathbf{s},\ \boldsymbol{\ell} + \mathbf{s},\ \mathbf{u} + \mathbf{s})$ for $(\bar{\mathbf{x}}, \mathbf{c}_E, \mathbf{c}_I, \boldsymbol{\ell}, \mathbf{u})$ in the KKT conditions (1.15) and observing that KKT conditions remain unchanged.

2.3.3 Equality constraints homogenization

Synopsis `HomogenizeEq` derives a QP with homogeneous equality constraints, i.e. with zero ERHS

$$\texttt{HomogenizeEq}(\mathsf{QP}_1) = \mathsf{QP}_2 \quad : \quad \mathbf{c}_E \to \mathbf{o}$$

$$\begin{aligned}
\mathsf{QP}_1 &= \mathsf{QP}[\ \mathbf{A}, \mathbf{b} && \mid \mathbf{B}_E, \mathbf{c}_E && \mid \mathbf{B}_I, \mathbf{c}_I && \mid \boldsymbol{\ell}, \mathbf{u} &&] \\
\mathsf{QP}_2 &= \mathsf{QP}[\ \mathbf{A}, \mathbf{b} - \mathbf{A}\widetilde{\mathbf{x}} && \mid \mathbf{B}_E, \mathbf{o} && \mid \mathbf{B}_I, \mathbf{c}_I - \mathbf{B}_I\widetilde{\mathbf{x}} && \mid \boldsymbol{\ell} - \widetilde{\mathbf{x}}, \mathbf{u} - \widetilde{\mathbf{x}} &&] \qquad (2.6) \\
\overline{\mathsf{QP}}_2 &= \overline{\mathsf{QP}}[\ \bar{\mathbf{x}} && \mid \bar{\boldsymbol{\lambda}}_E && \mid \bar{\boldsymbol{\lambda}}_I && \mid \bar{\boldsymbol{\lambda}}_\ell, \bar{\boldsymbol{\lambda}}_u &&] \\
\overline{\mathsf{QP}}_1 &= \overline{\mathsf{QP}}[\ \bar{\mathbf{x}} + \widetilde{\mathbf{x}} && \mid \bar{\boldsymbol{\lambda}}_E && \mid \bar{\boldsymbol{\lambda}}_I && \mid \bar{\boldsymbol{\lambda}}_\ell, \bar{\boldsymbol{\lambda}}_u &&] \qquad (2.7)
\end{aligned}$$

$$\widetilde{\mathbf{x}} \quad \text{solves} \quad \mathbf{B}_E\widetilde{\mathbf{x}} = \mathbf{c}_E \qquad (2.8)$$

$$\mathsf{QP}_2 = \texttt{Shift}(\mathsf{QP}_1, -\widetilde{\mathbf{x}}) \qquad (2.9)$$

This QP transform is a special case of the Affine space shift Section 2.3.2 with $\mathbf{s} = -\widetilde{\mathbf{x}}$ where $\widetilde{\mathbf{x}}$ is a given arbitrary vector satisfying (2.8). For the feasible set Ω_1 of QP_1, it apparently holds that

$$\Omega_1 \subset \Omega_E = \{\mathbf{x} \in \mathbb{R}^n \ : \ \mathbf{B}_E\mathbf{x} = \mathbf{c}_E\}.$$

Ω_E is an affine space, a linear manifold of the form

$$\Omega_E = \widetilde{\mathbf{x}} + \operatorname{Ker}\mathbf{B}_E. \qquad (2.10)$$

The nullspace of $\mathbf{B}_E$

$$\operatorname{Ker}\mathbf{B}_E = \{\mathbf{x} \in \mathbb{R}^n \ : \ \mathbf{B}_E\mathbf{x} = \mathbf{o}\},$$

is a linear space containing our desired feasible set Ω_2,

$$\Omega_2 \subset \operatorname{Ker}\mathbf{B}_E.$$

The relation (2.10) implies $\operatorname{Ker}\mathbf{B}_E = \Omega_E - \widetilde{\mathbf{x}}$ which means shifting all constraint vectors as shown in (2.6). Once $\bar{\mathbf{x}}_2$ is found, we can recover $\bar{\mathbf{x}}_1$ by using (2.7). A typical choice of $\widetilde{\mathbf{x}}$ is the least squares solution of the underdetermined system (2.8)

$$\widetilde{\mathbf{x}} = \mathbf{B}_E^T(\mathbf{B}_E\mathbf{B}_E^T)^{-1}\mathbf{c}_E = \mathbf{B}_E^R\mathbf{c}_E. \qquad (2.11)$$

2.3.4 Enforcing equality constraints using penalty

Synopsis `PenalizeEq` enforces the equality constraints using the penalty method

$$\texttt{PenalizeEq}(\mathsf{QP}_1) = \mathsf{QP}_2 \quad : \quad [\,\mathbf{B}_E \mid \mathbf{c}_E\,] \to [\,\square \mid \square\,]$$

$$\begin{aligned}
\mathsf{QP}_1 &= \mathsf{QP}[\,\mathbf{A}, \mathbf{b} && \mid \mathbf{B}_E, \mathbf{c}_E && \mid \mathbf{B}_I, \mathbf{c}_I \mid \boldsymbol{\ell}, \mathbf{u} \quad] \\
\mathsf{QP}_2 &= \mathsf{QP}[\,\mathbf{A} + \rho\mathbf{B}_E^T\mathbf{B}_E,\ \mathbf{b} + \rho\mathbf{B}_E^T\mathbf{c}_E && \mid \square, \square && \mid \mathbf{B}_I, \mathbf{c}_I \mid \boldsymbol{\ell}, \mathbf{u} \quad] && (2.12) \\
\overline{\mathsf{QP}}_2 &= \overline{\mathsf{QP}}[\,\bar{\mathbf{x}} && \mid \square && \mid \bar{\boldsymbol{\lambda}}_I \quad \mid \bar{\boldsymbol{\lambda}}_\ell, \bar{\boldsymbol{\lambda}}_u\,] \\
\overline{\mathsf{QP}}_1 &= \overline{\mathsf{QP}}[\,\bar{\mathbf{x}} && \mid \rho(\mathbf{B}_E\bar{\mathbf{x}} - \mathbf{c}_E) && \mid \bar{\boldsymbol{\lambda}}_I \quad \mid \bar{\boldsymbol{\lambda}}_\ell, \bar{\boldsymbol{\lambda}}_u\,] && (2.13)
\end{aligned}$$

$$\rho > 0 \quad \text{(penalty parameter)}, \tag{2.14}$$

$$\mathbf{B}_E, \mathbf{B}_I \text{ may have dependent rows.} \tag{2.15}$$

This QP transform enforces the equality constraints by building them into the Hessian, more specifically by adding the *penalization term* $\rho\mathbf{B}_E^T\mathbf{B}_E$ to $\mathbf{A}$. This term penalizes the violation of the equality constraints $\mathbf{B}_E\mathbf{x} = \mathbf{c}_E$. Moreover, it may be considered as a regularization term because $\mathbf{A} + \rho\mathbf{B}_E^T\mathbf{B}_E$ is SPD provided $\mathbf{A}$ is positive semidefinite and $\operatorname{Ker}\mathbf{A} \cap \operatorname{Ker}\mathbf{B}_E = \{\mathbf{o}\}$, see Lemma 1.2 in [10].

The larger is the penalty parameter ρ the nearer is the solution of the derived problem $\bar{\mathbf{x}}_2$ to the solution of the original problem $\bar{\mathbf{x}}_1$. Informally said, when the penalty ρ tends to infinity, the equality $\bar{\mathbf{x}}_2 = \bar{\mathbf{x}}_1$ is almost exactly satisfied. For a finite ρ the equality constraints are enforced only approximately. However, they can be satisfied to an arbitrary given precision. This is why they are omitted in (2.12).

Let us discuss this transform in more detail, based on observations from [10] where only a particular case of equality constrained problems is considered. Assume problem $\widehat{\mathsf{QP}_2}$ which is the same as QP_2 except that the equality constraints are still explicitly present, i.e.

$$\widehat{\mathsf{QP}_2} = \mathsf{QP}[\,\mathbf{A} + \rho\mathbf{B}_E^T\mathbf{B}_E,\ \mathbf{b} + \rho\mathbf{B}_E^T\mathbf{c}_E \mid \mathbf{B}_E, \mathbf{c}_E \mid \mathbf{B}_I, \mathbf{c}_I \mid \boldsymbol{\ell}, \mathbf{u}\,].$$

The objective functions of QP_1 and $\widehat{\mathsf{QP}_2}$ read respectively

$$f_0(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{x}^T\mathbf{b}, \tag{2.16}$$

$$\begin{aligned}
f_\rho(\mathbf{x}) &= \frac{1}{2}\mathbf{x}^T(\mathbf{A} + \rho\mathbf{B}_E^T\mathbf{B}_E)\mathbf{x} - \mathbf{x}^T(\mathbf{b} + \rho\mathbf{B}_E^T\mathbf{c}_E) = \\
&= f_0(\mathbf{x}) + \frac{\rho}{2}||\mathbf{B}_E\mathbf{x} - \mathbf{c}||^2,
\end{aligned} \tag{2.17}$$

and their gradients with respect to $\mathbf{x}$ are

$$\nabla_{\mathbf{x}} f_0(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b}, \tag{2.18}$$

$$\begin{aligned}\nabla_{\mathbf{x}} f_\rho(\mathbf{x}) &= (\mathbf{A} + \rho\mathbf{B}_E^T\mathbf{B}_E)\mathbf{x} - (\mathbf{b} + \rho\mathbf{B}_E^T\mathbf{c}_E) = \\ &= \nabla_{\mathbf{x}} f_0(\mathbf{x}) + \rho\mathbf{B}_E^T(\mathbf{B}_E\mathbf{x} - \mathbf{c}_E).\end{aligned} \tag{2.19}$$

Lagrangians of the objective functions take the form

$$L_0(\mathbf{x}, \boldsymbol{\lambda}) = f_0(\mathbf{x}) \quad + \boldsymbol{\lambda}_E^T(\mathbf{B}_E\mathbf{x} - \mathbf{c}_E) + \omega_r(\mathbf{x}, \boldsymbol{\lambda}), \tag{2.20}$$

$$\begin{aligned}L_\rho(\mathbf{x}, \boldsymbol{\lambda}) &= f_\rho(\mathbf{x}) \quad + \boldsymbol{\lambda}_E^T(\mathbf{B}_E\mathbf{x} - \mathbf{c}_E) + \omega_r(\mathbf{x}, \boldsymbol{\lambda}) = \\ &= f_0(\mathbf{x}) \quad + \boldsymbol{\lambda}_E^T(\mathbf{B}_E\mathbf{x} - \mathbf{c}_E) + \omega_r(\mathbf{x}, \boldsymbol{\lambda}) + \frac{\rho}{2}||\mathbf{B}_E\mathbf{x} - \mathbf{c}||^2 = \\ &= L_0(\mathbf{x}, \boldsymbol{\lambda}) + \frac{\rho}{2}||\mathbf{B}_E\mathbf{x} - \mathbf{c}||^2,\end{aligned} \tag{2.21}$$

where $\omega_r(\mathbf{x}, \boldsymbol{\lambda}) = \boldsymbol{\lambda}_r^T h_r(\mathbf{x})$, $h_r(\mathbf{x}) = \left[h_I^T(\mathbf{x}),\ h_\ell^T(\mathbf{x}),\ h_u^T(\mathbf{x})\right]^T$ and $\boldsymbol{\lambda}_r = \left[\boldsymbol{\lambda}_I^T,\ \boldsymbol{\lambda}_\ell^T,\ \boldsymbol{\lambda}_u^T\right]^T$. Let us also introduce the feasible set of the inequality constraints

$$\Omega_r = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{B}_I\mathbf{x} \leq \mathbf{c}_I \wedge \boldsymbol{\ell} \leq \mathbf{x} \leq \mathbf{u}\}.$$

L_ρ is called an *augmented Lagrangian* of QP_1. Gradients of both Lagrangians with respect to $\mathbf{x}$ read

$$\nabla_{\mathbf{x}} L_0(\mathbf{x}, \boldsymbol{\lambda}) = \nabla_{\mathbf{x}} f_0(\mathbf{x}) \quad + \mathbf{B}_E^T\boldsymbol{\lambda}_E + \nabla_{\mathbf{x}}\omega_r(\mathbf{x}, \boldsymbol{\lambda}), \tag{2.22}$$

$$\begin{aligned}\nabla_{\mathbf{x}} L_\rho(\mathbf{x}, \boldsymbol{\lambda}) &= \nabla_{\mathbf{x}} f_\rho(\mathbf{x}) \quad + \mathbf{B}_E^T\boldsymbol{\lambda}_E + \nabla_{\mathbf{x}}\omega_r(\mathbf{x}, \boldsymbol{\lambda}) = \\ &= \nabla_{\mathbf{x}} f_0(\mathbf{x}) \quad + \mathbf{B}_E^T(\boldsymbol{\lambda}_E + \rho(\mathbf{B}_E\mathbf{x} - \mathbf{c}_E)) + \nabla_{\mathbf{x}}\omega_r(\mathbf{x}, \boldsymbol{\lambda}) = \\ &= \nabla_{\mathbf{x}} L_0(\mathbf{x}, \boldsymbol{\lambda}) + \rho\mathbf{B}_E^T(\mathbf{B}_E\mathbf{x} - \mathbf{c}_E),\end{aligned} \tag{2.23}$$

where $\nabla_{\mathbf{x}}\omega_r(\mathbf{x}, \boldsymbol{\lambda}) = \mathbf{B}_I^T\bar{\boldsymbol{\lambda}}_I - \bar{\boldsymbol{\lambda}}_\ell + \bar{\boldsymbol{\lambda}}_u$.

The first KKT condition (1.15a) for the solution of QP_1 and $\widehat{\mathsf{QP}_2}$ takes the form, respectively,

$$\nabla_{\mathbf{x}} L_0(\mathbf{x}, \boldsymbol{\lambda}) = \mathbf{o}, \tag{2.24}$$

$$\nabla_{\mathbf{x}} L_\rho(\mathbf{x}, \boldsymbol{\lambda}) = \mathbf{o}. \tag{2.25}$$

Let $(\bar{\mathbf{x}}, \bar{\boldsymbol{\lambda}})$ be a KKT pair of QP_1, thus it satisfies (2.24) and let us substitute $\mathbf{c}_E = \mathbf{B}_E\bar{\mathbf{x}}$ into (2.25). We can see that QP_1 has exactly the same KKT pair as $\widehat{\mathsf{QP}_2}$, i.e. the problems are equivalent.

Furthermore, let us show how the penalty enforces the equality constraints. Assume arbitrary fixed $\widehat{\boldsymbol{\lambda}}_E \in \mathbb{R}_E^m$ and

$$(\bar{\mathbf{x}}_\rho, \widehat{\boldsymbol{\lambda}}) = \arg\min_{\mathbf{x}\in\mathbb{R}^n} \max_{\substack{\boldsymbol{\lambda}_r \geq \mathbf{o} \\ \boldsymbol{\lambda}_E = \widehat{\boldsymbol{\lambda}}_E}} L_\rho(\mathbf{x}, \boldsymbol{\lambda}), \tag{2.26}$$

$$\bar{\mathbf{x}}_0 = \arg\min_{\mathbf{x}\in\mathbb{R}^n} L_0(\mathbf{x}, \widehat{\boldsymbol{\lambda}}). \tag{2.27}$$

Then the solution $\bar{\mathbf{x}}$ of QP_1 (which is at the same time solution of $\widehat{\mathsf{QP}_2}$) satisfies

$$L_0(\bar{\mathbf{x}}_\rho, \widehat{\boldsymbol{\lambda}}) + \frac{\rho}{2}||\mathbf{B}_E\bar{\mathbf{x}}_\rho - \mathbf{c}_E||^2 = L_\rho(\bar{\mathbf{x}}_\rho, \widehat{\boldsymbol{\lambda}}) \overset{(2.26)}{\leq} L_\rho(\bar{\mathbf{x}}, \widehat{\boldsymbol{\lambda}}) = f_0(\bar{\mathbf{x}}) + \omega_r(\bar{\mathbf{x}}, \widehat{\boldsymbol{\lambda}}),$$

so that we get

$$||\mathbf{B}_E\bar{\mathbf{x}}_\rho - \mathbf{c}_E||^2 \leq \frac{2}{\rho}\big(f_0(\bar{\mathbf{x}}) + \omega_r(\bar{\mathbf{x}}, \widehat{\boldsymbol{\lambda}}) - L_0(\bar{\mathbf{x}}_\rho, \widehat{\boldsymbol{\lambda}})\big).$$

Finally, this implies using $L_0(\bar{\mathbf{x}}_0, \widehat{\boldsymbol{\lambda}}) \leq L_0(\bar{\mathbf{x}}_\rho, \widehat{\boldsymbol{\lambda}})$ following from (2.27) that

$$||\mathbf{B}_E\bar{\mathbf{x}}_\rho - \mathbf{c}_E||^2 \leq \frac{2}{\rho}\big(f_0(\bar{\mathbf{x}}) + \omega_r(\bar{\mathbf{x}}, \widehat{\boldsymbol{\lambda}}) - L_0(\bar{\mathbf{x}}_0, \widehat{\boldsymbol{\lambda}})\big). \tag{2.28}$$

We can conclude that the feasibility error $||\mathbf{B}_E\bar{\mathbf{x}}_\rho - \mathbf{c}_E||$ expressing violation of the second KKT condition (1.15b) can be made arbitrarily small.

Now let us focus on the satisfaction of the first KKT condition (2.24). Let us denote

$$\bar{\boldsymbol{\lambda}}_{E\rho} = \widehat{\boldsymbol{\lambda}}_E + \rho(\mathbf{B}_E\bar{\mathbf{x}}_\rho - \mathbf{c}_E), \qquad \bar{\boldsymbol{\lambda}}_\rho = \begin{bmatrix} \bar{\boldsymbol{\lambda}}_{E\rho} \\ \widehat{\boldsymbol{\lambda}}_r \end{bmatrix}. \tag{2.29}$$

Then

$$\mathbf{A}\bar{\mathbf{x}}_\rho - \mathbf{b} + \mathbf{B}_E^T\bar{\boldsymbol{\lambda}}_{E\rho} + \nabla_{\mathbf{x}}\omega_r(\bar{\mathbf{x}}_\rho, \bar{\boldsymbol{\lambda}}_\rho) \overset{(2.22)}{=} \nabla_{\mathbf{x}}L_0(\bar{\mathbf{x}}_\rho, \bar{\boldsymbol{\lambda}}_\rho) \overset{(2.23)(2.29)}{=} \nabla_{\mathbf{x}}L_\rho(\bar{\mathbf{x}}_\rho, \widehat{\boldsymbol{\lambda}}) \overset{(2.26)}{=} \mathbf{o},$$

so that $(\bar{\mathbf{x}}_\rho, \bar{\boldsymbol{\lambda}}_\rho)$ satisfies the first KKT condition (2.24) exactly.

The given $\widehat{\boldsymbol{\lambda}} \in \mathbb{R}^m$, optimal in terms of inequality constraints, can be considered as an approximation of the solution Lagrange multipliers $\bar{\boldsymbol{\lambda}}$ of QP_1. Nevertheless, from discussion above it is obvious that $\bar{\boldsymbol{\lambda}}_\rho$ is always better approximation. We can conclude that $(\bar{\mathbf{x}}_\rho, \bar{\boldsymbol{\lambda}}_\rho)$ approximates the KKT pair of QP_1 with an arbitrarily small error. Moreover, $\bar{\mathbf{x}}_\rho$ and $\bar{\boldsymbol{\lambda}}_\rho$ are obtained performing only inequality constrained minimization. Thus the lack of equality constraints in QP_2 is justified.

By this means, we get only an approximate KKT pair whose error depends indirectly on the penalty ρ and the least nonzero singular value of $\mathbf{B}_E$, and directly on the regular condition number of $\mathbf{A}$. See Section 4.2 in [10] for rigorous feasibility and approximation error estimates. This simple penalty method is a base for the more sophisticated algorithm of Section 3.2 which applies the penalty method in the loop with progressive correction of $\boldsymbol{\lambda}_E$. Herewith, convergence rate and both approximation and feasibility errors are much less affected by the a priori choice of ρ.

2.3.5 Preconditioning by the orthogonal projector

Synopsis `PreconditionByP` preconditions the objective function using the orthogonal projector onto the nullspace of $\mathbf{B}_E$

$$\texttt{EnforceEq}(\mathsf{QP}_1) = \mathsf{QP}_2$$

$$\begin{aligned}
\mathsf{QP}_1 &= \mathsf{QP}[\ \mathbf{A}, \mathbf{b} \quad | \ \mathbf{B}_E, \mathbf{o} \quad | \ \mathbf{B}_I, \mathbf{c}_I \ | \ \boldsymbol{\ell}, \mathbf{u} \ ] \\
\mathsf{QP}_2 &= \mathsf{QP}[\ \mathbf{PAP}, \mathbf{Pb} \ | \ \mathbf{B}_E, \mathbf{o} \quad | \ \mathbf{B}_I, \mathbf{c}_I \ | \ \boldsymbol{\ell}, \mathbf{u} \ ] \qquad (2.30) \\
\overline{\mathsf{QP}}_2 &= \overline{\mathsf{QP}}[\ \bar{\mathbf{x}} \quad | \ \bar{\boldsymbol{\lambda}}_E \quad | \ \bar{\boldsymbol{\lambda}}_I \quad | \ \bar{\boldsymbol{\lambda}}_\ell, \bar{\boldsymbol{\lambda}}_u \ ] \\
\overline{\mathsf{QP}}_1 &= \overline{\mathsf{QP}}[\ \bar{\mathbf{x}} \quad | \ \bar{\boldsymbol{\lambda}}_E + (\mathbf{B}_E^T)^L(\mathbf{b} - \mathbf{A}\mathbf{x}) \ | \ \bar{\boldsymbol{\lambda}}_I \quad | \ \bar{\boldsymbol{\lambda}}_\ell, \bar{\boldsymbol{\lambda}}_u \ ] \qquad (2.31)
\end{aligned}$$

$$\mathbf{P} = \mathbf{I} - \mathbf{B}_E^R\mathbf{B}_E, \quad \operatorname{Im}\mathbf{P} = \operatorname{Ker}\mathbf{B}_E \qquad (2.32)$$

This is a special case of preconditioning where the preconditioner is the matrix $\mathbf{P}$ from (2.32), which is a projector onto $\operatorname{Ker}\mathbf{B}_E$. Let us remind that

$$\mathbf{Q} = \mathbf{B}_E^T(\mathbf{B}_E\mathbf{B}_E^T)^{-1}\mathbf{B}_E = \mathbf{B}_E^T(\mathbf{B}_E^T)^L = \mathbf{B}_E^R\mathbf{B}_E \quad \text{and} \quad \mathbf{P} = \mathbf{I} - \mathbf{Q} \qquad (2.33)$$

are complementary orthogonal projectors onto $\operatorname{Im}\mathbf{B}_E^T$ and $\operatorname{Ker}\mathbf{B}_E$, respectively. This means they have the following properties

$$\operatorname{Im}\mathbf{P} = \operatorname{Ker}\mathbf{B}_E = \operatorname{Ker}\mathbf{Q}, \qquad (2.34)$$

$$\operatorname{Im}\mathbf{Q} = \operatorname{Im}\mathbf{B}_E^T = \operatorname{Ker}\mathbf{P}, \qquad (2.35)$$

$$\operatorname{Im}\mathbf{P} \oplus \operatorname{Im}\mathbf{Q} = \operatorname{Ker}\mathbf{B}_E \oplus \operatorname{Im}\mathbf{B}_E^T = \mathbb{R}^n. \qquad (2.36)$$

Let us derive the relation between equality constraint multipliers of QP_1 and QP_2. We can notice that the optimality conditions (1.15) differ only in the statinonarity condition (1.15a) which reads for QP_1 and QP_2, respectively,

$$\nabla_{\mathbf{x}} L_1(\mathbf{x}, \boldsymbol{\lambda}) = \mathbf{A}\mathbf{x} - \mathbf{b} + \mathbf{B}_E^T\boldsymbol{\lambda}_E + \mathbf{B}_I^T\boldsymbol{\lambda}_I + \boldsymbol{\lambda}_u - \boldsymbol{\lambda}_\ell = \mathbf{o}, \qquad (2.37)$$

$$\nabla_{\mathbf{x}} L_2(\mathbf{x}, \boldsymbol{\lambda}) = \mathbf{PAP}\mathbf{x} - \mathbf{Pb} + \mathbf{B}_E^T\boldsymbol{\lambda}_E + \mathbf{B}_I^T\boldsymbol{\lambda}_I + \boldsymbol{\lambda}_u - \boldsymbol{\lambda}_\ell = \mathbf{o}. \qquad (2.38)$$

Let us split the equality constraint multipliers and inequality constraint ones. The KKT pairs of QP_1 and QP_2 must respectively satisfy

$$\nabla_{\mathbf{x}} L_1(\bar{\mathbf{x}}_1, \bar{\boldsymbol{\lambda}}_{E1}, \bar{\boldsymbol{\lambda}}_{r1}) = \mathbf{A}\bar{\mathbf{x}}_1 - \mathbf{b} + \mathbf{B}_E^T\bar{\boldsymbol{\lambda}}_{E1} + \mathbf{r}(\bar{\boldsymbol{\lambda}}_{r1}) = \mathbf{o}, \qquad (2.39)$$

$$\nabla_{\mathbf{x}} L_2(\bar{\mathbf{x}}_2, \bar{\boldsymbol{\lambda}}_{E2}, \bar{\boldsymbol{\lambda}}_{r2}) = \mathbf{PAP}\bar{\mathbf{x}}_2 - \mathbf{Pb} + \mathbf{B}_E^T\bar{\boldsymbol{\lambda}}_{E2} + \mathbf{r}(\bar{\boldsymbol{\lambda}}_{r2}) = \mathbf{o}, \qquad (2.40)$$

where

$$\boldsymbol{\lambda}_r = \left[\boldsymbol{\lambda}_I^T,\ \boldsymbol{\lambda}_\ell^T,\ \boldsymbol{\lambda}_u^T\right]^T,$$
$$\mathbf{r}(\boldsymbol{\lambda}_r) = \mathbf{B}_I^T\boldsymbol{\lambda}_I + \boldsymbol{\lambda}_u - \boldsymbol{\lambda}_\ell. \tag{2.41}$$

Assume that $\bar{\mathbf{x}}_2$ and $\bar{\boldsymbol{\lambda}}_{r2}$ are respectively the solution and the optimal multipliers of inequality constraints of QP_2, satisfying optimality conditions (1.15b)–(1.15e). Denote $\mathbf{r} = \mathbf{r}(\bar{\boldsymbol{\lambda}}_{r2})$.

First of all, let us seek the equality constraint multiplier $\bar{\boldsymbol{\lambda}}_{E2}$ such that the stationarity condition (2.40) is satisfied. Let us proceed similarly to Section 1.7.3. The equation (2.40) can be equivalently expressed as an over-determined linear system

$$\mathbf{B}_E^T\boldsymbol{\lambda}_{E2} = -(\mathbf{PAP}\bar{\mathbf{x}}_2 - \mathbf{Pb} + \mathbf{r}), \tag{2.42}$$

and the optimal equality constraint multiplier $\bar{\boldsymbol{\lambda}}_{E2}$ must solve this system. Thanks to our assumptions, there exists one and only one such $\bar{\boldsymbol{\lambda}}_{E2}$ and is equal to the least-square solution

$$\bar{\boldsymbol{\lambda}}_{E2} = -(\mathbf{B}_E^T)^L\left(\mathbf{PAP}\bar{\mathbf{x}}_2 - \mathbf{Pb} + \mathbf{r}\right).$$

Noticing $\mathbf{PAP}\bar{\mathbf{x}}_2, \mathbf{Pb} \in \operatorname{Ker}\mathbf{B}_E$, we get $\mathbf{B}_E(\mathbf{PAP}\bar{\mathbf{x}}_2 - \mathbf{Pb}) = \mathbf{o}$, which means the equation above reads

$$\bar{\boldsymbol{\lambda}}_{E2} = -(\mathbf{B}_E^T)^L\mathbf{r}. \tag{2.43}$$

Let us show that such $\bar{\boldsymbol{\lambda}}_{E2}$ indeed solves the equation (2.42). We substitute back into this relation and move $\mathbf{r}$ to the same side,

$$-\mathbf{B}_E^T(\mathbf{B}_E^T)^L\mathbf{r} + \mathbf{r} \stackrel{?}{=} -(\mathbf{PAP}\bar{\mathbf{x}}_2 - \mathbf{Pb}). \tag{2.44}$$

The left-hand side is equal to

$$-\mathbf{Qr} + \mathbf{r} = \mathbf{Pr}.$$

Regarding the right hand side, we assume $\bar{\mathbf{x}}_2$ is the solution of QP_2, so from (2.40) it follows that

$$-(\mathbf{PAP}\bar{\mathbf{x}}_2 - \mathbf{Pb}) = \mathbf{Pr}, \tag{2.45}$$

hence the equality (2.44) is proven. We can conclude that for the given $\bar{\mathbf{x}}_2$ and $\bar{\boldsymbol{\lambda}}_{r2}$, there is a unique equality constraint multiplier $\bar{\boldsymbol{\lambda}}_{E2}$ of QP_2, taking the form

$$\bar{\boldsymbol{\lambda}}_{E2} = -(\mathbf{B}_E^T)^L\mathbf{r}(\bar{\boldsymbol{\lambda}}_{r2}). \tag{2.46}$$

Furthermore, we shall derive how the KKT pair $(\bar{\mathbf{x}}_1, \bar{\boldsymbol{\lambda}}_{E1}, \bar{\boldsymbol{\lambda}}_{r1})$ of QP_1 can be recovered from $(\bar{\mathbf{x}}_2, \bar{\boldsymbol{\lambda}}_{E2}, \bar{\boldsymbol{\lambda}}_{r2})$. Let us try to use $\bar{\mathbf{x}}_1 = \bar{\mathbf{x}}_2$ and $\bar{\boldsymbol{\lambda}}_{r1} = \bar{\boldsymbol{\lambda}}_{r2}$. In this case, satisfaction of all but the first KKT conditions (1.15) is inherited from QP_2. Let us carry out the corresponding equality constraint multipliers $\bar{\boldsymbol{\lambda}}_{E1}$. From (2.39) it easily follows that

$$\mathbf{r} = -(\mathbf{A}\bar{\mathbf{x}}_1 - \mathbf{b} + \mathbf{B}_E^T\bar{\boldsymbol{\lambda}}_{E1}). \tag{2.47}$$

Substituting this term into (2.46), we get

$$\bar{\boldsymbol{\lambda}}_{E2} = (\mathbf{B}_E^T)^L(\mathbf{A}\bar{\mathbf{x}}_1 - \mathbf{b}) \;+\; \bar{\boldsymbol{\lambda}}_{E1}.$$

Hence,

$$\bar{\boldsymbol{\lambda}}_{E1} = \bar{\boldsymbol{\lambda}}_{E2} - (\mathbf{B}_E^T)^L(\mathbf{A}\bar{\mathbf{x}}_1 - \mathbf{b}). \tag{2.48}$$

Finally, let us show that the suggested triplet $(\bar{\mathbf{x}}_1, \bar{\boldsymbol{\lambda}}_{E1}, \bar{\boldsymbol{\lambda}}_{r1})$ meets the stationarity condition (2.39) of QP_1:

$$\begin{aligned}
\nabla_{\mathbf{x}} L_1(\bar{\mathbf{x}}_1, \bar{\boldsymbol{\lambda}}_{E1}, \bar{\boldsymbol{\lambda}}_{r1}) &= \mathbf{A}\bar{\mathbf{x}}_1 - \mathbf{b} + \mathbf{B}_E^T\bar{\boldsymbol{\lambda}}_{E1} + \mathbf{r}(\bar{\boldsymbol{\lambda}}_{r1}) = \\
&\overset{(2.48)}{=} (\mathbf{A}\bar{\mathbf{x}}_2 - \mathbf{b}) + \mathbf{B}_E^T\left(\bar{\boldsymbol{\lambda}}_{E2} - (\mathbf{B}_E^T)^L(\mathbf{A}\bar{\mathbf{x}}_2 - \mathbf{b})\right)\mathbf{r}(\bar{\boldsymbol{\lambda}}_{r2}) = \\
&= (\mathbf{A}\bar{\mathbf{x}}_2 - \mathbf{b}) - \mathbf{B}_E^T(\mathbf{B}_E^T)^L(\mathbf{A}\bar{\mathbf{x}}_2 - \mathbf{b}) + \mathbf{B}_E^T\bar{\boldsymbol{\lambda}}_{E2} + \mathbf{r}(\bar{\boldsymbol{\lambda}}_{r2}) = \\
&\overset{(2.46)}{=} \mathbf{P}(\mathbf{A}\bar{\mathbf{x}}_2 - \mathbf{b}) - \mathbf{B}_E^T(\mathbf{B}_E^T)^L\mathbf{r}(\bar{\boldsymbol{\lambda}}_{r2}) + \mathbf{r}(\bar{\boldsymbol{\lambda}}_{r2}) = \\
&= \mathbf{P}\left(\mathbf{A}\bar{\mathbf{x}}_2 - \mathbf{b} + \mathbf{r}(\bar{\boldsymbol{\lambda}}_{r2})\right) \overset{(2.45)}{=} \mathbf{o}.
\end{aligned} \tag{2.49}$$

2.3.6 Eliminating homogeneous equality constraints with the orthogonal projector

Synopsis `PreconditionByP` eliminates homogeneous equality constraints using the orthogonal projector onto $\operatorname{Ker}\mathbf{B}_E$ provided there are no other constraints prescribed

$$\texttt{EnforceEq}(\mathsf{QP}_1) = \mathsf{QP}_2 \quad : \quad [\,\mathbf{B}_E \mid \mathbf{c}_E = \mathbf{o}\,] \to [\,\square \mid \square\,]$$

$$\begin{aligned}
\mathsf{QP}_1 &= \mathsf{QP}[\,\mathbf{A}, \mathbf{b} \quad\;\; \mid \mathbf{B}_E, \mathbf{o} \quad\;\; \mid \square, \square \mid \square, \square\,] \\
\mathsf{QP}_2 &= \mathsf{QP}[\,\mathbf{PAP}, \mathbf{Pb} \mid \square, \square \quad\;\; \mid \square, \square \mid \square, \square\,] \qquad (2.50) \\
\overline{\mathsf{QP}}_2 &= \overline{\mathsf{QP}}[\,\bar{\mathbf{x}} \quad\;\; \mid \square \quad\;\; \mid \square \quad\;\; \mid \square, \square\,] \\
\overline{\mathsf{QP}}_1 &= \overline{\mathsf{QP}}[\,\mathbf{P}\bar{\mathbf{x}} \quad\;\; \mid (\mathbf{B}_E^T)^L(\mathbf{b} - \mathbf{A}\bar{\mathbf{x}}) \mid \square \quad\;\; \mid \square, \square\,] \qquad (2.51)
\end{aligned}$$

$$\mathbf{P} = \mathbf{I} - \mathbf{B}_E^R\mathbf{B}_E, \quad \operatorname{Im}\mathbf{P} = \operatorname{Ker}\mathbf{B}_E \tag{2.52}$$

This is a special case of the transform described in Section 2.3.5 where no other than linear equality constraints are prescribed. In this case, the orthogonal projector $\mathbf{P}$ from (2.52) enforces on its own the equality constraints. When combined with the homogenization transform (see Section 2.3.3) and CG method solving the resulting unconstrained QP, we get a simple method for solution of equality constrained QP.

Showing that equality constraints are eliminated in this case is easy. From (2.40) and from the fact that no inequality constraints are prescribed, it follows that

$$-\mathbf{r}(\boldsymbol{\lambda}_{r2}) = \mathbf{PAP}\bar{\mathbf{x}}_2 - \mathbf{Pb} + \mathbf{B}_E^T\bar{\boldsymbol{\lambda}}_{E2} = \mathbf{o}. \tag{2.53}$$

Moreover, from obvious facts

$$\operatorname{Ker}\mathbf{B}_E \ \ni\ \mathbf{PAP}\bar{\mathbf{x}}_2 - \mathbf{Pb} \ =\ -\mathbf{B}_{E2}^T\bar{\boldsymbol{\lambda}}_{E2} \ \in\ \operatorname{Im}\mathbf{B}_E^T,$$

and from orthogonality of the spaces we get

$$\mathbf{PAP}\bar{\mathbf{x}}_2 - \mathbf{Pb} \ =\ \mathbf{o} \ =\ \mathbf{B}_E^T\bar{\boldsymbol{\lambda}}_{E2}.$$

The left equality is actually the stationarity condition of QP_2 and corresponds to unconstrained problem whose solution is also solution of QP_1. The feasibility condition of QP_1, $\mathbf{B}_E\mathbf{x}_1 = \mathbf{o}$, is satisfied automatically for $\bar{\mathbf{x}}_1 = \mathbf{P}\bar{\mathbf{x}}_2$.

Note that when a Krylov subspace method is used for solving the resulting problem, solution approximations do not leave $\operatorname{Im}\mathbf{P} = \operatorname{Ker}\mathbf{B}_E$. Thus the Hessian of QP_2 can be simplified to $\mathbf{PA}$. Moreover, for the solution of QP_2 it holds $\bar{\mathbf{x}} \in \operatorname{Ker}\mathbf{B}_E$ so that the solution of QP_1 is the same vector $\mathbf{P}\bar{\mathbf{x}} = \bar{\mathbf{x}}$. These facts follow from the fact

$$\mathcal{K}(\mathbf{PAP}, \mathbf{PAPx}_0 - \mathbf{Pb}) = \mathcal{K}(\mathbf{PA}, \mathbf{PAPx}_0 - \mathbf{Pb}) \subseteq \operatorname{Ker}\mathbf{B}_E,$$

where $\mathcal{K}(\mathbf{M}, \mathbf{x})$ denotes any Krylov space generated by matrix $\mathbf{M}$ and vector $\mathbf{x}$, and $\mathbf{x}_0$ is an arbitrary initial solution guess.

2.3.7 Dualization

Synopsis `Dualize` transforms the given QP into the dual one. Solution of the dual QP is the vector of the optimal Lagrange multiplier of the original QP.

$$\texttt{Dualize}(\mathsf{QP}_1) = \mathsf{QP}_2$$

$$\begin{array}{lllll}
\mathsf{QP}_1 = \mathsf{QP}[\ \mathbf{A}, \mathbf{b} & |\ \mathbf{B}_E, \mathbf{c}_E & |\ \mathbf{B}_I, \mathbf{c}_I & |\ \square, \square &] \\
\mathsf{QP}_2 = \mathsf{QP}[\ \mathbf{F}, \mathbf{d} & |\ \mathbf{G}, \mathbf{e} & |\ \square, \square & |\ \mathbf{o}, \square, \mathcal{I} &] \qquad (2.54a) \\
\overline{\mathsf{QP}}_2 = \overline{\mathsf{QP}}[\ \bar{\boldsymbol{\lambda}} & |\ \bar{\boldsymbol{\alpha}} & |\ \square & |\ \bar{\boldsymbol{\beta}}, \square, \mathcal{I} &] \\
\overline{\mathsf{QP}}_1 = \overline{\mathsf{QP}}[\ \mathbf{A}^\dagger(\mathbf{b} - \mathbf{B}^T\bar{\boldsymbol{\lambda}}) - \mathbf{R}\bar{\boldsymbol{\alpha}} & |\ \bar{\boldsymbol{\lambda}}_E & |\ \bar{\boldsymbol{\lambda}}_I & |\ \square, \square &] \qquad (2.54b)
\end{array}$$

$$\mathbf{B}_E, \mathbf{B}_I \text{ may have dependent rows,} \tag{2.54c}$$

$$\mathbf{R} \in \mathbb{R}^{n\times d}, \qquad \operatorname{Im}\mathbf{R} = \operatorname{Ker}\mathbf{A}, \qquad \dim\operatorname{Im}\mathbf{R} = d, \tag{2.54d}$$

$$\mathbf{B} = \begin{bmatrix}\mathbf{B}_I\\ \mathbf{B}_E\end{bmatrix}, \qquad \mathbf{c} = \begin{bmatrix}\mathbf{c}_I\\ \mathbf{c}_E\end{bmatrix}, \qquad \bar{\boldsymbol{\lambda}} = \begin{bmatrix}\bar{\boldsymbol{\lambda}}_I\\ \bar{\boldsymbol{\lambda}}_E\end{bmatrix} \tag{2.54e}$$

$$\mathbf{F} = \mathbf{B}\mathbf{A}^\dagger\mathbf{B}^T, \qquad \mathbf{d} = \mathbf{B}\mathbf{A}^\dagger\mathbf{b} - \mathbf{c}, \qquad \mathcal{I} = (1,\dots,m_I), \tag{2.54f}$$

$$\mathbf{G} = \mathbf{R}^T\mathbf{B}^T, \qquad \mathbf{e} = \mathbf{R}^T\mathbf{b}. \tag{2.54g}$$

First of all, note that box constraints are not considered here as they can be merged with the linear inequality constrains $\mathbf{B}_I\mathbf{x} \le \mathbf{c}_I$ using the `EliminateBox` transform according to Section 2.3.1. The following discussion is based on the proof of Proposition 2.22 in [10]. Assume that $(\bar{\mathbf{x}}, \bar{\boldsymbol{\lambda}})$ is a KKT pair for QP_1, so that it meets KKT condition (1.15) which now more specifically read

$$\nabla_{\mathbf{x}} L(\bar{\mathbf{x}}, \bar{\boldsymbol{\lambda}}) = \mathbf{A}\bar{\mathbf{x}} - \mathbf{b} + \mathbf{B}^T\bar{\boldsymbol{\lambda}} = \mathbf{o}, \tag{2.55a}$$

$$[\nabla_{\boldsymbol{\lambda}} L(\bar{\mathbf{x}}, \bar{\boldsymbol{\lambda}})]_{\mathcal{I}} = [\mathbf{B}\bar{\mathbf{x}} - \mathbf{c}]_{\mathcal{I}} \le \mathbf{o}, \tag{2.55b}$$

$$[\nabla_{\boldsymbol{\lambda}} L(\bar{\mathbf{x}}, \bar{\boldsymbol{\lambda}})]_{\mathcal{E}} = [\mathbf{B}\bar{\mathbf{x}} - \mathbf{c}]_{\mathcal{E}} = \mathbf{o}, \tag{2.55c}$$

$$\bar{\boldsymbol{\lambda}}_{\mathcal{I}}^T[\mathbf{B}\bar{\mathbf{x}} - \mathbf{c}]_{\mathcal{I}} = 0, \tag{2.55d}$$

$$\bar{\boldsymbol{\lambda}}_{\mathcal{I}} \ge \mathbf{o}, \tag{2.55e}$$

where $\mathcal{I} = (1,\dots,m_I)$ and $\mathcal{E} = (m_I+1,\dots,n)$, $\mathcal{I}\cup\mathcal{E} = (1,\dots,n)$, denote the index set corresponding to inequality and equality constraints, respectively, i.e.

$$\mathbf{B}_{\mathcal{I}} = \mathbf{B}_I \quad \text{and} \quad \mathbf{B}_{\mathcal{E}} = \mathbf{B}_E.$$

For a given vector $\boldsymbol{\lambda} \in \mathbb{R}^m$, the linear system (2.55a) is solvable with respect to $\mathbf{x}$ if and only if

$$\mathbf{b} - \mathbf{B}^T\boldsymbol{\lambda} \in \operatorname{Im}\mathbf{A}. \tag{2.56}$$

This can be expressed in terms of the matrix $\mathbf{R}$ whose columns span the kernel of $\mathbf{A}$ as

$$\mathbf{R}^T(\mathbf{B}^T\boldsymbol{\lambda} - \mathbf{b}) = \mathbf{o}. \tag{2.57}$$

If the latter condition is satisfied, then we can use any symmetric generalized inverse $\mathbf{A}^\dagger$ to find all solutions of (2.55a) with respect to $\mathbf{x}$ in the form

$$\mathbf{x}(\boldsymbol{\lambda}, \boldsymbol{\alpha}) = \mathbf{A}^\dagger(\mathbf{b} - \mathbf{B}^T\boldsymbol{\lambda}) - \mathbf{R}\boldsymbol{\alpha}, \quad \boldsymbol{\alpha} \in \mathbb{R}^d. \tag{2.58}$$

Note that in a special case of nonsingular[1] $\mathbf{A}$, $\operatorname{Ker}\mathbf{A} = \{\mathbf{o}\}$, $d = 0$, $\mathbf{R} = \square^{(n,0)}$, $\boldsymbol{\alpha} = \square^{(0,1)}$ and $\mathbf{A}^\dagger = \mathbf{A}^{-1}$, i.e. the previous equation is simplified into the form

$$\mathbf{x}(\boldsymbol{\lambda}) = \mathbf{A}^{-1}(\mathbf{b} - \mathbf{B}^T\boldsymbol{\lambda}). \tag{2.59}$$

[1] i.e. SPD in our case as we assume only SPS matrices

After substituting for $\mathbf{x}$ into (2.55b)–(2.55d), we get

$$[-\mathbf{B}\mathbf{A}^\dagger\mathbf{B}^T\boldsymbol{\lambda} + (\mathbf{B}\mathbf{A}^\dagger\mathbf{b} - \mathbf{c}) - \mathbf{B}\mathbf{R}\boldsymbol{\alpha}]_{\mathcal{I}} \leq \mathbf{o}, \tag{2.60a}$$

$$[-\mathbf{B}\mathbf{A}^\dagger\mathbf{B}^T\boldsymbol{\lambda} + (\mathbf{B}\mathbf{A}^\dagger\mathbf{b} - \mathbf{c}) - \mathbf{B}\mathbf{R}\boldsymbol{\alpha}]_{\mathcal{E}} = \mathbf{o}, \tag{2.60b}$$

$$\boldsymbol{\lambda}_{\mathcal{I}}^T[-\mathbf{B}\mathbf{A}^\dagger\mathbf{B}^T\boldsymbol{\lambda} + (\mathbf{B}\mathbf{A}^\dagger\mathbf{b} - \mathbf{c}) - \mathbf{B}\mathbf{R}\boldsymbol{\alpha}]_{\mathcal{I}} = 0. \tag{2.60c}$$

From the other side, let us introduce

$$\Lambda(\boldsymbol{\lambda}, \boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\lambda}^T\mathbf{B}\mathbf{A}^\dagger\mathbf{B}^T\boldsymbol{\lambda} - \boldsymbol{\lambda}^T(\mathbf{B}\mathbf{A}^\dagger\mathbf{b} - \mathbf{c}) + \boldsymbol{\alpha}^T(\mathbf{R}^T\mathbf{B}^T\boldsymbol{\lambda} - \mathbf{R}^T\mathbf{b}), \tag{2.61}$$

$$\mathbf{g} = \nabla_{\boldsymbol{\lambda}}\Lambda(\boldsymbol{\lambda}, \boldsymbol{\alpha}) = \mathbf{B}\mathbf{A}^\dagger\mathbf{B}^T\boldsymbol{\lambda} - (\mathbf{B}\mathbf{A}^\dagger\mathbf{b} - \mathbf{c}) + \mathbf{B}\mathbf{R}\boldsymbol{\alpha}. \tag{2.62}$$

It is the Lagrangian and its gradient corresponding to the equality constrained QP

$$\mathsf{QP}_{2a} = \mathsf{QP}[\ \mathbf{B}\mathbf{A}^\dagger\mathbf{B}^T, \mathbf{B}\mathbf{A}^\dagger\mathbf{b} - \mathbf{c} \mid \mathbf{R}^T\mathbf{B}^T, \mathbf{R}^T\mathbf{b} \mid \square, \square \mid \square, \square\ ]. \tag{2.63}$$

However, we can rewrite the relations (2.60) using $\mathbf{g}$ from (2.62), yielding

$$\mathbf{g}_{\mathcal{I}} \geq \mathbf{o}, \quad \mathbf{g}_{\mathcal{E}} = \mathbf{o}, \quad \text{and} \quad \boldsymbol{\lambda}_{\mathcal{I}}^T\mathbf{g}_{\mathcal{I}} = 0. \tag{2.64}$$

Comparing (2.64) with the special form of KKT conditions (1.26) for bound and equality constrained problems, it is obvious that (2.64) are the KKT conditions for

$$\mathsf{QP}_{2b} = \mathsf{QP}[\ \mathbf{B}\mathbf{A}^\dagger\mathbf{B}^T, \mathbf{B}\mathbf{A}^\dagger\mathbf{b} - \mathbf{c} \mid \mathbf{R}^T\mathbf{B}^T, \mathbf{R}^T\mathbf{b} \mid \square, \square \mid \mathbf{o}, \square, \mathcal{I}\ ], \tag{2.65}$$

with $\boldsymbol{\lambda}$, $\boldsymbol{\alpha}$, and $\boldsymbol{\beta} = \mathbf{g}_{\mathcal{I}}$ being the solution vector, the Lagrange multipliers of the equality constraints $\mathbf{R}^T\mathbf{B}^T = \mathbf{R}^T\mathbf{b}$, and the Lagrange multipliers of the bound constraint $\boldsymbol{\lambda}_{\mathcal{I}} \geq \mathbf{o}$, respectively. The latter follows from (1.27). The solution signature of QP_{2b} reads

$$\overline{\mathsf{QP}}_{2b} = \overline{\mathsf{QP}}[\ \bar{\boldsymbol{\lambda}} \mid \bar{\boldsymbol{\alpha}} \mid \square \mid \bar{\boldsymbol{\beta}}, \square, \mathcal{I}\ ]. \tag{2.66}$$

It suffices to use the notation introduced in (2.54) to get that $\mathsf{QP}_{2b} = \mathsf{QP}_2$. Let us remind that the optimal bound constraint multiplier $\bar{\boldsymbol{\beta}}$ of QP_2 is fully determined by the pair $(\bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\alpha}})$ which can be called the KKT pair of QP_2 as stated in Section 1.5. Note also that in case of QPs without inequality constraints, we get $\mathsf{QP}_2 = \mathsf{QP}_{2b} = \mathsf{QP}_{2a}$.

Finally, let us check that if $(\bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\alpha}})$ is a KKT pair for QP_2, then $(\bar{\mathbf{x}}, \bar{\boldsymbol{\lambda}})$ is a KKT pair for QP_1, where $\bar{\mathbf{x}}$ is defined by (2.58). Presume $(\bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\alpha}})$ is a KKT pair of QP_2. First of all, it of course means that $\bar{\boldsymbol{\lambda}}$ meets the primal feasibility condition with respect to QP_2 so that $\bar{\boldsymbol{\lambda}}_{\mathcal{I}} \geq \mathbf{o}$, and the dual feasibility condition (2.55e) of QP_1 is met. Then the KKT conditions (2.64) hold,

$$\bar{\mathbf{g}}_{\mathcal{I}} \geq \mathbf{o}, \quad \mathbf{g}_{\mathcal{E}} = \mathbf{o}, \quad \bar{\boldsymbol{\lambda}}_{\mathcal{I}}^T\mathbf{g}_{\mathcal{I}} = 0,$$

where

$$\begin{aligned}\bar{\mathbf{g}} = \nabla_{\bar{\boldsymbol{\lambda}}}\Lambda(\bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\alpha}}) &\overset{(2.62)}{=} \mathbf{B}\mathbf{A}^\dagger\mathbf{B}^T\bar{\boldsymbol{\lambda}} - (\mathbf{B}\mathbf{A}^\dagger\mathbf{b} - \mathbf{c}) + \mathbf{B}\mathbf{R}\bar{\boldsymbol{\alpha}} = \\ &= -\mathbf{B}[\mathbf{A}^\dagger(\mathbf{b} - \mathbf{B}^T\bar{\boldsymbol{\lambda}}) - \mathbf{R}\bar{\boldsymbol{\alpha}}] \overset{(2.58)}{=} -\mathbf{B}\bar{\mathbf{x}} + \mathbf{c},\end{aligned}$$

hence

$$[\mathbf{B}\bar{\mathbf{x}} - \mathbf{c}]_{\mathcal{I}} \leq \mathbf{o}, \quad [\mathbf{B}\bar{\mathbf{x}} - \mathbf{c}]_{\mathcal{E}} = \mathbf{o}, \quad \bar{\boldsymbol{\lambda}}_{\mathcal{I}}^T [\mathbf{B}\bar{\mathbf{x}} - \mathbf{c}]_{\mathcal{I}} = 0,$$

which means KKT conditions (2.55b)–(2.55d) of QP_1 are satisfied. To prove the stationarity condition (2.55a), notice that from (2.56) it follows

$$\exists \mathbf{y} \in \mathbb{R}^n : \mathbf{b} - \mathbf{B}^T \bar{\boldsymbol{\lambda}} = \mathbf{A}\mathbf{y}.$$

Thus

$$\begin{aligned} \mathbf{A}\bar{\mathbf{x}} - \mathbf{b} + \mathbf{B}^T \bar{\boldsymbol{\lambda}} &= \mathbf{A}(\mathbf{A}^\dagger(\mathbf{b} - \mathbf{B}^T \bar{\boldsymbol{\lambda}}) - \mathbf{R}\bar{\boldsymbol{\alpha}}) - \mathbf{b} + \mathbf{B}^T \bar{\boldsymbol{\lambda}} = \\ &= \mathbf{A}(\mathbf{A}^\dagger \mathbf{A}\mathbf{y} - \mathbf{R}\bar{\boldsymbol{\alpha}}) - \mathbf{A}\mathbf{y} = \\ &= \mathbf{A}\mathbf{A}^\dagger \mathbf{A}\mathbf{y} - \mathbf{A}\mathbf{y} = \mathbf{o}. \end{aligned}$$

To conclude, we have proven that if $(\bar{\mathbf{x}}, \bar{\boldsymbol{\lambda}})$ is a KKT pair of QP_1, i.e. solves (2.55), then $\bar{\boldsymbol{\lambda}}$ is a feasible vector for QP_2 which satisfies the related KKT conditions (2.64). Reminding that $\mathbf{A}^\dagger$ is SPS by assumption, so that $\mathbf{B}\mathbf{A}^\dagger\mathbf{B}^T$ is also SPS, we can conclude that $\bar{\boldsymbol{\lambda}}$ solves QP_2. Moreover, we have shown that any solution $\bar{\mathbf{x}}$ of QP_1 can be obtained using the formula (2.58) from a KKT pair $(\bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\alpha}})$ of QP_2 (2.54a).

2.4 Examples

This section presents several examples how the particular QP transforms can be used for different types of QP. We make use of the introduced notation with minimal comments. We do not make references to any particular solver here, instead we denote by `Solve` *some* solver which is able to carry out the solution of the given QP.

2.4.1 Equality constrained QP

(a) `HomogenizeEq` and `PreconditionByP`

The equality constraints can be eliminated with the `HomogenizeEq` transform of Section 2.3.3 and the `PreconditionByP` transform of Section 2.3.6. The resulting QP can be solved by any solver for unconstrained minimization, i.e. any SPS linear system solver.

$$\begin{aligned} \mathsf{QP}_1 \quad &= \mathsf{QP}[\ \mathbf{A},\ \mathbf{b} && | \ \mathbf{B}_E,\ \mathbf{c}_E && | \ \square,\ \square \ | \ \square,\ \square\ ]; \\ \\ \mathsf{QP}_2 \quad &= \texttt{HomogenizeEq}(\mathsf{QP}_1) \\ &= \mathsf{QP}[\ \mathbf{A},\ \mathbf{b} - \mathbf{A}\widetilde{\mathbf{x}} && | \ \mathbf{B}_E,\ \mathbf{o} && | \ \square,\ \square \ | \ \square,\ \square\ ], \end{aligned}$$

where
$\widetilde{\mathbf{x}} = \mathbf{B}_E^R \mathbf{c}_E;$

$$\begin{aligned}
\mathsf{QP}_3 &= \texttt{PreconditionByP}(\mathsf{QP}_2) \\
&= \mathsf{QP}[\ \mathbf{PAP},\ \mathbf{P}(\mathbf{b} - \mathbf{A}\widetilde{\mathbf{x}}) \quad | \square,\ \square \quad | \square,\ \square \,|\, \square,\ \square\ ], \\
&\quad \text{where} \\
&\quad \mathbf{P} = \mathbf{I} - \mathbf{B}_E^R \mathbf{B}_E; \\
\overline{\mathsf{QP}}_3 &= \texttt{Solve}(\mathsf{QP}_3) \\
&= \overline{\mathsf{QP}}[\ \bar{\mathbf{x}} \quad | \square \quad | \square \quad | \square,\ \square\ ];
\end{aligned}$$

$$\begin{aligned}
\overline{\mathsf{QP}}_2 &= \overline{\texttt{PreconditionByP}}(\mathsf{QP}_3) \\
&= \overline{\mathsf{QP}}[\ \mathbf{P}\bar{\mathbf{x}} \quad | (\mathbf{B}_E^T)^L(\mathbf{b} - \mathbf{A}\bar{\mathbf{x}}) \,|\, \square \quad | \square,\ \square\ ];
\end{aligned}$$

$$\begin{aligned}
\overline{\mathsf{QP}}_1 &= \overline{\texttt{HomogenizeEq}}(\mathsf{QP}_2) \\
&= \overline{\mathsf{QP}}[\ \mathbf{P}\bar{\mathbf{x}} + \widetilde{\mathbf{x}} \quad | (\mathbf{B}_E^T)^L(\mathbf{b} - \mathbf{A}\bar{\mathbf{x}}) \,|\, \square \quad | \square,\ \square\ ].
\end{aligned}$$

(b) `PenalizeEq`

The equality constraints can be enforced by embedding them into the Hessian with the `PenalizeEq` transform of Section 2.3.4. The resulting QP can be solved by any solver for unconstrained minimization, i.e. any SPS linear system solver. This is simpler to implement then (a) but we get only an approximate KKT pair whose error depends on the penalty ρ and spectral properties of $\mathbf{A}$ and $\mathbf{B}_E$, see Section 2.3.4.

$$\mathsf{QP}_1 = \mathsf{QP}[\ \mathbf{A},\ \mathbf{b} \quad | \mathbf{B}_E,\ \mathbf{c}_E \quad | \square,\ \square \,|\, \square,\ \square\ ];$$

$$\begin{aligned}
\mathsf{QP}_2 &= \texttt{PenalizeEq}(\mathsf{QP}_1) \\
&= \mathsf{QP}[\ \mathbf{A} + \rho \mathbf{B}_E^T \mathbf{B}_E,\ \mathbf{b} + \rho \mathbf{B}_E^T \mathbf{c}_E \,|\, \square,\ \square \quad | \square,\ \square \,|\, \square,\ \square\ ], \\
&\quad \text{where} \\
&\quad \rho > 0;
\end{aligned}$$

$$\begin{aligned}
\overline{\mathsf{QP}}_2 &= \texttt{Solve}(\mathsf{QP}_2) \\
&= \overline{\mathsf{QP}}[\ \bar{\mathbf{x}} \quad | \square \quad | \square \quad | \square,\ \square\ ];
\end{aligned}$$

$$\overline{\mathsf{QP}}_1 = \overline{\texttt{PenalizeEq}}(\overline{\mathsf{QP}}_2)$$

$$= \overline{\mathsf{QP}}[\ \bar{\mathbf{x}} \qquad\qquad\qquad\qquad\qquad\qquad | \ \rho(\mathbf{B}_E\bar{\mathbf{x}} - \mathbf{c}) \ | \ \square \qquad | \ \square,\ \square\ ].$$

To circumvent the need for a proper penalty ρ and for sake of robustness, the `PenalizeEq` transform can be replaced by the more sophisticated algorithm of Section 3.2 which applies the penalty method in the loop with progressive correction of $\boldsymbol{\lambda}_E$.

(c) `Dualize`

Dualization embeds the primal constraints into the dual Hessian. The dual Hessian has the size equal to the number of rows of $\mathbf{B}_E$, and is typically better conditioned than the primal one.

A is SPD. In case $\mathbf{A}$ is SPD, the dual problem is unconstrained and can be solved by any solver for unconstrained minimization, i.e. any SPS linear system solver.

$$\mathsf{QP}_1 = \mathsf{QP}[\ \mathbf{A},\ \mathbf{b} \qquad\qquad\qquad\qquad\qquad | \ \mathbf{B}_E,\ \mathbf{c}_E \ | \ \square,\ \square \ | \ \square,\ \square\ ];$$

$$\begin{aligned}\mathsf{QP}_2 &= \mathtt{Dualize}(\mathsf{QP}_1)\\ &= \mathsf{QP}[\ \mathbf{B}_E\mathbf{A}^{-1}\mathbf{B}_E^T,\ \mathbf{B}_E\mathbf{A}^{-1}\mathbf{b} - \mathbf{c}_E \ | \ \square,\ \square \quad | \ \square,\ \square \ | \ \square,\ \square\ ];\end{aligned}$$

$$\begin{aligned}\overline{\mathsf{QP}}_2 &= \mathtt{Solve}(\mathsf{QP}_2)\\ &= \overline{\mathsf{QP}}[\ \bar{\boldsymbol{\lambda}}_E \qquad\qquad\qquad\qquad | \ \square \qquad | \ \square \qquad | \ \square,\ \square\ ];\end{aligned}$$

$$\overline{\mathsf{QP}}_1 = \overline{\mathsf{QP}}[\ \mathbf{A}^{-1}(\mathbf{b} - \mathbf{B}^T\bar{\boldsymbol{\lambda}}_E) \qquad\qquad | \ \bar{\boldsymbol{\lambda}}_E \qquad | \ \square \qquad | \ \square,\ \square\ ].$$

A is SPS. If $\mathbf{A}$ is only SPS, then the dual problem is equality constrained, since the dual equality constraints arise in the form $\mathbf{R}^T\mathbf{B}_E^T = \mathbf{R}^T\mathbf{b}$, where columns of $\mathbf{R}$ form a base of $\operatorname{Ker}\mathbf{A}$. In this case, the dual QP can be solved by any solver able to solve this kind of problems, or the previous approaches (a) or (b) can be applied to eliminate the equality constraints.

$$\mathsf{QP}_1 = \mathsf{QP}[\ \mathbf{A},\ \mathbf{b} \qquad\qquad\qquad\qquad\qquad | \ \mathbf{B}_E,\ \mathbf{c}_E \qquad | \ \square,\ \square \ | \ \square,\ \square\ ];$$

$$\begin{aligned}\mathsf{QP}_2 &= \mathtt{Dualize}(\mathsf{QP}_1)\\ &= \mathsf{QP}[\ \mathbf{B}_E\mathbf{A}^{\dagger}\mathbf{B}_E^T,\ \mathbf{B}_E\mathbf{A}^{\dagger}\mathbf{b} - \mathbf{c}_E \qquad | \ \mathbf{R}^T\mathbf{B}_E^T,\ \mathbf{R}^T\mathbf{b} \ | \ \square,\ \square \ | \ \square,\ \square\ ];\end{aligned}$$

$$\overline{\mathsf{QP}}_2 = \left\{\begin{array}{l}\mathtt{Solve}(\mathsf{QP}_2),\\ \text{or apply approach (a) or (b)}\end{array}\right\}$$

$$= \overline{\mathsf{QP}}[\ \bar{\boldsymbol{\lambda}}_E \qquad | \ \bar{\boldsymbol{\alpha}} \qquad | \square \qquad | \square,\ \square\ ];$$

$$\overline{\mathsf{QP}}_1 = \overline{\mathsf{QP}}[\ \mathbf{A}^\dagger(\mathbf{b} - \mathbf{B}^T\bar{\boldsymbol{\lambda}}_E) - \mathbf{R}\bar{\boldsymbol{\alpha}} \qquad | \ \bar{\boldsymbol{\lambda}}_E \qquad | \square \qquad | \square,\ \square\ ].$$

Note that the SPD case can be considered a special case of the SPS case with $\mathbf{R} = \square$.

2.4.2 Bound constrained QP

(a) `Dualize`

For bound constrained QP, the dimension of the dual Hessian is equal to the number of constrained variables, so this approach can be efficient when this number is much lower than the dimension of $\mathbf{A}$. We will deal only with the case when $\mathbf{A}$ is SPD. Then the dual formulation contains only partial bound constraints. The primal bound constraints can be converted to general linear inequality constraints $\mathbf{B}\mathbf{x} \leq -\boldsymbol{\ell}$, where

$$\mathbf{B} = \begin{bmatrix} -\mathbf{I}^{(|\mathcal{I}|,|\mathcal{I}|)} & \mathbf{O}^{(|\mathcal{I}|,n-|\mathcal{I}|)} \end{bmatrix}.$$

$$\mathsf{QP}_1 = \mathsf{QP}[\ \mathbf{A},\ \mathbf{b} \qquad | \square,\ \square \,|\, \square,\ \square \,|\, \boldsymbol{\ell},\ \square,\ \mathcal{I}\ ];$$

$$\begin{aligned}
\mathsf{QP}_2 &= \texttt{Dualize}(\mathsf{QP}_1) \\
&= \mathsf{QP}[\ \mathbf{B}\mathbf{A}^{-1}\mathbf{B}^T,\ \mathbf{B}\mathbf{A}^{-1}\mathbf{b} - \mathbf{c} \quad | \square,\ \square \,|\, \square,\ \square \,|\, \mathbf{o},\ \square \quad] \\
&= \mathsf{QP}[\ [\mathbf{A}^{-1}]_{\mathcal{I},\mathcal{I}},\ \mathbf{c}_{\mathcal{I}} - [\mathbf{A}^{-1}]_{\mathcal{I},*}\mathbf{b} \,|\, \square,\ \square \,|\, \square,\ \square \,|\, \mathbf{o},\ \square \quad];
\end{aligned}$$

$$\begin{aligned}
\overline{\mathsf{QP}}_2 &= \texttt{Solve}(\mathsf{QP}_2) \\
&= \overline{\mathsf{QP}}[\ \bar{\boldsymbol{\lambda}}_\ell \qquad | \square \quad | \square \quad | \square,\ \square \quad];
\end{aligned}$$

$$\overline{\mathsf{QP}}_1 = \overline{\mathsf{QP}}[\ \mathbf{A}^{-1}(\mathbf{b} - \mathbf{B}^T\bar{\boldsymbol{\lambda}}_\ell) \qquad | \ \bar{\boldsymbol{\lambda}}_\ell \quad | \square \quad | \square,\ \square \quad].$$

The arising submatrices of $\mathbf{A}^{-1}$ can be expressed using the related Schur complement as

$$[\mathbf{A}^{-1}]_{\mathcal{I},\mathcal{I}} = \left(\mathbf{A}_{\mathcal{I},\mathcal{I}} - \mathbf{A}_{\mathcal{I},\mathcal{R}}\mathbf{A}_{\mathcal{R},\mathcal{R}}^{-1}\mathbf{A}_{\mathcal{I},\mathcal{R}}^T\right)^{-1}, \tag{2.67}$$

$$[\mathbf{A}^{-1}]_{\mathcal{I},*} = \left[[\mathbf{A}^{-1}]_{\mathcal{I},\mathcal{I}}, \quad -[\mathbf{A}^{-1}]_{\mathcal{I},\mathcal{I}}(\mathbf{A}_{\mathcal{R},\mathcal{R}}^{-1}\mathbf{A}_{\mathcal{I},\mathcal{R}}^T)^T\right], \tag{2.68}$$

where $\mathcal{R}$ denotes the index set for which $\mathcal{I} \cup \mathcal{R} = (1, \ldots, n)$, $\mathcal{I} \cap \mathcal{R} = \emptyset$. This form can be useful when $|\mathcal{I}| << |\mathcal{R}|$ as then the matrix $\mathbf{A}_{\mathcal{R},\mathcal{R}}^{-1}\mathbf{A}_{\mathcal{I},\mathcal{R}}^T$, appearing in both (2.67) and (2.68), may be explicitly assembled by solving a linear system with $|\mathcal{I}|$ right hand sides.

2.4.3 Bound and equality constrained QP

(a) `PenalizeEq`

The equality constraints can be enforced by embedding them into the Hessian with the `PenalizeEq` transform of Section 2.3.4. The resulting QP can be solved by any solver for bound constrained minimization. We get only an approximate KKT pair whose error depends on the penalty ρ and spectral properties of $\mathbf{A}$ and $\mathbf{B}_E$, see Section 2.3.4.

$$\mathsf{QP}_1 \quad = \mathsf{QP}[\ \mathbf{A},\ \mathbf{b} \qquad\qquad | \ \mathbf{B}_E,\ \mathbf{c}_E \qquad | \ \square,\ \square \ | \ \ell,\ \square,\ \mathcal{I} \ \ ];$$

$$\begin{aligned}\mathsf{QP}_2 \quad &= \texttt{PenalizeEq}(\mathsf{QP}_1)\\ &= \mathsf{QP}[\ \mathbf{A} + \rho\mathbf{B}_E^T\mathbf{B}_E,\ \mathbf{b} + \rho\mathbf{B}_E^T\mathbf{c}_E \ | \ \square,\ \square \qquad | \ \square,\ \square \ | \ \ell,\ \square,\ \mathcal{I} \ \ ],\\ &\quad \text{where}\\ &\quad \rho > 0;\end{aligned}$$

$$\begin{aligned}\overline{\mathsf{QP}}_2 \quad &= \texttt{Solve}(\mathsf{QP}_2)\\ &= \overline{\mathsf{QP}}[\ \bar{\mathbf{x}} \qquad\qquad\qquad | \ \square \qquad\quad | \ \square \qquad | \ \bar{\boldsymbol{\lambda}}_\ell,\ \square,\ \mathcal{I}\ ];\end{aligned}$$

$$\begin{aligned}\overline{\mathsf{QP}}_1 \quad &= \overline{\texttt{PenalizeEq}}(\overline{\mathsf{QP}}_2)\\ &= \overline{\mathsf{QP}}[\ \bar{\mathbf{x}} \qquad\qquad\qquad | \ \rho(\mathbf{B}_E\bar{\mathbf{x}} - \mathbf{c}) \ | \ \square \qquad | \ \bar{\boldsymbol{\lambda}}_\ell,\ \square,\ \mathcal{I}\ ].\end{aligned}$$

2.4.4 General QP

Dualization embeds the primal constraints into the dual Hessian. The dual Hessian has the size equal to the number of rows of $\mathbf{B}_E$, and is typically better conditioned than the primal one. In case $\mathbf{A}$ is SPD, the dual problem is bound constrained. If $\mathbf{A}$ is only SPS, then the dual problem is bound and equality constrained, since the dual equality constraints arise in the form $\mathbf{R}^T\mathbf{B}_E^T = \mathbf{R}^T\mathbf{b}$, where columns of $\mathbf{R}$ form a base of $\operatorname{Ker}\mathbf{A}$.

$$\mathsf{QP}_1 \quad = \mathsf{QP}[\ \mathbf{A},\ \mathbf{b} \qquad | \ \mathbf{B}_E,\ \mathbf{c}_E \qquad | \ \mathbf{B}_I,\ \mathbf{c}_I \quad | \ \ell,\ \mathbf{u},\ \mathcal{I} \qquad];$$

$$\begin{aligned}\mathsf{QP}_2 \quad &= \texttt{EliminateBox}(\mathsf{QP}_1)\\ &= \mathsf{QP}[\ \mathbf{A},\ \mathbf{b} \qquad | \ \mathbf{B}_E,\ \mathbf{c}_E \qquad | \ \widetilde{\mathbf{B}}_I,\ \widetilde{\mathbf{c}}_I \quad | \ \square,\ \square \qquad],\\ &\quad \text{where}\\ &\quad p = |\mathcal{I}|, \qquad\qquad \mathcal{J}_I = (1, \ldots, m_I),\\ &\quad \mathcal{J}_\ell = (m_I + 1, \ldots, m_I + |\mathcal{I}|), \qquad \mathcal{J}_u = (m_I + |\mathcal{I}| + 1, \ldots, m_I + 2|\mathcal{I}|),\\ &\quad \mathcal{J} = \mathcal{J}_I \cup \mathcal{J}_\ell \cup \mathcal{J}_u,\end{aligned}$$

$$\widetilde{\mathbf{B}}_I = \begin{bmatrix} \mathbf{B}_I \\ -\mathbf{I}^{(p,p)} \quad \mathbf{O}^{(p,n-p)} \\ \mathbf{I}^{(p,p)} \quad \mathbf{O}^{(p,n-p)} \end{bmatrix} \in \mathbb{R}^{|\mathcal{J}|\times n}, \qquad \widetilde{\mathbf{c}}_I = \begin{bmatrix} \mathbf{c}_I \\ -\boldsymbol{\ell}_{\mathcal{I}} \\ \mathbf{u}_{\mathcal{I}} \end{bmatrix} \in \mathbb{R}^{|\mathcal{J}|};$$

$$\begin{aligned}
\mathsf{QP}_3 &= \texttt{Dualize}(\mathsf{QP}_2) \\
&= \mathsf{QP}[\ \mathbf{F},\ \mathbf{d} \quad | \ \mathbf{G},\ \mathbf{e} \quad | \ \square,\ \square \quad | \ \mathbf{o},\ \square,\ \mathcal{J} \quad], \\
&\text{where} \\
\mathcal{E} &= (|\mathcal{J}|+1, \ldots, |\mathcal{J}|+m_E), \\
\mathbf{B} &= \begin{bmatrix} \widetilde{\mathbf{B}}_I \\ \mathbf{B}_E \end{bmatrix} \in \mathbb{R}^{(|\mathcal{J}|+|\mathcal{E}|)\times n}, \qquad \mathbf{c} = \begin{bmatrix} \widetilde{\mathbf{c}}_I \\ \mathbf{c}_E \end{bmatrix} \in \mathbb{R}^{(|\mathcal{J}|+|\mathcal{E}|)}, \\
\mathbf{F} &= \mathbf{B}\mathbf{A}^\dagger\mathbf{B}^T, \qquad \mathbf{d} = \mathbf{B}\mathbf{A}^\dagger\mathbf{b} - \mathbf{c}, \\
\mathbf{G} &= \mathbf{R}^T\mathbf{B}^T, \qquad \mathbf{e} = \mathbf{R}^T\mathbf{b};
\end{aligned}$$

$$\begin{aligned}
\mathsf{QP}_4 &= \texttt{HomogenizeEq}(\mathsf{QP}_3) \\
&= \mathsf{QP}[\ \mathbf{F},\ \widetilde{\mathbf{d}} \quad | \ \mathbf{G},\ \mathbf{o} \quad | \ \square,\ \square \quad | \ -\widetilde{\boldsymbol{\lambda}},\ \square,\ \mathcal{J}\ ], \\
&\text{where} \\
\widetilde{\boldsymbol{\lambda}} &= \mathbf{G}^R\mathbf{e}, \qquad \widetilde{\mathbf{d}} = \mathbf{d} - \mathbf{F}\widetilde{\boldsymbol{\lambda}};
\end{aligned}$$

$$\begin{aligned}
\mathsf{QP}_5 &= \texttt{PreconditionByP}(\mathsf{QP}_4) \\
&= \mathsf{QP}[\ \mathbf{PFF},\ \mathbf{P}\widetilde{\mathbf{d}} \quad | \ \mathbf{G},\ \mathbf{o} \quad | \ \square,\ \square \quad | \ -\widetilde{\boldsymbol{\lambda}},\ \square,\ \mathcal{J}\ ], \\
&\text{where} \\
\mathbf{P} &= \mathbf{I} - \mathbf{G}^R\mathbf{G}; \\
\mathsf{QP}_6 &= \texttt{PenalizeEq}(\mathsf{QP}_5) \\
&= \mathsf{QP}[\ \mathbf{H},\ \mathbf{P}\widetilde{\mathbf{d}} \quad | \ \square,\ \square \quad | \ \square,\ \square \quad | \ -\widetilde{\boldsymbol{\lambda}},\ \square,\ \mathcal{J}\ ], \\
&\text{where} \\
\mathbf{H} &= \mathbf{PFP} + \rho\mathbf{G}^T\mathbf{G};
\end{aligned}$$

$$\begin{aligned}
\overline{\mathsf{QP}}_6 &= \texttt{Solve}(\mathsf{QP}_6) \\
&= \overline{\mathsf{QP}}[\ \widehat{\boldsymbol{\lambda}} \quad | \ \square \quad | \ \square \quad | \ \bar{\boldsymbol{\beta}},\ \square,\ \mathcal{J} \quad], \\
&\text{where} \\
\bar{\boldsymbol{\beta}} &= \left[\mathbf{H}\widehat{\boldsymbol{\lambda}} - \mathbf{P}\widetilde{\mathbf{d}}\right]_{\mathcal{J}};
\end{aligned}$$

$$\overline{\mathsf{QP}}_5 = \overline{\texttt{PenalizeEq}}(\overline{\mathsf{QP}}_6)$$

$$= \overline{\mathsf{QP}}[\ \widehat{\boldsymbol{\lambda}} \quad | \ \widehat{\boldsymbol{\alpha}} \quad | \ \square \quad | \ \bar{\boldsymbol{\beta}},\ \square,\ \mathcal{J}\ ],$$

where

$$\widehat{\boldsymbol{\alpha}} = \rho \mathbf{G} \widehat{\boldsymbol{\lambda}};$$

$$\begin{aligned}\overline{\mathsf{QP}}_4 &= \overline{\texttt{PreconditionByP}}(\overline{\mathsf{QP}}_5)\\ &= \overline{\mathsf{QP}}[\ \widehat{\boldsymbol{\lambda}} \quad | \ \bar{\boldsymbol{\alpha}} \quad | \ \square \quad | \ \bar{\boldsymbol{\beta}},\ \square,\ \mathcal{J}\ ],\end{aligned}$$

where

$$\bar{\boldsymbol{\alpha}} = \widehat{\boldsymbol{\alpha}} + (\mathbf{G}^T)^L(\widetilde{\mathbf{d}} - \mathbf{F}\widehat{\boldsymbol{\lambda}});$$

$$\begin{aligned}\overline{\mathsf{QP}}_3 &= \overline{\texttt{HomogenizeEq}}(\overline{\mathsf{QP}}_4)\\ &= \overline{\mathsf{QP}}[\ \bar{\boldsymbol{\lambda}} \quad | \ \bar{\boldsymbol{\alpha}} \quad | \ \square \quad | \ \bar{\boldsymbol{\beta}},\ \square,\ \mathcal{J}\ ];\end{aligned}$$

where

$$\bar{\boldsymbol{\lambda}} = \widetilde{\boldsymbol{\lambda}} + \widehat{\boldsymbol{\lambda}};$$

$$\begin{aligned}\overline{\mathsf{QP}}_2 &= \overline{\texttt{Dualize}}(\overline{\mathsf{QP}}_3)\\ &= \overline{\mathsf{QP}}[\ \bar{\mathbf{x}} \quad | \ \bar{\boldsymbol{\lambda}}_E \quad | \ \widetilde{\boldsymbol{\lambda}}_I \quad | \ \square,\ \square\ ],\end{aligned}$$

where

$$\bar{\mathbf{x}} = \mathbf{A}^\dagger(\mathbf{b} - \mathbf{B}^T\bar{\boldsymbol{\lambda}}) - \mathbf{R}\bar{\boldsymbol{\alpha}},$$
$$\bar{\boldsymbol{\lambda}}_E = [\bar{\boldsymbol{\lambda}}]_{\mathcal{E}}, \qquad \widetilde{\boldsymbol{\lambda}}_I = [\bar{\boldsymbol{\lambda}}]_{\mathcal{J}};$$

$$\begin{aligned}\overline{\mathsf{QP}}_1 &= \overline{\texttt{EliminateBox}}(\overline{\mathsf{QP}}_2)\\ &= \overline{\mathsf{QP}}[\ \bar{\mathbf{x}} \quad | \ \bar{\boldsymbol{\lambda}}_E \quad | \ \bar{\boldsymbol{\lambda}}_I \quad | \ \bar{\boldsymbol{\lambda}}_\ell,\ \bar{\boldsymbol{\lambda}}_u,\ \mathcal{I}\ ].\end{aligned}$$

where

$$\bar{\boldsymbol{\lambda}}_I = [\widetilde{\boldsymbol{\lambda}}_I]_{\mathcal{J}_I}, \quad \bar{\boldsymbol{\lambda}}_\ell = [\widetilde{\boldsymbol{\lambda}}_I]_{\mathcal{J}_\ell}, \quad \bar{\boldsymbol{\lambda}}_u = [\widetilde{\boldsymbol{\lambda}}_I]_{\mathcal{J}_u},$$

Notes

1. The procedure above remains of course valid for a special case of SPD $\mathbf{A}$. In this case $\mathbf{A}^\dagger = \mathbf{A}^{-1}$, $\mathbf{R} = \square$ and hence $(\mathbf{G}, \mathbf{e}) = (\square, \square)$, and the transforms `HomogenizeEq`, `PreconditionByP` and `PenalizeEq` reduce to identity mappings, i.e. $\mathsf{QP}_6 = \mathsf{QP}_5 = \mathsf{QP}_4 = \mathsf{QP}_3$.
2. Use of `PreconditionByP` is optional but recommended. If only the equality constraints are present (non-empty), this transform already enforces them, see Section 2.3.6 and (a)

in Section 2.4.1; it anyway shrinks the spectrum of the Hessian and simplifies optimal penalization [12].

3. The procedure above allows to solve arbitrary QP, including all the previous special cases, requiring only a solver for bound constrained minimization if any kind of inequality constraints are prescribed, and a solver for unconstrained minimization, i.e. an SPS linear system solver, otherwise.
4. To circumvent the need of a proper penalty ρ and for sake of robustness, the `PenalizeEq` transform can be replaced by the more sophisticated algorithm of Section 3.2 which applies the penalty method in the loop with progressive correction of $\boldsymbol{\lambda}_E$.
5. We will show in Section 4.5 that FETI DDM for both linear and contact problems can be viewed as specializations of the procedure above only with a specific structure of the input data and implementation of the generalized inverse.

CHAPTER III

QP algorithms

This chapter focuses on two key algorithms implemented in PermonQP (Section 7.4): MPRGP for bound constrained QP and SMALBE for bound and equality constrained QP. These algorithms were proposed by Dostál [10]. Their main ideas are presented in the first two sections. The final section presents the author's modifications dealing with issues connected with the termination phase of SMALBE-M.

3.1 MPRGP algorithm

MPRGP (Modified Proportioning and Reduced Gradient Projection) [10, 15, 24] is an efficient algorithm for solution of the convex QP with simple bounds

$$\min \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x} \quad \text{s.t.} \quad \mathbf{x}_{\mathcal{I}} \geq \boldsymbol{\ell}. \tag{3.1}$$

The bounds can be only partial (Section 1.5), $\mathcal{I}$ denotes the index set specifying the solution indices where the bounds are applied. MPRGP can be considered a modification of the Polyak algorithm using active sets. The proportioning algorithm is combined with the gradient projections, a test to decide when to leave the active set, and three types of steps to generate a sequence of iterates $\mathbf{x}^k$ approximating the solution:

1. proportioning step – removes indices from the active set,
2. conjugate gradient (CG) step – generates the next approximation in the active set if the current approximation is proportional (i.e. meeting a special criterion related to chopped, free and reduced free gradients, see [10]),

3. expansion step – defined by the free gradient projection with a fixed steplength $\bar{\alpha}$, expands the active set.

Instead of verifying the KKT conditions directly, the algorithm evaluates the *projected gradient* $\mathbf{g}^P$, given componentwise by

$$\mathbf{g}_i^P = \begin{cases} \mathbf{g}_i & \text{for} \quad x_i > l_i \quad \text{or} \quad i \notin \mathcal{I}, \\ \min(\mathbf{g}_i, 0) & \text{for} \quad x_i = l_i \quad \text{and} \quad i \in \mathcal{I}, \end{cases} \tag{3.2}$$

where x_i and l_i is the i-th component of $\mathbf{x}$ and $\boldsymbol{\ell}$, respectively, and $\mathbf{g} = \mathbf{A}\mathbf{x} - \mathbf{b}$ is the gradient of the objective function. The algorithm stops, when $||\mathbf{g}^P||$ is sufficiently small. MPRGP has a known rate of convergence given in terms of the spectral condition number of the Hessian, comparable to that of Conjugate Gradients applied to the corresponding unconstrained QP [10].

3.2 SMALBE algorithm

SMALBE (Semi-Monotonic Augmented Lagrangian with Bound and Equality) [10] is a variant of the inexact augmented Lagrangian algorithm. It can be used to solve a bound and equality constrained QP

$$\min \ \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x} \quad \text{s.t.} \quad \mathbf{x}_{\mathcal{I}} \geq \boldsymbol{\ell} \quad \text{and} \quad \mathbf{B}_E\mathbf{x} = \mathbf{o}. \tag{3.3}$$

Particularly, the dual QP of the general QP (1.5) takes this form, as discussed in Section 2.3.7. The SMALBE algorithm is based on the outer loop refining the equality constraint Lagrange multipliers $\boldsymbol{\lambda}_E$.

In each outer iteration, the inner loop solving an auxiliary minimization problem

$$\min_{\mathbf{x}} \ L(\mathbf{x}, \boldsymbol{\lambda}_E, \rho) \quad \text{s.t.} \quad \mathbf{x}_{\mathcal{I}} \geq \boldsymbol{\ell} \tag{3.4}$$

is performed, where L is the *augmented Lagrangian* defined as

$$L(\mathbf{x}, \boldsymbol{\lambda}_E, \rho) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x} + \boldsymbol{\lambda}_E^T\mathbf{B}_E\mathbf{x} + \frac{\rho}{2}||\mathbf{B}_E\mathbf{x}||^2.$$

Using just a different bracketing, the inner problem (3.4) is a QP with the penalized Hessian $(\mathbf{A} + \rho\mathbf{B}_E^T\mathbf{B}_E)$ and updated RHS $(\mathbf{b} - \mathbf{B}_E^T\boldsymbol{\lambda}_E)$,

$$\min_{\mathbf{x}} \ \frac{1}{2}\mathbf{x}^T(\mathbf{A} + \rho\mathbf{B}_E^T\mathbf{B}_E)\mathbf{x} - (\mathbf{b} - \mathbf{B}_E^T\boldsymbol{\lambda}_E)^T\mathbf{x} \quad \text{s.t.} \quad \mathbf{x}_{\mathcal{I}} \geq \boldsymbol{\ell}. \tag{3.5}$$

The outer loop also conditionally updates the M parameter (SMALBE-M) or the penalty ρ (SMALBE-ρ) based on the increase of L with respect to $\boldsymbol{\lambda}_E$. SMALBE-M is generally preferred as it uses a fixed ρ and hence the Hessian and its spectrum is not altered. M is divided by the

Algorithm 1 Semi-monotonic augmented Lagrangian method for bound and equality constrained problems with update of M (**SMALBE-M**).

Require: $\varepsilon > 0,\ \eta > 0,\ \beta > 1,\ M_0 > 0,\ \rho > 0,\ \mathbf{x}^0$
Ensure: $\bar{\mathbf{x}} = \arg\min \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x}$ s.t. $\mathbf{x} \geq \boldsymbol{\ell}$ and $\mathbf{B}_E\mathbf{x} = \mathbf{o}$
1: $\boldsymbol{\lambda}_E^0 = \mathbf{o}$; $M_1 = M_0$ {initialization}
2: **for** $k = 1, 2, \ldots$ **do**
3: $\boldsymbol{\lambda}_E^k = \boldsymbol{\lambda}_E^{k-1} + \rho\mathbf{B}_E\mathbf{x}^{k-1}$ {update Lagrange multipliers}
4: find $\mathbf{x}^k \geq \boldsymbol{\ell}$ such that
$\max\left\{||\mathbf{g}^P(\mathbf{x}^k, \boldsymbol{\lambda}_E^k, \rho)||,\ ||\mathbf{B}_E\mathbf{x}^k||\right\} < \varepsilon||\mathbf{b}||$,
or $||\mathbf{g}^P(\mathbf{x}^k, \boldsymbol{\lambda}_E^k, \rho)|| < \min\left\{M_k||\mathbf{B}_E\mathbf{x}^k||,\ \eta\right\}$,
starting with $\mathbf{x}^{k-1}$
{solve the inner bound constrained QP (3.5) with adaptive precision}
5: **if** $\max\left\{||\mathbf{g}^P(\mathbf{x}^k, \boldsymbol{\lambda}_E^k, \rho)||,\ ||\mathbf{B}_E\mathbf{x}^k||\right\} < \varepsilon||\mathbf{b}||$ **then**
6: **return** $\bar{\mathbf{x}} = \mathbf{x}^k$ {solution found, return}
7: **end if**
8: **if** $L(\mathbf{x}^k, \boldsymbol{\lambda}_E^k, \rho) < L(\mathbf{x}^{k-1}, \boldsymbol{\lambda}_E^{k-1}, \rho) + \rho||\mathbf{B}_E\mathbf{x}^k||^2/2$ **then**
9: $M_{k+1} = M_k/\beta$ {update the balancing parameter M}
10: **else**
11: $M_{k+1} = M_k$
12: **end if**
13: **end for**

update constant $\beta > 1$ if the increase of L with respect to $\boldsymbol{\lambda}_E$ is not sufficient. The SMALBE-M algorithm for bound and equality constrained QP is presented here as Algorithm 1.

The inner loop (Line 4 in Algorithm 1) may be implemented by any algorithm which is able to solve the bound constrained QP (3.5), such as MPRGP (Section 3.1). The inner solver has to obey the special stopping criterion (also shown in Line 4) assessing the norm of the projected gradient $\mathbf{g}^P$ (3.2), obtained now from the gradient of L with respect to $\mathbf{x}$

$$\mathbf{g}(\mathbf{x}, \boldsymbol{\lambda}_E, \rho) = (\mathbf{A} + \rho\mathbf{B}_E^T\mathbf{B}_E)\mathbf{x} - (\mathbf{b} - \mathbf{B}_E^T\boldsymbol{\lambda}_E). \tag{3.6}$$

Algorithm 1 is a slightly modified version of Algorithm 6.2 in [10]. First of all, the update of the Lagrange multipliers in line 3 has been moved to the beginning of the loop. This modification has two advantages: (1) $\mathbf{x}^0$ is a parameter whereas $\boldsymbol{\lambda}_E^0$ is given which is more natural, (2) only two generations of $\boldsymbol{\lambda}_E$ at a time are needed (easier implementation).

Furthermore, the stopping criterion in line 4 is specified more in detail here because it is important to stop the inner loop if the solution of the original ("outer") QP is already found. In [10], the full stopping criterion is discussed separately in Section 6.10. However, we intentionally do not mention here the part of the stopping criterion checking whether the projected gradient $\mathbf{g}^P$ alone has already reached the desired precision,

$$\textbf{or} \quad ||\mathbf{g}^P(\mathbf{x}^k, \boldsymbol{\lambda}_E^k, \rho)|| < \varepsilon||\mathbf{b}||, \tag{3.7}$$

because it has turned out to be troublesome and should be handled carefully. We will discuss these issues and propose solution in Section 3.3.

Here we give hints of "reasonable" values of the parameters:

$$\begin{aligned}
\varepsilon &\in [10^{-10}, 10^{-5}] \text{ (required tolerance)},\\
\eta &= m_\eta ||\mathbf{b}||,\ m_\eta \in [10^{-2}, 10^{0}],\\
\beta &\in [2, 10],\\
M_0 &= m_M ||\mathbf{A}||,\ m_M \in [10^{-1}, 10^{3}],\\
\rho &= m_\rho ||\mathbf{A}||,\ m_\rho \in [1, 10].
\end{aligned}$$

Compared to the basic penalty method, SMALBE is able to find an approximate KKT pair meeting the given precision with no need for a large penalty, avoiding ill-conditioning. Compared to the basic augmented Lagrangian method, the introduced adaptivity weakens the need of the proper selection of the penalty sequence and eliminates necessity of exact solution of the inner problems. Optimality results were presented in [10, 13, 16, 17].

3.3 SMALBE-M with improved termination phase

The basic form of SMALBE-M (Algorithm 1, Section 3.2) has a hidden imperfection, revealed by experimenting with our PERMON solvers (Chapter 7). Let us show this issue on a small problem of elastic cube generated by PermonCube (Section 7.1). Figure 3.1 shows the progress of $||\mathbf{g}^P||$ and $||\mathbf{B}_E\mathbf{x}||$. We can see the solution approximation $\mathbf{x}^k$ satisfying

$$\max\left\{||\mathbf{g}^P(\mathbf{x}^k, \boldsymbol{\lambda}_E^k, \rho)||,\ ||\mathbf{B}_E\mathbf{x}^k||\right\} < \varepsilon||\mathbf{b}||$$

is found much later than the approximation $\mathbf{x}^j$, $j < k$, satisfying just

$$||\mathbf{g}^P(\mathbf{x}^j, \boldsymbol{\lambda}_E^j, \rho)|| < \varepsilon||\mathbf{b}||.$$

Let us now add a standalone stopping criterion

$$||\mathbf{g}^P(\mathbf{x}^k, \boldsymbol{\lambda}_E^k, \rho)|| < \varepsilon||\mathbf{b}||$$

to the algorithm and denote by K the first index for which the relation becomes true. In Figure 3.2, we can see it is not suitable to include this criterion without any special handling. In each outer iteration with the index greater than K, the inner solver performs only one step of MPRGP and stops because the criterion is met again. The result is the inner solver makes not enough iterations to reduce the equality constraint violation norm $||\mathbf{B}_E\mathbf{x}||$ after the update of the multipliers $\boldsymbol{\lambda}_E$ in the outer loop. Thus $||\mathbf{B}_E\mathbf{x}||$ decreases only very slowly. To get it under the desired tolerance, even more total inner iterations then before are needed.

Let us try to resolve this issue by enforcing some minimal number of inner steps N after reaching the desired tolerance with the gradient. Figures 3.3 to 3.5 show the resulting behaviour. With the $N = 20$ (Figure 3.5), we have already got a slightly better result compared to the basic approach of Figure 3.1. However, the improvement is not convincing and the optimal N is problem-specific.

There is a tool that we have not exploited so far – update of the penalty ρ. Although it worsens the condition number of the penalized Hessian, it is valuable in situations when the decrease of $||\mathbf{B}_E\mathbf{x}||$ needs to be accelerated. It is a good idea to start to increase ρ as late as in the K-th step, and do it carefully with a reasonable update factor β. Figure 3.6 presents behaviour of the algorithm for $\beta = 2$. We can see that with this setting, $||\mathbf{g}^P||$ starts to slightly oscillate around the value of ε, and $||\mathbf{B}_E\mathbf{x}||$ decreases rapidly to this value, taking just a few further iterations with $k > K$.

Is is also advisable to limit the number of outer iterations to some finite value $k_{\max}$ to avoid an infinite loop in pathological cases. We have always used $k_{\max} = 100$. The parameter M should never decrease below the value E/ε where E is the "machine epsilon"[1] and we recommend to bound it with this value or greater. The resulting algorithm with the late update of the penalty and with the limits discussed above is presented here as Algorithm 2.

[1] $E \approx 1.11 \times 10^{16}$ for double precision.

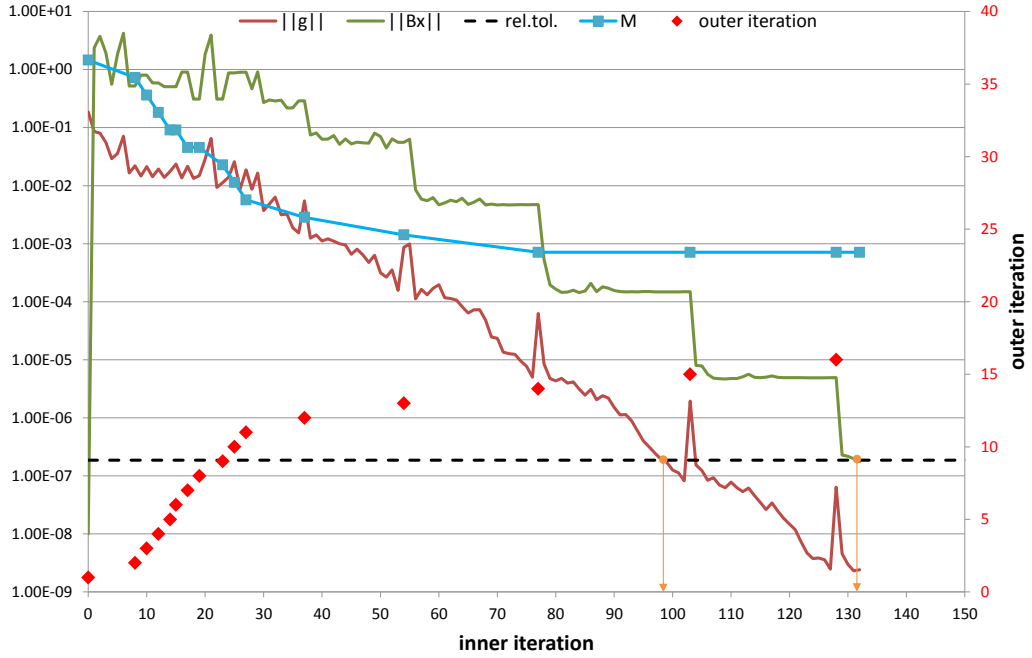


Figure 3.1: The issue of the late termination.

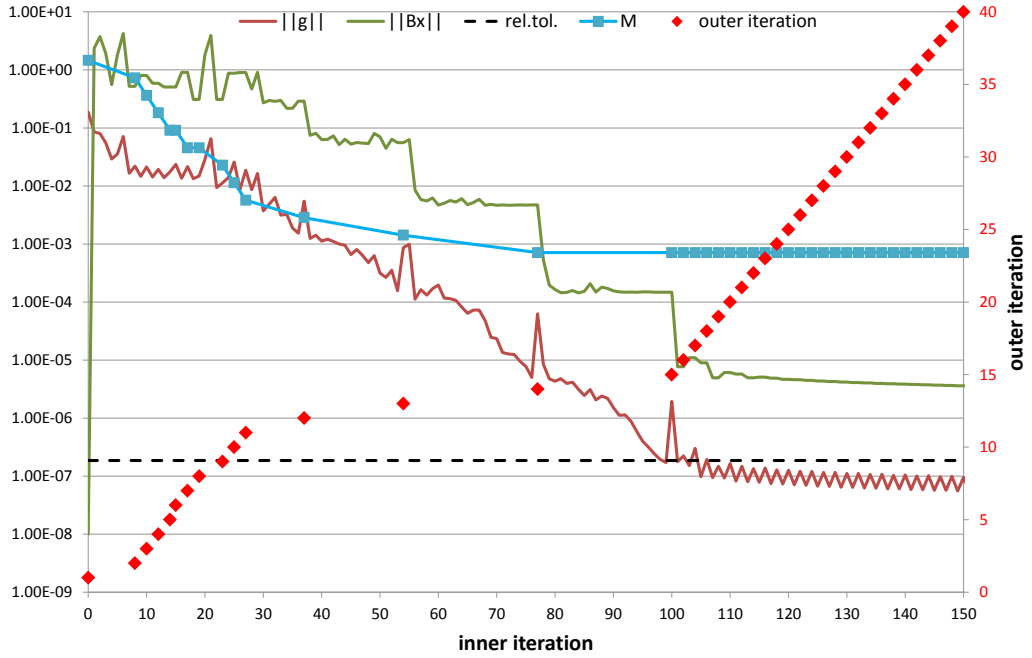


Figure 3.2: The issue caused by the criterion $||\mathbf{g}^P(\mathbf{x}^k, \boldsymbol{\lambda}_E^k, \rho)|| < \varepsilon||\mathbf{b}||$.

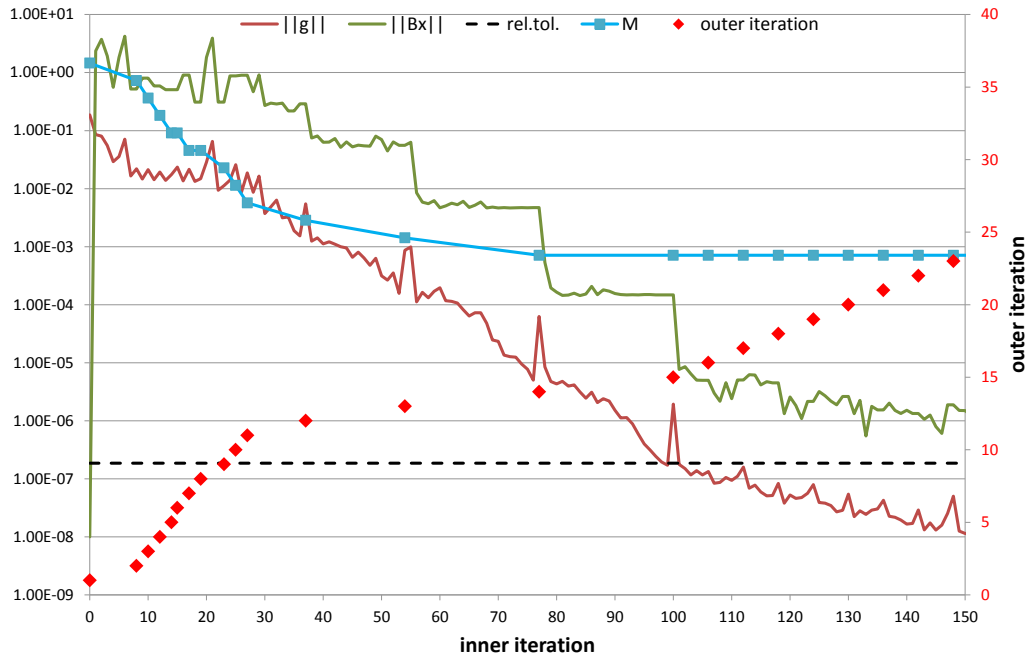


Figure 3.3: Behaviour of $||\mathbf{B}_E\mathbf{x}||$ for the minimal number of inner steps $N = 5$.

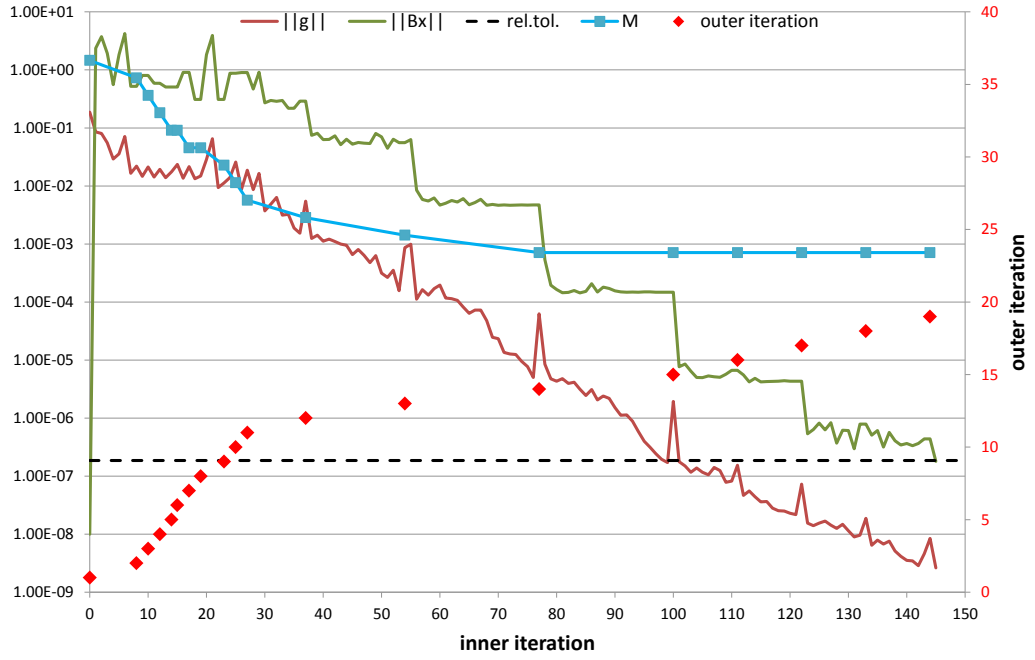


Figure 3.4: Progress for $N = 10$.

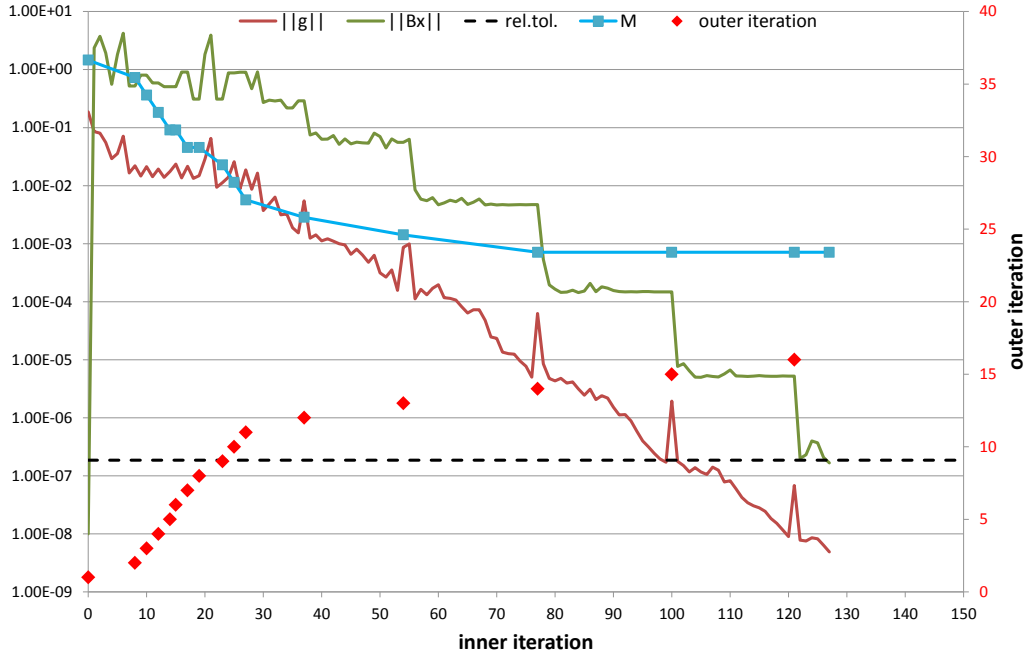


Figure 3.5: Progress for $N = 20$.

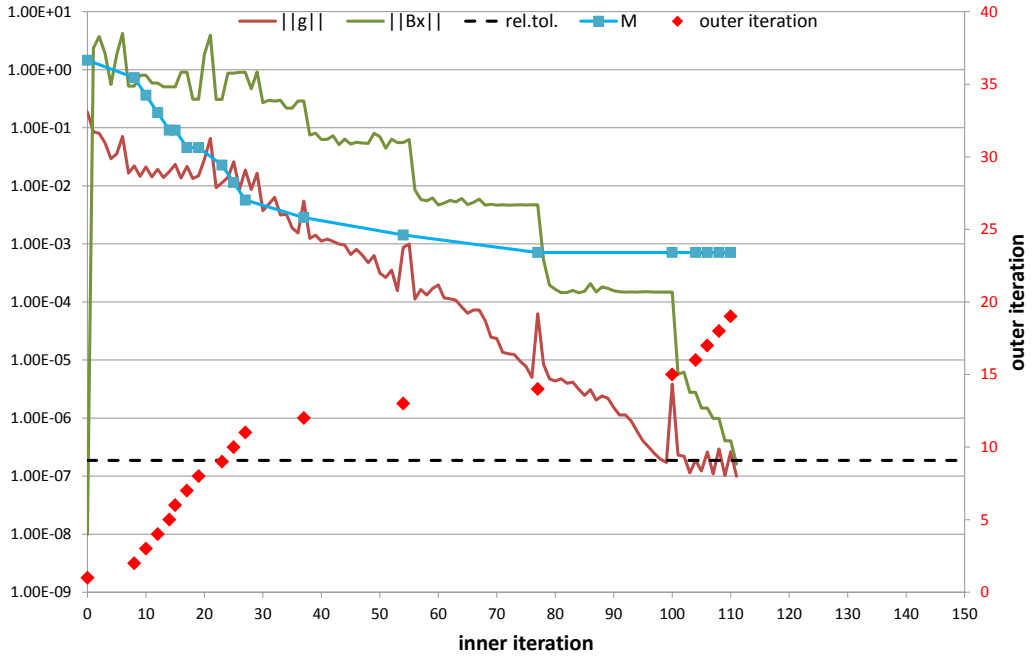


Figure 3.6: The issue is resolved with the late update of penalty.

Algorithm 2 SMALBE-M with improved termination

Require: $\mathbf{x}^0,\ \boldsymbol{\lambda}_E^0,\ \varepsilon > 0,\ \eta > 0,\ k_{\max} > 0,$
$M_0 > 0,\ \beta_M > 1,\ M_{\min} \in (0,1),$
$\rho_0 > 0,\ \beta_\rho > 1$

Ensure: $\bar{\mathbf{x}} = \arg\min \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x}$ s.t. $\mathbf{x} \geq \boldsymbol{\ell}$ and $\mathbf{B}_E\mathbf{x} = \mathbf{o}$

$\boldsymbol{\lambda}_E^0 = \mathbf{o}$; $\rho_1 = \rho_0$; $M_1 = M_0$ {initialization}
$S = 1$ {set state to 1}
for $k = 1, 2, \ldots, k_{\max}$ **do**
  $\boldsymbol{\lambda}_E^k = \boldsymbol{\lambda}_E^{k-1} + \rho_{k-1}\mathbf{B}_E\mathbf{x}^{k-1}$ {update Lagrange multipliers}
  find $\mathbf{x}^k \geq \boldsymbol{\ell}$ such that
  $\max\left\{||\mathbf{g}^P(\mathbf{x}^k, \boldsymbol{\lambda}_E^k, \rho_k)||,\ ||\mathbf{B}_E\mathbf{x}^k||\right\} < \varepsilon||\mathbf{b}||,$
  or $||\mathbf{g}^P(\mathbf{x}^k, \boldsymbol{\lambda}_E^k, \rho_k)|| < \max\left\{\min\left\{M_k||\mathbf{B}_E\mathbf{x}^k||,\ \eta\right\},\ \varepsilon||\mathbf{b}||\right\},$
  starting with $\mathbf{x}^{k-1}$
    {solve the inner bound constrained QP with adaptive precision}
  if $\max\left\{||\mathbf{g}^P(\mathbf{x}^k, \boldsymbol{\lambda}_E^k, \rho_k)||,\ ||\mathbf{B}_E\mathbf{x}^k||\right\} < \varepsilon||\mathbf{b}||$ **then**
    return $\bar{\mathbf{x}} = \mathbf{x}^k$ {solution found, return}
  end if
  if $||\mathbf{g}^P(\mathbf{x}^k, \boldsymbol{\lambda}_E^k, \rho_k)|| < \varepsilon||\mathbf{b}||$ **then**
    $S = 3$ {change state to 3}
  end if
  if $S == 1$ **then**
    if $L(\mathbf{x}^k, \boldsymbol{\lambda}_E^k, \rho_k) < L(\mathbf{x}^{k-1}, \boldsymbol{\lambda}_E^{k-1}, \rho_{k-1}) + \rho_k||\mathbf{B}_E\mathbf{x}^k||^2/2$ **then**
      $M_{k+1} = M_k/\beta_M$ {update the balancing parameter M}
    else
      $M_{k+1} = M_k$
    end if
    if $M_{k+1}/\beta < M_{\min}$ **then**
      $S = 2$ {change state to 2}
    end if
      $\rho_{k+1} = \rho_k$ {ρ is not updated in state 1}
  else if $S == 2$ **then**
    $M_{k+1} = M_k$; $\rho_{k+1} = \rho_k$ {nothing is updated in state 2}
  else if $S == 3$ **then**
    $M_{k+1} = M_k$ {M is not updated in state 3}
    $\rho_{k+1} = \beta_\rho\rho_k$ {update the penalty ρ unconditionally}
  end if
end for

CHAPTER IV

TFETI DDM

FEM simulations with a large enough number of variables are not solvable on usual personal computers due to memory and computational limits. Such problems can be solved only on parallel computers. Suitable numerical methods are needed for that such as domain decomposition methods (DDMs) or multigrid. DDMs provide a very natural way of parallelism as they solve the original problem by decomposing ("tearing") into smaller independent subdomain problems. A DDM basically constitutes a mathematical framework how to get a correct solution, continuous across subdomain interfaces, in presence of such tearing.

Finite Element Tearing and Interconnecting (FETI) methods form a successful subclass of DDMs. They belong to non-overlapping methods and combine iterative and direct solvers. The FETI methods allow highly accurate computations scaling up to tens of thousands of processors and billions of unknowns. In FETI, subdomain stiffness matrices are assembled and factorized independently, and so are the subsequent triangular systems solved, whereas conditions of continuity of the solution across subdomain interfaces (gluing conditions) form separate linear equality constraints which couple the subdomains' solutions in a neigbour-wise manner. In the specific flavour of FETI we typically use, called Total FETI (TFETI), Dirichlet boundary conditions are enforced by means of equality constraints, too. Hence, even in case of linear elasticity, it is advantageous to handle the resulting problem as a special case of QP (equality constrained).

This chapter serves as a brief introduction to the DDMs of the FETI type. It is mostly compilation but presents a new view on FETI as a chain of QP transforms in Section 4.5.

4.1 Non-overlapping domain decomposition

Non-overlapping DDMs are based on decomposing the original spatial domain into non-overlapping subdomains. A non-overlapping DDM consists of three basic parts:

1. the meshing part,
2. the assembly part,
3. the algebraic part.

For a very nice overview of non-overlapping DDMs, see [26].

4.1.1 Meshing part

Parallel mesh generation of the complex geometry is one of the open problems from the last decade. There are two basic approaches [84]:

1. Parallelize directly the mesh generation algorithm.
2. Decompose the geometry into partitions, for each partition use a sequential mesh generator.

An obvious advantage of the second approach is that existing high quality sequential meshers such as Netgen (see Section 6.1.1) can be used. More about this topic together with a survey of existing codes implementing parallel mesh generation can be found in [84].

For non-overlapping DDMs, the submeshes of the global mesh are handled completely separately; there are neither overlapping cells nor a ghost-layer. The degrees of freedom (DOFs) on submesh interfaces are duplicated into each intersecting submesh, i.e. each submesh is "complete" and "self-contained". Volume meshing can then be done completely separately for each submesh.

4.1.2 Conformity conditions

We restrict ourselves to the case of element-oriented domain decompositions (each element belongs to one and only one subdomain) which are conforming to the mesh, satisfying these *conformity conditions* [26, 63]:

- there is a one-to-one correspondence between DOFs along the subdomain interfaces;
- approximation spaces are the same along the subdomain interfaces;
- FE models (beam, shell etc.) are the same along the subdomain interfaces.

4.1.3 DOF numberings

Let us introduce several DOF numberings, useful for further discussions:

1. *global* – a unique global DOF numbering before the DOF duplication connected with the non-overlapping domain decomposition,
2. *local* – after the decomposition and DOF duplication, DOFs of each subdomain are numbered starting from 0 independently of other subdomains,
3. *interface* – similar to local, but only the DOFs residing on subdomain interfaces are numbered.

4.1.4 Assembly part

The second part, the assembly of algebraic objects, is done completely separately for each submesh. The stiffness matrix and load vector resulting from a submesh form a "local problem". The local problems are independent so that the global stiffness matrix is block-diagonal. Thus the assembly process can be done by any sequential FEM code for all submeshes at once in parallel.

4.1.5 Algebraic part

The final "algebraic" part is essentially a mathematical approach how to deal with the non-overlapping mesh decomposition described above. This part is what is actually typically called DDM. To "glue" the subdomains together, a mapping between interfaces is needed. If the conformity conditions of Section 4.1.2 have been met in the meshing part, this mapping is just a many-to-one mapping from the *interface* numbering to the *global* numbering. We call it *local-to-global mapping (l2g)*. This is a minimal additional information needed by the DDM solver in contrast to mainstream linear system solvers. The mainstream solvers work with the monolithic global stiffness matrix coming from the "undecomposed" mesh, and their parallelization consists in use of a distributed matrix storage (e.g. `MPIAIJ` in PETSc) for the stiffness matrix.

4.2 FETI methods

The class of methods called FETI (Finite Element Tearing and Interconnecting) turned out to be one of the most successful for parallel solution of elliptic partial differential equations arising from many engineering problems. FETI methods use dual formulation. In the original primal problem, the unknowns represent displacements, and "gluing" equality constraints express connectivity between subdomains. Elimination of the primal variables reduces the original problem to a smaller, better conditioned one. Its unknowns are Lagrange multipliers of

the above-mentioned "gluing" constraints. These multipliers "glue together" the subdomains so that the computed primal displacements are continuous across the subdomain interfaces.

FETI-1 [20, 21] is a non-overlapping domain decomposition method. The original FETI-1 method assumes that the boundary subdomains inherit Dirichlet conditions from the original problem where the conditions are embedded into the linear system arising from FEM. This actually means that subdomains whose interfaces intersect the Dirichlet boundary are fixed while others are kept floating; in the linear algebra speech, the corresponding subdomain stiffness matrices are non-singular and singular, respectively. Dimensions of kernels of subdomain stiffness matrices may vary from zero to a certain positive maximum. They correspond to two extremes: boundary subdomains with sufficient Dirichlet data to fix their rigid body motions, and to freely floating subdomains, respectively.

The basic idea of the Total-FETI (TFETI) method [86, 8, 88, 94] is to keep all the subdomains floating and enforce the Dirichlet boundary conditions by means of a constraint matrix and Lagrange multipliers, similarly to the above-mentioned gluing conditions. This simplifies implementation of Dirichlet conditions as well as the stiffness matrix pseudoinverse. The key point is that kernels of subdomain stiffness matrices are known a priori, have the same dimension and can be formed without any computation from the mesh data. Furthermore, each local stiffness matrix can be regularized cheaply, and the inverse of the resulting nonsingular matrix is at the same time a pseudoinverse of the original singular one [5, 9, 39].

A contact problem of mechanics leads to a primal QP with linear inequality constraints, representing non-penetration conditions. If the dualization, being part of the FETI procedure, is applied, then these general linear inequality constraints are transformed into simple lower bound constraints which allow use of specialized QP solvers for this type of problems. We combine the TFETI method [8] with optimal QP algorithms by Dostál et al [10, 11, 17]. A unique feature of this approach is theoretically supported numerical scalability and high parallel scalability.

4.3 TFETI for linear elasticity

Let us consider a partitioning of the global domain Ω into N_S subdomains Ω^s. Using subdomain-wise FE discretization, we get subdomain stiffness matrices $\mathbf{K}^s \in \mathbb{R}^{n_s \times n_s}$ and subdomain load vectors $\mathbf{f}^s \in \mathbb{R}^{n_s}$, where n_s denotes number of DOFs within the submesh corresponding to the subdomain Ω^s. Our goal is to find unknown displacements $\mathbf{u}^s \in \mathbb{R}^{n_s}$ of each subdomain. The global stiffness matrix $\mathbf{K}$, global load vector $\mathbf{f}$ and global displacement vector $\mathbf{u}$ then possess

the following layout

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}^1 & & \\ & \ddots & \\ & & \mathbf{K}^{N_S} \end{bmatrix} \in \mathbb{R}^{n \times n}, \quad \mathbf{f} = \begin{bmatrix} \mathbf{f}^1 \\ \vdots \\ \mathbf{f}^{N_S} \end{bmatrix} \in \mathbb{R}^n, \quad \mathbf{u} = \begin{bmatrix} \mathbf{u}^1 \\ \vdots \\ \mathbf{u}^{N_S} \end{bmatrix} \in \mathbb{R}^n. \tag{4.1}$$

Here, $n = \sum_{s=1}^{N_S} n_s$ stands for the global number of DOFs of the decomposed problem, including DOFs duplicated across subdomain interfaces. This count is called *primal dimension*. Note that the block-diagonal layout of $\mathbf{K}$ is the main source of parallelism within the FETI methods.

In case of the conformal domain decomposition (see Section 4.1.2), connectivity of the subdomains can be prescribed easily with the interface compatibility ("gluing") condition

$$\mathbf{B}_g \mathbf{u} = \mathbf{o}, \tag{4.2}$$

where $\mathbf{B}_g \in \mathbb{R}^{m_g \times n}$ is a signed Boolean matrix, consisting again of subdomain-wise blocks $\mathbf{B}_g^s$,

$$\mathbf{B}_g = \begin{bmatrix} \mathbf{B}_g^1 & \dots & \mathbf{B}_g^{N_S} \end{bmatrix}. \tag{4.3}$$

Note that for non-conformal domain decomposition, we can proceed similarly using mortar methods [82] but the matrix $\mathbf{B}_g$ will no longer be signed Boolean and will contain substantially more nonzeros.

Using the TFETI approach, the subdomains coinciding with the Dirichlet boundary are treated as floating, their stiffness matrices are not modified in the FEM phase and remain singular, and "gluing to the Dirichlet boundary" is prescribed in the same way as the gluing condition above,

$$\mathbf{B}_d \mathbf{u} = \mathbf{c}_d, \tag{4.4}$$

where $\mathbf{B}_d = \begin{bmatrix} \mathbf{B}_d^1 & \dots & \mathbf{B}_d^{N_S} \end{bmatrix} \in \mathbb{R}^{m_d \times n}$ is a Boolean matrix and $\mathbf{c}_d^T = \begin{bmatrix} (\mathbf{c}_d^1)^T & \dots & (\mathbf{c}_d^{N_S})^T \end{bmatrix} \in \mathbb{R}^{m_d}$ contains prescribed displacements on the Dirichlet boundary. Obviously, $\mathbf{B}_d^s \neq \mathbf{O}$ if and only if Ω^s intersects with the Dirichlet boundary. We can rewrite the two conditions (4.3) and (4.4) into one,

$$\mathbf{B}\mathbf{u} = \mathbf{c}, \tag{4.5}$$

where

$$\mathbf{B} = \begin{bmatrix} \mathbf{B}_g \\ \mathbf{B}_d \end{bmatrix} \in \mathbb{R}^{m \times n}, \qquad \mathbf{c} = \begin{bmatrix} \mathbf{o}^{(m_g)} \\ \mathbf{c}_d \end{bmatrix} \in \mathbb{R}^m, \qquad m = m_g + m_d.$$

The total number of scalar constraints m is called *dual dimension*. Note that any other types of equality constraints, e.g. multipoint constraints [41], can be handled analogously.

Discretization and decomposition process discussed above leads to the primal QP

$$\min \frac{1}{2}\mathbf{u}^T \mathbf{K}\mathbf{u} - \mathbf{f}^T \mathbf{u} \quad \text{s.t.} \quad \mathbf{B}\mathbf{u} = \mathbf{c}. \tag{4.6}$$

We can write the Lagrangian associated with the problem (4.6) as

$$L(\mathbf{u}, \boldsymbol{\lambda}) = \frac{1}{2}\mathbf{u}^T\mathbf{K}\mathbf{u} - \mathbf{f}^T\mathbf{u} + \boldsymbol{\lambda}^T\mathbf{B}\mathbf{u}, \tag{4.7}$$

where $\boldsymbol{\lambda} = \begin{bmatrix} \boldsymbol{\lambda}_g \\ \boldsymbol{\lambda}_d \end{bmatrix} \in \mathbb{R}^m$ is a vector of unknown Lagrange multipliers with subvectors $\boldsymbol{\lambda}_g \in \mathbb{R}^{m_g}$ and $\boldsymbol{\lambda}_d \in \mathbb{R}^{m_d}$ corresponding to gluing conditions and Dirichlet conditions, respectively. It is well known that (4.6) is equivalent to the saddle point problem

$$(\bar{\mathbf{u}}, \bar{\boldsymbol{\lambda}}) = \arg \inf_{\mathbf{u}} \sup_{\boldsymbol{\lambda}} L(\mathbf{u}, \boldsymbol{\lambda}). \tag{4.8}$$

After eliminating the primal variables $\mathbf{u}$, we get the dual minimization problem

$$\min \Theta(\boldsymbol{\lambda}) \quad \text{s.t.} \quad \mathbf{R}^T(\mathbf{f} - \mathbf{B}^T\boldsymbol{\lambda}) = \mathbf{o}, \tag{4.9}$$

where

$$\Theta(\boldsymbol{\lambda}) = \frac{1}{2}\boldsymbol{\lambda}^T\mathbf{B}\mathbf{K}^\dagger\mathbf{B}^T\boldsymbol{\lambda} - \boldsymbol{\lambda}^T\mathbf{B}\mathbf{K}^\dagger\mathbf{f}.$$

Let us remind that $\mathbf{K}^\dagger$ denotes a generalized inverse that satisfies $\mathbf{K}\mathbf{K}^\dagger\mathbf{K} = \mathbf{K}$.

The block-diagonal matrix $\mathbf{R}$ consists of subdomain-wise blocks $\mathbf{R}^s$. For each $s = 1, \ldots, N_S$, columns of $\mathbf{R}^s \in \mathbb{R}^{n_s \times d_s}$ form a base of the kernel of the subdomain stiffness matrix $\mathbf{K}^s$, where d_s is the dimension of the kernel. The global matrix $\mathbf{R} \in \mathbb{R}^{n \times d}$, $d = \sum_{s=1}^{N_S} d_s$, inherits this property, i.e. columns of the matrix $\mathbf{R}$ form a base of the kernel of $\mathbf{K}$ with the dimension d. The local kernel dimension d_s can be e.g. for Poisson problem $d_s = 1$, for a problem of 2D elasticity $d_s = 3$, and for a problem 3D elasticity $d_s = 6$, provided TFETI is used and the decomposition has produced only simply connected domains.

Let us denote

$$\mathbf{F} = \mathbf{B}\mathbf{K}^\dagger\mathbf{B}^T, \quad \mathbf{G} = \mathbf{R}^T\mathbf{B}^T, \quad \mathbf{e} = \mathbf{R}^T\mathbf{f}, \quad \widetilde{\mathbf{d}} = \mathbf{B}\mathbf{K}^\dagger\mathbf{f}.$$

The problem (4.9) in the new notation reads

$$\min \frac{1}{2}\boldsymbol{\lambda}^T\mathbf{F}\boldsymbol{\lambda} - \boldsymbol{\lambda}^T\widetilde{\mathbf{d}} \quad \text{s.t} \quad \mathbf{G}\boldsymbol{\lambda} = \mathbf{e}. \tag{4.10}$$

We can now proceed by homogenizing the equality constraints, i.e. transform the problem (4.10) to minimization on the kernel of $\mathbf{G}$. Let $\widetilde{\boldsymbol{\lambda}}$ satisfy

$$\mathbf{G}\widetilde{\boldsymbol{\lambda}} = \mathbf{e}. \tag{4.11}$$

It is natural, though not necessary, to use the least-square solution

$$\widetilde{\boldsymbol{\lambda}} = \mathbf{G}^R\mathbf{e}. \tag{4.12}$$

The solution of (4.10) can then be sought in the form

$$\boldsymbol{\lambda} = \boldsymbol{\mu} + \widetilde{\boldsymbol{\lambda}}. \tag{4.13}$$

Since

$$\frac{1}{2}\boldsymbol{\lambda}^T\mathbf{F}\boldsymbol{\lambda} - \boldsymbol{\lambda}^T\widetilde{\mathbf{d}} = \frac{1}{2}\boldsymbol{\mu}^T\mathbf{F}\boldsymbol{\mu} - \boldsymbol{\mu}^T(\widetilde{\mathbf{d}} - \mathbf{F}\widetilde{\boldsymbol{\lambda}}) + \frac{1}{2}\widetilde{\boldsymbol{\lambda}}^T\mathbf{F}\widetilde{\boldsymbol{\lambda}} - \widetilde{\boldsymbol{\lambda}}^T\widetilde{\mathbf{d}},$$

and the last two terms are constant with respect to $\boldsymbol{\mu}$, the problem (4.10) is equivalent to

$$\min \frac{1}{2}\boldsymbol{\mu}^T\mathbf{F}\boldsymbol{\mu} - \boldsymbol{\mu}^T\mathbf{d} \quad \text{s.t} \quad \mathbf{G}\boldsymbol{\mu} = \mathbf{o}, \tag{4.14}$$

where $\mathbf{d} = \widetilde{\mathbf{d}} - \mathbf{F}\widetilde{\boldsymbol{\lambda}}$. Let us return to the old notation, $\boldsymbol{\lambda} := \boldsymbol{\mu}$.

Our final step is based on observation that the problem (4.14) is equivalent to

$$\min \frac{1}{2}\boldsymbol{\lambda}^T\mathbf{PFP}\boldsymbol{\lambda} - \boldsymbol{\lambda}^T\mathbf{P}\mathbf{d} \quad \text{s.t.} \quad \mathbf{G}\boldsymbol{\lambda} = \mathbf{o}, \tag{4.15}$$

where

$$\mathbf{Q} = \mathbf{G}^T(\mathbf{G}\mathbf{G}^T)^{-1}\mathbf{G} \qquad \text{and} \qquad \mathbf{P} = \mathbf{I} - \mathbf{Q}$$

are orthogonal projectors onto $\operatorname{Im}\mathbf{G}^T$ and $\operatorname{Ker}\mathbf{G}$, respectively. Within the FETI methodology, $\operatorname{Ker}\mathbf{G}$ is called *natural coarse space*, and the action of $(\mathbf{G}\mathbf{G}^T)^{-1}$ is called *coarse problem*.

Note that in this case of equality constrained QP, the introduction of the projector $\mathbf{P}$ causes $\boldsymbol{\lambda} \in \operatorname{Ker}\mathbf{G}$, i.e. the homogeneous equality constraints are automatically satisfied. Hence, we get unconstrained QP which is equivalent to the linear system

$$\mathbf{PFP}\boldsymbol{\lambda} = \mathbf{P}\mathbf{d}. \tag{4.16}$$

This final problem can be solved with an arbitrary linear system solver. If a Krylov subspace method is used, then the right $\mathbf{P}$ in the Hessian can be omitted. The conjugate gradient method is a good choice thanks to the classical estimate by Farhat, Mandel and Roux of the spectral condition number [20], which holds unchanged for TFETI:

$$\kappa(\mathbf{PFP}|\mathrm{Im}\mathbf{P}) \leq C\frac{H}{h}, \tag{4.17}$$

where h is the discretization parameter (indirect measure of the problem size) and H is the decomposition parameter (indirect measure of the number of subdomains).

4.4 TFETI for frictionless contact problems

The TFETI method can be extended with relative ease to allow solution of contact problems. First of all, there is an additional linearized contact non-penetration condition

$$\mathbf{B}_c\mathbf{u} \leq \mathbf{c}_c, \tag{4.18}$$

with the related Lagrange multipliers $\boldsymbol{\lambda}_c$. We can incorporate $\mathbf{B}_c$ into $\mathbf{B}$, $\mathbf{c}_c$ into $\mathbf{c}$, and $\boldsymbol{\lambda}_c$ into $\boldsymbol{\lambda}$ and denote by $\mathcal{E}$ and $\mathcal{I}$ index sets corresponding to rows with equalities and inequalities, respectively. Namely,

$$\mathbf{B} = \left[\begin{array}{c}\mathbf{B}_{\mathcal{E}}\\ \hline \mathbf{B}_{\mathcal{I}}\end{array}\right] = \left[\begin{array}{c}\mathbf{B}_g\\ \mathbf{B}_d\\ \hline \mathbf{B}_c\end{array}\right] \in \mathbb{R}^{m\times n}, \quad \mathbf{c} = \left[\begin{array}{c}\mathbf{c}_{\mathcal{E}}\\ \hline \mathbf{c}_{\mathcal{I}}\end{array}\right] = \left[\begin{array}{c}\mathbf{c}_g\\ \mathbf{c}_d\\ \hline \mathbf{c}_c\end{array}\right] \in \mathbb{R}^{m}, \quad \boldsymbol{\lambda} = \left[\begin{array}{c}\boldsymbol{\lambda}_{\mathcal{E}}\\ \hline \boldsymbol{\lambda}_{\mathcal{I}}\end{array}\right] = \left[\begin{array}{c}\boldsymbol{\lambda}_g\\ \boldsymbol{\lambda}_d\\ \hline \boldsymbol{\lambda}_c\end{array}\right] \in \mathbb{R}^{m}, \quad (4.19)$$

with modified dual dimension $m = m_g + m_d + m_c$. We then proceed analogously to Section 4.3, substituting this altered $\mathbf{B}$ and taking into account that $\mathbf{B}_{\mathcal{I}}$ corresponds to inequalities. The primal QP (4.6) then reads

$$\min \frac{1}{2}\mathbf{u}^T\mathbf{K}\mathbf{u} - \mathbf{f}^T\mathbf{u} \quad \text{s.t.} \quad \mathbf{B}_{\mathcal{E}}\mathbf{u} = \mathbf{c}_{\mathcal{E}} \quad \text{and} \quad \mathbf{B}_{\mathcal{I}}\mathbf{u} \le \mathbf{c}_{\mathcal{I}}. \tag{4.20}$$

Thanks to the notation, the related Lagrangian (4.7) remains formally unchanged. The saddle point formulation (4.8) must include the dual feasibility condition $\boldsymbol{\lambda}_{\mathcal{I}} \ge \mathbf{o}$ from (1.15c),

$$(\bar{\mathbf{u}}, \bar{\boldsymbol{\lambda}}) = \arg\inf_{\mathbf{u}} \sup_{\boldsymbol{\lambda}_{\mathcal{I}}\ge\mathbf{o}} L(\mathbf{u}, \boldsymbol{\lambda}). \tag{4.21}$$

This bound propagates to the dual formulation (4.10),

$$\min \frac{1}{2}\boldsymbol{\lambda}^T\mathbf{F}\boldsymbol{\lambda} - \boldsymbol{\lambda}^T\widetilde{\mathbf{d}} \quad \text{s.t} \quad \mathbf{G}\boldsymbol{\lambda} = \mathbf{e} \quad \text{and} \quad \boldsymbol{\lambda}_{\mathcal{I}} \ge \mathbf{o}. \tag{4.22}$$

After homogenization, returning to the original notation, and employing the orthogonal projectors, we finally get minimization on the subset of $\operatorname{Ker}\mathbf{G}$ analogous to (4.15),

$$\min \frac{1}{2}\boldsymbol{\lambda}^T\mathbf{P}\mathbf{F}\mathbf{P}\boldsymbol{\lambda} - \boldsymbol{\lambda}^T\mathbf{P}\mathbf{d} \quad \text{s.t.} \quad \mathbf{G}\boldsymbol{\lambda} = \mathbf{o} \quad \text{and} \quad \boldsymbol{\lambda}_{\mathcal{I}} \ge \boldsymbol{\ell}, \tag{4.23}$$

where $\boldsymbol{\ell} = -\widetilde{\boldsymbol{\lambda}}_{\mathcal{I}}$.

The estimate (4.17) remains valid for the resulting QP (4.23). Note also that the problem (4.6) can be considered a special case of the problem (4.20) with $(\mathbf{B}_c, \mathbf{c}_c) = (\square, \square)$ and empty index set $\mathcal{I}$. The same holds for the final formulations (4.16) and (4.23).

The final QP (4.23) can be solved by combination of the SMALBE algorithm of Section 3.2, and the MPRGP algorithm of Section 3.1.

4.5 TFETI as a sequence of QP transforms

The TFETI method can be expressed in terms of QP transforms introduced in Chapter 2 as a specific instance of the procedure of Section 2.4.4 with

$$\mathsf{QP}_1 = \mathsf{QP}[\ \mathbf{K},\ \mathbf{f}\ |\ \left[\begin{smallmatrix}\mathbf{B}_g\\ \mathbf{B}_d\end{smallmatrix}\right],\ \left[\begin{smallmatrix}\mathbf{o}^{(m_g)}\\ \mathbf{c}_d\end{smallmatrix}\right]\ |\ \mathbf{B}_c,\ \mathbf{c}_c\ |\ \square,\ \square\ ], \tag{4.24}$$

$$\overline{\mathsf{QP}}_1 = \overline{\mathsf{QP}}[\ \mathbf{u} \quad | \ \begin{bmatrix} \boldsymbol{\lambda}_g \\ \boldsymbol{\lambda}_d \end{bmatrix} \quad | \ \boldsymbol{\lambda}_c \quad | \ \square,\ \square\ ]. \tag{4.25}$$

Herewith, we can easily follow how the KKT pair of the original primal QP (4.20) is recovered from the KKT pair $(\widehat{\boldsymbol{\lambda}}, \widehat{\boldsymbol{\alpha}})$ of the last QP (4.23). It follows from Section 2.4.4 that the KKT pair $(\bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\alpha}})$ of the original dual problem (4.22) is derived from $(\widehat{\boldsymbol{\lambda}}, \widehat{\boldsymbol{\alpha}})$ as

$$\bar{\boldsymbol{\alpha}} = \widehat{\boldsymbol{\alpha}} + (\mathbf{G}^T)^L(\mathbf{d} - \mathbf{F}\widehat{\boldsymbol{\lambda}}), \tag{4.26}$$

$$\bar{\boldsymbol{\lambda}} = \widetilde{\boldsymbol{\lambda}} + \widehat{\boldsymbol{\lambda}}, \tag{4.27}$$

and the primal KKT pair is $(\bar{\mathbf{u}}, \bar{\boldsymbol{\lambda}})$ where

$$\bar{\mathbf{u}} = \mathbf{K}^\dagger(\mathbf{f} - \mathbf{B}^T\bar{\boldsymbol{\lambda}}) - \mathbf{R}\bar{\boldsymbol{\alpha}}. \tag{4.28}$$

CHAPTER V

Implicit equality constraint orhonormalization

This chapter presents a simple but powerful ingredient for massively parallel solution of variational inequalities with equality constraints using augmented Lagrangian methods. It has been discovered within joint effort with Alexandros Markopoulos. A paper on this topic is currently under preparation [102].

In the following sections, we discuss how we can take advantage of $\widetilde{\mathbf{B}}_E$ with orthonormal rows in TFETI QP transforms (Section 4.5) and the SMALBE algorithm (Section 3.2), yet to avoid the explicit orthogonalization of $\mathbf{B}_E$. Actually, only the action of $(\mathbf{B}_E\mathbf{B}_E^T)^{-1}$ is needed which is implementable in several efficient ways (Section 7.5.3).

5.1 Equality constraint orhonormalization

Let us consider a bound and equality constrained problem

$$\min \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{x}^T\mathbf{b} \quad \text{s.t} \quad \mathbf{B}_E\mathbf{x} = \mathbf{c}_E \quad \text{and} \quad \mathbf{x}_I \geq \mathbf{o}. \tag{5.1}$$

By pre-multiplying both sides of equality constraints with a non-singular matrix $\mathbf{T}$ such that $\mathbf{T}\mathbf{B}_E$ has orthonormal rows, we get a new problem

$$\min \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{x}^T\mathbf{b} \quad \text{s.t} \quad \widetilde{\mathbf{B}}_E\mathbf{x} = \widetilde{\mathbf{c}}_E \quad \text{and} \quad \mathbf{x}_I \geq \mathbf{o}. \tag{5.2}$$

where

$$\widetilde{\mathbf{B}}_E = \mathbf{T}\mathbf{B}_E, \qquad \widetilde{\mathbf{c}}_E = \mathbf{T}\mathbf{c}_E. \tag{5.3}$$

This orthonormalization is needed to get optimal results for penalty method and the derived SMALBE algorithm. The reason is that the penalized term resulting from the orthonormal equality constraints preserves the spectrum of the original Hessian.

We are here interested in large sparse matrices distributed across computational nodes. There is no implementation of explicit sparse QR factorization fulfilling such requirements available. Another option is the Classical Gram-Schmidt process (CGS) which is more appropriate for distributed vectors than the Modified Gram-Schmidt process. Thanks to the iterative refinement, stability is not an issue. However, the CGS is suitable up to about a thousand of vectors because of the combination of its algorithmic complexity (number of the processed vectors is an arithmetic series) and communication demands (at least one scalar product for each processed vector).

Efficient Cholesky factorization implementations for distributed memory are available in several libraries (Section 6.5). Cholesky factorization can be used in this way

$$\mathbf{B}_E\mathbf{B}_E^T \overset{\text{Cholesky}}{=} \mathbf{L}\mathbf{L}^T, \tag{5.4}$$

$$\mathbf{T} \quad = \quad \mathbf{L}^{-1}. \tag{5.5}$$

However, the actions of the sole $\mathbf{L}^{-1}$ and $\mathbf{L}^{-T}$ are not available in most cases. Fortunately, their need can be circumvented as described below.

5.2 Equality constraint homogenization

We shall adapt the QP transform described in Section 2.3.3. Let $\widetilde{\mathbf{x}}$ satisfy

$$\widetilde{\mathbf{B}}_E\widetilde{\mathbf{x}} = \widetilde{\mathbf{c}}_E. \tag{5.6}$$

It is natural to use the least-square solution

$$\widetilde{\mathbf{x}} = \widetilde{\mathbf{B}}_E^R\widetilde{\mathbf{c}}_E, \tag{5.7}$$

which is thanks to orthonormality

$$\widetilde{\mathbf{x}} = \widetilde{\mathbf{B}}_E^T\widetilde{\mathbf{c}}_E. \tag{5.8}$$

The solution of (5.2) can then be sought in the form

$$\mathbf{x} = \widehat{\mathbf{x}} + \widetilde{\mathbf{x}}. \tag{5.9}$$

Since

$$\frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{x}^T\mathbf{b} = \frac{1}{2}\widehat{\mathbf{x}}^T\mathbf{A}\widehat{\mathbf{x}} - \widehat{\mathbf{x}}^T(\mathbf{b} - \mathbf{A}\widetilde{\mathbf{x}}) + \frac{1}{2}\widetilde{\mathbf{x}}^T\mathbf{A}\widetilde{\mathbf{x}} - \widetilde{\mathbf{x}}^T\mathbf{b},$$

and the last two terms are constant with respect to $\widehat{\mathbf{x}}$, the problem (5.2) is equivalent to

$$\min \frac{1}{2}\widehat{\mathbf{x}}^T\mathbf{A}\widehat{\mathbf{x}} - \widehat{\mathbf{x}}^T\mathbf{d} \quad \text{s.t} \quad \widetilde{\mathbf{B}}_E\widehat{\mathbf{x}} = \mathbf{o} \quad \text{and} \quad \widehat{\mathbf{x}}_I \geq \boldsymbol{\ell}, \tag{5.10}$$

where $\mathbf{d} = \mathbf{b} - \mathbf{A}\widetilde{\mathbf{x}}$ and $\boldsymbol{\ell} = -\widetilde{\mathbf{x}}_I$. Let us return to the old notation, $\mathbf{x} := \widehat{\mathbf{x}}$,

$$\min \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{x}^T\mathbf{d} \quad \text{s.t} \quad \widetilde{\mathbf{B}}_E\mathbf{x} = \mathbf{o} \quad \text{and} \quad \mathbf{x}_I \geq \boldsymbol{\ell}. \tag{5.11}$$

Notice that using (5.3) and (5.5), we can rewrite (5.8) as

$$\widetilde{\mathbf{x}} \;=\; \widetilde{\mathbf{B}}_E^T\widetilde{\mathbf{c}}_E \;=\; \mathbf{B}_E^T\mathbf{T}^T\mathbf{T}\mathbf{c}_E \;=\; \mathbf{B}_E^T\mathbf{L}^{-T}\mathbf{L}^{-1}\mathbf{c}_E \;=\; \mathbf{B}_E^T(\mathbf{L}\mathbf{L}^T)^{-1}\mathbf{c}_E.$$

Additionally, exploiting (5.4) we get

$$\widetilde{\mathbf{x}} = \mathbf{B}_E^T(\mathbf{B}_E\mathbf{B}_E^T)^{-1}\mathbf{c}_E = \mathbf{B}_E^R\mathbf{c}_E, \tag{5.12}$$

i.e. (5.8) *can be rewritten using only the original dual constraint matrix* $\mathbf{B}_E$ *with no explicit standalone* $\mathbf{T}$.

5.3 Preconditioning by the orthogonal projector

Proceeding with the QP transform of Section 2.3.5, the problem (5.11) can be reformulated as

$$\min \frac{1}{2}\mathbf{x}^T\mathbf{P}\mathbf{A}\mathbf{P}\mathbf{x} - \mathbf{x}^T\mathbf{P}\mathbf{d} \quad \text{s.t} \quad \widetilde{\mathbf{B}}_E\mathbf{x} = \mathbf{o} \quad \text{and} \quad \mathbf{x}_I \geq \boldsymbol{\ell}, \tag{5.13}$$

where

$$\mathbf{P} = \widetilde{\mathbf{B}}_E^T(\widetilde{\mathbf{B}}_E\widetilde{\mathbf{B}}_E^T)^{-1}\widetilde{\mathbf{B}}_E = \widetilde{\mathbf{B}}_E^T\widetilde{\mathbf{B}}_E = \mathbf{B}_E^R\mathbf{B}_E,$$

using the very same trick as in the previous section. *Again, no explicit* $\mathbf{T}$ *is introduced.*

5.4 SMALBE-M algorithm modification

The original SMALBE-M algorithm for bound and equality constrained QP has been presented as Algorithm 1 in Section 3.2. The augmented Lagrangian and its gradient for the problem (5.13) read

$$L(\mathbf{x}, \boldsymbol{\lambda}_E, \rho) = \frac{1}{2}\mathbf{x}^T(\mathbf{P}\mathbf{A}\mathbf{P} + \rho\widetilde{\mathbf{B}}_E^T\widetilde{\mathbf{B}}_E)\mathbf{x} - (\mathbf{P}\mathbf{d})^T\mathbf{x} + \mathbf{x}^T\widetilde{\mathbf{B}}_E^T\boldsymbol{\lambda}_E, \tag{5.14}$$

$$\mathbf{g}(\mathbf{x}, \boldsymbol{\lambda}_E, \rho) = (\mathbf{P}\mathbf{A}\mathbf{P} + \rho\widetilde{\mathbf{B}}_E^T\widetilde{\mathbf{B}}_E)\mathbf{x} - \mathbf{P}\mathbf{d} + \widetilde{\mathbf{B}}_E^T\boldsymbol{\lambda}_E, \tag{5.15}$$

and the projected gradient $\mathbf{g}^P$ is obtained by projecting $\mathbf{g}$ into the superset of the feasible set related to the bound constraints. Standalone $\boldsymbol{\lambda}_E^k$ appears in Algorithm 1 only in its own update (line 3). In other places it appears only as a part of L and $\mathbf{g}$ defined in (5.14) and (5.15),

respectively, where it is pre-multiplied by $\widetilde{\mathbf{B}}_E^T$. We can alter the line 3 of Algorithm 1 in order to directly update $\widetilde{\mathbf{B}}_E^T\boldsymbol{\lambda}_\mathbf{E}$ and no longer explicitly evaluate $\boldsymbol{\lambda}_E^k$,

$$(\widetilde{\mathbf{B}}_E^T\boldsymbol{\lambda}_\mathbf{E})^{k+1} := (\widetilde{\mathbf{B}}_E^T\boldsymbol{\lambda}_\mathbf{E})^k + \rho\widetilde{\mathbf{B}}_E^T\widetilde{\mathbf{B}}_E\mathbf{x}^k. \tag{5.16}$$

Finally, the action of $\widetilde{\mathbf{B}}_E$ appears in the stopping criterion in the equality constraint violation norm $||\widetilde{\mathbf{B}}_E\mathbf{x}^k||$ which can be evaluated as

$$||\widetilde{\mathbf{B}}_E\mathbf{x}^k|| = \sqrt{(\mathbf{x}^k, \widetilde{\mathbf{B}}_E^T\widetilde{\mathbf{B}}_E\mathbf{x}^k)}. \tag{5.17}$$

But again $\widetilde{\mathbf{B}}_E^T\widetilde{\mathbf{B}}_E = \mathbf{B}_E^R\mathbf{B}_E$ with no explicitly appearing $\mathbf{T}$.

Notice that after these modifications SMALBE-M contains only actions of $\widetilde{\mathbf{B}}_E^T\widetilde{\mathbf{B}}_E$ but no actions of $\widetilde{\mathbf{B}}_E$ and $\mathbf{T}$ alone.

Part II

Implementation

CHAPTER VI

Open source software for numerical modelling

Open source libraries promote advances in computational science by allowing users to use them for their own science and make derived works. For supercomputing, it is important that the number of processors used for computations is not limited by the number of the owned licenses. All the open source libraries mentioned in this chapter are related by focus to our own PERMON software. Some of them even already belong to its dependencies, some can be potentially used to extend its capabilities, some are just based on similar ideas. Several closed source software products are also mentioned here as they are strongly relevant for this thesis' topics.

6.1 Meshing

This subsection lists some freely available tools for automatic mesh generation. A more complete list of them can be found e.g. in [42].

6.1.1 Netgen

Netgen [48] is a well-known automatic 3D tetrahedral mesh generator. It comes as a C++ program library as well as stand-alone program with its own graphical user interface. It accepts input from constructive solid geometry (CSG) or boundary representation (BRep) from STL file format. The connection to a geometry kernel allows the handling of IGES and STEP files. Netgen contains modules for mesh optimization and hierarchical mesh refinement. There is a

general purpose FEM library on top of Netgen called **NGSolve** [49]. Netgen is open source based on the LGPL license.

6.1.2 TetGen

TetGen [72] is a program and C++ program library to generate tetrahedral meshes of any 3D polyhedral domains. TetGen generates exact constrained Delaunay tetrahedralizations, boundary conforming Delaunay meshes, and Voronoi partitions. TetGen provides various features to generate good quality and adaptive tetrahedral meshes suitable for numerical methods, such as FEM or finite volume methods. They include mesh quality and size control; adaptive mesh refinement (AMR); mesh coarsening; surface mesh refinement; facet self-intersection detection and others. TetGen is distributed under the terms of the GNU Affero General Public License (AGPLv3).

6.1.3 Triangle

Triangle [78] generates exact Delaunay triangulations, constrained Delaunay triangulations, conforming Delaunay triangulations, Voronoi diagrams, and high-quality triangular meshes. The latter can be generated with no small or large angles, and are thus suitable for finite element analysis. The product is no longer maintained. Available under a custom open source license.

6.1.4 ViennaMesh

ViennaMesh [81] provides a unified C++, template-based interface for various mesh related tools. These tools cover mesh generation, adaptation, and classification of multi-segmented meshes and geometries for unstructured two- and three-dimensional meshes. The goal is to provide applications with an additional back-end layer for mesh generation, allowing to seamlessly exchange mesh tools, for example, mesh generation kernels. Currently the following external mesh generation kernels are available: Triangle, Tetgen, and Netgen – all described above. Several tools are available to further improve the quality of the generated meshes. Most notable is a hull adaptation module which significantly improves the quality of a hull mesh and therefore improves the quality of the subsequent volume meshing step too. ViennaMesh is available under the LGPL.

There is a related product, **ViennaGrid** [80] for handling of structured and unstructured meshes in arbitrary spatial dimensions using different coordinate systems. It employs a highly configurable internal representation of meshes, while providing a uniform interface for the storage and access of data on mesh elements as well as STL-compatible iteration over such elements. ViennaGrid is available under the permissive MIT/X11 license.

6.1.5 Pamgen

Pamgen [30, 76] is a different type of product – it is focused on producing, in parallel, during simulation code execution, multi-billion element hexahedral or quadrilateral meshes of simple topologies. The Pamgen library provides functions to mimic the process of reading mesh data from a processor specific mesh file through a file access API. During a parallel simulation, each processor provides Pamgen with a terse specification of the desired finite element mesh, the total number of processors, and the index of the local processor. At that time, the Pamgen library calculates the mesh for the local processor and its relation to mesh on adjacent processors. The processor may then call functions that return descriptions of its local the finite element mesh and the relationship of that mesh to the mesh on adjacent processors. Pamgen is licensed LGPL.

6.2 Partitioning

6.2.1 METIS

METIS [43] is a set of serial programs for partitioning graphs, partitioning finite element meshes, and producing fill reducing orderings for sparse matrices. The algorithms implemented in METIS are based on the multilevel recursive-bisection, multilevel k-way, and multi-constraint partitioning schemes developed in Karypis lab. METIS provides fast sparse matrix orderings which reduce the computational and storage requirements up to an order of magnitude and are suitable for parallel direct factorization. From version 5.0, Metis provides support for enforcing finite element mesh partitions to be contiguous. METIS is licensed under the Apache License, Version 2.0.

6.2.2 ParMETIS

ParMETIS [54] is an MPI-based parallel library that implements a variety of algorithms for partitioning unstructured graphs, meshes, and for computing fill-reducing orderings of sparse matrices. ParMETIS extends the functionality provided by METIS and includes routines that are especially suited for parallel adaptive mesh refinement (AMR) computations and large scale numerical simulations. The algorithms implemented in ParMETIS are based on the parallel multilevel k-way graph-partitioning, adaptive repartitioning, and parallel multi-constrained partitioning schemes developed in Karypis lab. ParMETIS is distributed under the custom license[1], allowing free use for educational and research purposes, other uses require prior approval.

[1] `http://glaros.dtc.umn.edu/gkhome/metis/parmetis/download`

6.2.3 SCOTCH

The purpose of SCOTCH [64] is to apply graph theory with a divide and conquer approach to scientific computing problems such as graph and mesh partitioning, static mapping, and sparse matrix orderings. Its capabilities can be used through a set of stand-alone programs as well as through the libSCOTCH library, which offers both C and Fortran interfaces. SCOTCH provides algorithms to partition graph structures, as well as mesh structures defined as node-element bipartite graphs and which can also represent hypergraphs. Its running time is linear in the number of edges of the source graph, and logarithmic in the number of vertices of the target graph for mapping computations. It provides many tools to build, check, and display graphs, meshes and matrix patterns. It is written in C and uses the POSIX interface, which makes it highly portable. **PT-SCOTCH** uses the MPI interface, and optionally the POSIX threads. SCOTCH is developed by the Laboratoire Bordelais de Recherche en Informatique (LaBRI) within the ScAlApplix project of INRIA Bordeaux - Sud-Ouest. It is distributed as free software, under the terms of the CeCILL-C license[2].

6.3 FEM libraries

This section presents an alphabetically sorted selection of open source libraries implementing discretization of PDEs using FEM. It is perhaps impossible to provide here a fully exhaustive list but we have done our best to point out products that are in our humble opinion well-known, widely used, maintained, and support high-performance computing (HPC). A more complete list of FEM software can be found e.g. in [83].

6.3.1 deal.II

deal.II [73] is a C++ program library aimed to enable rapid development of modern adaptive FEM codes. It provides tools for adaptive meshes, grid handling and refinement, handling of degrees of freedom, input of meshes and output of results in graphics formats, etc. Support for dimension independent programming is included by means of integer template parameters, i.e. without penalties on run-time and memory consumption [4]. A complete stand-alone linear algebra library and interfaces to Trilinos and PETSc are offered. deal.II supports both threading (using TBB) and message passing parallelization models. A limitation for industrial use lies in the fact that triangles and tetrahedra elements are not directly supported by design. deal.II is licensed LGPL v2.1.

[2] http://www.cecill.info/licences/Licence_CeCILL-C_V1-en.html

6.3.2 DUNE

DUNE (the Distributed and Unified Numerics Environment) [18] is a modular toolbox for solving partial differential equations (PDEs) with grid-based methods. It supports the easy implementation of discretization methods like FEM, Finite Volumes (FV), and also Finite Differences (FD). The underlying idea of DUNE is to create slim interfaces allowing an efficient use of legacy and/or new libraries. C++ template design enables very different implementations of the same concept (i.e. grids, solvers, and so on) using a common interface at a very low overhead. DUNE contains its own parallel linear algebra module called ISTL (the Iterative Solver Template Library). DUNE is licensed under the GPL v2 with a "runtime exception".

6.3.3 Elmer

Elmer [60, 61] is an open source multiphysical simulation software mainly developed by CSC – IT Center for Science (CSC), Finland. Elmer development was started 1995 in collaboration with Finnish Universities, research institutes and industry. After its open source publication in 2005, the use and development of Elmer has become international. Elmer includes physical models of fluid dynamics, structural mechanics, electromagnetics, heat transfer and acoustics, for example. Unlike other tools in this section, Elmer is a self-contained SW package which can be directly used by the user without coding. Released under GPL v2.

6.3.4 Feel++

Feel++ [22] is a relatively new C++ library for arbitrary order continuous or discontinuous Galerkin methods (such as FEM, hp-FEM, spectral element method and reduced basis method) in 1D, 2D and 3D. It is aimed to be a small and manageable library which shall nevertheless encompass a wide range of numerical methods and techniques. It offers a language for variational formulations embedded into C++ for maximal mathematical expressivity. It depends on PETSc for linear and non-linear solvers, and optionally interfaces SLEPc for sparse standard and generalized eigenvalue solvers. It supports Gmsh (mesh generation and post-processing) and Paraview (post-processing). Feel++ code license is LGPL v3.

6.3.5 FEniCS

The FEniCS Project [74] is a collection of interoperable software components for efficient automated solution of PDEs. FEniCS also provides tools for exploring and development of new methods. It has an extensive list of features including automated solution of variational problems, automated error control and adaptivity, a comprehensive library of FEs, and many more. PDEs can be specified in near-mathematical notation (as FE variational problems) and solved automatically thanks to the DOLFIN C++/Python library. The DOLFIN component provides a problem solving environment for models based on PDEs and implements core parts of the

functionality of FEniCS, including data structures and algorithms for computational meshes and finite element assembly. FEniCS provides unified access through a common wrapper layer to a range of linear algebra backends – PETSc, Trilinos/Epetra, uBLAS and MTL4. The backend may be easily switched, parallel computing is supported through the PETSc and Epetra backends. All FEniCS core components are licensed under the LGPL v3.

6.3.6 FreeFem++

FreeFem++ [23, 28] is a software to solve numerically PDEs in 2D and 3D with FEM. It is written in C++. It has its own user language to set and control the problem. The FreeFem++ language allows a quick specification of linear PDEs with the variational formulation of a linear steady state problem. Scripts can be written to solve nonlinear, time-depend or coupled problems. One can solve problems with moving domain or eigenvalue problems, do mesh automatic mesh generation and adaption, compute error indicators, etc. Integrated graphical environment called **FreeFem++-cs** is offered. FreeFem++ is distributed under the LGPL v2.1.

6.3.7 Hermes

Hermes [31] is a C++ library for rapid development of adaptive hp-FEM and hp-DG solvers, with emphasis on nonlinear, time-dependent, multi-physics problems. Novel hp-adaptivity algorithms help solve a large variety of problems ranging from ODE and stationary linear PDE to complex time-dependent nonlinear multiphysics PDE systems. PETSc, Trilinos, PARALUTION, SuperLU, MUMPS and UMFPACK can be used as matrix solvers. A standard way to use Hermes is to write short C++ user programs, but for those who prefer a graphical interface, there is a graphical engineering tool based on Hermes called **Agros2D** [1]. Hermes is licensed LGPL v3.

6.3.8 libMesh

The **libMesh** [37, 38] library provides a framework for the numerical simulation of PDEs using arbitrary unstructured discretizations on serial and parallel platforms. A major goal of the library is to provide support for adaptive mesh refinement (AMR) computations in parallel while allowing a research scientist to focus on the physics they are modelling. libMesh currently supports 1D, 2D, and 3D steady and transient simulations on a variety of popular geometric and finite element types. The library makes use of high-quality, existing software whenever possible. PETSc or Trilinos are used for the solution of linear systems on both serial and parallel platforms, and LASPack is included with the library to provide linear solver support on serial machines. An optional interface to SLEPc is also provided for solving both standard and generalized eigenvalue problems. libMesh is licensed under the LGPL v3.

6.3.9 MOOSE

The Multiphysics Object-Oriented Simulation Environment (MOOSE) [44] is a FE multiphysics framework primarily developed by Idaho National Laboratory. It provides a high-level interface to PETSc and libMesh. MOOSE presents a straightforward API that aligns well with the real-world problems. Features include fully-coupled, fully-implicit multiphysics solver; dimension independent physics; modular development; built-in mesh adaptivity; Continuous and Discontinuous Galerkin (DG); intuitive parallel multiscale solves; dimension agnostic, parallel geometric search (for contact related applications); flexible, pluggable graphical user interface; physics modules providing general capability for solid mechanics, phase field modelling, Navier-Stokes, heat conduction and more. The 2014 R&D 100 Award Winner in Software category. Everything in MOOSE is licensed under the LGPL.

6.3.10 OOFEM

OOFEM [50] is a free FE code with object oriented architecture for solving mechanical, transport and fluid mechanics problems that operates on various platforms. The aim of this project is to develop efficient and robust tool for FEM computations as well as to provide modular and extensible environment for future development. The fundamental part is a modular and extensible FEM kernel (OOFEMlib). Its features include full extensibility; full restart support; staggered analysis; parallel processing (based on domain decomposition, message passing paradigms, and dynamic load balancing engine); adaptive analysis; eXtented Finite Element Method (XFEM); Iso-Geometric Analysis (IGA). OOFEM possesses interfaces to sparse linear and eigenvalue solver libraries (PETSc, SLEPc, IML, and SPOOLES). OOFEM is released under LGPL v2.1.

6.4 Toolkits for numerical computations

This section discusses well-known general toolkits for fundamental numerical computations which provide linear algebra, linear system solvers, preconditioners and so on. They provide infrastructure for PDE solution on all major hardware platforms and operating systems of personal computers as well as supercomputers.

6.4.1 PETSc

PETSc (Portable, Extensible Toolkit for Scientific Computation) [2, 3, 67] PETSc is a suite of data structures and routines for the scalable parallel solution of scientific applications modelled by PDEs. It supports MPI, shared memory, and GPUs through CUDA or OpenCL, as well as hybrid MPI-shared memory or MPI-GPU parallelism. It provides parallel matrix data structures and algorithms, preconditioners, linear and nonlinear system solvers, time integration, data

structures and operations for unstructured and structured meshes, debugging and profiling tools, and others.

PETSc provides many of the mechanisms needed within parallel application codes, such as parallel matrix and vector assembly routines. The library is organized hierarchically, enabling users to employ the level of abstraction that is most appropriate for a particular problem. By using techniques of object-oriented programming, PETSc provides enormous flexibility for users.

PETSc also provides extensive set of interfaces to many external libraries (sparse and dense linear system solvers, preconditioners, input/output, partitioning, and others). This is a popular feature as it greatly reduces effort needed for integration of all those libraries into the user's code. PETSc is written in ANSI C, callable also from C++, FORTRAN, Python, Java and MATLAB. Is is distributed under a permissive open source license (2-clause BSD).

There are two important libraries extending PETSc that use the same "look & feel": **SLEPc** (Scalable Library for Eigenvalue Problem Computations) [32, 66] for eigenvalue problem solution (licensed LGPL v3), and **TAO** (Toolkit for Advance Optimization) for nonlinear optimization (Section 6.6.1). TAO became a part of PETSc since its version 3.5.

6.4.2 Trilinos

The Trilinos Project [75] is an effort to develop algorithms and enabling technologies using modern object-oriented software design for the solution of large-scale, complex multi-physics engineering and scientific problems, while still leveraging the value of established libraries such as PETSc, METIS, SuperLU, Aztec, the BLAS and LAPACK. It consists of relatively autonomous, loosely coupled packages with common web page, build system, and basic infrastructure. Particularly, its Epetra package provides the parallel sparse linear algebra foundation layer. Its more modern, template-based successor is the Tpetra package. Trilinos emphasizes abstract interfaces for maximum flexibility of component interchanging, and provides a full-featured set of concrete classes that implement all abstract interfaces. The code is written mainly in C++ with FORTRAN and Python bindings. Most Trilinos source code, including the code developed at Sandia National Laboratories, is licensed either LGPL or BSD.

6.4.3 PARALUTION

PARALUTION [52] is a library that enables performing various sparse iterative solvers and preconditioners on multi/many-core CPU, GPU, and Intel Xeon Phi/MIC devices. Based on C++, it provides a generic and flexible design that allows seamless integration with other scientific software packages. PARALUTION contains Krylov subspace solvers, Multigrid, Deflated PCG, Fixed-point iteration schemes, Mixed-precision schemes and fine-grained parallel preconditioners based on splitting, ILU factorization with levels, multi-elimination ILU factorization, additive Schwarz and approximate inverse. The library also provides iterative eigenvalue solvers. The PARALUTION library is released under the dual license model – GPL v3 and commercial.

6.4.4 ViennaCL

The Vienna Computing Library (ViennaCL) [79] is a scientific computing library written in C++ and provides CUDA, OpenCL and OpenMP computing backends. It enables simple, high-level access to the vast computing resources available on parallel architectures such as GPUs and is primarily focused on common linear algebra operations (BLAS levels 1, 2 and 3) and the solution of large systems of equations by means of iterative methods with optional preconditioners. ViennaCL wrappers are available in PETSc. ViennaCL is distributed under the MIT/X11 open source license.

6.5 Parallel sparse direct linear solvers

6.5.1 MUMPS

MUMPS (MUltifrontal Massively Parallel sparse direct Solver) [46] is a package for solving systems of linear equations with square sparse matrix that can be either unsymmetric, symmetric positive definite, or general symmetric. MUMPS employs a multifrontal method for LU and LDLT factorization [40]. MUMPS exploits both parallelism arising from sparsity in the system matrix and from dense factorizations kernels. The main features of the MUMPS package include the solution of the transposed system, input of the matrix in assembled format (distributed or centralized) or elemental format, error analysis, iterative refinement, scaling of the original matrix, out-of-core capability, parallel analysis, detection of null pivots, basic estimate of rank deficiency and null space basis for symmetric matrices, and computation of a Schur complement matrix. MUMPS offers several built-in ordering algorithms and a tight interface to some external ordering packages. MUMPS is available in various arithmetics (real or complex, single or double precision). The software is mainly written in Fortran 90 although a C interface is available. It uses pure MPI parallelization and relies on ScaLAPACK for auxiliary parallel dense solves. It is released under the CeCILL-C license[3].

6.5.2 SuperLU

SuperLU [71] is a general purpose library for the direct solution of large, sparse, nonsymmetric systems of linear equations on high-end computers, based on the supernodal LU factorization method. The library is written in C and is callable from either C or Fortran. The library routines perform an LU factorization with numerical pivoting and triangular system solves (forward and backward substitution). Routines are provided to perform iterative-refinement, equilibrate the system, estimate the condition number, calculate the relative backward error, and estimate error

[3] http://www.cecill.info/licences/Licence_CeCILL-C_V1-en.html

bounds for the refined solutions. The factorization algorithm uses a graph reduction technique to reduce graph traversal time in the symbolic analysis, and data movement between levels of the memory hierarchy is reduced through loop ordering and the use of dense matrix operations in the numerical kernel. For the distributed memory implementation, a two-dimensional block cyclic matrix distribution is used to enhance scalability. SuperLU contains a collection of three related subroutine libraries: sequential SuperLU for uniprocessors, the multithreaded version (SuperLU_MT) for medium-size SMPs, and the MPI version (SuperLU_DIST) for large distributed memory machines. It is released under a custom open-source license[4].

6.5.3 PARDISO

PARDISO [53] is according to its authors "a thread-safe, high-performance, robust, memory efficient and easy to use software for solving large sparse symmetric and unsymmetric linear systems of equations on shared-memory and distributed-memory multiprocessors." It supports unsymmetric, structurally symmetric or symmetric, real or complex, positive definite or indefinite, hermitian matrices. It supports both shared and distributed memory models. PARDISO is mature, efficient and feature-rich product but it is the only library from all mentioned here that has a closed-source license. However, an older version of PARDISO from 2006 is distributed as a part of Intel MKL. This one is interfaced by PETSc (Section 6.4.1).

6.5.4 Other sparse direct solvers

Another interesting sparse direct solver is PasTiX[55], usable in a hybrid MPI/multi-threaded manner which is still quite a rare feature. SuiteSparse [70] features several well-known sparse direct solvers, for instance KLU, UMFPACK or CHOLMOD. They do not support the distributed memory model. However, the latter one supports GPU acceleration.

6.6 QP solvers

6.6.1 TAO

The Toolkit for Advanced Optimization (TAO) [47] focuses on the development of algorithms and software for the solution of large-scale optimization problems on high-performance architectures. Areas of interest include unconstrained and box-constrained optimization, nonlinear least squares problems, optimization problems with partial differential equation constraints, and variational inequalities and complementarity constraints. TAO has become a part of PETSc (Section 6.4.1) since its version 3.5. As such, it has the same licence, i.e. 2-clause BSD.

[4] `http://crd-legacy.lbl.gov/~xiaoye/SuperLU/License.txt`

6.6.2 OOQP – object-oriented software for quadratic programming

OOQP [51] is an object-oriented C++ package, based on a primal-dual interior-point method, for solving convex QPs. It contains code that can be used "out of the box" to solve a variety of structured QPs, including general sparse QPs, QPs arising from support vector machines, Huber regression problems, and QPs with bound constraints. OOQP itself is not parallelized, it can however wrap a parallel linear algebra library such as PETSc (Section 6.4.1). The source code seems to be abandoned, the latest commit to the source repository was on November 1, 2014. It is licensed under a custom open source license[5].

6.6.3 QuadProg++

QuadProg++ [59] is a sequential C++ library for strictly convex QP which implements the Goldfarb-Idnani active-set dual method. It is licensed under LGPL. The code is no longer maintained, the last update was on January 8, 2013.

6.6.4 CGAL

CGAL (The Computational Geometry Algorithms Library) [6] is an open source software project that provides easy access to efficient and reliable geometric algorithms in the form of a C++ library. This library offers data structures and algorithms like triangulations, Voronoi diagrams, Boolean operations on polygons and polyhedra, point set processing, arrangements of curves, surface and volume mesh generation, geometry processing, alpha shapes, convex hull algorithms, shape analysis, AABB and KD trees and others. Its package Linear and Quadratic Programming Solver contains algorithms for linear programming and QP. The algorithms are exact, i.e. the solution is computed in terms of multiprecision rational numbers. The resulting solution is certified: along with the claims that the problem under consideration has an optimal solution, is infeasible, or is unbounded, the algorithms also deliver proofs for these facts. These proofs can easily (and independently from the algorithms) be checked for correctness. The solution algorithms are based on a generalization of the simplex method to quadratic objective functions. CGAL is distributed under a dual license scheme: GPL/LGPL and commercial.

6.6.5 Elemental

Elemental [19] is a library for distributed-memory direct linear algebra and optimization, with the term direct being the best single descriptor for the class of algorithms containing dense linear algebra, sparse-direct linear algebra, and Interior Point Methods for convex optimization. Elemental currently supports C++11, C, and Python interfaces. There is also a separately maintained R interface called R-Elemental [62], and a Julia interface is under development.

[5] http://pages.cs.wisc.edu/~swright/ooqp/COPYRIGHT.html

Interfaces to other languages, such as Fortran 90, can be built on top of the C interface. Elemental currently supports distributed dense and sparse Linear, Quadratic, and Second-Order Cone Programs via Mehrotra Predictor-Corrector primal-dual Interior Point Methods. All files distributed with Elemental, with the exception of Elementals custom ParMETIS extensions (which can easily be disabled), are distributed under the terms of the New BSD (BSD 3-clause) license.

6.6.6 qpOASES

qpOASES [58] is an open-source C++ implementation of the recently proposed OASES (Online Active SEt Strategy), which was inspired by important observations from the field of parametric quadratic programming (QP). It has several theoretical features that make it particularly suited for model predictive control (MPC) applications. Further numerical modifications have made qpOASES a reliable QP solver, even when tackling semi-definite, ill-posed or degenerated QP problems. Moreover, several interfaces to third-party software like MATLAB or Simulink are provided that make qpOASES easy-to-use even for users without knowledge of C/C++. qpOASES can solve any QP and also does a good job in detecting infeasible or unbounded QP problem formulations. Current development is mainly supported by researchers at the Interdisciplinary Center for Scientific Computing at Heidelberg University and at ABB Corporate Research. qpOASES is distributed under the LGPL, version 2.1.

6.6.7 CVXOPT – Python Software for Convex Optimization

CVXOPT [7] is a free software package for convex optimization based on the Python programming language. It can be used with the interactive Python interpreter, on the command line by executing Python scripts, or integrated in other software via Python extension modules. Its main purpose is to make the development of software for convex optimization applications straightforward by building on Pythons extensive standard library and on the strengths of Python as a high-level programming language. CVXOPT is itself sequential but may be used with threaded BLAS. CVXOPT is released under the GPL v3.

6.6.8 HQP – Huge Quadratic Programming

HQP [33] is a sequential solver for nonlinearly constrained large-scale optimization. It is intended for problems with sufficient regular sparsity structure. Such optimization problems arise e.g. from the numerical treatment of optimal control problems. External interfaces allow the formulation of optimization problems based on widely used model formats. HQP consists of two main parts: the actual HQP optimizer and the front-end Omuses. Both parts are designed as framework in the programming language C++. The actual HQP optimizer treats nonlinearly constrained problems with a sequential quadratic programming (SQP) algorithm. An interior-point method is applied to the solution of convex quadratic subproblems. The implementation

is based on sparse matrix codes of the Meschach library for matrix computations in C. The matrix library was extended with additional routines for the analysis and direct solution of sparse equation systems. HQP is available under the GPL/LGPL.

6.6.9 GALAHAD

GALAHAD [25] is a thread-safe library of Fortran 2003 packages for solving nonlinear optimization problems. At present, the areas covered by the library are unconstrained and bound-constrained optimization, quadratic programming, nonlinear programming, systems of nonlinear equations and inequalities, and nonlinear least squares problems. It provides MATLAB support and generic interfaces for external sparse direct solvers, including those that work in parallel or out-of-core. It uses a proprietary dual academic/commercial license[6].

6.6.10 PENOPT

PENOPT [56] is a set of sequential computer programs for mathematical optimization and related specialized optimization programs. Its goal is to develop a unified approach to problems of nonlinear programming (NLP) and (linear and nonlinear) semidefinite programming (SDP). The solvers are based on a generalized Augmented Lagrangian method combined with the Trust Region algorithm. PENOPT flagship PENNON solves optimization problems with general smooth objective function and combination of standard nonlinear constraints with linear and bilinear matrix inequality constraints. PENLAB is a MATLAB-based code which is able to solve almost all types of problems that PENNON can. It was developed in cooperation with NAG Ltd. Other packages of PENOPT include PENSDP (a solver for linear semidefinite programming problems), PENBMI (solver for optimization and feasibility problems with quadratic objective function and linear and bilinear matrix inequalities) and PENFMO (code for Free Material Optimization). PENLAB is available under an open-source license (GPL v3.0). All other PENOPT solvers use a proprietary license.

6.6.11 Gurobi

The Gurobi Optimizer [27] is a state-of-the-art solver for mathematical programming. The solvers in the Gurobi Optimizer were designed from the ground up to exploit modern architectures and multi-core processors, using the most advanced implementations of the latest algorithms. It includes the following solvers: linear programming solver, mixed-integer linear programming solver, mixed-integer quadratic programming solver, quadratic programming solver, quadratically constrained programming solver, mixed-integer quadratically constrained programming solver. Gurobi is commercial software but free of charge for academic users.

[6] `http://www.galahad.rl.ac.uk/cou.html`

6.6.12 MOSEK

MOSEK [45] is a tool for solving mathematical optimization problems such as linear programs, quadratic programs, conic problems, mixed integer problems. Due to its powerful state-of-the-art interior-point optimizer for linear, quadratic and conic problems, MOSEK is widely employed in the financial, energy and forestry industry. It includes interfaces to languages such as C, Java, .NET and Python. MOSEK is commercial software but free of charge for academic users.

CHAPTER VII

PERMON toolbox

PERMON (Parallel, Efficient, Robust, Modular, Object-oriented, Numerical) is a newly emerging collection of software libraries, uniquely combining QP algorithms and DDMs. It is built on top of the well-known PETSc framework for numerical computations (mainly its linear algebra part) and adds new specific functionality – QP algorithms, DDMs of the FETI type, and some application-specific algorithms (for e.g. contact mechanics, elasto-plasticity). Among the main applications are contact problems of mechanics.

The central part of PERMON is its Solver Core consisting of PermonFLLOP and PermonQP modules. Moreover, PERMON includes application-specific solver modules (PermonPlasticity, PermonMultiBody), discretization tools (PermonCube, PermonMembrane and interfaces with external discretization software), and support tools.

The PermonFLLOP package is focused on non-overlapping DDM of the FETI type, allowing efficient and robust utilization of contemporary parallel computers for problems with billions of unknowns. Any FEM software can be used to generate mesh and assemble the stiffness matrices and load vectors per each subdomain independently. Additionally, a mapping from the local to the global numbering of degrees of freedom is needed, and non-penetration and friction information in case of contact problems. All these data are passed to PermonFLLOP, which prepares auxiliary data needed in the DDM.

PermonQP, a general purpose QP solver, is then called in the backend to solve the resulting equality constrained problem with additional inequality constraints in case of contact problems. PermonQP can be used also standalone for applications where QP arise like least-squares regression, data fitting, data mining, support vector machines, control systems, and others.

This chapter follows the process of solving contact problems with PERMON (Figure 7.1). The mesh is “teared” into subdomains and each of them is discretized separately with FEM. This

decomposition and discretization is implemented by the PermonMembrane and PermonCube (Section 7.1) packages, respectively. They are actually "placeholders" of a full-featured FEM library, allowing easy benchmarking of Solver Core. Section 7.2 introduces this Solver Core. The subdomain problems are "interconnected" by means of FETI using PermonFLLOP (Section 7.3). Finally, the resulting QP problem is solved by the PermonQP module (Section 7.4), which contains implementations of specific algorithms for inequality constrained problems, particularly the MPRGP algorithm (Section 3.1). Additionally, Section 7.5 deals with using direct solvers for the local and coarse problems of FETI in PERMON Solver Core.

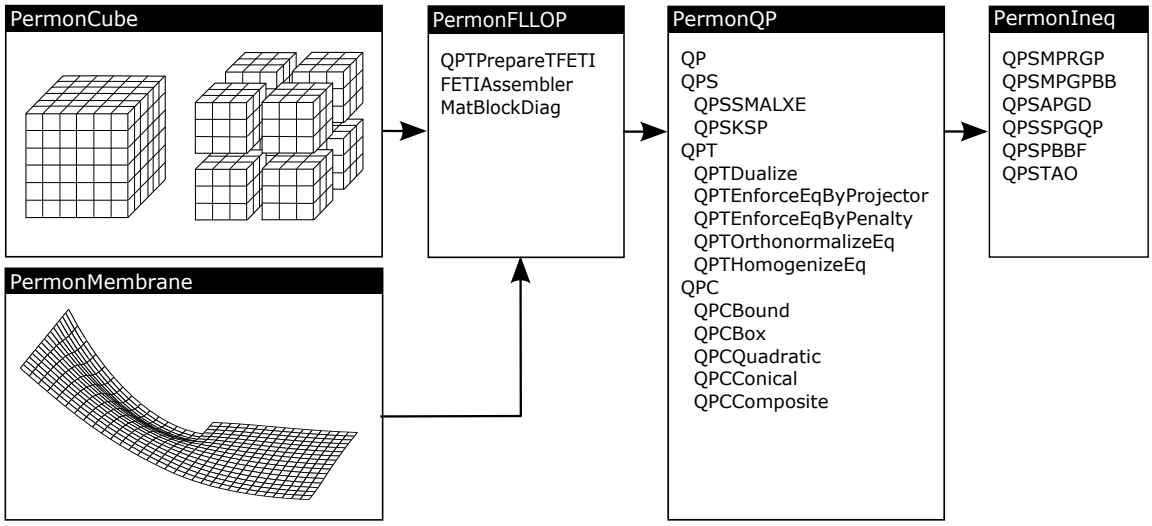


Figure 7.1: Process of solving contact problems with PERMON.

Papers making use of numerical experiments with PERMON include [85–94] and [97–101, 103, 104, 106, 107].

7.1 PermonCube and PermonMembrane

For rapid development and testing of our solvers, PermonMembrane and PermonCube packages [92, 93, 105] were developed. They implement the first and second part of the non-overlapping DDM (Section 4.1) in a massively parallel way for simple benchmarks generated in runtime. PermonCube is similar by focus to the software package Pamgen [30, 76]. Although it provides so far only a cubical mesh, the FEM part of the code does not rely on this specific type of mesh, and works with that as if it were an unstructured mesh, simulating decomposed FEM processing of real world problems. PermonMembrane generates several different simple benchmarks of an elastic membrane or two membranes in mutual interaction, with or without domain decomposition.

The parallel mesh generation is controlled by two groups of parameters. In PermonCube, the number of subdomains is managed by parameters X, Y, Z, and similarly the number of elements per subdomain is given by x, y, z (both considered in the respective axis directions). In PermonMembrane, the situation is similar, only parameters Z and z are missing. The decomposition parameter H and the discretization parameter h is given as $H = \frac{L}{X}$ and $h = \frac{H}{x}$, respectively, where L denotes the size of the whole domain.

PermonCube can be executed from the command line as follows:

```
./runtestmpi 512 -X 8 -Y 8 -Z 8 -x 16 -y 16 -z 16 -c  -@ -qp_E_orth_type gs
```

where 512 is the total number of MPI processes, `-c` means a contact problem and `-@` is a separator of PermonCube and PermonFLLOP parameters, e.g. in this case specifying Gram-Schmidt process for equality constraint matrix orthonormalization.

Essential data, generated by PermonCube, PermonMembrane or any other FEM software, are the subdomain stiffness matrices $\mathbf{K}^s$ and the subdomain RHS vectors $\mathbf{f}^s$, $s = 1, \ldots, N_S$ where N_S denotes the total number of subdomains, $N_S = XYZ$ for PermonCube and $N_S = 2XY$ for PermonMembrane. In the DDM context, an additional object, the previously described local-to-global interface DOF mapping $l2g$, has to be created. These data are passed to PermonFLLOP, described in Section 7.3.

7.2 PERMON Solver Core

PERMON Solver Core consists of the PermonQP and PermonFLLOP modules. They depend on PETSc [2, 3, 67] and use its coding style.

1. **PermonQP** [96] provides a base for solution of linear systems and quadratic programming (QP) problems. It includes data structures, transforms, algorithms, and supporting functions for QP.
2. **PermonFLLOP** [95] (FETI Light Layer on Top of PETSc) is an extension of PermonQP that adds support for DDM of the FETI type.

The combination of DDM and QP algorithms is what makes PERMON unique. These modules are currently under preparation for publishing under the FreeBSD open source license. They will be discussed in the following sections.

7.3 PermonFLLOP

PermonFLLOP (FETI Light Layer on Top of PETSc) [95] is an extension of the PermonQP package, implementing the algebraic part of DDMs of the FETI type (Chapter 4). PermonFLLOP

ad started as a standalone package FLLOP implementing the TFETI DDM (Chapter 4) and had been a predecessor of the whole PERMON suite. Heavy refactoring made more general use possible. There have been two main directions of generalization:

1. FETI can be applied not only to variational equalities (arising e.g. from FEM applied to linear elasticity problems) but also to variational inequalities (arising e.g. from contact problems).
2. The "algebraic" part of the FETI method was generalized to a specific combination of data structures, QP transforms, direct and iterative solvers. Some of these ingredients make sense also out of the scope of the FETI method. For instance, dualization can be useful also for undecomposed problems; on the other hand, decomposed problems can be solved without dualization. Thus these generic ingredients were moved to the PermonQP module whereas the ingredients specific for FETI have remained inside PermonFLLOP.

Let us show how PermonFLLOP is implemented from the user's perspective. First of all, it takes from the FEM software the subdomain stiffness matrices $\mathbf{K}^s$ and the subdomain load vectors $\mathbf{f}^s$ as sequential data for each subdomain Ω^s, $s = 1, \ldots, N_S$. Note that we assume here each processor core owns only one subdomain, PermonFLLOP has nevertheless an experimental feature of allowing more than one subdomain per core, i.e. an array of $\mathbf{K}^s$ and $\mathbf{f}^s$ is passed per subdomain. PermonFLLOP enriches the independent subdomain data with the global context so that $\mathbf{K}$ and $\mathbf{f}$ are effectively created from $\mathbf{K}^s$ and $\mathbf{f}^s$, respectively.

The "gluing" signed Boolean matrix $\mathbf{B}_g$ is created based on the local-to-global mapping $l2g$ as described in [107]. The FEM software can skip the processing of the Dirichlet conditions and rather hand it over to PermonFLLOP, resulting in greater flexibility. PermonFLLOP allows to enforce Dirichlet boundary conditions either by the constraint matrix $\mathbf{B}_d$ (TFETI approach), or by a classical technique of embedding them directly into $\mathbf{K}$ and $\mathbf{f}$ (FETI-1 approach). It is also possible to mix these two approaches.

Furthermore, PermonFLLOP assembles the nullspace matrix $\mathbf{R}$ using one of the following options. The first option is to use a numerical approach [26], and the second one is to generate $\mathbf{R}$ as rigid body modes from the mesh nodal coordinates [91]. The latter is typical for TFETI and is considered here.

Currently, PermonFLLOP requires $\mathbf{B}_I$ and $\mathbf{c}_I$ from the caller. We strive to overcome this limitation in the future so that the non-penetration conditions will be specified in a way more natural for engineers. Listing 7.1 shows how a FEM software (such as PermonCube) typically calls PermonFLLOP to solve a decomposed contact problem.

```
Mat Ks,BIs;  Vec fs,cI,coords;  IS l2g,dbcis;  MPI_Comm comm;  FLLOP fllop;

/*  Generate the data. */

/*  Create FLLOP living in communicator comm. */
FllopCreate(comm, &fllop);
```

```
/*  Set the subdomain stiffness matrix and load vector. */
FllopSetStiffnessMatrix(fllop, Ks);
FllopSetLoadVector(fllop, fs);

/*  Set the local-to-global mapping for gluing. */
FllopSetLocalToGlobalMapping(fllop, l2g);

/*  Specify the Dirichlet conditions in the local numbering
    and tell FLLOP to enforce them by means of the B matrix. */
FllopAddDirichlet(fllop, dbcis, FETI_LOCAL, FETI_DBC_B);

/*  Set vertex coordinates for rigid body modes. */
FllopSetCoordinates(fllop, coords);

/*  Set the non-penetration inequality constraints. */
FllopSetIneq(fllop, BIs, cI);

FllopSolve(fllop);
```

Listing 7.1: PermonCube calls PermonFLLOP

In the `FllopSolve` function, PermonFLLOP passes the global primal data $\mathbf{K}$, $\mathbf{f}$, $\mathbf{B}$ and $\mathbf{R}$ to PermonQP (Section 7.4), calls a specific series of QP transforms (Section 4.5) provided by PermonQP, resulting in the bound and equality constrained QP (4.23), which is then solved with the `QPSSolve` function. Listing 7.2 in Section 7.4 presents a sketch of the `FllopSolve` function.

Open source DDM codes are relatively rare. Let us mention the Multilevel BDDC solver library (BDDCML) by J. Šístek et al. [65, 69], PETSc BDDC preconditioner implementation by S. Zampini [57], and the HPDDM code by P. Jolivet and F. Nataf [34–36] tested with FreeFem++ (FEM software described in Section 6.3.6).

7.4 PermonQP

PermonQP [96] allows solving QPs with an SPS Hessian and any combination of linear equality and inequality constraints including unconstrained QP. It provides a basic framework for QP solution (data structures, transformations, and supporting functions), a wrapper of PETSc KSP linear solvers for unconstrained and equality-constrained QP, a variant of the augmented Lagrangian method called SMALXE (Section 7.4.5, and several concrete solvers for bound constrained minimization (PermonIneq). Its programming interface (API) is designed to be easy-to-use, but also efficient and HPC-oriented.

PermonQP can wrap TAO solvers (Section 6.6.1) and, the other way around, extend TAO with possibility to solve problems with linear equality and/or inequality constraints. The interface is currently under development.

7.4.1 QP transforms

PermonQP separates QP problems, transforms and solvers. QP transforms are implemented in PermonQP according to notions and results in Chapter 2. Let us briefly remind that a QP transform derives from the given original QP a new QP which is simpler or has some better properties. Performing several QP transforms in a sequence results in a so called QP chain (Figure 7.2). It is of course required that the solution of each but the last QP can be efficiently computed from the solution of its corresponding descendant QP.

QP transforms often allow use of efficient solvers that are not compatible with the original QP. However, they are themselves solver-neutral. This is the reason why we decided to decouple them by design from solvers which also implies decoupling of QP problems.

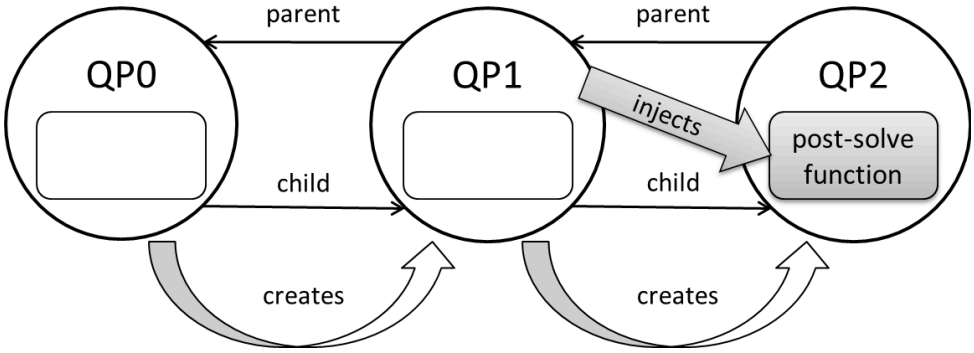


Figure 7.2: QP chain.

The solution process in PermonQP is divided into the following sequence of actions:

1. QP problem specification;
2. a chain of QP transforms generating a chain of QP problems where the last one is passed to the solver;
3. automatic or manual choice of an appropriate QP solver;
4. the QP solver is called to solve the last QP in the chain;
5. a chain of reconstructions in the reverse order of QP transforms to get a solution of the original QP.

PermonQP implements all the QP transforms presented in Chapter 2. For instance, `QPTHomogenizeEq` (Figure 7.3) transforms the original QP with general equality constraints to the new one with homogeneous equality constraints. Let us label the original and derived QP as QP_1 and QP_2, respectively. We only remind that this transform finds a particular solution $\mathbf{x}_P$ satisfying $\mathbf{B}_E\mathbf{x}_P = \mathbf{c}_E$, zeroes the equality constraint RHS and modifies all other RHSs.

In PermonQP, each function representing a QP transform creates a new instance QP_2 of the `QP` class based on the original QP_1. The data objects being altered by the given QP transform are copied from QP_1, modified and stored to QP_2. Otherwise, QP_1 and QP_2 only share pointers

to the same data object. Furthermore, links between QP_1 and QP_2 are created; QP_1 obtains a *child* link to QP_2, QP_2 gets a *parent* link to QP_1 By this means, the QP chain is generated as a doubly linked list (Figure 7.2).

The solution $\bar{\mathbf{x}}_2$ is generally not equal to the solution $\bar{\mathbf{x}}_1$. The asociated reconstruction function $\left(\mathsf{QP}_2 \to \mathsf{QP}_1\right)$ must be called to carry out $\bar{\mathbf{x}}_1 = \left(\mathsf{QP}_2 \to \mathsf{QP}_1\right)(\bar{\mathbf{x}}_2)$. For the above-mentioned case, the reconstruction function is $\left(\mathsf{QP}_2 \to \mathsf{QP}_1\right)(\mathbf{x}) = \mathbf{x} + \mathbf{x}_P$, so it holds that $\bar{\mathbf{x}}_1 = \bar{\mathbf{x}}_2 + \mathbf{x}_P$. In PermonQP, the reconstruction function is injected into the child QP by the transform function. Once the solution of the last QP is computed, the solver triggers a series of the reconstruction functions in LIFO manner, i.e. the reconstruction function of the last QP is called first.

We also need to store somewhere the auxiliary data created by a transform and needed by the asociated reconstruction function. In our case, it is the vector $\mathbf{x}_P$. For this purpose so called *reconstruction context* is used; it is a `void` pointer, injected to the child QP together with the reconstruction function. The notions mentioned above are illustrated by Figure 7.3.

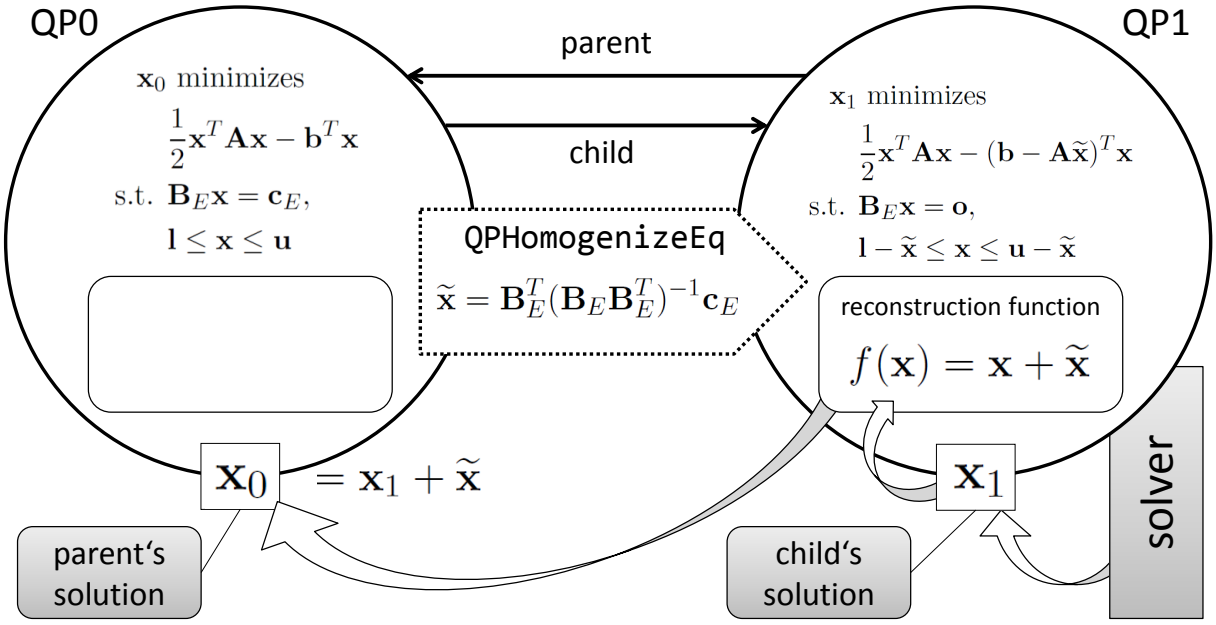


Figure 7.3: Example of QP transform and reconstructions of the solution – homogenization of the equality constraints.

7.4.2 PETSc object design

It has been already mentioned that PERMON is built on top of PETSc. PETSc is designed using strong data encapsulation. Hence, any collection of data (for instance, a sparse matrix) is stored in a way that is completely private from the application code. The application code can manipulate the data only through a well-defined interface, as it does not "know" how the data are stored internally [68].

PETSc is designed around several classes (e.g. `Vec` (vectors), `Mat` (matrices, both dense and sparse)), which inherit from the common abstract class `PetscObject`. These classes are each implemented using C structs, that contain the data and function pointers for operations on the data (much like virtual functions in classes in C++). Each class consists of three parts:

1. a (small) common part shared by all `PetscObjects`.
2. another common part shared by all PETSc implementations of the given class and
3. a private part used by a particular implementation (called type in PETSc).

These parts form a simple 3-level inheritance scheme. For example,

1. all objects (`PetscObject`) share data fields common to all objects such as the communicator and name;
2. additionally, all matrix (`Mat`) types share a function table of operations that may be performed on the matrix and some additional data fields such as the matrix size;
3. additionally, a particular matrix implementation (say compressed sparse row) has its own data fields for storing the actual matrix values and sparsity pattern [68].

7.4.3 PermonQP API

A class used to specify a QP problem is called simply `QP`. It is a data structure containing at least the Hessian matrix **A**, RHS **b** and the solution vector **x**. In PermonQP, these objects are called `Operator, Rhs, SolutionVector`, respectively. To insert them into the `QP` structure, setters named `QPSet<ObjectName>` are called by the user code (e.g. `QPSetOperator`).

Additionally, any combination of these constraints can be specified:

1. equality constraints (`Eq`),
2. inequality constraints (`Ineq`),
3. box constraints (`Box`).

Objects that are not specified (i.e. set to `NULL`) are handled as zero or empty objects with an exception of box constraints, for which `NULL` means an inactive constraint with all values equal to $-\infty$ or ∞ for lower bound or upper bound constraints, respectively. PermonQP allows setting multiple linear equality constraints with `QPAddEq` and uses internally the `MATNEST` composite matrix type from PETSc to handle them.

A `QP` object may point to its parent and child, forming a doubly linked list – the QP chain. Every QP transforms is implemented as a function which takes a `QP` as a first argument. It traverses through the QP chain, following the child pointers and finds the last `QP` in the chain. It then creates a new derived `QP`, setting its parent link to point to the last `QP` and a pointer to the appropriate reconstruction function, corresponding to the particular QP transform. The child link of the last `QP` is set to point to the new `QP` and this new `QP` becomes a new end of the QP chain. A future plan is to implement QP transforms as a separate class.

The third important concept within PermonQP are QP solvers, covered by the `QPS` class. From this abstract class, several types corresponding to different QP algorithms (for instance `QPSSMALXE`) inherit behaviour common to all QP solvers. The type can set explicitly using the function `QPSSetType`, otherwise the function `QPSSetDefaultType` is automatically called in `QPSSetUp`, selecting an appropriate solver compatible with the given constraints. The `QP` to be solved is set by `QPSSetQP`. Other properties are similar to PETSc KSP linear solvers, for example `QPSSetTolerances` can be used to set the relative and absolute convergence tolerance, the divergence tolerance and the maximum number of iterations. Once the problem and algorithm settings are specified, the iterative solution phase can be triggered by calling `QPSSolve`. The solver then automatically finds the last `QP` in the chain, solves this last `QP` and triggers the sequence of reconstruction functions in the reversed order of the QP transforms, so that solution vectors of all `QP` instances in the chain are populated with their actual solutions.

Listing 7.2 shows how the PermonQP API is used within the `FllopSolve` function of PermonFLLOP. This listing serves at the same time as an example of usage of the concepts described above.

```
/* FllopSolve() function */

/*  Subdomain data. */
Mat Ks,BIs,Bgs,Bds,Rs;  Vec fs;
/*  Global data. */
Mat K, BI, Bg, Bd, R ;  Vec f, cI, cd;
/*  QP problem, QP solver. */
QP qp; QPS qps;

/*  Create a QP data structure. */
QPCreate(comm, &qp);

/*  Globalise the data. */
MatCreateBlockDiag(Ks, &K);
MatCreateBlockDiag(Rs, &R);
MatMerge(Bgs, &Bg); MatMerge(Bds, &Bd);
MatMerge(BIs, &BI); VecMerge(fs, &f);

/*  Set the QP data. */
QPSetOperator(qp, K);
QPSetOperatorNullspace(qp, R);
QPSetRHS( qp, f);
QPAddEq(  qp, Bg, NULL);
QPAddEq(  qp, Bd, cd);
QPSetIneq(qp, BI, cI);

/*  Basic sequence of QP transforms
    giving (T)FETI method.
    QPTFetiPrepare() can be used
    instead for convenience.
    QP chain is created in backend. */
QPTScale(qp);
QPTDualize(qp);
QPTScale(qp);
QPTHomogenizeEq(qp);
QPTEnforceEqByProjector(qp);

/*  Create a PermonQP solver. */
QPSCreate(comm, &qps);

/*  Set the QP to be solved. */
QPSSetQP(qps, qp);

/*  Solve, i.e. hand over to PermonQP.
    The last QP in the chain is solved.
*/
QPSSolve(qps);
```

Listing 7.2: PermonFLLOP calls PermonQP

7.4.4 Linear operators

PETSc matrices (`Mat`) are in fact more general than the name might suggest. They do not include only "explicit matrices" whose entries are directly accessible, but more generally linear operators on finite dimensional vector spaces ("implicit matrices"), whose entries may or not

be available but the matrix-vector product (operator application) always is. PermonQP adds several new such linear operators mainly due to enable elegant implementation of QP transforms and domain decomposition.

MATBLOCKDIAGMPI represents a distributed block-diagonal operator made of local (one per an MPI process) sequential blocks of arbitrary type. It is used to implement e.g. $\mathbf{K}$, $\mathbf{K}^\dagger$ and $\mathbf{R}$ in TFETI (Section 4.3).

MATBLOCKDIAGSEQ abstracts a local block-diagonal operator made of smaller sequential blocks. It is owned by one MPI process but it supports finer shared memory parallelization using OpenMP threading where each thread owns one or more blocks, provided the blocks are thread-safe. It can be used as a local block of `MATBLOCKDIAGMPI`, resulting hybrid MPI+OpenMP parallelization.

MATINV serves as a matrix inverse in the implicit form. It wraps the PETSc `KSPSolve` method of the `KSP` class into the `MatMult` method of the `Mat` class. The former serves for solving linear systems, the latter carries out the matrix-vector product or, in the linear map speech, applies the linear operator. Additionally, `MATINV` implements also a generalized inverse of a matrix with the known kernel – the approach from [9] is used where the original matrix is regularized and the inverse of the regularized matrix is a pseudoinverse of the original matrix. `MATINV` is used for $(\mathbf{K}^\dagger)^s$ and $(\mathbf{G}\mathbf{G}^T)^{-1}$ in TFETI (Sections 4.3 and 7.5). In case of $\mathbf{K}^\dagger$, it is used together with `MATBLOCKDIAGSEQ` and/or `MATBLOCKDIAGMPI`.

MATSUM and MATPROD represent an implicit matrix-matrix sum and product, respectively. It provides a matrix-vector product which makes use of the matrix-vector products of the underlying matrices. For example the operator $\mathbf{PFP}$ in (4.15) is implemented as `MATPROD` whose matrix-vector product with a vector $\mathbf{x}$ is carried out as $\mathbf{PFPx} = \mathbf{Px} + \mathbf{Fx} + \mathbf{Px}$.

7.4.5 SMALXE

A key part of the PermonQP package is a particular subclass of the `QPS` class called `QPSSMALXE`. It implements SMALBE-M (Algorithm 1, Section 3.2) in a slightly generalized manner. This section will discuss selected modifications with respect to the original algorithm.

`QPSSMALXE` takes care of the linear equality constraints while the QP with the "rest of constraints" and the penalization term in its objective is passed to the inner solver, which is a separate `QPS` instance. Based on the "rest of constraints", the inner `QPS` can be any solver for unconstrained, bound constrained, box constrained QP or generalized QP with separable convex constraints.

Hence, our SMALXE algorithm covers SMALE (SemiMonotonic Augmented Lagrangian for Equality constrained QP [10]) SMALBE (SemiMonotonic Augmented Lagrangian for Bound and Equality constrained QP, Section 3.2) for bound and equality constrained QP and SMALSE

(SemiMonotonic Augmented Lagrangian for Separable and Equality constrained QP [14]) – the actual variant is given only by the actual inner `QPS` instance.

`QPSSMALXE` injects into the inner solver a specific stopping criterion according to the line 4 in Algorithm 1, following the *inversion of control (IoC)* design principle – the inner solver implementation itself "does not know" about SMALXE and has no specific code related to it.

7.4.6 General QP solver

Let us show that PermonQP can be used to solve any QP of the form (1.5).

1. In case of *unconstrained* QPs, PermonQP makes use of the PETSc (Section 6.4.1) KSP package which includes both direct and iterative linear system solvers, including interfaces to many external solvers.
2. For solution of *bound or box constrained* QPs, several special concrete solvers are implemented. The default one is MPRGP described in this work (Section 3.1). We call this suite of solvers for bound constrained problems PermonIneq. It had been described as a standalone package in our earlier works but it is now considered a part of PermonQP. We are currently implementing a wrapper of TAO (Section 6.6.1) which will provide several new algorithms.
3. To enforce sole *equality constraints*, the orthogonal projectors (Section 2.3.6), penalty method (Section 2.3.4) or SMALXE algorithm (Section 7.4.5) can be used. In all three cases, the original equality constrained QP is transformed into an unconstrained one, i.e. reduced to the case of Item 1.
4. *Bound and equality* constrained QPs can be solved by SMALXE (Section 7.4.5) which "filters out" the equality constraints, moving them into the Hessian. The auxiliary problem with the modified Hessian and just the bound constraints is then passed to the inner solver for bound constrained QP (Item 2).
5. QPs with *general linear inequality constraints* can be transformed into bound constrained ones using dualization (Section 2.3.7). Item 2 then may be applied.
6. QPs with *general linear equality and inequality constraints* can be transformed into bound and equality constrained ones using dualization (Section 2.3.7). Item 4 then may be applied.

7.5 Direct solvers in PERMON Solver Core

FETI methods blend iterative and direct solvers. The main loop solving dual problem is solved by an iterative solver, e.g. conjugate gradients (CG). In each iteration, auxiliary problems related to the application of an unassembled system operator are solved: (1) action of the stiffness matrix pseudoinverse and (2) the coarse problem.

The first auxiliary problem is the stiffness matrix pseudoinverse action. The stiffness matrix is perfectly block-diagonal with each block corresponding to one subdomain – we call them subdomain stiffness matrices. Each subdomain is entirely owned by a single process and so is the block. The stiffness matrix pseudoinverse is again block-diagonal where each block is the pseudoinverse of the corresponding subdomain stiffness matrix, carried out by the owning process. This action does not include any data transfers, so it is an embarrassingly parallel problem.

The second auxiliary problem is the coarse problem (CP) appearing in the application of the projector onto the kernel of so called natural coarse space matrix. The CP couples all subdomains and accelerates convergence. However, this problem does not possess such a nice structure and some communication is needed in this case.

Natural effort using the massively parallel computers is to maximize the number of subdomains so that sizes of subdomain stiffness matrices are reduced which accelerates their factorization and subsequent forward/backward substitutions carrying out the pseudoinverse applications. On the other hand, negative effect of that is an increase of the null space dimension and the number of Lagrange multipliers on the subdomain interfaces (dual dimension), which decelerate the CP solution. It can hardly be solved sequentially on the master core for large scale problems; thus it becomes a bottleneck. Let us attempt to address this issue.

This section and the corresponding numerical experiments in Section 8.2 are gathered from the author's papers [88, 89, 98, 90, 104].

7.5.1 Local and coarse problems in TFETI

It is obvious from theory as well as from practical experiments that FETI has two major hotspots: (1) $\mathbf{K}^\dagger$ application and (2) CP solution. They are both implemented using PermonQP's `MATINV` (Section 7.4.4). Direct solvers[1] are typically employed for the sake of robustness – solving realistic problems, slow convergence or even divergence of the top-level solver occurs when inexact solvers are used.

Parallelization is achieved mainly by distributing diagonal blocks of $\mathbf{K}$ over processors, each block reflecting a subdomain. We strive to maximize the number of subdomains to reduce the sizes of the subdomain stiffness matrices, accelerating their factorization and $\mathbf{K}^\dagger$ actions. Furthermore, thanks to the estimate (4.17), decomposition into more subdomains maintaining the fixed discretization leads to reduction of the condition number of $\mathbf{K}$ and thus the number of iterations.

A drawback is increasing null space dimension decelerating the CP solution – it is a kind of a communicating vessels effect. Furthermore, for a sufficiently large number of subdomains, the CP matrix ($\mathbf{G}\mathbf{G}^T$) may be too large to fit into the memory of one computational unit.

[1] triangular solves (forward and backward substitutions) with triangular matrices obtained from the complete Cholesky or LU factorization of the original matrix

Unfortunately, the CP matrix is not block-diagonal. Thus, it is inevitable to use some *parallel* sparse direct solver for the CP solution.

7.5.2 Stiffness matrix pseudoinverse action

The stiffness matrix $\mathbf{K}$ as well as its *pseudoinverse* $\mathbf{K}^\dagger$ possess a perfect block-diagonal layout and can be implemented using a block-diagonal matrix composite type where subblocks are sequential matrices. Let us denote the total number of subdomains by N_S, the index of current subdomain by $s = 1, \ldots, N_S$ and the associated $\mathbf{X}$-object's portion by $\mathbf{X}_s$, in accordance with notation in Section 4.3.

The pseudoinverse $\mathbf{K}^\dagger$ is implemented in the following way. During the preprocessing phase, each core regularizes the subdomain stiffness matrix $\mathbf{K}_s$ using the method from [5, 9],

$$\mathbf{K}_s^{reg} = regularize(\mathbf{K}_s),$$

yielding the regularized matrix $\mathbf{K}^{reg}$ satisfying

$$(\mathbf{K}^{reg})^{-1} = (\mathbf{K})^\dagger.$$

Obviously, the same holds for the subdomain-wise diagonal blocks,

$$(\mathbf{K}_s^{reg})^{-1} = (\mathbf{K}_s)^\dagger, \quad s = 1, \ldots, N_S.$$

Each regularized block $\mathbf{K}_s^{reg}$ is factorized using LU or Cholesky factorization,

$$(\mathbf{L}_s^{\mathbf{K}}, \mathbf{U}_s^{\mathbf{K}}) = factorize(\mathbf{K}_s^{reg}),$$

so that it holds

$$\mathbf{K}_s^{reg} = \mathbf{L}_s^{\mathbf{K}} \mathbf{U}_s^{\mathbf{K}},$$

and $\mathbf{L}_s^{\mathbf{K}}$ and $\mathbf{U}_s^{\mathbf{K}}$ is a lower triangular and upper triangular matrix, respectively. In case of Cholesky factorization, $\mathbf{U}_s^{\mathbf{K}} = (\mathbf{L}_s^{\mathbf{K}})^T$.

The application of $\mathbf{K}^\dagger$ then consists of purely local forward and backward substitutions once in each iteration:

$$\mathbf{K}_s^\dagger \mathbf{v}_s = \mathbf{U}_s^{\mathbf{K}} \backslash (\mathbf{L}_s^{\mathbf{K}} \backslash \mathbf{v}_s)$$

(using $\backslash$ symbol the same way as in MATLAB). It is obvious that these so-called *subdomains problems* are purely local and therefore parallelizable without any data transfers – they are *embarrassingly parallel*.

7.5.3 Coarse problem solution

The natural coarse space matrix $\mathbf{G}$ is computed in such a way that each of the cores owns the sparse sequential matrices $\mathbf{R}_s$ and $\mathbf{B}_s$, so that this core computes local block $\mathbf{G}_s = \mathbf{R}_s^T \mathbf{B}_s^T$ of $\mathbf{G}$

matrix without any communication (Figure 7.4). We then redistribute the horizontal sequential sparse blocks $\mathbf{G}_s$ into vertical ones (i.e. horizontal $\mathbf{G}_s^T$). The sparsity pattern of $\mathbf{G}$ for the cube decomposed into 8 subdomains is illustrated in Figure 7.5.

According to the observations, the actions of $\mathbf{G}$ and $\mathbf{G}^T$ take approximately the same time for different $\mathbf{G}$ matrix distributions (assembled $\mathbf{G}$ distributed into horizontal blocks, assembled $\mathbf{G}$ distributed into vertical blocks, unassembled $\mathbf{G}$ kept in the form $\mathbf{R}^T\mathbf{B}^T$). So the time and level of communication in actions of the projector $\mathbf{Q} = \mathbf{G}^T(\mathbf{G}\mathbf{G}^T)^{-1}\mathbf{G}$ depend primarily on the implementation of the CP solution

$$\mathbf{G}\mathbf{G}^T\mathbf{x} = \mathbf{y},$$

which can hardly be solved sequentially on the master core for large scale problems because of the memory limitations and inefficiency as noted above. The sparsity pattern of $\mathbf{G}\mathbf{G}^T$ for the cube decomposed into 8 subdomains is illustrated in Figure 7.6.

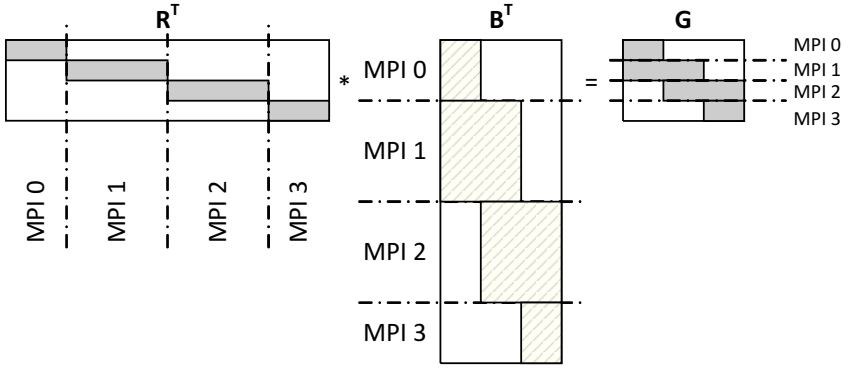


Figure 7.4: Distributed computation $\mathbf{G} = \mathbf{R}^T\mathbf{B}^T$.

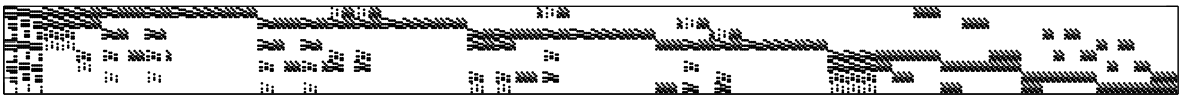


Figure 7.5: The sparsity pattern of $\mathbf{G}$.

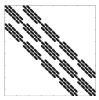


Figure 7.6: The sparsity pattern of $\mathbf{G}\mathbf{G}^T$.

We have suggested and compared several strategies for parallel CP solution [89, 104]:

1. directly using LU or Cholesky factorization,
2. applying explicit inverse of $\mathbf{G}\mathbf{G}^T$,

3. iteratively using conjugate gradients (with Jacobi preconditioner),
4. orthonormalizing rows of G so that the CP is eliminated.

The orthonormalization approach with the classical Gram-Schmidt method starts to fail when the nullspace is large enough (thousands) because round-off errors become an issue whereas the modified or iterative Gram-Schmidt methods have better numerical properties but are less scalable. So we have abandoned this approach so far.

The iterative approach destroys robustness of the FETI method if the relative tolerance of the iterative solver is greater than $\sim 10^{-10}$. Still this approach could possibly be favourable if the iterative solver was equipped with a proper preconditioner and/or deflation technique. But we have so far not managed to find these.

Therefore, we will further speak only about the first two strategies. Let us now describe them in detail.

Strategy 1. $\mathbf{GG}^T$ is factorized in the preprocessing phase. During the solution phase, each application of $(\mathbf{GG}^T)^{-1}$ consists of the forward and backward substitution using a parallel direct solver:

$$(\mathbf{GG}^T)^{-1}\mathbf{w} = \mathbf{U}^{\mathbf{GG^T}} \backslash (\mathbf{L}^{\mathbf{GG^T}} \backslash \mathbf{w}).$$

Strategy 2. A parallel direct solver is employed for the computation of the *explicit inverse* of $\mathbf{GG}^T$. During the preprocessing phase, $\mathbf{GG}^T$ is factorized and then the explicit inverse $(\mathbf{GG}^T)^{-1}$ is computed. In the solution phase, its application consists in the parallel dense matrix-vector product $(\mathbf{GG}^T)^{-1}\mathbf{w}$.

Concerning use of a direct solver, the CP dimension is not large enough to justify the fully parallel approach, i.e. using the whole global communicator – communication would take over computation for large enough number of subdomains. Instead, we propose a proper *partial* parallelization of this CP solution, i.e. using groups of processes (subcommunicators). We divide all processes of the global `PETSC_COMM_WORLD` communicator into the subcommunicators using PETSc built-in "pseudopreconditioner" `PCREDUNDANT`; the number of these subcommunicators is N_r (number of cores doing redundant work); this means the number of cores in each subcommunicator is $\approx N_c/N_r$.

The explicit inverse is assembled in the following way. Each of N_r subcommunicators is assigned a contiguous portion of N_n/N_r columns of the identity matrix taken as RHS. The result of the forward/backward substitutions is the corresponding portion of N_n/N_r columns of the resulting explicit inverse $(\mathbf{GG}^T)^{-1}$, stored as a $N_n \times (N_n/N_r)$ dense matrix distributed vertically across the subcommunicator. Taking advantage of the symmetry of $(\mathbf{GG}^T)^{-1}$, each subcommunicator's block is transposed in parallel and the blocks are then merged one below the other in the proper order forming the complete $(\mathbf{GG}^T)^{-1}$ matrix, distributed vertically across the global communicator. Note that this merge means only logical reassignment from

the subcommunicator to the global communicator with no actual data movements. A scheme of this strategy is depicted in Figure 7.7.

Numerical tests of the aforementioned approaches are presented in Section 8.2.

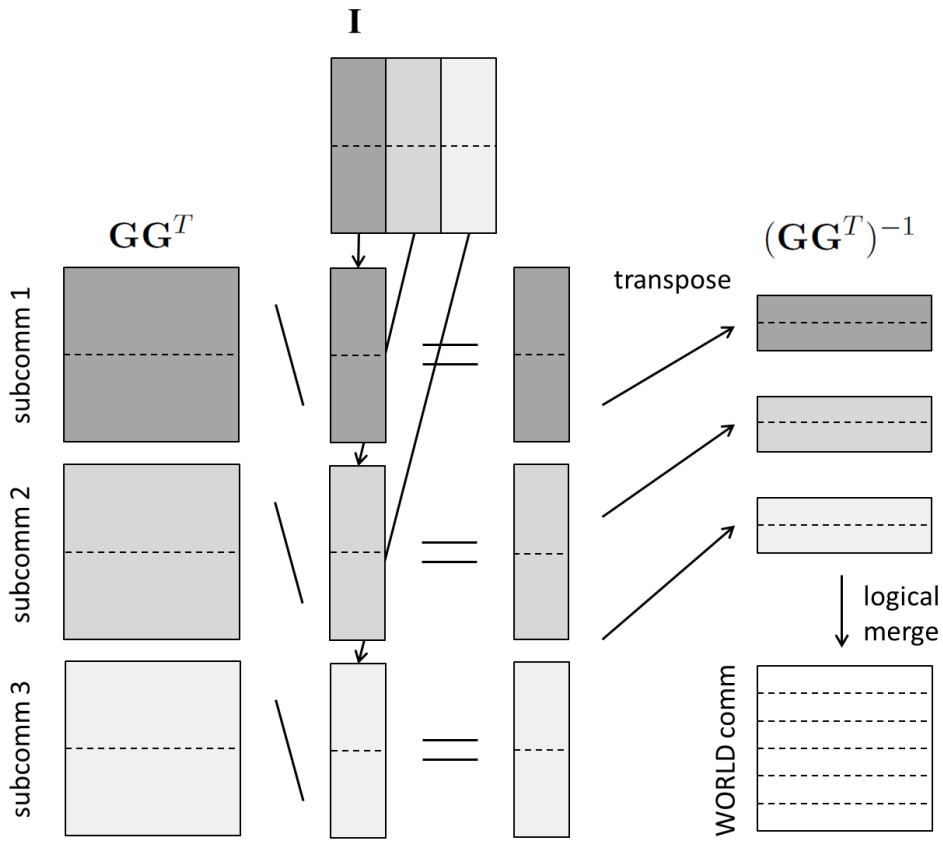


Figure 7.7: Scheme of $(\mathbf{G}\mathbf{G}^T)^{-1}$ implementation using Strategy 2.

CHAPTER VIII

Numerical experiments with PERMON

To conclude this thesis, several numerical experiments are presented. Section 8.1.1 introduces the machines used for the experiments. Section 8.2 focuses on comparison of direct solvers used for the stiffness matrix pseudoinverse and CP solution in PERMON Solver Core, following general discussion in Section 7.5. Section 8.3 presents PERMON performance results for model benchmarks generated by PermonCube and PermonMembrane (Section 7.1). Section 8.4 shows that PERMON is able to participate in solving real-world problems.

The same notation as in Section 4.3 is used: total number of subdomain, primal dimension (number of DOFs after decomposition), dual dimension and stiffness matrix kernel dimension (defect) are denoted by N_S, n, m and d, respectively. Within this work, the number of used processes is always equal to N_S and hence no extra symbol is introduced.

8.1 Machines

Let us introduce the supercomputers used for our numerical experiments.

8.1.1 ARCHER

As of March 2016, ARCHER is the latest UK national supercomputing service, #41 in the current TOP500 list [77]. It is based around a Cray XC30 supercomputer with 4920 nodes, 118,080 cores and 1.56 Pflops of theoretical peak performance. It consists of 4544 standard nodes with 64 GB memory (12 groups, 109,056 cores) and 376 nodes with 128 GB memory (1 group, 9024 cores). All compute nodes are connected together in the Dragonfly topology by

the Aries interconnect. Each compute node contains two 2.7 GHz, 12-core E5-2697 v2 (Ivy Bridge) series processors. Within the node, the two processors are connected by two QuickPath Interconnect (QPI) links. The memory is arranged in a non-uniform access (NUMA) form: each 12-core processor is a single NUMA region with local memory of 32 GB (or 64 GB for high-memory nodes).

8.1.2 HECToR

The HECToR supercomputer [29] was a predecessor of ARCHER (Section 8.1.1). It had been operated by EPCC before its decommission in 2014. In our experiments, we used the last Phase 3 system (Cray XE6). This system was contained in 30 cabinets and comprised of a total of 704 compute blades. Each blade contained four compute nodes giving a total of 2816 compute nodes, each with two 16-core AMD Opteron 2.3GHz Interlagos processors. This amounts to a total of 90,112 cores. Each 16-core socket was coupled with a Cray Gemini routing and communications chip. Each 16-core processor shared 16 GB of memory. The theoretical peak performance of the phase 3 system was over 800 Tflops.

8.1.3 Salomon

The Salomon cluster is operated by IT4Innovations National Supercomputing Center, Czech Republic. It is #48 in the current TOP500 list [77]. It consists of 1008 compute nodes, giving a total of 24,192 compute cores with 129TB RAM and over 2 Pflops theoretical peak performance. Each node is a x86–64 computer with two Intel Xeon E5–2680v3 12-core processors (24 cores per node) and at least 128GB RAM. Nodes are interconnected by 7D Enhanced hypercube Infiniband network. Salomon consists of 576 nodes without accelerators and 432 nodes equipped with Intel Xeon Phi MIC accelerators.

8.2 Evaluation of direct solvers

Following Section 7.5, efficiency of MUMPS and SuperLU sparse direct solvers (Section 6.5) is compared here for the hotspots of the FETI method: the $\mathbf{K}^{\dagger}$ action and CP solution. Matrices and vectors for numerical experiments were obtained from a regular decomposition and discretization of a model problem of an elastic cube with edge length of 1 mm, Young modulus 2.0e+5 MPa and Poisson ratio 0.3. Dirichlet boundary conditions are prescribed on three sides (for each side there are zero displacements in normal direction). On one of the free sides there is a Neumann condition with pressure 10 MPa in the normal direction. To illustrate the efficiency of various direct solvers we used a discretization of the cube into 4,096,000 elements and a decomposition into 512; 1000; 4096; 8000 subdomains. The regular discretizations and

decompositions were chosen to ensure a uniform workload of all cores. The benchmarks were run on HECToR (Section 8.1.2).

8.2.1 Results for pseudoinverse action

The time for the preprocessing and the average time for one application of $\mathbf{K}^{\dagger}$ for PETSc, MUMPS and SuperLU are shown in Table 8.1 and graphically illustrated in Figures 8.1 and 8.2 (log scale); there is a nice time reduction due to the decomposition of the domain into more and more subdomains, decreasing the local matrix dimension. The best results were obtained with the MUMPS library. For the factorization of the problem with the largest subdomain dimension, SuperLU was 10 times worse and PETSc built-in LU even 25 times worse. It is obvious that the multifrontal approach is very suitable for the subdomain stiffness matrix's structure.

	N_S	512	1000	4096	8000
direct	n/N_S	27,783	14,739	3993	2187
solver	iter. count	30	26	18	15
PETSc	$\mathbf{K}^{reg}$ fact.	2.03e+02	7.44e+01	3.35e+00	8.06e-01
	$\mathbf{K}^{\dagger}$ actions	2.64e-01	1.17e-01	1.90e-02	9.05e-03
	fact. + all actions	210.8	77.4	3.7	0.9
MUMPS	$\mathbf{K}^{reg}$ fact.	7.93e+00	3.05e+00	5.72e-01	2.11e-01
	$\mathbf{K}^{\dagger}$ actions	1.50e-01	6.34e-02	1.76e-02	9.11e-03
	fact. + all actions	12.4	4.7	0.9	0.3
SuperLU	$\mathbf{K}^{reg}$ fact.	8.55e+01	1.88e+01	1.49e+00	5.32e-01
	$\mathbf{K}^{\dagger}$ actions	6.38e-01	2.11e-01	3.04e-02	1.35e-02
	fact. + all actions	104.6	24.3	2.0	0.7

Table 8.1: Performance of $\mathbf{K}^{reg}$ factorization / $\mathbf{K}^{\dagger}$ action / factorization + all actions for varying decompositions in seconds.

8.2.2 Results for coarse problem solution

The performance results of direct solvers for varying decomposition and sequential CP solution are depicted in Table 8.2 and illustrated in Figures 8.3 and 8.4.

Figures 8.1 to 8.4 illustrate the communicating vessels effect: the computational savings for $\mathbf{K}^{reg}$ factorization and $\mathbf{K}^{\dagger}$ action reached by the decomposition into more subdomains are eliminated by an increasing computational and communication requirements for the CP solution – $\mathbf{G}\mathbf{G}^T$ factorization and $(\mathbf{G}\mathbf{G}^T)^{-1}$ action.

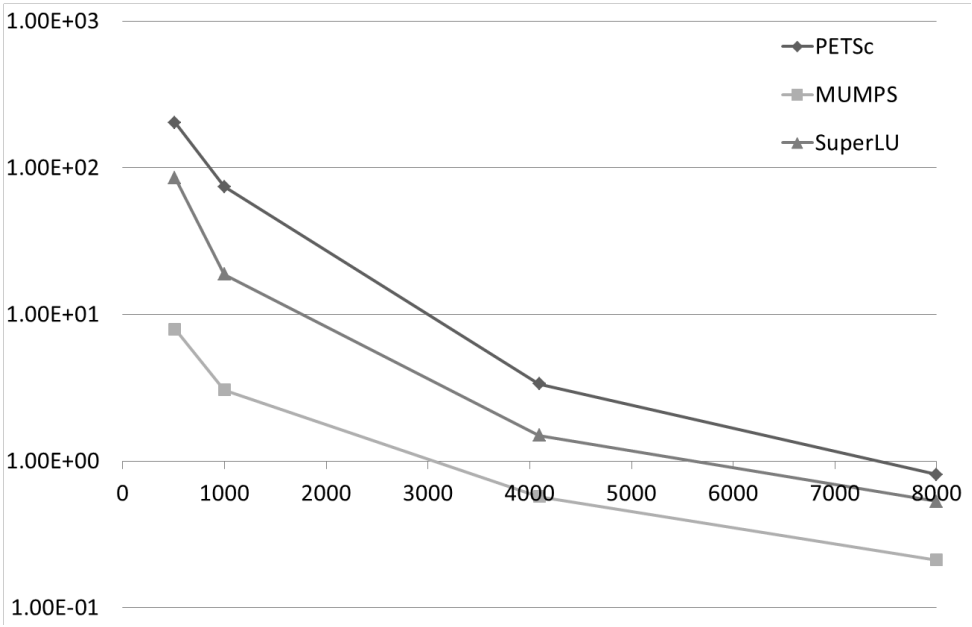


Figure 8.1: Times for $\mathbf{K}^{reg}$ factorization (log. scale)

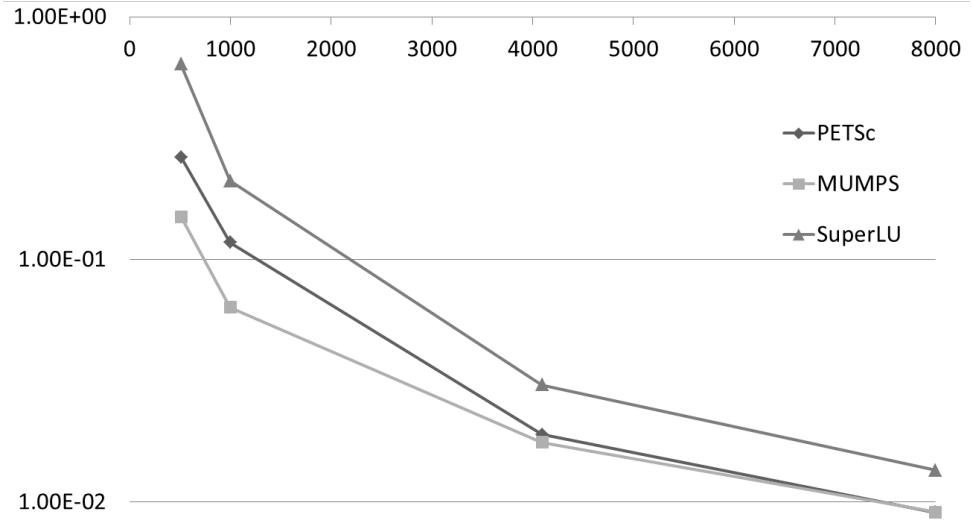


Figure 8.2: Times for $\mathbf{K}^{\dagger}$ action (log. scale)

direct solver	N_S / d	512 / 3072	1000 / 6000	4096 / 24,576	8000 / 48,000
PETSc	$\mathbf{G}\mathbf{G}^T$ fact.	1.07e+00	4.55e+00	8.50e+01	3.37e+02
	$(\mathbf{G}\mathbf{G}^T)^{-1}$ actions	1.28e-02	3.24e-02	2.36e-01	6.21e-01
	fact. + all actions	1.5	5.4	89.2	346.7
MUMPS	$\mathbf{G}\mathbf{G}^T$ fact.	2.73e-01	7.34e-01	7.94e+00	2.65e+01
	$(\mathbf{G}\mathbf{G}^T)^{-1}$ actions	1.12e-02	2.32e-02	1.34e-01	2.99e-01
	fact. + all actions	0.6	1.3	10.4	31.0
SuperLU	$\mathbf{G}\mathbf{G}^T$ fact.	1.14e+00	3.67e+00	4.28e+01	1.78e+02
	$(\mathbf{G}\mathbf{G}^T)^{-1}$ actions	4.49e-02	1.24e-01	8.63e-01	2.42e+00
	fact. + all actions	2.5	6.9	58.4	214.2

Table 8.2: Performance of sequential $\mathbf{G}\mathbf{G}^T$ factorization / $(\mathbf{G}\mathbf{G}^T)^{-1}$ action / factorization + all actions on the master core for varying decompositions in seconds.

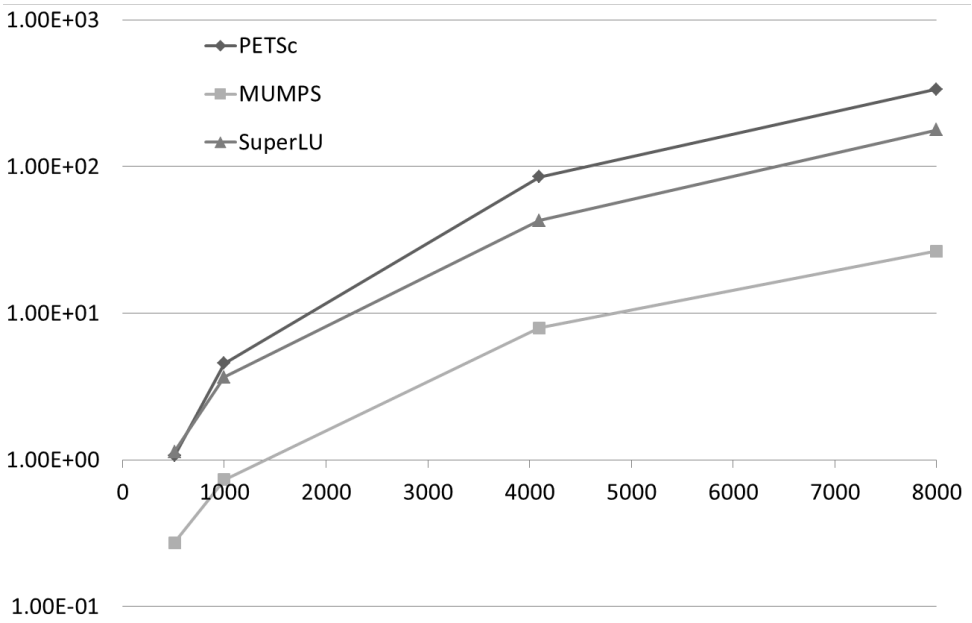


Figure 8.3: Times for $\mathbf{G}\mathbf{G}^T$ factorization in seq. case (log. scale)

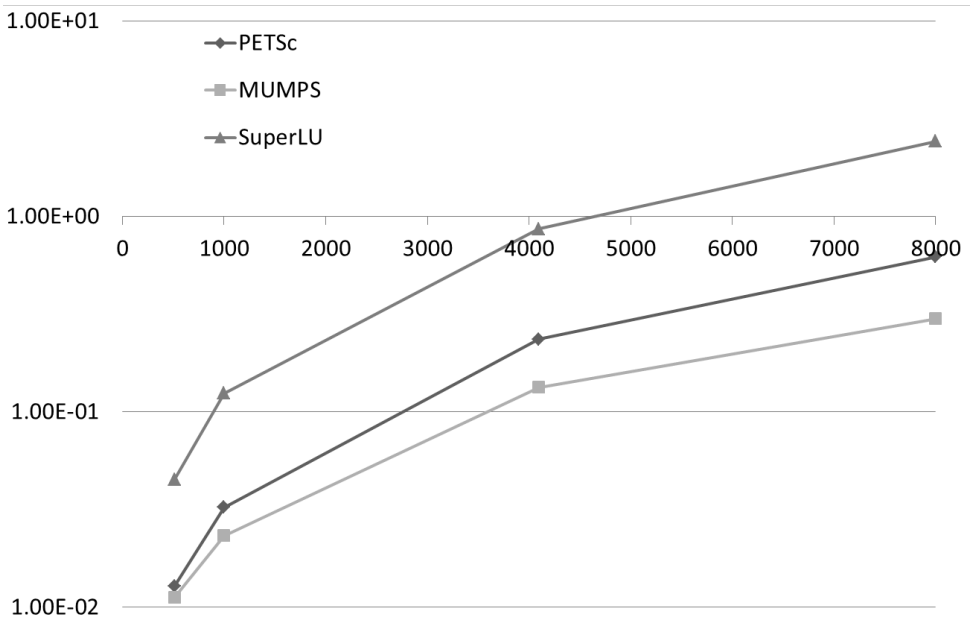


Figure 8.4: Times for $(\mathbf{G}\mathbf{G}^T)^{-1}$ action in seq. case (log. scale)

Significant improvement can be achieved by means of the partial parallelization of the CP solution and one of two strategies described in Section 7.5.3. The results are shown in Tables 8.3 and 8.4, and graphically in Figures 8.5 and 8.6.

Concerning Strategy 1, the supernodal approach represented by the SuperLU_DIST library proved to be more suitable than the multifrontal approach for the given type of matrix; it offers better scalability of the forward/backward substitution so that the CP action's time can be reduced significantly when a high number of cores per subcommunicator is used – the optimal number for our problem is 800 which corresponds to $N_r = 10$.

Concerning Strategy 2, results show that a low number of columns of $(GG^T)^{-1}$ per subcommunicator is more important than a high number of processors engaged in the factorization and solution. Thus, in contrary to Strategy 1, low number of cores per subcommunicator should be used. In this case, the better scalability of SuperLU_DIST forward/backward substitutions has no impact, and MUMPS gives slightly better times. The optimal number of cores per subcommunicator is in our case 8 which corresponds to $N_r = 1000$.

Let us now compare these two strategies. It is obvious from Figures 8.5 and 8.6 that Strategy 1 gives better times when the number of iterations is lower than ~ 500. For higher numbers of iterations the Strategy 2 starts to win although having more expensive preprocessing. This can be interesting for ill conditioned elasto-static problems but even more interesting for contact problems where the number of iterations is always higher. Finally, the greatest effect will be seen for all problems that are solved using outer iteration on top of FETI: shape optimization, elasto-plasticity, transient problems.

	direct solver	N_r / N_S/N_r	2000 4	1000 8	500 16	125 64	20 400	10 800	5 1600
factorization	M		23.3	15.4	16.1	17.7	11.8	12.2	9.1
	S		29.4	20.4	16.2	9.1	6.4	6.6	6.8
1 action	M		0.90	0.41	0.26	0.20	0.14	0.18	0.23
	S		1.03	0.50	0.41	0.28	0.14	0.06	0.11
fact. + 100 actions	M		114	56.1	42.1	37.9	25.4	29.7	32.5
	S		132	70.3	56.9	36.7	20.8	**12.7**	17.3
fact. + 1000 actions	M		926	422	276	220	148	187	243
	S		1059	519	423	285	150	**68.2**	112

Table 8.3: Performance of MUMPS (M) and SuperLU_DIST (S) for Strategy 1 depending on the subcommunicator's size for the decomposition into 8000 subdomains (in seconds); the best variant is printed in bold.

		N_r	2000	1000	500	125	20	10	5
	direct	N_S/N_r	4	8	16	64	400	800	1600
	solver	d/N_r	24	48	96	384	2400	4800	9600
factorization	M		23.3	15.4	16.1	17.7	11.8	12.2	9.1
	S		29.4	20.4	16.2	9.1	6.4	6.6	6.8
inv. computation	M		21.7	19.5	25.0	77.6	326	840	2246
(d/N_r cols per subcomm)	S		24.7	24.0	39.1	106	346	296	1008
redistribution			0.02	0.03	0.04	0.11	2.1	2.5	3.5
1 action						0.0072			
setup + 100 actions	M		45.7	**35.7**	41.8	96.1	341	855	2260
	S		54.9	45.1	56.0	116	355	305	1019
setup + 1000 actions	M		52.2	**42.2**	48.3	103	348	862	2266
	S		61.4	51.6	62.6	122	361	312	1026

Table 8.4: Performance of MUMPS (M) and SuperLU_DIST (S) for Strategy 2 depending on the subcommunicator's size for the decomposition into 8000 subdomains (in seconds); the best variant is printed in bold.

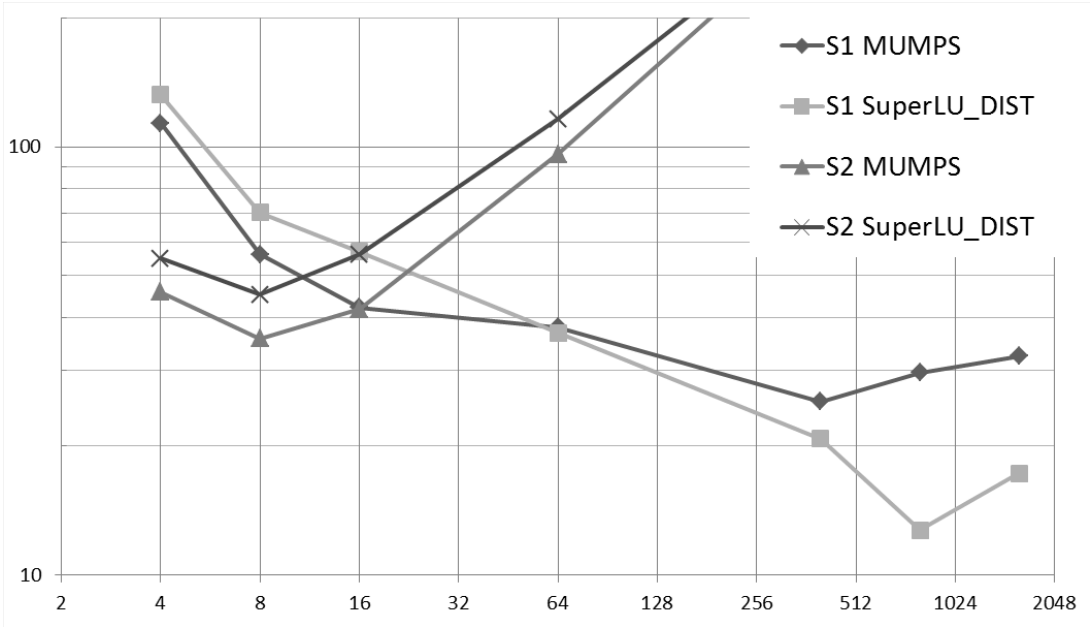


Figure 8.5: Times of CP preprocessing and 100 $(\mathbf{G}\mathbf{G}^T)^{-1}$ actions depending on the subcommunicator's size, the strategy (S1 = Strategy 1, S2 = Strategy 2), and the direct solver for the decomposition into 8000 subdomains.

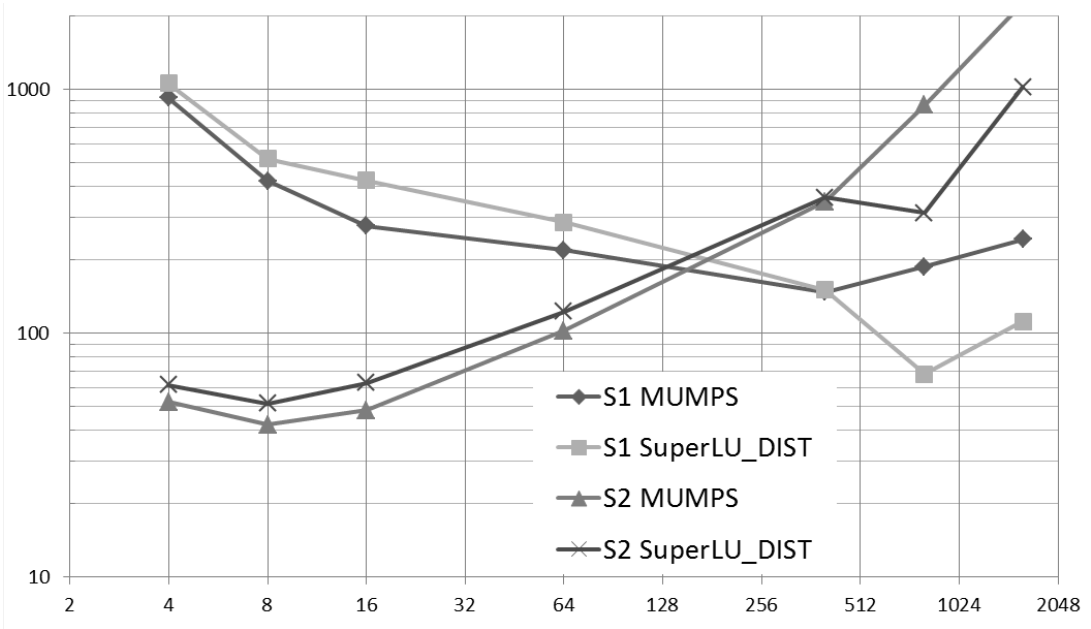


Figure 8.6: Times of CP preprocessing and 1000 $(\mathbf{G}\mathbf{G}^T)^{-1}$ actions depending on the subcommunicator's size, the strategy (S1 = Strategy 1, S2 = Strategy 2), and the direct solver for the decomposition into 8000 subdomains.

8.2.3 Summary

Without CP parallelization we are not able to solve large problems because the whole CP resides in the master process' memory which is of course limited. Furthermore, the master performs the sequential computation while all other cores have to wait; this breaks the scalability of the whole method. So there is no other way than using some parallel direct solver. On the other hand, engaging all processes in "world" communicator into the CP solution leads to an enormous communication overhead. Therefore, we conclude that the CP should be solved only in the subcommunicators of appropriate size.

Two strategies for CP solution were introduced in Section 7.5.3:

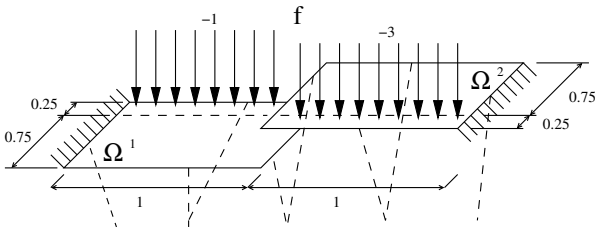
1. factorization + forward/backward substitutions,
2. factorization + explicit inverse assembly + dense matrix-vector products.

It was mentioned that the choice of the strategy depends on the expected number of iterations given by the class of the solved problem. For more then 500 iterations, Strategy 2 with more expensive preprocessing starts to pay off. For the used Cray XE6 architecture, its vendor supplied libraries and our PETSc-based implementation (PermonFLLOP + PermonQP) we can recommend following approaches: Strategy 1: SuperLU_DIST with N_r=16; Strategy 2: MUMPS with $N_r = 1000$.

8.3 Model contact problems

We consider three model problems depicted in Figure 8.7. First two are scalar problems consisting of two membranes in mutual contact at adjacent edges. The solution $u(x, y)$ can be interpreted as a vertical displacement of two membranes stretched by normalized horizontal forces and pressed together by vertical forces with density $f(x, y)$. The inequality constraints result from requiring nonpenetration of the adjacent edges of the membranes, with the edge of the right membrane above the edge of the left membrane and by pressing the left membrane down by the right one at the contact points. The first problem, where the right membrane has its right edge fixed, is coercive (Figures 8.7a and 8.7b). The second problem is semicoercive since the right membrane is completely floating (Figures 8.7c and 8.7d).

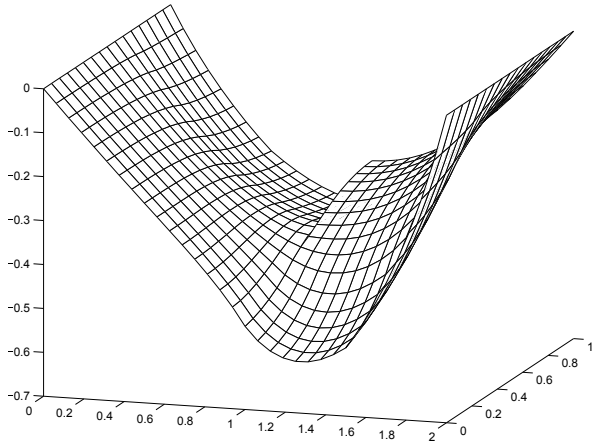
As a model 3D linear elasticity contact problem, we consider an elastic cube with the bottom face fixed, the top one loaded with a vertical surface force directed downwards, and the right one in contact with a rigid obstacle (Figures 8.7e and 8.7f). The loading force density is $f_z = 465$ N/mm^2, Young's modulus $E = 2 \cdot 10^5$ MPa, Poisson's ratio $\mu = 0.33$.



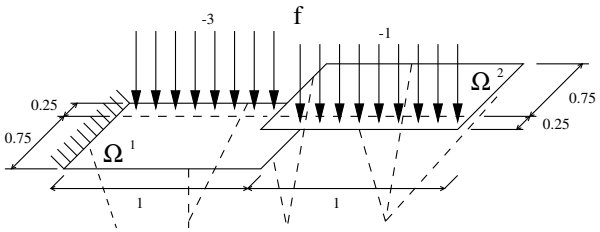
$\Gamma_u^1 \quad \Gamma_f^1 \quad \Gamma_c \quad \Gamma_f^2 \quad \Gamma_u^2$

$$f(x,y) = \begin{cases} -1 & \text{for} \quad (x,y) \in (0,1) \times [0.75,1) \\ 0 & \text{for} \quad (x,y) \in (0,1) \times [0,0.75) \\ & \text{and} \quad (x,y) \in (1,2) \times [0.25,1) \\ -3 & \text{for} \quad (x,y) \in (1,2) \times [0,0.25) \end{cases}$$

(a)



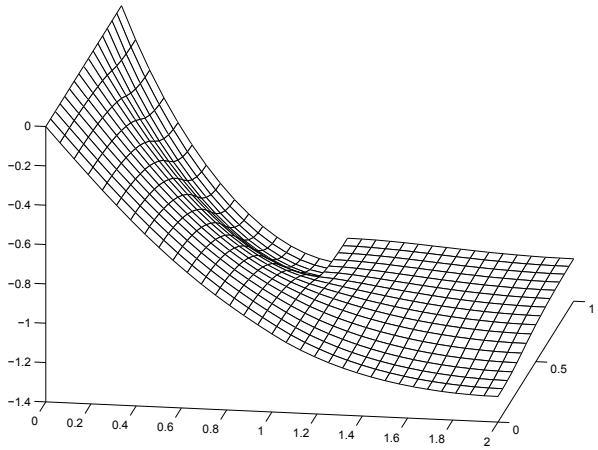
(b)



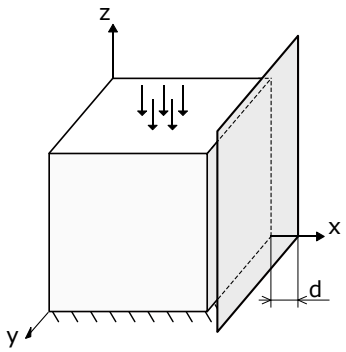
$\Gamma_u^1 \quad \Gamma_f^1 \quad \Gamma_c \quad \Gamma_f^2$

$$f(x,y) = \begin{cases} -3 & \text{for} \quad (x,y) \in (0,1) \times [0.75,1) \\ 0 & \text{for} \quad (x,y) \in (0,1) \times [0,0.75) \\ & \text{and} \quad (x,y) \in (1,2) \times [0.25,1) \\ -1 & \text{for} \quad (x,y) \in (1,2) \times [0,0.25) \end{cases}$$

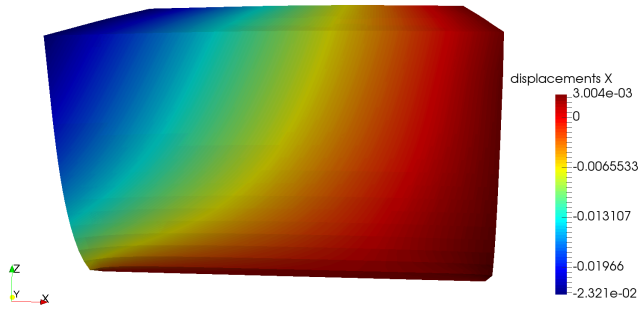
(c)



(d)



(e)



(f)

Figure 8.7: Model problems: coercive (a, b) and semicoercive (c, d) scalar contact problem of two membranes, and elastic cube with a rigid obstacle (e, f) – problem specification (left) and solution (resulting displacements, right).

8.3.1 Decomposition and discretization

Regarding the first two problems, the coercive and semicoercive membrane problems, each of two membranes was first partitioned into subdomains with the sidelengths $H \in \{1/8,\ 1/11,\ 1/16,\ 1/24,\ 1/32\}$. The square subdomains were then discretized by the regular grids with the discretization parameter $h = H/128$, so that the ratio H/h was kept constant. The third problem, the elastic cube, was decomposed into subdomains with sidelengths $H \in \{1/5,\ 1/6,\ 1/8,\ 1/10,\ 1/13\}$ and discretized with $h = 1/20$ and again with constant H/h. In all cases, the splitting was chosen in order to get the numbers of subdomains near the powers of two. The corresponding total numbers of subdomains well as the primal, dual and kernel dimensions can be found in Tables 8.5 and 8.6. Let us remind that the dual dimension is the total number of gluing, Dirichlet and non-penetration interface constraints, i.e. number of rows of the matrix $\mathbf{B}$.

$1/H$	N_S	n	m (coercive)	m (semicoercive)	d
8	128	2,130,048	32,160	31,142	128
11	242	4,027,122	61,377	59,978	242
16	512	8,520,192	130,872	128,838	512
24	1152	19,170,432	296,144	293,094	1152
32	2048	34,080,768	527,976	523,910	2048

Table 8.5: Dimension settings for coercive and semicoercive problem with $h = H/128$.

$1/H$	N_S	n	m	d
5	125	3,472,875	469,392	750
6	216	6,001,128	832,194	1296
8	512	14,224,896	2,035,950	3072
10	1000	27,783,000	4,051,602	6000
13	2197	61,039,251	9,055,080	13,182

Table 8.6: Dimension settings for cube in 3D with $h = H/24$.

8.3.2 Solver settings

Let us illustrate the numerical scalability of TFETI for contact problems (Section 4.4) combined with modified SMALBE (Sections 3.2, 3.3 and 5.4) and MPRGP (Section 3.1), and weak parallel scalability of their PERMON implementations on the three model problems described above.

We used the following parameters setting for the SMALBE and MPRGP algorithms:

$$M_0 = 100||\mathbf{PFP}||,\ \rho = 2||\mathbf{PFP}||,\ \eta = 0.1||\mathbf{Pd}|| \quad \text{and} \quad \Gamma = 1,$$

where the matrix norms were approximated using the power method. These values are default in PermonQP and they have been chosen based on many comparative numerical tests. Needless to say, the optimal values for particular problems may slightly differ.

An important feature for large problems could an adaptive selection of the expansion step length $\bar{\alpha}$ according to Algorithm 5.11 in [10]. This feature has been under development in PermonQP during preparation of this thesis. One can anticipate that the default values of parameters will have to be rethought once the feature is finished.

The stopping criterion was

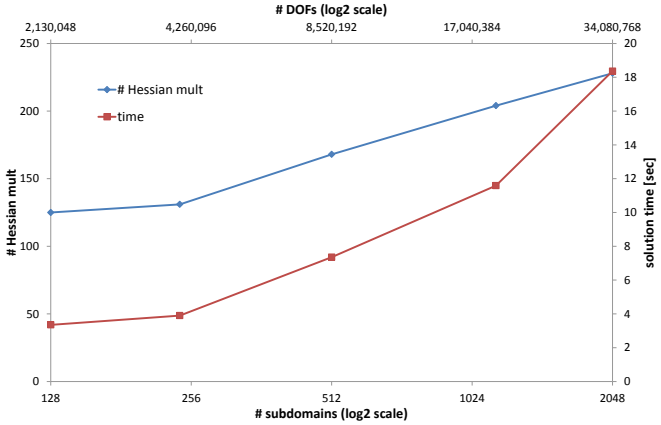
$$||\mathbf{g}^P|| \leq \epsilon||\mathbf{Pd}|| \ \wedge \ ||\mathbf{G}\boldsymbol{\lambda}|| \leq \epsilon||\mathbf{Pd}||, \qquad \epsilon = 10^{-4}.$$

The stiffness matrix pseudoinverse $\mathbf{K}^\dagger$ was implemented using the Cholesky factorization from the MUMPS library [46]. The approach from [9] was used where the original matrix $\mathbf{K}$ is regularized and the inverse of the regularized matrix is a pseudoinverse of $\mathbf{K}$.

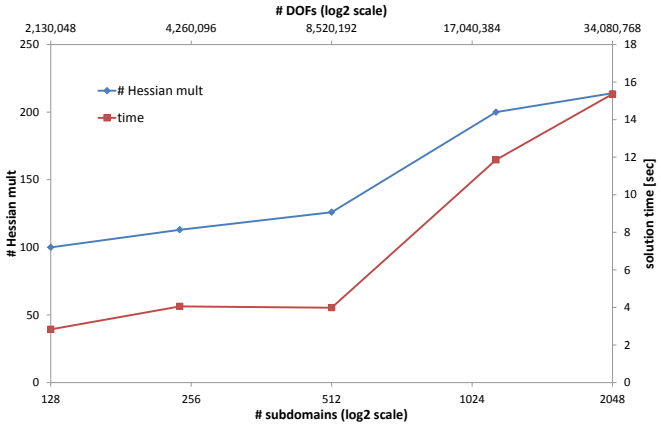
Thanks to implicit orthonormalization presented in Chapter 5, it is possible to use for contact problems the same CP as for linear ones. The CP (action of $(\mathbf{G}\mathbf{G}^T)^{-1}$) was solved by the LU factorization from the SuperLU_DIST library [71] in subcommunicators of size $N_S^{1/2}$ and $N_S^{2/3}$ for PermonMembrane and PermonCube, respectively. Each subcommunicator was solving the same CP redundantly using Strategy 1 (Section 7.5.3).

8.3.3 Performance results

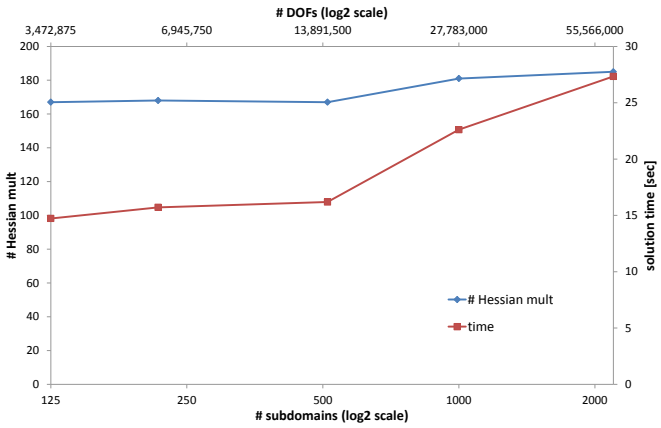
The benchmarks were run on ARCHER (Section 8.1.1). The performance results are shown in Figure 8.8. All the graphs illustrate the numerical and weak parallel scalability up to more than 2000 cores. The numerical scalability of the used TFETI + SMALBE + MPRGP approach has been theoretically proven in [10]. It says that keeping the ratio H/h constant, the number of Hessian multiplications is bound by a constant for any problem size. The numerical scalability graphs (with circle marks) reveal the PermonQP implementation fulfils this fairly well. Parallel scalability graphs (with box marks) show the total solution times, i.e. time spent in PermonFLLOP and PermonQP including necessary pre- and post-processing steps before and after the iterative phase. Each parallel scalability graph follows the shape of the respective numerical scalability graph up to ca. 1000 subdomains. Then some worse scalable parts of the implementation start to spoil the parallel scalability. They include e.g. the implementation of the $\mathbf{B}$ actions and the matrix-matrix product $\mathbf{G} * \mathbf{G}^T$. Improving these parts is work-in-progress.



(a)



(b)



(c)

Figure 8.8: Graphs of numerical and weak parallel scalability for the coercive (a) and semicoercive (b) membrane problems, and the cube problem (c) .

8.4 Real world problems

To show applicability of PERMON to real-world problems, two benchmarks have been run. Progress in solution of real world problems is currently limited by the FEM implementation. Within this thesis, only a file-based interface with the in-house MatSol library has been used. It is written in MATLAB which presents major computational and licensing limitations. In-memory interfaces with open source FEM libraries such as libMesh (Section 6.3.8), Elmer (Section 6.3.3) and SIFEL[1] are work-in-progress.

The first benchmark simulates a spanner touching a nut at a priori unknown points (Figure 8.9). Meshing was done in ANSYS. Quadratic tetrahedra elements with midnodes were used. The mesh was decomposed into 50 subdomains using METIS (Section 6.2.1). FEM assembly was performed by Tomáš Brzobohatý (IT4Innovations) using the in-house MatSol library written in MATLAB. It is the first real-world contact problem solved with PERMON. The same solver settings as in Section 8.3.2 were used. This benchmark was run on Salomon (Section 8.1.3). Within this benchmark, we have tested a special type of scalability – how the number of iterations relates to the relative tolerance ε. Timings are shown in Table 8.8.

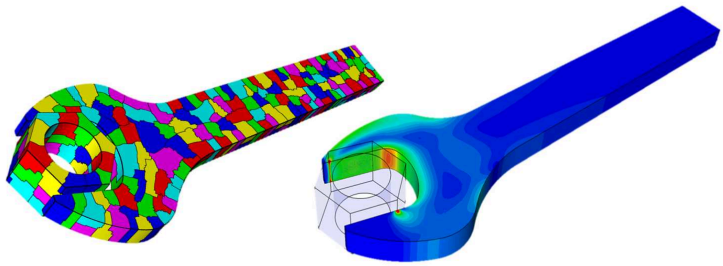


Figure 8.9: The spanner benchmark – decomposition (left) and solution (resulting displacements, right).

The second benchmark uses a geometry of the car engine block (Figure 8.10). Dirichlet boundary conditions were prescribed in one cylinder. Meshing was done using ANSYS on a symmetric multiprocessor system. Quadratic tetrahedra elements with midnodes were used. The mesh was decomposed into 5012 subdomains by METIS (Section 6.2.1). FEM assembly was performed by Tomáš Brzobohatý (IT4Innovations) using the in-house MatSol library written in MATLAB. This benchmark was run on HECToR (Section 8.1.2). Timings are shown in Table 8.8.

[1] an in-house FEM code developed by the group of Jaroslav Kruis at ČVÚT

ε	outer iter. count	inner iter. count	Hessian actions count	total sol. time
1E-04	10	469	543	6.64
1E-06	11	635	711	8.62
1E-08	13	855	929	11.11
1E-10	14	1039	1117	13.53
1E-12	16	1303	1380	16.46

Table 8.7: The spanner problem – tolerance vs. number of iterations. Times in seconds; $n = 177,402$; $m = 27,534$; $d = 300$; $N_S = 50$; $\mathbf{K}^\dagger$ using MUMPS; $(\mathbf{G}\mathbf{G}^T)^{-1}$ using SuperLU_DIST, Strategy 1, $N_r = 1$.

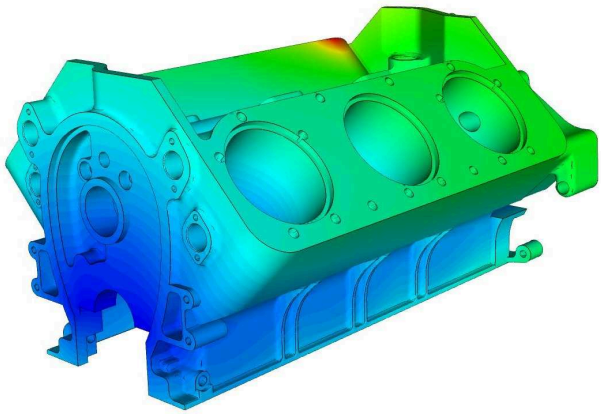


Figure 8.10: The car engine benchmark.

$\mathbf{K}^{reg}$ fact.	$\mathbf{G}\mathbf{G}^T$ fact.	total setup	iter. count	all $\mathbf{K}^\dagger$ actions	all $(\mathbf{G}\mathbf{G}^T)^{-1}$ actions	total iter. solve
3.89	18.0	28.2	181	15.0	74.8	233.0

Table 8.8: Performance of PermonFLLOP TFETI for the engine problem. Times in seconds; $n = 98,214,558$; $m = 13,395,882$; $d = 30,072$; $N_S = 5012$; $\mathbf{K}^\dagger$ using MUMPS; $(\mathbf{G}\mathbf{G}^T)^{-1}$ using SuperLU_DIST, Strategy 1, $N_r = 16$.

Conclusion

This thesis presents theoretical and practical aspects of the set of libraries called PERMON. Its crucial part, PERMON Solver Core, consists mainly of two packages: PermonQP and PermonFLLOP. PermonQP is a general-purpose package for large-scale QP, whereas PermonFLLOP brings DDM of FETI type with more narrow application area but with ability to solve much larger problems.

Almost no new purely theoretical results have been presented in this work. However, several observations, which may seem trivial, have a great impact on the implementation. For instance, the implicit orthonormalization of equality constraints (Chapter 5) has enabled us to solve problems decomposed into more than thousand subdomains which had not been possible with explicit orthonormalization (QR factorization, Gram-Schmidt process). I therefore hope the value of this work lies in the transition of theory into practice.

I have spent last almost five years of hard work with programming PERMON Solver Core. Starting with my nearest colleague David Horák, several young colleagues have joined this effort gradually, and I have become something like a coordinator of this software project. The implementation process has been a challenging effort. Several times, shift in functionality introduced regression in performance. One example is a support for multiple separate equality constraints. It is attractive for users and allows more general usage and use of special optimized matrix types for different equality constraints, but it had on its own slowed down the code substantially for larger problems because of performance issues of the underlying PETSc nested matrix implementation, and it took some time to cope with that.

PERMON brought me First Prize in Joseph Fourier Prize 2014 and several prizes at the university. A poster about PERMON was presented e.g. at the SC15 conference in Austin, Texas.

Ongoing and future work

In PermonFLLOP, substantial amount of work has been expended together with Alena Vašatová and Martin Čermák to optimize the assembly and action of the constraint matrix $\mathbf{B}$ within FETI. This topic is partly covered in a new paper [107] which is about to come out this year. Moreover, we are preparing a more theoretical paper regarding the spectral properties of $\mathbf{B}$, their optimization and their impact on convergence. Together with Radim Sojka and Jakub Kružík, we further optimize coarse problem solution and implement support for multiple subdomains per process. With David Horák, we currently develop a method of hiding the coarse problem communication. In future, PermonFLLOP will solve more complex problems including plasticity and shape optimization, and we will also study applications outside structure mechanics such as Helmholtz problems.

Concerning PermonQP, much effort has been put into implementation of separable convex constraints which will allow solution of contact problems with friction, within joint effort with Lukáš Pospíšil. This feature waits for merging with the main branch and testing. Other interesting work-in-progress in QP includes preconditioning of QP algorithms and new QP applications. In future, we would like to implement in PERMON also alternative QP algorithms such as interior point methods and use PERMON within the sequential quadratic programming (SQP) method for nonlinear optimization.

The PERMON software should be published this year as open source. But it is still only beginning. We will strive to offer it to users who need to solve large scale QP problems and have no appropriate tool for that or who are left to expensive closed source solvers with limited parallelism. PERMON will be presented at the First European PETSc User Meeting in Vienna, June 2016.

Bibliography

[1] *Agros2D*. URL: http://www.agros2d.org/ (visited on 10/29/2015).

[2] 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*. Ed. by E. Arge, A. M. Bruaset, and H. P. Langtangen. Birkhäuser Press, 1997, pp. 163–202. DOI: 10.1007/978-1-4612-1986-6_8.

[3] S. Balay et al. *PETSc – Portable, Extensible Toolkit for Scientific Computation*. URL: http://www.mcs.anl.gov/petsc (visited on 03/23/2016).

[4] W. Bangerth. "Using Modern Features of C++ for Adaptive Finite Element Methods: Dimension-Independent Programming in deal.II". In: *Proceedings of the 16th IMACS World Congress 2000, Lausanne, Switzerland, 2000*. Ed. by M. Deville and R. Owens. Document Sessions/118-1. 2000. URL: http://tinyurl.com/hpu3zxg (visited on 03/23/2016).

[5] T. Brzobohatý et al. "Cholesky decomposition with fixing nodes to stable computation of a generalized inverse of the stiffness matrix of a floating structure". In: *International Journal for Numerical Methods in Engineering* 88.5 (2011), pp. 493–509. DOI: 10.1002/nme.3187.

[6] *CGAL – The Computational Geometry Algorithms Library*. URL: http://doc.cgal.org/ (visited on 02/01/2016).

[7] *CVXOPT*. URL: http://cvxopt.org/ (visited on 02/01/2016).

[8] Z. Dostál, D. Horák, and R. Kučera. "Total FETI – an easier implementable variant of the FETI method for numerical solution of elliptic PDE". In: *Communications in Numerical Methods in Engineering* 22.12 (2006), pp. 1155–1162. DOI: 10.1002/cnm.881.

[9] Z. Dostál, T. Kozubek, A. Markopoulos, and M. Menšík. "Cholesky decomposition of a positive semidefinite matrix with known kernel". In: *Applied Mathematics and Computation* 217.13 (2011), pp. 6067–6077. DOI: 10.1016/j.amc.2010.12.069.

[10] Z. Dostál. *Optimal Quadratic Programming Algorithms, with Applications to Variational Inequalities*. Vol. 23. Springer, New York, US, 2009.

[11] Z. Dostál, A. Friedlander, and S. A. Santos. "Augmented Lagrangians with Adaptive Precision Control for Quadratic Programming with Simple Bounds and Equality constraints". In: *SIAM Journal on Optimization* 13 (2003), pp. 1120–1140. DOI: 10.1137/S1052623499362573.

[12] Z. Dostál and D. Horák. "Scalable FETI with optimal dual penalty for a variational inequality". In: *Numerical Linear Algebra with Applications* 11 (2004), pp. 455–472. DOI: 10.1002/nla.355.

[13] Z. Dostál and D. Horák. "Theoretically Supported Scalable FETI for Numerical Solution of Variational Inequalities". In: *SIAM Journal on Numerical Analysis* 45.2 (2007), pp. 500–513. DOI: 10.1137/050639454.

[14] Z. Dostál and T. Kozubek. "An optimal algorithm and superrelaxation for minimization of a quadratic function subject to separable convex constraints with applications". In: *Mathematical programming, Ser. A* 135 (2012), pp. 195–220. DOI: 10.1007/s10107-011-0454-2.

[15] Z. Dostál and J. Schöberl. "Minimizing quadratic functions subject to bound constraints". In: *Computational Optimization and Applications* 30.1 (Jan. 2005), pp. 23–43. DOI: 10.1023/B:COAP.0000049888.80264.25.

[16] Z. Dostál et al. "FETI based algorithms for contact problems: scalability, large displacements and 3D Coulomb friction". In: *Computer Methods in Applied Mechanics and Engineering* 194.2–5 (2005), pp. 395–409. DOI: 10.1016/j.cma.2004.05.015.

[17] Z. Dostál et al. "Scalable TFETI algorithm for the solution of multibody contact problems of elasticity". In: *International Journal for Numerical Methods in Engineering* 82.11 (2010), pp. 1384–1405. ISSN: 1097-0207. DOI: 10.1002/nme.2807.

[18] *DUNE*. URL: http://www.dune-project.org/ (visited on 10/29/2015).

[19] *Elemental: distributed-memory dense and sparse-direct linear algebra and optimization*. URL: http://libelemental.org/ (visited on 02/01/2016).

[20] C. Farhat, J. Mandel, and F.-X. Roux. "Optimal convergence properties of the FETI domain decomposition method". In: *Computer Methods in Applied Mechanics and Engineering* 115 (1994), pp. 365–385. DOI: 10.1016/0045-7825(94)90068-X.

[21] C. Farhat and F.-X. Roux. "An Unconventional Domain Decomposition Method for an Efficient Parallel Solution of Large-Scale Finite Element Systems". In: *SIAM Journal on Scientific and Statistical Computing*. 13th ser. 1 (1992). DOI: 10.1137/0913020.

[22] *Feel++*. URL: http://www.feelpp.org/ (visited on 10/29/2015).

[23] *FreeFem++*. URL: http://www.freefem.org/ff++/ (visited on 10/29/2015).

[24] A. Friedlander, J. M. Martínez, and M. Raydan. "New method for large-scale box constrained convex quadratic minimization problems". In: *Optimization Methods and Software* 5.1 (1995), pp. 57–74.

[25] *GALAHAD*. URL: http://www.galahad.rl.ac.uk/ (visited on 02/01/2016).

[26] P. Gosselet and C. Rey. "Non-overlapping domain decomposition methods in structural mechanics". English. In: *Archives of Computational Methods in Engineering* 13.4 (2006), pp. 515–572. ISSN: 1134-3060. DOI: 10.1007/BF02905857.

[27] *Gurobi: a solver for sparse nonlinear optimization*. URL: http://www.gurobi.com/ (visited on 02/01/2016).

[28] F. Hecht. "New development in FreeFem++". In: *J. Numer. Math.* 20.3-4 (2012), pp. 251–265. ISSN: 1570-2820.

[29] *HECToR*. URL: http://www.hector.ac.uk/ (visited on 10/29/2015).

[30] D. M. Hensinger, R. R. Drake, J. G. Foucar, and T. A. Gardiner. *Pamgen, a library for parallel generation of simple finite element meshes.* Tech. rep. SAND2008-1933. Sandia National Laboratories, 2008. DOI: 10.2172/932881. URL: http://www.osti.gov/scitech/biblio/932881 (visited on 03/23/2016).

[31] *Hermes*. URL: http://www.hpfem.org/hermes/ (visited on 10/29/2015).

[32] V. Hernandez, J. E. Roman, and V. Vidal. "SLEPc: A Scalable and Flexible Toolkit for the Solution of Eigenvalue Problems". In: *ACM Trans. Math. Softw.* 31.3 (Sept. 2005), pp. 351–362. ISSN: 0098-3500. DOI: 10.1145/1089014.1089019.

[33] *HQP: a solver for sparse nonlinear optimization*. URL: http://hqp.sourceforge.net/ (visited on 02/01/2016).

[34] P. Jolivet et al. *HPDDM – high-performance unified framework for domain decomposition methods*. URL: https://github.com/hpddm/hpddm (visited on 10/29/2015).

[35] P. Jolivet. "PhD Thesis". PhD thesis. Université de Grenoble, 2014. URL: http://jolivet.perso.enseeiht.fr/thesis.pdf (visited on 03/23/2016).

[36] P. Jolivet, F. Hecht, F. Nataf, and C. Prud'homme. "Scalable Domain Decomposition Preconditioners for Heterogeneous Elliptic Problems". In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. SC '13. New York, NY, USA: ACM, 2013, 80:1–80:11. ISBN: 978-1-4503-2378-9. DOI: 10.1145/2503210.2503212.

[37] B. S. Kirk et al. *libMesh*. URL: http://libmesh.sourceforge.net (visited on 10/29/2015).

[38] B. S. Kirk, J. W. Peterson, R. H. Stogner, and G. F. Carey. "libMesh: a C++ library for parallel adaptive mesh refinement/coarsening simulations". In: *Engineering with Computers* 22.3-4 (2006), pp. 237–254. ISSN: 0177-0667. DOI: 10.1007/s00366-006-0049-3.

[39] R. Kučera, T. Kozubek, and A. Markopoulos. "On large-scale generalized inverses in solving two-by-two block linear systems". In: *Linear Algebra and Its Applications* 438.7 (2013), pp. 3011–3029. DOI: 10.1016/j.laa.2012.09.027.

[40] J. W. H. Liu. "The Multifrontal Method for Sparse Matrix Solution: Theory and Practice". In: *SIAM Review* 34.1 (1992), pp. 82–109. ISSN: 0036-1445. URL: http://www.jstor.org/stable/2132786.

[41] M. Mashayekhi. "Finite Element Method online course, Lesson 16". 2013. URL: http://mashayekhi.iut.ac.ir/sites/mashayekhi.iut.ac.ir/files//files_course/lesson_16.pdf.

[42] *Mesh Generation & Grid Generation on the Web: Software*. URL: http://tinyurl.com/mesh-sw-list (visited on 03/23/2016).

[43] *METIS*. URL: http://tinyurl.com/libmetis (visited on 03/23/2016).

[44] *MOOSE Framework*. URL: http://mooseframework.org/ (visited on 10/29/2015).

[45] *MOSEK ApS*. URL: https://www.mosek.com/ (visited on 02/01/2016).

[46] *MUMPS*. URL: http://graal.ens-lyon.fr/MUMPS/ (visited on 10/29/2015).

[47] T. Munson et al. *TAO Users Manual*. Tech. rep. ANL/MCS-TM-322. Argonne National Laboratory, 2015. URL: http://tinyurl.com/tao-man (visited on 03/23/2016).

[48] *Netgen Mesh Generator*. URL: http://sourceforge.net/projects/netgen-mesher/ (visited on 10/29/2015).

[49] *NGSolve*. URL: http://sourceforge.net/projects/ngsolve/ (visited on 10/29/2015).

[50] *OOFEM*. URL: http://www.oofem.org/en/ (visited on 10/29/2015).

[51] *OOQP - object oriented software for quadratic programming*. URL: http://pages.cs.wisc.edu/~swright/ooqp/ (visited on 02/01/2016).

[52] *PARALUTION*. URL: http://www.paralution.com/ (visited on 10/29/2015).

[53] *PARDISO*. URL: http://www.pardiso-project.org/ (visited on 10/29/2015).

[54] *ParMETIS*. URL: http://tinyurl.com/parmetis (visited on 03/23/2016).

[55] *PaStiX*. URL: http://pastix.gforge.inria.fr/ (visited on 10/29/2015).

[56] *PENOPT*. URL: http://www.penopt.com/ (visited on 02/01/2016).

[57] *PETSc PCBDDC manual page*. URL: http://tinyurl.com/pcbddc (visited on 03/23/2016).

[58] *qpOASES*. URL: https://projects.coin-or.org/qpOASES (visited on 02/01/2016).

[59] *QuadProg++*. URL: http://quadprog.sourceforge.net/ (visited on 02/01/2016).

[60] P. Råback et al. *Elmer*. URL: https://www.csc.fi/web/elmer (visited on 03/21/2016).

[61] P. Råback and M. Malinen. *Overview of Elmer*. Tech. rep. 2016. URL: ftp://ftp.funet.fi/index/elmer/doc/ElmerOverview.pdf (visited on 03/21/2016).

[62] *R-Elemental: Wrappers to connect R to Elemental linear algebra library*. URL: https://github.com/rocanale/R-Elemental (visited on 02/01/2016).

[63] D. Rixen. "Substructuring and dual methods in structural analysis". PhD thesis. University of Liège, Belgium, 1997.

[64] *SCOTCH*. URL: http://www.labri.fr/perso/pelegrin/scotch/ (visited on 02/11/2016).

[65] J. Šístek et al. *The Multilevel BDDC solver library (BDDCML)*. URL: http://tinyurl.com/jgxnwyz (visited on 03/23/2016).

[66] *SLEPc*. URL: http://www.grycap.upv.es/slepc/ (visited on 10/29/2015).

[67] B. F. Smith et al. *PETSc Users Manual*. Tech. rep. ANL-95/11 - Revision 3.5. Argonne National Laboratory, 2014. URL: http://tinyurl.com/petsc-man (visited on 03/23/2016).

[68] B. F. Smith et al. *PETSc Developers Manual*. Tech. rep. Argonne National Laboratory, 2015. URL: http://tinyurl.com/petsc-dev-man (visited on 03/23/2016).

[69] B. Sousedík, J. Šístek, and J. Mandel. "Adaptive-Multilevel BDDC and its parallel implementation". In: *Computing* 95.12 (2013), pp. 1087–1119. ISSN: 1436-5057. DOI: 10.1007/s00607-013-0293-5.

[70] *SuiteSparse: a suite of sparse matrix algorithms*. URL: http://faculty.cse.tamu.edu/davis/suitesparse.html (visited on 10/29/2015).

[71] *SuperLU*. URL: http://acts.nersc.gov/superlu/ (visited on 10/29/2015).

[72] *TetGen*. URL: http://wias-berlin.de/software/tetgen/ (visited on 10/29/2015).

[73] *The deal.II Finite Element Library*. URL: http://dealii.org/ (visited on 10/29/2015).

[74] *The FEniCS Project*. URL: http://fenicsproject.org/ (visited on 10/29/2015).

[75] *The Trilinos Project*. URL: http://trilinos.org/ (visited on 10/29/2015).

[76] *The Trilinos Project: Pamgen*. URL: http://trilinos.org/packages/pamgen/ (visited on 10/29/2015).

[77] *TOP500 List - November 2015*. URL: http://www.top500.org/list/2015/11/ (visited on 03/14/2016).

[78] *Triangle*. URL: http://www.cs.cmu.edu/~quake/triangle.html (visited on 10/29/2015).

[79] *ViennaCL*. URL: http://viennacl.sourceforge.net/ (visited on 10/29/2015).

[80] *ViennaGrid*. URL: http://viennagrid.sourceforge.net/ (visited on 10/29/2015).

[81] *ViennaMesh*. URL: http://viennamesh.sourceforge.net/ (visited on 10/29/2015).

[82] O. Vlach, Z. Dostál, and T. Kozubek. "On Conditioning of Constraints Arising from Variationally Consistent Discretization of Contact Problems and Duality Based Solvers". In: *Comput. Meth. in Appl. Math.* 15.2 (2015), pp. 221–231. DOI: 10.1515/cmam-2014-0031.

[83] *Wikipedia: List of finite element software packages*. URL: http://tinyurl.com/wiki-num-lib (visited on 03/23/2016).

[84] Y. Yilmaz et al. *Parallel Mesh Generation, Migration and Partitioning for the Elmer Application*. Tech. rep. 2012. URL: http://www.prace-ri.eu/meshing/ (visited on 03/23/2016).

Author's publications, indexed

[85] M. Čermák, V. Hapla, A. Markopoulos, and T. Karásek. "Solving elastoplastic problems with different preconditioners". In: *AIP Conference Proceedings*. Vol. 1648. 2015. ISBN: 978-073541287-3. DOI: 10.1063/1.4913031.

[86] M. Čermák et al. "Total-FETI domain decomposition method for solution of elasto-plastic problems". In: *Advances in Engineering Software* 84 (2015), pp. 48–54. ISSN: 0965-9978. DOI: 10.1016/j.advengsoft.2014.12.011.

[87] V. Hapla, M. Čermák, A. Markopoulos, and D. Horák. "FLLOP: A Massively Parallel Solver Combining FETI Domain Decomposition Method and Quadratic Programming". In: *2014 IEEE Intl Conf on High Performance Computing and Communications (HPCC 2014)*. 2014, pp. 320–327. ISBN: 978-147996123-8. DOI: 10.1109/HPCC.2014.56.

[88] V. Hapla, D. Horák, and M. Merta. "Use of Direct Solvers in TFETI Massively Parallel Implementation". In: *Applied Parallel and Scientific Computing. 11th International Conference, PARA 2012, Helsinki, Finland, June 10-13, 2012, Revised Selected Papers*. Ed. by P. Manninen and P. Öster. Vol. 7782. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 192–205. ISBN: 978-364236802-8. DOI: 10.1007/978-3-642-36803-5_14.

[89] V. Hapla and D. Horák. "TFETI Coarse Space Projectors Parallelization Strategies". In: *Parallel Processing and Applied Mathematics. 9th International Conference, PPAM 2011, Torun, Poland, September 11-14, 2011. Revised Selected Papers*. Ed. by R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Waśniewski. Vol. 7203.Part I. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 152–162. ISBN: 978-3-642-31463-6. DOI: 10.1007/978-3-642-31464-3_16.

[90] D. Horák and V. Hapla. "TFETI coarse problem massively parallel implementation". In: *ECCOMAS 2012 - European Congress on Computational Methods in Applied Sciences and Engineering, e-Book Full Papers*. 2012, pp. 8260–8267. ISBN: 978-395035370-9.

[91] T. Kozubek et al. "Total FETI domain decomposition method and its massively parallel implementation". In: *Advances in Engineering Software* 60-61 (2013), pp. 14–22. ISSN: 0965-9978. DOI: 10.1016/j.advengsoft.2013.04.001.

[92] A. Markopoulos, M. Čermák, V. Hapla, and R. Halama. "Implementation of the plasticity solver in the PermonCube software package". In: *AIP Conference Proceedings*. Vol. 1648. 2015. ISBN: 978-073541287-3. DOI: 10.1063/1.4913035.

[93] A. Markopoulos, V. Hapla, M. Čermák, and M. Fusek. "Massively parallel solution of elastoplasticity problems with tens of millions of unknowns using PermonCube and FLLOP packages". In: *Applied Mathematics and Computation* (2015), pp. 698–710. ISSN: 0096-3003. DOI: 10.1016/j.amc.2014.12.097.

[94] M. Merta, A. Vašatová, V. Hapla, and D. Horák. "Parallel Implementation of Total-FETI DDM with Application to Medical Image Registration". In: *Domain Decomposition Methods in Science and Engineering XXI*. Ed. by T. Sassi et al. Vol. 98. Lecture Notes in Computational Science and Engineering. Springer Verlag, 2014, pp. 917–925. ISBN: 978-3-319-05788-0. DOI: 10.1007/978-3-319-05789-7_89.

Author's publications, unindexed

[95] V. Hapla et al. *PermonFLLOP*. URL: http://industry.it4i.cz/en/products/permon/fllop/ (visited on 10/29/2015).

[96] V. Hapla et al. *PermonQP*. URL: http://industry.it4i.cz/en/products/permon/qp/ (visited on 10/29/2015).

[97] V. Hapla et al. "PERMON toolbox combining discretization, domain decomposition, and quadratic programming". In: *Seminar on Numerical Analysis*. 2015. ISBN: 978-80-86407-55-5. URL: http://tinyurl.com/sna-proc-15.

[98] V. Hapla and D. Horák. "A Comparison of FETI Natural Coarse Space Projector Implementation Strategies". In: *Proceedings of PARENG2013*. 2013. Chap. Paper 6. DOI: 10.4203/ccp.101.6.

[99] V. Hapla, D. Horák, A. Markopoulos, and L. Říha. "FLLOP: a novel massively parallel QP solver". In: *Seminar on Numerical Analysis*. 2014. ISBN: 978-80-87136-16-4. URL: http://tinyurl.com/sna-proc-14.

[100] V. Hapla, D. Horák, and M. Merta. "Software design of TFETI massively parallel implementation". In: *Seminar on Numerical Analysis*. 2012. ISBN: 978-80-7372-821-2. URL: http://tinyurl.com/sna-proc-12.

[101] V. Hapla, D. Horák, and F. Staněk. "FLLOP: a massively parallel QP solver". In: *Seminar on Numerical Analysis*. 2013. ISBN: 978-80-86407-34-0. URL: http://tinyurl.com/sna-proc-13.

[102] V. Hapla and A. Markopoulos. "Implicit dual equality constraint matrix orhonormalization for TFETI+SMALBE approach to solution of variational inequalities with equality constraints". In preparation.

[103] V. Hapla et al. "Solving contact mechanics problems with PERMON". In: *High Performance Computing in Science and Engineering. HPCSE 2015, Soláň, Czech Republic, May 25-28, 2015. Revised Selected Papers*. Ed. by T. Kozubek et al. Lecture Notes in Computer Science. 2016. Accepted.

[104] T. Kozubek, D. Horák, and V. Hapla. *FETI Coarse Problem Parallelization Strategies and Their Comparison*. Tech. rep. 2012. URL: http://www.prace-project.eu/IMG/pdf/feticoarseproblemparallelization.pdf.

[105] A. Markopoulos, V. Hapla, et al. *PermonCube*. URL: http://industry.it4i.cz/en/products/permon/cube/ (visited on 10/29/2015).

[106] M. Merta et al. "Numerical libraries solving large-scale problems developed at IT4Innovations Research Programme Supercomputing for Industry". In: *Perspectives in Science* 7 (2016). 1st Czech-China Scientific Conference 2015, open access, pp. 140–150. ISSN: 2213-0209. DOI: 10.1016/j.pisc.2015.11.023. URL: http://tinyurl.com/hmyqu67 (visited on 03/23/2016).

[107] A. Vašatová, M. Čermák, and V. Hapla. "Parallel implementation of the FETI DDM constraint matrix on top of PETSc for the PermonFLLOP package". In: *Parallel Processing and Applied Mathematics. 11th International Conference, PPAM 2015, Krakow, Poland, September 6-9, 2015. Revised Selected Papers*. Ed. by R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Waśniewski. Lecture Notes in Computer Science. 2016. Accepted.