

ALGORITHMS AND SOFTWARE FOR STRUCTURED SEMIDEFINITE OPTIMIZATION

By

SOODEH HABIBI

A thesis submitted to
the University of Birmingham
for the degree of
PHD OF APPLIED MATHEMATICS

EC-ITN-POEMA Project
School of Mathematics
College of Engineering and Physical Sciences
University of Birmingham
January 2024

UNIVERSITY OF
BIRMINGHAM

University of Birmingham Research Archive

e-theses repository

This unpublished thesis/dissertation is copyright of the author and/or third parties. The intellectual property rights of the author or third parties in respect of this work are as defined by The Copyright Designs and Patents Act 1988 or as modified by any successor legislation.

Any use made of information contained in this thesis/dissertation must be in accordance with that legislation and must be properly acknowledged. Further distribution or reproduction in any format is prohibited without the permission of the copyright holder.

ABSTRACT

The goal of this thesis is to find solutions to large-and-sparse linear Semidefinite Programs (SDPs) with low-rank solutions and/or low-rank data, and we introduce a new code for the solution of these problems. We are investigating an Interior-Point (IP) approach to solve these problems. A general bottleneck of IP methods is assembling the so-called Schur complement equation and finding a solution for it, in which the matrix becomes increasingly ill-conditioned as the interior-point method makes progress toward the solution. To tackle this challenge, instead of the direct solver (Cholesky factorization), we suggest using a Preconditioned Conjugate Gradient (PCG) method within an interior-point semidefinite program algorithm and introduces a novel and efficient preconditioner that fully utilizes the low-rank information. Efficient preconditioners allow PCG to converge to an approximate solution of the linear equation in a few iterations, independent of the ill-conditioning of the Schur complement matrix. The efficiency is demonstrated by numerical experiments using the truss topology optimization problems of growing dimension which is formulated as large-scale SDP with the low-rank solution and the sensor network localization problems. The numerical results demonstrate that our preconditioner is working better than preconditioners recently proposed by Zhang and Lavaei [86], based on a similar idea. Also, our Matlab implementation outperforms MOSEK [57] and other available IP-based software. Furthermore, we use this IP approach to solve linear SDPs arising from higher-order Lasserre relaxations of Unconstrained Binary Quadratic optimization Problems (UBQPs). For these problems, the preconditioner makes use of the low-rank structure of the solution of the relaxations. Thus, we re-write the moment relaxations and use an ℓ_1-penalty approach within the IP solver to deal with the arising linear equality constraints.

The efficiency of this approach is demonstrated by numerical experiments with the MAXCUT and other randomly generated problems and a comparison with a state-of-the-art semidefinite solver and the ADMM method. As a by-product, we observe that the second-order relaxation is often high enough to deliver a globally optimal solution of the original problem.
In addition to this study, we look at how Polynomial Optimization techniques can be applied to Geometric Modeling problems. We discuss some examples of geometric modeling problems, and how they can be solved utilizing polynomial optimization tools. At the end, we report and analyze some experimental findings.

Keywords— Semidefinite optimization, interior-point methods, preconditioned conjugate gradients, Binary quadratic optimization, truss topology optimization, MAXCUT problem, Lasserre relaxations, sensor network localization

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to the following individuals and organizations for their invaluable contributions and support throughout the completion of this thesis:

First and foremost, I am deeply grateful to my supervisor, Professor Michal Kočvara, for his guidance, expertise, and continuous support. I am very thankful to you for giving me this opportunity and I feel very lucky to have a supervisor with such kindness, patience, and understanding.

I would also like to extend my appreciation to my secondment advisors, Professor Michael Stingl and Professor Bernard Mourrain, for their valuable input and collaboration. Thank you for sharing your knowledge and experience with me. I learned a lot from you and your expertise significantly enhanced the quality of this study.

I am endlessly grateful to my amazing family for their unwavering support, understanding, and encouragement throughout this journey. It has been the most difficult thing in my life to be away from you but your love and belief in me have been a constant source of motivation.

Many thanks to my great friends, especially my office mates in room 318 and my housemates in Wesley International House, who made adjusting to and living in a new country significantly easier for me. You all made me feel loved and are like a family away from home. There are too many names to mention but you know who you are and I feel extremely fortunate to have you all.

The research presented in this thesis was part of the POEMA project which is funded by the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie Actions funding scheme. POEMA supported this thesis in many ways and gave me great

opportunities by providing resources and a professional environment to learn and develop my research skills.

Finally, I would like to thank the professors and faculty members at the University of Birmingham who provided an excellent academic environment for research.

Contents

Page

List of Figures **i**

List of Tables **iii**

Notation **v**

Nomenclature **viii**

1 Introduction **1**

I Theory **7**

2 Background **8**

2.1 Semidefinite Programming 8

2.1.1 SDP and duality 10

2.2 Interior-Point Algorithm 15

2.2.1 Basic framework 16

2.2.2 Newton direction 17

2.2.3 Interior-point algorithm 18

3 Polynomial and Semi-algebraic Optimization **21**

3.1 Polynomial Optimization 21

3.2 Convex Relaxations 22
3.2.1 Sum of Squares Approach 23
3.2.2 Moment Approach 29
3.2.3 Computing the minimizers 41

4 Theory of Loraine **44**
4.1 Low-rank semidefinite optimization 45
4.1.1 Basic assumptions 45
4.1.2 Low-rank assumption 46
4.2 Preconditioners 47
4.2.1 Rank of H 49
4.2.2 H_α preconditioner 51
4.2.3 H_β and hybrid preconditioner 57
4.3 Solving SDPs with linear equality constraints 59
4.3.1 Schur Complement Method 60
4.3.2 l_1-Penalty Approach 62
4.4 The code Loraine 63
4.4.1 Special features 63
4.4.2 Data input 64
4.4.3 Loraine options 65
4.4.4 Loraine setting 66

II Applications and Numerical Results **68**

5 Truss topology optimization **69**
5.1 Truss notation 69
5.2 Truss topology problem, rank of the solution 71
5.3 Truss topology problem, vibration constraints 75

5.4 Numerical experiments 77
5.4.1 Problem database 77
5.4.2 Numerical results 79
5.4.3 Other software 80
5.4.4 No preconditioner 84

6 Sensor network localization with noisy data **86**
6.1 The problem 86
6.2 SDP relaxation 87
6.3 Numerical experiments 90
6.3.1 Problem database 90
6.3.2 Numerical results 90

7 Unconstrained binary quadratic optimization problems **94**
7.1 UBQP notation 94
7.2 Forms of SDP relaxations 97
7.2.1 Lasserre hierarchy of moment problems 97
7.2.2 Re-writing the problem 100
7.2.3 Equality constraints by ℓ_1-penalty 101
7.2.4 Loraine and problem (7.9) 104
7.3 ADMM for SDP relaxations 107
7.3.1 The ADMM-SDP algorithm 107
7.3.2 ADMM as a warm start for Loraine 109
7.4 Numerical experiments 110
7.4.1 MAXCUT problems 111
7.4.2 Randomly generated problems 116
7.4.3 BQPGKA problems 120

8 Geometric modeling problems **122**

8.1 Geometric Modeling and Polynomials 122

8.2 Polynomial Optimization Problems and Convex Relaxations 123

8.2.1 Sum of Squares Relaxations 124

8.2.2 Moment Relaxations 126

8.2.3 Computing the minimizers 127

8.3 Minimal enclosing ellipsoids of semi-algebraic sets 129

8.4 Parameterized surfaces 136

8.4.1 Closest point and surface-surface intersection 137

8.4.2 Bounding box and enclosing ellipsoid of parametric surfaces 138

8.5 Robots and mechanisms 139

8.5.1 Direct kinematic problem 141

8.5.2 Bounding box of robot workspace 144

8.5.3 Enclosing ellipsoid of robot workspace 146

8.5.4 Trajectories of a parallel robot 147

9 Conclusions **150**

Bibliography **153**

List of Figures

4.1 The distribution of eigenvalues of H in logarithmic scaling (on the left) and $H_\alpha^{-1}H$ is linear scaling (on the right) in tru7. 57

4.2 Problem tru7e; eigenvalues of $D^\top X_{\text{lin}} S_{\text{lin}}^{-1} D$ (red) and $\sum_{i=1}^{p} \mathbf{A}_i^\top (W_i^0 \otimes W_i^0)\mathbf{A}_i + \tilde{V}\tilde{V}^\top$ (blue) in iteration 2, 17 and 30 (left to right). 58

5.1 A problem with five nodes vanishing at the optimum; initial structure (left) and optimal solution (right). 74

5.2 Optimal results for problems tru25 (left) and vib19 (right). 78

5.3 Left: The history of the condition numbers of the preconditioned matrix in problem tru9. Right: Eigenvalues of the preconditioned matrix in the last iteration of Loraine. 79

5.4 Loraine in tru$*$ (blue) and tru$*$e (red) problems. CPU time vs number of variables in log-log scale. 81

5.5 Loraine in vib$*$ (blue) and vib$*$e (red) problems. CPU time vs number of variables in log-log scale. 81

5.6 Loraine (red) and MOSEK (blue) in tru$*$ problems. CPU time per iteration vs number of variables in log-log scale. 83

6.1 Sensor network localization problem: initial data (left) and optimal solution of the SDP relaxation (right). Black: anchors; magenta: exact positions of sensors; red (left): known distances between anchors and sensors; blue (left): known mutual distances of sensors; blue (right): computed positions of sensors. 88

6.2 Loraine (red) and MOSEK (blue) in SNL$*$ problems. CPU time per iteration vs number of variables in log-log scale. 92
6.3 Problem SNL-80 solved using the hybrid preconditioner. 93

7.1 The printout of low-precision ADMM and warm-started Loraine for the weighted MAXCUT-50. 114
7.2 CPU times in log-log scale for weighted (left) and unweighted (right) MAXCUT problems. Loraine (blue), MOSEK (brown), ADMM (green) and hybrid ADMM-Loraine (red). 115
7.3 CPU times in log-log scale for random BQP problems. Loraine (blue), MOSEK (brown) and ADMM (green). 119

8.1 A teapot model composed on 20 patches of bicubic polynomial parametrisations. . 137
8.2 Closest points and intersection point for bicubic parametric surfaces. 138
8.3 Closest points, intersection point and minimal enclosing ellipsoid for bicubic parametric surfaces. 139
8.4 Parallel robots with a fixed platform (red platform), moving platform (blue platform) and extensible arms (yellow arms). 140
8.5 The new position of the platform is shown in violet such that the middle of the blue platform is at a given point in the space (the green point). 144
8.6 On the left, we show a comparison of the minimal enclosing ellipsoid and bounding box. The initial position and extreme position of the platform minimizing y are presented on the right. 147
8.7 On the left, the trajectory of the center of the platform with 5 arm lengths fixed is shown. The trajectory of the center of the platform with 4 arm lengths fixed is shown on the right. 149

List of Tables

5.1 Eigenvalues of X^* and S^* for problem in Fig. 5.1. 75

5.2 Problems tru<n> and tru<n>e and problems vib<n> and vib<n>e, number of variables n, size of the LMI constraint m and number of linear constraints. 78

5.3 Loraine in `tru<n>` and `vib<n>` problems 80

5.4 Loraine direct and other solvers in `tru<n>` problems 82

5.5 Other solvers 83

5.6 Loraine with no preconditioner 85

6.1 Problems `SNL-<n>`, number of variables n, size of the LMI constraint m and number of linear constraints. 90

6.2 Loraine, MOSEK and ADMM in `SNL-<n>` problems 91

7.1 Problems `MAXCUT-<n>`, relaxation order $\omega = 2$: dimensions for formulations (7.4) and (7.9) 112

7.2 Loraine, MOSEK, ADMM and ADMM-Loraine in weighted `MAXCUT-<n>` problems, relaxation order $\omega = 2$ 113

7.3 Loraine, MOSEK, ADMM and ADMM-Loraine in unweighted `MAXCUT-<n>` problems, relaxation order $\omega = 2$ 115

7.4 Randomly generated UBQP problems, relaxation order $\omega = 2$: dimensions for formulations (7.4) and (7.9) 118

7.5 Randomly generated UBQP problems, relaxation order $\omega = 2$: Loraine, MOSEK and ADMM 118

7.6 Problem dimensions for formulations (7.4) and (7.9) as functions of the relaxation order; here s is the number of variables in BQP (7.1) and ω the relaxation order and $\frac{s^4}{24} \lessapprox n \leq 2^s - 1$ 119

7.7 Randomly generated UBQP problem, $n = 9$, relaxation order $\omega = 2, \ldots, 5$: problem dimensions and CPU times for Loraine-direct and Loraine-iterative 120

7.8 BQPGKA problems, relaxation order $\omega = 2$: Mosek and Loraine 121

Notation

$a \in \mathbb{R}$	Real scalars
$\mathbf{x} \in \mathbb{R}^n$	n-dimensional real vectors where $\mathbf{x} = (\mathrm{x}_1, \ldots, \mathrm{x}_n) \in \mathbb{R}^n$
$\mathrm{x}_I = (\mathrm{x}_i)_{i \in I} \in \mathbb{R}^{\lvert I \rvert}$	Set-indexing notation for vectors $\mathbf{x} \in \mathbb{R}^n$ and sets $I \subset \{1, \ldots, n\}$
$\mathbf{x} \geq \mathbf{y}$	$\mathrm{x}_i \geq \mathrm{y}_i$ for all i
$A \in \mathbb{R}^{n \times n}$	$n \times n$ real matrices
$A_{IJ} = [a_{ij}]_{i \in I, j \in J} \in \mathbb{R}^{\lvert I \rvert \times \lvert J \rvert}$	Set-indexing notation for matrices
$A \succ 0$	A is positive definite, i.e. $\mathbf{d}^\top A \mathbf{d} > 0$ for all $\mathbf{d} \in \mathbb{R}^n \backslash \{0\}$
$A \succeq 0$	A is positive semidefinite, i.e. $\mathbf{d}^\top A \mathbf{d} \geq 0$ for all $\mathbf{d} \in \mathbb{R}^n$
$A \in \mathbb{S}^n$	$n \times n$ symmetric matrices
$A \in \mathbb{S}^n_+$	$n \times n$ symmetric positive semidefinite matrices
$A \in \mathbb{S}^n_{++}$	$n \times n$ symmetric positive definite matrices
$A \bullet B$	$\bullet$ is the Frobenius inner product of two matrices A and B, i.e. $A \bullet B = \mathrm{tr}(A^\top B)$
$\mathrm{tr}(A)$	Trace of the matrix A, i.e. $\mathrm{tr}(A) = \sum_{i=1}^n a_{ii}$
$\langle A, B \rangle$	The inner product of matrices A and B
$\lambda_i(A), i = 1, \ldots n$	eigenvalues of $A \in \mathbb{R}^{n \times n}$
$\kappa(A)$	condition number of matrix A, i.e. $\kappa(A) = \lambda_1(A)/\lambda_n(A)$ where $\lambda_1(A) \leq \cdots \leq \lambda_n(A)$

vec(A)	non-symmetrized vectorization of matrix $A \in \mathbb{R}^{n\times n}$, i.e. vec($A$)=$(a_{11}, \ldots, a_{n1}, a_{12}, \ldots a_{n2}, \ldots a_{nn})^\top$
svec(A)	symmetrized vectorization of matrix $A \in \mathbb{S}^n$, i.e. svec(A)=$(a_{11}, \sqrt{2}a_{21} \ldots, \sqrt{2}a_{n1}, a_{22}, \sqrt{2}a_{31}, \ldots \sqrt{2}a_{n2}, \ldots a_{nn})^\top$
mat($\mathbf{x}$)	The inverse map of vec
smat($\mathbf{x}$)	The inverse map of svec
$A \otimes B$	Kronecker product of matrices A and B
$\|\mathbf{x}\|_1$	The ℓ_1 norm that is the sum of the absolute values of the vector $\mathbf{x}$
$\|\mathbf{x}\|_2$	The ℓ_2 norm that is the square root of the sum of the squared vector values
$\mathbb{R}[\mathbf{x}]$	Ring of the real polynomials in the variable $\mathbf{x} = (\mathrm{x}_1, \ldots, \mathrm{x}_n)$
$\mathbb{R}[\mathbf{x}]_d$	Ring of the real polynomials in the variable $\mathbf{x}$ with degree up to d
$\Sigma[\mathbf{x}] \subset \mathbb{R}[\mathbf{x}]$	Space of the SOS polynomials
$\mathbb{N}^n_d$	$\mathbb{N}^n_d := \{\alpha \in \mathbb{N}^n : \|\alpha\| \leq d\}$ where for a multi-index $\alpha \in \mathbb{N}^n$, $\|\alpha\| := \sum_{i=1}^{n} \alpha_i$
$\mathbf{x}^\alpha, \alpha \in \mathbb{N}^n$	List of monomials
$\mathrm{v}_d(\mathbf{x})$	Set of all the monomials $\mathbf{x}^\alpha$ with degree less than or equal to d and dimension n, i.e. $\mathrm{v}_d(\mathbf{x}) = (\mathbf{x}^\alpha),\ \alpha \in \mathbb{N}^n_d$
$\mathrm{y} = (y_\alpha) \subset \mathbb{R}, \alpha \in \mathbb{N}^n$	An infinite sequence of real numbers
$e = \mathbf{1} := (1, \ldots, 1)^\top$	(or e_n) Vector of ones of appropriate dimension
0	Zero matrices of appropriate dimension
0_n	$n-$vector of all zeros
I	Identity matrix with ones on the main diagonal and zero elsewhere

$\widetilde{E}_{k\ell}$	Matrix with 1 on the (k,ℓ) position if $k=\ell$, $\frac{1}{2}$ on the (k,ℓ) and (ℓ,k) positions if $k\neq\ell$ and zeros everywhere else
$n \gg m$	n is much greater than m
$n \approx m$	n is approximately equal to m
$n \lessapprox m$	n is lower bound approximation of m

Nomenclature

BQP Binary quadratic optimization problem

CAD Computer aided design

CG Conjugate gradient

IP Interior-point

KKT Karush- Kuhn-Tucker

LMI linear matrix inequality

Loraine Low-rank interior-point

LP Linear Programming

NT Nesterov-Todd

PCG Preconditioned conjugate gradient

POP Polynomial optimization problems

SCE Schur complement equation

SDP Semidefinite program

SMW Sherman-Morrison-Woodbury

SOS Sum-of-squares

TTO Truss topology optimization

UBQP Unconstrained binary quadratic optimization problem

Chapter One

Introduction

Polynomial optimization is an important area of optimization that focuses on optimizing a polynomial function under some polynomial constraints. These problems have a wide range of applications in many fields, including engineering, economics, computer science, statistics, and many others. Also, it provides a solid mathematical foundation for solving complex optimization problems and helps to develop efficient algorithms and computational tools to solve real-world problems. Polynomial optimization is NP-hard and it is suggested to approximate these problems using a hierarchy of convex (semidefinite) relaxations. There are two ways to create such relaxations; the Sum of Squares (SOS) representation of nonnegative polynomials and the dual theory of moments. The general approach started with the work of Shor [66, 67, 68, 69] and Nesterov [58]. Then, it was further developed by Lasserre [47, 48, 50] and Parrilo [61, 62], and there is broad literature available on this. The weakness of these relaxations is that in higher order relaxations, we need to solve the huge resulting Semidefinite Programming problems (SDP). Thus, we need to use SDP solvers for these large-scale problems.

The first efficient solvers for semidefinite optimization emerged more than twenty years ago. All of the general-purpose solvers have been using second-order algorithms, predominantly the Interior-Point (IP) method. In every iteration of such an algorithm, one has to solve a system

of linear equations $H\mathbf{x} = g$ of size $n \times n$, where n is the dimension of the unknown vector $\mathbf{x}$ in the linear semidefinite programming problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} c^\top \mathbf{x} \quad \text{subject to} \quad \sum_{i=1}^{n} \mathrm{x}_i A_i^{(k)} - B^{(k)} \succcurlyeq 0\,, \quad k = 1, \ldots, p$$

with $A_i^{(k)},\ B^{(k)} \in \mathbb{R}^{m\times m}$. The assembly and solution of this linear system is a well-known bottleneck of these solvers. This is the case even for problems with sparse data and sparse linear systems when the assembly of the system requires $\mathcal{O}\left(pnm^3\right)$ flops, and the solution by sparse Cholesky factorization $\mathcal{O}\left(n^\alpha\right)$ flops with some $\alpha \in [1, 3]$.

On the other hand, many real-world applications lead to large-scale SDP, often unsolvable by current general-purpose software. There are, essentially, three ways how to approach this conundrum:

1. Reformulating the problem to make it more suitable for general-purpose solvers. This can be done, for instance, by facial reduction [44, 63, 89] or by decomposition of large matrix inequalities (and thus reducing m) into several smaller ones [38, 41].

2. Using a different algorithm, such as spectral bundle [32], ADMM [59], augmented Lagrangian [54, 88], optimization on manifolds [36], randomized algorithms [84] or techniques of nonlinear programming [13, 14].

3. Using one of the second-order algorithms with Cholesky factorization replaced by an iterative method for the solution of the linear systems $H\mathbf{x} = g$; see, e.g., [42, 75, 86].

The last approach is particularly attractive whenever $n \gg m$, and addresses both bottlenecks of a second-order SDP solver.

- The assembling of the system matrix: an iterative solver only needs matrix-vector multiplications, and so the matrix does not have to be explicitly assembled and stored.

- The solution of the linear system itself: an iterative solver can handle very large systems, as compared to a sparse Cholesky solver, under the assumption of good conditioning of the matrix or the existence of a good preconditioner.

When the data matrices $A_i^{(k)}$ are of very low rank (such as rank one), the complexity of the linear system assembling can be substantially reduced by using this fact. This is particularly true for rank-one dense matrices. This fact is, however, rarely utilized in standard SDP software, due to the complications related to a different data input.

Our goal in this thesis is to introduce Loraine (Low-rank interior-point), a new general-purpose interior-point SDP solver targeted to problems with *low-rank data* and *low-rank solutions*. It employs special treatment of low-rank data matrices and, in particular, an option to use an iterative Krylov type method for the solution of the linear system. As the choice of an efficient preconditioner in a Krylov type method is problem dependent, in particular, in the context of optimization algorithms, we will focus on problems with expected very low-rank solutions.

SDP problems with low-rank solutions are common in relaxations of optimization problems in different areas such as combinatorial optimization [9], approximation theory [16, 49], control theory [49], structural optimization [72], and power systems [53].

It is *not* our goal to find a low-rank solution in case of non-unique solutions; indeed, it is well-known that an interior-point method will converge to a maximal complementary solution. However, if the method converges to a low-rank solution, we will use this fact to improve its computational complexity and memory demands. The proposed preconditioners are not limited to problems with (theoretically exactly) low-rank solutions, they can be efficiently applied when the solution has a few outlying eigenvalues.

There is one major difference between Loraine to other algorithms for SDP problems with low-rank solutions, such as SDPLR [14, 13], optimization on manifolds [36] and other approaches

[6, 84]. All these methods require the knowledge of a guaranteed (possibly tight) upper bound on the rank of the solution. If the estimate of the rank was lower than the actual rank, the algorithm would not find a solution. In our approach, the rank information is merely used to speed up a standard, well-established algorithm. If our estimated rank is smaller than the actual one, we will only see it in possibly more iterations of the Krylov solver; the convergence behavior of the optimization algorithm will be unchanged.

The actual optimization algorithm used in Loraine is a "standard" primal-dual interior-point method. The algorithm mimics the reliable primal-dual algorithm with Nesterov-Todd direction, as used, e.g., in the SDPT3 solver [74, 75].

Our preconditioner is motivated by the work of Zhang and Lavaei [86]. They presented a preconditioner for the Conjugate Gradient (CG) method within a standard IP method that makes it more efficient for large-and-sparse low-rank SDPs. We partly follow their approach, though there are some major differences: we abandon a crucial assumption on the sparsity of the data matrices (Assumption 2 in [86]); our method allows for SDPs with more than one LMI and with (simple) linear constraints; the complexity of computing our preconditioner is substantially lower; we can also efficiently solve problems with a few outlying eigenvalues in the optimal solution.

To test Loraine and the underlying algorithms, we have generated a library of problems with low-rank solutions. The first group arises from truss topology optimization without and with vibration constraints, formulated as SDP. These problems are characterized by very low-rank solutions and scalability, in the sense that one can generate a range of problems of various dimensions but of the same type. By comparing Loraine with other SDP solvers, we will demonstrate the efficiency of our approach. Next, we tested our algorithm on linear SDPs arising from Lasserre relaxations of the MAXCUT problem, randomly generated BQPs and a library of problems called BQPGKA. These problems are large-scale and have a low-rank solution of their SDP relaxations. In order to use the designed preconditioner, we rewrite the moment relaxations. As IP methods

cannot handle resulting linear equality constraints, we use an ℓ_1-penalty approach. The last group of problems is the Sensor Network Localization (SNL) problem with noisy data.

We further discuss applications of polynomial optimization problems in geometric modeling by presenting examples and analyzing the results.

The thesis is organized as follows: In Chapters 2 and 3, we discuss semidefinite programming, the interior point method, polynomial optimization, and convex relaxations which are the backgrounds of our research. Chapter 4 presents the theory of our developed algorithm called Loraine and it is divided into four sections: Section 4.1 introduces the SDP problems we want to solve and some basic assumptions. In Section 4.2, we introduce the preconditioners used with the iterative solvers. Section 4.3 describes two methods for solving SDPs with linear equality constraints. The last section of this chapter, Section 4.4, presents the solver Loraine, its special features, and the choice of parameters to use it. Chapters 5-7 describe our applications, truss topology optimization problems, the sensor network localization problem and Lasserre hierarchies of the unconstrained binary quadratic optimization problems, and the results of our numerical experiments on these problems. Loraine described in Chapter 4 and some of the numerical results presented in Chapters 5-7 are brought together in papers titled 'Loraine - An interior-point solver for low-rank semidefinite programming' [30] and 'Barrier and penalty methods for low-rank semidefinite programming with application to truss topology design' [28]. Also, as we needed to rewrite the moment relaxations of UBQPs to be able to utilize Loraine for these problems, we collected the results of Chapter 7 in more detail in a separate paper under the title of 'Solving Lasserre relaxations of unconstrained binary quadratic optimization problems by an interior-point method' [31]. Finally, in Chapter 8, we look at how Polynomial Optimization techniques can be used to solve Geometric Modeling problems and we present some examples performed in Julia. The result of this chapter is published by Springer as a chapter on 'Polynomial Optimization in Geometric Modeling' in the book titled 'Polynomial Optimization, Moments, and Applications' [29].
In the following, a more precise list of this thesis's contributions is presented:

- **Introduction of Loraine:** The primary contribution of this thesis is the introduction of Loraine, an Interior point solver designed for large-and-sparse semidefinite programming with low-rank solutions or solutions with few outlying eigenvalues, and/or problems with low-rank data. Loraine utilizes a preconditioned conjugate gradient method within an interior-point SDP algorithm with an efficient preconditioner fully utilizing the low-rank information. Loraine is demonstrating efficiency in various scenarios through numerical experiments on truss topology optimization, Lasserre relaxations of unconstrained binary quadratic optimization problems, and sensor network localization problems [28, 30].

- **Experiments with Lasserre Relaxations of UBQPs:** This contribution lies in our experiments involving Lasserre relaxations of the unconstrained binary quadratic optimization problems. To adapt these relaxations to Loraine's assumptions, we introduce a reformulation of the SDP relaxation of BQP, and we use an ℓ_1-penalty approach. Loraine is then tested on MAXCUT problems, randomly generated BQPs, and the BQPGKA library [31].

- **Application to Geometric Modeling Problems:** The thesis comprehensively reviews the application of Polynomial Optimization techniques to Geometric Modeling problems. A novel approach to address the Minimal Enclosing Ellipsoids problem is introduced, particularly when the center point is treated as a free variable rather than predefined [29].

Part I

Theory

Chapter Two

Background

In this chapter and the following one, we describe the necessary background of our work. Semidefinite programming and the interior-point method are described in this chapter and Chapter 3 focuses on polynomial optimization and semidefinite relaxations.

2.1 Semidefinite Programming

Semidefinite programming is one of the major tools for solving polynomial optimization problems, and there are some powerful software packages in this area. In this thesis, we introduce another software, called Loraine, with specific features to solve them.

First, we introduce some useful notations. We denote by $\mathbb{S}^m, \mathbb{S}^m_+$ and $\mathbb{S}^m_{++}$, respectively, the vector space of $m \times m$ real symmetric matrices, positive semidefinite and positive definite matrices. The symbol $\bullet$ denotes the Frobenius inner product of two matrices in the mentioned vector spaces, $A \bullet B = \text{tr}(A^\top B) = \sum_{i,j} A_{ij} B_{ij}$. Also, for symmetric matrices A and B, $A \bullet B = \text{svec}(A)^\top \text{svec}(B)$, where 'svec' of a matrix is the symmetrized vectorization of it. A matrix $M \in \mathbb{S}^m$ is called positive semidefinite (PSD) with the notation $M \succeq 0$ or $M \in \mathbb{S}^m_+$ (respectively positive definite

with the notation $M \succ 0$ or $M \in \mathbb{S}^m_{++}$) if $\mathbf{x}^\top M\mathbf{x} \geq 0$, for all $\mathbf{x} \in \mathbb{R}^m$ (respectively > 0, for all $\mathbf{x} \in \mathbb{R}^m \backslash \{0\}$). Any PSD matrix with rank r can be decomposed by Gram decomposition and be written as $M = VV^\top$ for some $V \in \mathbb{R}^{n\times r}$ [33]. We use this property later in Section 4.2 for developing a preconditioner.

SDP is linear optimization over the cone of positive semidefinite matrices. In both SDP and Linear Programming (LP), we want to optimize a linear function over the feasibility region. The feasible region in LP and SDP is made of the vector variable $\mathbf{x} \in \mathbb{R}^n, \mathbf{x} \geq 0$, and symmetric matrix variable $X \in \mathbb{S}^n, X \succeq 0$, respectively. LP and SDP problems are defined as the following standard forms:

$$\begin{aligned} &\min_{\mathbf{x}\in\mathbb{R}^m} \ c^\top \mathbf{x} \\ &\text{s.t.} \quad a_j^\top \mathbf{x} = b_j, \quad j = 1, \ldots, n, \\ &\qquad\quad \mathbf{x} \geq 0 \end{aligned} \tag{LP}$$

and

$$\begin{aligned} &\sup_{X\in\mathbb{S}^m} \ C \bullet X \\ &\text{s.t.} \quad A_j \bullet X = b_j, \quad j = 1, \ldots, n, \\ &\qquad\quad X \succeq 0, \end{aligned} \tag{SDP}$$

where $b_j \in \mathbb{R}$ and $c, a_j \in \mathbb{R}^n$, $C, A_j \in \mathbb{S}^m$. Here, the problem (SDP) is in the standard primal form of a semidefinite program, and we say X is *primal feasible* if $A_j \bullet X = b_j, j = 1, \ldots, n$, and $X \succeq 0$.

The dual SDP is defined as

$$\begin{aligned} &\inf_{\mathbf{y}\in\mathbb{R}^n} \ b^\top \mathbf{y} \\ &\text{s.t.} \quad \sum_{j=1}^{n} \mathrm{y}_j A_j - C \succeq 0 \quad . \end{aligned} \tag{SDD}$$

As mentioned before, this problem is a linear programming problem in the convex but non-polyhedral cone of positive semidefinite matrices. The constraint in (SDD) is also called linear matrix inequality constraint (LMI).

2.1.1 SDP and duality

Definition 2.1.1. *(Strict feasibility) Let us consider p^* and d^* to be the optimal values of* (SDP) *and* (SDD)*, respectively. The* (SDP) *problem is strictly feasible if there exists $X \succ 0$ which is primal feasible. Also,* (SDD) *is strictly feasible if there exists* y *with $\sum_{j=1}^{n} \mathrm{y}_j A_j - C \succ 0$. Strict feasibility is also called Slater's constraint qualification or Slater's regularity condition.*

Definition 2.1.2. *(Duality gap) Let us consider the primal and dual feasible sets be*

$$P = \{X \mid A_j \bullet X = b_j,\ j = 1, \ldots, n,\ X \succeq 0\},$$

and

$$D = \{\mathbf{y} \mid \sum_{j=1}^{n} \mathrm{y}_j A_j - C \succeq 0\},$$

Respectively. Let us consider $X \in P$ and $\mathbf{y} \in D$, then

$$C \bullet X - b^\top \mathbf{y}$$

is called the duality gap of (SDP) *and* (SDD)*.*

Now, by considering these definitions, we discuss the weak and strong duality theorems, in the following.

Theorem 2.1.3. *(Weak duality theorem) Let us consider the primal problem* (SDP) *and its dual* (SDD) *where both are feasible problems. Then, the weak duality holds which means that $p^* \leq d^*$.*

Proof. This proof is straightforward. If we consider X and $\mathbf{y}$ to be feasible for (SDP) and (SDD), respectively, then

$$0 \leq X \bullet (\sum_{j=1}^{n} \mathrm{y}_j A_j - C) = \sum_{j=1}^{n} (X \bullet A_j)\mathrm{y}_j - (C \bullet X) = b^\top \mathbf{y} - (C \bullet X).$$

The first inequality in the right is implied from the following Theorem 2.1.4. □

Theorem 2.1.4. *([18, Theorem A.8]) A symmetric matrix A is positive semidefinite if and only if for all positive semidefinite matrices B, $A \bullet B \geq 0$.*

Proof. From right to left: We want to show that if $A \bullet B \geq 0$ for all $B \succeq 0$ then the symmetric matrix A is positive semidefinite. Let $B = \mathbf{x}\mathbf{x}^\top$ where $\mathbf{x} \in \mathbb{R}^n$ is given, so $B \succeq 0$. Then

$$0 \leq A \bullet B = A \bullet \mathbf{x}\mathbf{x}^\top = \sum_{i,j} A_{ij}\mathrm{x}_i\mathrm{x}_j = \mathbf{x}^\top A\mathbf{x}.$$

From left to right: Now, we want to prove that if A and B are both positive semidefinite matrices, then their Frobenius inner product is also positive semidefinite. We have

$$A \bullet B = \mathrm{tr}(A^{\frac{1}{2}}A^{\frac{1}{2}}B^{\frac{1}{2}}B^{\frac{1}{2}}) = \mathrm{tr}(A^{\frac{1}{2}}B^{\frac{1}{2}}B^{\frac{1}{2}}A^{\frac{1}{2}}) = \|A^{\frac{1}{2}}B^{\frac{1}{2}}\|^2 \geq 0.$$

□

The strong duality theorem, guaranteeing a zero duality gap, holds for any feasible linear optimization (LP) problems when either the primal or dual problem is feasible. This theorem states that if the primal LP problem has an optimal solution, then the dual LP problem also has an optimal solution, and the optimal values of the objective functions are equal. In semidefinite programming, this duality does not always hold and it can be obtained under *Slater conditions*. The following example shows strong duality for SDPs does not always hold.

Example 2.1.5. *Let us consider the* (SDP) *where $n = m = 3$ and our data matrices are as follows:*

$$C = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{pmatrix}, \quad b = \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \\ 0 \end{pmatrix}$$

and

$$A_1 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad A_2 = \begin{pmatrix} 0 & 0 & 2 \\ 0 & 0 & 0 \\ 2 & 0 & 0 \end{pmatrix}, \; A_3 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

By considering the matrix $X = \begin{pmatrix} \mathrm{x}_{11} & \mathrm{x}_{12} & \mathrm{x}_{13} \\ \mathrm{x}_{12} & \mathrm{x}_{22} & \mathrm{x}_{23} \\ \mathrm{x}_{13} & \mathrm{x}_{23} & \mathrm{x}_{33} \end{pmatrix}$*, this primal SDP problem would be written as*

$$\begin{aligned} &\max \ -\mathrm{x}_{33} \\ &\textit{s.t.} \quad 2\mathrm{x}_{12} + 2\mathrm{x}_{22} + \mathrm{x}_{33} = \frac{1}{2} \\ &\qquad 4\mathrm{x}_{13} = \frac{1}{2} \\ &\qquad 2\mathrm{x}_{22} = 0 \\ &\qquad X \succeq 0. \end{aligned}$$

As $\mathrm{x}_{22} = 0$*, based on the positive definiteness of* X*,* $\mathrm{x}_{12} = 0$*. Finally, from the first constraint, we would have* $\mathrm{x}_{33} = \frac{1}{2}$ *and* $p^* = -\frac{1}{2}$*.*

Now, the dual of this SDP would be written as

$$\begin{aligned} &\min \ \frac{\mathrm{y}_1 + \mathrm{y}_2}{2} \\ &\textit{s.t.} \quad \begin{pmatrix} 0 & \mathrm{y}_1 & 2\mathrm{y}_2 \\ \mathrm{y}_1 & 2\mathrm{y}_1 + 2\mathrm{y}_3 & 0 \\ 2\mathrm{y}_2 & 0 & \mathrm{y}_1 + 1 \end{pmatrix} \succeq 0. \end{aligned}$$

As the first entry of the constraint matrix is zero and because of the positive definiteness of it, the first row and columns should just include zero entries. So, $\mathrm{y}_1 = \mathrm{y}_2 = 0$ *and to have the feasible solution* $\mathrm{y}_3 \geq 0$*. Based on this set of feasible solutions* $d^* = 0$*. This is an example that violates Slater's conditions and demonstrates that strong duality, which holds for linear programs, does not hold in general for semidefinite programs.*

Before we go through the strong duality theorem, we present the Theorem 2.1.6 which is needed throughout the proof. This theorem is called *Seperation theorem for convex sets*.

Theorem 2.1.6. *([18, Theorem B.4]) Let us consider two nonempty convex sets* C_1 *and* C_2 *in* $\mathbb{R}^m$*.*

The relative interiors of these two sets are disjoint if and only if there exists a $r \in \mathbb{R}^m$ *such that*

$$\sup_{\mathbf{x}\in C_1} \; r^\top \mathbf{x} \leq \inf_{\mathbf{y}\in C_2} \; r^\top \mathbf{y}$$

and

$$\inf_{\mathbf{x}\in C_1} \; r^\top \mathbf{x} < \sup_{\mathbf{y}\in C_2} \; r^\top \mathbf{y}.$$

Theorem 2.1.7. *([18, Theorem 2.2])(Strong duality theorem) Let us consider the problem* (SDP) *and its dual* (SDD)*. The strong duality, i.e.* $p^* = d^*$*, holds in any of the following circumstances:*

(i) If (SDD) *is strictly feasible and* (SDP) *is feasible, then* $p^* = d^*$.

(ii) If (SDP) *is strictly feasible and* (SDD) *is feasible, then* $p^* = d^*$.

Proof. It is enough to prove (i), and as the dual of the dual problem is primal, the statement (ii) will also be proved.

In this proof, we consider the (SDP) and (SDD) problems be equivalently written as

$$p^* := \inf_{X\in\mathbb{S}^m} \Big\{ -C \bullet X \;\Big|\; A_j \bullet X = b_j,\; j = 1, \ldots, n,\; X \succeq 0 \Big\},$$

and

$$d^* := \sup_{\mathbf{y},S} \Big\{ -b^\top \mathbf{y} \;\Big|\; \sum_{j=1}^{n} \mathrm{y}_j A_j - C = S,\; S \succeq 0 \Big\}$$

respectively. By this new formulation, the weak duality holds when $d^* \leq p^*$. For $b = 0$, strong duality will trivially hold, and the optimal solution of (SDP) will be zero. Thus, we consider the case where $b \neq 0$.

First, we define a nonempty convex set

$$M = \Big\{ S \in \mathbb{S}^m \;\Big|\; S = \sum_{j=1}^{n} \mathrm{y}_j A_j - C,\; -b^\top \mathbf{y} \geq d^*,\; \mathbf{y} \in \mathbb{R}^n \Big\}.$$

As this set is nonempty, its relative interiors and the relative interiors of the cone $\mathbb{S}^m_+$ will be disjoint. Otherwise, it will contradict with d^* being the optimal value of the dual problem.

Now, as these two convex sets are disjoint, we can use the separation theorem Theorem 2.1.6, by associating $\mathbb{S}^m$ with $\mathbb{R}^{\frac{1}{2}m(m+1)}$ and we use this property of the Frobenius inner product that $\text{svec}(A)^\top \text{svec}(B) = A \bullet B$ for $A, B \in \mathbb{S}^m$. Thus, there exist a $R \in \mathbb{S}^m$ such that

$$\sup_{S \in M} (S \bullet R) \leq \inf_{U \in \mathbb{S}^m_+} (U \bullet R). \tag{2.1}$$

$\mathbb{S}^m_+$ is a cone, so the infimum on the right must be either zero or $-\infty$. As the set M is nonempty, then the infimum is zero and $R \succeq 0$. In addition, one has

$$\sup_{S \in M} (S \bullet R) \leq 0. \tag{2.2}$$

For each $S \in M$, there exist a $\mathbf{y} \in \mathbb{R}^n$ such that $-b^\top \mathbf{y} \geq d^*$ and by multiplying R to the either sides of the constrain $S = \sum_{j=1}^n \mathrm{y}_j A_j - C$, we get

$$\sum_{j=1}^n \mathrm{y}_j (A_j \bullet R) = (S + C) \bullet R = S \bullet R + C \bullet R$$

and from (2.2), the next inequality follows

$$\sum_{j=1}^n \mathrm{y}_j (A_j \bullet R) \leq C \bullet R. \tag{2.3}$$

Based on this inequality, one can see the function

$$f(\mathbf{y}) = \sum_{j=1}^n \mathrm{y}_j (A_j \bullet R)$$

is bounded above on the half-space $b^\top \mathbf{y} \leq -d^*$. This happens when for some $\beta \geq 0$,

$$A_j \bullet R = \beta b_j, \quad j = 1, \ldots, n. \tag{2.4}$$

If $\beta = 0$, then $A_j \bullet R = 0$ for all $j = 1, \ldots, n$ and by (2.3), $C \bullet R \geq 0$. By the strict feasibility of the dual problem, there exists $(\mathbf{y}^0, S^0)$ in the feasible set D in which $S^0 \succ 0$. By the inner product of S^0 and R, one gets

$$S^0 \bullet R = \sum_{j=1}^n \mathrm{y}_j^0 (A_j \bullet R) - C \bullet R = 0 - C \bullet R \leq 0.$$

Thus, $S^0 \bullet R \leq 0$ which is contradictiong with $S^0 \succ 0$ and $R \succeq 0$. So, $\beta > 0$ and as the optimal value of the SDP problem X^* must be positive semidefinite, we can define

$$X^* = \frac{1}{\beta} R \succeq 0.$$

This X^* is feasible for the SDP problem as by (2.4), we get $A_j \bullet X = b_j,\ j = 1, \ldots, n$. By multiplying each sides of (2.3) by $-\frac{1}{\beta}$, for all $\mathbf{y} \in \mathbb{R}^n$ such that $-b^\top \mathbf{y} \geq d^*$, one will have

$$-C \bullet x^* \leq -b^\top \mathbf{y},$$

so, $-C \bullet X^* \leq d^*$ and equivalently $p^* \leq d^*$ and by the weak duality $p^* = d^*$.

□

2.2 Interior-Point Algorithm

In this section, we will describe a primal-dual predictor-corrector interior-point method for SDP which is using the Nesterov–Todd (NT) direction. We will closely follow the articles by Todd, Toh and Tütüncü [74] and by Toh and Kojima [76] and only repeat formulas needed to present the structure of the matrix that is required to develop the preconditioners in Section 4.2. The basic framework of our interior-point solver thus, more or less, mimics that of the software SDPT3.

The general SDP problem of our concern and its features is introduced in Section 4.1. This problem includes linear constraints and several matrix variables; however, for the sake of simplicity, we ignore the linear constraints in the following development and consider only one matrix variable. More variables can, of course, be formally included in a single block diagonal matrix.

Therefore, in this section, we consider the standard SDP with one matrix variable $X \in \mathbb{S}^m$

and no linear constraints

$$\begin{aligned} &\min_{X} \; C \bullet X \\ &\text{subject to } A_j \bullet X = b_j, \quad j = 1, \ldots, n \\ &\qquad X \succcurlyeq 0\,. \end{aligned} \tag{2.5}$$

Here $A_j \in \mathbb{S}^m$, $j = 1, \ldots, n$, $C \in \mathbb{S}^m$, $b \in \mathbb{R}^n$ are data of the problem, and the dual problem of this SDP is:

$$\begin{aligned} &\max_{\mathbf{y},\, S} \; b^\top \mathbf{y} \\ &\text{subject to } \sum_{j=1}^{n} \mathrm{y}_j A_j + S = C, \\ &\qquad S \succcurlyeq 0\,. \end{aligned} \tag{2.6}$$

where $\mathbf{y} \in \mathbb{R}^n$ and $S \in \mathbb{S}^m$.

2.2.1 Basic framework

Let $\mathbf{A}$ be an $m^2 \times n$ matrix defined by

$$\mathbf{A} := [\mathrm{vec}A_1, \ldots, \mathrm{vec}A_n].$$

Our algorithm follows the standard steps of a primal-dual interior-point method. We write down optimality conditions for (2.5) with relaxed complementarity condition:

$$\begin{aligned} \mathbf{A}^\top \mathrm{vec}(X) &= b, \\ \mathbf{A}\mathbf{y} + S &= C, \\ XS &= \sigma\mu I\,, \end{aligned} \tag{2.7}$$

where $\mu = \frac{X \bullet S}{m}$ and σ is a centering parameter to be specified below. This system of nonlinear equations is solved repeatedly until convergence.

2.2.2 Newton direction

The system (2.7) is solved approximately by Newton's method, where in every iteration of the method we solve the following system of linear equations in variables $(\Delta X, \Delta \mathbf{y}, \Delta S)$, the Newton direction:

$$\mathbf{A}^\top \text{vec}(\Delta X) = r_p, \tag{2.8a}$$

$$\mathbf{A}\Delta \mathbf{y} + \text{vec}(\Delta S) = \text{vec}(R_d), \tag{2.8b}$$

$$H_W(X\Delta S + \Delta X S) = \sigma\mu I - H_W(XS)\,. \tag{2.8c}$$

Here

$$r_p := b - \mathbf{A}^\top \text{vec}(X), \quad R_d := C - S - \sum_{i=1}^{n} \mathrm{y}_i A_i,$$

and H_W is a linear transformation guaranteeing symmetry of the resulting matrix, in particular

$$H_W(M) = \frac{1}{2}\left(WMW^{-1} + W^{-T}M^T W^T\right)$$

with some invertible matrix $W \in \mathbb{S}^m_{++}$ known as the *scaling matrix* [87]. There are different ways to construct this W. By assuming $(X, \mathbf{y}, S)$ to be the current iterate, the most studied choices of W are primal scaling (where $W = X$), dual scaling (where $W = S^{-1}$) and Nesterov-Todd (NT) scaling. We define the so-called Nesterov-Todd scaling matrix

$$W = S^{-\frac{1}{2}}\left(S^{\frac{1}{2}} X S^{\frac{1}{2}}\right)^{\frac{1}{2}} S^{-\frac{1}{2}} = X^{\frac{1}{2}}\left(X^{-\frac{1}{2}} S X^{-\frac{1}{2}}\right)^{\frac{1}{2}} X^{\frac{1}{2}}$$

satisfying $X = WSW$ and $S = WX^{-1}W$. This is the scaling matrix used in Loraine.

The primary objective of scaling is to determine a symmetrization strategy through the operator $H_W(M)$, ensuring the symmetry of any matrix M. The simplest symmetrization involves setting W as the identity matrix, as suggested by Alizadeh, Haeberly, and Overton. Additionally, the Nesterov-Todd primal-dual scaling used in this context adjusts the scale of primal and dual variables relative to each other. This helps prevent, for example, excessively large numbers in S and excessively small numbers in X.

As in the case of linear programming, instead of solving the linear system of $2m^2 + n$ equations (2.8a)–(2.8c) directly, we can solve a Schur Complement Equation (SCE) involving only $\Delta\mathbf{y}$. The general bottleneck of any interior-point method for SDP is assembling and solving this SCE

$$H\Delta\mathbf{y} = r. \tag{2.9a}$$

Here H is the Schur complement matrix with elements

$$H_{ij} = A_i \bullet WA_jW, \quad i, j = 1, \ldots, n, \tag{2.9b}$$

and

$$r = r_p + \mathbf{A}^\top \text{vec}\left(WR_dW + WSW\right) . \tag{2.9c}$$

For details of computing W, r, ΔX and ΔS efficiently for the NT scaling, see [76].

By considering (2.9a) and (2.9b) we have the following linear system

$$(H\Delta\mathbf{y})_i = A_i \bullet \left[W\left(\sum_{j=1}^{n} \Delta \mathrm{y}_i A_j\right) W\right] = r_i, \quad i = 1, \ldots, n. \tag{2.10}$$

Finally, vectorizing the matrix variables allows (2.10) to be written as

$$\left(\mathbf{A}^\top (W \otimes W)\mathbf{A}\right)\Delta\mathbf{y} = r\,. \tag{2.11}$$

2.2.3 Interior-point algorithm

In the following Algorithm 1, we use a second-order correction to the NT direction. Suppose that we have approximations δX and δS to the search directions. The correction R_{NT} is given by

$$R_{NT} = -\left(G^{-1}\left(\delta X \delta S\right) G + G^\top \left(\delta S \delta X\right) G^{-T}\right) ./ \left(de^\top + ed^\top\right), \tag{2.12}$$

where "./" is entry-wise division, G satisfies $W = GG^\top$, $d := \text{diag}(G^\top SG)$ and $e := (1, \ldots, 1)^\top$. With this correction, we define a new right-hand side of (2.9b) as

$$r = r_p + \mathbf{A}^\top \text{vec}\left(WR_dW + WSW - \sigma\mu S^{-1} - GR_{NT}G^\top\right) . \tag{2.13}$$

The algorithm essentially constitutes a Newton step but with the requirement that, at the end of the process, we remain within the interior of the feasible set. It begins by computing the Newton direction, followed by calculating step lengths α and β to ensure feasibility within the region. While a line search is typically employed, it is unnecessary here due to the linear SDP nature of the problem. The precise step to the boundary of the feasible region can be computed, and so to remain within the interior, the step size is shortened by choosing $\tau = 0.9$. This adjustment ensures that we stay within the interior, as choosing $\tau = 1$ would already result in a positive semidefinite matrix, precisely reaching the boundary.

Algorithm 1 NT-IP Algorithm

Given an initial iterate $(X^0, \mathbf{y}^0, S^0)$, with X^0, S^0 positive definite. Choose $\tau = 0.9$. Let $\mu = \frac{X \bullet S}{m}$.

1: **for** $\ell = 0, 1, 2, \ldots$ **do**

2: *Predictor step*: For $(X^\ell, \mathbf{y}^\ell, S^\ell)$ compute the Newton direction $(\delta X, \delta \mathbf{y}, \delta S)$ from (2.8a)-(2.8c) and (2.9a) with right-hand side (2.9c).

3: Find

$$\alpha := \min\left(1, \frac{-\tau}{\lambda_{\min}\left((X^\ell)^{-1}\delta X\right)}\right), \quad \beta := \min\left(1, \frac{-\tau}{\lambda_{\min}\left((S^\ell)^{-1}\delta S\right)}\right) \tag{2.14}$$

ensuring that $X^\ell + \alpha\delta X$ and $S^\ell + \beta\delta S$ are positive definite. Set

$$\sigma := \left[\frac{(X^\ell + \alpha\delta X) \bullet (S^\ell + \beta\delta S)}{X^\ell \bullet S^\ell}\right]^3.$$

4: *Corrector step*: Compute the Newton correction $(\Delta X, \Delta \mathbf{y}, \Delta S)$ from (2.8a)-(2.8c) and (2.9a) with right-hand side (2.13).

5: Update $(X^\ell, \mathbf{y}^\ell, S^\ell)$ by

$$X^{\ell+1} = X^\ell + \alpha\Delta X, \quad \mathbf{y}^{\ell+1} = \mathbf{y}^\ell + \beta\Delta \mathbf{y}, \quad S^{\ell+1} = S^\ell + \beta\Delta S$$

where α and β are chosen as in (2.14) with δX, δS replaced by ΔX, ΔS, so that $X^{\ell+1}$ and $S^{\ell+1}$ are positive definite.

6: **end for**

Chapter Three

Polynomial and Semi-algebraic Optimization

In this chapter, we talk about polynomial optimization problems, their importance, and the way how these global minimization problems are translated into SDP problems. Then, we describe SOS and moment approach to solve these problems. Throughout this chapter, we largely follow books by Lasserre [46] and Laurent [52].

3.1 Polynomial Optimization

We consider the global optimization problem

$$\begin{aligned} f^* := \inf_{\mathbf{x}} \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & \mathrm{g}_j(\mathbf{x}) \geq 0, \qquad j = 1, \ldots, m \\ & \mathbf{x} \in \mathbb{R}^n, \end{aligned} \tag{3.1}$$

with $f, \mathrm{g}_j : \mathbb{R}^n \to \mathbb{R}$ being continuous function.

There is no general-purpose method to solve (3.1), even to reach an approximation of f^*. However,

if we restrict this problem to a polynomial optimization problem, we might be able to solve it exactly or at least as close as desired to f^*. Polynomial Optimization Problems (POP) are problems of the form

$$\inf_{\mathbf{x}} \{f(\mathbf{x}) : \mathbf{x} \in \mathbf{K}\}, \tag{3.2}$$

where

$$\mathbf{K} := \{\mathbf{x} \in \mathbb{R}^n : \mathrm{g}_j(\mathbf{x}) \geq 0, \quad j = 1, \ldots, m\}. \tag{3.3}$$

$\mathbf{K}$ is called a semi-algebraic set and is a feasible set in which the functions $f, \mathrm{g}_j : \mathbb{R}^n \to \mathbb{R},\ j = 1, \ldots, m,$ are all polynomials. Problem (3.2) is a special instance of nonlinear nonconvex optimization. The specificity of polynomial optimization can help us to design a technique that may reach the global optimum. A significant subclass of POP is the class of linear and quadratic optimization problems. Notice that minimizing a polynomial is the same as finding the largest scalar λ such as $f(\mathbf{x}) - \lambda$ is nonnegative over the feasible region $\mathbf{K}$.

In the following section, we will discuss the importance of nonnegative polynomials and how, if a nonnegative polynomial has a sum of squares representation, we can solve it using SDP methods.

3.2 Convex Relaxations

There are various approaches to solving polynomial optimization problems, such as using a hierarchy of convex (semidefinite) relaxations to approximate (3.2). Such relaxations can be built using two methods: the SOS representation of nonnegative polynomials and the dual theory of moments. This section will only briefly discuss the main development and these two dual approaches. Each will give additional complementary information about the problem. In addition, more examples of these relaxations can be found in Section 8.2.

3.2.1 Sum of Squares Approach

Let $\mathbb{R}[\mathbf{x}]$ be the ring of real multivariate polynomials in the n-tuple variables $\mathbf{x} = (\mathrm{x}_1, \ldots, \mathrm{x}_n)$. The polynomial $p \in \mathbb{R}[\mathbf{x}]$ for $\mathbf{x} \in \mathbb{R}^n$ is a Sum of Squares (SOS) if for a finite number of polynomials $(p_j)_{j\in J} \subset \mathbb{R}[\mathbf{x}]$, it can be expressed as

$$\mathbf{x} \mapsto p(\mathbf{x}) = \sum_{j\in J} p_j(\mathbf{x})^2. \tag{3.4}$$

The next lemma demonstrates that the degree of p must necessarily be even and the degree of every p_j must be bounded by $\frac{\deg(p)}{2}$.

Lemma 3.2.1. *([52, Lemma 3.1]) If $p \in \mathbb{R}[\mathbf{x}]$ is a SOS, then $\deg(p)$ is even and any decomposition $p = \sum_{j\in J} p_j^2$ where $p_j \in \mathbb{R}[\mathbf{x}]$ satisfies $\deg(p_j) \leq \frac{\deg(p)}{2}$ for all j.*

Obviously, every polynomial which is SOS is nonnegative on $\mathbb{R}^n$. The following theorem shows that for the real polynomial with a single variable (univariate case), the ring $\mathbb{R}[\mathrm{x}]$ has a fundamental property that every nonnegative polynomial is a SOS of polynomials. However, not all nonnegative polynomials are SOS in multiple dimensions (multivariate case).

Theorem 3.2.2. *([46, Theorem 2.3].) One can write a polynomial $p \in \mathbb{R}[\mathrm{x}]$ with even degree as the sum of the squares of other polynomials, which means that for $h_i \in \mathbb{R}[\mathrm{x}], i = 1, \ldots, k$, $p(\mathrm{x}) = \sum_{i=1}^{k} h_i(\mathrm{x})^2$, if and only if p is nonnegative.*

Let $\Sigma[\mathbf{x}] \subset \mathbb{R}[\mathbf{x}]$ denote the space of SOS polynomials, and let $\mathbb{N}_d^n := \{\alpha \in \mathbb{N}^n : |\alpha| \leq d\}$ in which for a multi-index $\alpha \in \mathbb{N}^n$, $|\alpha| := \sum_{i=1}^{n} \alpha_i$. For the set of all monomials $\mathbf{x}^\alpha$ with degree less than or equal to d and dimension $s(d) := \binom{n+d}{d}$, consider the vector

$$\begin{aligned} \mathrm{v}_d(\mathbf{x}) &= (\mathbf{x}^\alpha), \qquad \alpha \in \mathbb{N}_d^n \\ &= (1, \mathrm{x}_1, \ldots, \mathrm{x}_n, \mathrm{x}_1^2, \mathrm{x}_1\mathrm{x}_2, \ldots, \mathrm{x}_{n-1}\mathrm{x}_n, \mathrm{x}_n^2, \ldots, \mathrm{x}_1^d, \ldots, \mathrm{x}_n^d)^T. \end{aligned} \tag{3.5}$$

These monomials are the canonical basis of the polynomials vector space $\mathbb{R}[\mathbf{x}]_d$ with degrees up to d. In (3.4), consider the polynomial $p_j(\mathbf{x}) = \overline{p_j}^T \mathrm{v}_d(\mathbf{x})$, where $\mathrm{v}_d(\mathbf{x})$ is a vector in the monomial base, and $\overline{p_j}, j \in J$, is the vector of coefficients of the polynomial in a monomials form. By this definition of $p_j(\mathbf{x})$, we can rewrite (3.4) for all $\mathbf{x} \in \mathbb{R}^n$ as

$$p(\mathbf{x}) = \sum_{j \in J} \mathrm{v}_d(\mathbf{x})^T \overline{p_j}\ \overline{p_j}^T \mathrm{v}_d(\mathbf{x}) = \mathrm{v}_d(\mathbf{x})^T (\sum_{j \in J} \overline{p_j}\ \overline{p_j}^T) \mathrm{v}_d(\mathbf{x}).$$

The polynomial p is said to be *homogeneous* if all its terms have the same degree.

Define $Q = \sum_{j \in J} \overline{p_j}\ \overline{p_j}^T$. So, if a positive semidefinite matrix Q exists as indicated above, the polynomial $p(\mathbf{x})$ is SOS. The ensuing proposition is the result of this.

Proposition 3.2.3. *([46, Proposition 2.1].) A polynomial $p \in \mathbb{R}[\mathbf{x}]_{2d}$ has an SOS decomposition if and only if a real symmetric and positive semidefinite matrix $Q \in \mathbb{R}^{s(d) \times s(d)}$ exists such that $p(\mathbf{x}) = \mathrm{v}_d(\mathbf{x})^T Q \mathrm{v}_d(\mathbf{x})$, for all $\mathbf{x} \in \mathbb{R}^n$.*

One benefit of this claim is that we can solve problems with nonnegative polynomials with SOS representations using semidefinite optimization techniques.

For a given SOS polynomial $p \in \mathbb{R}[\mathbf{x}]_{2d}$, the matrix Q must satisfy the identity $p(\mathbf{x}) = \mathrm{v}_d(\mathbf{x})^T Q \mathrm{v}_d(\mathbf{x})$ for all $\mathbf{x} \in \mathbb{R}^n$. We can write

$$\mathrm{v}_d(\mathbf{x}) \mathrm{v}_d(\mathbf{x})^T = \sum_{\alpha \in \mathbb{N}^n_{2d}} \mathrm{B}_\alpha \mathbf{x}^\alpha,$$

where $\mathrm{B}_\alpha, \alpha \in \mathbb{N}^n_{2d}$ are $s(d) \times s(d)$ real symmetric matrices. To determine if the polynomial $\mathbf{x} \mapsto p(\mathbf{x}) = \sum_\alpha p_\alpha \mathbf{x}^\alpha$ is SOS, it can be simplified to solving a semidefinite optimization problem. This problem finds $Q \in \mathbb{R}^{s(d) \times s(d)}$ where

$$Q = Q^T, \quad Q \succeq 0, \quad \langle Q, B_\alpha \rangle = p_\alpha, \quad \forall \alpha \in \mathbb{N}^n_{2d}, \tag{3.6}$$

which is a traceable convex optimization problem. There are efficient software packages available for this purpose, such as Mosek [57], SDPT3 [74, 75], and Loraine the software presented in this thesis.

In the following example, we illustrate how to find whether a polynomial with two variables is an SOS.

Example 3.2.4. *Take a look at the polynomial in* $\mathbb{R}[\mathbf{x}] = \mathbb{R}[\mathrm{x}_1, \mathrm{x}_2]$

$$\mathbf{x} \mapsto p(\mathbf{x}) = 17\mathrm{x}_1^4 + 12\mathrm{x}_1^3\mathrm{x}_2 - 4\mathrm{x}_1\mathrm{x}_2^3 - 11\mathrm{x}_1^2\mathrm{x}_2^2 + 4\mathrm{x}_2^4, \quad \mathbf{x} \in \mathbb{R}^2.$$

To determine whether the homogeneous polynomial p *is SOS, it can be written as*

$$\begin{aligned} p(\mathrm{x}_1, \mathrm{x}_2) &= \begin{pmatrix} \mathrm{x}_1^2 & \mathrm{x}_2^2 & \mathrm{x}_1\mathrm{x}_2 \end{pmatrix} \begin{bmatrix} q_{11} & q_{12} & q_{13} \\ q_{12} & q_{22} & q_{23} \\ q_{13} & q_{23} & q_{33} \end{bmatrix} \begin{pmatrix} \mathrm{x}_1^2 \\ \mathrm{x}_2^2 \\ \mathrm{x}_1\mathrm{x}_2 \end{pmatrix} \\ &= q_{11}\mathrm{x}_1^4 + q_{22}\mathrm{x}_2^4 + (q_{33} + 2q_{12})\mathrm{x}_1^2\mathrm{x}_2^2 + 2q_{13}\mathrm{x}_1^3\mathrm{x}_2 + 2q_{23}\mathrm{x}_1\mathrm{x}_2^3, \end{aligned}$$

for some $Q \succeq 0$*. Thus, we get*

$$Q \succeq 0, \quad q_{11} = 17, \quad q_{22} = 4, \quad q_{33} + 2q_{12} = -11, \quad 2q_{13} = 12, \quad 2q_{23} = -4$$

in order to have an identity. There is a particular solution

$$0 \preceq Q = \begin{bmatrix} 17 & -8 & 6 \\ -8 & 4 & -2 \\ 6 & -2 & 5 \end{bmatrix} = HH^T, \quad \textit{for} \quad H = \begin{bmatrix} 4 & 1 \\ -2 & 0 \\ 1 & 2 \end{bmatrix}.$$

So, p *can be written as*

$$p(\mathbf{x}) = (4\mathrm{x}_1^2 - 2\mathrm{x}_2^2 + \mathrm{x}_1\mathrm{x}_2)^2 + (\mathrm{x}_1^2 + 2\mathrm{x}_1\mathrm{x}_2)^2$$

which is an SOS.

Positivity on a Compact Basic Semi-algebraic Set

In this section, we consider $\mathbf{K} \subset \mathbb{R}^n$ as in (3.3) which is a compact semi-algebraic set. We discuss the ways to get a positivity certificate over $\mathbf{K}$. To do this, we define a quadratic module and a preordering set.

Definition 3.2.5. *The **quadratic module** generated by the finite family* $(\mathrm{g}_i, i = 1, \ldots, m) \subset \mathbb{R}[\mathbf{x}]$ *is defined as the set*

$$\mathcal{Q}(\mathrm{g}) := \Big\{\sigma_0 + \sum_{i=1}^{m} \sigma_i \mathrm{g}_i : \sigma_0, \sigma_i \in \Sigma[\mathbf{x}],\ i = 1, \ldots, m\Big\}. \tag{3.7}$$

*The **preordering** on* $\mathbb{R}[\mathbf{x}]$ *generated by the polynomials* $\mathrm{g}_1, \ldots, \mathrm{g}_m$ *is defined as the set*

$$\mathcal{P}(\mathrm{g}) := \Big\{ \sum_{J \subseteq \{1,\ldots,m\}} \sigma_J \mathrm{g}_J : \sigma_J \in \Sigma[\mathbf{x}]\Big\}, \tag{3.8}$$

in which for $J \subseteq \{1, \ldots, m\}$, $\mathrm{g}_J \in \mathbb{R}[\mathbf{x}]$ *indicates the polynomial* $\mathbf{x} \mapsto \mathrm{g}_J(\mathbf{x}) = \prod_{j \in J} \mathrm{g}_j(\mathbf{x})$, *where* $\mathrm{g}_\emptyset$ *is defined as* 1. *It is clear that* $\mathcal{Q}(\mathrm{g}) \subseteq \mathcal{P}(\mathrm{g})$.

Assumption 3.2.1. *Let* $(\mathrm{g}_j)_{j=1}^m \subset \mathbb{R}[\mathbf{x}]^n$. *There exist* $u \in \mathcal{Q}(\mathrm{g})$ *such that the level set* $\{\mathbf{x} \in \mathbb{R}^n : u(\mathbf{x}) \geq 0\}$ *is compact.*

Since $\mathbf{K} \subseteq \{\mathbf{x} : u(\mathbf{x}) \geq 0\}$ for any $u \in \mathcal{Q}(\mathrm{g})$, Assumption 3.2.1 indicates that $\mathbf{K}$ is compact. This assumption is on the description of $\mathbf{K}$ rather than the set $\mathbf{K}$. For example, if the set $\{\mathbf{x} \in \mathbb{R}^n : \mathrm{g}_j(\mathbf{x}) \geq 0\}$ is compact for some j, then Assumption 3.2.1 holds. The following theorem shows different conditions under which the positivity certificate can be obtained.

Theorem 3.2.6. *([46, Theorem 2.13, Theorem 2.14, Theorem 2.15], [45] and [71].)*

1. *(Schmüdgen's Positivstellensatz) Assume that* $\mathbf{K}$ *as in* (3.3) *is compact. If* $f \in \mathbb{R}[\mathbf{x}]$ *is strictly positive on* $\mathbf{K}$ *then* $f \in \mathcal{P}(\mathrm{g})$.

2. *(Putinar's Positivstellensatz) Let us consider* $\mathbf{K}$ *to be as defined before and let Assumption 3.2.1 holds. If* $f \in \mathbb{R}[\mathbf{x}]$ *is strictly positive on* $\mathbf{K}$ *then* $f \in \mathcal{Q}(\mathrm{g})$.

3. *(Positivstellensatz for general* $\mathbf{K}$*) For the general* $\mathbf{K}$, $f \in \mathbb{R}[\mathbf{x}]$ *is strictly positive on* $\mathbf{K}$ *if and only if*

$$\exists p, q \in \mathcal{P}(\mathrm{g}) \quad \textit{s.t.} \quad pf = q + 1,$$

and, $f \in \mathbb{R}[\mathbf{x}]$ is positive on $\mathbf{K}$ if and only if

$$\exists p, q \in \mathcal{P}(\mathrm{g}) \quad \& \quad \exists k \in \mathbb{N} \quad \textit{s.t.} \quad pf = f^{2k} + q.$$

4. *(Archimedean condition) Let Assumption 3.2.1 holds, then*

$$\forall p \in \mathbb{R}[\mathbf{x}], \quad \exists N \in \mathbb{N} \quad \textit{for which} \quad N \pm p \in \mathcal{Q}(\mathrm{g}).$$

Sum-of-Squares (SOS) Relaxation

The SOS relaxation is a type of convex relaxation method that replaces polynomial non-negativity constraints with SOS constraints. These SOS constraints are computationally more manageable and can be formulated using semidefinite programs. By applying this relaxation, nonlinear polynomial optimization problems can be transformed into a sequence of convex problems, whose level of complexity is determined by a single-degree parameter. In this section, this hierarchy of semidefinite relaxations is provided. The optimal values of these relaxations form a monotone non-decreasing sequence of lower bounds that converge to the global optimum f^* of problem (3.1). It means that we design bounds that can be computed using SDP. We need to restrict the degree of the SOS multiplier σ_j in $\Sigma[\mathbf{x}]$. We pick an integer d which is our degree bound and we restrict the quadratic module to the case where all the summons have the degree at most $2d$. So, to approximate the solutions of (3.2), we define the following finite-dimensional convex cones, also called **truncated quadratic module** generated by g_j for $j = 0, \ldots, m$ and with $\mathrm{g}_0 = 1$ to be

$$\mathcal{Q}(\mathrm{g})_d := \Big\{ \sum_{j=0}^{m} \sigma_j \mathrm{g}_j : \sigma_j \in \Sigma[\mathbf{x}],\ \deg(\sigma_j \mathrm{g}_j) \leq 2d,\ j = 0, \ldots, m \Big\}. \tag{3.9}$$

We can equivalently say

$$\mathcal{Q}(\mathrm{g})_d := \Big\{ \sum_{j=0}^{m} \sigma_j \mathrm{g}_j : \sigma_j \in \Sigma[\mathbf{x}]_{2d},\ j = 0, \ldots, m \Big\},$$

where as mentioned before, $\Sigma[\mathbf{x}]_d = \{p = \sum_i q_i^2,\ q_i \in \mathbb{R}[\mathbf{x}]_{\frac{d}{2}}\}$ is the convex cone of polynomials of degree $\leq d$, which are sums of squares, and $\mathbb{R}[\mathbf{x}]_d$ is the vector space of polynomials of this

degree in the variables $\mathbf{x} = (\mathrm{x}_1, \ldots, \mathrm{x}_n)$.

The truncated quadratic module $\mathcal{Q}(\mathrm{g})_d$ is a convex cone by demonstrating that it maintains its properties under multiplication by positive scalars and addition of its elements. Moreover, the polynomials in $\mathcal{Q}(\mathrm{g})_d$ have been designed to be non-negative on $\mathbf{K}$.

As we mentioned before, minimizing a polynomial $f(\mathbf{x})$ over $\mathbf{K}$ is the same as finding the largest scaler λ such as $f(\mathbf{x}) - \lambda$ is nonnegative over the feasible region $\mathbf{K}$. So,

$$f^* = \inf_{\mathbf{x}\in\mathbf{K}} f(\mathbf{x}) = \Big\{ \sup_{\lambda,\mathbf{x}\in\mathbf{K}} \lambda \quad \text{s.t.} \quad f(\mathbf{x}) - \lambda \geq 0 \Big\}. \tag{3.10}$$

In the SOS case, the SOS bound of order d is defined as the largest scalar λ for which the polynomial $f(\mathbf{x}) - \lambda$ lies in the truncated quadratic module of degree $2d$. This means that we replace (3.10) and we approximate the solution of (3.2) by the solution of the following convex optimization problem:

$$\begin{aligned} f_{\mathrm{SOS},d} = \sup_{\lambda,\mathbf{x}\in\mathbf{K}} \ & \lambda \\ \text{s.t.} \quad & \lambda \in \mathbb{R} \\ & f(\mathbf{x}) - \lambda \in \mathcal{Q}(\mathrm{g})_d \,. \end{aligned} \tag{3.11}$$

As discussed before, SOS can be modeled using SDP, so each SOS bound of order d can be computed via SDP. From [48, Theorem 4.2], we know that as the order d grows, we get bounds that are better and better and they converge asymptotically to the f^*. Thus, $f_{\mathrm{SOS},d} \leq f_{\mathrm{SOS},d+1} \leq f^*$ and if $\mathbf{K}$ compact $\lim_{d\to\infty} f_{\mathrm{SOS},d} = f^*$. The reason for this asymptotic convergence can be directly derived from the Positivstellensatz of Putinar, Theorem 3.2.6. The SOS polynomials $(\sigma_j) \subset \Sigma[\mathbf{x}]$ give a Putinar certificate of nonnegativity for the polynomial f on $\mathbf{K}$.

The convex cones $\mathcal{Q}_\ell$ are tractable since they consist of combinations of SOS cones $\Sigma[\mathbf{x}]_{2d}$. Problem (3.11) is also considered tractable since it can be formulated as a semidefinite program, which can be efficiently solved using standard convex optimization methods, such as interior point methods [18, 74].

Let $\mathrm{v}_d(\mathbf{x})_j$ denote a basis of $\mathbb{R}[\mathbf{x}]_d$. Then (3.11) is implemented as a semidefinite program of the form

$$\begin{aligned} f_{\mathrm{SOS},d} = \sup\ & \lambda \\ \text{s.t.}\quad & \lambda \in \mathbb{R},\ A_j \succcurlyeq 0, \\ & f(\mathbf{x}) - \lambda - \sum_{j=1}^{m} g_j(\mathbf{x}) \mathrm{v}_d(\mathbf{x})_j^T A_j \mathrm{v}_d(\mathbf{x})_j = 0\,, \end{aligned}$$

where $A_j \in \mathbb{R}^{s(d)\times s(d)}$ is positive semidefinite i.e. $A_j \succcurlyeq 0$ and it is shown in Proposition 3.2.3. The last polynomial constraint can be written as

$$\sum_{\alpha\in\mathbb{N}^n_{2d}} c_\alpha(\lambda, A_1, \ldots, A_m)\mathbf{x}^\alpha = 0$$

where $(\mathbf{x}^\alpha)_{\alpha\in\mathbb{N}^n_{2d}}$ is the monomial basis of $\mathbb{R}[\mathbf{x}]_d$. It corresponds to a sequence of coefficient constraints $c_\alpha(\lambda, \mathbf{A}) = 0$, which are linear in $\lambda,\ \mathbf{A} = (A_1, \ldots, A_m)$.

We verify that $\mathcal{Q}_d \subset \mathcal{Q}_{d+1}$ so that $(\mathcal{Q}_d)$ is a hierarchy of nested convex finite dimensional cones.

So far, we have discussed how to use SOS to design a hierarchy of lower bounds (SOS bounds) of growing orders. We have seen that the hierarchy of bounds converges to the minimum of f on the semialgebraic set $\mathbf{K}$. This SOS approach is useful because it gives certificates for the nonnegativity of the polynomial. In the next section, we describe the dual approach (Moment approach) which gives us additional complementary information, not only about minimum value of f, but also about global minimizers of f on $\mathbf{K}$.

3.2.2 Moment Approach

As defined above for all $\mathbf{x} \in \mathbb{R}^n, \alpha \in \mathbb{N}^n$, $\mathbf{x}^\alpha$ is a monomial $\mathrm{x}_1^{\alpha_1} \cdot \mathrm{x}_2^{\alpha_2} \cdot \cdots \cdot \mathrm{x}_n^{\alpha_n}$ in $\mathbb{R}[\mathbf{x}]$. A *Borel measure* on $\mathbb{R}^n$ refers to a set function on the $\sigma-$algebra formed by the open sets of $\mathbb{R}^n$ and is

also known as the Borel σ-field $\mathcal{B}$ of $\mathbb{R}^n$ that is nonnegative and satisfies countable additivity [46]. Throughout this section, we consider the Borel measure on $\mathbb{R}^n$. So, when we say 'a measure' on $\mathbb{R}^n$, we mean a (positive) Borel measure on $\mathbb{R}^n$. *Support* of the measure μ on $\mathbb{R}^n$ is the smallest closed set $S \in \mathbb{R}^n$ for which $\mu(\mathbb{R}^n \backslash S) = 0$ and is shown as supp(μ). Thus, μ is a measure on $\mathbf{K}$ or a measure supported by $\mathbf{K} \subseteq \mathbb{R}^n$ if supp(μ)$\subseteq \mathbf{K}$.

For $\mathbf{x} \in \mathbb{R}^n$, $\delta_{\mathbf{x}}$ denotes the *Dirac measure* at $\mathbf{x}$, with support $\{\mathbf{x}\}$ and mass 1 at $\mathbf{x}$ and mass 0 elsewhere.

When the supp$(\mu) = \{\mathrm{x}_1, \ldots, \mathrm{x}_r\}$ and it is finite, then for some $\lambda_1, \ldots, \lambda_r > 0$, $\mu = \sum_{i=1}^{r} \lambda_i \delta_{\mathrm{x}_i}$. These x_i are called the *atoms* of μ and μ is called *r-atomic* measure.

Definition 3.2.7. *([46, Definition 2.31]) **The full moment problem:** Let $(\mathrm{g}_i)_{i=1}^m \subset \mathbb{R}[\mathbf{x}]$, and $\mathbf{K} \subset \mathbb{R}^n$ be the set given in* (3.3)*. An infinite sequence of real numbers $\mathrm{y} = (y_\alpha) \subset \mathbb{R}, \alpha \in \mathbb{N}^n$, is a $\mathbf{K}$-moment sequence whether there exists a finite Borel measure μ supported on $\mathbf{K}$, such that*

$$y_\alpha = \int_K \mathbf{x}^\alpha d\mu, \qquad \forall \alpha \in \mathbb{N}^n. \tag{3.12}$$

In other words, y *is called the sequence of moments of the measure μ.*

***The truncated moment problem:** Given $(\mathrm{g}_i)_{i=1}^m, \mathbf{K} \subset \mathbb{R}^n$ as in* (3.3)*, a finite set $\Delta \subset \mathbb{N}^n$, and a finite sequence of real numbers $\mathrm{y} = (y_\alpha), \alpha \in \Delta$,* y *is a $\mathbf{K}$-moment whether there exists a measure μ supported on $\mathbf{K}$, such that*

$$y_\alpha = \int_{\mathbf{K}} \mathbf{x}^\alpha d\mu, \qquad \forall \alpha \in \Delta. \tag{3.13}$$

Definition 3.2.8. *In both full and truncated cases, a measure μ as in* (3.12) *and* (3.13) *is said to be a **representing measure** of the sequence* y.

Definition 3.2.9. ***The Riesz linear functional:** The Riesz linear functional $L_{\mathrm{y}} : \mathbb{R}[\mathbf{x}] \to \mathbb{R}$ for a real sequence $\mathrm{y} = (y_\alpha) \subset \mathbb{R}$, constructed by:*

$$f\big(= \sum_{\alpha \in \mathbb{N}^n} f_\alpha \mathbf{x}^\alpha\big) \mapsto L_{\mathrm{y}}(f) = \sum_{\alpha \in \mathbb{N}^n} f_\alpha y_\alpha, \qquad \forall f \in \mathbb{R}[\mathbf{x}], \tag{3.14}$$

also

$$\mathbf{x}^\alpha \mapsto L_{\mathrm{y}}(\mathbf{x}^\alpha) := y_\alpha.$$

The goal of the polynomial optimization (3.2) is to find $\mathbf{x}$ in the semi-algebraic set $\mathbf{K}$ with the smallest f value possible. Every point $\mathbf{x} \in \mathbf{K}$ can be viewed as a measure and it can be modeled by a Dirac measure on $\mathbf{K}$. Since we are minimizing over $\mathbf{x}$, instead of minimizing over the Dirac measure at $\mathbf{x}$, we can minimize over the convex combination of Dirac measures at points in $\mathbf{K}$, which results in minimizing over probability measures on $\mathbf{K}$. As a result, we can reformulate our polynomial optimization problem to a problem of finding μ such that the expected value of f with respect to this measure is as small as possible. Then, if we have μ, we can define

$$\begin{aligned} f_{\min} = \inf_{\mathbf{x}\in\mathbf{K}} f(\mathbf{x}) &= \inf_{\mu} \int_{\mathbf{K}} f(\mathbf{x}) d\mu \quad \text{s.t. } \mu \text{ is a probability measure on } \mathbf{K} \\ &= \inf\, L_{\mathrm{y}}(f) \quad \text{s.t. } L \text{ has a representing measure } \mu \text{ on } \mathbf{K}. \end{aligned}$$

Definition 3.2.10. *Consider* $\mathrm{y} = (y_\alpha)$ *of dimension* $s(2d)$. *The* ***moment matrix*** *is a real symmetric matrix* $\mathrm{M}_d(\mathrm{y})$ *of dimension* $s(d)$ *and defined as*

$$\mathrm{M}_d(\mathrm{y})(\alpha, \beta) = L_{\mathrm{y}}(\mathbf{x}^\alpha \mathbf{x}^\beta) = y_{\alpha+\beta}, \quad \forall \alpha, \beta \in \mathbb{N}^n_d \tag{3.15}$$

in which rows and columns indexed by $\alpha \in \mathbb{N}^n_d$. *Equivalently,* $\mathrm{M}_d(\mathrm{y}) = L_{\mathrm{y}}(\mathrm{v}_d(\mathbf{x})\mathrm{v}_d(\mathbf{x})^T)$. *That is, we apply* L_{y} *to each entry of the matrix* $\mathrm{v}_d(\mathbf{x})\mathrm{v}_d(\mathbf{x})^T$.

Example 3.2.11. *The moment matrix* $\mathrm{M}_1(\mathrm{y})$ *which is for* $n = d = 1$ *is a Hankel matrix:*

$$\begin{pmatrix} 1 & \mathrm{x} & \mathrm{x}^2 & \mathrm{x}^3 \\ \mathrm{x} & \mathrm{x}^2 & \mathrm{x}^3 & \mathrm{x}^4 \\ \mathrm{x}^2 & \mathrm{x}^3 & \mathrm{x}^4 & \mathrm{x}^5 \\ \mathrm{x}^3 & \mathrm{x}^4 & \mathrm{x}^5 & \mathrm{x}^6 \end{pmatrix} \to \mathrm{M}_3(\mathrm{y}) = \begin{pmatrix} y_0 & y_1 & y_2 & y_3 \\ y_1 & y_2 & y_3 & y_4 \\ y_2 & y_3 & y_4 & y_5 \\ y_3 & y_4 & y_5 & y_6 \end{pmatrix}$$

For a two-dimensional case let us consider an example with $n = d = 2$. *In this case,* $\mathrm{M}_2(\mathrm{y})$ *is of Hankel type and we have*

$$\mathrm{M}_2(\mathrm{y}) = \begin{array}{c} \\ 1 \\ \mathrm{x}_1 \\ \mathrm{x}_2 \\ \\ \mathrm{x}_1^2 \\ \mathrm{x}_1\mathrm{x}_2 \\ \mathrm{x}_2^2 \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & & \mathrm{x}_1 & \mathrm{x}_2 & & \mathrm{x}_1^2 & \mathrm{x}_1\mathrm{x}_2 & \mathrm{x}_2^2 \end{array} \\ \begin{pmatrix} y_{00} & | & y_{10} & y_{01} & | & y_{20} & y_{11} & y_{02} \\ & - & - & - & - & - & - & - \\ y_{10} & | & y_{20} & y_{11} & | & y_{30} & y_{21} & y_{12} \\ y_{01} & | & y_{11} & y_{02} & | & y_{21} & y_{12} & y_{03} \\ & - & - & - & - & - & - & - \\ y_{20} & | & y_{30} & y_{21} & | & y_{40} & y_{31} & y_{22} \\ y_{11} & | & y_{21} & y_{12} & | & y_{31} & y_{22} & y_{13} \\ y_{02} & | & y_{12} & y_{03} & | & y_{22} & y_{13} & y_{04} \end{pmatrix} \end{array}$$

Definition 3.2.12. *Let us consider polynomial* $\mathrm{g} \in \mathbb{R}[\mathbf{x}]$ *with coefficient vector* $\mathrm{g} = (\mathrm{g}_\gamma)$. *The matrix* $\mathrm{M}_d(\mathrm{gy})$ *is called* ***localizing matrix*** *with respect to* y *and* g *and defined by*

$$\mathrm{M}_d(\mathrm{gy})(\alpha, \beta) = L_\mathrm{y}(\mathrm{g}(\mathbf{x})\mathbf{x}^\alpha\mathbf{x}^\beta) = \sum_{\gamma \in \mathbb{N}^n} \mathrm{g}_\gamma y_{\alpha+\beta+\gamma}, \quad \forall \alpha, \beta \in \mathbb{N}^n_d \tag{3.16}$$

in which rows and columns are indexed by $\alpha \in \mathbb{N}^n_d$. *The localizing matrix localizes the support of a representing measure for* y. *Equivalently,* $\mathrm{M}_d(\mathrm{gy}) = L_\mathrm{y}(\mathrm{gv}_d(\mathbf{x})\mathrm{v}_d(\mathbf{x})^T)$. *That is, we apply* L_y *to each entry of the matrix* $\mathrm{g}(\mathbf{x})\mathrm{v}_d(\mathbf{x})\mathrm{v}_d(\mathbf{x})^T$. *If* $r = \deg(\mathrm{g})$, $\mathrm{M}_d(\mathrm{gy}) = \sum_{\alpha \in \mathbb{N}^n_{2d+r}} \mathrm{C}_\alpha y_\alpha$ *where* $\mathrm{C}_\alpha, \alpha \in \mathbb{N}^n_{2d+r}$ *are real symmetric matrices and defined by*

$$\mathbf{x} \mapsto \mathrm{g}(\mathbf{x})\mathrm{v}_d(\mathbf{x})\mathrm{v}_d(\mathbf{x})^T = \sum_{\alpha \in \mathbb{N}^n_{2d+r}} \mathrm{C}_\alpha \mathbf{x}^\alpha, \quad \mathbf{x} \in \mathbb{R}^n.$$

The moment matrix $\mathrm{M}_d(\mathrm{y})$ defines a bilinear form on $\mathbb{R}[\mathbf{x}]_\mathbf{d}$

$$\langle p, q \rangle_\mathrm{y} := L_\mathrm{y}(pq) = \langle \mathbf{p}, \mathrm{M}_\mathbf{d}(\mathrm{y})\mathbf{q} \rangle = \mathbf{p}^\top \mathrm{M}_\mathbf{d}(\mathrm{y})\mathbf{q}$$

in which $p, q \in \mathbb{R}[\mathbf{x}]_\mathbf{d}$. Here, $\mathbf{p}, \mathbf{q} \in \mathbb{R}^{\mathbf{s}(\mathbf{d})}$ represent the coefficient vectors of this p and q respec-

tively. Also, if y has a representing measure μ then for every $p \in \mathbb{R}[\mathbf{x}]$,

$$\langle \mathbf{p}, \mathrm{M}_d(\mathrm{y})\mathbf{p} \rangle = L_{\mathrm{y}}(p^2) = \int_{\mathbb{R}^n} p^2 d\mu \geq 0.$$

Similarly to this, for all $j \in J$,

$$\langle \mathbf{p}, \mathrm{M}_d(\mathrm{g_j y})\mathbf{p} \rangle = L_{\mathrm{y}}(\mathrm{g}_j p^2) = \int_{\mathbb{R}^n} \mathrm{g}_j p^2 d\mu.$$

From these equalities, one can see that

$$\Big[L_{\mathrm{y}}(p^2) \geq 0, \quad \forall p \in \mathbb{R}[\mathbf{x}] \Big] \Longleftrightarrow \Big[\mathrm{M}_d(\mathrm{y}) \succeq 0 \Big],$$

and, for all $j \in J$,

$$\Big[L_{\mathrm{y}}(\mathrm{g}_j p^2) \geq 0, \quad \forall p \in \mathbb{R}[\mathbf{x}] \Big] \Longleftrightarrow \Big[\mathrm{M}_d(\mathrm{g_j y}) \succeq 0 \Big].$$

A Hierarchy of the Moment Relaxation

In this part, we explain the Lasserre hierarchy which is a hierarchy of semidefinite relaxations. The optimum obtained from these relaxations forms a sequence of lower bounds which is monotone and non-decreasing, and it converges to the global optimum f^*. Consider the polynomial optimization problem (3.2) with the feasible set $\mathbf{K}$ (3.3). We assume that the set $\mathbf{K}$ is a compact basic semi-algebraic subset of $\mathbb{R}^n$.

To approximate the polynomial problem (3.2), Lasserre [48] proposed the following technique. First, we have:

$$f^* = \inf_{\mathbf{x} \in \mathbf{K}} f(\mathbf{x}) = \inf_{\mu} \int_{\mathbf{K}} f(\mathbf{x}) d\mu.$$

Here is where the second infimum comes from. For any $\mathbf{x}_0 \in \mathbf{K}$, $p(\mathbf{x}_0) = \inf_\mu \int_{\mathbf{K}} f(\mathbf{x}) d\mu$ for the Dirac measure $\mu := \delta_{\mathbf{x}_0}$, we see that $f^* \geq \inf_\mu \int_{\mathbf{K}} f(\mathbf{x}) d\mu$. On the other hand, for all $\mathbf{x} \in \mathbf{K}$, $f(\mathbf{x}) \geq f^*$ and since μ is the probability measure $\int_{\mathbf{K}} f(\mathbf{x}) d\mu \geq \int_{\mathbf{K}} f^* d\mu = f^*$.

Now, we can reformulate this problem again. We can see that $\int f(\mathbf{x}) d\mu = \sum \alpha f_\alpha \int f(\mathbf{x}_0) d\mu =$

$f^\top \mathrm{y}$. So,

$$f^* = \inf_{\mathrm{y}} \ f^\top \mathrm{y} \quad \text{s.t. } \mathrm{y}_0 = 1, \ \mathrm{y} \text{ has a representing measure on } \mathbf{K}. \tag{3.17}$$

Consider the feasible set $\mathbf{K}$ (3.3) with an additional constraint $\mathrm{g}_0 = 1$, and $v_j := [(\deg(\mathrm{g}_j))/2], j = 0, \ldots, m$. We can replace the constraint in (3.17) with the weaker conditions $\mathrm{M}_d(\mathrm{y}) \succeq 0$ and $\mathrm{M}_{d-v_j}(\mathrm{g}_j \mathrm{y}) \succeq 0, \forall j$, and for $d \geq d_0$ in which $d_0 := \max[[(\deg(f))/2], \max_{j=1,\ldots,m} v_j]$, get the following SDP

$$\begin{aligned} \rho_d^{\text{mom}} = \inf_{\mathrm{y}} \ & L_{\mathrm{y}}(f) \\ \text{s.t. } & \mathrm{M}_d(\mathrm{y}) \succeq 0 \\ & \mathrm{M}_{d-v_j}(\mathrm{g}_j \mathrm{y}) \succeq 0, \ j = 1, \ldots, m \\ & \mathrm{y}_0 = 1. \end{aligned} \tag{3.18}$$

Here $\mathrm{M}_d(\mathrm{y})$ and $\mathrm{M}_{d-v_j}(\mathrm{g}_j \mathrm{y})$ are the moment matrix and localizing matrix, respectively, associated with y and the polynomial g_j, and L_{y} is the Riesz linear function as described before.

This SDP is a relaxation of (3.2) and by changing the parameter d, it defines a hierarchy of semidefinite relaxation of that. Here is the reason for it being a hierarchy: let $\mathbf{x} \in \mathbf{K}$ be a feasible solution and $\mathrm{y} := \mathrm{v}_{2d}(\mathbf{x})$, which is equal to $\mathrm{x}^\alpha, \alpha \in \mathbb{N}^n_{2d}$. The vector y is the vector of moments of the Dirac measure $\mu := \delta_{\mathbf{x}}$ at $\mathbf{x} \in \mathbf{K}$ of order up to $2d$. Thus, y is a feasible solution of (3.18) with objective value of $L_{\mathrm{y}}(f) = f(\mathbf{x})$. This means $\rho_d^{\text{mom}} \leq f^*$, for all d. In addition, as the SDP (3.18) with $d+1$ includes more constraints than with d, for all $d \geq d_0$, $\rho_{d+1}^{\text{mom}} \geq \rho_d^{\text{mom}}$.

By considering the moment and localizing matrices in (3.18) for some symmetric real matrices $C^j_\alpha, j = 1, \ldots, m$, and $\alpha \in \mathbb{N}^n_{2d}$ to be

$$M_d(\mathrm{y}) = \sum_{|\alpha| \leq 2d} y_\alpha C^0_\alpha, \quad \mathrm{M}_{d-v_j}(\mathrm{g}_j \mathrm{y}) = \sum_{|\alpha| \leq 2d} y_\alpha C^j_\alpha, \quad j = 1, \ldots, m,$$

the dual of (3.18) will be written as the SDP problem

$$\begin{aligned} \rho_d^{\text{sos}} = \sup_{\lambda, X_j} \quad & \lambda \\ \text{s.t.} \quad & f_\alpha - \lambda 1_{\alpha=0} = \sum_{j=0}^{m} \langle X_j, C_\alpha^j \rangle, \quad \forall \alpha \in \mathbb{N}_{2d}^n \\ & X_j \succeq 0, \quad j = 0, \ldots, m, \end{aligned} \tag{3.19}$$

where $X_j,\ j = 0, \ldots, m$, are all real symmetric matrices. From these problems and the weak duality, we can obviously see that $\rho_d^{\text{sos}} \leq \rho_d^{\text{mom}} \leq \rho_{d+1}^{\text{mom}} \leq f^*$.

The compact form of this dual is the SDP

$$\begin{aligned} \rho_d^{\text{sos}} = \sup_{\lambda, \{\sigma_j\}} \quad & \lambda \\ \text{s.t.} \quad & f - \lambda = \sum_{j=0}^{m} \sigma_j \mathrm{g}_j, \quad \sigma_j \in \Sigma[\mathbf{x}]_{d-v_j}, \quad j = 0, \ldots, m. \end{aligned} \tag{3.20}$$

Here is the explanation of where this compact form comes from: multiply each side of the equality constraint in (3.19) by $\mathbf{x}^\alpha$ and sum up to obtain

$$\sum_{\alpha \in \mathbb{N}_{2d}^n} f_\alpha \mathbf{x}^\alpha - \lambda = \sum_{j=0}^{m} \Big\langle X_j, \sum_{\alpha} C_\alpha^j \mathbf{x}^\alpha \Big\rangle, \quad \forall \mathbf{x} \in \mathbb{R}^n,$$

This is equal to

$$\begin{aligned} f(\mathbf{x}) - \lambda &= \sum_{j=0}^{m} \mathrm{g}_j(\mathbf{x}) \Big\langle X_j, \mathrm{v}_{d-v_j}(\mathbf{x}) \mathrm{v}_{d-v_j}(\mathbf{x})^T \Big\rangle \\ &= \sum_{j=0}^{m} \mathrm{g}_j(\mathbf{x}) \mathrm{v}_{d-v_j}(\mathbf{x}) X_j \mathrm{v}_{d-v_j}(\mathbf{x})^T, \quad \forall \mathbf{x} \in \mathbb{R}^n. \end{aligned}$$

By writing $X_j \succeq 0$ as $\sum_{l=1}^{t_j} q_{jl} q_{jl}^T$, for every $j = 0, \ldots, m$ and some $t_j \in \mathbb{N}$, we have

$$f(\mathbf{x}) - \lambda = \sum_{j=0}^{m} \mathrm{g}_j(\mathbf{x}) \left(\sum_{l=1}^{t_j} \underbrace{(q_{jl}^T \mathrm{v}_{d-v_j}(\mathbf{x}))^2}_{\mathrm{q}_{jl}(\mathbf{x})^2} \right), \quad \forall \mathbf{x} \in \mathbb{R}^n,$$

where $\mathrm{q}_{jl} \in \mathbb{R}[\mathbf{x}]_{d-v_j}$ has a coefficient vector $q_{jl} \in \mathbb{R}^{s(d-v_j)}$, for all $l = 1, \ldots, t_j$, and all $j = 0, \ldots, m$. By considering the SOS polynomial $\sum_{l=1}^{t_j} \mathrm{q}_{jl}^2$ to be $\sigma_j \in \Sigma[\mathbf{x}]_{d-v_j}$, for every $j = 0, \ldots, m$, equivalently to the last equation we get

$$f - \lambda = \sum_{j=0}^{m} \sigma_j \mathrm{g}_j.$$

This demonstrates that the dual of (3.18) is a semidefinite program that reads (3.20) in compact form. Algorithm 2 [46] describes the general procedure for locating the global optimum of the polynomial optimization problem utilizing the Lasserre hierarchy. We demonstrate the convergence of this approach in the theorem below.

Theorem 3.2.13. *([46, Theorem 6.2].) Let Assumption 3.2.1 hold and consider the semidefinite relaxation* (3.18) *with optimal value ρ_d^{mom}. We have the following convergence result:*

- *$\rho_d^{mom} \uparrow f^*$ as $d \to \infty$.*
- *Assume $\mathrm{x}^* \in K$ to be the unique global minimizer of* (3.2) *and y^d be a nearly optimal solution of* (3.18) *with value $L_{\mathrm{y}}(f) \leq \rho_d^{mom} + 1/d$. Then, for every $j = 1, \ldots, m$, as $d \to \infty$, $L_{\mathrm{y}^d}(\mathrm{x}_j) \to \mathrm{x}_j^*$.*

It is worthwhile to mention that there is a duality between sums of squares and moment sequences. We consider the cones

$$\mathcal{M} := \{\mathrm{y} \mid \mathrm{y} \text{ has a representing measure on } \mathbf{K}\}$$

and

$$\mathcal{M}_{\succeq} := \{\mathrm{y} \mid M(\mathrm{y}) \succeq 0\},$$

and $\mathcal{P}, \Sigma \in \mathbb{R}[\mathbf{x}]$ be the cone of nonnegative polynomials and cone of polynomials that are sums of squares, respectively with their dual cones

$$\mathcal{P}^* = \{L \in \mathbb{R}[\mathbf{x}]^* \mid L(p) \geq 0,\ \forall p \in \mathcal{M}\},$$

Algorithm 2 Constrained Polynomial Optimization [46]

- **Input** A polynomial $\mathbf{x} \mapsto f(\mathbf{x})$. A set $\mathbf{K} = \{\mathbf{x} \in \mathbb{R}^n : \mathrm{g}_j(\mathbf{x}) \geq 0, j = 1, \ldots, m\}$ in which g_j are polynomials of degree $2v_j$ or $2v_j - 1$ for $j = 1, \ldots, m$. Let k be the order of the highest relaxation.
- **Output** The value $f^* = \inf_{\mathbf{x}\in\mathbf{K}} f(\mathbf{x})$ as well as a list of global minimizers or a lower bound ρ_k on f^*.

(i) Solve the SDP problem (3.18) with the optimal value ρ_d^{mom} and optimal solution y^* if it exists.

(ii) If there is no y^*, ρ_d^{mom} will be only the lower bound of f^* ($\rho_d^{\text{mom}} \leq f^*$). If $d < k$, $d \to d + 1$ and go back to the step (i); otherwise finish and return ρ_k

(iii) If rank $\mathrm{M}_{d-v}(\mathrm{y}^*) =$ rank $\mathrm{M}_d(\mathrm{y}^*)$ with $v := \max_j v_j$, then $\rho_d^{\text{mom}} = f^*$ and there are at least $r =$ rank $\mathrm{M}_d(\mathrm{y}^*)$ global minimizers.

(iv) If rank $\mathrm{M}_{d-v}(\mathrm{y}^*) \neq$ rank $\mathrm{M}_d(\mathrm{y}^*)$ and $d < k$, $d \to d+1$ and go back to the step (i); otherwise finish and return ρ_k which providing the lower bound of f^*.

and

$$\Sigma^* = \{L \in \mathbb{R}[\mathbf{x}]^* \mid L(p^2) \geq 0,\ \forall p \in \mathbb{R}[\mathbf{x}]\}.$$

Proposition 3.2.14. *([52, Proposition 4.9]) The cones $\mathcal{M}$ and $\mathcal{P}$ are duals of each other. Which means that $\mathcal{P} = \mathcal{M}^*$, $\mathcal{M} = \mathcal{P}^*$. Also, the same duality holds for $\mathcal{M}_{\succeq}$ and Σ and $\mathcal{M}_{\succeq} = \Sigma^*$, $\Sigma = (\mathcal{M}_{\succeq})^*$*

Finite convergence

In the feasible set $\mathbf{K}$ specified in (3.3), an equality constraint $\mathrm{g}_j(\mathbf{x}) = 0$ is treated as two inequality constraints. These constraints are $\mathrm{g}_j(\mathbf{x}) \geq 0$ and $-\mathrm{g}_j(\mathbf{x}) \geq 0$. Thus, for those equality constraints the semidefinite constraint $\mathrm{M}_{d-v_j}(\mathrm{g}_j\mathrm{y}) \succeq 0$ in (3.18) becomes the equality constraint $\mathrm{M}_{d-v_j}(\mathrm{g}_j\mathrm{y}) = 0$. Finite convergence of the hierarchy of semidefinite relaxation (3.18) occurs under some nonlinear programming-related conditions. We must make a distinction between equality constraints and inequality constraints in order to describe such conditions. As a result, the basic semi-algebraic set $\mathbf{K} \subset \mathbb{R}^n$ will have the general form represented as in the following:

$$\mathbf{K} := \{\mathbf{x} \in \mathbb{R}^n : \mathrm{g}_j(\mathbf{x}) \geq 0,\ j = 1, \ldots, m_1; h_l(\mathbf{x}) = 0,\ l = 1, \ldots, m_2\}, \tag{3.21}$$

where m_1, m_2 are integers and $\mathrm{g}_j, h_l \in \mathbb{R}[\mathbf{x}]$ are some polynomials. Lets $\tau_l = [(\deg h_l)/2],\ l = 1, \ldots, m_2$. Now, we reformulate the moment semidefinite relaxation (3.18) as this problem

$$\begin{aligned} \rho_d^{\text{mom}} = \inf_{\mathrm{y}} \quad & L_{\mathrm{y}}(f) \\ \text{s.t.} \quad & \mathrm{M}_d(\mathrm{y}) \succeq 0 \\ & \mathrm{M}_{d-v_j}(\mathrm{g}_j\mathrm{y}) \succeq 0,\ j = 1, \ldots, m_1 \\ & \mathrm{M}_{d-\tau_l}(h_l\mathrm{y}) = 0,\ j = 1, \ldots, m_2 \\ & y_0 = 1 \end{aligned} \tag{3.22}$$

and its dual, the SOS relaxation

$$\begin{aligned} \rho_d^{\text{sos}} = \sup_{\lambda,\{\sigma_j\}} \quad & \lambda \\ \text{s.t.} \quad & f - \lambda = \sum_{j=0}^{m_1} \sigma_j \mathrm{g}_j + \sum_{l=1}^{m_2} \psi_l h_l, \\ & \sigma_j \in \Sigma[\mathbf{x}]_{d-v_j}, \quad j = 1, \ldots, m_1, \\ & \psi_l \in \mathbb{R}[\mathbf{x}]_{d-\tau_l}, \quad l = 1, \ldots, m_2. \end{aligned} \tag{3.23}$$

The semidefinite relaxation (3.22) is exact when $\rho_d^{\text{mom}} = f^*$. Consider the set $J(\mathbf{x}^*)$, which includes the inequality constraints active at $\mathbf{x}^*$, that is, $J(\mathbf{x}^*) := \{j : \mathrm{g}_j(\mathbf{x}^*) = 0\}$.

Definition 3.2.15. *[5] A vector $(\mathbf{x}^*, \lambda^*, \mathrm{v}^*) \in K \times \mathbb{R}^{m_1} \times \mathbb{R}^{m_2}$ is called **Karush-Kuhn-Tucker (KKT) vector** if it satisfies*

$$\begin{aligned} & \nabla f(\mathbf{x}^*) = \sum_{j=1}^{m_1} \lambda_j^* \nabla \mathrm{g}_j(\mathbf{x}^*) + \sum_{l=1}^{m_2} v_l^* \nabla h_l(\mathbf{x}^*), \\ & \lambda_j^* \mathrm{g}_j(\mathbf{x}^*) = 0, \quad j = 1, \ldots, m_1 \quad [complementarity] \\ & \lambda_j^* \geq 0, \quad j = 1, \ldots, m_1. \end{aligned} \tag{3.24}$$

*For the KKT vector $(\mathbf{x}^*, \lambda^*, \mathrm{v}^*)$ we define the **Lagrangian** function $L(\cdot, \lambda^*, \mathrm{v}^*) : \mathbb{R}^n \to \mathbb{R}$ as the polynomial*

$$\mathbf{x} \mapsto \quad L(\mathbf{x}, \lambda^*, \mathrm{v}^*) := f(x) - f^* - \sum_{j=1}^{m_1} \lambda_j^* \mathrm{g}_j(\mathbf{x}) + \sum_{l=1}^{m_2} v_l^* h_l(\mathbf{x}).$$

Such a KKT vector exists, if $\mathbf{x}^*$ is a local minimizer and the gradients $(\nabla \mathrm{g}_j(\mathbf{x}^*), \nabla h_l(\mathbf{x}^*))$ are linearly independent for $j \in J(\mathbf{x}^*), l = 1, \ldots, m_2$ [5, Theorem 4.3.7]. In addition, strict complementarity holds if $\lambda_j^* > 0$ for all $j \in J(\mathbf{x}^*)$. When we define $\mathbf{K}$ in its general form (3.21), Assumption 3.2.1 reads as follows.

Assumption 3.2.2. *There exists $M > 0$ such that the quadratic polynomial $\mathbf{x} \mapsto M - \|\mathbf{x}\|^2$ can be written as*

$$M - \|\mathbf{x}\|^2 = \sum_{j=1}^{m_1} \sigma_j \mathrm{g}_j + \sum_{l=1}^{m_2} \psi_l h_l, \tag{3.25}$$

for some $\psi_l \in \mathbb{R}[\mathbf{x}]$ *and some SOS polynomials* $\sigma_j \in \Sigma[\mathbf{x}]$.

The last theorem demonstrates the finite convergence of this hierarchy relaxation.

Theorem 3.2.16. *([46, Theorem 6.5].) Let* $\mathbf{K}$ *be as in* (3.21) *and Assumption 3.2.2 hold. Assume that, at every global minimizer* $\mathbf{x}^* \in \mathbf{K}$ *of problem* (3.2) *with* $f^* = f(\mathbf{x}^*)$ *the following holds:*

- *The gradients* $(\nabla \mathrm{g}_j(\mathbf{x}^*), \nabla h_l(\mathbf{x}^*))$ *are linearly independent for* $j \in J(\mathbf{x}^*),\ l = 1, \ldots, m_2$*, and as a result there exists a KKT vector* $(\mathbf{x}^*, \lambda^*, \mathrm{v}^*)$*,*
- *strict complementarity holds,*
- *second-order sufficiency holds. That is, for every* $u \neq 0$, $\mathrm{u}^T \nabla^2_{\mathbf{xx}} L(\mathbf{x}^*, \lambda^*, \mathrm{v}^*)\mathrm{u} > 0$ *holds, such that for* $j \in J(\mathbf{x}^*)$ *with* $\lambda_j > 0$ *and* $l = 1, \ldots, m_2$*, we have* $\nabla \mathrm{g}_j(\mathbf{x}^*)^T u = 0$ *and* $\nabla h_l(\mathbf{x}^*)^T u = 0$*.*

Then finite convergence occurs, i.e., $\rho_d^{mom} = f^*$ *, for some* $d \in \mathbb{N}$ *and* (3.22) *has an optimal solution.*
If in addition the ideal $\langle h \rangle = \langle h_1, \ldots, h_{m_2} \rangle$ *is real then there exist* $d \in \mathbb{N}$*, polynomials* $\psi_l \in \mathbb{R}[\mathbf{x}]_{d-\tau_l}, l = 1, \ldots, m_2$*, and SOS polynomials* $\sigma_j \in \Sigma[\mathbf{x}]_{d-v_j}, j = 0, \ldots, m_1$*, such that*

$$f - f^* = \sum_{j=1}^{m_1} \sigma_j \mathrm{g}_j + \sum_{l=1}^{m_2} \psi_l h_l, \tag{3.26}$$

i.e. $\rho_d^{sos} = f^*$ *and* (3.23) *has an optimal solution.*

The next theorem, Theorem 3.2.18, by Fialkow shows that there is a finite convergence on the order of relaxations and how we can realize the relaxation is exact. Before that, we need to define the flat extension of a matrix.

Definition 3.2.17. *([52, Definition 1.1]) Let* X *be a symmetric block matrix*

$$X = \begin{pmatrix} A & B \\ B^\top & C \end{pmatrix}. \tag{3.27}$$

If rank X=rank A, then X is called a flat extension of A. In other words, there exists a matrix W such that $B = AW$ and $C = B^\top W = W^\top AW$. When X is a flat extension of A, $X \succeq 0$ iff $A \succeq 0$.

Theorem 3.2.18. *([52, Theorem 5.33].) Let $\mathbf{K}$ be the semialgebraic set defined in* (3.3)*, $d_{\mathbf{K}} = \max_{j=1,\ldots,m} v_j$ and $r =$ rank $M_d(\mathrm{y})$ for some d. The statements that follow are equivalent*

i. $\mathrm{y} \in \mathbb{R}^{\mathbb{N}^n_{2d}}$ *has a r-atomic representing measure μ on $\mathbf{K}$.*

ii. $M_d(\mathrm{y}) \succeq 0$ *and* y *can be extended to a vector* $\tilde{\mathrm{y}} \in \mathbb{R}^{\mathbb{N}^n_{2(d+d_{\mathbf{K}})}}$ *where* $M_{d+d_{\mathbf{K}}}(\tilde{\mathrm{y}})$ *is a flat extension of* $M_d(\mathrm{y})$ *and* $M_d(\mathrm{g}_j\tilde{\mathrm{y}}) \succeq 0$ *for* $j = 1, \ldots, m$.

Moreover, for $r_j :=$ rank $M_d(\mathrm{g}_j\tilde{\mathrm{y}})$, we have that $r - r_j$ of the atoms in the support of μ belong to the set of roots of the $\mathrm{g}_j(\mathbf{x})$, and μ is a representing measure for $\tilde{\mathrm{y}}$.

The statement (i) in the last theorem means that y is a global solution and it is equivalent to (ii) which shows the existence of the flat extension for that particular d.

3.2.3 Computing the minimizers

In this section, We discuss the algorithm to extract the global minimizer $\mathbf{x}^* \in \mathbf{K}$. The following theorem shows the sufficient condition to obtain the finite convergence and compute the global minimizer.

Theorem 3.2.19. *([46, Theorem 6.6]) Consider the function f in the set of real polynomials $\mathbb{R}[\mathbf{x}]$ and assume that the optimal value of* (3.18)*, ρ_d^{mom} has happened at y^*. If*

$$\textit{rank}\, \mathrm{M}_{d-v}(\mathrm{y}^*) = \textit{rank}\, \mathrm{M}_d(\mathrm{y}^*), \quad \textit{for } v = \max_{j=1,\ldots,m} v_j, \tag{3.28}$$

then $\rho_d^{mom} = f^*$, and Algorithm 3 can be employed to extract at least r global minimizers, where r corresponds to the rank of $\mathrm{M}_d(\mathrm{y}^*)$.

If the equality (3.28) holds which also means that the semidefinite relaxation (3.18) is exact at s steps of hierarchy, one can compute rank $\mathrm{M}_d(\mathrm{y}^*)$ global minimizers of (3.2) by using the following algorithm ([46, Algorithm 6.9].

Algorithm 3 The extraction algorithm

- **Input** The moment matrix $\mathrm{M}_s(\mathrm{y}^*)$ of rank r satisfying (3.28).
- **Output** The r points $\mathbf{x}^*(i),\ i = 1, \ldots, r$, global minimizers of (3.2)

1. Compute the Cholesky factorization $\mathrm{M}_s(\mathrm{y}^*) = \mathbf{V}\mathbf{V}^\top$.
2. Reduce $\mathbf{V}$ to a column echelon form $\mathbf{U}$.
3. Compute the multiplication matrices $\mathbf{N}_i,\ i = 1, \ldots, n$ from $\mathbf{U}$.
4. For randomly generated coefficients $\lambda_i,\ i = 1, \ldots, n$ compute $\mathbf{N} := \sum_{i=1}^n \lambda_i \mathbf{N}_i$.
5. Compute the Schur decomposition $\mathbf{N} = \mathbf{Q}\mathbf{T}\mathbf{Q}^\top$.
6. Compute the matrix $Q = [q_1\ q_2\ \ldots\ q_r]$.
7. Return $\mathbf{x}_i^*(j) = q_j^\top \mathbf{N}_i q_i$, for every $i = 1, 2, \ldots, n$, and every $j = 1, \ldots, r$.

Example 3.2.20. *([46, Example 6.10]) Consider the polynomial optimization*

$$\begin{aligned} \min_{\mathbf{x}} \quad & -(\mathrm{x}_1 - 1)^2 - (\mathrm{x}_1 - \mathrm{x}_2)^2 - (\mathrm{x}_2 - 3)^2 \\ \textit{s.t.} \quad & (\mathrm{x}_1 - 1)^2 \le 1, \\ & (\mathrm{x}_1 - \mathrm{x}_2)^2 \le 1, \\ & (\mathrm{x}_2 - 3)^2 \le 1. \end{aligned}$$

For the first SDP relaxation $\rho_1^{mom} = -3$ and rank $\mathrm{M}_1(\mathrm{y}^*) = 3$, however for the second one $\rho_2^{mom} = -2$ with the same rank of the moment matrix. Thus, (3.28) *satisfies, and based on the last*

theorem $\rho_2^{mom} = f^* = -2$. *The monomial basis is* $\mathrm{v}_2(\mathbf{x}) = (1, \mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_1^2, \mathrm{x}_1\mathrm{x}_2, \mathrm{x}_2^2)^T$. *The column echelon form of the Cholesky factor of* $\mathrm{M}_2(\mathrm{y}^*)$ *reads as*

$$\mathbf{U} = \begin{bmatrix} 1 & & \\ 0 & 1 & \\ 0 & 0 & 1 \\ -2 & 3 & 0 \\ -4 & 2 & 2 \\ -6 & 0 & 5 \end{bmatrix}$$

Pivot entries associated with the generating basis $\mathrm{w}(\mathbf{x}) = (1, \mathrm{x}_1, \mathrm{x}_2)^T$, *and by* $\mathrm{v}_2(\mathbf{x}) = \mathbf{U}\mathrm{w}(\mathbf{x})$, *three global minimizers must satisfy:*

$$\begin{aligned} \mathrm{x}_1^2 &= -2 + 3\mathrm{x}_1, \\ \mathrm{x}_1\mathrm{x}_2 &= -4 + 2\mathrm{x}_1 + 2\mathrm{x}_2, \\ \mathrm{x}_2^2 &= -6 + 5\mathrm{x}_2 \end{aligned} \quad .$$

The multiplication matrices that contain the coefficients of the monomials can be extracted from the rows in $\mathbf{U}$ *as*

$$\mathbf{N}_1 = \begin{bmatrix} 0 & 1 & 0 \\ -2 & 3 & 0 \\ -4 & 2 & 2 \end{bmatrix}, \quad \mathbf{N}_2 = \begin{bmatrix} 0 & 0 & 1 \\ -4 & 2 & 2 \\ -6 & 0 & 5 \end{bmatrix}.$$

By random convex combination of these two matrices and computing the Schur decomposition in step 6 of the algorithm, we reach the three global minimizers for this problem which are:

$$\mathbf{x}^*(1) = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \mathbf{x}^*(2) = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \mathbf{x}^*(3) = \begin{bmatrix} 2 \\ 3 \end{bmatrix}.$$

Chapter Four

Theory of Loraine

In this chapter, we present the theoretical foundations of Loraine, the algorithm developed within the framework of this thesis. While Loraine is a general-purpose SDP solver, its main feature is the ability to efficiently solve SDP problems with low-rank solutions. There are several approaches for these problems, most well known by Burer and Monteiro [13, 14] which is based on the low-rank factorization of the matrix variables. These methods necessitate knowing the guaranteed upper bound on the rank of the solution and the algorithm would not find a solution if the estimated rank was lower than the actual rank. The main difference of Loraine and these methods is that if the estimated rank is smaller than the real one, it will not affect the convergence of the algorithm and it might need more iterations in the iterative solver.

The first section of this chapter is thus focused on the low-rank feature of Loraine.

4.1 Low-rank semidefinite optimization

We consider a (primal) semidefinite optimization problem with matrix variables $X_i \in \mathbb{S}^{m_i},\ i = 1, \ldots, p$, and with linear constraints

$$\begin{aligned} \min_{X_1,\ldots,X_p,\,\mathbf{x}_{\text{lin}}} \ & \sum_{i=1}^{p} C_i \bullet X_i + d^\top \mathbf{x}_{\text{lin}} \\ \text{subject to } & \sum_{i=1}^{p} A_j^{(i)} \bullet X_i + (D^\top \mathbf{x}_{\text{lin}})_j = b_j, \quad j = 1, \ldots, n \\ & X \succcurlyeq 0,\ \mathbf{x}_{\text{lin}} \geq 0 \end{aligned} \tag{4.1}$$

together with its Lagrangian dual

$$\begin{aligned} \max_{\mathbf{y},\,S_1,\ldots,S_p,\,\mathbf{s}_{\text{lin}}} \ & b^\top \mathbf{y} \\ \text{subject to } & \sum_{j=1}^{n} \mathrm{y}_j A_j^{(i)} + S_i = C_i, \quad i = 1, \ldots, p \\ & D\mathbf{y} + \mathbf{s}_{\text{lin}} = d \\ & S_i \succcurlyeq 0, \quad i = 1, \ldots, p \\ & \mathbf{s}_{\text{lin}} \geq 0\,. \end{aligned} \tag{4.2}$$

Here $A_j^{(i)} \in \mathbb{S}^{m_i}$, $j = 1, \ldots, n$, $i = 1, \ldots, p$, $C_i \in \mathbb{S}^{m_i}$, $i = 1, \ldots, p$, $b \in \mathbb{R}^n$, $D \in \mathbb{R}^{\nu \times n}$, $d \in \mathbb{R}^\nu$ are data of the problem.

4.1.1 Basic assumptions

We make the following assumptions.

Assumption 4.1.1. *There exist strictly feasible $X_i \in \mathbb{S}^{m_i}_{++}$, $\mathbf{x}_{\text{lin}} \in \mathbb{R}^\nu_{++}$, $\mathbf{y} \in \mathbb{R}^n$, $S_i \in \mathbb{S}^{m_i}_{++}$, $\mathbf{s}_{\text{lin}} \in \mathbb{R}^\nu_{++}$, $i = 1, \ldots, p$ satisfying the equality constraints in* (4.1) *and* (4.2) *(Slater's condition).*

Assumption 4.1.2. *Define the matrices $\mathbf{A}_i = [\mathrm{vec} A_1^{(i)}, \ldots, \mathrm{vec} A_n^{(i)}]$, $i = 1, \ldots, p$. We assume that matrix-vector products with $\mathbf{A}_i$ and $\mathbf{A}_i^\top$ may each be computed in $\mathcal{O}(n)$ flops and memory.*

Assumption 4.1.3. *The inverse $(D^\top D)^{-1}$ and matrix-vector product with $(D^\top D)^{-1}$ may each be computed in $\mathcal{O}(n)$ flops and memory.*

Assumption 4.1.4. *The dimensions of X_i, $i = 1, \ldots, p$, are much smaller than the number of constraints, i.e., $m_i \ll n$.*

Remark 4.1.1. *Assumption 4.1.3 is satisfied, for instance, in the case of box constraints on variable y in the dual formulation. Assumption 4.1.4 is not really needed as a "mathematical" assumption, the iterative methods presented in the paper would work without it. However, the complexity of these methods would then be comparable with standard software using Cholesky factorization. The presented methods will be only superior to the standard software under this assumption.*

4.1.2 Low-rank assumption

As mentioned in the Introduction, our approach is focused on linear systems with matrices having few large outlying eigenvalues. This, of course, includes matrices with very low rank but also matrices with "approximately" low rank. These matrices frequently appear in optimization algorithms, such as interior-point methods, when the "exact" low rank is only attained at the (unreached) optimum. Moreover, the approach allows us to solve problems with matrices of high rank but with a few large outlying eigenvalues, such as the problems in Chapter 6.

Because our approach covers SDP problems with (genuinely) low-rank solutions and for the lack of better terminology for the larger class of matrices with outlying eigenvalues, we still call our main assumption "low-rank" and the problems we are targeting "SDP problems with low-rank solutions".

Our main assumption used in the design of an efficient preconditioner concerns the eigenvalue distribution of the solution $X^* = (X_1^*, \ldots, X_p^*)$ of (4.1). We assume that X^* has a very small number of large outlying eigenvalues. This includes problems where X_i^* are of a very low rank.

Assumption 4.1.5. *Let X^* be the solution of* (4.1)*. We assume that, for $i = 1, \ldots, p$, X_i^* has k outlying eigenvalues, i.e., that*

$$(0 \leq)\ \lambda(X_i^*)_1 \leq \cdots \leq \lambda(X_i^*)_{m_i-k} \ll \lambda(X_i^*)_{m_i-k+1} \leq \cdots \leq \lambda(X_i^*)_{m_i}\,,$$

where k is very small, typically smaller than 10 and, often, equal to 1.

Remark 4.1.2. *In order to emphasize the difference between our algorithm and other "low-rank algorithms", such as SDPLR [13] and the low-rank preconditioner introduced in [86]—that otherwise served as our motivation—we include the following remarks.*

1. *We do not assume anything about the distribution of the first $(m_i - k)$ eigenvalues of X_i^*; they may not be equal to zero, they may not be clustered.*
2. *We make no assumption about the "rank" of the solution $\mathbf{x}_{\text{lin}}^*$ of* (4.1)*, i.e., about the number of active constraints in $D\mathbf{y} \leq d$ in* (4.2)*. This can be arbitrarily large or small.*
3. *We do not assume that the matrix $\mathbf{A}^\top \mathbf{A}$ is easily invertible. Indeed, it is* not *in our application in Chapter 5, contrary to examples in [86] where this matrix is diagonal.*

4.2 Preconditioners

As we mentioned before, the general bottleneck of the algorithm Algorithm 1 defined in the Section 2.2 is the assembling and solving of the Schur complement equation (2.11). One way to solve this equation is using an iterative method, in particular, a Krylov-type method such as the method of conjugate gradients (CG) or the minimum residual method (MINRES).

The Schur complement matrix H is large and the assembling of it may be expensive. Besides, and more importantly, it becomes increasingly ill-conditioned as the IP method makes progress toward the solution. So a successful CG-based solution for our linear systems must rely

on an efficient preconditioner. In this chapter, we introduce two preconditioners implemented in Loraine and targeted pro-SDP problems with low-rank solutions, and we recall some existing ones. As mentioned before in Section 4.1, we consider the general form of SDP (4.1) and its dual (4.2). While, formally, (4.1) and (4.2) can be written in the form of (2.5) and (2.6) (mentioned to describe NT-IP Algorithm), respectively, the more detailed form is needed for software implementation and we will thus use it in the following. Furthermore, the formal separation of linear constraints is important for the derivation of the preconditioner.

Accordingly, we have the following new notation: $\mathbf{A}_i$ is the matrix $\mathbf{A}$ for the i-th LMI, which is

$$\mathbf{A}_i = [\mathrm{vec}A_1^{(i)}, \ldots, \mathrm{vec}A_n^{(i)}],$$

and W_i is the scaling matrix W for the i-th LMI.

Our goal is to solve efficiently linear systems with matrix the Schur complement matrix H. In designing the preconditioner, we will assume that H can be written as a sum

$$H = \widehat{H} + \sum_{i=1}^{p} B_i \,.$$

The idea is to replace the matrices B_i by their approximation $\widetilde{B}_i$, $i = 1, \ldots, p$, with the aim to simplify computation of the inverse of $\sum_{i=1}^{p} \widetilde{B}_i$. We will use the following general theorem.

Theorem 4.2.1. *Let $H = \widehat{H} + \sum_{i=1}^{p} B_i$ and let $\widetilde{B}_i$, $i = 1, \ldots, p$, be matrices given such that $P := \widehat{H} + \sum_{i=1}^{p} \widetilde{B}_i$ is positive definite. Assume that, for $i = 1, \ldots, p$,*

$$\lambda_{\max}\left(P^{-1/2}(B_i - \widetilde{B}_i)P^{-1/2}\right) \leq \overline{\varepsilon}_i \quad \textit{and} \quad \lambda_{\min}\left(P^{-1/2}(B_i - \widetilde{B}_i)P^{-1/2}\right) \geq \underline{\varepsilon}_i \,.$$

Then

$$\kappa\left(P^{-1/2}HP^{-1/2}\right) \leq \frac{1 + \sum_{i=1}^{p} \overline{\varepsilon}_i}{1 + \sum_{i=1}^{p} \underline{\varepsilon}_i} \,.$$

Proof. We have

$$H = \sum_{i=1}^{p} B_i + P - \sum_{i=1}^{p} \widetilde{B}_i,$$

and thus

$$P^{-1/2} H P^{-1/2} = P^{-1/2} \left(\sum_{i=1}^{p} (B_i - \widetilde{B}_i) \right) P^{-1/2} + I \,.$$

Hence

$$\lambda_k \left(P^{-1/2} H P^{-1/2} \right) = \lambda_k \left(P^{-1/2} \left(\sum_{i=1}^{p} (B_i - \widetilde{B}_i) \right) P^{-1/2} \right) + 1, \quad k = 1, \ldots, n \,.$$

Using the inequalities

$$\lambda_{\max} \left(P^{-1/2} \left(\sum_{i=1}^{p} (B_i - \widetilde{B}_i) \right) P^{-1/2} \right) \leq \sum_{i=1}^{p} \lambda_{\max} \left(P^{-1/2} (B_i - \widetilde{B}_i) P^{-1/2} \right)$$

and

$$\lambda_{\min} \left(P^{-1/2} \left(\sum_{i=1}^{p} (B_i - \widetilde{B}_i) \right) P^{-1/2} \right) \geq \sum_{i=1}^{p} \lambda_{\min} \left(P^{-1/2} (B_i - \widetilde{B}_i) P^{-1/2} \right)$$

we get the required result. □

4.2.1 Rank of H

The Schur complement matrix H can be written as a sum

$$H = \sum_{i=1}^{p} H_{\text{lmi}}^{i} + H_{\text{lin}} \,,$$

where H_{lin} is associated with linear constraints. Recall from (2.11) that each H_{lmi}^{i} is computed as

$$H_{\text{lmi}}^{i} = \mathbf{A}_i^{\top} (W_i \otimes W_i) \mathbf{A}_i \,, \tag{4.3}$$

where W_i is the NT scaling matrix.

In the next three Lemmas 4.2.2–4.2.4, we will omit the subscript i and refer to the problem (4.1) with $p = 1$, for simplicity, before returning to the full notation in Theorem 4.2.5. The next

lemma shows that W has the same rank as X. So, as X is low-rank, W will be low-rank. Later in this section, we will use this low-rank property of W to present the preconditioners.

Let (X, S) be a pair of solutions to (4.1). By complementarity, we know that $XS = 0$, and $\text{rank}(X) + \text{rank}(S) = k + \sigma \leq m$. Assume $k + \sigma = m$, i.e., strict complementarity. From $XS = 0,\ X \succcurlyeq 0,\ S \succeq 0$, we know that X and S are simultaneously diagonalizable, i.e., there exists U such that $X = U\Lambda_X U^\top$, $S = U\Lambda_S U^\top$, where Λ_X and Λ_S are the diagonal matrices of eigenvalues and $U^\top U = I$ [33, Theorem 1.3.12].

Lemma 4.2.2. *Let X, S be as above. Let $W \in \mathbb{S}^m$ be any matrix such that $X = WSW$. Then rank(W) = rank(X).*

Proof. Assume, without loss of generality, that eigenvalues of X and S are sorted such that $\Lambda_X = \text{Diag}\,(\lambda_1, \ldots, \lambda_k, 0, \ldots, 0)$ and $\Lambda_S = \text{Diag}\,(0, \ldots, 0, \mu_1, \ldots, \mu_{m-k})$. Then $X = WSW$ is equivalent to $U\Lambda_X U^\top = WU\Lambda_S\, U^\top W$, and so, as $U^\top U = I$, $\Lambda_X = U^\top WU\Lambda_S\, U^\top WU$. Define $Z = U^\top WU$, so that $\Lambda_X = Z^\top \Lambda_S\, Z$. Because $(\Lambda_X)_{i,i} = 0$ for $i > k$, it must hold that $z_{:,i}^\top z_{:,i} = 0$, $i > k$, where $z_{:,i}$ is the i-th column of Z. As Z is positive semidefinite, it means that $Z_{i,j} = 0$ for $i, j > k$, i.e., the leading $k \times k$ submatrix of Z is a rank-k non-zero matrix and the rest of the matrix is zero. Hence, $\text{rank}(Z) = k$ and, as $W = UZU^\top$, so is the rank of W. □

The proof of the next lemma is straightforward.

Lemma 4.2.3. *Let $X \in \mathbb{S}^m$ such that* $\text{rank}\, X = k$, $k \leq m$. *Then* $\text{rank}(X \otimes X) = k^2$.

Lemma 4.2.4. *Let $Y \in \mathbb{S}^m$ such that* $\text{rank}\, Y = k$, $k \leq m$, *and $A \in \mathbb{R}^{n\times m}$, $n < m$. Then* $\text{rank}(AYA^\top) \leq k$.

Proof. Because $\text{rank}\, Y = k$, we have $Y = \sum_{i=1}^k \mathbf{y}^{(i)}(\mathbf{y}^{(i)})^\top$ with some $\mathbf{y}^{(i)} \in \mathbb{R}^m$, $i = 1, \ldots, k$. Hence $AYA^\top = A\left(\sum_{i=1}^k \mathbf{y}^{(i)}(\mathbf{y}^{(i)})^\top\right)A^\top = \sum_{i=1}^k \mathbf{z}^{(i)}(\mathbf{z}^{(i)})^\top$ with $\mathbf{z}^{(i)} = A\mathbf{y}^{(i)}$. Therefore

$\operatorname{rank}(AYA^\top) = \operatorname{rank}\sum_{i=1}^{k}\mathbf{z}^{(i)}(\mathbf{z}^{(i)})^\top \leq k$. A strict inequality occurs, trivially, when $n < k$. It can also occur when $\mathbf{z}^{(i)}$, $i = 1, \ldots, k$, are linearly dependent. □

By combining the above results we get the following theorem.

Theorem 4.2.5. *Let, for some $i \in \{1, \ldots, p\}$, W_i be the scaling matrix from Section 2.2 and let $\operatorname{rank} W_i = k$. Let further H^i_{lmi} be defined as in* (4.3)*. Then $\operatorname{rank} H^i_{\text{lmi}} \leq k^2$.*

Remark 4.2.1. *All results in this section have been presented under the assumption of the exact rank of the involved matrices. This is, of course, only the limit case in our algorithms. The actual matrices have outlying eigenvalues and converge to the low-rank matrices. The message of Theorem 4.2.5 remains the same, though: when W_i has k outlying eigenvalues, then H^i_{lmi} will have at most k^2 outlying eigenvalues. This is the fact on which we build the preconditioner.*

4.2.2 H_α preconditioner

The preconditioner introduced in this section has been motivated by the work of Zhang and Lavaei [86] and, in its derivation, we partly follow their paper; however, the new preconditioner differs in substantial detail, as explained below. Also, our assumption about the matrix W is weaker.

In Section 2.2, we introduced the scaling matrix W and different ways of constructing it. This matrix becomes progressively ill-conditioned as the IP method approaches the solution. This ill-conditioning of W is a result of Lemma 4.2.2. Assuming that the solution matrices X_i^*, $i = 1, \ldots, p$, have ranks k_i, the rank of matrices W_i will also tend to k_i.

The main idea of the preconditioner is to utilize Assumption 4.1.5 and decompose matrices W_i accordingly to their expected rank (k_i) as follows:

$$W_i = W_i^0 + U_i U_i^\top, \tag{4.4}$$

where U_i are $m_i \times k_i$ matrices of full column rank. Assume that the eigenvalues of W_i are sorted as follows

$$\lambda_1(W_i) \leq \cdots \leq \lambda_{m_i-k_i}(W_i) \ll \lambda_{m_i-k_i+1}(W_i) \leq \cdots \leq \lambda_{m_i}(W_i)\,.$$

Decomposition (4.4) can be readily computed by spectral decomposition of W_i:

$$W_i = \begin{bmatrix} V_i^s & V_i^l \end{bmatrix} \begin{bmatrix} \Lambda_i^s & 0 \\ 0 & \Lambda_i^l \end{bmatrix} \begin{bmatrix} V_i^s & V_i^l \end{bmatrix}^\top,$$

where $\Lambda_i^s = \operatorname{diag}(\lambda_1(W_i), \ldots, \lambda_{m_i-k_i}(W_i))$ and $\Lambda_i^l = \operatorname{diag}(\lambda_{m_i-k_i+1}(W_i), \ldots, \lambda_{m_i}(W_i))$. Then, by choosing τ_i such that $\lambda_1(W_i) \leq \tau_i < \lambda_{m_i-k_i}(W_i)$, we obtain the following form of (4.4):

$$W_i = \underbrace{\begin{bmatrix} V_i^s & V_i^l \end{bmatrix} \begin{bmatrix} \Lambda_i^s & 0 \\ 0 & \tau_i I \end{bmatrix} \begin{bmatrix} V_i^s & V_i^l \end{bmatrix}^\top}_{W_i^0} + \underbrace{V_i^l(\Lambda_i^l - \tau I)(V_i^l)^\top}_{U_i U_i^\top}. \tag{4.5}$$

Recall now the form of the Schur complement matrix for problem (4.1):

$$H = \sum_{i=1}^{p} \mathbf{A}_i^\top (W_i \otimes W_i)\mathbf{A}_i + D^\top X_{\text{lin}} S_{\text{lin}}^{-1} D, \tag{4.6}$$

in which $X_{\text{lin}} = \operatorname{diag}(\mathbf{x}_{\text{lin}})$ and $S_{\text{lin}} = \operatorname{diag}(\mathbf{s}_{\text{lin}})$.

Remark 4.2.2. *Formally, we could treat the linear constraints, if present, as matrix constraints with diagonal matrices and perform the above decomposition for these constraints, too. However, it is rather unlikely that the corresponding part of the solution, vector* $\mathbf{x}_{\text{lin}}$ *would be "low rank", i.e., that only a very few of the linear constraints would be active. This fact could then ruin the whole idea. Here we propose to treat the linear constraints as "full rank" and add the matrix* $D^\top X_{lin} S_{lin}^{-1} D$ *to the preconditioner as is. In two of our applications, truss topology optimization and sensor network localization, the linear constraints only consist of upper and lower bounds (in the dual formulation), hence the matrix* $D^\top X_{lin} S_{lin}^{-1} D$ *is diagonal and satisfies Assumption 4.1.3. The third application, Chapter 7, relaxation of the unconstrained binary quadratic optimization problem, the matrix* $D^\top X_{lin} S_{lin}^{-1} D$ *is chordal and sparse (Lemma 7.2.8), so sparse Cholesky factorization leads to zero fill-in and is thus very efficient.*

By substituting the splittings (4.5) for $i = 1, \ldots, p$ into the matrix (4.6), we get

$$\begin{aligned} H &= \sum_{i=1}^{p} \mathbf{A}_i^\top \left((W_i^0 + U_i U_i^\top) \otimes (W_i^0 + U_i U_i^\top) \right) \mathbf{A}_i + D^\top X_{\text{lin}} S_{\text{lin}}^{-1} D \\ &= \sum_{i=1}^{p} \mathbf{A}_i^\top \left(W_i^0 \otimes W_i^0 + U_i U_i^\top \otimes W_i^0 + W_i^0 \otimes U_i U_i^\top + U_i U_i^\top \otimes U_i U_i^\top \right) \mathbf{A}_i \\ &\qquad + D^\top X_{\text{lin}} S_{\text{lin}}^{-1} D. \end{aligned}$$

Using the identity $\mathbf{A}^\top(\Phi \otimes \Xi)\mathbf{A} = \mathbf{A}^\top(\Xi \otimes \Phi)\mathbf{A}$ for any $\Phi, \Xi \in \mathbb{R}^{m_i \times m_i}$ with $\mathbf{A}$ defined as above ([86, Lemma 6] and [34, Chap. 4.2, Problem 25]), we obtain

$$H = \sum_{i=1}^{p} \mathbf{A}_i^\top (W_i^0 \otimes W_i^0) \mathbf{A}_i + \sum_{i=1}^{p} \mathbf{A}_i^\top (U_i \otimes \Gamma_i)(U_i \otimes \Gamma_i)^\top \mathbf{A}_i + D^\top X_{\text{lin}} S_{\text{lin}}^{-1} D,$$

where Γ_i is any matrix satisfying $\Gamma_i \Gamma_i^\top = 2W_i^0 + U_i U_i^\top$. Next we define $V_i = \mathbf{A}_i^\top (U_i \otimes \Gamma_i)$ for $i = 1, \ldots, p$. Then,

$$\begin{aligned} H &= \sum_{i=1}^{p} \mathbf{A}_i^\top (W_i^0 \otimes W_i^0) \mathbf{A}_i + D^\top X_{\text{lin}} S_{\text{lin}}^{-1} D + \sum_{i=1}^{p} V_i V_i^\top \\ &= \sum_{i=1}^{p} \underbrace{\mathbf{A}_i^\top (W_i^0 \otimes W_i^0) \mathbf{A}_i}_{H_i^0} + D^\top X_{\text{lin}} S_{\text{lin}}^{-1} D + \tilde{V} \tilde{V}^\top , \end{aligned} \tag{4.7}$$

where $\tilde{V} = [V_1, \ldots, V_p]$.

We now approximate $\mathbf{A}_i^\top (W_i^0 \otimes W_i^0) \mathbf{A}_i$ by $\tau_i^2 I$ in (4.7), to define the H_α preconditioner, which is

$$H_\alpha = \underbrace{\left(\sum_{i=1}^{p} \tau_i^2 I + D^\top X_{\text{lin}} S_{\text{lin}}^{-1} D \right)}_{\mathbf{A}_\alpha} + \tilde{V} \tilde{V}^\top , \tag{4.8}$$

or, in short,

$$H_\alpha = \mathbf{A}_\alpha + \tilde{V} \tilde{V}^\top . \tag{4.9}$$

By using the Sherman-Morrison-Woodbury (SMW) formula, its inverse will be

$$H_\alpha^{-1} = \mathbf{A}_\alpha^{-1} (I - \tilde{V} \Theta^{-1} \tilde{V}^\top \mathbf{A}_\alpha^{-1}) , \tag{4.10}$$

where $\Theta = I + \tilde{V}^\top \mathbf{A}_\alpha^{-1} \tilde{V}$ is the Schur complement. Notice that, by Assumption 4.1.3, the inverse of $\mathbf{A}_\alpha$ is inexpensive.

Theorem 4.2.6. *Assume that, for $i = 1, \ldots, p$,*

$$\lambda_{\max}\left(H_\alpha^{-1/2}(H_i^0 - \tau_i^2 I)H_\alpha^{-1/2}\right) \leq \overline{\varepsilon}_i \quad \textit{and} \quad \lambda_{\min}\left(H_\alpha^{-1/2}(H_i^0 - \tau_i^2 I)H_\alpha^{-1/2}\right) \geq \underline{\varepsilon}_i\,.$$

Then

$$\kappa\left(H_\alpha^{-1/2} H H_\alpha^{-1/2}\right) \leq \frac{1 + \sum_{i=1}^{p} \overline{\varepsilon}_i}{1 + \sum_{i=1}^{p} \underline{\varepsilon}_i}\,.$$

Proof. The result follows directly from Theorem 4.2.1 by setting $\widehat{H} = D^\top X_{\text{lin}} S_{\text{lin}}^{-1} D + \tilde{V}\tilde{V}^\top$, $B_i = \mathbf{A}_i^\top (W_i^0 \otimes W_i^0)\mathbf{A}_i$ and $\widetilde{B}_i = \tau_i^2 I$, $i = 1, \ldots, p$. □

Complexity of the PCG

- We compute the Schur complement

$$\Theta = I + [\mathbf{A}_1^\top (U_1 \otimes \Gamma_1) \ldots \mathbf{A}_p^\top (U_1 \otimes \Gamma_p)]^\top \mathbf{A}_\alpha^{-1} [\mathbf{A}_1^\top (U_1 \otimes \Gamma_1) \ldots \mathbf{A}_p^\top (U_p \otimes \Gamma_p)],$$

  block-wise, with the (i,j)-block $\Theta_{ij} = B_i^\top B_j$, where $B_i = R^{-1}\mathbf{A}_i^\top (U_i \otimes \Gamma_i)$, $i = 1, \ldots, p$, and R is the Cholesky factor of $\mathbf{A}_\alpha$. Then, we multiply $(C_i =) R^{-1}\mathbf{A}_i^\top$, which is $\mathcal{O}(n)$ flops. The sparsity of $\mathbf{A}_i$ is maintained in C_i, and so $C_i d$ is still $\mathcal{O}(n)$ flops for an arbitrary vector d. Hence, as $(U_i \otimes \Gamma_i)$ has $m_i k_i$ columns, the product $C_i(U_i \otimes \Gamma_i)$ requires $\mathcal{O}(n m_i k_i)$ flops. Finally, we compute $B_i^\top B_j$, which requires $\mathcal{O}(m_i m_j k_i k_j)$ flops, if B_i and B_j are sparse as well. If this is not the case, the complexity would be $\mathcal{O}(m_i m_j k_i k_j n)$, however the additional factor n in the complexity formula can be avoided by reordering calculations appropriately. Define $\widehat{m} = \max_{i=1,\ldots,p} m_i$ and $\hat{k} = \max_{i=1,\ldots,p} k_i$. Forming and factorizing the Schur complement (i.e., forming the preconditioner) requires $\mathcal{O}\left(p^2 \widehat{m}^2 \hat{k}^2 + pn\widehat{m}\hat{k} + \widehat{m}^3 \hat{k}^3\right)$ flops; the last term comes from the Cholesky factorization of Θ.

Algorithm 4 Solution of the linear system $H\Delta y = r$ by PCG

Given data $k > 0$ (solution rank), $W_i \in \mathbb{S}^m_{++}$, $i = 1, \ldots, p$, $r \in \mathbb{R}^n$ (right-hand side) and an initial iterate $\Delta \mathbf{y}^0$.

procedure SETTING UP PRECONDITIONER H_α

  for $i = 1, \ldots, p$ **do** ▷ computing the decomposition (4.7)

    Compute eigenvalue decomposition $W_i = V_i \Lambda_i V_i^\top$ and set $\tau_i = \lambda_{\min}(W_i)$.

    Form the matrices W_i^0 and U_i by (4.5).

    Compute the Cholesky factorization $\Gamma_i \Gamma_i^\top = 2W_i^0 + U_i U_i^\top$.

  end for

  Compute the Schur complement $\Theta = I + \tilde{V}^\top \mathbf{A}_\alpha^{-1} \tilde{V}$ and its Cholesky factorization $\Theta = LL^\top$.

end procedure

procedure PCG

  Use standard PCG algorithm to solve $H\Delta \mathbf{y} = r$ until $\|H\Delta \mathbf{y} - r\|/\|r\| \leq \varepsilon_{\text{CG}}$.

  At each PCG iteration:

    Compute the matrix-vector product with H using (2.10).

    Compute the matrix-vector product with H_α^{-1} by means of the SMW in (4.10), using $\Theta^{-1} = L^{-\top} L^{-1}$.

end procedure

- Each iteration of the PCG method is dominated by the matrix-matrix product in (2.10) requiring $\mathcal{O}\left(\sum_{i=1}^{p} m_i^3\right)$ flops and application of the preconditioner (4.10) requiring $\mathcal{O}\left(\sum_{i=1}^{p} m_i k_i\right)^2 + \mathcal{O}\left(n \sum_{i=1}^{p} m_i k_i\right)$ flops.

- The number of PCG iterations is a little unpredictable, in particular, when approaching the solution of the problem. However, our experience shows that this number is usually well below 10 and does not exceed 100 in the worst problems. Dropping the lower-order terms and assuming $n \approx \widehat{m}^2$, we will need $\mathcal{O}\left(\widehat{m}^3 \hat{k}^3\right)$ flops in the preconditioner and $\mathcal{O}\left(\widehat{m}^3 \hat{k}\right)$ flops in one PCG iteration. We come to the conclusion that the cost of the preconditioner is about the same as the cost of the solution of the linear system by the PCG method.

In Figure 4.1 we can see why H_α is nearly a perfect preconditioner. The figure on the left shows the distribution of the eigenvalues of Schur complement matrix for the problem `tru7` in the last iteration of IP (see Section 5.4). We observe that there is one distinct eigenvalue and the remaining are spread from almost 50 to 3×10^8. The second figure on the right-hand side shows the distribution of the eigenvalues of the preconditioned matrix $H_\alpha^{-1} H$. Most of the eigenvalues are equal to 1 and the rest do not exceed 1.36. Thus, we can see that the preconditioned matrix has clustered eigenvalues with a very small condition number, and this is the case we are looking for.

$\widetilde{H}$ preconditioner of [86] The main idea of proposing preconditioner H_α comes from Zhang and Lavaei [86]. In their paper, they do not consider linear constraints and approximate $\mathbf{A}_i^\top (W_i^0 \otimes W_i^0) \mathbf{A}_i$ by $\tau_i^2 \mathbf{A}_i^\top \mathbf{A}_i$ in (4.7), the result yielding the following preconditioner denoted in [86] by $\tilde{H}$ (and adopted here to the case of more LMIs and additional linear constraints):

$$\widetilde{H} = \sum_{i=1}^{p} \tau_i^2 \mathbf{A}_i^\top \mathbf{A}_i + D^\top X_{\text{lin}} S_{\text{lin}}^{-1} D + \tilde{V} \tilde{V}^\top . \tag{4.11}$$

Obviously, the only difference between H_α and $\widetilde{H}$ is in the first term of the sum. As this term needs to be inverted in the SMW formula, $\widetilde{H}$ requires an additional assumption that it is easily

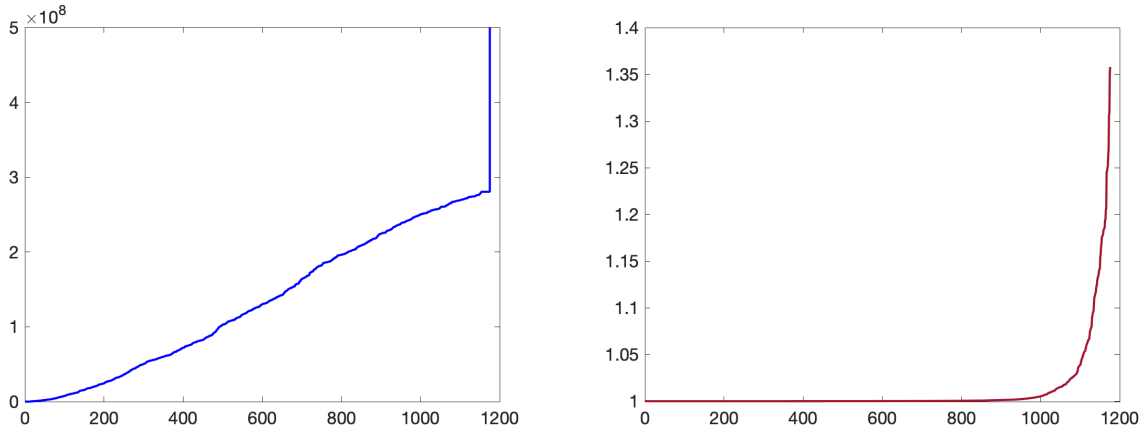


Figure 4.1: The distribution of eigenvalues of H in logarithmic scaling (on the left) and $H_\alpha^{-1}H$ is linear scaling (on the right) in tru7.

invertible. This assumption is not satisfied in our problems. Later in Section 5.4, Table 5.5, we will demonstrate that, despite being seemingly simple, H_α is at least as efficient as $\widetilde{H}$ and requires much less computational time.

4.2.3 H_β and hybrid preconditioner

Recall the definition of the matrix H:

$$H = \sum_{i=1}^{p} \mathbf{A}_i^\top (W_i^0 \otimes W_i^0)\mathbf{A}_i + \tilde{V}\tilde{V}^\top + D^\top X_{\text{lin}} S_{\text{lin}}^{-1} D\,.$$

It turns out that, in our numerical examples, the last term is dominating in the first iterations of the IP algorithm, before the low-rank structure of W is clearly recognized. This is demonstrated in Figure 4.2 that shows the distribution of eigenvalues of matrices $D^\top X_{\text{lin}} S_{\text{lin}}^{-1} D$ (in red) and $\sum_{i=1}^{p} \mathbf{A}_i^\top (W_i^0 \otimes W_i^0)\mathbf{A}_i + \tilde{V}\tilde{V}^\top$ (in blue) in iterations 2, 17, 30 (final) in problem `tru7e` (see Section 5.4). This observation led to the idea of a simplified preconditioner called H_β defined as

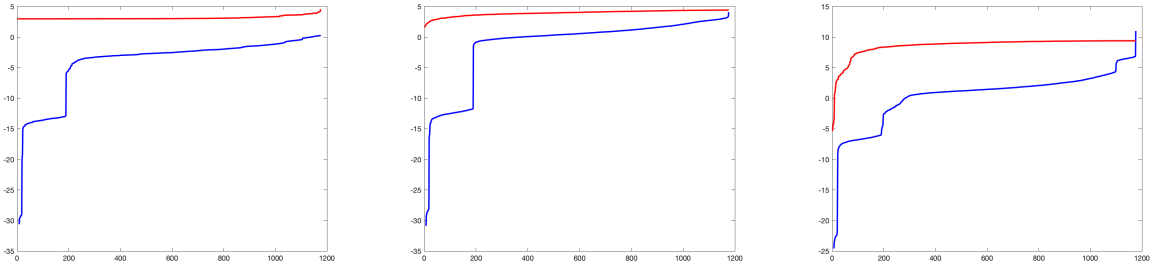


Figure 4.2: Problem tru7e; eigenvalues of $D^\top X_{lin} S_{lin}^{-1} D$ (red) and $\sum_{i=1}^{p} \mathbf{A}_i^\top (W_i^0 \otimes W_i^0) \mathbf{A}_i + \tilde{V}\tilde{V}^\top$ (blue) in iteration 2, 17 and 30 (left to right).

follows

$$H_\beta = \sum_{i=1}^{p} \tau_i^2 I + D^\top X_{\text{lin}} S_{\text{lin}}^{-1} D, \tag{4.12}$$

in which τ_i is defined as in the previous section. This matrix is easy to invert by Assumption 4.1.3; in fact, the matrix is diagonal in many problems. It is therefore an extremely "cheap" preconditioner that is efficient in the first iterations of the IP algorithm.

For relevant problems, we therefore recommend using a *hybrid preconditioner*: we start the IP iterations with H_β and, once it becomes inefficient, switch to the more expensive but more efficient H_α. See an example in Section 6.3.2.

4.3 Solving SDPs with linear equality constraints

In this chapter, we will look at how to solve SDPs with linear equality constraints using the IP method. Let us consider solving this problem

$$\begin{aligned} &\min_{\mathbf{y}\in\mathbb{R}^n, S\in\mathbb{S}^m} && b^\top \mathbf{y} \\ &\text{subject to} && \sum_{i=1}^{n} \mathrm{y}_i A_i + S = C, \\ &&& S \succeq 0, \\ &&& \Gamma^\top \mathbf{y} = \gamma. \end{aligned} \tag{4.13}$$

Here $A_i \in \mathbb{S}^m$, $i = 1, \ldots, n$, $C \in \mathbb{S}^m$, $b \in \mathbb{R}^n$, $\Gamma \in \mathbb{R}^{n\times\iota}$, $\gamma \in \mathbb{R}^\iota$ are data of the problem. As previously stated, we are utilizing the interior-point method through this thesis. Interior-point methods have been designed for optimization problems with inequality constraints and it is unable to directly handle equality constraints. One of the ways to overcome this problem is to compute the search direction of each iteration by using the Schur complement method. In addition, there are various approaches to dealing with equality constraints and reaching our desired SDP with all inequality constraints, starting with writing equality as two inequalities; see [4] for an overview focused on SDP interior-point algorithms. One of the other ways to treat the equality constraints is by the ℓ_1-penalty approach. In the following, we discuss both Schur complement method and ℓ_1-penalty approach.

4.3.1 Schur Complement Method

In this method, we start with replacing the nonsmooth constraint $S \succeq 0$ in (4.13) by the smooth, strongly convex and self-concordant penalty function $\mu \log \det S$, we get

$$\begin{aligned} \min_{\mathbf{y}\in\mathbb{R}^n, S\in\mathbb{S}^m} \quad & b^\top \mathbf{y} - \mu \log \det S \\ \text{subject to} \quad & \sum_{i=1}^{n} \mathrm{y}_i A_i + S = C, \\ & \Gamma^\top \mathbf{y} = \gamma. \end{aligned} \tag{4.14}$$

The Lagrangian of this problem is defined as

$$L(\mathbf{y}, X, S, \mathbf{z}) = b^\top \mathbf{y} - \mu \log \det S - \langle X, \sum_{i=1}^{n} \mathrm{y}_i A_i + S - C \rangle - \mathbf{z}^\top (\Gamma^\top \mathbf{y} - \gamma). \tag{4.15}$$

From KKT conditions we have

$$\mathbf{y}: \quad -b_i + \langle A_i, X \rangle + (\Gamma \mathbf{z})_i = o, \quad \forall i = 1, \ldots, n, \tag{4.16a}$$

$$X: \quad \sum_{i=1}^{n} \mathrm{y}_i A_i + S = C, \tag{4.16b}$$

$$S: \quad XS = \mu I, \tag{4.16c}$$

$$\mathbf{z}: \quad \Gamma^\top \mathbf{y} = \gamma \tag{4.16d}$$

$$X \succeq 0, \; S \succeq 0.$$

The main step at each iteration of the Interior-point method is the computation of the search direction $(\Delta X, \Delta \mathbf{y}, \Delta S, \Delta \mathbf{z})$ from the Newton equation below:

$$\langle A_i, \Delta X \rangle + (\Gamma \Delta \mathbf{z})_i = b_i - \langle A_i, X \rangle - (\Gamma \mathbf{z})_i = (r_p)_i, \quad \forall i = 1, \ldots, n, \tag{4.17a}$$

$$\sum_{i=1}^{n} \Delta \mathrm{y}_i A_i + \Delta S = C - S - \sum_{i=1}^{n} \mathrm{y}_i A_i = R_d, \tag{4.17b}$$

$$X \Delta S + \Delta X S = \sigma \mu I - XS = R_c, \tag{4.17c}$$

$$\Gamma^\top \Delta \mathbf{y} = \gamma - \Gamma^\top \mathbf{y} = r_z. \tag{4.17d}$$

First, define $\mathbf{A}$ to be an $m^2 \times n$ matrix defined by

$$\mathbf{A} := [\text{vec}A_1, \ldots, \text{vec}A_n].$$

Using the above equations, we can write the following system

$$\begin{pmatrix} 0 & \mathbf{A}^\top & 0 & \Gamma \\ \mathbf{A} & 0 & I & 0 \\ 0 & E & F & 0 \\ \Gamma^\top & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \Delta\mathbf{y} \\ \text{vec}(\Delta X) \\ \text{vec}(\Delta S) \\ \Delta\mathbf{z} \end{pmatrix} = \begin{pmatrix} r_p \\ \text{vec}(R_d) \\ \text{vec}(R_c) \\ r_z \end{pmatrix}. \tag{4.18}$$

The Schur complement equation for SDP without free variable is

$$(\mathbf{A}^\top E^{-1} F \mathbf{A})\Delta\mathbf{y} = r_p + \mathbf{A}^\top E^{-1} F \text{vec}(R_d) - \mathbf{A}^\top E^{-1} \text{vec}(R_c). \tag{4.19}$$

For rewriting the Schur complement equation for our case, first we consider the equations coming from (4.18).

$$1: \quad \mathbf{A}^\top \text{vec}(\Delta X) + \Gamma\Delta\mathbf{z} = r_p, \tag{4.20}$$

$$2: \quad \mathbf{A}\Delta\mathbf{y} + \text{vec}(\Delta S) = \text{vec}(R_d), \tag{4.21}$$

$$3: \quad E\text{vec}(\Delta X) + F\text{vec}(\Delta S) = \text{vec}(R_c), \tag{4.22}$$

$$4: \quad \Gamma^\top \Delta\mathbf{y} = r_z. \tag{4.23}$$

By replacing $\text{vec}(\Delta S)$ coming from (4.21) in (4.22), we have

$$\Delta X = \text{mat}[E^{-1}\text{vec}(R_c) - E^{-1}F\text{vec}(R_d) + E^{-1}F\mathbf{A}\Delta\mathbf{y}]. \tag{4.24}$$

Again, by substituting ΔX in (4.20), we will have

$$(\mathbf{A}^\top E^{-1} F \mathbf{A})\Delta\mathbf{y} = r_p + \mathbf{A}^\top E^{-1} F \text{vec}(R_d) - \mathbf{A}^\top E^{-1} \text{vec}(R_c) - \Gamma\Delta\mathbf{z}. \tag{4.25}$$

Now, by considering (4.23)and (4.25), the one variable equation for calculating $\Delta\mathbf{z}$ will be written as

$$\Gamma^\top(\mathbf{A}^\top E^{-1} F \mathbf{A})^{-1}\Gamma\Delta\mathbf{z} = \Gamma^\top(\mathbf{A}^\top E^{-1} F \mathbf{A})^{-1}(r_p + \mathbf{A}^\top E^{-1} F \text{vec}(R_d) - \mathbf{A}^\top E^{-1} \text{vec}(R_c)) - r_z. \tag{4.26}$$

Assuming that $\Delta\mathbf{z}$ has been computed from (4.26), we can compute $\Delta\mathbf{y}$ from (4.25). Then, we can reach ΔS and ΔX from (4.21) and (4.22), respectively.

4.3.2 l_1-Penalty Approach

For the given SDP (4.13), we treat the linear equality constraint $\Gamma^\top\mathbf{y} - \gamma = 0$ by using ℓ_1-penalty. By this penalization, (4.13) leads to the following problem

$$\begin{aligned} \min_{\mathbf{y}\in\mathbb{R}^n, S\in\mathbb{S}^m} \quad & b^\top\mathbf{y} + \mu\|\Gamma^\top\mathbf{y} - \gamma\|_1 \\ \text{subject to} \quad & \sum_{i=1}^{n} \mathrm{y}_i A_i + S = C, \\ & S \succeq 0 \end{aligned}$$

with a penalty parameter $\mu > 0$.

We now introduce two new variables, $\mathbf{r} \in \mathbb{R}^n$, $\mathbf{s} \in \mathbb{R}^n$, satisfying

$$\Gamma^\top\mathbf{y} - \gamma = \mathbf{r} - \mathbf{s}, \quad \mathbf{r} \geq 0,\ \mathbf{s} \geq 0\,.$$

By substituting this equality in the last problem, we have the problem

$$\begin{aligned} \min_{\mathbf{y}\in\mathbb{R}^n, S\in\mathbb{S}^m} \quad & b^\top\mathbf{y} + \mu\sum_{i=1}^{n}(\mathrm{s}_i + \mathrm{r}_i) \\ \text{subject to} \quad & \sum_{i=1}^{n} \mathrm{y}_i A_i + S = C, \\ & \Gamma^\top\mathbf{y} - \gamma = \mathbf{r} - \mathbf{s}, \\ & \mathbf{r} \geq 0, \mathbf{s} \geq 0, S \succeq 0. \end{aligned}$$

We eliminate the variable $\mathbf{r}$ by using the identity $\mathbf{r} = \Gamma^\top \mathbf{y} - \gamma + \mathbf{s}$, and we reach the problem

$$\begin{aligned} \min_{\mathbf{y}\in\mathbb{R}^n, S\in\mathbb{S}^m} \; & b^\top \mathbf{y} + \mu \sum_{i=1}^{n} \left((\Gamma^\top \mathbf{y} - \gamma)_i + 2\mathrm{s}_i \right) \\ \text{subject to } & C - \sum_{i=1}^{n} \mathrm{y}_i A_i \succeq 0, \\ & \Gamma^\top \mathbf{y} - \gamma + \mathbf{s} \geq 0, \\ & \mathbf{s} \geq 0. \end{aligned}$$

Now, here is our final problem with all linear inequality constraints, that the Interior-point method is designed for and can solve it.

4.4 The code Loraine

Algorithms presented in the previous chapters have been implemented in a code Loraine (for LOw-RAnk INtErior point). The implementation was done in MATLAB and Julia programming environments and is available as open source[1].

4.4.1 Special features

Loraine is a general-purpose solver for any linear SDP with linear equality and inequality constraints. Compared to other general-purpose SDP software, it particularly targets two classes of problems.

[1] github.com/kocvara/Loraine.m

Problems with low-rank solutions

Loraine is using the preconditioned conjugate gradient method, as described in detail in Section 4.2. In particular, the user can choose between the direct and iterative solver, the type of preconditioner and the expected rank of the solution. The direct solver relies on the implementation of the (sparse or dense) Cholesky factorization provided by MATLAB or Julia.

Problems with low-rank data input

This feature is only useful when a direct solver is used. In this case, the bottleneck of an interior-point algorithm is the computation of the Schur complement matrix in (2.9b) and (2.11). When the data matrices A_i are of low rank (typically of rank one), the complexity of (2.9b) can be drastically reduced. The user has a choice

- to provide vectors $(a_i^j)_k,\ k = 1, \ldots, r$, defining matrices $A_j^{(i)} = \sum_{k=1}^{r} (a_i^j)_k (a_i^j)_k^\top$ (field `Avec` in the Data input section below);
- to indicate that the input matrices $A_j^{(i)}$ are all expected to be of rank one. Then their decomposition to a vector-vector product can be automatically computed by Loraine.

4.4.2 Data input

The notation in the input file is related to the dual formulation of the problem (4.2) without slack variables. The input for the MATLAB version of Loraine is a MATLAB structure with the following fields:

type ... 'sdp' [field used for future extensions]

name ... 'name of the problem'

nvar ... number of variables n

c ... objective vector b

nlmi ... number of linear matrix inequalities p

msizes ... dimensions of the LMIs $m_1, \ldots, m_p$

nlin ... number of linear constraints ν

lsi_op ... (optional) binary vector of length nlin, set to zero for equality constraints and one for inequality constraints; if this vector is not present and $\nu > 0$, inequality type constraints are assumed.

A ... matrices $A_j^{(i)}$ stored in MATLAB sparse format as cells `A{i,j}`

Avec ... vectors $(a_i^j)_k,\ k = 1, \ldots, r$, defining matrices $A_j^{(i)} = \sum_{k=1}^{r} (a_i^j)_k (a_i^j)_k^\top$

C ... matrix C of linear constraints

d ... vector d of linear constraints

We also provide several converters, e.g., from SDPA input files. For the Julia version of Loraine, the MATLAB structure can be exported and read by the Julia code.

4.4.3 Loraine options

The following parameters can be changed by the user:

```
kit = 1;           % 0..direct solver; 1..iterative solver
tol_cg = 1e-2;     % initial tolerance for iterative solver
```

```
tol_cg_up = 0.5;    % tolerance update
tol_cg_min = 1e-6; % minimal tolerance for CG solver
cg_type = 'minres';% 'minres' or 'cg' implemented
eDIMACS = 1e-6;     % epsilon for DIMACS error stopping criterion
prec = 1;      % 0..no; 1..H_alpha; 2..H_beta; 4..hybrid
erank = 1;     % estimated rank of the solution
mup = 1000;    % initial penalty parameter mu for l1-penalization
verb = 2;      % 2..full output, 1..short output, 0..no output
datarank = 0; % -1..A_i expected rank one; 0..full rank; 1..A_i by vectors
initpoint = 0;% 0..Loraine heuristics, 1..SDPT3-like heuristics
```

4.4.4 Loraine setting

To test Loraine and compare it with the other algorithms, we have generated a library of problems. These problems arise from truss topology optimization (TTO), sensor network localization (SNL), and Lasserre relaxations of MAXCUT problems. In part II, Chapter 5-8, we present results of solving these problems obtained by Loraine, together with (sometimes brief) comparison with other SDP software. For this comparison, we use the solver MOSEK [57] which is also based on an interior-point algorithm. While this comparison may seem unfair, as MOSEK uses a direct solver to solve the linear systems, we believe that it gives a good perspective of the clear advantage of iterative solvers with good preconditioners, when available and when applicable. We further present a comparison with our MATLAB implementation of the ADMM method, based on [82].

For the truss problems, we also offer a comparison with SDPNAL+ [83], an implementation of the semi-smooth Newton-CG augmented Lagrangian method, and SDPLR [13] using a nonlinear programming algorithm based on low-rank factorization of the variables. In the following, the Loraine setting to solve these problems is mentioned.

The algorithms presented in this thesis were implemented in MATLAB and Julia codes. The codes use DIMACS stopping criteria [56] with $\varepsilon_{\text{DIMACS}} = 10^{-5}$.

The stopping parameter for the CG method ε_{CG} in Algorithm 4 has been set and updated as follows: we start with $\varepsilon_{\text{CG}} = 0.01$ and multiply it after every major iteration by 0.5, until it reaches the value 10^{-6}; then we continue with this value till convergence of the optimization algorithm.

Unless stated otherwise, we use the following hybrid preconditioner called H_{hyb}: we start with H_β (4.12) until the criterion below is satisfied and then switch to H_α (4.8):

$$N_{\text{cg_iter}} > kp\sqrt{n}/10 \;\;\&\;\; N_{\text{iter}} > \sqrt{n}/60\,.$$

Here N_{iter} is the current IP iteration and $N_{\text{cg_iter}}$ is the number of CG steps needed to solve the corrector equation in this iteration. This criterion is, obviously, purely heuristic. We further choose the value of τ in H_α and H_β as

$$\tau = \lambda_1(W_i) + 0.5\,\text{mean}(\lambda_1(W_i), \ldots, \lambda_{m-k_i}(W_i))\,.$$

To solve the problems, we used the default setting of Loraine parameters as shown before in this section, with the following exceptions, as explained in the respective paragraphs below:

SNL: `tol_cg = 1e-1, eDIMACS = 1e-4, erank = 12, initpoint = 1`

MAXCUT: `eDIMACS = 1e-6`

All problems were solved on an iMac desktop computer with 3.6 GHz 8-Core Intel Core i9 and 64 GB 2667 MHz DDR4 using MATLAB R2022b.

Part II

Applications and Numerical Results

Chapter Five

Truss topology optimization

The notation used throughout this chapter is specific to this application and unrelated to the rest of the thesis.

In this chapter, we test Loraine on a library of problems arising from truss topology optimization, both with and without vibration constraints, formulated as an SDP. Additionally, we analyze the efficiency of Loraine by comparing the numerical results with those obtained from other SDP solvers, such as MOSEK. The notations and formulations of the truss topology optimization problems follow the conventions brought in the chapter by Kočvara [40].

5.1 Truss notation

By truss, we understand a mechanical structure, an assemblage of pin-jointed uniform straight bars made of elastic material, such as steel or aluminum. The bars can only carry axial tension and compression. We denote by m the number of bars and by N the number of joints. The positions of the joints are collected in a vector $\mathbf{y}$ of dimension $\widetilde{n} := dim \cdot N$ where dim is the spatial dimension of the truss. The material properties of bars are characterized by their Young's moduli E_i, the bar

lengths are denoted by ℓ_i and bar volumes by t_i, $i = 1, \ldots, m$.

For an i-th bar, let γ_i be the vector of directional cosines; e.g., for $dim = 2$, we define

$$\gamma_i = \frac{1}{\ell_i}\left(-(\mathrm{y}_1^{(k)} - \mathrm{y}_1^{(j)}), \quad -(\mathrm{y}_2^{(k)} - \mathrm{y}_2^{(j)}), \quad \mathrm{y}_1^{(k)} - \mathrm{y}_1^{(j)}, \quad \mathrm{y}_2^{(k)} - \mathrm{y}_2^{(j)}\right)^\top,$$

where $\mathrm{y}^{(j)}$ and $\mathrm{y}^{(k)}$ are the end-points of the bar.

Let $f \in \mathbb{R}^{\widetilde{n}}$ be a load vector of nodal forces. The response of the truss to the load f is measured by nodal displacements collected in a displacement vector $\mathbf{u} \in \mathbb{R}^{\widetilde{n}}$. Some of the displacement components may be restricted: a node can be fixed in a wall, then the corresponding displacements are prescribed to be zero. The number of free nodes multiplied by the spatial dimension will be denoted by n and we will assume that $f \in \mathbb{R}^n$ and $\mathbf{u} \in \mathbb{R}^n$.

We introduce the bar stiffness matrices K_i and assemble them in the global stiffness matrix of the truss

$$K(\mathbf{t}) = \sum_{i=1}^{m} \mathrm{t}_i K_i = \sum_{i=1}^{m} \mathrm{t}_i \frac{E_i}{\ell_i^2} \gamma_i \gamma_i^\top, \quad i = 1, \ldots, m \tag{5.1}$$

and introduce the equilibrium equation

$$K(\mathbf{t})\mathbf{u} = f\,. \tag{5.2}$$

Assumption 5.1.1. *$K(\mathbf{1})$ with $\mathbf{1} = (1, 1, \ldots, 1) \in \mathbb{R}^m$ is positive definite and the load vector f is in the range space of $K(\mathbf{1})$.*

5.2 Truss topology problem, rank of the solution

Let $0 \leq \underline{\mathrm{t}}_i \leq \overline{\mathrm{t}}_i,\ i = 1, \ldots, m$, and γ be a positive constant. The basic "minimum volume" truss topology optimization (TTO) problem reads as follows:

$$\min_{\mathbf{t}\in\mathbb{R}^m, \mathbf{u}\in\mathbb{R}^n} \sum_{i=1}^{m} \mathrm{t}_i \tag{5.3}$$

subject to

$$K(\mathbf{t})\mathbf{u} = f$$

$$f^\top \mathbf{u} \leq \gamma$$

$$\underline{\mathrm{t}}_i \leq \mathrm{t}_i \leq \overline{\mathrm{t}}_i, \quad i = 1, \ldots, m\,.$$

The following version of the Schur complement theorem can be found, e.g., in [3].

Lemma 5.2.1. *Let $\mathbf{t} \in \mathbb{R}^m$, $\mathbf{t} \geq 0$, and $\gamma \in \mathbb{R}$ be fixed. Then there exists $\mathbf{u} \in \mathbb{R}^n$ satisfying*

$$K(\mathbf{t})\mathbf{u} = f \qquad \textit{and} \qquad f^\top \mathbf{u} \leq \gamma$$

if and only if

$$\begin{pmatrix} \gamma & -f^\top \\ -f & K(\mathbf{t}) \end{pmatrix} \succeq 0\,.$$

From the above lemma, we immediately get an equivalent SDP formulation of the problem:

$$\min_{\mathbf{t}\in\mathbb{R}^m} \sum_{i=1}^{m} \mathrm{t}_i \tag{5.4}$$

subject to

$$\begin{pmatrix} \gamma & -f^\top \\ -f & K(\mathbf{t}) \end{pmatrix} \succcurlyeq 0$$

$$\underline{\mathrm{t}}_i \leq \mathrm{t}_i \leq \overline{\mathrm{t}}_i, \quad i = 1, \ldots, m\,.$$

The dual to (5.4) reads as follows.

$$\max_{\substack{X\in\mathbb{S}^{n+1}\\ \overline{\rho}\in\mathbb{R}^m,\underline{\rho}\in\mathbb{R}^m}} \begin{pmatrix} -\gamma & f^\top \\ f & 0 \end{pmatrix} \bullet X - \sum_{i=1}^m \overline{\rho}_i \overline{\mathrm{t}}_i + \sum_{i=1}^m \underline{\rho}_i \underline{\mathrm{t}}_i \tag{5.5}$$

subject to

$$\begin{pmatrix} 0 & 0 \\ 0 & K_i \end{pmatrix} \bullet X - \overline{\rho}_i + \underline{\rho}_i = 1, \quad i = 1,\dots,m$$

$$X \succcurlyeq 0$$

$$\overline{\rho}_i \geq 0,\ \underline{\rho}_i \geq 0, \quad i = 1,\dots,m\,.$$

Lemma 5.2.2. *Problems* (5.4)*,* (5.5) *satisfy Assumption 4.1.2 and Assumption 4.1.3.*

Proof. Every vector γ_i in (5.1) has at most 4 non-zero elements (6 in 3D space) and thus every matrix K_i has at most 16 (36) non-zero elements, independently of the size of the problem; hence Assumption 4.1.2 is satisfied. Assumption 4.1.3 is trivially satisfied for box constraints on t_i. □

Proposition 5.2.3. [1, Chap. 2.5] *Let* $\mathbf{t}^* \in \mathbb{R}^m$ *be a solution of the minimum volume problem* (5.3) *with* $\underline{\mathrm{t}}_i = 0$*,* $i = 1,\dots,m$*. There exists a sequence* $\{\mathbf{t}_k\}_{k=1}^\infty$*,* $\mathbf{t_k} \in \mathbb{R}^m$*, with the following properties*

- $\mathbf{t}_k$ *is a solution of* (5.3) *with* $\underline{\mathrm{t}}_i = \varepsilon_k$*,* $i = 1,\dots,m$ *and* $\varepsilon_k \to 0$ *as* $k \to \infty$*;*
- $\mathbf{t}_k \to \mathbf{t}^*$ *as* $k \to \infty$*.*

Theorem 5.2.4. *There exists a solution* $(X^*, \overline{\rho}^*, \underline{\rho}^*) \in \mathbb{S}^{n+1} \times \mathbb{R}^m \times \mathbb{R}^m$ *of the dual SDP problem* (5.5) *such that the rank of* X^* *is one.*

Proof. Let $\mathbf{t}^*$ be a solution of the primal SDP problem (5.4) associated with (5.5). Denote by $S^* \in \mathbb{S}^{n+1}$ the primal slack matrix variable

$$S^* = \begin{pmatrix} \gamma & -f^\top \\ -f & K(\mathbf{t}^*) \end{pmatrix}.$$

This matrix is complementary to any solution X of the dual SDP problem, i.e.,

$$\langle X, S^* \rangle = 0\,.$$

As X and S^* are commuting and thus simultaneously diagonalizable, we have the following rank condition (see, e.g., [18]):

$$\operatorname{rank}(S^*) + \operatorname{rank}(X) \leq n + 1\,. \tag{5.6}$$

If $\underline{\mathrm{t}}_i > 0$, $i = 1, \ldots, m$, the matrix $K(\mathbf{t}^*)$ is positive definite and, due to Assumption 5.1.1, has a full rank n. Hence, by the above rank condition, the rank of any dual solution X is at most one. Since we exclude trivial solutions with $X \equiv 0$, the rank of X is exactly equal to one.

Assume now that $\underline{\mathrm{t}}_i = 0$, $i = 1, \ldots, m$, i.e., the matrix $K(\mathbf{t}^*)$ can be rank deficient. Due to Proposition 5.2.3, there is a sequence of solutions $\{S_k\}$ to the problem (5.4) with $(\underline{\mathrm{t}}_i)_k = \varepsilon_k$, $\varepsilon_k \to 0$ as $k \to \infty$, such that $S_k \to S^*$. Associated with $\{S_k\}$ there is a sequence of dual solutions $\{X_k\}$, such that any pair (S_k, X_k) satisfies the complementarity condition and thus the rank condition. By this construction, the matrices $K(\mathrm{t}_k)$ are strictly positive definite and thus, due to Assumption 5.1.1, have full rank n. From the rank condition (5.6) it follows that the rank of X_k is at most one and thus exactly equal to one. Hence there is a sequence of vectors $\{\mathbf{x}_k\}$ such that $X_k = \mathbf{x}_k \mathbf{x}_k^\top$ for each k. Then $X^* = \mathbf{x}^* (\mathbf{x}^*)^\top$ is a solution of the dual SDP problem and thus complementary to S^*. By construction, X^* is a rank-one matrix. □

Corollary 5.2.5. *Let $\underline{\mathrm{t}}_i > 0$ for $i = 1, \ldots, m$. Let $(X^*, \overline{\rho}^*, \underline{\rho}^*) \in \mathbb{S}^{n+1} \times \mathbb{R}^m \times \mathbb{R}^m$ be a solution of* (5.5)*. Then X^* has rank one.*

Problems with approximate low rank The conclusion of Corollary 5.2.5 changes when we assume that $\underline{\mathrm{t}}_i = 0$ for $i = 1, \ldots, m$. In this case, some bars may “vanish" at the optimum (the corresponding $\mathrm{t}_i = 0$). Moreover, some nodes of the structure may only be connected by these vanishing bars and hence also “vanish" from the optimal structure. As a consequence, the

stiffness matrix $K(\mathbf{t})$ will become singular with as many zero eigenvalues as the vanished nodes (not connected to the optimal structure). Then

$$\operatorname{rank}(S^*) < n \quad \text{and thus} \quad \operatorname{rank}(X^*) > 1 \quad \text{is possible}\,.$$

Hence, while there is always a rank-one solution by Theorem 5.2.4, in this case, there might be other solutions with a higher rank. In particular, when we solve the problems by an interior point method, the pair of solutions (S^*, X^*) will be maximally complementary ([18, Thm. 3.4]), so the rank of X^* will be equal to the number of the vanished nodes plus one.

Example 5.2.6. *Figure 5.1 shows an example of a truss with potentially six nodes (the remaining three nodes are fixed by boundary conditions); at the optimum, nodes 1, 2, 3, 4, 6 vanished—they are not connected to the optimal structure but are still present in the problem and its stiffness matrix. Hence the solution of the dual problem may have a rank of up to* $(1 + 2 \times 5) = 11$. *The*

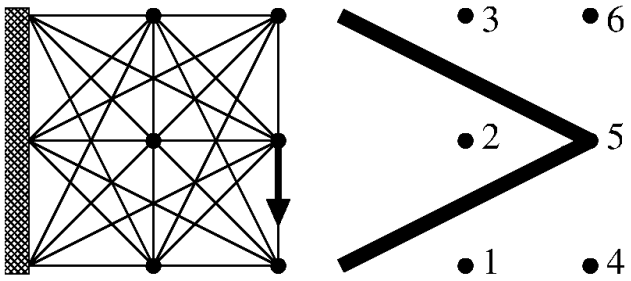


Figure 5.1: A problem with five nodes vanishing at the optimum; initial structure (left) and optimal solution (right).

associated Table 5.1 presents, column-wise, eigenvalues of X^* *and* S^* *for* $\underline{\mathrm{t}} > 0$ *(when rank-one solution* X^* *is expected) and eigenvalues of* X^* *and* S^* *for* $\underline{\mathrm{t}} = 0$ *(when one outlying eigenvalue of* X^* *is expected).*

Notice that the number of the vanished nodes depends on the initial setting of the problem and can be rather high, as compared to the total number of the nodes.

Table 5.1: Eigenvalues of X^* and S^* for problem in Fig. 5.1.

$\lambda(X^*)$, $\underline{t} > 0$	$\lambda(S^*)$, $\underline{t} > 0$	$\lambda(X^*)$, $\underline{t} = 0$	$\lambda(S^*)$, $\underline{t} = 0$
1.980482521248879	0.000000000000102	2.064483254260574	0.000000000010001
0.000000102260913	0.000003116874928	0.010238711360358	0.000000000013521
0.000000086085241	0.000003956920433	0.009214785863400	0.000000000013687
0.000000068216515	0.000006409263345	0.005367091514209	0.000000000016235
0.000000054110869	0.000007529457678	0.005013654697043	0.000000000017372
0.000000042818584	0.000009477992158	0.003545367975067	0.000000000018683
0.000000039334355	0.000011708852890	0.002923233469802	0.000000000020485
0.000000033270286	0.000012193659241	0.002610639717586	0.000000000021874
0.000000032811718	0.000014115281046	0.001846306250799	0.000000000024806
0.000000028005339	0.000014761068984	0.001771660792931	0.000000000028448
0.000000027431494	0.000016565957597	0.001555022631457	0.000000000032822
0.000000000001029	0.399999306252456	0.000000000000031	0.400000000006957
0.000000000000041	10.10000007997102	0.000000000000001	10.10000000001012

5.3 Truss topology problem, vibration constraints

Formulation of the basic truss topology problem as an SDP is rather academic. It was shown, e.g., in [35] that it is equivalent to a convex nonlinear programming problem that can be solved very efficiently by interior point methods. The SDP formulation gains significance once we add more, important and practical, constraints to the problem. In particular, it was shown in [2, 3] that a constraint on natural (free) vibrations of the optimal structure leads to a linear matrix inequality and that this is, arguably, the best way how to treat this constraint.

The free vibrations are the squares of the eigenvalues of the following generalized eigenvalue problem

$$K(\mathbf{t})\mathbf{w} = \lambda \left(M(\mathbf{t}) + M_0 \right) \mathbf{w} \,. \tag{5.7}$$

Here $M(\mathbf{t}) = \sum_{i=1}^{m} \mathrm{t}_i M_i$ is the so-called mass matrix that collects information about the mass distribution in the truss. The matrices M_i are positive semidefinite and have the same sparsity structure as K_i. The non-structural mass matrix M_0 is a constant, typically diagonal matrix with very few

nonzero elements.

Low vibrations are dangerous and may lead to structural collapse. Hence we typically require the smallest free vibration to be bigger than some threshold, that is:

$$\lambda_{\min} \geq \overline{\lambda} \quad \text{for a given } \overline{\lambda} > 0$$

where $\lambda_{\min}$ is the smallest eigenvalue of the generalized eigenvalue problem (5.7). This constraint can be equivalently written as a linear matrix inequality

$$K(\mathbf{t}) - \overline{\lambda}\,(M(\mathbf{t}) + M_0) \succcurlyeq 0 \tag{5.8}$$

which is to be added to the basic truss topology problem. As (5.8) is a linear matrix inequality in variable $\mathbf{t}$, it is natural to add this constraint to the primal SDP formulation (5.4). We will thus get the following linear SDP formulation of the truss topology design with a vibration constraint:

$$\min_{\mathbf{t}\in\mathbb{R}^m} \sum_{i=1}^{m} \mathrm{t}_i \tag{5.9}$$

subject to

$$\begin{pmatrix} \gamma & -f^\top \\ -f & K(\mathrm{t}) \end{pmatrix} \succcurlyeq 0$$

$$K(\mathbf{t}) - \overline{\lambda}\,(M(\mathbf{t}) + M_0) \succcurlyeq 0$$

$$\underline{\mathrm{t}}_i \leq \mathrm{t}_i \leq \overline{\mathrm{t}}_i, \quad i = 1, \ldots, m\,.$$

It has been shown, e.g., in [72] that, when $\underline{\mathrm{t}}_i > 0$, the rank of the optimal dual variable to the second matrix inequality is equal to the multiplicity of the smallest eigenvalue of (5.7). This multiplicity depends on the geometry of the optimal structure but is, typically, very low, usually not bigger than 1 or 2. Problem (5.9) thus fits in our framework of SDPs with very low-rank solutions.

5.4 Numerical experiments

5.4.1 Problem database

We generated a set of truss topology optimization problems of various sizes, both with and without the vibration constraint. All problems have the same geometry, boundary conditions and loads and only differ in the number of potential nodes and bars. These problems are standard testing problems characterized by very low-rank solutions and scalability, meaning that one can generate problems of various dimensions but of the same type. This may not be possible for more complicated geometries. The geometry and loading for TTO problems without vibration (formulation (5.4)) are as shown in Figure 5.1-left, while the data for problems with the vibration constraint (formulation (5.9)) differ only in the horizontal orientation of the load vector. We have generated four groups of problems with the following naming convention. In each group, all nodes in the initial ground structure are connected by potential bars.

tru<n> standard TTO problem (5.4) with $\underline{t} = 0$, discretized by $n \times n$ nodes;

tru<n>e standard TTO problem (5.4) with $\underline{t} = \varepsilon > 0$, discretized by $n \times n$ nodes;

vib<n> TTO problem with the vibration constraint (5.9) with $\underline{t} = 0$, discretized by $n \times n$ nodes;

vib<n>e TTO problem with the vibration constraint (5.9) with $\underline{t} = \varepsilon > 0$, discretized by $n \times n$ nodes.

Table 5.2 present dimensions of the generated problems. Notice that these problems satisfy Assumption 4.1.4.

For illustration, Figure 5.2 shows optimal results for problems tru25 and vib19.

Table 5.2: Problems tru<n> and tru<n>e and problems vib<n> and vib<n>e, number of variables n, size of the LMI constraint m and number of linear constraints.

problem	n	m	lin. constr.
tru3(e)	36	13	72
tru5(e)	300	41	600
tru7(e)	1176	85	2352
tru9(e)	3240	145	6480
tru11(e)	7260	221	14520
tru13(e)	14196	313	28392
tru15(e)	25200	421	50400
tru17(e)	41616	545	83232
tru19(e)	64980	685	129960
tru21(e)	97020	841	194040
tru23(e)	139656	1013	279312
tru25(e)	195000	1201	390000

problem	n	m	lin. constr.
vib3(e)	36	(13, 12)	72
vib5(e)	300	(41, 40)	600
vib7(e)	1176	(85, 84)	2352
vib9(e)	3240	(145, 144)	6480
vib11(e)	7260	(221, 220)	14520
vib13(e)	14196	(313, 312)	28392
vib15(e)	25200	(421, 420)	50400
vib17(e)	41616	(545, 544)	83232
vib19(e)	64980	(685, 684)	129960
vib21(e)	97020	(841, 840)	194040
vib23(e)	139656	(1013, 1012)	279312
vib25(e)	195000	(1201, 1200)	390000

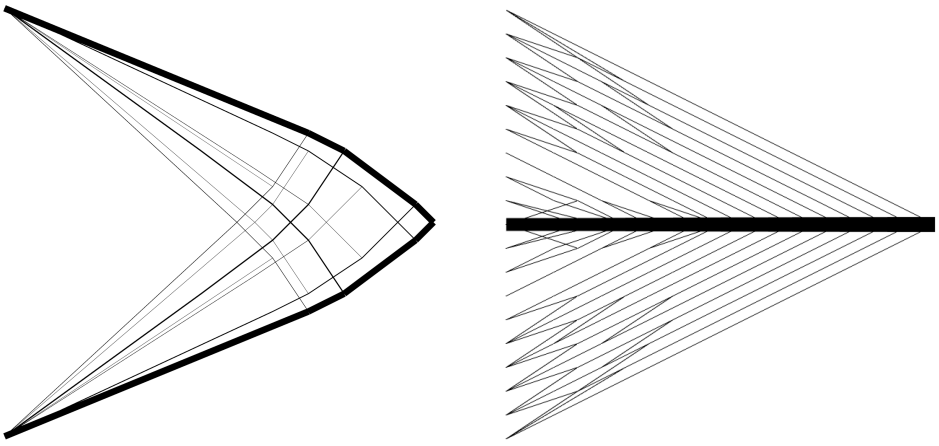


Figure 5.2: Optimal results for problems tru25 (left) and vib19 (right).

5.4.2 Numerical results

In this section, we will present numerical results for the truss topology problems from the problem database.

Before presenting the results, we want to illustrate the efficiency of the used preconditioners, in particular, in the light of the assumption of Theorem 4.2.6. In Figure 5.3–left we present the

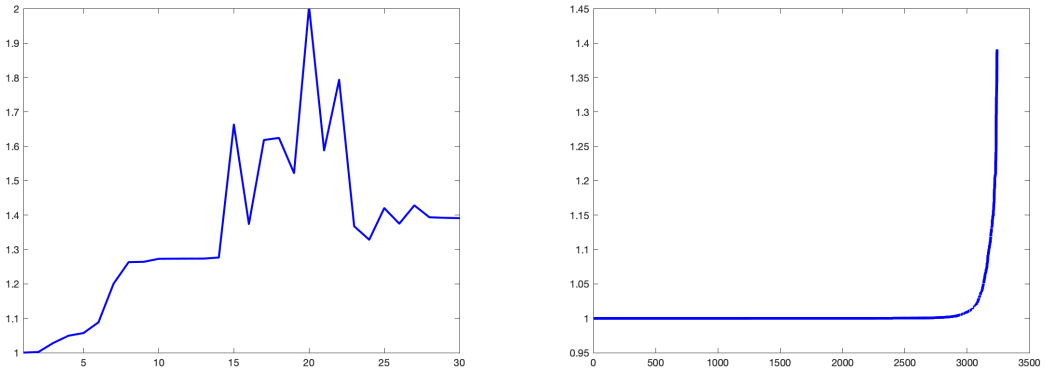


Figure 5.3: Left: The history of the condition numbers of the preconditioned matrix in problem tru9. Right: Eigenvalues of the preconditioned matrix in the last iteration of Loraine.

history of the condition numbers of the preconditioned matrices for problem tru9. The condition numbers are consistently very low. For this problem, we also computed the values of $\underline{\varepsilon}$ and $\overline{\varepsilon}$ from Theorem 4.2.6 and found that the estimate of the condition number in this theorem was always sharp. Figure 5.3–right depicts all eigenvalues of the preconditioned matrix in the last iteration. Most of the eigenvalues are very close or equal to one and the matrices have only a few outlying eigenvalues, albeit still not far from one.

We now present results for all tru and vib problems using Loraine. These are shown in Table 5.3. The table presents the number of iterations of the respective algorithm and the total number of CG iterations using the hybrid preconditioner H_{hyb}. This is followed by CPU time spent in the optimization solver and time per iteration. Notice that Loraine solves two linear systems per

iteration (predictor and corrector). For all problems, we set the expected rank of the dual solutions $k = 1$.

Table 5.3: Loraine in `tru<n>` and `vib<n>` problems

problem	iter	CG iter	CPU	CPU/iter
tru3	16	122	0.01	0.00
tru5	21	190	0.08	0.00
tru7	27	236	0.24	0.01
tru9	31	333	0.65	0.02
tru11	36	370	1.6	0.04
tru13	45	500	4.5	0.10
tru15	52	882	11	0.22
tru17	53	980	24	0.45
tru19	64	1310	51	0.80
tru21	65	1325	84	1.29
tru23	72	2450	189	2.63
tru25	87	2148	317	3.64

problem	iter	CG iter	CPU	CPU/iter
vib3	20	209	0.04	0.00
vib5	31	411	0.21	0.01
vib7	39	501	0.67	0.02
vib9	47	663	2.2	0.05
vib11	59	995	6.1	0.10
vib13	69	1153	16	0.23
vib15	80	1480	37	0.46
vib17	94	1781	92	0.98
vib19	108	2116	189	1.75
vib21	120	2844	401	3.34
vib23	130	2720	718	5.52
vib25	143	3752	1321	9.24

The following Figures 5.4 and 5.5 depict graphs of CPU times versus the number of variables for all four groups of problems, presented in the log-log scale. We can see that the time grows almost linearly (factor $\sim$1.25) with the size of the problem.

5.4.3 Other software

In this section, we give a brief comparison of Loraine with other SDP software. The next Section 5.4.3 presents results of Loraine with *direct* solver and rank-one input data, MOSEK, SDPNAL+ and SDPLR; we again give the number of iterations of the solver, CPU time and (for Loraine and MOSEK) CPU time per one iteration. The data matrices for the truss problems are of rank

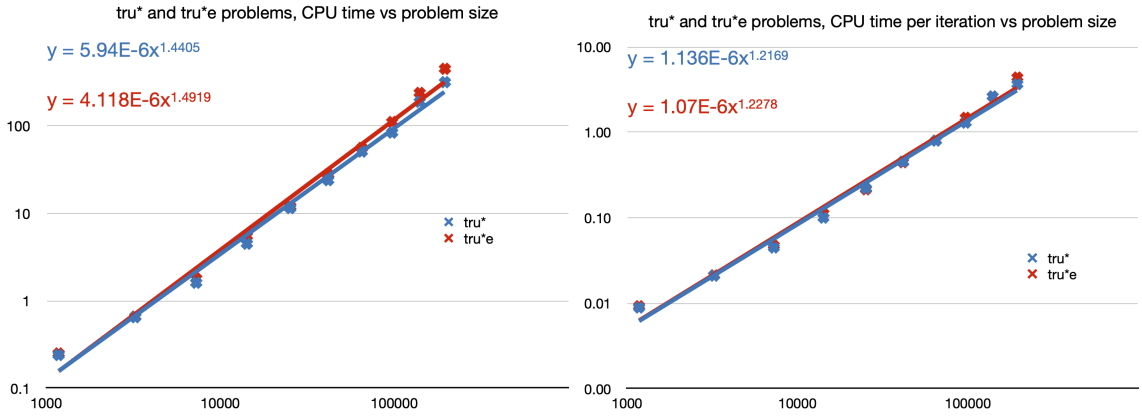


Figure 5.4: Loraine in tru∗ (blue) and tru∗e (red) problems. CPU time vs number of variables in log-log scale.

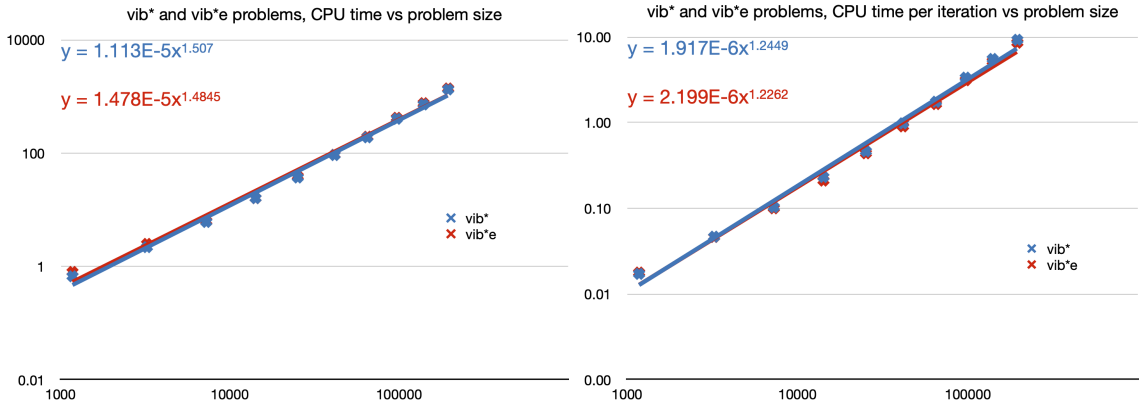


Figure 5.5: Loraine in vib∗ (blue) and vib∗e (red) problems. CPU time vs number of variables in log-log scale.

Table 5.4: Loraine direct and other solvers in `tru<n>` problems

	Loraine direct			MOSEK			SDPNAL+		SDPLR	
problem	iter	time	time/iter	iter	time	time/iter	iter	time	iter	time
tru3	16	0.01	0.001	14	0.22	0.02	440	0.5	15	0.01
tru5	21	0.09	0.004	15	0.23	0.02	9674	29	18	1.7
tru7	27	0.46	0.02	17	0.78	0.05	6324	55	19	21
tru9	31	4.3	0.14	20	5.6	0.28	47579	468	20	179
tru11	36	31	0.86	26	44	1.69	maxit		22	2469
tru13	45	239	5.31	29	235	8.10			maxit	
tru15	52	1216	23.38	32	1079	33.72				
tru17	51	4583	89.86	37	4671	126.24				

one; see (5.1). While Loraine is much more efficient for these problems with an iterative solver, the goal of this comparison is to show that when handling the rank-one data efficiently, we get a code that is (at least per one iteration) faster than other software even with a direct solver.

We only present results for problems `tru3`–`tru17`; for the larger problems MOSEK exceeded the available 64GB RAM. SDPNAL+ and SDPLR were used with default stopping tolerance. None of these codes can solve problems bigger than `tru11` before reaching the maximum number of iterations, either internal or global. While Loraine solutions satisfy DIMACS criteria with 10^{-5}, other codes often finish with lower precision. The ADMM is not included in this comparison as, even for the smallest problems, the method stagnated at a point far from the solution. In Figure 5.6 we give a comparison of the CPU time per iteration for Loraine (iterative) and MOSEK. While MOSEK actually exceeds the theoretical complexity with coefficient 2.2, the complexity of Loraine is almost linear for these problems.

The next Table 5.5 gives results of Loraine with the preconditioner H_α compared with other solvers that could potentially benefit from the low rank information: Loraine with the preconditioner $\widetilde{H}$ by Zhang and Lavaei [86]; SDPNAL+ [83], an implementation of semismooth Newton-

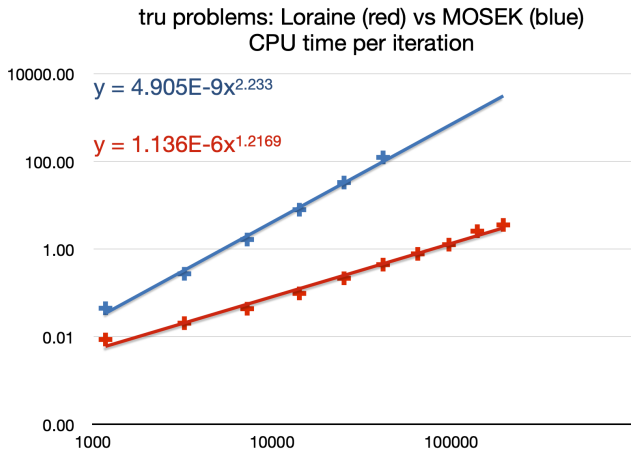


Figure 5.6: Loraine (red) and MOSEK (blue) in tru∗ problems. CPU time per iteration vs number of variables in log-log scale.

CG augmented Lagrangian method; and SDPLR [13] using a nonlinear programming algorithm based on low-rank factorization of the variables. SDPNAL+ and SDPLR were used with default stopping tolerance. None of these codes can solve problems bigger than tru11 before reaching the maximum number of iterations, either internal or global. While Loraine solutions satisfy DIMACS criteria with 10^{-5}, other codes often finish with lower precision, as documented by the final value of the objective function.

Table 5.5: Other solvers

	Loraine with H_α			Loraine with $\widetilde{H}$			SDPNAL+		SDPLR	
	iter	CG iter	CPU	iter	CG iter	CPU	iter	CPU	iter	CPU
tru3	16	122	0.01	16	131	0.02	440	0.5	15	0.01
tru5	21	190	0.08	21	197	0.3	9674	29	18	1.7
tru7	27	237	0.27	27	348	3.7	6324	55	19	21
tru9	31	331	0.94	31	570	42	47579	468	20	179
tru11	36	344	2.62	36	625	303	maxit		22	2469

5.4.4 No preconditioner

Recall the two contributing factors to the efficiency of the presented methods: a) the dedicated preconditioner and b) the very fact of using an iterative method for the solution of linear systems, a method requiring only matrix-vector multiplications, without the need to assemble and store the actual matrix. While our main focus was on the preconditioner, the second factor is just as important. Indeed, it turns out that for several problem classes, no preconditioner is actually needed; typically for the matrix completion problems used in [6, 86] or problems resulting from the relaxation of combinatorial optimization problems. The typical feature of these problems is the lack of linear constraints and the fact that the matrix $\mathbf{A}^\top\mathbf{A}$ is diagonal. In such a case, our H_α is almost identical to $\widetilde{H}$ of [86].

For example, when solving problem `gpp500-1` from SDPLIB [12] (solution of rank 6), Loraine needed 288 total CG steps (2 in the last IP iteration) and 96 seconds to solve the problem using H_α and 4045 CG steps (539 in the last IP iteration) and (only) 14 seconds to solve the problem without preconditioner. In another example, we solved the matrix completion problem ([6, 86]) with $n = 36000$, $m = 1200$ and $\operatorname{rank} X^* = 1$; in this case Loraine needed 580 total CG steps and 53 seconds to solve the problem using H_α and 1108 CG steps and 48 seconds to solve the problem without preconditioner.

To demonstrate that this is *not* the case of the truss topology problems, we include Table 5.6. It presents the results of Loraine with no preconditioner for problems that are still solvable before reaching the maximum number of CG iterations in one linear system, set to 100000. We clearly see the advantage of using the preconditioner; confront Table 5.3.

Table 5.6: Loraine with no preconditioner

	iter	CG iter	CPU		iter	CG iter	CPU
tru3	16	346	0.01	vib3	21	3115	0.06
tru5	21	2385	0.15	vib5	31	46949	2.76
tru7	27	10091	0.97	vib7	40	166135	24.9
tru9	30	33163	7.3	vib9	48	432416	181
tru11	36	77700	45				

Chapter Six

Sensor network localization with noisy data

In this chapter, we analyze the application of Loraine to the SDP relaxation of the sensor network localization problem.

6.1 The problem

Assume there are m distinct (sensor) points in $\mathbf{x}_i \in \mathbb{R}^d$, whose locations are to be determined, and ν other fixed (anchor) points, whose locations $a_1, \ldots, a_\nu$ are known. The Euclidean distance δ_{ij} between the ith and jth sensor point is known if $(i,j) \in \mathcal{I}_\mathbf{x} = \{(i,j) \mid \|\mathbf{x}_i - \mathbf{x}_j\| = \delta_{ij} \leq \rho\}$, where ρ is a fixed parameter called the radio range. Similarly, the Euclidean distance $\overline{\delta}_{kj}$ between the kth anchor and jth sensor point is known if $(k,j) \in \mathcal{I}_a = \{(k,j) \mid \|a_k - \mathbf{x}_j\| = \overline{\delta}_{kj} \leq \rho\}$. The *sensor network localization* (SNL) problem is to

$$\begin{aligned} &\text{Find } \mathbf{x}_i \in \mathbb{R}^d,\ i = 1, \ldots, m, \text{ for which} \\ &\qquad \|\mathbf{x}_i - \mathbf{x}_j\|^2 = \delta_{ij}^2, \qquad (i,j) \in \mathcal{I}_\mathbf{x} \\ &\qquad \|a_k - \mathbf{x}_j\|^2 = \overline{\delta}_{kj}^2, \qquad (k,j) \in \mathcal{I}_a\,. \end{aligned} \tag{6.1}$$

Typical networks of this type consist of a large number of densely deployed sensor nodes which gather local data and communicate with other nearby nodes. The sensor data from the nodes are relevant only if we know to what location they refer. The problem arises in applications as different as the habitat monitoring system in the Great Duck Island, detecting volcano eruptions, industrial control in semiconductor manufacturing plants, structural health monitoring, battlefield surveillance, moving object tracking, asset location, and the problem of determining protein molecule structure [10].

Figure 6.1–left shows an example of a network with 9 anchors and 512 sensors.

In many applications the data δ_{ij} are noisy. Assume perturbation of δ_{ij} and $\overline{\delta}_{ik}$ by random noises ε_{ij} and ε_{ik}, respectively:

$$\delta_{ij} = \hat{\delta}_{ij}|1 + \varepsilon_{ij}|, \quad (i,j) \in \mathcal{I}_{\mathbf{x}}$$
$$\overline{\delta}_{ik} = \hat{\overline{\delta}}_{ik}|1 + \varepsilon_{ik}|, \quad (i,j) \in \mathcal{I}_a$$

where $\hat{\delta}_{ij}$ are the true distances. Obviously, the problem (6.1) may now become infeasible.

6.2 SDP relaxation

In a series of papers, Biswas, Ye and their co-authors [11, 10, 81] proposed a relaxation of the problem based on the semidefinite programming formulation. The next lines follow the development from Biswas and Ye [11].

Let $X = [\mathbf{x}_1\ \mathbf{x}_2\ \ldots\ \mathbf{x}_m]$ be a $d \times m$ unknown matrix. Then

$$\|\mathbf{x}_i - \mathbf{x}_j\|^2 = (e_i - e_j)^T X^T X (e_i - e_j)$$
$$\|a_k - \mathbf{x}_j\|^2 = (a_k; -e_j)^T \begin{bmatrix} I_d \\ X^T \end{bmatrix} [I_d\ X](a_k; -e_j)$$

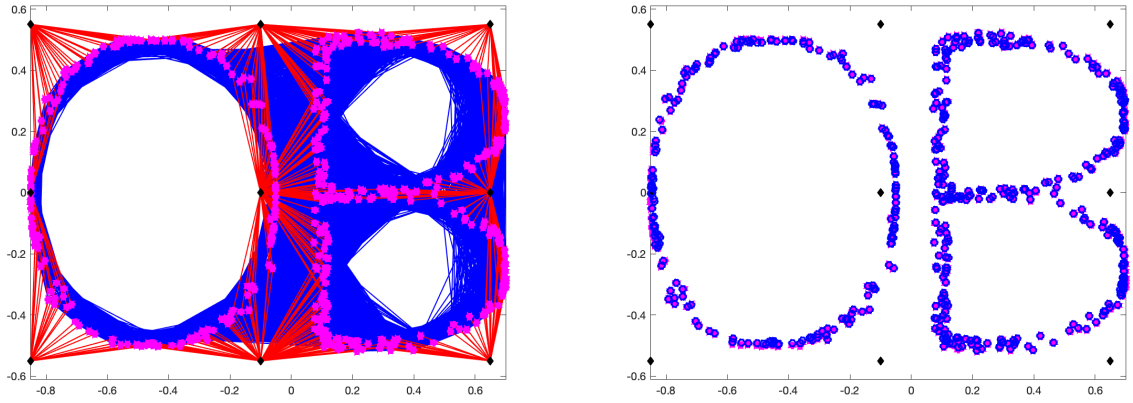


Figure 6.1: Sensor network localization problem: initial data (left) and optimal solution of the SDP relaxation (right). Black: anchors; magenta: exact positions of sensors; red (left): known distances between anchors and sensors; blue (left): known mutual distances of sensors; blue (right): computed positions of sensors.

and the original problem (6.1) can be equivalently written as

$$\text{Find } \mathbf{x}_i \in \mathbb{R}^d,\ i = 1, \dots, m, \text{ for which} \tag{6.2}$$

$$(e_i - e_j)^T X^T X (e_i - e_j) = \delta_{ij}^2$$

$$(a_k; -e_j)^T \begin{pmatrix} I_d & X \\ X^T & Y \end{pmatrix} (a_k; -e_j) = \bar{\delta}_{kj}^2$$

$$Y = X^T X\,.$$

We now relax the equality $Y = X^T X$ to $Y \succcurlyeq X^T X$, which is equivalent to the linear matrix inequality

$$Z = \begin{pmatrix} I_d & X \\ X^T & Y \end{pmatrix} \succcurlyeq 0\,.$$

The semidefinite optimization problem

$$\min_{Z,\mathbf{u},\mathbf{v}} \sum_{(i,j)\in\mathcal{I}_\mathbf{x}} (\mathrm{u}_{ij} + \mathrm{v}_{ij}) + \sum_{(k,j)\in\mathcal{I}_a} (\mathrm{u}_{kj} + \mathrm{v}_{kj}) \tag{6.3}$$

subject to

$$\begin{aligned}
&Z_{1:d,1:d} = I_d \\
&(0; e_i - e_j)^T Z (0; e_i - e_j) - \mathrm{u}_{ij} + \mathrm{v}_{ij} = \delta_{ij}^2, \quad (i,j) \in \mathcal{I}_\mathbf{x} \\
&(a_k; -e_j)^T Z (a_k; -e_j) - \mathrm{u}_{kj} + \mathrm{v}_{kj} = \bar{\delta}_{kj}^2, \quad (k,j) \in \mathcal{I}_a \\
&Z \succcurlyeq 0 \\
&\mathrm{u}_{ij}, \mathrm{v}_{ij} \geq 0, \quad (i,j) \in \mathcal{I}_\mathbf{x}\,, \qquad \mathrm{u}_{kj}, \mathrm{v}_{kj} \geq 0, \quad (k,j) \in \mathcal{I}_a
\end{aligned}$$

is called the *full SDP relaxation* (FSDP) of (6.1). It has been shown by So and Ye [70] that for certain "good" problems with no data noise this relaxation is exact.

For larger sensor network localization problems, the FSDP relaxation is difficult to solve numerically. The SDP problem has a large number of variables and the constraint matrix is thus *large* and *full*. In order to solve the problem, one may exploit the sparsity of $\mathcal{I}_\mathbf{x}$ and $\mathcal{I}_a$ at the relaxation modeling level. Several approaches have been published on this subject. A simple, yet powerful approach was proposed by Wang et al. [81] and called *edge-based relaxation* (ESDP). However, ESDP does not maintain the localizability of FSDP: the solution can have a high rank even for uniquely localizable problems. For problems with no data noise, Krislock and Wolkowicz [44] proposed a very efficient algorithm based on facial reductions. This approach, however, cannot be extended to the problems with noisy data.

As mentioned above, for many noiseless problems the relaxation is exact, i.e., the rank of the matrix Z is two. Then, for problems with small noise, Z is expected to have two large eigenvalues and several, perhaps many, small nonzero ones. The problem thus satisfies our low-rank Assumption 4.1.5 and can be efficiently solved by Loraine. This will be demonstrated in Section 7.4.

Table 6.1: Problems SNL-<n>, number of variables n, size of the LMI constraint m and number of linear constraints.

problem	n	m	lin. constr.
SNL-16	1677	130	3348
SNL-32	6818	258	13630
SNL-48	14886	386	29766
SNL-64	26767	514	53528
SNL-80	40840	642	81674
SNL-96	58956	770	117906
SNL-112	82137	898	164268
SNL-128	109464	1026	218922

6.3 Numerical experiments

6.3.1 Problem database

We consider SNL problems of growing dimension with nine anchors and n sensors randomly distributed around the letters O and B; see Figure 6.1. The radio range ρ (see the beginning of Chapter 6) was set to $\frac{1}{5}$ of the maximal vertical distance of the anchors. The known distances between the sensors were perturbed by 1% Gaussian noise. Table 6.1 shows the dimensions of the generated problems.

6.3.2 Numerical results

Recall from Section 6 that problems with noisy data lead to optimal solution Z^* of (6.3) with two outlying and a number of small, nonzero eigenvalues. Selecting $k = 2$ therefore did not lead to an

Table 6.2: Loraine, MOSEK and ADMM in `SNL-<n>` *problems*

	Loraine			MOSEK		ADMM	
problem	iter	CG iter	time	iter	time	iter	time
SNL-16	21	395	1.4	13	1.0	6239	12
SNL-32	25	532	7.4	15	21	19343	362
SNL-48	27	676	21	17	150	14328	1050
SNL-64	27	705	40	16	611	81938	17554
SNL-80	28	1064	84	21	2052		>50000
SNL-96	29	969	120	27	6579	time	
SNL-112	30	1293	224	memory			
SNL-128	30	1316	299				

efficient preconditioner and we had to increase the estimate of the rank of Z^*. For the problems we solved, $k = 12$ turned out to be a good compromise between the quality of the preconditioner and the CPU time needed to apply it. Due to the ill-conditioning of the problems, we relaxed the stopping criterion to `eDIMACS=1e-4` [56]; the same criterion was used for the ADMM code.

Table 6.2 presents the results for Loraine, MOSEK and ADMM. While ADMM turned out to be more successful than for the truss problems, it is still much slower than the two IP codes. In this table, the number of CG iterations sums up the iterations with preconditioners H_β and H_α. For instance, in problem `SNL-64`, the average number of CG iterations with H_α was 10 and the maximal number (in the last IP iteration) was 23. In Figure 6.2, we give a comparison of the CPU time per iteration for Loraine and MOSEK.

For these problems, we also demonstrate the usefulness of the hybrid preconditioner (see Section 4.2). Figure 6.3 shows the iteration log for problem `SNL-80`; the first IP iterations are solved using H_β; once it requires too many iterations, we switch to H_α.

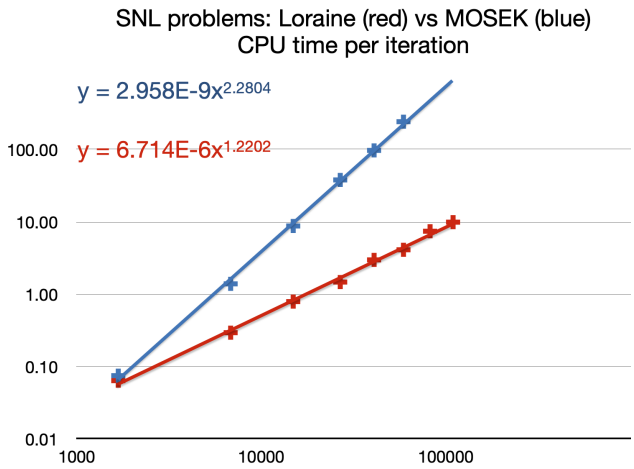


Figure 6.2: Loraine (red) and MOSEK (blue) in `SNL*` *problems. CPU time per iteration vs number of variables in log-log scale.*

```
*** Loraine v0.1 ***
Number of variables: 40839      Preconditioner     :     4
Matrix size(s)     :   642      Expected rank      :    12
Linear constraints : 81674
it        obj         error    cg_iter time/iter
 1   4.37357010e+02  3.32e+05       4     0.34
 2   4.79694156e+02  7.79e+04       4     0.40
 3   4.75844194e+02  8.44e+03      38     0.81
...
 9   1.64854663e+00  7.24e+02      45     0.71
10  -4.55049809e+00  1.73e+02      69     0.91
11  -1.14083700e+01  2.25e+01     112     1.23
Switching to preconditioner 1
12  -1.95288379e+01  8.02e+00      14     2.74
13  -2.43383931e+01  4.02e+00      13     2.65
14  -2.97196245e+01  2.32e+00      14     2.70
...
25  -4.17128514e+01  1.71e-03      40     4.61
26  -4.17274123e+01  6.84e-04      58     5.95
27  -4.17345508e+01  2.33e-04      81     7.67
28  -4.17372049e+01  7.21e-05     111     9.71
*** Total CG iterations:     1064
*** Total CPU time:      80.70 seconds
```

Figure 6.3: Problem `SNL-80` solved using the hybrid preconditioner.

Chapter Seven

Unconstrained binary quadratic optimization problems

In this chapter, we test Loraine on unconstrained binary quadratic problems. To do so, we use the linear SDPs resulting from Lasserre relaxations of these problems. To be able to use Loraine, we introduce a reformulation of the SDP relaxation of BQP, so it would be consistent with Loraine's assumptions. Finally, we test our algorithm on MAXCUT problems, randomly generated BQPs and BQPGKA library.

7.1 UBQP notation

Unconstrained binary quadratic optimization problems (UBQP) represent a surprisingly wide class of important optimization problems; see, e.g., the comprehensive overview [39]. The famous MAXCUT problem is a typical representative of this class. It is thus not surprising that they attract a great deal of attention among algorithm and software developers. Some of the most efficient algorithms for finding a global solution of UBQP combine branch-and-bound techniques with semidefinite programming (SDP) relaxations to obtain good lower bounds. These are represented

by software like Biq Mac [64], BiqCrunch [43] and BiqBin [26].

The SDP relaxations are typically based on Shor's relaxation [7] which is equivalent to the first-order Lasserre relaxation [49]. While higher-order relaxations would deliver much tighter lower bounds (if not exact solutions), the dimensions of the arising SDP problems are considered prohibitively large already for medium-sized UBQPs. For this reason, many authors proposed various techniques to strengthen the first-order relaxations; see, e.g., [27, 23, 15] and the references therein.

Our goal is to show that (at least) the second-order Lasserre relaxations are solvable by a specialized SDP software. We will also demonstrate that the second-order relaxations are, indeed, superior to the first-order ones and, in many cases, already deliver global solutions of the UBQP. This may allow one to avoid the branch-and-cut machinery, whose behavior is known to be sometimes unpredictable and replace it entirely with robust algorithms of known complexity.

Our aim is to solve UBQP

$$\min_{\mathbf{x}\in\mathbb{R}^s} \mathbf{x}^\top Q\mathbf{x} \quad \text{subject to} \quad \mathrm{x}_i \in \mathcal{B}\,, \quad i = 1,\dots,s \tag{7.1}$$

with a symmetric matrix $Q \in \mathbb{R}^{s\times s}$, where $\mathcal{B}$ is either the set $\{0,1\}$ or the set $\{-1,1\}$. *We do not assume any sparsity in* Q, it is a generally dense matrix. In order to find a global optimum, we use Lasserre hierarchy of SDP problems—relaxations—of growing dimension [49]. The SDP relaxations have the form

$$\begin{aligned} &\min_{\mathbf{y}\in\mathbb{R}^n} q^\top \mathbf{y} \\ &\text{subject to } M(\mathbf{y}) := \sum_{i=1}^{n} \mathrm{y}_i M_i - M_0 \succeq 0\,. \end{aligned} \tag{7.2}$$

Here M is a so-called moment matrix, a (generally) dense matrix of a very specific form. In particular, if the solution of (7.1) is unique and the order of the relaxation is big enough, then $\operatorname{rank} M(\mathbf{y}^*) \in \{1,2\}$, where $\mathbf{y}^*$ is a solution of (7.2) (see Section 7.2 below).

These problems are known to be difficult to solve due to the quickly growing dimension of the problem with the order of the relaxation; see, e.g., [37]. Since we do not assume any sparsity in Q, we cannot use sparse techniques, such as those proposed in [78, 79, 80]. Instead, we propose to use our software *Loraine* introduced in Chapter 4.

Several authors observed and confirmed by numerical experiments that SDP reformulation of UBQP (the first-order relaxation) can be efficiently solved by an SDP variant of the ADMM method, see [82]. We will show that not only can this observation be extended to higher-order relaxations but that inexact ADMM results can be efficiently used as a warm start for the IP algorithm in Loraine. This is due to the choice of the preconditioner used within the iterative solver in Loraine. Based on this observation, we will propose a new, *hybrid ADMM-Loraine algorithm* that, for a certain class of problems, will be superior to the single algorithms.

In this chapter, we want to use Loraine to solve this class of problems. As mentioned in Section 4.1, Loraine was developed for a primal SDP (4.1) and its dual (4.2). In UBQPs we just have one lmi, so here we consider $p = 1$. For these problems, Loraine is also efficient under Assumption 4.1.1-4.1.4, but the last Assumption 4.1.5 is not satisfied by problem (7.2) so, in the next section, we will:

(i) re-write SDP relaxation (7.2) in the primal form (4.1) by introducing auxiliary variables and additional linear equality constraints;

(ii) treat the new linear equality constraints by ℓ_1-penalty approach, in order to replace equalities by inequalities, as required by the interior-point algorithm;

(iii) show that the matrix associated with the new linear inequality constraints is block diagonal with blocks of very small size and thus satisfies Assumption 4.1.2;

(iv) show that Assumption 4.1.4 is naturally satisfied for the re-written problem.

7.2 Forms of SDP relaxations

7.2.1 Lasserre hierarchy of moment problems

Let $\mathcal{B} = \{-1, 1\}$. By introducing a matrix variable $X = \mathbf{x}\mathbf{x}^\top$, problem (7.1) can be equivalently written as the following SDP problem with a rank constraint:

$$\begin{aligned} &\min_{X \in \mathbb{S}^s} Q \bullet X \\ &\text{subject to } X_{i,i} = 1, \quad i = 1, \ldots, s \\ &X \succeq 0 \\ &\operatorname{rank} X = 1\,. \end{aligned} \tag{7.3}$$

Remark 7.2.1. *For $\mathcal{B} = \{0, 1\}$, we first use the substitution $\tilde{\mathbf{x}} = 2\mathbf{x} - 1$ to get a problem with objective function $\tilde{\mathbf{x}}^\top Q\tilde{\mathbf{x}} + 2e^\top Q\tilde{\mathbf{x}}$ and constraints $\tilde{\mathrm{x}}_i \in \{-1, 1\}$. Then we set $\widetilde{Q} = \begin{pmatrix} 0 & e^\top Q \\ Qe & Q \end{pmatrix} \in \mathbb{S}^{s+1}$ where e is the vector of all ones, introduce a variable $\widetilde{X} \in \mathbb{S}^{s+1}$, $\widetilde{X} = \begin{pmatrix} 1 \\ \tilde{\mathbf{x}} \end{pmatrix} \begin{pmatrix} 1 & \tilde{\mathbf{x}}^\top \end{pmatrix}$ and solve problem* (7.3) *with Q, X replaced by $\widetilde{Q}, \widetilde{X}$.*

In view of the above remark, *in the rest of this chapter, we will only consider the case* $\mathcal{B} = \{-1, 1\}$.

Now, to each unique component of the matrix $X = \mathbf{x}\mathbf{x}^\top$ we assign a variable y_k. Considering the constraints $X_{i,i} = 1$ and the symmetry of X, we thus have $\mathbf{y} \in \mathbb{R}^r$ with $r = s(s-1)/2$. Replacing the variable X by $\mathbf{y}$, the constraint $X \succeq 0$ can then be written as $\sum_{i=1}^r M_i \mathrm{y}_i + I \succeq 0$, where $M_i \in \mathbb{S}^s$ are suitable matrices; for details, see [49]. Also, for a practical example illustrating this concept, please refer to Example 7.2.1 in the following of this section.

By ignoring the rank constraint, we can now define the first-order Lasserre relaxation of

(7.3) as the following *moment problem*:

$$\begin{aligned} &\min_{\mathbf{y}\in\mathbb{R}^r} q^\top \mathbf{y} \\ &\text{subject to } \sum_{i=1}^{r} M_i \mathrm{y}_i + I \succeq 0\,, \end{aligned} \tag{7.4}$$

where $q \in \mathbb{R}^r$ is a restriction of $\operatorname{svec} Q$ on off-diagonal elements of Q.

Remark 7.2.2. *Formally, we should write the objective function in* (7.4) *as* $q^\top \mathbf{y} + \operatorname{trace} Q$. *For convenience, we ignore the constant term* $\operatorname{trace} Q$ *in the following development.*

Example 7.2.1. *For instance, for* $s = 3$ *we use the vector (monomial basis)* $\begin{pmatrix} \mathrm{x}_1, & \mathrm{x}_2, & \mathrm{x}_3 \end{pmatrix}^\top$*; then* $r = 3$ *and*

$$M_1 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix},\ M_2 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix},\ M_3 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

and the moment matrix $\sum_{i=1}^{r} M_i \mathrm{y}_i + I$ *is associated with the original variables as*

$$(M(\mathbf{y}) =) \sum_{i=1}^{r} M_i \mathrm{y}_i + I = \begin{pmatrix} 1 & \mathrm{y}_1 & \mathrm{y}_2 \\ \mathrm{y}_1 & 1 & \mathrm{y}_3 \\ \mathrm{y}_2 & \mathrm{y}_3 & 1 \end{pmatrix} \longleftrightarrow \begin{pmatrix} \mathrm{x}_1 \\ \mathrm{x}_2 \\ \mathrm{x}_3 \end{pmatrix} \begin{pmatrix} \mathrm{x}_1 \\ \mathrm{x}_2 \\ \mathrm{x}_3 \end{pmatrix}^\top = \begin{pmatrix} \mathrm{x}_1^2 & \mathrm{x}_1\mathrm{x}_2 & \mathrm{x}_1\mathrm{x}_3 \\ \mathrm{x}_1\mathrm{x}_2 & \mathrm{x}_2^2 & \mathrm{x}_2\mathrm{x}_3 \\ \mathrm{x}_1\mathrm{x}_3 & \mathrm{x}_2\mathrm{x}_3 & \mathrm{x}_3^2 \end{pmatrix}.$$

The Lasserre hierarchy consists in adding higher-order monomials in the monomial basis and, therefore, the higher-order relaxation problems *will all be of the form* (7.4) only of growing dimension.

Example 7.2.2. *The second-order relaxation in Example 7.2.1 will lead to a* (6×6) *moment matrix associated with the dyadic product of the vector of monomials of order up to two* $(\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3, \mathrm{x}_1^2, \mathrm{x}_2^2, \mathrm{x}_3^2, \mathrm{x}_1\mathrm{x}_2, \mathrm{x}_1\mathrm{x}_3, \mathrm{x}_2\mathrm{x}_3)^\top$ *with itself.*

As some of the terms in the higher-order moment matrix are repeated and some can be simplified using the constraint $\mathrm{x}_i^2 = 1$ (e.g., $\mathrm{x}_1\mathrm{x}_2^2$ is reduced to x_1) and thus even more terms are

repeated, the number of variables in the higher-order relaxations is smaller than the number of the elements of the upper triangle of the matrix (see also Example 7.2.9 below).

Denote the order of the Lasserre relaxation by ω.

Definition 7.2.3. *The* Lasserre relaxation of order ω *of the problem* (7.1) *is given by problem* (7.4) *with general dimensions* n, m*, replacing* r, s*, respectively; i.e.,* $\mathbf{y} \in \mathbb{R}^n$ *and* $M_i \in \mathbb{S}^m$*,* $i = 1, \ldots, n$*.*

The dimensions n, m grow quickly with the order of the relaxation. In particular, we have

$$\frac{s^4}{24} \lessapprox n \leq 2^s - 1 \quad \text{and} \quad m = \sum_{i=1}^{\omega} \binom{s}{i}. \tag{7.5}$$

(The authors are not aware of any exact formula for n or the lower bound on n; the lower bound in (7.5) is our estimate based on numerical experiments. Obviously, the higher ω the bigger n until it reaches the "saturation point" $2^s - 1$.)

The dimension of the problem makes it very challenging for standard SDP solvers based on second-order methods, such as interior-point methods; see [37] and the numerical examples in the last section of this paper.

It was shown by Fawzi et al. [22] and Sakaue et al. [65], and numerically confirmed in [37], that for these type of problems, the sequence of these approximations is *finite* and the upper bound on the order of the relaxation to obtain exact solution of (7.1) is $\lceil n/2 \rceil$. Laurent [51] showed that this is also a lower bound for MAXCUT problems with unweighted complete graphs.

To determine whether a relaxation of order ω is exact, i.e., whether its solution is the solution of (7.1), we use the rank of the moment matrix; see [49, Thm. 6.19]. In particular, assuming that (7.1) has p global minimizers, the rank of the moment matrix will be less than or equal to p. Therefore, if we assume that (7.1) has a unique solution, then the moment matrix of the exact relaxation will be of rank one.

Remark 7.2.3. *Notice that, when* $\mathcal{B} = \{-1, 1\}$*, every global minimizer* $\mathbf{x}$ *will have a "symmetric"*

counterpart $-\mathbf{x}$, as there are only quadratic terms in the problem. In this case, by unique minimizer, we understand "unique up to the multiple by -1" and the rank of the exact relaxation will be two.

Above, we spoke about an "exact relaxation" and a "rank of the moment matrix". When solving the problem by an interior point method, we can only speak about a numerical rank defined below.

Definition 7.2.4. *Consider $A \in \mathbb{S}^m_+$ with eigenvalues $\lambda_i(A)$ and with the largest eigenvalue $\lambda_{max}(A)$. Let $\varepsilon > 0$. The* numerical rank *of A is defined as*

$$r_\varepsilon(A) = |\mathcal{I}|, \quad \mathcal{I} = \left\{ i \in \{1, \ldots, m\} \mid \frac{\lambda_i(A)}{\lambda_{max}(A)} \geq \varepsilon \right\} .$$

Definition 7.2.5. *For problems with a unique solution and $\mathcal{B} = \{-1, 1\}$, the relaxation of order ω is called* exact *if the numerical rank of the corresponding moment matrix is less than or equal to two.*

7.2.2 Re-writing the problem

As mentioned in the Introduction, problem (7.4) is in the "wrong" form for our solver Loraine. While the solution $M^* := M(\mathbf{y}^*)$ of (7.4) has low rank (in particular, $\operatorname{rank} M^* = 2$ when the solution of (7.1) is unique), Loraine expects the solution of an SDP problem in the *primal form* (4.1) to have low rank. Our next goal is thus to rewrite (7.4) in the dual form; in other words, to rewrite the dual to (7.4) in the *dual form* (4.2).

The dual to (7.4) is the problem

$$\begin{aligned} &\max_{Z \in \mathbb{S}^m} -I \bullet Z \\ &\text{subject to } M_i \bullet Z = q_i, \quad i = 1, \ldots, n \\ &\qquad Z \succeq 0 \,, \end{aligned}$$

and, by vectorizing the matrices as $\mathbf{z} = \operatorname{svec} Z$ and $\mathbf{M} = (\operatorname{svec} M_1, \ldots, \operatorname{svec} M_n)^\top$, $\mathbf{M} \in \mathbb{R}^{n \times \widetilde{n}}$, it can be further written as

$$\begin{aligned} &\min_{\mathbf{z} \in \mathbb{R}^{\widetilde{n}}} (\operatorname{svec} I)^\top \mathbf{z} \\ &\text{subject to } \operatorname{smat}(\mathbf{z}) \succeq 0 \\ &\mathbf{M}\mathbf{z} = q \end{aligned} \tag{7.6}$$

with $\widetilde{n} = m(m+1)/2$. Now, from the fact that the problem (7.6) is the dual problem to (7.4), we have the following lemma:

Lemma 7.2.6. *The vector y in* (7.4) *is the Lagrangian multiplier to $\mathbf{M}\mathbf{z} = q$ in* (7.6)*.*

Problem (7.6) is now almost in the right form for our solver: it is in the primal form and the dual solution is a low-rank matrix. Notice that, despite Q being possibly a dense matrix, the data in (7.6) are always very sparse. One obstacle remains—the linear equality constraints in (7.6). We are addressing it in the next section.

7.2.3 Equality constraints by ℓ_1-penalty

As mentioned before in Section 4.3, Interior-point methods have been designed for optimization problems with inequality constraints and cannot directly handle equality constraints. There are various ways how to overcome this, starting with writing equality as two inequalities; see [4] for an overview focused on SDP interior-point algorithms. In our algorithm, we treat the equality constraints by the following ℓ_1-penalty approach.

Introducing ℓ_1-penalty described in Section 4.3.2 for the linear equality constraints in (7.6), we obtain the following problem

$$\begin{aligned} &\min_{\mathbf{z} \in \mathbb{R}^{\widetilde{n}}} (\operatorname{svec} I)^\top \mathbf{z} + \mu \|\mathbf{M}\mathbf{z} - q\|_1 \\ &\text{subject to } \operatorname{smat}(\mathbf{z}) \succeq 0\,, \end{aligned} \tag{7.7}$$

with a penalty parameter $\mu > 0$. Recall that this penalty is exact, in the sense that there exists $\mu_0 > 0$ such that the solution of (7.7) with $\mu \geq \mu_0$ is equivalent to the solution of (7.6) (see, e.g., [19]).

We now introduce two new variables, $\mathbf{r} \in \mathbb{R}^n$, $\mathbf{s} \in \mathbb{R}^n$, satisfying

$$\mathbf{M}\mathbf{z} - q = \mathbf{r} - \mathbf{s}, \quad \mathbf{r} \geq 0,\ \mathbf{s} \geq 0\,.$$

After substitution, (7.7) reads as

$$\begin{aligned} \min_{\mathbf{z}\in\mathbb{R}^{\widetilde{n}},\, \mathbf{r}\in\mathbb{R}^n,\, \mathbf{s}\in\mathbb{R}^n} \ & (\operatorname{svec} I)^\top \mathbf{z} + \mu \sum_{i=1}^{n} (\mathrm{r}_i + \mathrm{s}_i) \\ \text{subject to } & \operatorname{smat}(\mathbf{z}) \succeq 0 \\ & \mathbf{M}\mathbf{z} - q = \mathbf{r} - \mathbf{s} \\ & \mathbf{r} \geq 0,\ \mathbf{s} \geq 0\,. \end{aligned} \tag{7.8}$$

Using the identity $\mathbf{r} = \mathbf{M}\mathbf{z} - q + \mathbf{s}$ to eliminate variable r, we arrive at our final problem

$$\begin{aligned} \min_{\mathbf{z}\in\mathbb{R}^{\widetilde{n}},\, \mathbf{s}\in\mathbb{R}^n} \ & (\operatorname{svec} I)^\top \mathbf{z} + \mu \sum_{i=1}^{n} ((\mathbf{M}\mathbf{z} - q)_i + 2\mathrm{s}_i) \\ \text{subject to } & \operatorname{smat}(\mathbf{z}) \succeq 0 \\ & \mathbf{M}\mathbf{z} - q + \mathbf{s} \geq 0 \\ & \mathbf{s} \geq 0\,. \end{aligned} \tag{7.9}$$

Problem (7.9) is now in the dual form (4.2).

By Lemma 7.2.6, we know that $\mathbf{y}$ in (7.4) is the Lagrangian multiplier to the equality constraint in (7.6). The question is how to obtain $\mathbf{y}$ from (7.9), the ℓ_1-penalty reformulation of (7.6).

Lemma 7.2.7. *A solution* $\mathbf{y}$ *of* (7.4) *can be obtained from the solution of* (7.9) *as* $\mathrm{y}_i = \mu + \lambda_i,\ i = 1, \ldots, n$*, in which* μ *is the penalty parameter at the solution and* λ *is the Lagrangian multiplier to the constraint* $\mathbf{M}\mathbf{z} - q + \mathbf{s} \geq 0$*.*

Proof. We start with the optimality conditions of the problem (7.9). Using suitable matrices $\mathcal{E}_i \in \mathbb{S}^m$, we can write $\operatorname{smat}(\mathbf{z}) = \sum_{i=1}^{\widetilde{n}} \mathcal{E}_i \mathrm{z}_i$. The Lagrangian of (7.9) is then

$$\begin{aligned}\mathcal{L}_1(\mathbf{z}, \mathbf{s}, \Gamma, \lambda, \eta) = &- \left(f(\mathbf{z}) + \mu \sum_{i=1}^{n} \left((\mathbf{M}\mathbf{z} - q)_i + 2\mathrm{s}_i \right) \right) - \Gamma \bullet \sum_{i=1}^{\widetilde{n}} \mathcal{E}_i \mathrm{z}_i \\ &- \sum_{i=1}^{n} \lambda_i \left(\sum_{j=1}^{\tilde{n}} \mathbf{M}_{ij} \mathrm{z}_j - q_i + \mathrm{s}_i \right) - \sum_{i=1}^{n} \eta_i \mathrm{s}_i,\end{aligned}$$

where $\Gamma \succeq 0$ and $\lambda, \eta \geq 0$ are the corresponding Lagrangian multipliers and $f(\mathbf{z}) = (\operatorname{svec} I)^\top \mathbf{z}$. By using the KKT conditions at the solution we have

$$\begin{aligned}&\nabla_{\mathbf{z}} \mathcal{L}_1(\mathbf{z}^*, \mathbf{s}^*, \Gamma^*, \lambda^*, \eta^*) = 0 : \\ &- (\nabla f(\mathbf{z}^*))_j - \mu \sum_{i=1}^{n} \mathbf{M}_{ij} - \sum_{i=1}^{n} \mathbf{M}_{ij} \lambda_i^* - \Gamma^* \bullet \mathcal{E}_j = 0, \quad j = 1, \ldots, \widetilde{n}\end{aligned}$$

and so

$$(\nabla f(\mathbf{z}^*))_j = - \sum_{i=1}^{n} \mathbf{M}_{ij} (\mu + \lambda_i^*) - \Gamma^* \bullet \mathcal{E}_j, \quad j = 1, \ldots, \widetilde{n}\,. \tag{7.10}$$

Now, the Lagrangian of the problem (7.6) is

$$\mathcal{L}_2(\mathbf{z}, \Gamma, y) = -f(\mathbf{z}) - \Gamma \bullet \sum_{i=1}^{\widetilde{n}} \mathcal{E}_i \mathrm{z}_i - \sum_{i=1}^{n} \mathrm{y}_i \left(\sum_{j=1}^{\tilde{n}} \mathbf{M}_{ij} \mathrm{z}_j - q_i \right),$$

where, as above, $\Gamma \succeq 0$ and $\mathbf{y} \geq 0$ are the corresponding Lagrangian multipliers. From the KKT conditions, we have

$$\begin{aligned}&\nabla_{\mathbf{z}} \mathcal{L}_2(\mathbf{z}^*, \gamma^*, y^*) = 0 : \\ &- (\nabla f(\mathbf{z}^*))_j - \Gamma^* \bullet \mathcal{E}_j - \sum_{i=1}^{n} \mathbf{M}_{ij} \mathrm{y}_i^* = 0, \quad j = 1, \ldots, \widetilde{n}\end{aligned}$$

so

$$(\nabla f(\mathbf{z}^*))_j = - \sum_{i=1}^{n} \mathbf{M}_{ij} \mathrm{y}_i^* - \Gamma^* \bullet \mathcal{E}_j, \quad j = 1, \ldots, \widetilde{n}\,. \tag{7.11}$$

Thus, from (7.10) and (7.11), $\mathbf{y}$ can be written as

$$\mathrm{y}_i = \mu + \lambda_i, \quad i = 1, \ldots, n \tag{7.12}$$

and we are done. □

The question remains how to initialize and update the penalty parameter μ in order to get an exact solution of problem (7.6) using our interior-point solver. There are essentially two options.

(A) For the current value of μ, fully solve the penalized problem (7.9) by Loraine, check the residuum of the equality constraint in (7.6) for the optimal solution of (7.9) and, if bigger than a prescribed threshold, increase μ and repeat this process.

(B) Incorporate penalty update within the interior-point solver, following, e.g., [25] and [17]. That is, run the IP algorithm with an initial μ and, as soon as the residual of the equality constraint (7.6) gets bigger than a value dependent on the IP barrier parameter, increase μ and continue in IP iterations.

Option (A) is simple and robust but may also be expensive: we may need to solve several SDP problems to full optimality. Option (B) is more attractive from the efficiency point of view but is also more sensitive to the choice of parameters. We implemented (B) but had mixed experience with it so we returned to (A). The good news is that a class of BQP problems usually requires a single choice of μ in order to guarantee convergence to an optimal point with the norm of the residuum of the equality constraint smaller than a prescribed value; see the numerical experiments in the last section. Needless to say that one cannot start with a too big value of μ due to the potential ill-conditioning of the resulting SDP problem and resulting difficulties of the IP algorithm to solve the problem at all.

7.2.4 Loraine and problem (7.9)

Recall the definition of the matrix $\mathbf{M} = (\operatorname{svec} M_1, \ldots, \operatorname{svec} M_n)^T$ with matrices M_i introduced at the beginning of this section. We have the following result.

Lemma 7.2.8. *There exists a permutation matrix $P \in \mathbb{R}^{\widetilde{n}\times\widetilde{n}}$ such that $P\mathbf{M}^\top\mathbf{M}P^\top$ is a block diagonal matrix with small full blocks. In particular, $\mathbf{M}^\top\mathbf{M}$ is a sparse chordal matrix.*

Proof. Every matrix M_i localizes the corresponding variables in the moment matrix $\sum_{i=1}^{n} M_i \mathrm{y}_i + I$ in (7.4), whereas the elements of the moment matrix are either ones or the variables y_i; see also Example 7.2.1. As every M_i is associated with exactly one variable y_i, the nonzeros in this matrix are unique to the matrix. Therefore every column of matrix $\mathbf{M}$ will either contain all zeros or a single nonzero number one at the i-th position, $i = 1, \ldots, n$. We can now re-order the columns of $\mathbf{M}$ such that the first columns will only contain zeros, the next columns element 1 at the first position, the next ones element 1 at the second position, etc. This re-ordering will define the permutation matrix P. The claims follow. □

Example 7.2.9. *Consider a problem with $s = 3$ and relaxation order $\omega = 2$. Utilizing the constraints $\mathrm{x}_i^2 = 1$, we only consider variables associated with the monomial basis $b(\mathbf{x}) = (\mathrm{x}_1,\ \mathrm{x}_2,\ \mathrm{x}_3,\ \mathrm{x}_1\mathrm{x}_2,\ \mathrm{x}_2\mathrm{x}_3,\ x_1x_3)$:*

$$X = b(\mathbf{x})b^\top(\mathbf{x}) = \begin{pmatrix} \mathrm{x}_1^2 & \mathrm{x}_1\mathrm{x}_2 & \mathrm{x}_1\mathrm{x}_3 & \mathrm{x}_1^2\mathrm{x}_2 & \mathrm{x}_1\mathrm{x}_2\mathrm{x}_3 & \mathrm{x}_1^2\mathrm{x}_3 \\ \mathrm{x}_1\mathrm{x}_2 & \mathrm{x}_2^2 & \mathrm{x}_2\mathrm{x}_3 & \mathrm{x}_1\mathrm{x}_2^2 & \mathrm{x}_2^2\mathrm{x}_3 & \mathrm{x}_1\mathrm{x}_2\mathrm{x}_3 \\ \mathrm{x}_1\mathrm{x}_3 & \mathrm{x}_2\mathrm{x}_3 & \mathrm{x}_3^2 & \mathrm{x}_1\mathrm{x}_2\mathrm{x}_3 & \mathrm{x}_2\mathrm{x}_3^2 & \mathrm{x}_1\mathrm{x}_3^2 \\ \mathrm{x}_1^2\mathrm{x}_2 & \mathrm{x}_1\mathrm{x}_2^2 & \mathrm{x}_1\mathrm{x}_2\mathrm{x}_3 & \mathrm{x}_1^2\mathrm{x}_2^2 & \mathrm{x}_1\mathrm{x}_2^2\mathrm{x}_3 & \mathrm{x}_1^2\mathrm{x}_2\mathrm{x}_3 \\ \mathrm{x}_1\mathrm{x}_2\mathrm{x}_3 & \mathrm{x}_2^2\mathrm{x}_3 & \mathrm{x}_2\mathrm{x}_3^2 & \mathrm{x}_1\mathrm{x}_2^2\mathrm{x}_3 & \mathrm{x}_2^2\mathrm{x}_3^2 & \mathrm{x}_1\mathrm{x}_2\mathrm{x}_3^2 \\ \mathrm{x}_1^2\mathrm{x}_3 & \mathrm{x}_1\mathrm{x}_2\mathrm{x}_3 & \mathrm{x}_1\mathrm{x}_3^2 & \mathrm{x}_1^2\mathrm{x}_2\mathrm{x}_3 & \mathrm{x}_1\mathrm{x}_2\mathrm{x}_3^2 & \mathrm{x}_1^2\mathrm{x}_3^2 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & \mathrm{x}_1\mathrm{x}_2 & \mathrm{x}_1\mathrm{x}_3 & \mathrm{x}_2 & \mathrm{x}_1\mathrm{x}_2\mathrm{x}_3 & \mathrm{x}_3 \\ \mathrm{x}_1\mathrm{x}_2 & 1 & \mathrm{x}_2\mathrm{x}_3 & \mathrm{x}_1 & \mathrm{x}_3 & \mathrm{x}_1\mathrm{x}_2\mathrm{x}_3 \\ \mathrm{x}_1\mathrm{x}_3 & \mathrm{x}_2\mathrm{x}_3 & 1 & \mathrm{x}_1\mathrm{x}_2\mathrm{x}_3 & \mathrm{x}_2 & \mathrm{x}_1 \\ \mathrm{x}_2 & \mathrm{x}_1 & \mathrm{x}_1\mathrm{x}_2\mathrm{x}_3 & 1 & \mathrm{x}_1\mathrm{x}_3 & \mathrm{x}_2\mathrm{x}_3 \\ \mathrm{x}_1\mathrm{x}_2\mathrm{x}_3 & \mathrm{x}_3 & \mathrm{x}_2 & \mathrm{x}_1\mathrm{x}_3 & 1 & \mathrm{x}_1\mathrm{x}_2 \\ \mathrm{x}_3 & \mathrm{x}_1\mathrm{x}_2\mathrm{x}_3 & \mathrm{x}_1 & \mathrm{x}_2\mathrm{x}_3 & \mathrm{x}_1\mathrm{x}_2 & 1 \end{pmatrix}$$

i.e., variables $\mathbf{y} \in \mathbb{R}^7$ corresponding to $(\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3, \mathrm{x}_1\mathrm{x}_2,\ \mathrm{x}_2\mathrm{x}_3, \mathrm{x}_1\mathrm{x}_3, \mathrm{x}_1\mathrm{x}_2\mathrm{x}_3)$, the unique elements

of X. The matrices M_i simplify to

$$M_1 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}, \; M_2 = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \ldots, M_7 = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

and thus

$$\mathbf{M} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

and

$$\mathbf{M}P^\top = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

with a suitable permutation matrix P. Then $P\mathbf{M}^\top\mathbf{M}P^\top$ is block diagonal with block sizes 2 and 3.

Remark 7.2.4. *The size of the blocks in the matrix $P\mathbf{M}^\top\mathbf{M}P^\top$ is given by the count of the corresponding variables in the upper triangle of the moment matrix. This number grows with the relaxation order.*

Theorem 7.2.10. *Assume that* (7.1) *has a unique solution and that Lasserre hierarchy is exact for order* $\omega \geq 1$. *Then problem* (7.9) *corresponding to* ω *satisfies Assumptions 4.1.1–4.1.5.*

Proof. A strictly feasible point for (7.9) can be found, e.g., by choosing z as a unit vector and s positive, arbitrarily large. Similarly, the moment problem (7.4) is trivially strictly feasible by choosing $\mathbf{y} = 0$. Hence Assumption 4.1.1 holds. Every matrix $\mathcal{E}_i$ contains at most two nonzero elements, hence Assumption 4.1.2 is trivially satisfied. Assumption 4.1.3 holds by Lemma 7.2.8: because the matrix $\mathbf{M}^\top\mathbf{M}$ is chordal and sparse, sparse Cholesky factorization leads to zero fill-in and is thus very efficient. Assumption 4.1.4 is satisfied by the construction of the problem, as the number of variables $\widetilde{n}$ is proportional to the square of the size of the matrix inequality m. Finally, under the assumption of a unique solution to (7.1) and exact relaxation, the dual variable associated with the matrix inequality in (7.9) has rank at most two. Hence Assumption 4.1.5 is satisfied. □

7.3 ADMM for SDP relaxations

The Alternating Direction Method of Multipliers (ADMM) is a popular alternative to the interior-point methods for convex optimization problems. Its extension to semidefinite optimization was introduced in [82]. It proved to be efficient for certain SDP problems, among others many UBQPs ([82, Sec 4.3]). We thus briefly describe this algorithm and offer a numerical comparison with Loraine. Furthermore, we will introduce a new “hybrid" approach combining ADMM with Loraine.

7.3.1 The ADMM-SDP algorithm

The method consists of minimizing the augmented Lagrangian of the problem

$$\mathcal{L}_\rho(\mathbf{y}, S, X) = -b^\top\mathbf{y} + X \bullet \left(\sum_{i=1}^{n} \mathrm{y}_i A_i + S - C\right) + \frac{1}{2\rho}\left\|\sum_{i=1}^{n} \mathrm{y}_i A_i + S - C\right\|_F^2 \tag{7.13}$$

alternately, first with respect to $\mathbf{y}$, then with respect to S, and then updating X by

$$X^{(k+1)} = X^{(k)} + \frac{\sum_{i=1}^{n} \mathrm{y}_i^{(k+1)} A_i + S^{(k+1)} - C}{\rho}.$$

While the minimization with respect to $\mathbf{y}$ leads to an unconstrained problem and the minimum is obtained by solving the first-order optimality system of (linear) equations, according to the next Lemma, the minimization of $\mathcal{L}_\rho$ with respect to S can be formulated as a projection of the matrix $C - \sum_{i=1}^{n} \mathrm{y}_i A_i - \rho X$ onto the semidefinite cone $\mathbb{S}_+^m$.

Lemma 7.3.1. *Let $\mathcal{L}_\rho$ be the function* (7.13)*. Then, in this context, optimizing $\mathcal{L}_\rho$ with respect to S is equivalent to projecting the matrix $C - \sum_{i=1}^{n} \mathrm{y}_i A_i - \rho X$ onto the semidefinite cone $\mathbb{S}_+^m$.*

Proof. For optimizing the function $\mathcal{L}_\rho$ with respect to S, we can simply consider the function without the terms that do not contain S. So, the minimization problem would be written as

$$\begin{aligned}\min_S \ \mathcal{L}_\rho(\mathbf{y}, S, X) &\approx X \bullet S + \frac{1}{2\rho} \left\| \sum_{i=1}^{n} \mathrm{y}_i A_i + S - C \right\|_F^2 \\ &= X \bullet S + \frac{1}{2\rho} (\sum_{i=1}^{n} \mathrm{y}_i A_i + S - C) \bullet (\sum_{i=1}^{n} \mathrm{y}_i A_i + S - C).\end{aligned}$$

Once again, by focusing on the terms involving S, we obtain

$$\approx X \bullet S + \frac{1}{2\rho} \left(2(S \bullet \sum_{i=1}^{n} \mathrm{y}_i A_i) + S \bullet S - 2S \bullet C \right).$$

Now, we multiply both sides of this function by 2ρ:

$$\begin{aligned}&= 2\rho X \bullet S + 2S \bullet \sum_{i=1}^{n} \mathrm{y}_i A_i + S \bullet S - 2S \bullet C \\ &\approx (S - C + \sum_{i=1}^{n} \mathrm{y}_i A_i + \rho X) \bullet (S - C + \sum_{i=1}^{n} \mathrm{y}_i A_i + \rho X).\end{aligned}$$

We can rewrite the last equation using the Frobenius norm, and we arrive at the following projection problem:

$$\min_S \left\| S - C + \sum_{i=1}^{n} \mathrm{y}_i A_i + \rho X \right\|_F^2.$$

□

For $Z \in \mathbb{S}^m$, let $\mathcal{A}(Z)$ be a vector with elements $A_i \bullet Z$, $i = 1, \ldots, n$ and $[Z]_+$ the projection of Z on the semidefinite cone. With a penalty parameter $\rho > 0$ and a relaxation parameter $\sigma \in (0, \frac{1+\sqrt{5}}{2})$, the update of the primal-dual point $(\mathbf{y}, S, X)$ in the k-th iteration of ADMM is defined in Algorithm 5.

Algorithm 5 ADMM for SDP

Given an initial point $(\mathbf{y}^{(0)}, S^{(0)}, X^{(0)})$, parameters ρ and σ, and a stopping parameter $\varepsilon_{\text{ADMM}}$.

1: **for** $k = 0, 1, 2, \ldots$ **do**

2: Update $\mathbf{y}$ by $\mathbf{y}^{(k+1)} = -(\mathbf{A}\mathbf{A}^\top)^{-1}\left(\rho(\mathcal{A}(X^{(k)}) - b) + \mathcal{A}(S^{(k)} - C)\right)$

3: Update S by $\widehat{S} = C - \sum_{i=1}^n \mathrm{y}_i^{(k+1)} A_i - \rho X^{(k)}, \quad S^{(k+1)} = \left[\widehat{S}\right]_+$

4: Update X by $\widehat{X} = \frac{1}{\rho}\left(S^{(k+1)} - \widehat{S}\right), \quad X^{(k+1)} = (1-\sigma)X^{(k)} + \sigma\widehat{X}$

5: Update ρ

6: Check convergence

7: **end for**

For details, in particular the choice and update of ρ, see [82]. In the numerical experiments below, we use our MATLAB implementation[1] of the algorithm.

7.3.2 ADMM as a warm start for Loraine

When testing the ADMM algorithm with the MAXCUT problems considered in this thesis, we have made the following two observations:

(i) The method is more efficient when solving the reformulated (bigger) problem (7.9) than the smaller one (7.4). Moreover, for problem (7.9), the convergence is very fast in the first iterations, before it slows down: the algorithm gets quickly a “good" approximation of the solution, both primal and dual.

[1] https://github.com/kocvara/ADMM_for_SDP

(ii) Due to the nature of the ADMM method (projections on the semidefinite cone), the rank of the approximate solution is always "numerically exact"; in particular, the rank of $X^{(k)}$ is low when we are close enough to the solution.

The fact that a low-rank approximation of the solution can be obtained relatively quickly, together with the fact that ADMM is a primal-dual algorithm leads to the following idea: use a low-precision ADMM solution as a warm start for Loraine. Presuming that the ADMM approximation of X is already of the expected low rank (or very close to it), the preconditioner H_α will be extremely efficient during the remaining iterations of Loraine. We, therefore, propose the following *hybrid ADMM-Loraine* Algorithm 6.

Algorithm 6 ADMM-Loraine

1: Solve (7.9) by ADMM with stopping tolerance $\varepsilon_{\text{ADMM}}$. Save the primal-dual approximate solution $(\mathbf{y}, S, X)_{\text{ADMM}}$.

2: Solve (7.9) by Loraine with

- initial point $(\mathbf{y}, S, X)_{\text{ADMM}}$
- initial value of the stopping criterion for the CG method reduced to 10^{-6}.

Notice that, in order to get a primal-dual approximation of the solution, we have to solve the same problem formulation by ADMM and Loraine, i.e., formulation (7.9) with the ℓ_1-penalty and the same value of the penalty parameter μ, despite the fact that ADMM could directly handle the linear equality constraints.

7.4 Numerical experiments

In this section we report the results of our numerical experiments with relaxations of MAXCUT problems, randomly generated UBQPs and a library of problems called BQPGKA. The presented

algorithms will be compared with the state-of-the-art SDP solver MOSEK [57] which is relying on the interior-point method and uses a direct linear solver for the Schur complement equation.

The stopping criterion for our implementations of Loraine and ADMM was based on the DIMACS errors [56] that measure the (normalized) primal and dual feasibility, duality gap, and complementary slackness. The algorithms were stopped when all these errors were below 10^{-6}. MOSEK was used with default settings.

7.4.1 MAXCUT problems

For the first set of numerical experiments, we use MAXCUT problems, the standard test problems for UBQP algorithms. These problems are well-known optimization problems in which the vertices of an undirected graph are partitioned into two disjoint sets in such a way that the number of edges that cross the partition is maximized. In other words, the task is to find a cut of the largest weight, that is, to partition the set of nodes into two parts, S and S' such that the total weight of all arcs linking S and S' is as large as possible.

Let Γ be an undirected n-node graph and let the arcs (i, j) be associated with nonnegative weights a_{ij}. The MAXCUT problem is formulated as the following UBQP:

$$\max_{\mathbf{x}} \left\{ \frac{1}{4} \sum_{i,j=1}^{n} a_{ij}(1 - \mathrm{x}_i \mathrm{x}_j) \mid \mathrm{x}_i^2 = 1,\ i = 1, \ldots, n \right\} \tag{7.14}$$

and can be thus solved by the techniques presented above. The solution of the SDP relaxation (7.2) is of rank two whenever the relaxation is exact and the solution to (7.14) is unique. (Recall that uniqueness of $\mathbf{x}$ means uniqueness up to the multiple by -1.)

Obviously, a complete unweighted graph (with $a_{ij} \in \{0, 1\}$) may have many “symmetric" solutions. Therefore we generated two sets of problems of increasing size. Firstly, to avoid the non-uniqueness, we generated undirected, weighted, generally complete graphs with weights randomly

distributed between 0 and 12 with 20–50 nodes, using the MATLAB command `graph`. It turned out that for all these problems the relaxation order two in the Lasserre hierarchy is already high enough to deliver the optimal solution of the MAXCUT problem (7.14). Secondly, we generated a set of problems of the same sizes but for unweighted graphs. The dimensions of the corresponding SDP problems for relaxation order two are shown in Table 7.1.

Table 7.1: Problems `MAXCUT-<n>`*, relaxation order* $\omega = 2$*: dimensions for formulations* (7.4) *and* (7.9)

	problem (7.4)		problem (7.9)		
problem	variables	matrix size	variables	matrix size	lin. constr.
MAXCUT-20	6 196	211	28 561	211	12 390
MAXCUT-25	15 275	326	68 576	326	30 550
MAXCUT-30	31 930	466	140 741	466	63 860
MAXCUT-35	59 535	631	258 931	631	119 070
MAXCUT-40	102 090	821	439 521	821	204 180
MAXCUT-45	164 220	1 036	701 386	1 036	328 440
MAXCUT-50	251 175	1 276	1 065 901	1 276	502 350

We first present the results for the weighted problems, see Table 7.2. These problems are relatively "simple": the ADMM method is rather efficient (this is well-known from other studies), and MOSEK converges in a very small number of iterations, six to eight. However, the sheer size prevents MOSEK with a direct solver to solve larger problems. We present results for ADMM applied to both problem formulations, the original one (7.4) and the re-written one of larger dimension and with ℓ_1-penalty (7.9). Perhaps surprisingly, ADMM is more efficient for the latter problem, despite the bigger dimension. Moreover, unlike for ADMM applied to (7.4), the number of iterations of ADMM applied to (7.9) is almost independent of the problem size. The last four columns of Table 7.2 show the results for the hybrid ADMM-Loraine algorithm: we present the number of iterations of ADMM plus Loraine ('iter'), the time of ADMM ('timeA'), time of Loraine ('timeL') and the total time of the hybrid method. For these experiments, we have used the

stopping tolerance of ADMM $\varepsilon_{\text{ADMM}} = 5 \cdot 10^{-3}$ (problems 20–25) and $\varepsilon_{\text{ADMM}} = 5 \cdot 10^{-4}$ (problems 30–50).

Table 7.2: Loraine, MOSEK, ADMM and ADMM-Loraine in weighted `MAXCUT-<n>` *problems, relaxation order* $\omega = 2$

MAXCUT	Loraine for (7.9)			MOSEK (7.4)	ADMM for (7.4)		ADMM for (7.9)		ADMM-Loraine for (7.9)			
problem	iter	CG it	time	time	iter	time	iter	time	iter	timeA	timeL	time
20	18	559	3.8	9	3615	10	3042	13	628+4	3.9	0.8	4.7
25	20	728	11	78	4732	33	2735	28	181+5	2.5	2.9	5.4
30	21	1032	28	607	6770	99	3537	80	795+5	20	5.8	26
35	23	2183	96	2911	5255	164	3030	126	863+4	42	11	53
40	27	2275	186	mem	9611	500	1280	92	914+7	73	132	205
45	25	2521	335		16901	1400	2639	358	755+4	104	33	137
50	24	2540	528		19521	2951	3296	745	727+5	162	58	220

To better demonstrate the behavior of the hybrid method, in Figure 7.1 we show the output of both codes. We can see that the ADMM method, indeed, gets very quickly a good approximation of objective value, while Loraine uses the warm start very efficiently.

Next, we will try to solve the unweighted MAXCUT problems. Recall that these problems often have nonunique solutions, in particular problems with almost dense graphs. Moreover, relaxation order $\omega = 2$ may not be high enough to obtain an exact solution. We thus cannot expect the preconditioner to be efficient in those cases. This is, indeed, demonstrated in Table 7.3. This table, in addition to columns identical to Table 7.2, shows also the rank of the solution of the order-2 relaxation. As expected, Loraine is less efficient for problems with higher solution rank, in particular problems `MAXCUT-30` and `MAXCUT-50`. Still, it can solve these problems reliably. Again, the hybrid method is superior for these problems. The convergence behavior of the algorithms and their estimated complexity is further illustrated in Figure 7.2.

```
================================ A D M M  for  S D P =================
Number of LMI Constraints:   1
Number of Variables: 1065901
Maximal Constraint Size: 1276
------------------------------------------------------------------------
 iter    p-infeas     d-infeas       d-gap        error      objective
------------------------------------------------------------------------
  100   0.00058292   0.08603023   0.00093126    0.086030   1981.22444814
  200   0.00018776   0.02554937   0.00034476    0.025549   2028.80448106
  300   0.00018959   0.01054250   0.00063882    0.010542   2037.35317708
  400   0.00010212   0.00991976   0.00025128    0.009920   2025.69453823
  500   0.00005383   0.00972192   0.00013810    0.009722   2038.21968191
  600   0.00002508   0.00891292   0.00001945    0.008913   2026.90960413
  700   0.00002609   0.00281354   0.00010411    0.002814   2037.33886004
  727   0.00002899   0.00047182   0.00012837    0.000472   2039.90970508
------------------------------------------------------------------------
Total ADMM iterations: 727; Final precision: 4.72e-04; CPU Time 162.59s
========================================================================

 *** Loraine v0.1 ***
 Number of variables: 1065901
 Matrix size(s)     :    1276
 Linear constraints :  502350
 *** IP STARTS
 it      objective      error   cg_iter   CPU/it
  1   2.03994277e+03  3.77e-03       36     9.39
  2   2.03999424e+03  2.32e-03       33     9.55
  3   2.03999941e+03  2.50e-04       20     7.71
  4   2.03999992e+03  3.84e-05        9     5.80
  5   2.03999999e+03  5.96e-06        8     5.54
*** Total CG iterations:    106
*** Total CPU time:         58.24 seconds
```

Figure 7.1: The printout of low-precision ADMM and warm-started Loraine for the weighted `MAXCUT-50`.

Table 7.3: Loraine, MOSEK, ADMM and ADMM-Loraine in unweighted `MAXCUT-<n>` *problems, relaxation order* $\omega = 2$

MAXCUT	Loraine for (7.9)			MOSEK (7.4)	ADMM for (7.9)		ADMM-Loraine for (7.9)				
problem	iter	CG it	time	time	iter	time	iter	timeA	timeL	time	rank
20	17	735	4.3	9	1981	8	738+7	3.4	1.5	4.9	2
25	18	858	14	82	805	9	328+4	3.5	2.8	6.3	4
30	23	6482	145	635	2218	53	566+8	13	135	148	57
35	21	2604	128	3357	5251	229	1222+8	48	15	63	2
40	22	3538	305	mem	7137	575	1855+7	142	35	177	2
45	22	3225	536		6794	845	1081+4	139	49	188	4
50	25	12599	2593		5723	1162	1677+5	361	365	726	13

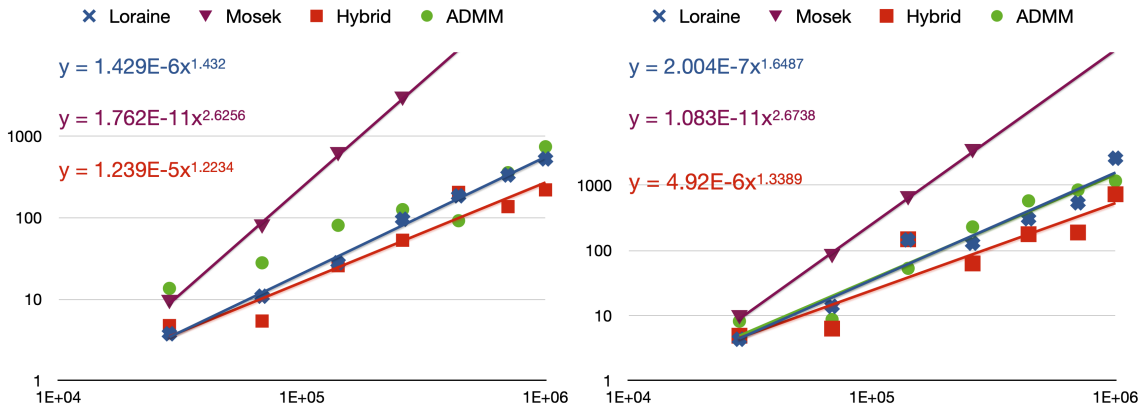


Figure 7.2: CPU times in log-log scale for weighted (left) and unweighted (right) MAXCUT problems. Loraine (blue), MOSEK (brown), ADMM (green) and hybrid ADMM-Loraine (red).

7.4.2 Randomly generated problems

We further conducted numerical experiments with matrices Q randomly generated. We are aware of the fact that problems with random data may not always be representative and may sometimes lead to false conclusions regarding algorithm behavior. However, we believe that these results still well demonstrate the efficiency of our approach.

It has been observed in [37] that problems with rank-one matrix Q may require relaxation order of up to $\omega = \lceil s/2 \rceil$ to reach the exact solution; a typical example is $Q = ee^\top$. On the other hand, for problems with matrix Q of rank 3, $\omega = 2$ was always sufficient in experiments performed in ([37, Fig.4]). We have thus considered two main classes of problems:

(A) problems with $\mathcal{B} = \{-1, 1\}$ and with $\operatorname{rank} Q = 1$ generated by the following MATLAB code

```
rng(0); q = randn(s,1); Q = q*q';
```

(B) problems with $\mathcal{B} = \{-1, 1\}$ and with a full-rank indefinite Q generated by the following MATLAB code

```
rng(0); q = randn(s,1); Q = q*q';
for k=1:s-1
    if ceil(k/2)*2 == k
        q = randn(s,1); Q = Q - q*q';
    else
        q = randn(s,1); Q = Q + q*q';
    end
end
```

Apart from random numbers generated with normal distribution, we also performed tests with uniform distribution (function `rand`) and lognormal distribution (function `logncdf`); in both cases the results and conclusions were rather similar to the above choice and are thus not reported here.

The exactness of the relaxation was measured by the numerical rank of the dual solution to the matrix inequality in problem (7.9)—when the rank was equal to 1 or 2 (depending on the set $\mathcal{B}$) the relaxation order was considered sufficient; see (7.2.5).

Remark 7.4.1. *For Q constructed as in (B), relaxation order $\omega = 2$ was sufficient to get an exact solution of* (7.1). *This observation, though, cannot be extended to any full-rank matrix Q. For instance, for $Q = ee^\top + \mathrm{Diag}(d), d \in \mathbb{R}^s$, where $Diag(d)$ is a diagonal matrix by placing vector d along the main diagonal, the lowest relaxation order will be $\omega = \lceil s/2 \rceil$. This is because $x_i^2 = 1$ and thus the diagonal elements of Q will be irrelevant in the optimization process and the resulting problem will be equivalent to that with a rank-one matrix $Q = ee^\top$.*

Full-rank Q and relaxation order $\omega = 2$

Using the MATLAB code from point (B) above, we generated problems of growing dimension $s = 10 \ldots 50$ and solved the corresponding SDP relaxations for order $\omega = 2$. The dimensions of the generated problems are reported in Table 7.4; the table shows problem sizes for the original SDP relaxation (7.4) and for the re-written problem (7.9). The computational results are presented in Table 7.5 and clearly demonstrate the efficiency of Loraine. The computational complexity of Loraine (applied to (7.9)) and MOSEK and ADMM (applied to (7.4)) is further illustrated in Figure 7.3. We do not report on the hybrid ADMM-Loraine algorithm. That is because ADMM applied to problem (7.9) (as required by the hybrid method) appears to be much less efficient than for the MAXCUT problems and so the hybrid algorithm is not even competitive to ADMM applied to (7.4).

Table 7.4: Randomly generated UBQP problems, relaxation order $\omega = 2$: dimensions for formulations (7.4) *and* (7.9)

	problem (7.4)		problem (7.9)		
UBQP size	variables	matrix size	variables	matrix size	lin. constraints
10	385	56	1596	56	770
15	1 940	121	7 381	121	3 880
20	6 195	211	22 366	211	12 390
25	15 275	326	53 301	326	30 550
30	31 930	466	108 811	466	63 860
35	59 535	631	199 369	631	119 070
40	102 090	821	337 431	821	204 180
45	164 220	1 036	537 166	1 036	328 440
50	251 175	1 276	814 726	1 276	502 350

Table 7.5: Randomly generated UBQP problems, relaxation order $\omega = 2$: Loraine, MOSEK and ADMM

	Loraine for (7.9)			MOSEK for (7.4)		ADMM for (7.4)	
UBQP size	iter	CG iter	time	iter	time	iter	time
10	10	256	0.1	6	0.2	1006	0.4
15	10	725	1.3	5	1.1	1928	2.2
20	11	423	2.4	6	9.3	2888	9.7
25	13	294	5.5	7	81	5622	48
30	13	326	14	7	496	8498	147
35	15	530	43	memory		11672	396
40	14	730	106			14362	880
45	16	896	230			14320	1369
50	16	1114	431			21701	3251

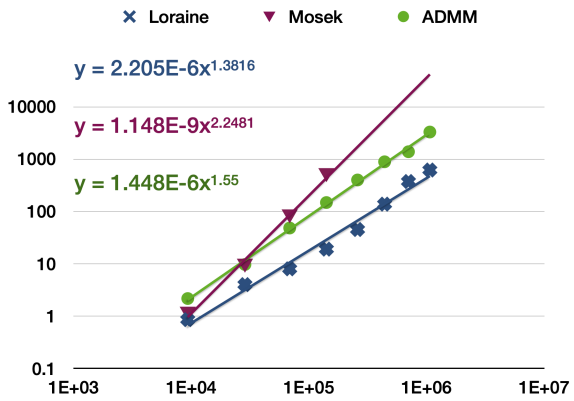


Figure 7.3: CPU times in log-log scale for random BQP problems. Loraine (blue), MOSEK (brown) and ADMM (green).

Rank-one Q and higher relaxation order

Let us recall from (7.5) the dependence of problem sizes on the relaxation order: for the original moment problem (7.4), we have

$$\frac{s^4}{24} \lessapprox n \leq 2^s - 1 \quad \text{and} \quad m = \sum_{i=1}^{\omega} \binom{s}{i}.$$

On the other hand, in the re-written problem (7.9) with ℓ_1-penalty term, we have $\widetilde{n}+n$ variables with $\widetilde{n} = m^2$, LMI of size m and $2n$ linear inequality constraints; see Table 7.6. This table, in

Table 7.6: Problem dimensions for formulations (7.4) *and* (7.9) *as functions of the relaxation order; here s is the number of variables in BQP* (7.1) *and ω the relaxation order and* $\frac{s^4}{24} \lessapprox n \leq 2^s - 1$

problem (7.4)		problem (7.9)		
variables	matrix size	variables	matrix size	lin. constraints
n	$\sum_{i=1}^{\omega} \binom{s}{i}$	$\left(\sum_{i=1}^{\omega} \binom{s}{i}\right)^2 + n$	$\sum_{i=1}^{\omega} \binom{s}{i}$	$2n$

particular, shows that the number of variables grows much more quickly for problem (7.9) than for the original problem (7.4) where it, eventually, reaches the finite limit $2^s - 1$. Therefore we cannot expect Loraine with iterative solver applied to (7.9) to be as efficient for higher-order relaxations

as it is for $\omega = 2$. This is clearly demonstrated in Table 7.7 comparing Loraine with a direct solver applied to problem (7.4) (this code was slightly faster than MOSEK for these problems, due to direct handling of the rank-one data matrices) with Loraine with the iterative solver applied to problem (7.9).

Table 7.7: Randomly generated UBQP problem, $n = 9$, relaxation order $\omega = 2, \ldots, 5$: problem dimensions and CPU times for Loraine-direct and Loraine-iterative

	Loraine-direct for (7.4)			Loraine-iterative for (7.9)				
ω	vars	matrix size	time	vars	matrix size	lin constr	CG iter	time
2	255	46	0.09	1081	46	510	99	0.08
3	465	130	0.77	8515	130	930	203	0.97
4	510	256	2.87	32896	256	1020	302	15.7
5	511	382	7.04	73153	382	1022	701	129

7.4.3 BQPGKA problems

Finally, we consider test problems from the literature. The next Table 7.8 presents results for binary problems with $\mathcal{B} = \{0, 1\}$ and randomly generated integer matrices Q, as provided by [60]; in the literature these problems are often in the literature called BQPGKA by [24]. We only selected problems with fully occupied matrices Q and those for which second-order relaxations are solvable by our software; i.e., problems 1b–4b, in the notation of [60]. Dimensions of the SDP problems are the same as in Table 7.4. From the results, we can again see the clear advantage of the iterative solver and the linear growth of CPU time per one CG iteration with the number of (SDP) variables.

Just as with the randomly generated problems with full-rank matrix above, relaxation order $\omega = 2$ was always sufficient to reach solution with a moment matrix of rank one and with optimal

objective value equal to that in [24].

Table 7.8: BQPGKA problems, relaxation order $\omega = 2$: Mosek and Loraine

	Mosek	Loraine			
size	CPU	iter	CG iter	CPU	cpu/CG iter
20	11	20	666	3	0.0045
30	629	15	835	24	0.0287
40	mem	14	769	82	0.1066
50		14	862	270	0.3132

Chapter Eight

Geometric modeling problems

In this chapter, we review applications of Polynomial Optimization techniques to Geometric Modeling problems. We present examples of topical problems in Geometric Modeling, illustrate their solution using Polynomial Optimization Tools, report some experimental results and analyse the behavior of the methods, showing what are their strengths and their limitations. Our new approach addresses the Minimal Enclosing Ellipsoids problem, which involves finding the ellipsoid with the smallest volume that fully contains a given polynomial patch. While existing research has focused on determining this ellipsoid when the center point is predefined (centered at the origin), our main contribution involves modeling the ellipsoid when the center point is not predefined, and it is treated as a free variable.

8.1 Geometric Modeling and Polynomials

Geometric modeling aims at describing shapes of the real world by digital models. These digital models are used in many domains: visualization in scientific computing, rendering and animation, design and conception, manufacturing, numerical simulation, analysis of physics phenomena, mechanics, performance optimization ...

Most of the digital models used nowadays in Computer Aided Design (CAD) involve models based on spline representations, which are piecewise polynomial functions with global regularity properties [21]. Among them, the representation or approximation of shapes by meshes is certainly the most popular and corresponds to piecewise linear splines. Higher order splines allow to increase the accuracy of the representation and the efficiency to describe complex shapes. By increasing the degree of the polynomials of the representation, one needs less pieces and thus less data for the same level of approximation. The “unreasonable" power of polynomial approximation makes piecewise polynomial representations ubiquitous in Geometric Modeling. Standard digital representations of shapes correspond to the image of simple domains like triangles, squares, cubes by piecewise polynomial maps, which define so called patches. These patches are trimmed and assembled together to define the boundary surface or the volume of an object.

Although piecewise polynomial models are very effective in representing complex shapes, they also require advanced methods and tools in practice. Many operations on shapes, such as patch intersections, distance computation, boundary volumes, etc., involve solving nonlinear and difficult problems [20]. In what follows, we illustrate how polynomial optimization methods can help solve these issues and discuss their practical performance. See also [55] for other examples of applications of polynomial optimization in geometric modeling.

8.2 Polynomial Optimization Problems and Convex Relaxations

As mentioned in Chapter 3, polynomial optimization problems are problems (3.2). Throughout this chapter, we consider the feasible semi-algebraic set (3.21) Containing both equality and inequality constraints which is

$$S = \{\mathbf{x} \in \mathbb{R}^n \text{ s.t. } \quad \mathrm{g}_j(\mathbf{x}) \geq 0,\ j = 1, \ldots, m_1, \quad h_l(\mathbf{x}) = 0,\ l = 1, \ldots, m_2\}$$

defined by the sign constraints $\mathbf{g} = (\mathrm{g}_1, \ldots, \mathrm{g}_{m_1})$ and the equality constraints $\mathbf{h} = (h_1, \ldots, h_{m_2})$, for polynomial functions $f, \mathrm{g}_j, h_l : \mathbb{R}^n \to \mathbb{R},\ j = 1, \ldots, m_1,\ l = 1, \ldots, m_2$. From now on to avoid confusion, we show this general set by S.

We also described SOS and moment relaxations in Chapter 3. In the following sections, we recall these problems for this general S, and we present examples of the construction of these relaxations and computing the minimizers using Julia packages `MomentTools`[1] and `DynamicPolynomials`.

8.2.1 Sum of Squares Relaxations

The SOS relaxation uses the following truncated quadratic module to approximate the solution of (3.2) on the set S:

$$\mathcal{Q}(\mathrm{g})_d := \Big\{ \sum_{j=0}^{m_1} \sigma_j \mathrm{g}_j + \sum_{l=1}^{m_2} \psi_l h_l : \sigma_j \in \Sigma[\mathbf{x}]_{2d-v_j}, \psi_l \in \mathbb{R}[\mathbf{x}]_{2d-\tau_l},\ j = 0, \ldots, m_1, l = 1, \ldots, m_2 \Big\},$$

in which g_j are the non-negativity polynomials of degree v_j and h_k are the equality polynomials of degree τ_l. Also, we take $g_0 = 1$ and $v_0 = 0$ for notation convenience.

In this relaxation, the solution of (3.2) will be approximated by the solution of the following convex optimization problem:

$$\begin{aligned} f_{\mathrm{sos},d} = \sup_{\lambda, \mathbf{x} \in S}\ & \lambda \\ \text{s.t.}\quad & \lambda \in \mathbb{R} \\ & f(\mathbf{x}) - \lambda \in \mathcal{Q}(\mathrm{g})_d\,. \end{aligned} \tag{8.1}$$

Let $\mathrm{v}_d(\mathbf{x})_j$ denote a basis of $\mathbb{R}[\mathbf{x}]_{\frac{2d-v_j}{2}}$ and $\mathbf{w}_l$ a basis of $\mathbb{R}[\mathbf{x}]_{2d-\tau_l}$ of dimension N_l. Then, by using the definition of the truncated quadratic module and Proposition 3.2.3, (8.1) can be written as a

[1] https://gitlab.inria.fr/AlgebraicGeometricModeling/MomentTools.jl

semidefinite program of the form

$$
\begin{aligned}
f_{\text{sos},d} = \sup\ & \lambda \\
\text{s.t.}\quad & \lambda \in \mathbb{R},\ A_j \succcurlyeq 0,,\ B_l \in \mathbb{R}^{N_l} \\
& f(\mathbf{x}) - \lambda - \sum_{j=1}^{m_1} g_j(\mathbf{x}) \mathrm{v}_d(\mathbf{x})_j^T A_j \mathrm{v}_d(\mathbf{x})_j - \sum_{l=1}^{m_2} h_l(\mathbf{x}) \mathbf{w}_l^T B_l = 0\,,
\end{aligned}
$$

where $A_j \in \mathbb{R}^{s(d-\frac{v_j}{2})\times s(d-\frac{v_j}{2})}$ is positive semidefinite. The last polynomial constraint can be written as

$$
\sum_{\alpha \in \mathbb{N}^n_{2d}} c_\alpha(\lambda, A_1, \ldots, A_{m_1}, B_1, \ldots, B_{m_2}) \mathbf{x}^\alpha = 0
$$

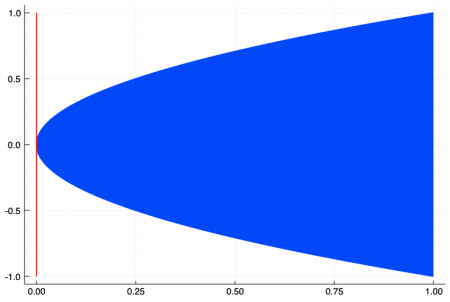
where as mentioned before $(\mathbf{x}^\alpha)_{\alpha \in \mathbb{N}^n_{2d}}$ is the monomial basis of $\mathbb{R}[\mathbf{x}]_d$.

Example 8.2.1. ***Example of a Sum-of-Squares relaxation***

We consider the semi-algebraic set S defined by

$$
\begin{aligned}
\mathbf{g} &= \{1 - x, x - y^2\} \\
\mathbf{h} &= \{\}
\end{aligned}
$$

corresponding to the blue domain on the adjacent figure. The objective function is $f = x$, corresponding to the vertical red line.



We construct a SoS relaxation at order $d = 2$. Here is how it can be constructed with the Julia packages `MomentTools` *and* `DynamicPolynomials`.

```
> using MomentTools, DynamicPolynomials
> X = @polyvar x y
> M = SOS.Model(:inf, x, [], [1-x, x-y^2], X, 2)
```

This gives a semidefinite program with three matrix variables $A_0 \in \mathbb{R}^{6\times 6}$ for $s_0 \in \Sigma[\boldsymbol{x}]_4$, $A_1 \in \mathbb{R}^{3\times 3}$ for $s_1 \in \Sigma[\mathbf{x}]_2$, $A_2 \in \mathbb{R}^{3\times 3}$ for $s_2 \in \Sigma[\mathbf{x}]_1$ and no B_l since $\mathbf{h} = \{\}$. There are 15 linear constraints corresponding to the coefficients of the 15 monomials of degree ≤ 4 in the variables x, y.

8.2.2 Moment Relaxations

The moment relaxation of problem (3.2) on the feasible set S is the following SDP:

$$\begin{aligned} \rho_d^{\text{mom}} = \inf_{\mathrm{y}} \quad & L_{\mathrm{y}}(f) \\ \text{s.t.} \quad & \mathrm{M}_d(\mathrm{y}) \succeq 0 \\ & \mathrm{M}_{d-v_j}(\mathrm{g}_j \mathrm{y}) \succeq 0,\ j = 1, \ldots, m_1 \\ & \mathrm{M}_{d-\tau_l}(h_l \mathrm{y}) = 0,\ j = 1, \ldots, m_2 \\ & y_0 = 1 \end{aligned} \tag{8.2}$$

where as mentioned before, $\mathrm{M}_d(\mathrm{y})$ and $\mathrm{M}_{d-v_j}(\mathrm{g}_j \mathrm{y})$ are the moment matrix and localizing matrix, respectively, associated with y and the polynomial g_j, and L_{y} is the Riesz linear functional represented in the basis $\mathbf{x}^\alpha$ of $\mathbb{R}[\mathbf{x}]_{2d}$ as described in Definition 3.2.9.

Example 8.2.2. ***Example of a Moment Relaxation***

The moment relaxation at order $d = 2$ for the previous example where $S = \{(x, y) \in \mathbb{R}^2 \mid 1 - x \geq 0, x - y^2 \geq 0\}$ and the objective function is $f = x$ is built as follows:

```
> using MomentTools, DynamicPolynomials
> X = @polyvar x y
> M = MOM.Model(:inf, x, [], [1-x, x-y^2], X, 2)
```

It is a convex optimization program on the moment sequence $L_{\mathrm{y}} \in \mathbb{R}^{15}$ with a linear objective function $L_{\mathrm{y}}(f)$ (as a function of L_{y}) and with SDP constraints on moment matrices in $\mathbb{R}^{6\times 6}, \mathbb{R}^{3\times 3}, \mathbb{R}^{3\times 3}$.

8.2.3 Computing the minimizers

This approach is explained in Section 3.2.3 to extract the global minimizers of (3.2), and the behavior of it on different geometric problems is illustrated using the package `MomentTools` as we can see in the next example.

Example 8.2.3. ***Example of minimizer computation***
We continue with the previous example, and solve the moment relaxation at order $d = 2$, using convex optimization tools from the `Mosek`[2] *library,*

```
> using MomentTools, DynamicPolynomials, MosekTools
> X = @polyvar x y
> v, M = minimize(x, [], [1-x, x-y^2], X, 2, Mosek.Optimizer)
```

This minimizer computation can be done directly as follows:

```
w, Xi = get_measure(M)
```

which yields the following weight ω and point Ξ for the approximation of L_y^* as a weighted sum of Dirac measures:

$$\omega = \begin{bmatrix} 1.0 \end{bmatrix}, \qquad \Xi = \begin{bmatrix} 0.0 \\ -0.0 \end{bmatrix},$$

where $L_y^* = (L_{y_\alpha}^*)$ and is the optimal sequence of pseudo-moments obtained from solving the convex optimization problem (3.22)*.*

In the following, we consider Example 3.2.20 and we show that the results of solving this problem by using Julia packages are the same as the ones reached by using Algorithm 3.

Example 8.2.4. *We solve the moment relaxation of the problem in Example 3.2.20 at order $d = 2$, using convex optimisation tools from the* `Mosek` *library where $f = -(x-1)^2-(x-y)^2-(y-3)^2$*

[2]https://github.com/MOSEK/Mosek.jl

and $S = \{(x, y) \in \mathbb{R}^2 \mid 1-(x-1)^2 \geq 0,\ 1-(x-y)^2 \geq 0,\ 1-(y-3)^2\}$ *which includes* $h = \{\}$ *and* $g = \{1-(x-1)^2, 1-(x-y)^2, 1-(y-3)^2\}$.

```
> using MomentTools, DynamicPolynomials, MosekTools
> X = @polyvar x y
> f = -(x-1)^2-(x-y)^2-(y-3)^2
> g = [1-(x-1)^2, 1-(x-y)^2,1-(y-3)^2]
> v, M = minimize(f, [], g, X, 2, Mosek.Optimizer)
```

This minimizer computation follows:

```
w, Xi = get_measure(M)
```

which yields the following weight ω *and point* Ξ *which are explained before*

$$\omega = \begin{bmatrix} 0.6299901086515592 \\ 0.2662556048855402 \\ 0.10375428631094992 \end{bmatrix}, \qquad \Xi = \begin{bmatrix} 1.0 & 2.0 & 2.0 \\ 2.0 & 2.0 & 3.0 \end{bmatrix}.$$

Here each columns of Ξ *indicate one of the global solutions.*

Solution of linear semidefinite optimization problems resulting from SoS or Moment relaxations is often the bottleneck of the approach. With increasing order of the relaxation, the size of the SDP problem grows very quickly. While the “small" problems can be solved by general-purpose SDP solvers such as `Mosek`, larger problems are beyond their reach. It is therefore necessary to use an algorithm and software exploiting the particular structure of SoS relaxations. One of the features of these SDP problems is the very low rank of the solution (moment matrix). This feature makes our SDP solver `Loraine`, explained in Chapter 4, a good candidate for solving these problems if all the assumptions are satisfied.

8.3 Minimal enclosing ellipsoids of semi-algebraic sets

In this section, we consider another type of optimization problem, which appears in geometric modeling, namely computing the minimal ellipsoid enclosing a semi-algebraic set. Let S be a semi-algebraic set defined as in Section 8.2:

$$S = \{\mathbf{x} \in \mathbb{R}^n \text{ s.t. } \quad \mathrm{g}_j(\mathbf{x}) \geq 0,\ j = 1, \ldots, m_1, \quad h_l(\mathbf{x}) = 0,\ l = 1, \ldots, m_2\}\,.$$

Let $\xi_i(\mathbf{x}),\ i = 1, \ldots, \nu$, be given polynomial functions. Define the set

$$\hat{S} = \{\boldsymbol{\eta} \in \mathbb{R}^\nu \text{ s.t. } \quad \eta_i = \xi_i(\mathbf{x}),\ i = 1, \ldots, \nu, \quad \mathbf{x} \in S\}\,.$$

This set typically represents a parametric subset of $\mathbb{R}^n$ described by parameters x; see the next section.

Assume that $\hat{S}$ is bounded in $\mathbb{R}^\nu$. We are looking for a smallest-volume ("minimal") ellipsoid E that is enclosing the set $\hat{S}$, i.e., all feasible positions of the ν-dimensional point $\boldsymbol{\xi}(\mathbf{x}) = (\xi_1(\mathbf{x}), \ldots, \xi_\nu(\mathbf{x}))$.

It is well known that finding a minimal enclosing ellipsoid of a set of points $S_P = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\} \subset \mathbb{R}^\nu$ amounts to solving a semidefinite optimization problem; see, e.g., [8, 73]. Let us first recall this formulation.

Assume that the convex hull of S_P has a nonempty interior. We consider an $n-$dimensional ellipsoid represented by a strictly convex quadratic inequality with $D \in \mathbb{S}^\nu, D \succ 0$:

$$E = \{\mathbf{y} \in \mathbb{R}^\nu \mid (\mathbf{y} - \mathbf{c})^T D(\mathbf{y} - \mathbf{c}) \leq 1\}\,.$$

The volume of such an ellipsoid is

$$\mathrm{Vol}(E) = (\det(D))^{-\frac{1}{2}}\,,$$

so minimizing it amounts to maximizing $\det(D)$ and, further, to maximizing $\log\det(D)$, as D is assumed to be positive definite.

The minimal ellipsoid enclosing the set S_P can be computed from the solution of the following convex semidefinite optimization problem

$$\begin{aligned} \sup_{Z\in\mathbb{S}^\nu,\,\mathbf{z}\in\mathbb{R}^\nu,\,\gamma\in\mathbb{R}} \quad & \log\det(Z) \\ \text{s.t.} \quad & 1-(\mathbf{x}^{(i)})^T Z\mathbf{x}^{(i)}+2\mathbf{z}^T\mathbf{x}^{(i)}-\gamma\geq 0, \quad i=1,\ldots,m\,, \\ & \begin{bmatrix} \gamma & \mathbf{z}^T \\ \mathbf{z} & Z \end{bmatrix} \succcurlyeq 0\,. \end{aligned} \tag{8.3}$$

where $\gamma=\mathbf{c}^T D\mathbf{c}$. Let $(Z^*,\mathbf{z}^*,\gamma^*)$ be the solution of the above problem, then the minimal ellipsoid containing S_P is given by

$$E=\{\mathbf{y}\in\mathbb{R}^n \mid (\mathbf{y}-\mathbf{c})^T D(\mathbf{y}-\mathbf{c})\leq 1\}$$

where $D=Z^*$ and $\mathbf{c}=D^{-1}\mathbf{z}^*$ (see, e.g., [8, Prop. 4.9.2]).

For completeness, we derive the dual to SDP (8.3) (see also [77]).

Lemma 8.3.1. *The dual problem to (8.3) reads as*

$$\begin{aligned} \min_{Y\in\mathbb{S}^\nu,\,\boldsymbol{\lambda}\in\mathbb{R}^m} \quad & \sum_{i=1}^m \lambda_i - \log\det(Y) - \nu \\ \text{s.t.} \quad & \begin{bmatrix} \displaystyle\sum_{i=1}^m \lambda_i & \displaystyle\sum_{i=1}^m \lambda_i(\mathbf{x}^{(i)})^T \\ \displaystyle\sum_{i=1}^m \lambda_i\mathbf{x}^{(i)} & \displaystyle Y-\sum_{i=1}^m \lambda_i\mathbf{x}^{(i)}(\mathbf{x}^{(i)})^T \end{bmatrix} \succcurlyeq 0\,, \\ & Y\succcurlyeq 0,\ \boldsymbol{\lambda}\geq 0\,. \end{aligned} \tag{8.4}$$

Proof. The Lagrangian function for (8.3) can be written as

$$\mathcal{L}(Z,\mathbf{z},\gamma;\boldsymbol{\lambda},\Lambda)=\log\det(Z)+\sum_{i=1}^m \lambda_i\left(1-(\mathbf{x}^{(i)})^T Z\mathbf{x}^{(i)}+2\mathbf{z}^T\mathbf{x}^{(i)}-\gamma\right)+\left\langle \Lambda,\begin{bmatrix} \gamma & \mathbf{z}^T \\ \mathbf{z} & Z \end{bmatrix}\right\rangle$$

with $\boldsymbol{\lambda} \in \mathbb{R}^m,\ \boldsymbol{\lambda} \geq 0$ and $\Lambda \in \mathbb{S}^{\nu+1},\ \Lambda \succcurlyeq 0$, leading to the following system of optimality conditions

$$Z^{-1} - \sum_{i=1}^{m} \lambda_i \left(\mathbf{x}^{(i)}(\mathbf{x}^{(i)})^T\right) + \Lambda_{2:\nu+1,2:\nu+1} = 0 \tag{8.5}$$

$$\sum_{i=1}^{m} \lambda_i \mathbf{x}^{(i)} + \Lambda_{2:\nu+1,1} = 0 \tag{8.6}$$

$$\sum_{i=1}^{m} \lambda_i - \Lambda_{1,1} = 0 \tag{8.7}$$

$$1 - (\mathbf{x}^{(i)})^T Z \mathbf{x}^{(i)} + 2\mathbf{z}^T \mathbf{x}^{(i)} - \gamma \geq 0\,, \quad i = 1, \ldots, m$$

$$\begin{bmatrix} \gamma & \mathbf{z}^T \\ \mathbf{z} & Z \end{bmatrix} \succcurlyeq 0,\ \boldsymbol{\lambda} \geq 0,\ \Lambda \succcurlyeq 0$$

$$\sum_{i=1}^{m} \lambda_i \left(1 - (\mathbf{x}^{(i)})^T Z \mathbf{x}^{(i)} + 2\mathbf{z}^T \mathbf{x}^{(i)} - \gamma\right) = 0, \quad \left\langle \Lambda, \begin{bmatrix} \gamma & \mathbf{z}^T \\ \mathbf{z} & Z \end{bmatrix} \right\rangle = 0\,.$$

Lagrangian dual to (8.3) reads as

$$\inf_{\boldsymbol{\lambda},\Lambda} \sup_{Z,\mathbf{z},\gamma} \mathcal{L}(Z, z, \gamma; \boldsymbol{\lambda}, \Lambda)$$

and using (8.5)–(8.7), we get

$$\begin{aligned} \sup_{Z,\mathbf{z},\gamma} \mathcal{L}(Z, \mathbf{z}, \gamma; \boldsymbol{\lambda}, \Lambda) &= \log\det(Z) + \sum_{i=1}^{m} \lambda_i - \sum_{i=1}^{m} \lambda_i (\mathbf{x}^{(i)})^T Z \mathbf{x}^{(i)} + \langle \Lambda_{2:\nu+1,2:\nu+1}\,, Z \rangle \\ &= \log\det(Z) + \sum_{i=1}^{m} \lambda_i - \left\langle \sum_{i=1}^{m} \lambda_i \mathbf{x}^{(i)}(\mathbf{x}^{(i)})^T - \Lambda_{2:\nu+1,2:\nu+1}\,, Z \right\rangle \\ &= \log\det(Z) + \sum_{i=1}^{m} \lambda_i - \nu\,. \end{aligned}$$

Setting $Y := Z^{-1} = \Lambda_{2:\nu+1,2:\nu+1} + \sum_{i=1}^{m} \lambda_i \mathbf{x}^{(i)}(\mathbf{x}^{(i)})^T$, we arrive at the objective function of the dual problem. The condition $\Lambda \succcurlyeq 0$, together with the equalities (8.5)–(8.7), then leads to the constraints in the dual problem (8.4). □

Now, to find the minimal enclosing ellipsoid of the semi-algebraic set $\hat{S}$, we generalize the

above problem and solve, to global optimality, the following *polynomial* SDP problem

$$\begin{aligned}\sup_{Z\in\mathbb{S}^\nu,\,\mathbf{z}\in\mathbb{R}^\nu,\,\gamma\in\mathbb{R},\,\mathbf{x}\in\mathbb{R}^n} \quad & \log\det(Z) \\ \text{s.t.} \quad & 1-\boldsymbol{\xi}(\mathbf{x})^T Z\boldsymbol{\xi}(\mathbf{x})+2\mathbf{z}^T\boldsymbol{\xi}(\mathbf{x})-\gamma\geq 0\,, \\ & \begin{bmatrix}\gamma & \mathbf{z}^T\\ \mathbf{z} & Z\end{bmatrix}\succcurlyeq 0\,, \\ & \mathrm{g}_j(\mathbf{x})\geq 0, \quad j=1,\ldots,m_1\,, \\ & h_l(\mathbf{x})=0, \quad l=1,\ldots,m_2\,.\end{aligned} \tag{8.8}$$

The relaxation of order d reads as

$$\begin{aligned}\sup_{\substack{Z\in\mathbb{S}^\nu,\,\mathbf{z}\in\mathbb{R}^\nu,\,\gamma\in\mathbb{R}\\ \{\sigma_j\},\,\{\psi_l\}}} \quad & \log\det(Z) \\ \text{s.t.} \quad & 1-\boldsymbol{\xi}(\mathbf{x})^T Z\boldsymbol{\xi}(\mathbf{x})+2\mathbf{z}^T\boldsymbol{\xi}(\mathbf{x})-\gamma \\ & \qquad -\sum_{j=1}^{m_1}\sigma_j(\mathbf{x})\mathrm{g}_j(\mathbf{x})-\sum_{l=1}^{m_2}\psi_l(\mathbf{x})h_l(\mathbf{x})\in\Sigma[\mathbf{x}]_d\,, \\ & \begin{bmatrix}\gamma & \mathbf{z}^T\\ \mathbf{z} & Z\end{bmatrix}\succcurlyeq 0\,, \\ & \sigma_j\in\Sigma[\mathbf{x}]_{\frac{d-v_j}{2}}\,, \quad j=1,\ldots,m_1\,, \\ & \psi_l\in\mathbb{R}[\mathbf{x}]_{d-\tau_l}, \quad l=1,\ldots,m_2\end{aligned} \tag{8.9}$$

To implement this convex program, we replace the objective function $\log\det(Z)$ by t for a point $(t,1,Z)$ in the *log-det cone*:

$$\mathcal{K}=\{(t,s,Z)\in\mathbb{R}\times\mathbb{R}\times\mathbb{R}^{\nu\times\nu}\mid s\log\det(Z/s)\geq t, Z\succcurlyeq 0, s>0\}.$$

Using the fact that every SoS polynomial can be represented by a positive semidefinite

matrix, we can rewrite (8.9) as the following convex SDP with linear matrix inequalities:

$$\max_{\substack{Z\in\mathbb{S}^{\nu},\,\mathbf{z}\in\mathbb{R}^{\nu},\,\gamma\in\mathbb{R}\\ X_j\in\mathbb{S}_+^{s\left(\frac{d-v_j}{2}\right)},\,Y_l\in\mathbb{S}^{s(d-\tau_l)}}} \log\det(Z) \tag{8.10}$$

$$\begin{aligned}
\text{s.t.}\quad & 1-\left(\boldsymbol{\xi}(\mathbf{x})^T Z\boldsymbol{\xi}(\mathbf{x})\right)_\alpha + 2\left(\mathbf{z}^T\boldsymbol{\xi}(\mathbf{x})\right)_\alpha - \gamma \\
&\qquad = \sum_{j=0}^{m_1}\left\langle X_j, C_\alpha^j\right\rangle + \sum_{k=1}^{m_2}\left\langle Y_l, D_\alpha^l\right\rangle, \qquad \text{for } \alpha = 0\,, \\
& -\left(\boldsymbol{\xi}(\mathbf{x})^T Z\boldsymbol{\xi}(\mathbf{x})\right)_\alpha + 2\left(\mathbf{z}^T\boldsymbol{\xi}(\mathbf{x})\right)_\alpha \\
&\qquad = \sum_{j=0}^{m_1}\left\langle X_j, C_\alpha^j\right\rangle + \sum_{l=1}^{m_2}\left\langle Y_l, D_\alpha^l\right\rangle, \qquad \text{for } |\alpha|\le d, \alpha\neq 0\,, \\
& \begin{bmatrix} \gamma & \mathbf{z}^T \\ \mathbf{z} & Z \end{bmatrix} \succcurlyeq 0\,, \\
& X_j \succcurlyeq 0, \quad j = 0,\ldots,m_1\,,
\end{aligned}$$

where $(\xi(\mathbf{x}))_\alpha$ is the α coefficient of the polynomial $\xi(\mathbf{x})$, $\langle A, B\rangle = \text{trace}(A^TB)$ for any matrices $A, B\in\mathbb{R}^{s\times s}$, C_α^j and D_α^l are matrices associated, respectively, with polynomials $\mathrm{g}_j(\mathbf{x})$ ($\mathrm{g}_0(\mathbf{x})=1$) and $h_l(\mathbf{x})$ via, $\mathrm{g}_j(\mathbf{x})\mathrm{v}_d(\mathbf{x})\mathrm{v}_d(\mathbf{x})^T = \sum_\alpha C_\alpha^j\mathbf{x}^\alpha$ and $h_l(\mathbf{x})\mathrm{v}_d(\mathbf{x})\mathrm{v}_d(\mathbf{x})^T = \sum_\alpha D_\alpha^l\mathbf{x}^\alpha$ for some basis v_d of $\mathbb{R}[\mathbf{x}]_d$, of size $s(d) = \begin{pmatrix} n+d \\ n \end{pmatrix}$.

Denote by $\boldsymbol{\xi}_\alpha\in\mathbb{R}^\nu$, $\alpha : |\alpha|\le d$, vectors containing $\alpha-$coefficients of polynomials $\xi_i(\mathbf{x})$, $i=1,\ldots,\nu$. Then $(\boldsymbol{\xi}(\mathbf{x})\boldsymbol{\xi}(\mathbf{x})^T)_\alpha = \sum_{\beta+\gamma=\alpha}\boldsymbol{\xi}_\beta\boldsymbol{\xi}_\gamma^T$.

Lemma 8.3.2. *The dual problem to (8.10) reads as*

$$\inf_{\Psi\in\mathbb{S}^{\nu},\,\boldsymbol{\lambda}\in\mathbb{R}^{s(d)}} \lambda_0 - \log\det(\Psi) - \nu \tag{8.11}$$

$$\text{s.t.} \begin{bmatrix} \lambda_0 & \sum\limits_{\alpha:|\alpha|\le 2\,d} \lambda_\alpha \boldsymbol{\xi}_\alpha^T \\ \sum\limits_{\alpha:|\alpha|\le 2\,d} \lambda_\alpha \boldsymbol{\xi}_\alpha & \Psi - \sum\limits_{\alpha:|\alpha|\le 2\,d} \lambda_\alpha \left(\sum\limits_{\beta+\gamma=\alpha} \boldsymbol{\xi}_\beta \boldsymbol{\xi}_\gamma^T \right) \end{bmatrix} \succcurlyeq 0\,,$$

$$\Psi \succcurlyeq 0\,,$$

$$\sum_{\alpha:|\alpha|\le 2\,d} \lambda_\alpha C_\alpha^j \succcurlyeq 0, \quad j = 0,\ldots,m_1\,,$$

$$\sum_{\alpha:|\alpha|\le 2\,d} \lambda_\alpha D_\alpha^l = 0, \quad l = 1,\ldots,m_2\,.$$

Proof. The Lagrangian function for (8.10) can be written as

$$\begin{aligned}
&\mathcal{L}(Z,\mathbf{z},\gamma,X,Y;\boldsymbol{\lambda},\Xi,\Theta) = \log\det(Z) \\
&\quad + \sum_{\alpha:|\alpha|\le 2\,d} \lambda_\alpha \left(1_{|\alpha=0} - (\boldsymbol{\xi}^T Z \boldsymbol{\xi})_\alpha + 2(\mathbf{z}^T\boldsymbol{\xi})_\alpha - \gamma_{|\alpha=0} - \sum_{j=0}^{m_1} \left\langle X_j, C_\alpha^j \right\rangle - \sum_{l=1}^{m_2} \left\langle Y_l, D_\alpha^l \right\rangle \right) \\
&\quad + \left\langle \Xi, \begin{bmatrix} \gamma & \mathbf{z}^T \\ \mathbf{z} & Z \end{bmatrix} \right\rangle + \sum_{j=1}^{m_1} \langle \Theta_j, X_j \rangle
\end{aligned}$$

with $\boldsymbol{\lambda} \in \mathbb{R}^{d[\ell]}$, $\Xi \in \mathbb{S}^{\nu+1}$, $\Xi \succcurlyeq 0$ and $\Theta \in \mathbb{S}^{\nu}$, $\Theta_j \succcurlyeq 0$, leading to the following system of

optimality conditions:

$$
\begin{aligned}
Z^{-1} - \sum_{\alpha:|\alpha|\le 2\,d} \lambda_\alpha \left(\sum_{\beta+\gamma=\alpha} \boldsymbol{\xi}_\beta \boldsymbol{\xi}_\gamma^T \right) + \Xi_{2:\nu+1,2:\nu+1} &= 0 \qquad (8.12)\\
\sum_{\alpha:|\alpha|\le 2\,d} \lambda_\alpha \boldsymbol{\xi}_\alpha + \Xi_{2:\nu+1,1} &= 0\\
\lambda_0 - \Xi_{1,1} &= 0\\
\sum_{\alpha:|\alpha|\le 2\,d} \lambda_\alpha C_\alpha^j - \Theta_j = 0, \quad j &= 0,\dots,m_1\\
\sum_{\alpha:|\alpha|\le 2\,d} \lambda_\alpha D_\alpha^l = 0, \quad l &= 1,\dots,m_2\,.
\end{aligned}
$$

The Lagrangian dual to (8.10) reads as

$$
\inf_{\boldsymbol{\lambda},\Xi,\Theta} \sup_{Z,\mathbf{z},\gamma,X,Y} \mathcal{L}(Z,\mathbf{z},\gamma,X,Y;\boldsymbol{\lambda},\Xi,\Theta)
$$

and, using (8.12), we get

$$
\begin{aligned}
&\sup_{Z,\mathbf{z},\gamma} \mathcal{L}(Z,\mathbf{z},\gamma,X,Y;\boldsymbol{\lambda},\Xi,\Theta) = \log\det(Z)\\
&\quad + \lambda_0 - \sum_{\alpha:|\alpha|\le 2\,d} \lambda_\alpha \left\langle Z, \sum_{\beta+\gamma=\alpha} \boldsymbol{\xi}_\beta \boldsymbol{\xi}_\gamma^T \right\rangle - \sum_{\alpha:|\alpha|\le 2\,d} \lambda_\alpha \left(\sum_{j=0}^{m_1} \left\langle X_j, C_\alpha^j \right\rangle + \sum_{l=1}^{m_2} \left\langle Y_l, D_\alpha^l \right\rangle \right)\\
&\quad + \left\langle \Xi_{2:\nu+1,s:\nu+1}, Z \right\rangle + \sum_{j=1}^{m_1} \left\langle \Theta_j, X_j \right\rangle\\
&= \log\det(Z) + \lambda_0 - \left\langle \sum_{\alpha:|\alpha|\le 2\,d} \lambda_\alpha \left(\sum_{\beta+\gamma=\alpha} \boldsymbol{\xi}_\beta \boldsymbol{\xi}_\gamma^T \right) - \Xi_{2:\nu+1,2:\nu+1}\,, Z \right\rangle\\
&= \log\det(Z) + \lambda_0 - \left\langle Z^{-1}, Z \right\rangle = \log\det(Z) + \lambda_0 - \nu\,.
\end{aligned}
$$

Setting $\Psi := Z^{-1} = \sum_{\alpha:|\alpha|\le 2\,d} \lambda_\alpha \left(\sum_{\beta+\gamma=\alpha} \boldsymbol{\xi}_\beta \boldsymbol{\xi}_\gamma^T \right) - \Xi_{2:\nu+1,2:\nu+1}$, we arrive at the objective function of the dual problem. The condition $\Xi \succcurlyeq 0$, together with the equalities (8.12), then leads to the constraints in the dual problem (8.11). □

The SDP formulation (8.11) corresponds to the following moment problem:

$$\begin{aligned}
\inf_{\Psi\in\mathbb{S}^\nu,\,\Lambda\in\mathbb{R}[\mathbf{x}]_d^*} \quad & \Lambda_0 - \log\det(\Psi) - \nu \\
\text{s.t.} \quad & \begin{bmatrix} \Lambda_0 & \Lambda(\boldsymbol{\xi})^T \\ \Lambda(\boldsymbol{\xi}) & \Psi - \Lambda(\boldsymbol{\xi}\boldsymbol{\xi}^T) \end{bmatrix} \succcurlyeq 0 \\
& \Psi \succcurlyeq 0\,, \\
& H_{\mathrm{g}_j}(\Lambda) \succcurlyeq 0, \quad j = 0,\ldots,m_1, \\
& \Lambda(\mathbf{x}^\alpha h_l) = 0, \quad |\alpha| \leq 2\,d - \tau_l, l = 1,\ldots,m_2
\end{aligned} \tag{8.13}$$

where $\Lambda(\boldsymbol{\xi}) = [\Lambda(\xi_1(\mathbf{x})),\ldots,\Lambda(\xi_\nu(\mathbf{x}))]$, $\Lambda(\boldsymbol{\xi}\boldsymbol{\xi}^T) = (\Lambda(\xi_i(\mathbf{x})\xi_j(\mathbf{x})))_{1\leq i,j\leq\nu}$, $\mathrm{g}_0 = 1$, $v_0 = 0$ and $H_{\mathrm{g}_j}(\Lambda) = (\Lambda(\mathrm{g}_j\mathbf{x}^{\alpha+\beta}))_{|\alpha|,|\beta|\leq\frac{2\,d-v_j}{2}}$, and $\mathbb{R}[\mathbf{x}]_d^*$ denote the dual to $\mathbb{R}[\mathbf{x}]_d$.

8.4 Parameterized surfaces

Most of the representations of shapes used in CAD are based on piecewise polynomial or rational parametrizations, namely the image of functions of the form $\sigma : \mathbf{u} \in D \mapsto (p_1(\mathbf{u}), p_2(\mathbf{u}), p_3(\mathbf{u})) \in \mathbb{R}^3$, where D is typically an interval, the unit box in $\mathbb{R}^2$ or the unit cube in $\mathbb{R}^3$ and p_i are spline or piecewise polynomial functions or a ratio of two spline functions. For the sake of simplicity, hereafter the functions p_i will be polynomial functions. We illustrate the use of optimization tools on two types of problems involving surfaces parameterized by polynomials over the domain $D = [0,1]^2$. These examples generalize easily to parametric volumes.

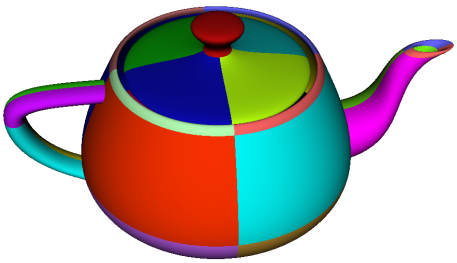


Figure 8.1: A teapot model composed on 20 *patches of bicubic polynomial parametrisations.*

8.4.1 Closest point and surface-surface intersection

Given a point $A = (a_1, a_2, a_3) \in \mathbb{R}^3$, finding the closest point to A on a parameterized surface $\sigma : (u_1, u_2) \in D = [0,1]^2 \mapsto (p_1(u_1, u_2), \ldots, p_3(u_1, u_2))$ can easily be stated as a minimisation problem. This minimisation problem is the problem (3.2) with

$$\begin{aligned} f &= \sum_{i=1}^{3} (a_i - \mathrm{x}_i)^2, \\ \mathbf{g} &= \{u_j,\ 1 - u_j, \quad j = 1, 2\}, \\ \mathbf{h} &= \{\mathrm{x}_i - p_i(u_1, u_2), \quad i = 1, 2, 3\}. \end{aligned} \tag{8.14}$$

The objective function is a polynomial of degree 2. We have 3 equality constraints $\mathrm{x}_i - p_i(u_1, u_2) = 0$, $i = 1, \ldots, 3$ and 4 sign constraints $0 \leq u_j \leq 1$. This usually gives a single closest point, except for points on the medial axis of the surface as illustrated in Figure 8.2.

To detect if two surfaces given by the parametrizations $\sigma_1 : \mathbf{u}_1 \in D \mapsto (p_{1,1}(\mathbf{u}_1), \ldots, p_{1,3}(\mathbf{u}_1)) \in \mathbb{R}^3$, $\sigma_2 : \mathbf{u}_2 \in D \mapsto (p_{2,1}(\mathbf{u}_2), \ldots, p_{2,3}(\mathbf{u}_2)) \in \mathbb{R}^3$ intersect, we use a slightly different formulation in order to get generically a single minimizer: We solve the optimisation problem (3.2) with

$$\begin{aligned} f &= \|\mathbf{u}_1 - \mathbf{u}_2\|^2, \\ \mathbf{g} &= \{u_{k,j},\ 1 - u_{k,j}, \quad j = 1, 2,\ k = 1, 2\}, \\ \mathbf{h} &= \{p_{1,i}(\mathbf{u}_1) - p_{2,i}(\mathbf{u}_2), \quad i = 1, 2, 3\}. \end{aligned} \tag{8.15}$$

Example 8.4.1. ***Example of closest point and intersection point computation***

We illustrate in Figure 8.2 the computation of the closest point based on the moment formulation (8.2) *for the polynomial objective and constraints* (8.14) *(the two yellow points, which are the closest to the red point on the pink patch) for patches of degree* 3 *in* u_1 *and* 3 *in* u_2 *(called bi-cubic patches). We also show an intersection point (the green point on the intersection of the pink and blue patches) corresponding to the solution of* (8.15)*. The orders of relaxation used for these computations are respectively* $d = 3$ *and* $d = 4$*.*

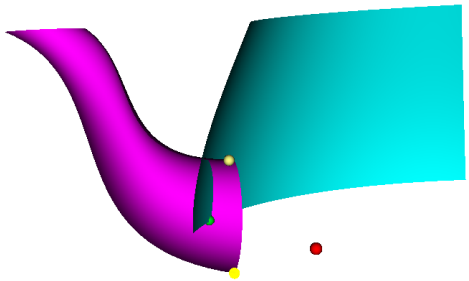


Figure 8.2: Closest points and intersection point for bicubic parametric surfaces.

8.4.2 Bounding box and enclosing ellipsoid of parametric surfaces

Computing simple minimal enclosing solids of a given shape is an important problem with many applications in animation, collision detection, simulation, robotics, etc.

For minimal enclosing axis-aligned bounding boxes of parametric surfaces, this reduces to solving problems of the form (8.14), where the objective function is replaced by $\pm \mathrm{x}_i$.

For minimal enclosing ellipsoids, we need to solve the polynomial SDP (8.8) using SoS relaxations (8.10). Notice that (8.10) is not a standard (linear) SDP, due to the determinant function. It can be solved, for instance, by software `Hypatia`[3] or `Mosek` using so called *LogDetTriangular*

[3] https://github.com/chriscoey/Hypatia.jl

cones.

Example 8.4.2. ***Example of minimal enclosing ellipsoid computation***

In Figure 8.3, we present an example of a minimal enclosing ellipsoid for $p_1(u,v) = u+v, p_2(u,v) = 2u + u^2 - uv + v^2, p_3(u,v) = v + \frac{1}{2}u^3 + \frac{1}{2}v^3$ *solved by* `MomentTools` *at relaxation level* $d = 2$*, with* `Mosek` *convex optimizer:*

```
> s = [u+v, 2*u+u^2-u*v+v^2, v+1/2*u^3+ 1/2*v^3]
> H = [x1-s[1], x2-s[2], x2-s[3]]
> G = [u-u^2, v-v^2]
> P = [x1, x2, x3]
> c, U, M = min_ellipsoid(P, H, G, X, 2, Mosek.Optimizer)
```

It returns the center `c` *and the matrix* `U`*, which columns are the principal axes of the ellipsoid.*

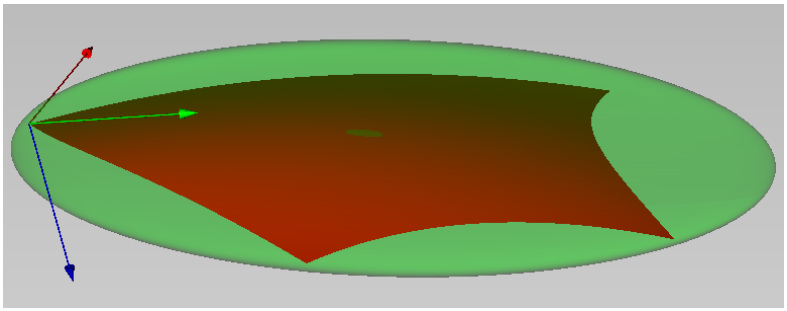


Figure 8.3: Closest points, intersection point and minimal enclosing ellipsoid for bicubic parametric surfaces.

8.5 Robots and mechanisms

A parallel robot is defined by a fixed platform

$$A = [A_1, \ldots, A_6] \subset \mathbb{R}^{3\times 6},$$

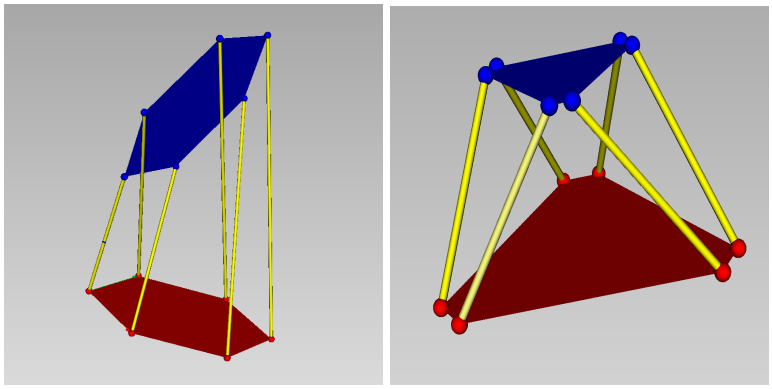


Figure 8.4: Parallel robots with a fixed platform (red platform), moving platform (blue platform) and extensible arms (yellow arms).

and a moving platform with the initial position

$$B = [B_1, \ldots, B_6] \subset \mathbb{R}^{3\times 6},$$

connected by extensible arms A_i–B_i. Figure 8.4 shows two examples of parallel robots. The robot on the right is known as the Stewart platform, and it is the geometry that we are considering in our numerical experiments.

We represent the displacement of the moving platform by a rotation R of the orthogonal group $O(\mathbb{R}^3)$ and a translation $T \in \mathbb{R}^3$ so that the position of the points of the moving platform is

$$\widehat{B}_i = RB_i + T, \quad i = 1, \ldots, 6$$

with the length of the arms

$$d_i = \|RB_i + T - A_i\|_2, \; i = 1, \ldots, 6.$$

The problem consists of analyzing the position of the moving platform under the constraints that the lengths of the arms are within some intervals:

$$m_i \leq d_i \leq M_i, \quad i = 1, \ldots, 6\,.$$

Here is the parameterization of the rotation by unit quaternion [85]:

$$R = \begin{bmatrix} u_1^2 + u_2^2 - u_3^2 - u_4^2 & -2u_1u_4 + 2u_2u_3 & 2u_1u_3 + 2u_2u_4 \\ 2u_1u_4 + 2u_2u_3 & u_1^2 - u_2^2 + u_3^2 - u_4^2 & -2u_1u_2 + 2u_3u_4 \\ -2u_1u_3 + 2u_2u_4 & 2u_1u_2 + 2u_3u_4 & u_1^2 - u_2^2 - u_3^2 + u_4^2 \end{bmatrix} \tag{8.16}$$

with $u_1^2 + u_2^2 + u_3^2 + u_4^2 = 1$ and the translation $T = [x, y, z]$. Then

$$\delta_i(u_1, u_2, u_3, u_4, x, y, z) = d_i^2 = \|B_i\|^2 + \|A_i\|^2 + \|T\|^2 + 2RB_i \cdot T - 2RB_i \cdot A_i - 2T \cdot A_i \tag{8.17}$$

is a polynomial function of $\mathbf{u} = (u_1, \ldots, u_4)$ and x, y, z of degree 2 in $\mathbf{u}$ and total degree 3 (here $\mathbf{v} \cdot \mathbf{w}$ stands for the standard inner product of vectors $\mathbf{v}, \mathbf{w} \in \mathbb{R}^p$).

8.5.1 Direct kinematic problem

The direct kinematic problem consists of finding the position of the moving platform such that a point C attached to the platform is at a given point X_0 in the space. To solve this problem, we search for a position of the platform, that minimizes the norm of the translation T, solving the optimization problem (3.2) with

$$\begin{aligned} f &= \|T\|^2, \\ \mathbf{g} &= \left\{\delta_i - m_i^2, M_i^2 - \delta_i, \quad i = 1, \ldots, 6\right\}, \\ \mathbf{h} &= \left\{(X_0)_j - (RC + T)_j, \quad j = 1, \ldots, 3, \right. \\ &\qquad \left. \|u\|^2 - 1\right\}. \end{aligned} \tag{8.18}$$

This is when we know the problem is feasible. If we are uncertain about the existence of a feasible solution, we reformulate the objective functions and the constraints to

$$\begin{aligned} f &= \|T\|^2 + \rho\|X_0 - (RC + T)\|^2, \\ \mathbf{g} &= \left\{\delta_i - m_i^2, M_i^2 - \delta_i, \quad i = 1, \ldots, 6\right\}, \\ \mathbf{h} &= \left\{\|u\|^2 - 1\right\}, \end{aligned} \tag{8.19}$$

where ρ is a penalty parameter (we choose $\rho = 1000$ in our experiments). The minimizer of this problem yields the translation(s) T^{new} and rotation(s) R^{new} required to locate the new position of the platform. This position is calculated by $B_i^{\text{new}} = R^{\text{new}} B_i + T^{\text{new}}$ for $i = 1, \ldots, 6$.

Example 8.5.1. ***Example of a direct kinematic problem***

In Figure 8.5, we illustrate the computational result of solving problem (3.2) *for finding the new position of a Stewart platform. The values of the fixed platform A_0 and the initial position of the moving platform B_0 of this geometry are produced in Julia by*

```
> function pl(x,r)  [r * cos(x), r * sin(x), 0] end
> A0    = [pl(0,1), pl(pi/12,1),pl(2pi/3,1), pl(2pi/3+pi/12,1),
           pl(4pi/3,1), pl(4pi/3+pi/12,1)]
> B0pre = [pl(5pi/3+pi/12,1/2), pl(pi,1/2), pl(pi/12+pi/3,1/2),
           pl(pi/3,1/2), pl(pi+pi/12,1/2), pl(5pi/3,1/2)]
> Xi0 = [1, 0, 0, 0, 0, 0, 1]
> R0  = Rotation(Xi0[1:4]...); T0 = Xi0[5:7]
> B0  = [R0*a + T0 for a in B0pre]
```

where the function `Rotation` *is the quaternion parametrization of rotations given in* (8.16)*. In this numerical experiment, we consider the point C to be the center of the moving platform and the point X_0 reached by the moving platform (shown in green in Figure 8.5), to be*

```
> C = sum(b for b in B0)/length(B0)
> X0 = [0.6,0.4,1]
```

In addition, the upper bound and lower bound on the lengths are

```
> D0 = norm.(B0-A0)
> ub = D0 .+ 2.5
```

```
> lb = D0 .- 2.5
```

where `D0` *is the vector of the lengths of the robot arms in the initial position. We consider the objective function and constraints to be*

```
> f = nrm2(T)
> g = []; for i in 1:6 g = [g; D1[i] - lb[i]^2; ub[i]^2 - D1[i]] end
> h = [u1^2+u2^2+u3^2+u4^2-1]
> h = [h; X0-(R*C+T)]
```

We solve the moment relaxation of order $d = 3$ *by using* `Mosek` *convex optimizer:*

```
> using MomentTools, DynamicPolynomials, MosekTools
> X = @polyvar u1 u2 u3 u4 x y z
> v, M = minimize(f, h, g, X, 3, Mosek.Optimizer)
> w, Xi = get_measure(M)
```

By selecting the minimizers `Xi[:,i]` *with a big enough weight* `w[i]`*, we obtain* T^{new} *and* R^{new} *needed to find the platform's new position:*

```
> Sol = []
> for i in 1:length(w)
     if abs(w[i]) > 0.1  push!(Sol,Xi[:,i])  end
  end
> R_new = Rotation(Sol[1][1:4]...)
> T_new = Sol[1][5:7]
```

and the new platform will be

```
> B_new = [R_new *a + T_new for a in B0]
```

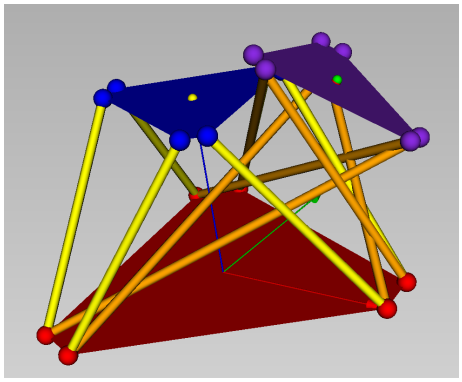


Figure 8.5: The new position of the platform is shown in violet such that the middle of the blue platform is at a given point in the space (the green point).

which is demonstrated in violet in Figure 8.5.

8.5.2 Bounding box of robot workspace

Let C be a point attached to the moving platform B; in our numerical experiments it is the center of the platform but it can be any other point. We will be looking for the axis-aligned bounding box problem of this point after all possible rotations and translations, i.e., of the point $\widehat{C} = RC + T$. The bounds can be obtained by solving the optimization problem (3.2) with

$$\begin{aligned} f &= \pm\, \widehat{C}_k = (RC+T)_k, \quad k = 1,2,3, \\ \mathbf{g} &= \left\{ \delta_i - m_i^2,\ M_i^2 - \delta_i, \quad i = 1,\ldots,6 \right\}, \\ \mathbf{h} &= \left\{ \|\mathbf{u}\|^2 - 1 \right\}. \end{aligned} \tag{8.20}$$

One needs to solve six optimization problems, three with objective $\widehat{C}_k$ and three with $-\widehat{C}_k$, $k = 1,2,3$, to find an interval for each coordinate of $\widehat{C}$. By using these values, we obtain the extreme positions for different faces of the intended bounding box.

Example 8.5.2. ***Example for computing the bounding box***

In Figure 8.6–left, we present the computational result of solving problem (3.2) *to find the bounding*

box of a Stewart platform. The coordinates of the platform points for this robot are as mentioned in the previous example. We choose the upper bound and lower bound of the lengths to be

```
> ub = D0 .+ 0.2
> lb = D0 .- 0.2
```

The constraints of this problem are

```
> g = []; for i in 1:6 g = [g; D1[i]-lb[i]^2; ub[i]^2-D1[i]; B1[i][3]] end
> h = [u1^2+u2^2+u3^2+u4^2-1]
```

where

```
> B1 = [R*a+T for a in B0]
```

As we mentioned, to find the bounding box, we need to solve six optimization problems. If we consider

```
> C_hat = R*C+T
```

where `C` *is the centre of the moving platform, the objective functions of these problems for* $i = 1, \ldots, 3$ *will be*

```
> f[i] = C_hat[i]
```

We solve six moment relaxations by using `Mosek` *software*

```
> for i in 1:3
    b[i], Mb[i] = minimize(f[i],h,g,X,l,opt)
    B[i], MB[i] = maximize(f[i],h,g,X,l,opt)
end
```

from which we deduce the bounding box shown in Figure 8.6–left. These moment relaxations are of order $d = 3$*. Only to find* `b[3]`*, we needed a higher order relaxation and we considered* $d = 4$*. In order to find out if the order of relaxation is sufficient, we check whether* $u_1^2 + u_2^2 + u_3^2 + u_4^2$ *is close enough to one at the solutions. In our numerical experiments to compute the minimum* `b[3]` *of* z*-coordinate, with the relaxation order* $d = 3$*, we find* 4 *approximate minimizers with the following norm for* $\mathbf{u}$*:*

```
>  norm sol u: 0.9345868913061018
>  norm sol u: 0.9796236439932896
>  norm sol u: 0.6612425286297167
>  norm sol u: 0.8916846004102044
```

Since these values are not close enough to 1*, we increase the order of the relaxation and for* $d = 4$*, we obtain 4 approximate minimizers with*

```
>  norm sol u: 0.9973888649655355
>  norm sol u: 0.9968397474090084
>  norm sol u: 0.9971812563313878
>  norm sol u: 0.9985517657681842
```

The relaxation at order $d = 4$ *gives a good enough approximation of the minimizers and of the minimum of* z*.*

8.5.3 Enclosing ellipsoid of robot workspace

The minimal enclosing ellipsoid for all positions of point C from the above section can now be found by the same technique as in Section 8.4.2. The points $\xi(\mathbf{x})$ in (8.8) will now be replaced by the point $\widehat{C}$, as a function of $u_1, u_2, u_3, u_4, x, y, z$ as in problem (8.20). Figure 8.6–left presents

a comparison of the bounding box and the minimal ellipsoid for the center point of the moving platform. The order of relaxation for computing the minimal ellipsoid is $d = 4$ in this example.

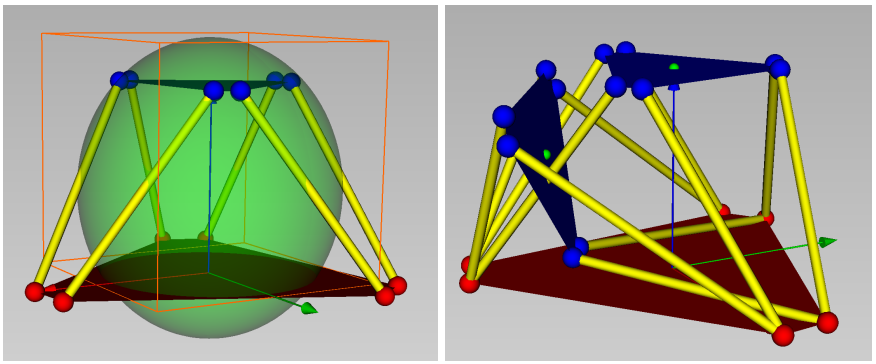


Figure 8.6: On the left, we show a comparison of the minimal enclosing ellipsoid and bounding box. The initial position and extreme position of the platform minimizing y are presented on the right.

8.5.4 Trajectories of a parallel robot

In this section, we analyze the trajectory of the point C attached to the platform B, first when the first interval constraint is satisfied and the other five lengths have a fixed value in their length interval; secondly when the first two interval constraints are satisfied and the other four lengths have a fixed value.

Case 1: Five fixed values In this case, the trajectory of the point C is the curve consisting of the possible positions of the C in the space. To obtain different positions of the platform under these constraints, we minimize and maximize $\widehat{C}_1$, so we will have an interval for extreme positions corresponding to the first coordinate. This means that we need to solve the problem (8.20) with objective $\pm\widehat{C}_1$ and the additional constraints

$$\delta_i = \delta_i^0, \quad i = 2, \ldots, 6\,,$$

in which $\delta^0 = (d^0)^2 = \|B_0 - A_0\|$. Next, we sample the interval between these extreme values at intermediate values μ_k, $k = 1, \ldots, m$ and solve feasibility problems with an additional equality constraint $\widehat{C}_1 - \mu_k = 0$ for $k = 1, \ldots, m$. This minimization is done by solving problem (3.2) with

$$\begin{aligned} f &= 1\,, \\ \mathbf{g} &= \left\{\delta_1 - m_1^2,\ M_1^2 - \delta_1\right\}, \\ \mathbf{h} &= \Big\{\widehat{C}_1 - \mu_k, \quad k = 1, \ldots, m\,, \\ &\qquad \delta_i - \delta_i^0, \quad i = 2, \ldots, 6\,, \\ &\qquad \|\mathbf{u}\|^2 - 1\Big\}\,. \end{aligned} \tag{8.21}$$

The minimizers of these problems yield the displacements (R, T) and the positions of C in the space. The same computation is repeated for the other coordinates $\widehat{C}_2, \widehat{C}_3$ to get a better sampling of the trajectory.

Case 2: Four fixed values We compute positions of C attached to the platform when the first two interval constraints are satisfied and the other four lengths have a fixed value. This traces out a surface of the positions of the point C. The problem is analogous to Case 1; only, we need to compute the interval for extreme positions corresponding to the first and second coordinates. Thus, we need to solve problem (3.2) with

$$\begin{aligned} f &= \pm\,\widehat{C}_k, \quad k = 1, 2\,, \\ \mathbf{g} &= \left\{\delta_i - m_i^2,\ M_i^2 - \delta_i, \quad i = 1, 2\right\}, \\ \mathbf{h} &= \left\{\delta_i - \delta_i^0, \quad i = 3, \ldots, 6\,, \right. \\ &\qquad \left. \|\mathbf{u}\|^2 - 1\right\}\,. \end{aligned} \tag{8.22}$$

If we consider μ and γ to be the values in the intervals obtained for the coordinates $\widehat{C}_1$ and $\widehat{C}_2$, respectively, the final feasibility problem to solve should satisfy all the previous constraints and

the additional constraints $\widehat{C}_1 = \mu,\ \widehat{C}_2 = \gamma$. The final problem to solve is problem (3.2) with

$$
\begin{aligned}
f &= 1\,, \\
\mathbf{g} &= \left\{\delta_i - m_i^2,\ M_i^2 - \delta_i, \quad i = 1, 2\right\}, \\
\mathbf{h} &= \Big\{\widehat{C}_1 - \mu,\ \widehat{C}_2 - \gamma,\ \|\mathbf{u}\|^2 - 1\,, \\
&\qquad \delta_i - \delta_i^0, \quad i = 2, \ldots, 6\Big\}\,.
\end{aligned}
\tag{8.23}
$$

As a result, we will have a set of points approximating a surface. We obtain these points by changing the constraints, choosing different values for $\widehat{C}_1$ and $\widehat{C}_2$ and solving the corresponding optimization problem.

Figure 8.7 shows the computational result of solving problem (3.2) to find the trajectories of a Stewart platform for these two cases. The order of relaxation for these problems is $d = 3$.

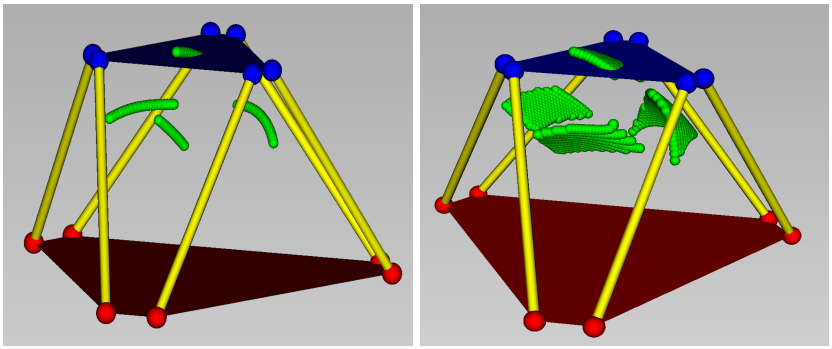


Figure 8.7: On the left, the trajectory of the center of the platform with 5 *arm lengths fixed is shown. The trajectory of the center of the platform with* 4 *arm lengths fixed is shown on the right.*

Chapter Nine

Conclusions

This thesis's research has mainly focused on solving semidefinite optimization problems coming from convex relaxations of polynomial optimizations. We have introduced new software Loraine for linear semidefinite optimization, particularly—but not exclusively—focused on problems with either low-rank solutions or low-rank data (or both). Loraine uses a preconditioner for the conjugate gradient method within an interior-point algorithm for large and sparse linear SDPs. The preconditioner utilizes the low rank of the SDP solutions. We have demonstrated Loraine's efficiency in solving three classes of well-known SDP problems; truss topology optimization, sensor network localization with noisy data, and unconstrained binary quadratic optimization problems. While our primary emphasis in these tests was on problems with low-rank solutions, we have also shown that the solely Matlab version of Loraine is competitive and may even outperform the recently most efficient SDP solvers when solving problems with low-rank data. The software is distributed as open source and is written in standard Matlab or Julia, allowing users to easily adapt it to their specific needs.

Our numerical experiments demonstrate the ability of Loraine to solve higher-order Lasserre relaxations of UBQP. The approach is particularly efficient for relaxation order two which, for most of the tested problems, was high enough to deliver either the exact solution of the UBQP or a good

approximation of it. We have also introduced a new, hybrid algorithm that uses an approximate solution obtained by ADMM as a warm start for the used interior point method. This algorithm is rather efficient for problems for which ADMM itself is relatively efficient, such as the MAXCUT problems. On the other hand, it may be inefficient once ADMM fails to converge quickly to an approximate solution.

While we tested our software for other problems from standard libraries, we do not report it here for one of the following reasons:

1. The available collections, such as SDPLIB [12], are not representative enough in the sense that they do not include enough problems of the same type. We did not want to pick up single problems.

2. Loraine mainly benefits from the use of an iterative solver for the linear system using a specific preconditioner targeted to problems with low-rank solutions. For many such problems, however, no preconditioner is needed for the convergence of the iterative method; these "easy" problems were not included in our testing. This is the case of matrix completion problems [6, 86] or some problems arising in the relaxation of combinatorial optimization. It is *not* the case of problems from the numerical experiments sections in Chapters 5-7; for those, the preconditioner is essential for convergence.

3. The problems do not satisfy our dimensional assumption $n \gg m$. For these problems, the preconditioner is still very efficient (assuming low-rank solutions) but the use of iterative methods does not bring any significant advantage to direct solvers. This is the case of many problems found in SDPLIB.

Besides this research, we have demonstrated how techniques of polynomial optimization can be used in the important area of geometric modeling. We focused on practical examples such as finding the bounding box or minimum enclosing ellipsoid for parametric surfaces or for a workspace

of parallel robots. The limitations of our approach are typical for polynomial optimization, in particular, the reliability of the numerical solutions of the resulting SDP problems and, sometimes, the necessity for higher order relaxations resulting in very large SDPs. The latter is, however, not too restrictive, given the small dimensions of the respective polynomial optimization problems. We thus believe that this technique will find further use, in particular, in robotics.

The studies and research conducted in this thesis have resulted in two papers. The first paper is titled 'Loraine—an interior-point solver for low-rank semidefinite programming' [30], and the second paper is titled 'Solving Lasserre relaxations of unconstrained binary quadratic optimization problems by an interior-point method' [31]. Additionally, a chapter titled 'Polynomial Optimization in Geometric Modeling' in the book 'Polynomial Optimisation, Moments, and Applications,' is published by Springer [29].

Bibliography

[1] Wolfgang Achtziger. “Optimierung von einfach und mehrfach belasteten Stabwerken”. PhD thesis. University of Bayreuth, Germany, 1993.

[2] Wolfgang Achtziger and Michal Kočvara. “On the maximization of the fundamental eigenvalue in topology optimization”. In: *Structural and Multidisciplinary Optimization* 34.3 (2007), pp. 181–195.

[3] Wolfgang Achtziger and Michal Kočvara. “Structural topology optimization with eigenvalues”. In: *SIAM Journal on Optimization* 18.4 (2008), pp. 1129–1164.

[4] Miguel F Anjos and Samuel Burer. “On handling free variables in interior-point methods for conic linear optimization”. In: *SIAM Journal on Optimization* 18.4 (2008), pp. 1310–1325.

[5] Mokhtar S. Bazaraa, Hanif D. Sherali, and C.M. Shetty. *Nonlinear Programming Theory and Algorithms*. New York: John Wiley, 1993.

[6] Stefania Bellavia, Jacek Gondzio, and Margherita Porcelli. “A Relaxed Interior Point Method for Low-Rank Semidefinite Programming Problems”. In: *arXiv preprint arXiv: 1909.06099* (2019).

[7] Aharon Ben-Tal and Arkadi Nemirovski. *Lectures on modern convex optimization: analysis, algorithms, and engineering applications*. Philadelphia: SIAM, 2001.

[8] Aharon Ben-Tal and Arkadi Nemirovski. *Lectures on modern convex optimization: analysis, algorithms, and engineering applications*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2001.

[9] Steven J. Benson, Yinyu Ye, and Xiong Zhang. "Solving Large-Scale Sparse Semidefinite Programs for Combinatorial Optimization". In: *SIAM Journal on Optimization* 10.2 (2000), pp. 443–461.

[10] Pratik Biswas, T-C Liang, K-C Toh, Yinyu Ye, and T-C Wang. "Semidefinite programming approaches for sensor network localization with noisy distance measurements". In: *IEEE transactions on automation science and engineering* 3.4 (2006), pp. 360–371.

[11] Pratik Biswas and Yinyu Ye. "Semidefinite programming for ad hoc wireless sensor network localization". In: *Proceedings of the 3rd international symposium on Information processing in sensor networks*. 2004, pp. 46–54.

[12] Brian Borchers. "SDPLIB 1.2, a library of semidefinite programming test problems". In: *Optimization Methods and Software* 11.1-4 (1999), pp. 683–690.

[13] Samuel Burer and Renato Monteiro. "A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization". In: *Mathematical Programming* 95.2 (2003), pp. 329–357.

[14] Samuel Burer and Renato Monteiro. "Local Minima and Convergence in Low-Rank Semidefinite Programming". In: *Mathematical Programming* 103 (July 2005), pp. 427–444.

[15] Juan S Campos, Ruth Misener, and Panos Parpas. "Partial Lasserre relaxation for sparse Max-Cut". In: *Optimization and Engineering* (2022), pp. 1–22.

[16] Emmanuel J Candès and Terence Tao. "The power of convex relaxation: Near-optimal matrix completion". In: *IEEE Transactions on Information Theory* 56.5 (2010), pp. 2053–2080.

[17] Frank E Curtis. "A penalty-interior-point algorithm for nonlinear constrained optimization". In: *Mathematical Programming Computation* 4.2 (2012), pp. 181–209.

[18] Etienne De Klerk. *Aspects of semidefinite programming: interior point algorithms and selected applications*. Vol. 65. Springer Science & Business Media, 2006.

[19] Gianni Di Pillo and Luigi Grippo. "Exact penalty functions in constrained optimization". In: *SIAM Journal on Control and Optimization* 27.6 (1989), pp. 1333–1360.

[20] Gerald Farin, Josef Hoschek, and Myung-Soo Kim. *Handbook of Computer Aided Geometric Design*. Elsevier, Aug. 2002.

[21] Gerald E. Farin. *Curves and Surfaces for CAGD: A Practical Guide*. 5th ed. Morgan Kaufmann Series in Computer Graphics and Geometric Modeling. San Francisco, CA: Morgan Kaufmann, 2001.

[22] Hamza Fawzi, James Saunderson, and Pablo A Parrilo. "Sparse sums of squares on finite abelian groups and improved semidefinite lifts". In: *Mathematical Programming* 160.1 (2016), pp. 149–191.

[23] Bissan Ghaddar, Juan C Vera, and Miguel F Anjos. "A dynamic inequality generation scheme for polynomial programming". In: *Mathematical Programming* 156.1 (2016), pp. 21–57.

[24] Fred Glover, Gary A Kochenberger, and Bahram Alidaee. "Adaptive memory tabu search for binary quadratic programs". In: *Management Science* 44.3 (1998), pp. 336–345.

[25] Nicholas IM Gould, Dominique Orban, and Philippe L Toint. "An interior-point l1-penalty method for nonlinear optimization". In: *Computational Science and Engineering Department, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England* (2003).

[26] Nicolo Gusmeroli, Timotej Hrga, Borut Lužar, Janez Povh, Melanie Siebenhofer, and Angelika Wiegele. "BiqBin: a parallel branch-and-bound solver for binary quadratic problems with linear constraints". In: *ACM Transactions on Mathematical Software (TOMS)* 48.2 (2022), pp. 1–31.

[27] Nebojša Gvozdenović and Monique Laurent. "The operator Ψ for the chromatic number of a graph". In: *SIAM Journal on Optimization* 19.2 (2008), pp. 572–591.

[28] Soodeh Habibi, Arefeh Kavand, Michal Kocvara, and Michael Stingl. "Barrier and penalty methods for low-rank semidefinite programming with application to truss topology design". In: (2021). arXiv: 2105.08529.

[29] Soodeh Habibi, Michal Kočvara, and Bernard Mourrain. "Polynomial Optimization in Geometric Modeling". In: *Polynomial Optimization, Moments, and Applications*. Ed. by Michal Kočvara, Bernard Mourrain, and Cordian Riener. Cham: Springer Nature Switzerland, 2023, pp. 181–206. DOI: 10.1007/978-3-031-38659-6_6.

[30] Soodeh Habibi, Michal Kočvara, and Michael Stingl. "Loraine – an interior-point solver for low-rank semidefinite programming". In: *Optimization Methods and Software* (2023), pp. 1–31. DOI: 10.1080/10556788.2023.2250522.

[31] Soodeh Habibi, Michal Kočvara, and Michael Stingl. "Solving Lasserre relaxations of unconstrained binary quadratic optimization problems by an interior-point method". submitted to Computational Optimization and Applications. 2023. URL: https://hal.science/hal-04076510.

[32] Christoph Helmberg and Franz Rendl. "A spectral bundle method for semidefinite programming". In: *SIAM Journal on Optimization* 10.3 (2000), pp. 673–696.

[33] Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge University Press, 2012.

[34] Roger A Horn and Charles R Johnson. *Topics in matrix analysis*. Cambridge University Press, 1991.

[35] Florian Jarre, Michal Kočvara, and Jochem Zowe. "Optimal truss design by interior-point methods". In: *SIAM Journal on Optimization* 8.4 (1998), pp. 1084–1107.

[36] M. Journée, F. Bach, P.-A. Absil, and R. Sepulchre. “Low-Rank Optimization on the Cone of Positive Semidefinite Matrices”. In: *SIAM Journal on Optimization* 20.5 (2010), pp. 2327–2351.

[37] Sunyoung Kim and Masakazu Kojima. “Binary quadratic optimization problems that are difficult to solve by conic relaxations”. In: *Discrete Optimization* 24 (2017), pp. 170–183.

[38] Sunyoung Kim, Masakazu Kojima, Martin Mevissen, and Makoto Yamashita. “Exploiting sparsity in linear and nonlinear matrix inequalities via positive semidefinite matrix completion”. In: *Mathematical Programming* 129.1 (2011), pp. 33–68.

[39] Gary Kochenberger, Jin-Kao Hao, Fred Glover, Mark Lewis, Zhipeng Lü, Haibo Wang, and Yang Wang. “The unconstrained binary quadratic programming problem: a survey”. In: *Journal of combinatorial optimization* 28.1 (2014), pp. 58–81.

[40] Michal Kočvara. “Chapter 11: Truss Topology Design by Conic Linear Optimization”. In: *Advances and Trends in Optimization with Engineering Applications*, pp. 135–147. DOI: 10.1137/1.9781611974683.ch11.

[41] Michal Kočvara. “Decomposition of arrow type positive semidefinite matrices with application to topology optimization”. In: *Mathematical Programming* (2020), pp. 1–30.

[42] Michal Kočvara and Michael Stingl. “On the solution of large-scale SDP problems by the modified barrier method using iterative solvers”. In: *Mathematical Programming* 109.2 (2007), pp. 413–444.

[43] Nathan Krislock, Jérôme Malick, and Frédéric Roupin. “Biqcrunch: A semidefinite branch-and-bound method for solving binary quadratic problems”. In: *ACM Transactions on Mathematical Software (TOMS)* 43.4 (2017), pp. 1–23.

[44] Nathan Krislock and Henry Wolkowicz. “Explicit sensor network localization using semidefinite representations and facial reductions”. In: *SIAM Journal on Optimization* 20.5 (2010), pp. 2679–2708.

[45] Jean-Louis Krivine. “Anneaux préordonnés”. In: *Journal d’analyse mathématique* 12 (1964), p. 307–326.

[46] J.B. Lasserre. *An Introduction to Polynomial and Semi-Algebraic Optimization*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2015.

[47] Jean B. Lasserre. “An Explicit Exact SDP Relaxation for Nonlinear 0-1 Programs”. In: June 2001, pp. 293–303.

[48] Jean B. Lasserre. “Global Optimization with Polynomials and the Problem of Moments”. In: *SIAM Journal on Optimization* 11.3 (2001), pp. 796–817.

[49] Jean B. Lasserre. *Moments, Positive Polynomials and Their Applications*. IMPERIAL COLLEGE PRESS, 2009.

[50] Jean B. Lasserre. “Polynomials nonnegative on a grid and discrete optimization”. In: *Transactions of the American Mathematical Society* 354 (Jan. 2002).

[51] Monique Laurent. “Lower bound for the number of iterations in semidefinite hierarchies for the cut polytope”. In: *Mathematics of operations research* 28.4 (2003), pp. 871–883.

[52] Monique Laurent. “Sums of Squares, Moment Matrices and Optimization Over Polynomials”. In: *Emerging Applications of Algebraic Geometry*. Ed. by Mihai Putinar and Seth Sullivant. New York, NY: Springer New York, 2009, pp. 157–270.

[53] Ramtin Madani, Abdulrahman Kalbat, and Javad Lavaei. “ADMM for sparse semidefinite programming with applications to optimal power flow problem”. In: *2015 54th IEEE Conference on Decision and Control (CDC)*. 2015, pp. 5932–5939.

[54] Jérôme Malick, Janez Povh, Franz Rendl, and Angelika Wiegele. “Regularization methods for semidefinite programming”. In: *SIAM Journal on Optimization* 20.1 (2009), pp. 336–356.

[55] Zoë Marschner, Paul Zhang, David Palmer, and Justin Solomon. “Sum-of-squares geometry processing”. In: *ACM Transactions on Graphics* 40.6 (Dec. 2021), pp. 1–13.

[56] H. D. Mittelmann. “An Independent Benchmarking of SDP and SOCP solvers”. In: *Mathematical Programming* 95 (2003), pp. 407–430.

[57] MOSEK ApS. *The MOSEK optimization toolbox for MATLAB manual. Version 9.0.* 2019. URL: http://docs.mosek.com/9.0/toolbox/index.html.

[58] Yurii Nesterov. “Squared Functional Systems and Optimization Problems”. In: *High Performance Optimization*. Ed. by Hans Frenk, Kees Roos, Tamás Terlaky, and Shuzhong Zhang. Boston, MA: Springer US, 2000, pp. 405–440.

[59] Danilo Elias Oliveira, Henry Wolkowicz, and Yangyang Xu. “ADMM for the SDP relaxation of the QAP”. In: *Mathematical Programming Computation* 10.4 (2018), pp. 631–658.

[60] Panos M Pardalos and Gregory P Rodgers. “Computational aspects of a branch and bound algorithm for quadratic zero-one programming”. In: *Computing* 45.2 (1990), pp. 131–144.

[61] Pablo Parrilo. “Semidefinite Programming Relaxations for Semialgebraic Problems”. In: *Mathematical Programming, Series B* 96 (2003).

[62] Pablo Parrilo. “Structured Semidenite Programs and Semialgebraic Geometry Methods in Robustness and Optimization”. In: *PhD thesis, California Institute of Technology* (2000).

[63] Frank Permenter and Pablo Parrilo. “Partial facial reduction: simplified, equivalent SDPs via approximations of the PSD cone”. In: *Mathematical Programming* 171.1 (2018), pp. 1–54.

[64] Franz Rendl, Giovanni Rinaldi, and Angelika Wiegele. “A branch and bound algorithm for Max-Cut based on combining semidefinite and polyhedral relaxations”. In: *International Conference on Integer Programming and Combinatorial Optimization*. Springer. 2007, pp. 295–309.

[65] Shinsaku Sakaue, Akiko Takeda, Sunyoung Kim, and Naoki Ito. “Exact SDP relaxations with truncated moment matrix for binary polynomial optimization problems”. In: *Proceedings of the 5th International Conference on Continuous Optimization (ICCOPT)*. 2016.

[66] Naum Zuselevich Shor. “An approach to obtaining global extremums in polynomial mathematical programming problems”. In: *Kibernetika* 5 (1987), pp. 102–106.

[67] Naum Zuselevich Shor. “Class of global minimum bounds of polynomial functions”. In: *Cybernetics* 23 (1987), pp. 731–734.

[68] Naum Zuselevich Shor. *Nondifferentiable optimization and polynomial problems.* Springer: Kluwer Academic Publishers, Dordrecht, 1998.

[69] Naum Zuselevich Shor. “Quadratic optimization problems”. In: *Tekhnicheskaya Kibernetika* (1987), pp. 128–139.

[70] Anthony Man-Cho So and Yinyu Ye. “Theory of semidefinite programming for sensor network localization”. In: *Mathematical Programming* 109.2 (2007), pp. 367–384.

[71] Gilbert Stengle. “A Nullstellensatz and a Positivstellensatz in Semialgebraic Geometry.” In: *Mathematische Annalen* 207 (1974), pp. 87–98.

[72] Michael Stingl, Michal Kočvara, and Günter Leugering. “A sequential convex semidefinite programming algorithm with an application to multiple-load free material optimization”. In: *SIAM Journal on Optimization* 20.1 (2009), pp. 130–155.

[73] Michael J. Todd. *Minimum-Volume Ellipsoids*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2016.

[74] Michael J. Todd, Kim-Chuan. Toh, and Reha H. Tütüncü. “On the Nesterov–Todd Direction in Semidefinite Programming”. In: *SIAM Journal on Optimization* 8.3 (1998), pp. 769–796.

[75] Kim-Chuan Toh. “Solving large scale semidefinite programs via an iterative solver on the augmented systems”. In: *SIAM Journal on Optimization* 14.3 (2004), pp. 670–698.

[76] Kim-Chuan Toh and Masakazu Kojima. “Solving Some Large Scale Semidefinite Programs via the Conjugate Residual Method”. In: *SIAM Journal on Optimization* 12.3 (2002), pp. 669–691.

[77] Lieven Vandenberghe, Stephen Boyd, and Shao-Po Wu. “Determinant maximization with linear matrix inequality constraints”. In: *SIAM Journal on Matrix Analysis and Applications* 19.2 (1998), pp. 499–533.

[78] Hayato Waki, Sunyoung Kim, Masakazu Kojima, Masakazu Muramatsu, and Hiroshi Sugimoto. “Algorithm 883: SparsePOP—a sparse semidefinite programming relaxation of polynomial optimization problems”. In: *ACM Transactions on Mathematical Software (TOMS)* 35.2 (2008), pp. 1–13.

[79] Jie Wang, Victor Magron, and Jean B. Lasserre. “TSSOS: A Moment-SOS hierarchy that exploits term sparsity”. In: *SIAM Journal on Optimization* 31.1 (2021), pp. 30–58.

[80] Jie Wang, Victor Magron, Jean B. Lasserre, and Ngoc Hoang Anh Mai. “CS-TSSOS: Correlative and term sparsity for large-scale polynomial optimization”. In: *arXiv preprint arXiv:2005.02828* (2020).

[81] Zizhuo Wang, Song Zheng, Yinyu Ye, and Stephen Boyd. “Further relaxations of the semidefinite programming approach to sensor network localization”. In: *SIAM Journal on Optimization* 19.2 (2008), pp. 655–673.

[82] Zaiwen Wen, Donald Goldfarb, and Wotao Yin. “Alternating direction augmented Lagrangian methods for semidefinite programming”. In: *Mathematical Programming Computation* 2.3 (2010), pp. 203–230.

[83] Liuqin Yang, Defeng Sun, and Kim-Chuan Toh. “SDPNAL+: a majorized semismooth Newton-CG augmented Lagrangian method for semidefinite programming with nonnegative constraints”. In: *Mathematical Programming Computation* 7.3 (2015), pp. 331–366.

[84] Alp Yurtsever, Joel A. Tropp, Olivier Fercoq, Madeleine Udell, and Volkan Cevher. “Scalable Semidefinite Programming”. In: *SIAM Journal on Mathematics of Data Science* (2020).

[85] Fuzhen Zhang. “Quaternions and matrices of quaternions”. In: *Linear Algebra and its Applications* 251 (1997), pp. 21–57.

[86] Richard Y. Zhang and Javad Lavaei. “Modified interior-point method for large-and-sparse low-rank semidefinite programs”. In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)* (Dec. 2017).

[87] Yin Zhang. “On Extending Some Primal–Dual Interior-Point Algorithms From Linear Programming to Semidefinite Programming”. In: *SIAM Journal on Optimization* 8.2 (1998), pp. 365–386.

[88] Xin-Yuan Zhao, Defeng Sun, and Kim-Chuan Toh. “A Newton-CG augmented Lagrangian method for semidefinite programming”. In: *SIAM Journal on Optimization* 20.4 (2010), pp. 1737–1765.

[89] Yuzixuan Zhu, Gábor Pataki, and Quoc Tran-Dinh. “Sieve-SDP: a simple facial reduction algorithm to preprocess semidefinite programs”. In: *Mathematical Programming Computation* 11.3 (2019), pp. 503–586.