A Robust Algorithm for Semidefinite Programming *

Xuan Vinh Doan †   Serge Kruk ‡   Henry Wolkowicz §

October 14, 2010

Contents

*This report is available at URL: orion.math.uwaterloo.ca˜hwolkowihenryreportsABSTRACTS.html

†Department of Combinatorics and Optimization, University of Waterloo, 200 University Avenue West, Waterloo, ON N2L 3G1, Canada

‡Dept. of Mathematics and Statistics, Oakland University, Rochester, MI

§Department of Combinatorics and Optimization, University of Waterloo, 200 University Avenue West, Waterloo, ON N2L 3G1, Canada

List of Tables

Abstract

Current successful methods for solving semidefinite programs, SDP, are based on primal-dual interior-point approaches. These usually involve a symmetrization step to allow for application of Newton's method followed by block elimination to reduce the size of the Newton equation. Both these steps create ill-conditioning in the Newton equation and singularity of the Jacobian at the optimum.

In order to avoid the ill-conditioning, we derive and test a numerically stable primal-dual interior-point method for SDP. We realize a distinct improvement in accuracy relative to current public domain software. This is true for random problems as well as for problems of special structure. The algorithm is based on a Gauss-Newton approach applied to a single bilinear form of the optimality conditions. The well-conditioned Jacobian allows for a preconditioned (matrix-free) iterative method for finding the search direction.

1 Introduction

We derive and test a backwards stable primal-dual interior-point method for semidefinite programming. We realize a distinct improvement in accuracy relative to current public domain software. This is for random problems as well as for problems of special structure. The algorithm is based on using a Gauss-Newton approach with a preconditioned iterative method for finding the search direction.

Primal-dual interior-point methods are currently the methods of choice for solving semidefinite programming, SDP, problems. However, current primal-dual interior-point methods are quite unstable. They cannot provide high accuracy solutions in general; and, they often fail for ill-conditioned problems. The instability arises from two steps. Since the optimality conditions for SDP are an overdetermined system of nonlinear equations, a symmetrization step is applied so that Newton's method can be used. This symmetrization changes a (possibly) well-posed problem into an ill-posed one where the Jacobian at optimality is singular. Then, block elimination is used in order to reduce the size of the resulting Newton system. However, this block elimination does not use any type of partial pivoting and again singularities are introduced. (This is discussed in [4, 15]. Further details are also given in Section 1.1, below.)

We study the Gauss-Newton method for solving SDPs and illustrate that high accuracy solutions can be obtained dependably for medium sized problems. We follow the approach in [7, 15] and use an inexact Gauss-Newton method to solve the perturbed optimality conditions.

1.1 Motivation and Central Problem

The primal-dual SDP pair we consider is

$$\text{(PSDP)} \qquad \begin{array}{rl} p^* := \min & \boldsymbol{C} \cdot \boldsymbol{X} \\ \text{s.t.} & \mathcal{A}(\boldsymbol{X}) = \boldsymbol{b}, \\ & \boldsymbol{X} \succeq 0, \end{array} \tag{1.1}$$

and

$$\text{(DSDP)} \qquad \begin{array}{rl} d^* := \max & \boldsymbol{b}^T \boldsymbol{y} \\ \text{s.t.} & \mathcal{A}^*(\boldsymbol{y}) + \boldsymbol{Z} = \boldsymbol{C}, \\ & \boldsymbol{Z} \succeq 0, \end{array} \tag{1.2}$$

where: $\boldsymbol{C}, \boldsymbol{X}, \boldsymbol{Z} \in \mathcal{S}^n$, $\mathcal{S}^n$ denotes the space of $n \times n$ real symmetric matrices equipped with the trace inner product, $\boldsymbol{C} \cdot \boldsymbol{D} = \text{trace}(\boldsymbol{C}\boldsymbol{D})$; and $\mathcal{A} : \mathcal{S}^n \to \mathbb{R}^m$ is a linear transformation, with $\mathcal{A}^*$ its adjoint transformation.

Under a suitable *constraint qualification* assumption such as Slater's condition or strict feasibility, the primal-dual solution $(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{Z})$ with $\boldsymbol{X}, \boldsymbol{Z} \succeq 0$ is optimal for the primal-dual pair (1.1) and (1.2) if and only if

$$F(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{Z}) := \begin{pmatrix} \mathcal{A}^*(\boldsymbol{y}) + \boldsymbol{Z} - \boldsymbol{C} \\ \mathcal{A}(\boldsymbol{X}) - \boldsymbol{b} \\ \boldsymbol{Z}\boldsymbol{X} \end{pmatrix} = \boldsymbol{0}. \tag{1.3}$$

Primal-dual interior-point methods maintain $\boldsymbol{X}, \boldsymbol{Z} \succ 0$, and with the *barrier parameter* $\mu \downarrow 0$, they find approximate solutions to the following perturbed optimality conditions:

$$F_\mu(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{Z}) := \begin{pmatrix} \mathcal{A}^*(\boldsymbol{y}) + \boldsymbol{Z} - \boldsymbol{C} \\ \mathcal{A}(\boldsymbol{X}) - \boldsymbol{b} \\ \boldsymbol{Z}\boldsymbol{X} - \mu \boldsymbol{I} \end{pmatrix} = \boldsymbol{0}, \quad \mu > 0. \tag{1.4}$$

Thus, these methods are based on *path following*. The perturbed system in (1.4) is overdetermined. Under nondegeneracy assumptions, the Jacobian is full rank at optimality, e.g., [1]. Current methods use Newton's method applied to various symmetrizations of (1.4). The two most popular symmetrizations are the so-called HKM and NT methods, see for example, [13]. But, the linearizations (Jacobian) of the symmetrized optimality conditions for both of these methods is singular at an optimum. This and the block elimination schemes used for finding the search directions both imply that one is solving an increasingly ill-conditioned linear system to find a search direction. Thus, it is extremely difficult to obtain high accuracy solutions; these algorithms are not backwards stable under finite precision arithmetic. Similarly, finding reasonable preconditioners for iterative methods is difficult if not impossible. In addition, the block eliminations make it difficult to exploit sparsity in the data. In this paper we propose a robust primal-dual interior/exterior-point method which uses an inexact Gauss-Newton approach with a matrix-free preconditioned conjugate gradient method. We do not change a well-conditioned system of optimality conditions to an ill-conditioned system. The method is able to attain high accuracy solutions as well as exploit sparsity.

1.2 Outline

The Gauss-Newton approach is described in Section 2. This includes a discussion on the conditioning in Section 2.3, proof of asymptotic convergence in Section 2.4, and a description of the preconditioning techniques in Section 2.5.

We apply our techniques to the Lovász Theta Function Problem in Section 3. Numerics and concluding remarks are given in Section 4.

2 Gauss-Newton Method

2.1 Matrix-Free Formulation

Let us consider the primal-dual SDP pair in (1.1) and (1.2). We assume the linear transformation $\mathcal{A}$ has full rank m. It can be represented as

$$\mathcal{A}(\boldsymbol{X}) = \boldsymbol{b} \Leftrightarrow \boldsymbol{A}_i \cdot \boldsymbol{X} = b_i, \quad \forall\, i = 1, \ldots, m.$$

The adjoint transformation $\mathcal{A}^*$ is then defined as $\mathcal{A}^*(\boldsymbol{y}) = \sum_{i=1}^{m} y_i \boldsymbol{A}_i$, for all $\boldsymbol{y} \in \mathbb{R}^m$. Here $\boldsymbol{A}_i \in \mathcal{S}^n, \forall i$.

The mapping vec : $\mathcal{M}^n \to \mathbb{R}^{n^2}$, where $\mathcal{M}^n$ is the set of all square $n \times n$ matrices, takes a matrix $\boldsymbol{M} \in \mathcal{M}^n$ and forms a vector $\boldsymbol{v} \in \mathbb{R}^{n^2}$ from its columns. The inverse mapping is Mat $:= \text{vec}^{-1}$, which takes a vector $\boldsymbol{v} \in \mathbb{R}^{n^2}$ and forms a matrix $\boldsymbol{M} \in \mathcal{M}^n$ column by column. Indeed $\boldsymbol{M} = \text{Mat}(\text{vec}(\boldsymbol{M}))$, for all $\boldsymbol{M} \in \mathcal{M}^n$.

We also define the triangular number, $t(n) = n(n+1)/2$ and the mapping svec : $\mathcal{S}^n \to \mathbb{R}^{t(n)}$ that takes a symmetric matrix $\boldsymbol{S} \in \mathcal{S}^n$ and forms a vector $\boldsymbol{v} \in \mathbb{R}^{t(n)}$ by concatenating n vectors $\boldsymbol{s}_{j-1} = \sqrt{2}\,(s_{ij})_{1 \le i < j}$ for all $j = 2, \ldots, n$ and $\boldsymbol{s}_n = \text{diag}(\boldsymbol{S})$. This mapping is an isometry under the 2-norm. The inverse mapping sMat $= \text{svec}^{-1}$ maps a vector $\boldsymbol{v} \in \mathbb{R}^{t(n)}$ into a symmetric matrix $\boldsymbol{S} \in \mathcal{S}^n$. We also have $\text{sMat}^* = \text{svec}$, since

$$\langle \text{sMat}(\boldsymbol{v}), \boldsymbol{S} \rangle = \text{trace}\,(\text{sMat}(\boldsymbol{v})\boldsymbol{S}) = \boldsymbol{v}^T \text{svec}(\boldsymbol{S}) = \langle \boldsymbol{v}, \text{svec}(\boldsymbol{S}) \rangle.$$

From setting $\boldsymbol{A} \in \mathbb{R}^{m \times t(n)}$ with rows $\boldsymbol{A}_{i,:} = \text{svec}(\boldsymbol{A}_i)$, for all $i = 1, \ldots, m$, we have:

$$\mathcal{A}(\boldsymbol{X}) = \boldsymbol{b} \Leftrightarrow \boldsymbol{A}\,\text{svec}(\boldsymbol{X}) = \boldsymbol{b}.$$

The nullspace of $\boldsymbol{A}$, null$(\boldsymbol{A})$, has dimension $t(n) - m$. Let us consider an orthonormal basis $\{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_{t(n)-m}\}$ of null$(\boldsymbol{A})$ and assume that we could find a primal feasible solution $\hat{\boldsymbol{X}} \succ 0$. Then

$$\mathcal{A}(\boldsymbol{X}) = \boldsymbol{b} \Leftrightarrow \text{svec}(\boldsymbol{X}) = \hat{\boldsymbol{x}} + \boldsymbol{Q}\boldsymbol{v},$$

where $\hat{\boldsymbol{x}} = \text{svec}(\hat{\boldsymbol{X}})$, $\boldsymbol{v} \in \mathbb{R}^{t(n)-m}$ and the columns of $\boldsymbol{Q} \in \mathbb{R}^{t(n) \times (t(n)-m)}$ are taken from the basis of null$(\boldsymbol{A})$.

Let us consider now the dual feasibility condition:

$$\mathcal{A}^*(\boldsymbol{y}) + \boldsymbol{Z} = \boldsymbol{C} \Leftrightarrow \operatorname{svec}(\boldsymbol{Z}) = \boldsymbol{c} - \boldsymbol{A}^T\boldsymbol{y},$$

where $\boldsymbol{c} = \operatorname{svec}(\boldsymbol{C})$. The optimality conditions in (1.4) are now equivalent to the following conditions:

$$G_\mu(\boldsymbol{v},\boldsymbol{y}) := \operatorname{sMat}(\boldsymbol{c} - \boldsymbol{A}^T\boldsymbol{y})\operatorname{sMat}(\hat{\boldsymbol{x}} + \boldsymbol{Q}\boldsymbol{v}) - \mu\boldsymbol{I} = \boldsymbol{0} \quad (\Leftrightarrow \operatorname{vec}(G_\mu(\boldsymbol{v},\boldsymbol{y})) = \boldsymbol{0}). \tag{2.1}$$

This is a single bilinear overdetermined system with $t(n)$ variables and n^2 equations and can be solved by the Gauss-Newton method. For each $\mu > 0$, there exists a unique primal-dual solution $(\boldsymbol{X}_\mu, \boldsymbol{y}_\mu, \boldsymbol{Z}_\mu)$ of (1.4), with $\boldsymbol{X}_\mu \succ 0, \boldsymbol{Z}_\mu \succ 0$, that lies on (and thus defines) the *central path*. The corresponding $\operatorname{svec}(\boldsymbol{Z}_\mu) = c - A^T\boldsymbol{y}_\mu$ and $\operatorname{svec}(\boldsymbol{X}_\mu) = \hat{\boldsymbol{x}} + \boldsymbol{Q}\boldsymbol{v}_\mu$, for appropriate $\boldsymbol{v}_\mu$, uniquely solves (2.1).

Assume that we can find a dual feasible solution $\hat{\boldsymbol{y}}$ such that $\hat{\boldsymbol{Z}} = \boldsymbol{C} - \mathcal{A}^*\hat{\boldsymbol{y}} \succ 0$. Then $(\boldsymbol{0}, \hat{\boldsymbol{y}})$ can be used as the initial solution $(\boldsymbol{v}_0, \boldsymbol{y}_0)$ for the Gauss-Newton method. For each iteration, the search direction $(\Delta\boldsymbol{v}, \Delta\boldsymbol{y})$ is calculated by (approximately) finding the least-squares solution of the Gauss-Newton equation

$$-G_\mu(\boldsymbol{v},\boldsymbol{y}) = G'_\mu(\boldsymbol{v},\boldsymbol{y})\begin{pmatrix}\Delta\boldsymbol{v}\\ \Delta\boldsymbol{y}\end{pmatrix}, \tag{2.2}$$

where $G'_\mu(\boldsymbol{v},\boldsymbol{y}) : \mathbb{R}^{t(n)-m} \times \mathbb{R}^m \to \mathcal{M}^n$. We now compute the Jacobian $J := G'_\mu$ and its adjoint J^*.

For all $\boldsymbol{v} \in \mathbb{R}^{t(n)-m}$, $\boldsymbol{y} \in \mathbb{R}^m$, define $\mathcal{X} : \mathbb{R}^m \to \mathcal{M}^n$ and $\mathcal{Z} : \mathbb{R}^{t(n)-m} \to \mathcal{M}^n$ by:

$$\begin{array}{ll} \boldsymbol{Z} := \boldsymbol{C} - \operatorname{sMat}(\boldsymbol{A}^T\boldsymbol{y}), & \boldsymbol{X} := \hat{\boldsymbol{X}} + \operatorname{sMat}(\boldsymbol{Q}\boldsymbol{v}), \\ \mathcal{Z}(\boldsymbol{v}) := \boldsymbol{Z}\operatorname{sMat}(\boldsymbol{Q}\boldsymbol{v}), & \mathcal{X}(\boldsymbol{y}) := -\operatorname{sMat}(\boldsymbol{A}^T\boldsymbol{y})\boldsymbol{X}. \end{array} \tag{2.3}$$

We then have $J = [\mathcal{Z} \mid \mathcal{X}]$ and the Gauss-Newton equation can be written as follows:

$$\begin{array}{rcl} -G_\mu(\boldsymbol{v},\boldsymbol{y}) &=& \mathcal{Z}(\Delta\boldsymbol{v}) + \mathcal{X}(\Delta\boldsymbol{y}) \\ &=& \boldsymbol{Z}\operatorname{sMat}(\boldsymbol{Q}\Delta\boldsymbol{v}) - \operatorname{sMat}(\boldsymbol{A}^T\Delta\boldsymbol{y})\boldsymbol{X} \\ &=& \left[\boldsymbol{C} - \operatorname{sMat}(\boldsymbol{A}^T\boldsymbol{y})\right]\operatorname{sMat}(\boldsymbol{Q}\Delta\boldsymbol{v}) - \operatorname{sMat}(\boldsymbol{A}^T\Delta\boldsymbol{y})\left[\hat{\boldsymbol{X}} + \operatorname{sMat}(\boldsymbol{Q}\boldsymbol{v})\right]. \end{array} \tag{2.4}$$

This is again an overdetermined (linear) system with $t(n)$ decision variables $(\Delta\boldsymbol{v}, \Delta\boldsymbol{y})$, and with n^2 equations. In order to find the least squares solution, we need to compute $J^* = \begin{bmatrix}\mathcal{Z}^*\\ \mathcal{X}^*\end{bmatrix}$ and

$$J^* \circ J = \begin{bmatrix}\mathcal{Z}^* \circ \mathcal{Z} & \mathcal{Z}^* \circ \mathcal{X}\\ \mathcal{X}^* \circ \mathcal{Z} & \mathcal{X}^* \circ \mathcal{X}\end{bmatrix}, \tag{2.5}$$

since the final system of linear equations we (implicitly) solve is the normal equations

$$J^* \circ J\begin{pmatrix}\Delta\boldsymbol{v}\\ \Delta\boldsymbol{y}\end{pmatrix} = -J^* \circ G_\mu(\boldsymbol{v},\boldsymbol{y}).$$

Consider $\boldsymbol{M} \in \mathcal{M}^n$, we have:

$$\begin{array}{rcl} \langle \boldsymbol{M}, \mathcal{X}(\boldsymbol{y})\rangle &=& -\operatorname{trace}(\boldsymbol{M}^T\operatorname{sMat}(\boldsymbol{A}^T\boldsymbol{y})\boldsymbol{X}) \\ &=& -\operatorname{trace}(\operatorname{sMat}(\boldsymbol{A}^T\boldsymbol{y})(\boldsymbol{X}\boldsymbol{M}^T)) \\ &=& -\operatorname{trace}\left(\operatorname{sMat}(\boldsymbol{A}^T\boldsymbol{y})\frac{1}{2}\left(\boldsymbol{X}\boldsymbol{M}^T + \boldsymbol{M}\boldsymbol{X}\right)\right) \\ &=& -\frac{1}{2}\boldsymbol{y}^T\boldsymbol{A}\operatorname{svec}(\boldsymbol{X}\boldsymbol{M}^T + \boldsymbol{M}\boldsymbol{X}) \\ &=& \langle \boldsymbol{y}, -\frac{1}{2}\boldsymbol{A}\operatorname{svec}(\boldsymbol{X}\boldsymbol{M}^T + \boldsymbol{M}\boldsymbol{X})\rangle, \end{array}$$

since $\mathrm{sMat}(\boldsymbol{A}^T\boldsymbol{y})\boldsymbol{X} \in \mathcal{M}^n$, $\mathrm{sMat}(\boldsymbol{A}^T\boldsymbol{y}), \boldsymbol{X} \in \mathcal{S}^n$, and in addition $\boldsymbol{X}\boldsymbol{M}^T + \boldsymbol{M}\boldsymbol{X} \in \mathcal{S}^n$. Thus

$$\mathcal{X}^*(\boldsymbol{M}) = -\frac{1}{2}\boldsymbol{A}\,\mathrm{svec}(\boldsymbol{X}\boldsymbol{M}^T + \boldsymbol{M}\boldsymbol{X}). \tag{2.6}$$

Similarly, we have:

$$\begin{aligned}
\langle \boldsymbol{M}, \mathcal{Z}(\boldsymbol{v})\rangle &= \mathrm{trace}(\boldsymbol{M}^T\boldsymbol{Z}\,\mathrm{sMat}(\boldsymbol{Q}\boldsymbol{v})) \\
&= \mathrm{trace}(\mathrm{sMat}(\boldsymbol{Q}\boldsymbol{v})(\boldsymbol{M}^T\boldsymbol{Z})) \\
&= \mathrm{trace}\left(\mathrm{sMat}(\boldsymbol{Q}\boldsymbol{v})\frac{1}{2}(\boldsymbol{M}^T\boldsymbol{Z} + \boldsymbol{Z}\boldsymbol{M})\right) \\
&= \frac{1}{2}\boldsymbol{v}^T\boldsymbol{Q}^T\,\mathrm{svec}(\boldsymbol{M}^T\boldsymbol{Z} + \boldsymbol{Z}\boldsymbol{M}) \\
&= \langle \boldsymbol{v}, \frac{1}{2}\boldsymbol{Q}^T\,\mathrm{svec}(\boldsymbol{M}^T\boldsymbol{Z} + \boldsymbol{Z}\boldsymbol{M})\rangle,
\end{aligned}$$

since $\boldsymbol{M}^T\boldsymbol{Z} \in \mathcal{M}^n$, $\mathrm{sMat}(\boldsymbol{Q}\boldsymbol{v}), \boldsymbol{Z} \in \mathcal{S}^n$, and in addition $\boldsymbol{M}^T\boldsymbol{Z} + \boldsymbol{Z}\boldsymbol{M} \in \mathcal{S}^n$. Thus

$$\mathcal{Z}^*(\boldsymbol{M}) = \frac{1}{2}\boldsymbol{Q}^T\,\mathrm{svec}(\boldsymbol{M}^T\boldsymbol{Z} + \boldsymbol{Z}\boldsymbol{M}). \tag{2.7}$$

Finally, the main operator J^*J that is implicitly used for solving the Gauss-Newton equation (2.2) can be computed using the composition of $\mathcal{Z}^*$ and $\mathcal{X}^*$ (and $\mathcal{Z}$ and $\mathcal{X}$). We get

$$\begin{aligned}
\mathcal{Z}^* \circ \mathcal{Z}(\boldsymbol{v}) &= \tfrac{1}{2}\boldsymbol{Q}^T\,\mathrm{svec}\left(\mathrm{sMat}(\boldsymbol{Q}\boldsymbol{v})\boldsymbol{Z}^2 + \boldsymbol{Z}^2\,\mathrm{sMat}(\boldsymbol{Q}\boldsymbol{v})\right) \\
\mathcal{X}^* \circ \mathcal{X}(\boldsymbol{y}) &= \tfrac{1}{2}\boldsymbol{A}\,\mathrm{svec}\left(\mathrm{sMat}(\boldsymbol{A}^T\boldsymbol{y})\boldsymbol{X}^2 + \boldsymbol{X}^2\,\mathrm{sMat}(\boldsymbol{A}^T\boldsymbol{y})\right) \\
\mathcal{Z}^* \circ \mathcal{X}(\boldsymbol{y}) &= \tfrac{1}{2}\boldsymbol{Q}^T\,\mathrm{svec}\left(\boldsymbol{Z}\,\mathrm{sMat}(\boldsymbol{A}^T\boldsymbol{y})\boldsymbol{X} + \boldsymbol{X}\,\mathrm{sMat}(\boldsymbol{A}^T\boldsymbol{y})\boldsymbol{Z}\right) \\
\mathcal{X}^* \circ \mathcal{Z}(\boldsymbol{v}) &= \tfrac{1}{2}\boldsymbol{A}\,\mathrm{svec}\left(\boldsymbol{X}\,\mathrm{sMat}(\boldsymbol{Q}\boldsymbol{v})\boldsymbol{Z} + \boldsymbol{Z}\,\mathrm{sMat}(\boldsymbol{Q}\boldsymbol{v})\boldsymbol{X}\right);
\end{aligned} \tag{2.8}$$

and the result, in the form of a congruence using $U = \begin{bmatrix} \boldsymbol{Q}^T\,\mathrm{svec} & 0 \\ 0 & \boldsymbol{A}\,\mathrm{svec} \end{bmatrix}$, is

$$J^* \circ J = \frac{1}{2}\begin{bmatrix} \boldsymbol{Q}^T\,\mathrm{svec} & 0 \\ 0 & \boldsymbol{A}\,\mathrm{svec} \end{bmatrix}\begin{bmatrix} (\cdot)\boldsymbol{Z}^2 + \boldsymbol{Z}^2(\cdot) & \boldsymbol{Z}(\cdot)\boldsymbol{X} + \boldsymbol{X}(\cdot)\boldsymbol{Z} \\ \boldsymbol{X}(\cdot)\boldsymbol{Z} + \boldsymbol{Z}(\cdot)\boldsymbol{X} & (\cdot)\boldsymbol{X}^2 + \boldsymbol{X}^2(\cdot) \end{bmatrix}\begin{bmatrix} \boldsymbol{Q}^T\,\mathrm{svec} & 0 \\ 0 & \boldsymbol{A}\,\mathrm{svec} \end{bmatrix}^*. \tag{2.9}$$

To exploit this structure, we introduce the two matrices of squared matrices

$$\boldsymbol{A}^{(2)} := \begin{bmatrix} \mathrm{svec}(\boldsymbol{A}_1^2)^T \\ \cdots \\ \mathrm{svec}(\boldsymbol{A}_i^2)^T \\ \cdots \\ \mathrm{svec}(\boldsymbol{A}_m^2)^T \end{bmatrix}, \qquad \boldsymbol{Q}^{(2)} := \begin{bmatrix} \mathrm{svec}(\boldsymbol{Q}_1^2) & \ldots & \mathrm{svec}(\boldsymbol{Q}_i^2) & \ldots & \mathrm{svec}(\boldsymbol{Q}_{t(n)-m}^2) \end{bmatrix}.$$

2.2 Algorithm Initialization

The approach described in the previous sections assumes that we have a feasible solution $(\hat{\boldsymbol{X}}, \hat{\boldsymbol{Z}})$. However, finding a feasible solution $(\hat{\boldsymbol{X}}, \hat{\boldsymbol{Z}})$ to start the algorithm is not an easy task; and, in general, it is as hard as solving the problem completely. In this section, we propose an algorithm with infeasible initial solutions. Given an initial (infeasible) solution $(\boldsymbol{X}_0, \boldsymbol{Z}_0)$ with $\boldsymbol{X}_0, \boldsymbol{Z}_0 \succ 0$, and corresponding $\boldsymbol{x}_0 = \mathrm{svec}(\boldsymbol{X}_0)$ and $\boldsymbol{z}_0 = \mathrm{svec}(\boldsymbol{Z}_0)$, we would like to find an optimal (and clearly feasible) solution $(\boldsymbol{X}^*, \boldsymbol{Z}^*)$. The feasibility of a solution $(\boldsymbol{X}, \boldsymbol{Z})$ or equivalently, $(\boldsymbol{x}, \boldsymbol{z})$, is equivalent to the existence of $(\boldsymbol{v}, \boldsymbol{y})$ such that $\boldsymbol{x} = \hat{\boldsymbol{x}} + \boldsymbol{Q}\boldsymbol{v}$ and $\boldsymbol{z} = \boldsymbol{c} - \boldsymbol{A}^T\boldsymbol{y}$, where $\hat{\boldsymbol{x}} = \boldsymbol{A}^T \left(\boldsymbol{A}\boldsymbol{A}^T\right)^{-1} \boldsymbol{b}$, the least-squares solution of $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$. Since we start the algorithm with an infeasible solution, the perturbed optimality conditions need to be written with $(\boldsymbol{v}, \boldsymbol{y}, \boldsymbol{x}, \boldsymbol{z})$ as decision variables:

$$F_\mu(\boldsymbol{v}, \boldsymbol{y}, \boldsymbol{x}, \boldsymbol{z}) := \begin{pmatrix} \boldsymbol{z} - \boldsymbol{c} + \boldsymbol{A}^T\boldsymbol{y} \\ \boldsymbol{x} - \hat{\boldsymbol{x}} - \boldsymbol{Q}\boldsymbol{v} \\ \operatorname{sMat}(\boldsymbol{z}) \operatorname{sMat}(\boldsymbol{x}) - \mu \boldsymbol{I} \end{pmatrix} =: \begin{pmatrix} \boldsymbol{r}_d \\ \boldsymbol{r}_x \\ \boldsymbol{R}_c \end{pmatrix} = \boldsymbol{0}. \tag{2.10}$$

To start the Gauss-Newton algorithm, the initial solution for $\boldsymbol{v}$ and $\boldsymbol{y}$ could simply be $\boldsymbol{v}_0 = \boldsymbol{0}$ and $\boldsymbol{y}_0 = \boldsymbol{0}$. In each iteration, the search direction $(\Delta\boldsymbol{v}, \Delta\boldsymbol{y}, \Delta\boldsymbol{x}, \Delta\boldsymbol{z})$ is calculated by (approximately) finding the least-squares solution of the Gauss-Newton equation

$$F_\mu'(\boldsymbol{v}, \boldsymbol{y}, \boldsymbol{x}, \boldsymbol{z}) \begin{pmatrix} \Delta\boldsymbol{v} \\ \Delta\boldsymbol{y} \\ \Delta\boldsymbol{x} \\ \Delta\boldsymbol{z} \end{pmatrix} = -F_\mu(\boldsymbol{v}, \boldsymbol{y}, \boldsymbol{x}, \boldsymbol{z}) = - \begin{pmatrix} \boldsymbol{r}_d \\ \boldsymbol{r}_x \\ \boldsymbol{R}_c \end{pmatrix}.$$

The above linearization yields

$$\begin{pmatrix} \Delta\boldsymbol{z} + \boldsymbol{A}^T\Delta\boldsymbol{y} \\ \Delta\boldsymbol{x} - \boldsymbol{Q}\Delta\boldsymbol{v} \\ \boldsymbol{Z} \operatorname{sMat}(\Delta\boldsymbol{x}) + \operatorname{sMat}(\Delta\boldsymbol{z})\boldsymbol{X} \end{pmatrix} = -F_\mu(\boldsymbol{v}, \boldsymbol{y}, \boldsymbol{x}, \boldsymbol{z}) = - \begin{pmatrix} \boldsymbol{r}_d \\ \boldsymbol{r}_x \\ \boldsymbol{R}_c \end{pmatrix},$$

where $\boldsymbol{X} = \operatorname{sMat}(\boldsymbol{x})$ and $\boldsymbol{Z} = \operatorname{sMat}(\boldsymbol{z})$.

We can now use the first two equations and substitute for $\Delta\boldsymbol{x} = \boldsymbol{Q}\Delta\boldsymbol{v} - \boldsymbol{r}_x$ and $\Delta\boldsymbol{z} = -\boldsymbol{A}^T\Delta\boldsymbol{y} - \boldsymbol{r}_d$ in the last equation to get:

$$G_\mu'(\boldsymbol{v}, \boldsymbol{y}) \begin{pmatrix} \Delta\boldsymbol{v} \\ \Delta\boldsymbol{y} \end{pmatrix} := \boldsymbol{Z} \operatorname{sMat}(\boldsymbol{Q}\Delta\boldsymbol{v} - \boldsymbol{r}_x) - \operatorname{sMat}(\boldsymbol{A}^T\Delta\boldsymbol{y} + \boldsymbol{r}_d)\boldsymbol{X} = -\boldsymbol{R}_c.$$

Moving constants to the right-hand side yields the equation

$$\boldsymbol{Z} \operatorname{sMat}(\boldsymbol{Q}\Delta\boldsymbol{v}) - \operatorname{sMat}(\boldsymbol{A}^T\Delta\boldsymbol{y})\boldsymbol{X} = \boldsymbol{Z} \operatorname{sMat}(\boldsymbol{r}_x) + \operatorname{sMat}(\boldsymbol{r}_d)\boldsymbol{X} - \boldsymbol{R}_c. \tag{2.11}$$

This system of equations is similar to (2.4) and we can solve it again using the Jacobian $J = [\mathcal{Z} \mid \mathcal{X}]$ and its adjoint J^*, where as above we have $\mathcal{X}(\boldsymbol{y}) = -\operatorname{sMat}(\boldsymbol{A}^T\boldsymbol{y})\boldsymbol{X}$, $\forall \boldsymbol{y} \in \mathbb{R}^m$, and $\mathcal{Z}(\boldsymbol{v}) = \boldsymbol{Z} \operatorname{sMat}(\boldsymbol{Q}\boldsymbol{v})$, $\forall \boldsymbol{v} \in \mathbb{R}^{t(n)-m}$.

We can also check the change in the residual when taking a step α. We have, $\boldsymbol{x} + \alpha_p\Delta\boldsymbol{x} = \boldsymbol{x} + \alpha_p(\boldsymbol{Q}\Delta\boldsymbol{v} - \boldsymbol{r}_x)$ and $\boldsymbol{z} + \alpha_d\Delta\boldsymbol{z} = \boldsymbol{z} - \alpha_d(\boldsymbol{A}^T\Delta\boldsymbol{y} + \boldsymbol{r}_d)$. The new residuals are

$$\boldsymbol{r}_d \leftarrow (\boldsymbol{z} - \alpha_d(\boldsymbol{A}^T\Delta\boldsymbol{y} + \boldsymbol{r}_d)) - \boldsymbol{c} + \boldsymbol{A}^T(\boldsymbol{y} + \alpha_d\Delta\boldsymbol{y}) = (1 - \alpha_d)\boldsymbol{r}_d,$$

$$\boldsymbol{r}_x \leftarrow \boldsymbol{x} + \alpha_p(\boldsymbol{Q}\Delta\boldsymbol{v} - \boldsymbol{r}_x) - \hat{\boldsymbol{x}} - \boldsymbol{Q}(\boldsymbol{v} + \alpha_p\Delta\boldsymbol{v}) = (1 - \alpha_p)\boldsymbol{r}_p.$$

This emphasizes the importance of taking a steplength of $\alpha_p = \alpha_d = 1$ early in the algorithm as we can then obtain exact feasibility from that point on.

2.3 Conditioning of the Gauss-Newton System

2.4 Local Convergence

In this section, we would like to study the convergence of the Gauss-Newton method with infeasible initial solutions. Consider the following general problem

$$F(\boldsymbol{x}, \boldsymbol{y}) = \begin{pmatrix} L(\boldsymbol{x}, \boldsymbol{y}) \\ R(\boldsymbol{x}) \end{pmatrix} = \mathbf{0},$$

where $L(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x} - \hat{\boldsymbol{x}} - \boldsymbol{T}\boldsymbol{y}$ is a linear function in $\boldsymbol{x}$ and $\boldsymbol{y}$, $\boldsymbol{T}$ is full-rank. We would like to solve the following linearized system of equations, and its equivalent system, in each iteration.

$$\left\{ F'(\boldsymbol{x}, \boldsymbol{y}) \begin{pmatrix} \Delta\boldsymbol{x} \\ \Delta\boldsymbol{y} \end{pmatrix} = -F(\boldsymbol{x}, \boldsymbol{y}) \right\} \Leftrightarrow \left\{ \begin{array}{l} \Delta\boldsymbol{x} - \boldsymbol{T}\Delta\boldsymbol{y} = -L(\boldsymbol{x}, \boldsymbol{y}) \\ R'(\boldsymbol{x})\Delta\boldsymbol{x} = -R(\boldsymbol{x}) \end{array} \right\}. \tag{2.12}$$

Thus $\Delta\boldsymbol{x} = \boldsymbol{T}\Delta\boldsymbol{y} - L(\boldsymbol{x}, \boldsymbol{y})$ and the second equation becomes:

$$R'(\boldsymbol{x})\boldsymbol{T}\Delta\boldsymbol{y} = -R(\boldsymbol{x}) + R'(\boldsymbol{x})L(\boldsymbol{x}, \boldsymbol{y}),$$

which is assumed to be an overdetermined system. We use the Gauss-Newton method to solve this equation and update $(\boldsymbol{x}, \boldsymbol{y})$ with $(\Delta\boldsymbol{x}, \Delta\boldsymbol{y})$. We obtain the following local convergence result.

Theorem 1. *Let R, L, T be defined as in (2.12), with $J_x(\boldsymbol{x}) := R'(\boldsymbol{x})$ and $J_y(\boldsymbol{x}) := J_x(\boldsymbol{x})\boldsymbol{T}$. Let $R(\boldsymbol{x})$ be twice continuous differentiable in an open convex set D. Let $J_x(\boldsymbol{x}) \in Lip_\gamma(D)$ with $||J_x(\boldsymbol{x})|| \le \alpha$, for all $\boldsymbol{x} \in D$; and, suppose that there exists $\boldsymbol{x}_* \in D$, $\boldsymbol{y}_*$, and $\lambda, \sigma \ge 0$, such that $L(\boldsymbol{x}_*, \boldsymbol{y}_*) = \mathbf{0}$, $J_x(\boldsymbol{x}_*)^T R(\boldsymbol{x}_*) = \mathbf{0}$, λ is the smallest eigenvalue of $J_y(\boldsymbol{x}_*)^T J_y(\boldsymbol{x}_*)$, and*

$$\left|\left|(J_x(\boldsymbol{x}) - J_x(\boldsymbol{x}_*))^T R(\boldsymbol{x}_*)\right|\right|_2 \le \sigma \left|\left|\boldsymbol{x} - \boldsymbol{x}_*\right|\right|_2, \quad \forall \boldsymbol{x} \in D.$$

Define the iteration

$$\begin{array}{ll} \Delta\boldsymbol{x}_k = \boldsymbol{T}\Delta\boldsymbol{y}_k - L(\boldsymbol{x}_k, \boldsymbol{y}_k), \forall k, & \Delta\boldsymbol{y}_k = (J_y(\boldsymbol{x}_k)^T J_y(\boldsymbol{x}_k))^{-1} J_y(\boldsymbol{x}_k)^T(-R(\boldsymbol{x}_k) + J_x(\boldsymbol{x}_k)L(\boldsymbol{x}_k, \boldsymbol{y}_k)), \\ \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta\boldsymbol{x}_k, & \boldsymbol{y}_{k+1} = \boldsymbol{y}_k + \Delta\boldsymbol{y}_k. \end{array}$$

Let $\tau := ||\boldsymbol{T}||_2$ and $c \in (1, \lambda/(\tau^2\sigma))$. If $\tau^2\sigma < \lambda$, then there exists $\epsilon > 0$ such that $\forall \boldsymbol{x}_0 \in N(\boldsymbol{x}_*, \epsilon)$ and $\boldsymbol{y}_0$ arbitrary, we have that: the sequences $\{x_k\}, \{y_k\}$ are well-defined; they converge to $(\boldsymbol{x}_*, \boldsymbol{y}_*)$ with $L(\boldsymbol{x}_k, \boldsymbol{y}_k) = \mathbf{0}, \forall k \ge 1$; and

$$||\boldsymbol{x}_{k+1} - \boldsymbol{x}_*|| \le \frac{\tau^2 c}{\lambda} \left(\sigma \left|\left|\boldsymbol{x}_k - \boldsymbol{x}_*\right|\right| + \frac{\alpha\gamma}{2} \left|\left|\boldsymbol{x}_k - \boldsymbol{x}_*\right|\right|^2 \right),$$

$$||\boldsymbol{x}_{k+1} - \boldsymbol{x}_*|| \le \frac{\lambda + \tau^2 c\sigma}{2\lambda} \left|\left|\boldsymbol{x}_k - \boldsymbol{x}_*\right|\right| < \left|\left|\boldsymbol{x}_k - \boldsymbol{x}_*\right|\right|.$$

Proof. Using induction and the formulations $\Delta\boldsymbol{y}_k = (J_y(\boldsymbol{x}_k)^T J_y(\boldsymbol{x}_k))^{-1} J_y(\boldsymbol{x}_k)^T(-R(\boldsymbol{x}_k) + J_x(\boldsymbol{x}_k)L(\boldsymbol{x}_k, \boldsymbol{y}_k))$, $\Delta\boldsymbol{x}_k = \boldsymbol{T}\Delta\boldsymbol{y}_k - L(\boldsymbol{x}_k, \boldsymbol{y}_k)$, we see that $L(\boldsymbol{x}_k, \boldsymbol{y}_k) = \mathbf{0}, \forall k \ge 1$ Thus we have:

$$\boldsymbol{x}_k - \boldsymbol{x}_* = \boldsymbol{T}(\boldsymbol{y}_k - \boldsymbol{y}_*), \quad \forall\, k \ge 1.$$

We now continue using induction. We have:

$$\begin{aligned}
\boldsymbol{y}_1 - \boldsymbol{y}_* &= \boldsymbol{y}_0 - \boldsymbol{y}_* + (J_y(\boldsymbol{x}_0)^T J_y(\boldsymbol{x}_0))^{-1} J_y(\boldsymbol{x}_0)^T \left[-R(\boldsymbol{x}_0) + J_x(\boldsymbol{x}_0) L(\boldsymbol{x}_0, \boldsymbol{y}_0)\right] \\
&= (J_y(\boldsymbol{x}_0)^T J_y(\boldsymbol{x}_0))^{-1} \boldsymbol{M},
\end{aligned}$$

where $\boldsymbol{M} = -J_y(\boldsymbol{x}_0)^T R(\boldsymbol{x}_0) + J_y(\boldsymbol{x}_0)^T J_x(\boldsymbol{x}_0) L(\boldsymbol{x}_0, \boldsymbol{y}_0) - J_y(\boldsymbol{x}_0)^T J_y(\boldsymbol{x}_0)(\boldsymbol{y}_* - \boldsymbol{y}_0)$. Rewriting, we obtain

$$\begin{aligned}
\boldsymbol{M} &= -J_y(\boldsymbol{x}_0)^T R(\boldsymbol{x}_*) + J_y(\boldsymbol{x}_0)^T \left[R(\boldsymbol{x}_*) - R(\boldsymbol{x}_0) - J_y(\boldsymbol{x}_0)(\boldsymbol{y}_* - \boldsymbol{y}_0) + J_x(\boldsymbol{x}_0) L(\boldsymbol{x}_0, \boldsymbol{y}_0)\right] \\
&= -J_y(\boldsymbol{x}_0)^T R(\boldsymbol{x}_*) + \boldsymbol{T}^T J_x(\boldsymbol{x}_0)^T \left[R(\boldsymbol{x}_*) - R(\boldsymbol{x}_0) - J_x(\boldsymbol{x}_0)(\boldsymbol{T}\boldsymbol{y}_* - \boldsymbol{T}\boldsymbol{y}_0 - \boldsymbol{x}_0 + \hat{\boldsymbol{x}} + \boldsymbol{T}\boldsymbol{y}_0\right] \\
&= -\boldsymbol{T}^T (J_x(\boldsymbol{x}_0) - J_x(\boldsymbol{x}_*))^T R(\boldsymbol{x}_*) + \boldsymbol{T}^T J_x(\boldsymbol{x}_0)^T \left[R(\boldsymbol{x}_*) - R(\boldsymbol{x}_0) - J_x(\boldsymbol{x}_0)(\boldsymbol{x}_* - \boldsymbol{x}_0)\right].
\end{aligned}$$

According to Dennis and Schnabel [2, Theorem 3.1.4], there exists $\epsilon_1 > 0$ such that $J_y(\boldsymbol{x})^T J_y(\boldsymbol{x})$ is nonsingular and

$$\left|\left|(J_y(\boldsymbol{x})^T J_y(\boldsymbol{x}))^{-1}\right|\right| \leq \frac{c}{\lambda}, \quad \forall\, \boldsymbol{x} \in N(\boldsymbol{x}_*, \epsilon_1),$$

where $c \in (1, \lambda/\sigma)$. We also have:

$$\left|\left|\boldsymbol{T}^T (J_x(\boldsymbol{x}_0) - J_x(\boldsymbol{x}_*))^T R(\boldsymbol{x}_*)\right|\right| \leq ||\boldsymbol{T}|| \left|\left|(J_x(\boldsymbol{x}_0) - J_x(\boldsymbol{x}_*))^T R(\boldsymbol{x}_*)\right|\right| \leq \tau\sigma \left|\left|\boldsymbol{x}_0 - \boldsymbol{x}_*\right|\right|, \quad \forall \boldsymbol{x} \in D.$$

Applying the Lipschitz condition of $J_x(.)$, we obtain the following equation

$$||R(\boldsymbol{x}_*) - R(\boldsymbol{x}_0) - J_x(\boldsymbol{x}_0)(\boldsymbol{x}_* - \boldsymbol{x}_0)|| \leq \frac{\gamma}{2} \left|\left|\boldsymbol{x}_0 - \boldsymbol{x}_*\right|\right|^2.$$

Thus we have:

$$\left|\left|\boldsymbol{T}^T J_x(\boldsymbol{x}_0)^T \left[R(\boldsymbol{x}_*) - R(\boldsymbol{x}_0) - J_x(\boldsymbol{x}_0)(\boldsymbol{x}_* - \boldsymbol{x}_0)\right]\right|\right| \leq \frac{\tau\alpha\gamma}{2} \left|\left|\boldsymbol{x}_0 - \boldsymbol{x}_*\right|\right|^2.$$

Combining these inequalities gives the following bound

$$||\boldsymbol{y}_1 - \boldsymbol{y}_*|| \leq \frac{c}{\lambda} \left(\tau\sigma \left|\left|\boldsymbol{x}_0 - \boldsymbol{x}_*\right|\right| + \frac{\tau\alpha\gamma}{2} \left|\left|\boldsymbol{x}_0 - \boldsymbol{x}_*\right|\right|^2\right).$$

Or equivalently, we have:

$$||\boldsymbol{x}_1 - \boldsymbol{x}_*|| \leq \frac{\tau^2 c}{\lambda} \left(\sigma \left|\left|\boldsymbol{x}_0 - \boldsymbol{x}_*\right|\right| + \frac{\alpha\gamma}{2} \left|\left|\boldsymbol{x}_0 - \boldsymbol{x}_*\right|\right|^2\right), \quad \forall \boldsymbol{x}_0 \in N(\boldsymbol{x}_*, \epsilon_1).$$

Now let $\epsilon = \min\left\{\epsilon_1, \dfrac{\lambda - \tau^2 c\sigma}{\tau^2 c\alpha\gamma}\right\}$. We have:

$$\begin{aligned}
||\boldsymbol{x}_1 - \boldsymbol{x}_*|| &\leq ||\boldsymbol{x}_0 - \boldsymbol{x}_*|| \left(\frac{\tau^2 c\gamma}{\lambda} + \frac{\tau^2 c\alpha\gamma}{2\lambda} \left|\left|\boldsymbol{x}_0 - \boldsymbol{x}_*\right|\right|\right) \\
&\leq \frac{\lambda + \tau^2 c\sigma}{2\lambda} \left|\left|\boldsymbol{x}_0 - \boldsymbol{x}_*\right|\right| \\
&< \left|\left|\boldsymbol{x}_0 - \boldsymbol{x}_*\right|\right|, \quad \forall \boldsymbol{x}_0 \in N(\boldsymbol{x}_*, \epsilon).
\end{aligned}$$

From these results, clearly $\boldsymbol{x}_1 \in N(\boldsymbol{x}_*, \epsilon)$ and the induction step is exactly the same as in the case $k = 0$ above. □

Applying Theorem 1 to the Gauss-Newton equation (2.10), we have:

$$L(\boldsymbol{z}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{v}) = \begin{pmatrix} \boldsymbol{z} - \boldsymbol{c} + \boldsymbol{A}^T \boldsymbol{y} \\ \boldsymbol{x} - \hat{\boldsymbol{x}} - \boldsymbol{Q}\boldsymbol{v} \end{pmatrix},$$

where in the theorem $(\boldsymbol{z}, \boldsymbol{x})$ takes on the value $\boldsymbol{x}$ and $(\boldsymbol{y}, \boldsymbol{v})$ takes on the value $\boldsymbol{y}$, with $\boldsymbol{T} = \begin{pmatrix} -\boldsymbol{A}^T & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{Q} \end{pmatrix}$. We also have

$$R_\mu(\boldsymbol{z}, \boldsymbol{x}) = \mathrm{sMat}(\boldsymbol{z})\,\mathrm{sMat}(\boldsymbol{x}) - \mu \boldsymbol{I}.$$

The optimal solution $(\boldsymbol{z}_*, \boldsymbol{x}_*)$ satisfies the condition $R_\mu(\boldsymbol{z}_*, \boldsymbol{x}_*) = 0$. Therefore, we can set $\sigma = 0$. We have: $\tau = ||\boldsymbol{T}|| = \sqrt{||\boldsymbol{A}||^2 + ||\boldsymbol{Q}||^2}$. Now consider $R'(\boldsymbol{z}, \boldsymbol{x})$. We have:

$$R'(\boldsymbol{z}, \boldsymbol{x}) \begin{pmatrix} \Delta \boldsymbol{z} \\ \Delta \boldsymbol{x} \end{pmatrix} = \boldsymbol{Z}\,\mathrm{sMat}(\Delta \boldsymbol{x}) + \mathrm{sMat}(\Delta \boldsymbol{z})\boldsymbol{X},$$

where $\boldsymbol{Z} = \mathrm{sMat}(\boldsymbol{z})$ and $\boldsymbol{X} = \mathrm{sMat}(\boldsymbol{x})$. For all $(\Delta \boldsymbol{z}, \Delta \boldsymbol{x})$, we have:

$$\left\| R'(\boldsymbol{z}, \boldsymbol{x}) \begin{pmatrix} \Delta \boldsymbol{z} \\ \Delta \boldsymbol{x} \end{pmatrix} \right\| \le ||\boldsymbol{Z}||\,||\Delta \boldsymbol{x}|| + ||\Delta \boldsymbol{z}||\,||\boldsymbol{X}|| \le \sqrt{||\boldsymbol{Z}||^2 + ||\boldsymbol{X}||^2}\sqrt{||\Delta \boldsymbol{z}||^2 + ||\Delta \boldsymbol{x}||^2}.$$

Thus, $||R'(\boldsymbol{z}, \boldsymbol{x})|| \le \sqrt{||\boldsymbol{Z}||^2 + ||\boldsymbol{X}||^2} = ||(\boldsymbol{z}, \boldsymbol{x})||$. Thus the parameter α can be calculated using the formulation $\alpha = \sup_{(\boldsymbol{z}, \boldsymbol{x}) \in D} ||(\boldsymbol{z}, \boldsymbol{x})||$. Similarly, we have:

$$\left[R'(\boldsymbol{z}_1, \boldsymbol{x}_1) - R'(\boldsymbol{z}_2, \boldsymbol{x}_2)\right] \begin{pmatrix} \Delta \boldsymbol{z} \\ \Delta \boldsymbol{x} \end{pmatrix} = (\boldsymbol{Z}_1 - \boldsymbol{Z}_2)\,\mathrm{sMat}(\Delta \boldsymbol{x}) + \mathrm{sMat}(\Delta \boldsymbol{z})(\boldsymbol{X}_1 - \boldsymbol{X}_2).$$

Thus for all $(\Delta \boldsymbol{z}, \Delta \boldsymbol{x})$,

$$\left\| \left[R'(\boldsymbol{z}_1, \boldsymbol{x}_1) - R'(\boldsymbol{z}_2, \boldsymbol{x}_2)\right] \begin{pmatrix} \Delta \boldsymbol{z} \\ \Delta \boldsymbol{x} \end{pmatrix} \right\| \le ||(\boldsymbol{z}_1, \boldsymbol{x}_1) - (\boldsymbol{z}_1, \boldsymbol{x}_1)||\,||(\Delta \boldsymbol{z}, \Delta \boldsymbol{x})||,$$

which implies that $||R'(\boldsymbol{z}_1, \boldsymbol{x}_1) - R'(\boldsymbol{z}_2, \boldsymbol{x}_2)|| \le ||(\boldsymbol{z}_1, \boldsymbol{x}_1) - (\boldsymbol{z}_1, \boldsymbol{x}_1)||$. We can set the parameter γ to be 1. The remaining parameter is the smallest eigenvalue λ of $J_y(\boldsymbol{z}_*, \boldsymbol{x}_*)^T J_y(\boldsymbol{z}_*, \boldsymbol{x}_*)$. Kruk et al. [7] show that under the strict complementary slackness condition of a unique optimal solution, the Jacobian J at the optimal solution is full rank ($\lambda > 0$). This implies that the algorithm is well-behaved under the strict complementary slackness condition. Theorem 1 indicates that there is a neighborhood around the optimal solution that the Gauss-Newton method will converges quadratically. However, it is difficult to estimate the value of λ. We need all of these parameters to define the neighborhood $N(\boldsymbol{x}_*, \epsilon)$ in which one can simply use affine scaling and the Gauss-Newton method instead of *damped* Gauss-Newton method, i.e. we can set the barrier parameter $\mu = 0$ and take full step lengths equal to one. A heuristic rule could be built based on the infeasibility of the current solution $(\boldsymbol{z}, \boldsymbol{x})$ since the damped Gauss-Newton method with step sizes different from 1 (with infeasible initial solutions) will not guarantee the feasibility after one step (see [15] for more details on other possible heuristic rules).

2.5 Presolve and Preconditioning

The purpose of a presolve is to construct well-conditioned and sparse bases for both the range and nullspace of $\boldsymbol{A}$. These bases play an important role in the matrix-vector multiplications involved in the algorithm. If $\boldsymbol{A}$ is sparse and $\boldsymbol{A} = [\boldsymbol{S} \mid \boldsymbol{E}]$, where $\boldsymbol{S} \in \mathbb{R}^{m\times m}$ is well-conditioned and $\boldsymbol{E} \in \mathbb{R}^{m\times(t(n)-m)}$, then we can use the columns of $\boldsymbol{Q} = \begin{bmatrix} -\boldsymbol{S}^{-1}\boldsymbol{E} \\ \boldsymbol{I} \end{bmatrix}$ for the basis of the nullspace of $\boldsymbol{A}$. The matrix inversion is fast and accurate if $\boldsymbol{S}$ is a well-conditioned triangular matrix. In addition, we can also consider row and column permutations on $\boldsymbol{A}$; therefore, the main task in the presolve is to find an $m \times m$ submatrix of $\boldsymbol{A}$ (up to a permutation of rows and columns) that is well-conditioned and (approximately) triangular. In this case, $\boldsymbol{S}$ and $\boldsymbol{E}$ are considered to be the *main inputs* to the algorithm.

If $\boldsymbol{A}$ is not sparse, it is better to keep $\boldsymbol{A}$ and $\boldsymbol{P} = -\boldsymbol{S}^{-1}\boldsymbol{E}$ as the main inputs (with $\boldsymbol{Q} = [\boldsymbol{P}; \boldsymbol{I}]$). And the major task of the presolve is to find a well-conditioned submatrix $\boldsymbol{S}$. This can be done by randomly picking m columns from $\boldsymbol{A}$ and reforming the submatrix $\boldsymbol{S}$ until the condition number satisfies some given conditions.

The most expensive part of the algorithm is solving the overdetermined system (2.4), or equivalently, (2.11). Preconditioning is essential to reduce the computational time when using iterative methods for this part. Mathematically, we want to find a nonsingular transformation $T : \mathbb{R}^{t(n)} \to \mathbb{R}^{t(n)}$ such that $J \circ T^{-1}$ is well-conditioned. Instead of solving the system (2.4) or (2.11), we would like to solve the systems

$$\begin{cases} J \circ T^{-1}(\Delta \boldsymbol{q}) = \boldsymbol{R} \\ T(\Delta \boldsymbol{v}, \Delta \boldsymbol{y}) = \Delta \boldsymbol{q}, \end{cases} \tag{2.13}$$

where $\boldsymbol{R}$ is the corresponding right-hand side.

Let $T^{-1} = S = \begin{bmatrix} S_v \\ S_y \end{bmatrix}$ where $S_v : \mathbb{R}^{t(n)} \to \mathbb{R}^{t(n)-m}$ and $S_y : \mathbb{R}^{t(n)} \to \mathbb{R}^m$. We then have:

$$J \circ T^{-1}(\boldsymbol{q}) = J(S_v(\boldsymbol{q}), S_y(\boldsymbol{q})) = \mathcal{Z}(S_v(\boldsymbol{q})) + \mathcal{X}(S_y(\boldsymbol{q})) = \mathcal{Z} \circ S_v(\boldsymbol{q}) + \mathcal{X} \circ S_y(\boldsymbol{q}). \tag{2.14}$$

To find the adjoint $(J \circ T^{-1})^*$, we note that $(T^{-1})^* = S^* = [S_v^* \mid S_y^*]$ implies:

$$(J \circ T^{-1})^*(\boldsymbol{M}) = (T^{-1})^* \circ J^*(\boldsymbol{M}) = (T^{-1})^* \left(\begin{bmatrix} \mathcal{Z}^*(\boldsymbol{M}) \\ \mathcal{X}^*(\boldsymbol{M}) \end{bmatrix} \right) = S_v^* \circ \mathcal{Z}^*(\boldsymbol{M}) + S_y^* \circ \mathcal{X}^*(\boldsymbol{M}), \tag{2.15}$$

where $\mathcal{X}^*$ and $\mathcal{Z}^*$ are formulated in (2.6) and (2.7), respectively.

Consider the special case in which T is a separable transformation with respect to $\Delta \boldsymbol{v}$ and $\Delta \boldsymbol{y}$: $T(\Delta \boldsymbol{v}, \Delta \boldsymbol{y}) = \begin{bmatrix} T_v(\Delta \boldsymbol{v}) \\ T_y(\Delta \boldsymbol{y}) \end{bmatrix}$, where $T_v : \mathbb{R}^{t(n)-m} \to \mathbb{R}^{t(n)-m}$ and $T_y : \mathbb{R}^m \to \mathbb{R}^m$ are both nonsingular. If $\boldsymbol{q} = (\boldsymbol{w}, \boldsymbol{u})$ with $\boldsymbol{w} \in \mathbb{R}^{t(n)-m}$ and $\boldsymbol{u} \in \mathbb{R}^m$, we then have:

$$J \circ T^{-1}(\boldsymbol{w}, \boldsymbol{u}) = \mathcal{Z} \circ T_v^{-1}(\boldsymbol{w}) + \mathcal{X} \circ T_y^{-1}(\boldsymbol{u}). \tag{2.16}$$

And the adjoint is written as follows:

$$(J \circ T^{-1})^*(\boldsymbol{M}) = \begin{bmatrix} (T_v^{-1})^* \circ \mathcal{Z}^*(\boldsymbol{M}) \\ (T_y^{-1})^* \circ \mathcal{X}^*(\boldsymbol{M}) \end{bmatrix}. \tag{2.17}$$

The first preconditioner we would like to consider is a simple diagonal scaling transformation $\mathcal{S}$. Since the original system of linear equations we need to solve is $J^* \circ J(\Delta \boldsymbol{v}, \Delta \boldsymbol{y}) = J^*(\boldsymbol{R})$, we would want to have $\mathcal{S}^* \circ \mathcal{S}$ approximate $J^* \circ J$. We define the diagonal preconditioner $\mathcal{S}$ as follows:

$$\mathcal{S}(\boldsymbol{e}_i) = ||J(\boldsymbol{e}_i)||_F \, \boldsymbol{e}_i, \quad \forall \, i = 1, \ldots, t(n), \tag{2.18}$$

where $\boldsymbol{e}_i$ is the i-th unit vector in $\mathbb{R}^{t(n)}$. Alternatively, suppose that the current iterations $Z, X \in \mathcal{S}^n$ are fixed. From (2.9), we see that the diagonal elements of the top left and bottom right blocks of $J^* \circ J$ are obtained using the rows of Q^T, A, respectively. Therefore, we can scale by diagonal matrices D_Q, D_A and obtain $D_Q Q, D_A A^T$ so that $\begin{bmatrix} D_Q^2(\boldsymbol{Q}^{(2)})^T & 0 \\ 0 & D_A^2(\boldsymbol{A}^{(2)}) \end{bmatrix} \begin{pmatrix} \operatorname{svec}(Z^2) \\ \operatorname{svec}(X^2) \end{pmatrix} = e$, the vector of all ones. This means we can scale Q^T, A, find the search direction, and *unscale* to return to the original problem. This avoids the preconditioning during the matrix-free phase. The matrices of squared matrices $\boldsymbol{A}^{(2)}, \boldsymbol{Q}^{(2)}$ need to be calculated only once, at the start of the algorithm. We see that

$$D_Q^{-2} = \operatorname{Diag}((\boldsymbol{Q}^{(2)})^T \operatorname{svec}(Z^2)), \quad D_A^{-2} = \operatorname{Diag}(\boldsymbol{A}^{(2)} \operatorname{svec}(X^2)).$$

The second preconditioner is the separable transformation (block diagonal) $\mathcal{Q}$ such that $\mathcal{Q}_v^* \circ \mathcal{Q}_v$ approximates $\mathcal{Z}^* \circ \mathcal{Z}$ and $\mathcal{Q}_y \circ \mathcal{Q}_y$ approximates $\mathcal{X}^* \circ \mathcal{X}$. Using the QR decomposition, we can decompose $\mathcal{Z}$ into $\mathcal{Q}_Z \circ \mathcal{R}_Z$, where $\mathcal{Q}_Z : \mathbb{R}^{t(n)} \to \mathcal{M}^n$ is a unitary transformation, and $\mathcal{R}_Z : \mathbb{R}^{t(n)-m} \to \mathbb{R}^{t(n)}$, with an upper triangular transformation matrix $\boldsymbol{R}_Z$. Clearly, we can set $\mathcal{Q}_v = \mathcal{P}_v \circ \mathcal{R}_Z$, where $\mathcal{P}_v$ is the projection transformation from $\mathbb{R}^{n^2}$ to $\mathbb{R}^{t(n)} - m$, since $\mathcal{R}_Z^* \circ \mathcal{R}_Z = \mathcal{Z}^* \circ \mathcal{Z}$. Similarly, we can set $\mathcal{Q}_y = \mathcal{P}_y \circ \mathcal{R}_X$ where $\mathcal{X} = \mathcal{Q}_X \circ \mathcal{R}_X$ under the QR decomposition and $\mathcal{P}_y$ is the projection transformation from $\mathbb{R}^{n^2}$ to $\mathbb{R}^m$.

We now would like to consider the properties of these two particular preconditioners. Consider the following measure which depends uniformly on all the eigenvalues of a positive definite matrix $\boldsymbol{X}$,

$$\omega(\boldsymbol{X}) := \frac{\operatorname{trace}(\boldsymbol{X})/n}{\det(\boldsymbol{X})^{\frac{1}{n}}}.$$

This measure is a pseudoconvex function. Note that a function is pseudoconvex if

$$(\boldsymbol{y} - \boldsymbol{x})^T \nabla f(\boldsymbol{x}) \geq 0 \Longrightarrow f(\boldsymbol{y}) \geq f(\boldsymbol{x}),$$

and for pseudoconvex functions, all stationary solutions are global minimizers (see for example, [11]). We now show that we have obtained the *optimal block diagonal preconditioner* with respect to the measure ω, thus extending the corresponding diagonal preconditioner result in [3].

Proposition 1. *The measure $\omega(\boldsymbol{A})$ satisfies*

1. $1 \leq \omega(\boldsymbol{A}) \leq \kappa(\boldsymbol{A}) < \dfrac{(\kappa(\boldsymbol{A}) + 1)^2}{\kappa(\boldsymbol{A})} \leq 4\omega^n(\boldsymbol{A})$, *where* $\kappa(\boldsymbol{A}) = ||\boldsymbol{A}|| \, \left|\left|\boldsymbol{A}^{-1}\right|\right|$, *with equality in the first and second inequality if and only if $\boldsymbol{A}$ is a multiple of the identity and equality in the last inequality if and only if*

$$\lambda_2 = \cdots = \lambda_{n-1} = \frac{\lambda_1 + \lambda_n}{2};$$

2. $\omega(\alpha \boldsymbol{A}) = \omega(\boldsymbol{A})$, *for all* $\alpha > 0$*;*

3. *if $n = 2$, $\omega(\boldsymbol{A})$ is isotonic with $\kappa(\boldsymbol{A})$.*

4. *The measure ω is pseudoconvex on the set of s.p.d. matrices, and thus any stationary point is a global minimizer of ω.*

5. *Let $\boldsymbol{V}$ be a full rank $m \times n$ matrix, $n \leq m$. Then the optimal column scaling that minimizes the measure ω, i.e.*
$$\min\ \omega((\boldsymbol{V}\boldsymbol{D})^T(\boldsymbol{V}\boldsymbol{D})),$$
over $\boldsymbol{D}$ positive, diagonal, is given by
$$D_{ii} = \frac{1}{\|\boldsymbol{V}_{:,i}\|}, i = 1, ..., n,$$
where $\boldsymbol{V}_{:,i}$ is the i-th column of $\boldsymbol{V}$.

6. *Let $\boldsymbol{V}$ be a full rank $m \times n$ matrix, $n \leq m$ with block structure $\boldsymbol{V} = \begin{bmatrix}\boldsymbol{V}_1 & \boldsymbol{V}_2 & \dots & \boldsymbol{V}_k\end{bmatrix}$. Then the optimal corresponding block diagonal scaling*
$$\boldsymbol{D} = \begin{bmatrix} \boldsymbol{D}_1 & 0 & 0 & \dots & 0 \\ 0 & \boldsymbol{D}_2 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \boldsymbol{D}_k \end{bmatrix}$$
that minimizes the measure ω, i.e.
$$\min\ \omega((\boldsymbol{V}\boldsymbol{D})^T(\boldsymbol{V}\boldsymbol{D})),$$
over $\boldsymbol{D}$ block diagonal, is given by
$$\boldsymbol{D}_i = \{\boldsymbol{V}_i^T\boldsymbol{V}_i\}^{-\frac{1}{2}}, i = 1, ..., k.$$

Proof. Items 1 to 5 are proved in [3].

To prove 6, let the blocked $\boldsymbol{V}$ be given. Then the arithmetic-geometric mean inequality yields
$$\begin{aligned} \omega((\boldsymbol{V}\boldsymbol{D})^T(\boldsymbol{V}\boldsymbol{D})) &= \frac{\operatorname{trace}(\boldsymbol{D}^T\boldsymbol{V}^T\boldsymbol{V}\boldsymbol{D})/n}{\det(\boldsymbol{D}^T\boldsymbol{V}^T\boldsymbol{V}\boldsymbol{D})^{\frac{1}{n}}} \\ &= \frac{\operatorname{trace}(\boldsymbol{V}^T\boldsymbol{V}\boldsymbol{D}\boldsymbol{D}^T/n}{\det(\boldsymbol{D}\boldsymbol{D}^T)^{\frac{1}{n}}\det(\boldsymbol{V}^T\boldsymbol{V})^{\frac{1}{n}}}. \end{aligned}$$

Let $\bar{\boldsymbol{D}} = \boldsymbol{D}\boldsymbol{D}^T$, we have, $\bar{\boldsymbol{D}}$ is also a block diagonal matrix with $\bar{\boldsymbol{D}}_i = \boldsymbol{D}_i\boldsymbol{D}_i^T$ for all $i = 1, \dots, k$. Thus
$$\begin{aligned} \omega((\boldsymbol{V}\boldsymbol{D})^T(\boldsymbol{V}\boldsymbol{D})) &= \frac{\sum_{i=1}^k(\boldsymbol{V}_i^T\boldsymbol{V}_i)\cdot\bar{\boldsymbol{D}}_i/n}{\det(\boldsymbol{V}^T\boldsymbol{V})^{\frac{1}{n}}\prod_{i=1}^k\det(\bar{\boldsymbol{D}}_i)^{\frac{1}{n}}} \\ &= \left(\frac{1}{n\det(\boldsymbol{V}^T\boldsymbol{V})^{\frac{1}{n}}}\right)\frac{\sum_{i=1}^k\bar{\boldsymbol{V}}_i\cdot\bar{\boldsymbol{D}}_i}{\prod_{i=1}^k\det(\bar{\boldsymbol{D}}_i)^{\frac{1}{n}}}, \end{aligned}$$

where $\bar{\boldsymbol{V}}_i = \boldsymbol{V}_i^T \boldsymbol{V}_i$ for all $i = 1, \ldots, k$. Consider the function $f(\bar{\boldsymbol{D}}_1, \ldots, \bar{\boldsymbol{D}}_k) = \dfrac{\sum_{i=1}^k \bar{\boldsymbol{V}}_i \cdot \bar{\boldsymbol{D}}_i}{\prod_{i=1}^k \det(\bar{\boldsymbol{D}}_i)^{\frac{1}{n}}}$, we have:

$$\frac{\partial f(\bar{\boldsymbol{D}}_1, \ldots, \bar{\boldsymbol{D}}_k)}{\partial \bar{\boldsymbol{D}}_i} = \prod_{j=1}^k \det(\bar{\boldsymbol{D}}_j)^{-\frac{1}{n}} \left(\bar{\boldsymbol{V}}_i - \frac{\sum_{j=1}^k \bar{\boldsymbol{V}}_j \cdot \bar{\boldsymbol{D}}_j}{n} (\bar{\boldsymbol{D}}_i^{-1})^T \right).$$

Let $\bar{\boldsymbol{D}}_i = \bar{\boldsymbol{V}}_i^{-1}$ for all $i = 1, \ldots, k$, we have: $\bar{\boldsymbol{D}}_i$ is symmetric and $\displaystyle\sum_{i=1}^k \bar{\boldsymbol{V}}_i \cdot \bar{\boldsymbol{D}}_i = n$ for all $i = 1, \ldots, k$. Thus $\dfrac{f(\bar{\boldsymbol{D}}_1, \ldots, \bar{\boldsymbol{D}}_k)}{\partial \bar{\boldsymbol{D}}_i} = \boldsymbol{0}$ for all $i = 1, \ldots, k$. The measure ω is pseudoconvex and similarly, the function $f(\bar{\boldsymbol{D}}_1, \ldots, \bar{\boldsymbol{D}}_k)$ is also pseudoconvex (quotient of a convex (linear) function and a positive concave function). Thus the matrices $(\bar{\boldsymbol{D}}_1, \ldots, \bar{\boldsymbol{D}}_k)$ with $\bar{\boldsymbol{D}}_i = \bar{\boldsymbol{V}}_i^{-1}$ for $i = 1, \ldots, k$, minimizes $f(\bar{\boldsymbol{D}}_1, \ldots, \bar{\boldsymbol{D}}_k)$. This implies that the symmetric block matrix $\boldsymbol{D}$ with $\boldsymbol{D}_i = (\boldsymbol{V}_i^T \boldsymbol{V}_i)^{-\frac{1}{2}}$ minimizes the measure $\omega((\boldsymbol{V}\boldsymbol{D})^T(\boldsymbol{V}\boldsymbol{D}))$. □

Note that the optimality condition only requires $\bar{\boldsymbol{D}}_i = \bar{\boldsymbol{V}}_i^{-1}$ for $i = 1, \ldots, k$, which means we can use QR decomposition to find the optimal $\boldsymbol{D}_i$ instead of matrix square root. In other words, if $\boldsymbol{V}_i$ has the QR decomposition $\boldsymbol{V}_i = \boldsymbol{Q}_i \boldsymbol{R}_i$, we can set $\boldsymbol{D}_i = \boldsymbol{R}_i^{-1}$ for all $i = 1, \ldots, k$ to minimize the measure $\omega((\boldsymbol{V}\boldsymbol{D})^T(\boldsymbol{V}\boldsymbol{D}))$.

3 Lovász Theta Function Problem

In this section, we apply the Gauss-Newton method to the Lovász theta function problem. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph; and let $n = |\mathcal{V}|$ and $m = |\mathcal{E}|$ be the number of nodes and edges, respectively. The Lovász theta number (defined in [9]) is the optimal value of the following SDP

$$\text{(TN)} \qquad \begin{array}{rll} \vartheta(\mathcal{G}) := p^* := & \max & \boldsymbol{E} \cdot \boldsymbol{X} \\ & \text{s.t.} & \boldsymbol{I} \cdot \boldsymbol{X} = 1 \\ & & \boldsymbol{E}_{ij} \cdot \boldsymbol{X} = 0, \quad \forall\,(i,j) \in \mathcal{E} \\ & & \boldsymbol{X} \succeq 0 \end{array} \tag{3.1}$$

The dual SDP is

$$\text{(DTN)} \qquad \begin{array}{rll} d^* := & \min & z \\ & \text{s.t.} & z\boldsymbol{I} + \displaystyle\sum_{(i,j)\in\mathcal{E}} y_{ij} \boldsymbol{E}_{ij} - \boldsymbol{Z} = \boldsymbol{E}, \\ & & \boldsymbol{Z} \succeq 0, \end{array} \tag{3.2}$$

where $\boldsymbol{y} = (y_{ij})_{(i,j)\in\mathcal{E}} \in \mathbb{R}^m$; $\boldsymbol{I}$ is the identity matrix; $\boldsymbol{E}$ is the matrix of all ones, $\boldsymbol{E}_{ij} = (e_i e_j^T + e_j e_i^T)/\sqrt{2}$ is the ij-th unit matrix in $\mathcal{S}^n$, and e_i is the i-th unit vector.

The theta number has important properties, e.g., it is tractable (can be computed in polynomial time) and it provides bounds for the stability and chromatic numbers of the graph, see e.g., [6, 8]. We now show how to use the Gauss-Newton approach to solve this Lovász theta function problem.

3.1 Matrix-Free Formulation

We define additional linear transformations on vectors and matrices for this problem. Let $\mathcal{G}^c = (\mathcal{V}, \mathcal{E}^c)$ be the complement graph of $\mathcal{G}$, i.e. $\mathcal{E}^c$ is the edge set complement to $\mathcal{E}$. Let $m_c := |\mathcal{E}^c| =$

$t(n) - m$, where $t(n) := \binom{n}{2}$, and consider two bijective index mappings, $\mathrm{ind}_{\mathcal{E}} : \mathcal{E} \to \{1, \ldots, m\}$, and $\mathrm{ind}_{\mathcal{E}^c} : \mathcal{E}^c \to \{1, \ldots, m_c\}$ with their corresponding inverses, $\mathrm{ind}_{\mathcal{E}}^{-1}$ and $\mathrm{ind}_{\mathcal{E}^c}^{-1}$.

We now define two linear transformations, $\mathrm{sMat}_{\mathcal{E}} : \mathbb{R}^m \to \mathcal{S}^n$ and $\mathrm{sMat}_{\mathcal{E}^c} : \mathbb{R}^{m_c} \to \mathcal{S}^n$ as follows:

$$\mathrm{sMat}_{\mathcal{E}}(\boldsymbol{y}) = \sum_{i=1}^{m} y_i \boldsymbol{E}_{\mathrm{ind}_{\mathcal{E}}^{-1}(i)}, \quad \mathrm{sMat}_{\mathcal{E}^c}(\boldsymbol{v}) = \sum_{i=1}^{m_c} v_i \boldsymbol{E}_{\mathrm{ind}_{\mathcal{E}^c}^{-1}(i)}.$$

Let $\mathrm{svec}_{\mathcal{E}} := \mathrm{sMat}_{\mathcal{E}}^{\dagger}$ denote the *Moore-Penrose generalized inverse* mapping from $\mathcal{S}^n$ to $\mathbb{R}^m$, $\mathrm{svec}_{\mathcal{E}}(\boldsymbol{S}) = \boldsymbol{y}$, where $y_i = S_{\mathrm{ind}_{\mathcal{E}}^{-1}(i)}\sqrt{2}$ for all $i = 1, \ldots, m$. Similarly, we can also define $\mathrm{svec}_{\mathcal{E}^c} := \mathrm{sMat}_{\mathcal{E}^c}^{\dagger}$. The mapping $\mathrm{svec}_{\mathcal{E}^c}$ (resp. $\mathrm{svec}_{\mathcal{E}}$) is an inverse mapping if we restrict to the subspace of symmetric matrices with zero in positions corresponding to the edge set $\mathcal{E}^c$ (resp. $\mathcal{E}$). The adjoint operator $\mathrm{sMat}_{\mathcal{E}^c}^{*} = \mathrm{svec}_{\mathcal{E}^c}$, since

$$\begin{aligned} \langle \mathrm{sMat}_{\mathcal{E}^c}(\boldsymbol{v}), \boldsymbol{S} \rangle &= \mathrm{trace}(\mathrm{sMat}_{\mathcal{E}^c}(\boldsymbol{v})\boldsymbol{S}) \\ &= \boldsymbol{v}^T \mathrm{svec}_{\mathcal{E}^c}(\boldsymbol{S}) = \langle \boldsymbol{v}, \mathrm{svec}_{\mathcal{E}^c}(\boldsymbol{S}) \rangle, \end{aligned}$$

and similarly, the adjoint operator $\mathrm{sMat}_{\mathcal{E}}^{*} = \mathrm{svec}_{\mathcal{E}}$.

Using these transformations, we can represent primal and dual feasibility as follows. Let the matrix $\boldsymbol{V} := \begin{pmatrix} -\boldsymbol{e}^T \\ \boldsymbol{I} \end{pmatrix} \in \mathbb{R}^{n \times (n-1)}$, we have: $\boldsymbol{X} \succeq 0$ is primal feasible if and only there exists $\boldsymbol{w} \in \mathbb{R}^{n-1}$ and $\boldsymbol{v} \in \mathbb{R}^{m_c}$ such that

$$\boldsymbol{X} = \frac{1}{n}\boldsymbol{I} + \mathrm{Diag}(\boldsymbol{V}\boldsymbol{w}) + \mathrm{sMat}_{\mathcal{E}^c}(\boldsymbol{v}),$$

where $\mathrm{Diag} : \mathbb{R}^n \to \mathcal{M}^n$ is the diagonal transformation with its diagonal adjoint $\mathrm{Diag}^* = \mathrm{Diag}^{\dagger} = \mathrm{diag} : \mathcal{M}^n \to \mathbb{R}^n$. We could also use a different matrix $\boldsymbol{V}$ to maintain the isometry,

$$\boldsymbol{V} := \begin{pmatrix} c_1 \boldsymbol{e}^T \\ c_2 \boldsymbol{E} + \boldsymbol{I} \end{pmatrix} \in \mathbb{R}^{n \times (n-1)},$$

where $c_1 = -1/\sqrt{n}$ and $c_2 = -1/(n + \sqrt{n})$.

Now let us consider the dual feasibility. Similarly, $\boldsymbol{Z} \succeq 0$ is dual feasible if and only if there exists $z \in \mathbb{R}$ and $\boldsymbol{y} \in \mathbb{R}^m$ such that

$$\boldsymbol{Z} = -\boldsymbol{E} + z\boldsymbol{I} + \mathrm{sMat}_{\mathcal{E}}(\boldsymbol{y}).$$

As in the general case, we would like to solve the perturbed optimality condition $\boldsymbol{Z}\boldsymbol{X} - \mu \boldsymbol{I} = \boldsymbol{0}$ in each iteration of the primal-dual path following interior algorithm. We again use an inexact Gauss-Newton approach with a matrix-free preconditioned conjugate gradient method.

The major operation of the Gauss-Newton approach is to calculate the Jacobian and its adjoint. The Jacobian $J := G'_{\mu}$ can be written as $J = [\mathcal{Z} \mid \mathcal{X}]$, where $\mathcal{Z}$ and $\mathcal{X}$ are two transformations defined as follows:

$$\mathcal{Z}(\boldsymbol{w}, \boldsymbol{v}) = \boldsymbol{Z}\left(\mathrm{Diag}(\boldsymbol{V}\boldsymbol{w}) + \mathrm{sMat}_{\mathcal{E}^c}(\boldsymbol{v})\right), \quad \mathcal{X}(z, \boldsymbol{y}) = \left(z\boldsymbol{I} + \mathrm{sMat}_{\mathcal{E}}(\boldsymbol{y})\right)\boldsymbol{X}.$$

In order to find the adjoint J^*, we need to compute $\mathcal{Z}^*$ and $\mathcal{X}^*$. Consider $\boldsymbol{M} \in \mathcal{M}^n$, we have:

$$\begin{aligned}
\langle \boldsymbol{M}, \mathcal{X}(z, \boldsymbol{y}) \rangle &= z \operatorname{trace}(\boldsymbol{M}^T \boldsymbol{X}) + \operatorname{trace}(\boldsymbol{M}^T \operatorname{sMat}_{\mathcal{E}}(\boldsymbol{y})\boldsymbol{X}) \\
&= z \operatorname{trace}(\boldsymbol{M}^T \boldsymbol{X}) + \operatorname{trace}\left(\operatorname{sMat}_{\mathcal{E}}(\boldsymbol{y})\frac{1}{2}\left(\boldsymbol{X}\boldsymbol{M}^T + \boldsymbol{M}\boldsymbol{X}\right)\right) \\
&= z \operatorname{trace}(\boldsymbol{M}^T \boldsymbol{X}) + \frac{1}{2}\boldsymbol{y}^T \operatorname{svec}_{\mathcal{E}}(\boldsymbol{X}\boldsymbol{M}^T + \boldsymbol{M}\boldsymbol{X}) \\
&= z \operatorname{trace}(\boldsymbol{M}^T \boldsymbol{X}) + \langle \boldsymbol{y}, \frac{1}{2} \operatorname{svec}_{\mathcal{E}}(\boldsymbol{X}\boldsymbol{M}^T + \boldsymbol{M}\boldsymbol{X})\rangle.
\end{aligned}$$

Thus we have:

$$\mathcal{X}^*(\boldsymbol{M}) = \begin{pmatrix} \operatorname{trace}(\boldsymbol{M}^T\boldsymbol{X}) \\ \frac{1}{2}\operatorname{svec}_{\mathcal{E}}(\boldsymbol{X}\boldsymbol{M}^T + \boldsymbol{M}\boldsymbol{X}) \end{pmatrix}. \tag{3.3}$$

Similarly, we have:

$$\begin{aligned}
\langle \boldsymbol{M}, \mathcal{Z}(\boldsymbol{w}, \boldsymbol{v}) \rangle &= \operatorname{trace}(\boldsymbol{M}^T \boldsymbol{Z}\,(\operatorname{Diag}(\boldsymbol{V}\boldsymbol{w}) + \operatorname{sMat}_{\mathcal{E}^c}(\boldsymbol{v})) \\
&= \operatorname{trace}(\boldsymbol{M}^T \boldsymbol{Z}\,(\operatorname{Diag}(\boldsymbol{V}\boldsymbol{w})) + \operatorname{trace}\left(\operatorname{sMat}_{\mathcal{E}^c}(\boldsymbol{v})\frac{1}{2}(\boldsymbol{M}^T\boldsymbol{Z} + \boldsymbol{Z}\boldsymbol{M})\right) \\
&= \langle \operatorname{diag}(\boldsymbol{M}^T\boldsymbol{Z}), \boldsymbol{V}\boldsymbol{w}\rangle + \frac{1}{2}\boldsymbol{v}^T \operatorname{svec}_{\mathcal{E}^c}(\boldsymbol{M}^T\boldsymbol{Z} + \boldsymbol{Z}\boldsymbol{M}) \\
&= \langle \boldsymbol{w}, \boldsymbol{V}^T \operatorname{diag}(\boldsymbol{M}^T\boldsymbol{Z})\rangle + \langle \boldsymbol{v}, \frac{1}{2}\operatorname{svec}_{\mathcal{E}^c}(\boldsymbol{M}^T\boldsymbol{Z} + \boldsymbol{Z}\boldsymbol{M})\rangle,
\end{aligned}$$

Thus the adjoint $\mathcal{Z}^*$ is

$$\mathcal{Z}^*(\boldsymbol{M}) = \begin{pmatrix} \boldsymbol{V}^T \operatorname{diag}(\boldsymbol{M}^T\boldsymbol{Z}) \\ \frac{1}{2}\operatorname{svec}_{\mathcal{E}^c}(\boldsymbol{M}^T\boldsymbol{Z} + \boldsymbol{Z}\boldsymbol{M}) \end{pmatrix}. \tag{3.4}$$

3.2 Presolve and Preconditioning

In this Lovász theta number problem, the bases are defined by the index mappings, $\operatorname{ind}_{\mathcal{E}}$ and $\operatorname{ind}_{\mathcal{E}^c}$. There are many ways to index the edge set $\mathcal{E}$ and $\mathcal{E}^c$. However, it is advantageous to index the edge sets according to a *good graph partitioning*. In addition, we can also change the order of nodes since the row permutation of the matrix $\boldsymbol{V}$ affects the bases. All of these issues will become clear when we discuss the diagonal and block preconditioners in the following sections.

3.2.1 Diagonal Preconditioning

In order to construct the diagonal preconditioner, we need to calculate $||J(\boldsymbol{e}_i)||_F$ for all unit vector $\boldsymbol{e}_i$ in $\mathbb{R}^{t(n)}$, $i = 1, \ldots, t(n)$. Since the variables are $(\boldsymbol{w}, \boldsymbol{v}, z, \boldsymbol{y})$, we consider four different cases, each of which corresponds to each variable.

(i) $1 \le i \le n-1$: $J(\boldsymbol{e}_i) = \boldsymbol{Z}\operatorname{Diag}(\boldsymbol{V}\boldsymbol{e}_i^w)$, where $\boldsymbol{e}_i^w$ is the i-th unit vector in $\mathbb{R}^{n-1}$. We have: $\boldsymbol{V}\boldsymbol{e}_i^w = [-1; \boldsymbol{e}_i^w]$. Thus

$$||J(\boldsymbol{e}_i)||_F = \sqrt{||\boldsymbol{Z}_{:,1}||^2 + ||\boldsymbol{Z}_{:,i+1}||^2}, \quad \forall\, i = 1, \ldots, n-1.$$

(ii) $i = n - 1 + j$, where $1 \le j \le m_c$: $J(\boldsymbol{e}_i) = \boldsymbol{Z}\,\mathrm{sMat}_{\mathcal{E}^c}(\boldsymbol{e}_j^v)$, where $\boldsymbol{e}_j^v$ is the j-th unit vector in $\mathbb{R}^{m_c}$. We have: $\mathrm{sMat}_{\mathcal{E}^c}(\boldsymbol{e}_j^v) = \boldsymbol{E}_{\mathrm{ind}_{\mathcal{E}^c}^{-1}(j)}$. Let $\mathrm{ind}_{\mathcal{E}^c}^{-1}(j) = (k_{\mathcal{E}^c}(j), l_{\mathcal{E}^c}(j))$, we have:

$$||J(\boldsymbol{e}_i)||_F = \frac{1}{\sqrt{2}}\sqrt{\left|\left|\boldsymbol{Z}_{:,k_{\mathcal{E}^c}(j)}\right|\right|^2 + \left|\left|\boldsymbol{Z}_{:,l_{\mathcal{E}^c}(j)}\right|\right|^2}, \quad \forall\, i = n-1+j,\; j = 1,\ldots,m_c.$$

(iii) $i = n + m_c$: $J(\boldsymbol{e}_i) = \boldsymbol{X}$. Thus $||J(\boldsymbol{e}_i)||_F = ||\boldsymbol{X}||_F$.

(iv) $i = n + m_c + j$, where $1 \le j \le m$: $J(\boldsymbol{e}_i) = \mathrm{sMat}_{\mathcal{E}}(\boldsymbol{e}_j^y)\boldsymbol{X}$, where $\boldsymbol{e}_j^y$ is the j-th unit vector in $\mathbb{R}^m$. We have: $\mathrm{sMat}_{\mathcal{E}}(\boldsymbol{e}_j^y) = \boldsymbol{E}_{\mathrm{ind}_{\mathcal{E}}^{-1}(j)}$. Similar to the previous case, we obtain the following formulation:

$$||J(\boldsymbol{e}_i)||_F = \frac{1}{\sqrt{2}}\sqrt{\left|\left|\boldsymbol{X}_{k_{\mathcal{E}}(j),:}\right|\right|^2 + \left|\left|\boldsymbol{X}_{l_{\mathcal{E}}(j),:}\right|\right|^2}, \quad \forall\, i = n+m_c+j,\; j = 1,\ldots,m,$$

where $(k_{\mathcal{E}}(j), l_{\mathcal{E}}(j)) = \mathrm{ind}_{\mathcal{E}}^{-1}(j)$.

3.2.2 Block-Diagonal Preconditioning

For block preconditioner, we need to find the structure of the matrix representations of $\mathcal{Z}^*\mathcal{Z}$ and $\mathcal{X}^*\mathcal{X}$. We first calculate the columns of $\mathcal{Z}^*\mathcal{Z}$. There are two cases:

1. Column i of of $\mathcal{Z}^*\mathcal{Z}$, $1 \le i \le n-1$:
   $\mathcal{Z}(\boldsymbol{e}_i) = \boldsymbol{Z}\,\mathrm{Diag}(\boldsymbol{V}\boldsymbol{e}_i^w)$. Let $\boldsymbol{M} = \mathcal{Z}(\boldsymbol{e}_i)$, we have: $\boldsymbol{M}$ has only two non-zero columns, the first one, $\boldsymbol{M}_{:,1} = -\boldsymbol{Z}_{:,1}$, and the $(i+1)$-the one, $\boldsymbol{M}_{:,i+1} = \boldsymbol{Z}_{:,i+1}$.

   Now consider $\mathcal{Z}^*(\boldsymbol{M})$, the first $n-1$ elements is $\boldsymbol{V}^T\,\mathrm{diag}(\boldsymbol{M}^T\boldsymbol{Z})$. Let $\boldsymbol{P} = \boldsymbol{M}^T\boldsymbol{Z}$, we have: $\boldsymbol{P}$ again has only two non-zero rows, the first and $(i+1)$-th row. The elements of the first row of $\boldsymbol{P}$ is

   $$P_{1,k} = -\boldsymbol{Z}_{:,1}^T\boldsymbol{Z}_{:,k}, \quad \forall\, k = 1,\ldots,n,$$

   and the elements of the $(i+1)$-th row is

   $$P_{i+1,k} = \boldsymbol{Z}_{:,i+1}^T\boldsymbol{Z}_{:,k}, \quad \forall\, k = 1,\ldots,n.$$

   Thus $\mathrm{diag}(\boldsymbol{P})$ has two non-zero elements, the first, $-\boldsymbol{Z}_{:,1}^T\boldsymbol{Z}_{:,1}$, and the $(i+1)$-th element, $\boldsymbol{Z}_{:,i+1}^T\boldsymbol{Z}_{:,i+1}$. We then have $\boldsymbol{V}^T\,\mathrm{diag}(\boldsymbol{P}) = \boldsymbol{Z}_{:,1}^T\boldsymbol{Z}_{:,1}\boldsymbol{e} + \boldsymbol{Z}_{:,i+1}^T\boldsymbol{Z}_{:,i+1}\boldsymbol{e}_i^w$. The first $(n-1)\times(n-1)$ block of $\mathcal{Z}^*\mathcal{Z}$ can be written as $\mathrm{Diag}\left(||\boldsymbol{Z}_{:,2}||_F^2,\ldots,||\boldsymbol{Z}_{:,n}||_F^2\right) + ||\boldsymbol{Z}_{:,1}||_F^2\,\boldsymbol{E}$. The Cholesky decomposition of this block can be obtained using the faster rank-one update algorithm since the Cholesky decomposition of diagonal matrices are easy to compute and $\boldsymbol{E} = \boldsymbol{e}\boldsymbol{e}^T$. Another approach is to analytically find $\boldsymbol{R}^{-1}$ with a special structure such that $\mathrm{Diag}\left(||\boldsymbol{Z}_{:,2}||_F^2,\ldots,||\boldsymbol{Z}_{:,n}||_F^2\right) + ||\boldsymbol{Z}_{:,1}||_F^2\,\boldsymbol{E} = \boldsymbol{R}^T\boldsymbol{R}$ and $\boldsymbol{R}^{-1}\boldsymbol{x}$ is easy to compute for any vector $\boldsymbol{x}$. We apply the following result:

   Lemma 1. *Consider matrix $\boldsymbol{A} = \boldsymbol{D} + \boldsymbol{u}\boldsymbol{u}^T$ where $\boldsymbol{D}$ is a positive definite diagonal matrix, then $\boldsymbol{A}$ can be decomposed as $\boldsymbol{R}^T\boldsymbol{R}$ with*

   $$\boldsymbol{R}^{-1} = \boldsymbol{D}^{-\frac{1}{2}}\left(\boldsymbol{I} + \frac{1}{\lambda}\left(\frac{1}{\sqrt{\lambda+1}} - 1\right)\boldsymbol{p}\boldsymbol{p}^T\right),$$

   where $\boldsymbol{p} = \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{u}$ and $\lambda = ||\boldsymbol{p}||^2$.

Proof. We have: $\boldsymbol{A} = \boldsymbol{D}^{\frac{1}{2}}\boldsymbol{D}^{\frac{1}{2}} + \boldsymbol{D}^{\frac{1}{2}}\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{u}\boldsymbol{u}^T\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{D}^{\frac{1}{2}} = \boldsymbol{D}^{\frac{1}{2}}\left(\boldsymbol{I} + \boldsymbol{p}\boldsymbol{p}^T\right)\boldsymbol{D}^{\frac{1}{2}}$.
Assume $\boldsymbol{p}\boldsymbol{p}^T$ has the eigenvalue decomposition $\boldsymbol{V}(\lambda\boldsymbol{E}_{11})\boldsymbol{V}^T$, we have: $\boldsymbol{V}\boldsymbol{V}^T = \boldsymbol{I}$, thus

$$\boldsymbol{A} = \boldsymbol{D}^{\frac{1}{2}}\boldsymbol{V}\left(\boldsymbol{I} + \lambda\boldsymbol{E}_{11}\right)\boldsymbol{V}^T\boldsymbol{D}^{\frac{1}{2}}.$$

$\boldsymbol{I} + \lambda\boldsymbol{E}_{11}$ is a diagonal matrix; therefore, we can compute $\boldsymbol{A}^{-1}$ as follows:

$$\boldsymbol{A}^{-1} = \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{V}\left(\boldsymbol{I} + \left(\frac{1}{\lambda+1} - 1\right)\boldsymbol{E}_{11}\right)\boldsymbol{V}^T\boldsymbol{D}^{-\frac{1}{2}}.$$

Applying the formulation for square roots of diagonal matrices, we have:

$$\boldsymbol{A}^{-1} = \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{V}\left(\boldsymbol{I} + \left(\frac{1}{\sqrt{\lambda+1}} - 1\right)\boldsymbol{E}_{11}\right)\boldsymbol{V}^T\boldsymbol{V}\left(\boldsymbol{I} + \left(\frac{1}{\sqrt{\lambda+1}} - 1\right)\boldsymbol{E}_{11}\right)\boldsymbol{V}^T\boldsymbol{D}^{-\frac{1}{2}}.$$

We also have $\lambda\boldsymbol{V}\boldsymbol{E}_{11}\boldsymbol{V}^T = \boldsymbol{p}\boldsymbol{p}^T$, thus

$$\boldsymbol{A}^{-1} = \boldsymbol{D}^{-\frac{1}{2}}\left(\boldsymbol{I} + \frac{1}{\lambda}\left(\frac{1}{\sqrt{\lambda+1}} - 1\right)\boldsymbol{p}\boldsymbol{p}^T\right)\left(\boldsymbol{I} + \frac{1}{\lambda}\left(\frac{1}{\sqrt{\lambda+1}} - 1\right)\boldsymbol{p}\boldsymbol{p}^T\right)\boldsymbol{D}^{-\frac{1}{2}} = \boldsymbol{S}\boldsymbol{S}^T,$$

where $\boldsymbol{S} = \boldsymbol{D}^{-\frac{1}{2}}\left(\boldsymbol{I} + \frac{1}{\lambda}\left(\frac{1}{\sqrt{\lambda+1}} - 1\right)\boldsymbol{p}\boldsymbol{p}^T\right)$.
Let $\boldsymbol{R} = \boldsymbol{S}^{-1}$, clearly we have $\boldsymbol{A} = \boldsymbol{R}^T\boldsymbol{R}$ and $\boldsymbol{R}^{-1} = \boldsymbol{S}$. $\square$

Clearly, in order to calculate $\boldsymbol{R}^{-1}\boldsymbol{x}$, we only need to calculate $\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{x}$, $\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{u}$. For this particular matrix, we can set $\boldsymbol{D} = \mathrm{Diag}\left(||\boldsymbol{Z}_{:,2}||_F^2, \ldots, ||\boldsymbol{Z}_{:,n}||_F^2\right)$ and $\boldsymbol{u} = ||\boldsymbol{Z}_{:,1}||_F\,\boldsymbol{e}$.

We now look at the remaining m_c elements of the i-th columns, $\boldsymbol{q} = \frac{1}{2}\mathrm{svec}_{\mathcal{E}^c}(\boldsymbol{P}+\boldsymbol{P}^T)$. $\boldsymbol{P}$ has two non-zero rows; therefore, $\boldsymbol{P}+\boldsymbol{P}^T$ has two non-zero rows and two non-zeros columns. The elements of these rows and columns are the same as the elements of corresponding rows in $\boldsymbol{P}$ except for four elements, $(1,1)$, $(1,i+1)$, $(i+1,1)$, and $(i+1,i+1)$. Note that the elements $(1,i+1)$ and $(i+1,1)$ are zeros as $-\boldsymbol{Z}_{:,1}^T\boldsymbol{Z}_{:,i} - \boldsymbol{Z}_{:,i}^T\boldsymbol{Z}_{:,1} = 0$. Let $\mathcal{E}_i^c$ be the set of all edges in $\mathcal{E}^c$ starting from node i, we have: for all $j \in \mathrm{ind}_{\mathcal{E}^c}(\mathcal{E}_1^c)$ such that $\mathrm{ind}_{\mathcal{E}^c}^{-1}(j) = (1, l_{\mathcal{E}^c}(j)) \neq (1, i+1)$, $q_j = -\frac{1}{\sqrt{2}}\boldsymbol{Z}_{:,1}^T\boldsymbol{Z}_{:,l_{\mathcal{E}^c}(j)}$. Similarly, for all $j \in \mathrm{ind}_{\mathcal{E}^c}(\mathcal{E}_{i+1}^c)$ such that $\mathrm{ind}_{\mathcal{E}^c}^{-1}(j) = (i+1, l_{\mathcal{E}^c}(j))$, $q_j = \frac{1}{\sqrt{2}}\boldsymbol{Z}_{:,i+1}^T\boldsymbol{Z}_{:,l_{\mathcal{E}^c}(j)}$. Other elements of $\boldsymbol{q}$ are zeros. Note that for all $n-1$ first columns, the number of non-zeros depends on $|\mathcal{E}_1^c|$. Thus in order to increase the sparsity of the resulting matrix, we should select the node 1 with the *largest* degree (in the original graph). This is one of the operations that we can consider for the presolve.

2. Column i of of $\mathcal{Z}^*\mathcal{Z}$, $i = n-1+j$, where $1 \leq j \leq m_c$:
$\mathcal{Z}(\boldsymbol{e}_i) = \boldsymbol{Z}\,\mathrm{sMat}_{\mathcal{E}^c}(\boldsymbol{e}_j^v)$. Let $\boldsymbol{M} = \mathcal{Z}(\boldsymbol{e}_i)$ and $\mathrm{ind}_{\mathcal{E}^c}^{-1}(j) = (k,l)$, we again have that $\boldsymbol{M}$ has only two non-zero columns, the k-th column, $\boldsymbol{M}_{:,k} = \frac{1}{\sqrt{2}}\boldsymbol{Z}_{:,l}$, and the l-th column, $\boldsymbol{M}_{:,l} = \frac{1}{\sqrt{2}}\boldsymbol{Z}_{:,k}$. Due to the symmetry of $\mathcal{Z}^*\mathcal{Z}$, we just need to look at the last m_c elements of these columns. Let $\boldsymbol{q} = \frac{1}{2}\mathrm{svec}_{\mathcal{E}^c}(\boldsymbol{P}+\boldsymbol{P}^T)$, where $\boldsymbol{P} = \boldsymbol{M}^T\boldsymbol{Z}$. We have $\boldsymbol{P}$ has two non-zero rows, the k-th

and l-th ones. The elements are $P_{k,p} = \frac{1}{\sqrt{2}}\boldsymbol{Z}_{:,l}^T\boldsymbol{Z}_{:,p}$ and $P_{l,p} = \frac{1}{\sqrt{2}}\boldsymbol{Z}_{:,k}^T\boldsymbol{Z}_{:,p}$ for all $p = 1, \dots, n$. Similar to the previous case, we have that elements of $\boldsymbol{P}+\boldsymbol{P}^T$ are the corresponding elements of $\boldsymbol{P}$ except for four elements (k,k), (k,l), (l,k), and (l,l). The element (k,l) (and (l,k)) has the value $\frac{1}{\sqrt{2}}\left(||\boldsymbol{Z}_{:,k}||_F^2 + ||\boldsymbol{Z}_{:,l}||_F^2\right)$. Thus we have: $q_j = \frac{1}{2}\left(||\boldsymbol{Z}_{:,k}||_F^2 + ||\boldsymbol{Z}_{:,l}||_F^2\right)$. Other non-zero elements of $\boldsymbol{q}$ are the elements with indices in the set $\mathrm{ind}_{\mathcal{E}^c}(\mathcal{E}_k^c \cup \mathcal{E}_l^c)$ and the computation of these elements is similar to the previous case. With this structure of non-zero elements for each column, we can see that the index mapping $\mathrm{ind}_{\mathcal{E}^c}$ with a *good graph partitioning* results in a block structure for this $m_c \times m_c$ block of $\mathcal{Z}^*\mathcal{Z}$. More specifically, there are two cases in which the block structure can be form. In the first case, three edges (i,j), (j,k), and (i,k) (edges of a clique of size 3) are indexed consecutively and results in 3×3 block. However, there is no special structure of this block to be exploited. We now focus on the second case in which the set of edges from a node i, (i,j) with $j \in \mathcal{S}_i$, $\mathcal{S}_i \subset \mathcal{E}_i^c$, are indexed consecutively. Let $\boldsymbol{Z}_{\mathcal{S}_i}$ be the $n \times |\mathcal{S}_i|$ submatrix of $\boldsymbol{Z}$ that consists of $|\mathcal{S}_i|$ columns $\boldsymbol{Z}_{:,j}$, $j \in \mathcal{S}_i$, the block we obtain is $\frac{1}{2}\left(||\boldsymbol{Z}_{:,i}||_F^2\boldsymbol{I} + \boldsymbol{Z}_{\mathcal{S}_i}^T\boldsymbol{Z}_{\mathcal{S}_i}\right)$. And this is another operation we need to consider for the presolve.

We continue with columns of $\mathcal{X}^*\mathcal{X}$. We also have two cases:

1. $i = 1$: $\mathcal{X}(\boldsymbol{e}_i) = \boldsymbol{X}$. Thus the first column is $\mathcal{X}^*(\boldsymbol{X})$ with the first element is $\mathrm{trace}(\boldsymbol{X}^2)$ and the remaining m elements are $\mathrm{svec}_{\mathcal{E}}(\boldsymbol{X}^2)$.

2. $i = j+1$, where $1 \le j \le m$: $\mathcal{X}(\boldsymbol{e}_i) = \mathrm{sMat}_{\mathcal{E}}(\boldsymbol{e}_j^y)\boldsymbol{X}$. Let $\boldsymbol{M} = \mathcal{X}(\boldsymbol{e}_i)$ and $\mathrm{ind}_{\mathcal{E}}(j) = (k,l)$, we have: $\boldsymbol{M}$ has only two non-zero rows, the k-th row, $\boldsymbol{M}_{k,:} = \frac{1}{\sqrt{2}}\boldsymbol{X}_{l,:}$ and the l-th row, $\boldsymbol{M}_{l,:} = \frac{1}{\sqrt{2}}\boldsymbol{X}_{k,:}$. Due to the symmetry of $\mathcal{X}^*\mathcal{X}$, we do not need to reconsider the first elements of these columns. Let $\boldsymbol{P} = \boldsymbol{M}\boldsymbol{X}$, we have, the remaining m elements are $\boldsymbol{q} = \frac{1}{2}\mathrm{svec}_{\mathcal{E}}(\boldsymbol{P}+\boldsymbol{P}^T)$. Similarly, $\boldsymbol{P}$ has only two non-zeros rows, the k-th and the l-th ones with the elements $P_{k,p} = \frac{1}{\sqrt{2}}\boldsymbol{X}_{l,:}\boldsymbol{X}_{:,p}$ and $P_{l,p} = \frac{1}{\sqrt{2}}\boldsymbol{X}_{l,:}\boldsymbol{X}_{:,p}$ for all $p = 1, \dots, n$. The elements of $\boldsymbol{P} + \boldsymbol{P}^T$ can be derived from elements of $\boldsymbol{P}$ in a similar way shown before. The element (k,l) is $\frac{1}{\sqrt{2}}\left(||\boldsymbol{X}_{k,:}||_F^2 + ||\boldsymbol{X}_{l,:}||_F^2\right)$. Thus we have: $q_j = \frac{1}{2}\left(||\boldsymbol{X}_{k,:}||_F^2 + ||\boldsymbol{X}_{l,:}||_F^2\right)$. Other non-zero elements of $\boldsymbol{q}$ are the elements with indices in the set $\mathrm{ind}_{\mathcal{E}}(\mathcal{E}_k \cup \mathcal{E}c_l)$, where $\mathcal{E}_i$ is the set of all edges in $\mathcal{E}$ starting from node i. With this structure of non-zero elements for each column, we can again see that the index mapping $\mathrm{ind}_{\mathcal{E}}$ with a *good graph partitioning* results in a block structure for this $m \times m$ block of $\mathcal{X}^*\mathcal{X}$, especially when edges in $\mathcal{E}$ from a single node are indexed consecutively. Similarly, the presolve can help us obtain this *good graph partitioning.*

4 Numerics and Conclusion

4.1 Numerical Results

We test three versions of the Gauss-Newton algorithm: (i) a general version for full matrices; (ii) a sparse version for sparse matrices; and (iii) a specialized version for the Lovász theta function problem, with different preconditioners. We start with the sparse version of the code. Data inputs are $\boldsymbol{C} \in \mathbb{R}^{n\times n}$, $\boldsymbol{A}_i \in \mathbb{R}^{n\times n}$, $i = 1, \ldots, m$, and $\boldsymbol{b} \in \mathbb{R}^m$. All matrices are sparse with density set (in MATLAB) to be $1/(4n)$. We set $n = 100$ and $m = 100$ and generate data inputs randomly. We run $N = 1000$ instances and record the (average) of: the number of iterations, the relative norm of $\boldsymbol{ZX}$, and the relative minimum eigenvalues of $\boldsymbol{X}$ and $\boldsymbol{Z}$, see Table 4.1. We apply the cross-over technique and use the diagonal preconditioner as the setting for our code. The tolerance is set to 10^{-12}. The average shown for the two accuracy measures is *log-average*. The two measures are

$$\text{RelZXnorm} = \frac{||\boldsymbol{ZX}||_F}{|\boldsymbol{C}\cdot\boldsymbol{X}|+1}, \quad \text{Relmineig} = \frac{\min\{\lambda_{\min}(\boldsymbol{X}), \lambda_{\min}(\boldsymbol{Z})\}}{|\boldsymbol{C}\cdot\boldsymbol{X}|+1}.$$

	Iteration	RelZXnorm	Relmineig
Average	18.66	2.62×10^{-15}	-8.79×10^{-16}
Best	14.00	2.78×10^{-16}	-3.86×10^{-17}
Worst	26.00	8.36×10^{-13}	-1.97×10^{-13}

Table 4.1: Accuracy measures for random sparse instances

We now select 20 hard instances in terms of number of iterations taken using our SRSD1 code. Using these 20 instances, we compare our solver and the four SDP solvers:

SeDuMi 1.3, CSDP 6.1.0, SDPA 7.3.1, SDPT3 4.0.

The tolerances (for the relative trace of $\boldsymbol{ZX}$) for these four solvers are set to be as small as possible without running into numerical problems. For SDPT3, it could be 10^{-13} or 10^{-14}. For the remaining solvers, 10^{-12} is sufficiently small and there are some cases, where we had to reduce the tolerance for SeDuMi or CSDP. The two accuracy measures are compared and in addition, we also record all six DIMACS error measures [12]. (The last measure is only comparable if both $\boldsymbol{X}$ and $\boldsymbol{Z}$ are positive semidefinite.) We also compare the computational times of all solvers for these 20 instances. Similarly, we also select 20 easy instances in terms of number of iterations taken using our SRSD1 code and compare with other solvers. The numerical results are shown in Table 4.2 and 4.3, respectively.

The computational time for SRSD1 is not competitive with the other solvers. One reason is that the code right now is all MATLAB code with no pre-compiled subroutines. However, the main reason is that the Gauss-Newton approach needs to solve a system of equations with $n(n+1)/2$ variables regardless of the number of constraints. We now keep $n = 100$ and vary the number of constraints from $m = 500$ to $m = 5000$. The instances are generated randomly with the same sparse density $1/(4n)$. For larger instances, we need more accuracy for the LSMR subroutine that iteratively solves for the Gauss-Newton direction. Therefore, for these instances, we use the block triangular preconditioner instead of the diagonal one. The average number of iterations and two accuracy measures for these instances are shown in the Table 4.4. The corresponding computational time ratios of SRSD2 to those of other solvers are then reported in Table 4.5.

HW ♣ to go along with the tests for changing m (number of constraints), we should generate problems with changing sparsity using given Z, X with given sparsity and optimal, i.e. generate the problem with sparse A and sparse Q and make these given sparse Z, X optimal. This is cheating

	SRSD1	SeDuMi	CSDP	SDPA	SDPT3
Iteration	23.45	19.15	16.50	16.30	24.50
RelZXnorm	7.18×10^{-15}	1.50×10^{-07}	8.92×10^{-08}	5.86×10^{-08}	1.01×10^{-10}
Relmineig	-1.12×10^{-15}	-8.72×10^{-15}	3.28×10^{-13}	1.46×10^{-13}	5.99×10^{-17}
DIMACS1	1.28×10^{-15}	7.93×10^{-11}	1.24×10^{-13}	2.48×10^{-13}	3.19×10^{-13}
DIMACS2	2.28×10^{-14}	0.00×10^{00}	0.00×10^{00}	0.00×10^{00}	0.00×10^{00}
DIMACS3	9.39×10^{-16}	3.53×10^{-16}	1.64×10^{-08}	1.44×10^{-15}	8.72×10^{-14}
DIMACS4	3.22×10^{-15}	3.42×10^{-13}	0.00×10^{00}	0.00×10^{00}	0.00×10^{00}
DIMACS5	1.19×10^{-15}	9.72×10^{-13}	1.54×10^{-09}	4.96×10^{-10}	1.10×10^{-13}
DIMACS6	4.65×10^{-16}	2.84×10^{-14}	1.16×10^{-09}	4.96×10^{-10}	2.01×10^{-13}
Time	61.89	1.23	0.60	0.74	1.13

Table 4.2: Performance measures for “hard” random sparse instances

	SRSD1	SeDuMi	CSDP	SDPA	SDPT3
Iteration	15.20	17.50	15.10	15.55	22.40
RelZXnorm	1.64×10^{-14}	1.66×10^{-07}	1.04×10^{-06}	2.17×10^{-08}	1.77×10^{-10}
Relmineig	-5.56×10^{-15}	-1.15×10^{-14}	2.27×10^{-13}	2.20×10^{-13}	1.21×10^{-16}
DIMACS1	1.24×10^{-15}	1.34×10^{-10}	1.38×10^{-13}	5.20×10^{-14}	2.74×10^{-13}
DIMACS2	1.15×10^{-13}	0.00×10^{00}	0.00×10^{00}	0.00×10^{00}	0.00×10^{00}
DIMACS3	8.35×10^{-16}	3.57×10^{-16}	1.04×10^{-08}	1.44×10^{-15}	8.95×10^{-14}
DIMACS4	1.32×10^{-14}	4.34×10^{-13}	0.00×10^{00}	0.00×10^{00}	0.00×10^{00}
DIMACS5	2.19×10^{-15}	1.98×10^{-12}	1.18×10^{-09}	4.85×10^{-10}	9.34×10^{-14}
DIMACS6	1.90×10^{-15}	2.95×10^{-14}	5.47×10^{-10}	4.85×10^{-10}	1.86×10^{-13}
Time	37.00	1.16	0.53	0.59	1.05

Table 4.3: Performance measures for “easy” random sparse instances

	SRSD2	SeDuMi	CSDP	SDPA	SDPT3
Iteration	14.82	15.73	15.36	16.00	21.09
RelZXnorm	6.93×10^{-15}	1.99×10^{-07}	1.91×10^{-07}	1.77×10^{-08}	2.27×10^{-09}
Relmineig	-3.69×10^{-16}	-4.94×10^{-15}	4.06×10^{-13}	9.06×10^{-14}	2.31×10^{-16}

Table 4.4: Average measures for random instances with different numbers of constraints

but it will generate valid very sparse problems. If we show as this sparsity increases our method does better then that would be very nice. So I think this is important to do!! So:

1. pick $D_X, D_Z \succeq 0$ diagonal with $D_X + D_Z \succ 0$, $D_X D_Z = 0$.
2. Pick U orthogonal and very sparse, e.g. with pieces of permutation matrices? Set $X = UD_XU^T$, $Z = UD_ZU^T$ so both X, Z are very sparse and they satisfy strict complementarity/optimality.
3. Now choose $A = [S|E]$ with S triangular-sparse and/or diagonal and E very sparse. So we have a sparse Q for the null space. Set $b = A \operatorname{svec} X$ y random and $C = Z + \operatorname{sMat} A^T y$.
4. So we now have X, y, Z optimal and we have a very sparse problem. let's make this problem more and more sparse and see how our algorithm compares with the others. Hopefully this shows that we exloit sparsity completely. I am sure that we can solve HUGE problems if they are very very sparse!!!
5. If we can figure out how to make the off diagonal blocks of J^*J zero at optimality, then our preconditioning will do well also. I will try and figure that out next.

m	500	1000	1500	2000	2500	3000	3500	4000	4500	5000
SeDuMi	168.74	49.99	21.47	4.18	1.92	1.89	1.09	0.66	0.68	0.32
CSDP	144.02	39.62	21.64	7.58	5.89	5.56	3.53	2.40	2.88	1.62
SDPA	488.28	203.06	134.49	50.99	37.09	33.37	20.98	15.55	17.83	9.41
SDPT3	87.49	27.96	16.10	6.00	3.80	4.42	2.11	2.30	2.89	1.81

Table 4.5: Time ratios for random instances with different numbers of constraints

In the previous test, we keep the sparse density constant while changing the number of constraint. We now would like to test our code with instances created with different sparse densities. We keep $n = 100$ and $m = 2500$ while varying the sparse density $1/(4sn)$ with $s = 1, \ldots, 10$. All parameter settings are the same as in the previous test. In addition, we also run our code with the diagonal preconditioner when the data is sparse enough. It turns out in this test, for $s \geq 3$, SRSD1 maintains the same level of accuracy but is more efficient than SRSD2 in computational time. This can be explained as follows. SRSD2 in general requires less iterations of the LSMR subroutine than SRSD1 and if the data is not sparse enough, this saving in time can compensate for the more expensive construction of its preconditioner. On the other hand, if the data is sparser, for example, when $s \geq 3$ in this case, the reduction in the number of LSMR iterations of SRSD1 gradually becomes more significant than that of SRSD2, which makes SRSD1 more efficient. The time comparison between SRSD1 and SRSD2 is shown in Table 4.6. Note that since with $s = 2$, SRSD2 is already more efficient than SRSD1, we do not run SRSD1 for the instance with $s = 1$. To compare with other solvers, we select the more efficient code between SRSD1 and SRSD2 for each instance. Basically, the first two instances are run with SRSD2 while the remaining ones are with SRSD1. The number of iterations and two accuracy measures are reported in Table 4.7. Computational time ratios of SRSD1/2 to other solvers are shown in Table 4.8.

We have seen that for well-conditioned instances, our code can achieve solutions with very high accuracy. We now move on to test the code with ill-conditioned instances, namely, the instances without strict complementary slackness or instances with which Slater's condition almost fails. To generate instances without strict complementary slackness, we use the code developed by Wei and

s	2	3	4	5	6	7	8	9	10
SRSD1	1080.72	201.44	182.98	230.35	102.31	76.47	80.29	57.92	52.29
SRSD2	447.55	362.80	314.22	485.58	485.93	312.37	410.73	359.74	352.83
Time ratio	0.41	1.80	1.72	2.11	4.75	4.08	5.12	6.21	6.75

Table 4.6: Computational times for random instances with different sparse densities

	SRSD1/2	SeDuMi	CSDP	SDPA	SDPT3
Iteration	13.80	16.80	16.20	16.20	24.30
RelZXnorm	4.47×10^{-15}	5.59×10^{-08}	1.30×10^{-07}	3.80×10^{-09}	3.65×10^{-11}
Relmineig	-7.59×10^{-16}	-2.24×10^{-16}	4.96×10^{-13}	8.71×10^{-14}	8.51×10^{-18}

Table 4.7: Average measures for random instances with different sparse densities

Wolkowicz [14]. The instances are generated with $n = 50$ and $m = 1000$. The general version of the code will be used since all matrices are dense. For these hard instances, we apply the block preconditioner without cross-over. The tolerance is set to be 10^{-14}. The results are shown in Table 4.9.

HW ♣ We should note to the reader that our code does much better on the RelZXnorm measure. can we say why? it is interesting that this is the case.

The accuracy indeed is indeed lower for these hard instances, even with our code. And the stopping criterion for all tested instances but one is when the number of iterations of LSMR subroutine reaches the maximum limit, which is set at $5n(n+1)$. SeDuMi and SDPA, and SDPT3 have all different kinds of numerical problems. CSDP controls the tolerance limit (around 10^{-08}); therefore, it does not incur any numerical problem even though the tolerance is set to be as small as 10^{-14}.

For instances with which Slater's condition almost fails, we generate the instances randomly using the alternative theorem for Slater's condition with respect to the dual problem. The setting is the same for the previous test with the tolerance of 10^{-14}, block preconditioner, and no cross-over. The results are in Table 4.10.

It turns out that for other solvers, instances with which Slater's condition almost fails are difficult to solve but not for our code. Even though no cross-over is used, half of the instances have negative relative minimum eigenvalue of $\boldsymbol{X}$ and $\boldsymbol{Z}$ with average DIMACS2 and DIMACS4 measures being 1.05×10^{-17} and 2.53×10^{-16}, respectively. We also solve these instances with cross-over and the results are better in terms of computational time (number of iterations) with the trade-off of negative relative minimum eigenvalue of both $\boldsymbol{X}$ and $\boldsymbol{Z}$ while maintaining the accuracy. The average performance measures of GRSD2 with cross-over is $(13.70, 3.40\times10^{-16}, -1.41\times10^{-13}, 2.78\times 10^{-13}, 5.17 \times 10^{-15}, 1.61 \times 10^{-12}, 9.97 \times 10^{-16}, 1.72 \times 10^{-14}, 4.71 \times 10^{-17}, 22.50)$. This is probably due to the fact that the problem is still well-conditioned even though Slater's condition almost fails. (The Jacobian is non-singular at the optimal solution.)

HW ♣ the fact that our code does well on these hard instances means that it is a very useful tool that can be used to test out other algorithms, i.e. to test other algorithms on hard problems and see how they do compared to a code that handles hard problems. so .. a good motivation/application for using our code.

s	1	2	3	4	5	6	7	8	9	10
SeDuMi	2.84	2.74	1.26	1.15	1.49	0.62	0.53	0.49	0.34	0.30
CSDP	7.33	12.17	6.07	6.77	8.06	4.14	3.21	3.39	2.49	2.17
SDPA	43.35	48.63	22.13	20.95	25.47	12.33	9.25	9.82	7.08	6.04
SDPT3	4.60	8.82	5.79	5.74	7.50	3.94	2.86	2.75	2.27	2.09

Table 4.8: Time ratios for random instances with different sparse densities

	GRSD2	SeDuMi	CSDP	SDPA	SDPT3
Iteration	29.20	13.60	12.10	12.00	22.70
RelZXnorm	1.32×10^{-12}	4.12×10^{-06}	6.53×10^{-05}	1.91×10^{-07}	2.97×10^{-08}
Relmineig	5.96×10^{-20}	6.20×10^{-14}	1.44×10^{-15}	3.25×10^{-12}	1.21×10^{-18}
DIMACS1	2.16×10^{-12}	2.01×10^{-07}	3.50×10^{-10}	3.79×10^{-07}	4.62×10^{-10}
DIMACS2	0.00×10^{00}	0.00×10^{00}	0.00×10^{00}	0.00×10^{00}	0.00×10^{00}
DIMACS3	1.81×10^{-11}	6.34×10^{-15}	2.10×10^{-08}	2.09×10^{-14}	1.54×10^{-11}
DIMACS4	0.00×10^{00}	1.48×10^{-14}	0.00×10^{00}	0.00×10^{00}	0.00×10^{00}
DIMACS5	2.07×10^{-12}	9.23×10^{-09}	1.23×10^{-08}	1.58×10^{-07}	5.70×10^{-11}
DIMACS6	1.86×10^{-12}	4.99×10^{-09}	4.25×10^{-08}	3.50×10^{-07}	1.06×10^{-10}
Time	281.45	64.02	69.50	27.31	60.25

Table 4.9: Performance measures for random instances without strict complementary slackness

	GRSD2	SeDuMi	CSDP	SDPA	SDPT3
Iteration	26.00	17.10	13.80	15.20	20.40
RelZXnorm	9.90×10^{-16}	6.14×10^{-07}	4.61×10^{-07}	2.05×10^{-07}	2.13×10^{-08}
Relmineig	1.05×10^{-17}	1.06×10^{-15}	1.60×10^{-12}	1.06×10^{-10}	1.47×10^{-15}
DIMACS1	2.78×10^{-13}	1.07×10^{-10}	1.10×10^{-12}	1.17×10^{-13}	8.41×10^{-11}
DIMACS2	0.00×10^{00} (*)	0.00×10^{00}	0.00×10^{00}	0.00×10^{00}	0.00×10^{00}
DIMACS3	1.42×10^{-12}	3.56×10^{-14}	5.21×10^{-09}	6.83×10^{-08}	2.59×10^{-13}
DIMACS4	0.00×10^{00} (*)	1.48×10^{-14}	0.00×10^{00}	0.00×10^{00}	0.00×10^{00}
DIMACS5	1.09×10^{-14}	2.77×10^{-12}	1.38×10^{-09}	2.02×10^{-14}	1.26×10^{-11}
DIMACS6	3.44×10^{-15}	3.16×10^{-12}	8.54×10^{-10}	4.43×10^{-08}	1.82×10^{-12}
Time	58.04	75.02	76.42	32.32	48.98

Table 4.10: Performance measures for random instances with Slater's condition almost failed

The final test is for the Lovász theta function problem. We generate random graph with $n = 100$ nodes and the number of edges is approximately $n(n-1)/4$. According to [5, 10], these are the most difficult instances to solve. We first test the specialized version of the code and the special block preconditioner that exploits the block structure of the problem. The tolerance is set to 10^{-12} and we run the code with cross-over for 100 random instances. The results for TRSD3 are shown in Table 4.11.

	Iteration	RelZXnorm	Relmineig
Average	18.31	4.85×10^{-15}	2.47×10^{-13}
Best	16.00	6.42×10^{-17}	-1.28×10^{-15}
Worst	23.00	9.74×10^{-13}	-3.58×10^{-11}

Table 4.11: Accuracy measures for random Lovász theta function instances

We now select 10 hard instances in terms of number of iterations and compare the results using other solvers including SRSD1 and TRSD1, both with cross-over since all these instances are well-conditioned. The results are in Tables 4.12 and 4.13. Similarly, 10 easy instances are selected and the numerical results for these instances are shown in Table 4.14 and 4.15.

	SeDuMi	CSDP	SDPA	SDPT3	TRSD3
Iteration	21.00	17.10	16.00	22.60	21.00
RelZXnorm	5.00×10^{-08}	1.56×10^{-08}	2.38×10^{-07}	1.73×10^{-11}	2.67×10^{-14}
Relmineig	3.34×10^{-14}	7.09×10^{-14}	3.49×10^{-12}	2.32×10^{-18}	6.74×10^{-13}
DIMACS1	1.84×10^{-11}	6.07×10^{-14}	5.09×10^{-14}	1.27×10^{-13}	1.25×10^{-16}
DIMACS2	0.00×10^{00}	0.00×10^{00}	0.00×10^{00}	0.00×10^{00}	3.11×10^{-15}
DIMACS3	9.09×10^{-16}	1.09×10^{-07}	1.52×10^{-14}	8.87×10^{-13}	8.75×10^{-15}
DIMACS4	2.39×10^{-13}	0.00×10^{00}	0.00×10^{00}	0.00×10^{00}	4.82×10^{-12}
DIMACS5	9.09×10^{-12}	4.44×10^{-10}	9.61×10^{-09}	1.47×10^{-13}	5.92×10^{-16}
DIMACS6	9.93×10^{-14}	2.10×10^{-10}	9.61×10^{-09}	7.08×10^{-14}	4.19×10^{-16}
Time	90.46	13.05	6.02	12.91	158.09

Table 4.12: Performance measures for “hard” random Lovász theta function instances

4.2 Concluding Remarks

Acknowledgment

We would like to thank Makoto Yamashita for helping us install SDPA 7.3.1 on our machine.

	SRSD1	TRSD1	TRSD3
Iteration	23.90	21.00	21.00
RelZXnorm	5.62×10^{-14}	3.26×10^{-14}	2.67×10^{-14}
Relmineig	2.23×10^{-12}	9.51×10^{-13}	6.74×10^{-13}
DIMACS1	9.89×10^{-17}	1.11×10^{-16}	1.25×10^{-16}
DIMACS2	5.76×10^{-15}	4.46×10^{-15}	3.11×10^{-15}
DIMACS3	1.89×10^{-14}	8.71×10^{-15}	8.75×10^{-15}
DIMACS4	1.59×10^{-11}	6.80×10^{-12}	4.82×10^{-12}
DIMACS5	1.73×10^{-15}	1.87×10^{-15}	5.92×10^{-16}
DIMACS6	1.75×10^{-15}	1.85×10^{-15}	4.19×10^{-16}
Time	214.99	126.74	158.09

Table 4.13: Performance measures for "hard" random Lovász theta function instances

	SeDuMi	CSDP	SDPA	SDPT3	TRSD3
Iteration	20.20	16.10	16.00	21.40	16.30
RelZXnorm	7.34×10^{-08}	8.33×10^{-08}	9.61×10^{-08}	6.76×10^{-11}	2.18×10^{-14}
Relmineig	3.54×10^{-14}	3.69×10^{-14}	1.97×10^{-12}	1.24×10^{-17}	5.36×10^{-13}
DIMACS1	1.78×10^{-11}	1.86×10^{-14}	2.28×10^{-15}	1.85×10^{-13}	1.23×10^{-16}
DIMACS2	0.00×10^{00}	0.00×10^{00}	0.00×10^{00}	0.00×10^{00}	4.02×10^{-15}
DIMACS3	8.21×10^{-16}	3.38×10^{-08}	1.55×10^{-14}	1.61×10^{-12}	8.70×10^{-15}
DIMACS4	2.51×10^{-13}	0.00×10^{00}	0.00×10^{00}	0.00×10^{00}	3.18×10^{-12}
DIMACS5	1.57×10^{-11}	1.35×10^{-10}	5.28×10^{-09}	2.92×10^{-13}	1.31×10^{-15}
DIMACS6	1.13×10^{-13}	8.91×10^{-11}	5.28×10^{-09}	1.88×10^{-13}	5.44×10^{-16}
Time	88.95	13.09	6.01	12.37	63.43

Table 4.14: Performance measures for "easy" random Lovász theta function instances

	SRSD1	TRSD1	TRSD3
Iteration	20.00	16.30	16.30
RelZXnorm	9.22×10^{-17}	3.03×10^{-14}	2.18×10^{-14}
Relmineig	3.41×10^{-15}	1.72×10^{-12}	5.36×10^{-13}
DIMACS1	1.35×10^{-16}	1.23×10^{-16}	1.23×10^{-16}
DIMACS2	2.48×10^{-17}	1.71×10^{-14}	4.02×10^{-15}
DIMACS3	1.74×10^{-14}	8.56×10^{-15}	8.70×10^{-15}
DIMACS4	2.41×10^{-14}	1.26×10^{-11}	3.18×10^{-12}
DIMACS5	1.91×10^{-16}	2.13×10^{-15}	1.31×10^{-15}
DIMACS6	2.67×10^{-17}	1.34×10^{-15}	5.44×10^{-16}
Time	96.39	47.34	63.43

Table 4.15: Performance measures for "easy" random Lovász theta function instances

References

[1] F. Alizadeh, J-P.A. Haeberly, and M.L. Overton. Complementarity and nondegeneracy in semidefinite programming. *Math. Programming*, 77:111–128, 1997. 3

[2] J.E. Dennis Jr. and R.B. Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations*, volume 16 of *Classics in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1996. Corrected reprint of the 1983 original. 9

[3] J.E. Dennis Jr. and H. Wolkowicz. Sizing and least-change secant methods. *SIAM J. Numer. Anal.*, 30(5):1291–1314, 1993. 12, 13

[4] M. Gonzalez-Lima, H. Wei, and H. Wolkowicz. A stable primal-dual approach for linear programming under nondegeneracy assumptions. *Comput. Optim. Appl.*, 44(2):213–247, 2009. 3

[5] G. Gruber and F. Rendl. Computational experience with ill-posed problems in semidefinite programming. *Comput. Optim. Appl.*, 21(2):201–212, 2002. 25

[6] D.E. Knuth. The sandwich theorem. *Electronic J. Combinatorics*, 1:48pp, 1994. 14

[7] S. Kruk, M. Muramatsu, F. Rendl, R.J. Vanderbei, and H. Wolkowicz. The Gauss-Newton direction in semidefinite programming. *Optim. Methods Softw.*, 15(1):1–28, 2001. 3, 10

[8] M. Laurent and F. Rendl. Semidefinite programming and integer programming. Technical Report PNA-R0210, CWI, Amsterdam, 2002. To appear as chapter of the Handbook on Discrete Optimization edited by K.Aardal, G. Nemhauser and R. Weismantel. 14

[9] L. Lovász. On the Shannon capacity of a graph. *IEEE Trans. Inform. Theory*, 25(1):1–7, 1979. 14

[10] J. Malick, J. Povh, F. Rendl, and A. Wiegele. Regularization methods for semidefinite programming. *SIAM Journal on Optimization*, 20(1):336–356, 2009. 25

[11] O.L. Mangasarian. *Nonlinear Programming*. McGraw-Hill, New York, NY, 1969. 12

[12] H.D. Mittelmann. An independent benchmarking of SDP and SOCP solvers. *Math. Program.*, 95(2, Ser. B):407–430, 2003. Computational semidefinite and second order cone programming: the state of the art. 20

[13] R.D.C. Monteiro and M.J. Todd. Path-following methods. In *Handbook of Semidefinite Programming*, pages 267–306. Kluwer Acad. Publ., Boston, MA, 2000. 3

[14] H. Wei and H. Wolkowicz. Generating and solving hard instances in semidefinite programming. *Math. Programming*, 125(1):31–45, 2010. 23

[15] H. Wolkowicz. Solving semidefinite programs using preconditioned conjugate gradients. *Optim. Methods Softw.*, 19(6):653–672, 2004. 3, 10

Index