

**DOMAIN-DRIVEN SOLVER (DDS):
A MATLAB-BASED SOFTWARE PACKAGE FOR CONVEX OPTIMIZATION
PROBLEMS IN DOMAIN-DRIVEN FORM**

MEHDI KARIMI AND LEVENT TUNÇEL

ABSTRACT. Domain-Driven Solver (DDS) is a MATLAB-based software package for convex optimization problems in Domain-Driven form [11]. The current version of DDS accepts every combination of the following function/set constraints: (1) symmetric cones (LP, SOCP, and SDP); (2) quadratic constraints; (3) direct sums of an arbitrary collection of 2-dimensional convex sets defined as the epigraphs of univariate convex functions (including as special cases, geometric programming and entropy programming); (4) epigraph of a matrix norm (including as a special case, minimization of nuclear norm over a linear subspace); (5) epigraph of quantum entropy; and (6) constraints involving Hyperbolic polynomials. DDS is a practical implementation of the infeasible-start primal-dual algorithm designed and analyzed in [11]. This manuscript contains the installation method of DDS and the input format for different types of constraints. To help the users in using DDS, we include some examples to illustrate the coding. We also discuss some implementation details and techniques we used to improve the efficiency.

Date: August 27, 2019.

Mehdi Karimi (m7karimi@uwaterloo.ca) and Levent Tunçel (ltuncel@math.uwaterloo.ca): Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada.
Research of the authors was supported in part by Discovery Grants from NSERC and by U.S. Office of Naval Research under award numbers: N00014-12-1-0049, N00014-15-1-2171 and N00014-18-1-2078.

CONTENTS

1. Introduction	4
1.1. Installation	4
2. How to use the DDS code	5
3. Adding functions and set constraints	6
3.1. Linear programming (LP) and second-order cone programming (SOCP)	7
3.2. Semidefinite programming (SDP)	8
3.3. Quadratic constraints	10
3.4. Constraints involving epigraphs of matrix norms	11
4. Constraints involving epigraphs of univariate functions (geometric, entropy, and p -norm programming)	14
4.1. Constraints involving power functions	18
5. Matrix monotonicity, quantum entropy	18
5.1. Adding quantum entropy based constraints	19
6. Constraints involving hyperbolic polynomials	20
6.1. Different formats for inputting multivariate polynomials	21
6.2. Adding constraints involving hyperbolic polynomials	22
7. Primal-heavy version of the algorithm	23
8. More numerical examples	24
8.1. LP-SOCP-SDP	25
8.2. LP-SOCP-SDP constraints combined with geometric and entropy ones	26
References	26
Appendix A. Calculating the predictor and corrector steps	27
Appendix B. Legendre-Fenchel conjugates of univariate convex functions	29

Appendix C. Implementation details for SDP and generalized epigraphs of matrix norms	31
C.1. SDP	31
C.2. Generalized epigraphs of matrix norms	33
Appendix D. Implementation of quantum entropy function and related constraints	35
Appendix E. Calculating Legendre-Fenchel conjugate and derivatives	37
E.1. Constraints defined by the 2-dimensional convex sets	38
Appendix F. Format of the input for some other solvers	43
F.1. MOSEK [15]	43
F.2. SDPT3 [23, 22]	44
F.3. SeDuMi [21]	44
F.4. CVX [8]	45

1. INTRODUCTION

The code DDS (Domain-Driven Solver) solves convex optimization problems of the form

$$(1) \quad \inf_x \{ \langle c, x \rangle : Ax \in D \},$$

where $x \mapsto Ax : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is a linear embedding, A and $c \in \mathbb{R}^n$ are given, and $D \subset \mathbb{R}^m$ is a closed convex set defined as the closure of the domain of a ϑ -self-concordant (s.c.) barrier Φ [18, 17]. In practice, the set D is typically formulated as $D = D_1 \oplus \cdots \oplus D_\ell$, where D_i is associated with a s.c. barrier Φ_i , for $i \in \{1, \dots, \ell\}$. Every input constraint for DDS may be thought of as either the convex set it defines or the corresponding s.c. barrier.

The current version of DDS accepts many functions and set constraints as we explain in this article. If a user has a nonlinear convex objective function $f(x)$ to minimize, one can introduce a new variable x_{n+1} and minimize a linear function x_{n+1} subject to the convex set constraint $f(x) \leq x_{n+1}$ (and other convex constraints in the original optimization problem). As a result, in this article we will talk about representing functions and convex set constraints interchangeably. The algorithm underlying the code also uses the Legendre-Fenchel (LF) conjugate Φ_* of Φ if it is computationally efficient. For the computation of LF conjugate, see Appendix E. Any new discovery of a s.c. barrier allows DDS to expand the classes of convex optimization problems it can solve as any new s.c. barrier Φ with a computable LF conjugate can be easily added to the code. DDS is a practical implementation of the primal-dual algorithm designed and analyzed in [11], which has the current best iteration complexity bound available for conic formulations. Stopping criteria for DDS and the way DDS suggests the status (“has an approximately optimal solution”, “is infeasible”, “is unbounded”, etc.) is based on the analyses in [12].

The algorithm used in DDS is an infeasible-start primal-dual path-following algorithm, and is of predictor corrector type [11]. The LHS matrix of the linear systems of equations determining the predictor and corrector steps have a similar form. In Appendix A, we explain how such linear systems are being solved for DDS.

1.1. Installation. The current version of DDS is written in MATLAB. This version is available from the website:

<http://www.math.uwaterloo.ca/~m7karimi/DDS.html>

To use DDS, the user can follow these steps:

- unzip DDS.zip;
- run MATLAB in the directory DDS;
- run the m-file DDS_startup.m.

2. HOW TO USE THE DDS CODE

In this section, we explain the format of the input for many popular classes of optimization problems. In practice, we typically have $D = \bar{D} - b$, where $\text{int}\bar{D}$ is the domain of a “canonical” s.c. barrier and $b \in \mathbb{R}^m$. For example, for LP, we typically have $D = \mathbb{R}_+^n + b$, where $b \in \mathbb{R}^m$ is given as part of the input data, and $-\sum_{i=1}^n \ln(x_i)$ is a s.c. barrier for \mathbb{R}_+^n . The command in MATLAB that calls DDS is

```
[x,y,info]=DDS(c,A,b,cons,OPTIONS);
```

Input Arguments:

cons: A cell array that contains the information about the type of constraints.

c,A,b: Input data for DDS: A is the coefficient matrix, c is the objective vector, b is the RHS vector (i.e., the shift in the definition of the convex domain D).

OPTIONS (optional): An array which contains information about the tolerance or initial points.

Output Arguments:

x,y: primal and dual points, respectively.

info: a structure array containing performance information such as `info.time`, which returns the CPU time for solving the problem.

Note that in the Domain-Driven setup, the primal problem is the main problem, and the dual problem is implicit for the user. This implicit dual problem is:

$$(2) \quad \inf_y \{ \delta_*(y|D) : A^\top y = -c, y \in D_* \},$$

where $\delta_*(y|D) := \sup\{\langle y, z \rangle : z \in D\}$, is the *support function* of D , and D_* is defined as

$$(3) \quad D_* := \{y : \langle y, h \rangle \leq 0, \quad \forall h \in \text{rec}(D)\},$$

where $\text{rec}(D)$ is the *recession cone* of D . For a primal feasible point $x \in \mathbb{R}^n$ which satisfies $Ax \in D$ and a dual feasible point $y \in D_*$, the duality gap is defined in [11] as

$$(4) \quad \langle c, x \rangle + \delta_*(y|D).$$

It is proved in [11] that the duality gap is well-defined and zero duality gap implies optimality. If DDS returns status “solved” for a problem (`info.status=1`), it means (x, y) is a pair of approximately feasible primal and dual points, with duality gap close to zero (based on tolerance). If `info.status=2`, the problem is suspected to be unbounded and the returned x is a point, approximately primal feasible with very small objective value ($\langle c, x \rangle \leq -1/\text{tol}$). If `info.status=3`, problem is suspected to be infeasible, and the returned y in D_* approximately satisfies $A^\top y = 0$ with $\delta_*(y|D) < 0$. If `info.status=4`, problem is suspected to be ill-conditioned.

The user is not required to input any part of the `OPTIONS` array. The default settings are:

- $tol = 10^{-8}$.
- The initial points x^0 and z^0 for the infeasible-start algorithm are chosen such that, assuming $D = D_1 \oplus \dots \oplus D_\ell$, the i th part of $Ax^0 + z^0$ is a canonical point in $\text{int}D_i$.

However, if a user chooses to provide `OPTIONS` as an input, here is how to define the desired parts: `OPTIONS.tol` may be given as the desired tolerance, otherwise the default $tol := 10^{-8}$ is used. `OPTIONS.x0` and `OPTIONS.z0` may be defined as the initial points as any pair of points $x^0 \in \mathbb{R}^n$ and $z^0 \in \mathbb{R}^m$ that satisfy $Ax^0 + z^0 \in \text{int}D$. If only `OPTIONS.x0` is given, then x^0 must satisfy $Ax^0 \in \text{int}D$. In other words, `OPTIONS.x0` is a point that strictly satisfies all the constraints.

In the following, we discuss the format of each input function/set constraint. Table 1 shows the classes of function/set constraints the current version of DDS accepts, plus the abbreviation we use to represent the constraint.

TABLE 1. Function/set constraints the current version of DDS accepts, and their abbreviations.

function/set constraint	abbreviation
LP	LP
SOCP	SOCP
Rotated SOCP	SOCPR
SDP	SDP
Quadratic Constraints	QC
Epigraph of a Matrix Norm	MN
Direct sum of 2-dimensional sets (geometric, entropy, and p -norm programming)	TD
Quantum Entropy	QE
Hyperbolic Polynomials	HB

3. ADDING FUNCTIONS AND SET CONSTRAINTS

From now on, we assume that the objective function is “ $\inf \langle c, x \rangle$ ”, and we show how to add various function/set constraints. Note that `A`, `b`, and `cons` are cell arrays in MATLAB.

$\text{cons}(\mathbf{k}, 1)$ represents type of the k th block of constraints by using the abbreviations of Table 1. For example, $\text{cons}(2, 1) = \text{'LP'}$ means that the second block of constraints are linear inequalities. It is advisable to group the constraints of the same type in one block, but not necessary.

3.1. Linear programming (LP) and second-order cone programming (SOCP). Suppose we want to add ℓ LP constraints of the form

$$(5) \quad A_L^i x + b_L^i \geq 0, \quad i \in \{1, \dots, \ell\},$$

where A_L^i is an m_L^i -by- n matrix, as the k th block of constraints. Then, we define

$$(6) \quad \mathbf{A}\{\mathbf{k}, 1\} = \begin{bmatrix} A_L^1 \\ \vdots \\ A_L^\ell \end{bmatrix}, \quad \mathbf{b}\{\mathbf{k}, 1\} = \begin{bmatrix} b_L^1 \\ \vdots \\ b_L^\ell \end{bmatrix}$$

$$\text{cons}\{\mathbf{k}, 1\} = \text{'LP'}, \quad \text{cons}\{\mathbf{k}, 2\} = [m_L^1, \dots, m_L^\ell].$$

Similarly to add ℓ SOCP constraints of the form

$$(7) \quad \|A_S^i x + b_S^i\| \leq (g_S^i)^\top x + d_S^i, \quad i \in \{1, \dots, \ell\},$$

where A_S^i is an m_S^i -by- n matrix for $i \in \{1, \dots, \ell\}$, as the k th block, we define

$$(8) \quad \mathbf{A}\{\mathbf{k}, 1\} = \begin{bmatrix} (g_S^1)^\top \\ A_S^1 \\ \vdots \\ (g_S^\ell)^\top \\ A_S^\ell \end{bmatrix}, \quad \mathbf{b}\{\mathbf{k}, 1\} = \begin{bmatrix} d_S^1 \\ b_S^1 \\ \vdots \\ d_S^\ell \\ b_S^\ell \end{bmatrix}$$

$$\text{cons}\{\mathbf{k}, 1\} = \text{'SOCP'}, \quad \text{cons}\{\mathbf{k}, 2\} = [m_S^1, \dots, m_S^\ell].$$

Let us see an example:

Example 3.1. Suppose we are given the problem:

$$(9) \quad \begin{aligned} \min \quad & c^\top x \\ \text{s.t.} \quad & [-2, 1]x \leq 1, \\ & \left\| \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix} x + \begin{bmatrix} 3 \\ 4 \end{bmatrix} \right\| \leq 2. \end{aligned}$$

Then we define

$$\begin{aligned} \text{cons}\{1,1\}=\text{'LP'}, \quad \text{cons}\{1,2\}=[1], \quad A\{1,1\} &= \begin{bmatrix} 2 & -1 \end{bmatrix}, \quad \mathbf{b}\{1,1\} = \begin{bmatrix} 1 \end{bmatrix}, \\ \text{cons}\{2,1\}=\text{'SOCP'}, \quad \text{cons}\{2,2\}=[2], \quad A\{2,1\} &= \begin{bmatrix} 0 & 0 \\ 2 & 1 \\ 1 & 3 \end{bmatrix}, \quad \mathbf{b}\{2,2\} = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}. \end{aligned}$$

The s.c. functions being used in DDS for these constraints are $-\ln(z)$ for \mathbb{R}_+ and $-\ln(t^2 - z^\top z)$ for $\{(z, t) \in \mathbb{R}^n \oplus \mathbb{R} : \|z\| \leq t\}$.

DDS also accepts constraints defined by the rotated second order cones:

$$(10) \quad \{(z, t, s) \in \mathbb{R}^n \oplus \mathbb{R} \oplus \mathbb{R} : \|z\|^2 \leq ts, t \geq 0, s \geq 0\},$$

which is handled by the s.c. barrier $-\ln(ts - z^\top z)$. The abbreviation we use is 'SOCPR'. To add ℓ rotated SOCP constraints of the form

$$(11) \quad \begin{aligned} \|A_S^i x + b_S^i\|^2 &\leq ((g_S^i)^\top x + d_S^i)((\bar{g}_S^i)^\top x + \bar{d}_S^i), \quad i \in \{1, \dots, \ell\}, \\ (g_S^i)^\top x + d_S^i &\geq 0, \quad (\bar{g}_S^i)^\top x + \bar{d}_S^i \geq 0, \end{aligned}$$

where A_S^i is an m_S^i -by- n matrix for $i \in \{1, \dots, \ell\}$, as the k th block, we define

$$(12) \quad \mathbf{A}\{\mathbf{k}, 1\} = \begin{bmatrix} (g_S^1)^\top \\ (\bar{g}_S^1)^\top \\ A_S^1 \\ \vdots \\ (g_S^\ell)^\top \\ (\bar{g}_S^\ell)^\top \\ A_S^\ell \end{bmatrix}, \quad \mathbf{b}\{\mathbf{k}, 1\} = \begin{bmatrix} d_S^1 \\ \bar{d}_S^1 \\ b_S^1 \\ \vdots \\ d_S^\ell \\ \bar{d}_S^\ell \\ b_S^\ell \end{bmatrix}$$

$$\text{cons}\{\mathbf{k}, 1\}=\text{'SOCPR'}, \quad \text{cons}\{\mathbf{k}, 2\} = [m_S^1, \dots, m_S^\ell].$$

3.2. Semidefinite programming (SDP). Consider ℓ SDP constraints in standard inequality (linear matrix inequality (LMI)) form:

$$(13) \quad F_0^i + x_1 F_1^i + \dots + x_n F_n^i \succeq 0, \quad i \in \{1, \dots, \ell\}.$$

F_j^i 's are n_i -by- n_i symmetric matrices. The above optimization problem is in the matrix form. To formulate it in our setup, we need to write it in the vector form. DDS has two internal functions `sm2vec` and `vec2sm`. `sm2vec` takes an n -by- n symmetric matrix and changes it into a vector in

\mathbb{R}^{n^2} by stacking the columns of it on top of one another in order. `vec2sm` changes a vector into a symmetric matrix such that

$$(14) \quad \text{vec2sm}(\text{sm2vec}(X))=X.$$

By this definition, it is easy to check that for any pair of n -by- n symmetric matrices X and Y we have

$$(15) \quad \langle X, Y \rangle = \text{sm2vec}(X)^\top \text{sm2vec}(Y).$$

To give (13) to DDS as the k th input block, we define:

$$(16) \quad \begin{aligned} A\{k, 1\} &:= \begin{bmatrix} \text{sm2vec}(F_1^1), \dots, \text{sm2vec}(F_n^1) \\ \vdots \\ \text{sm2vec}(F_1^\ell), \dots, \text{sm2vec}(F_n^\ell) \end{bmatrix}, & b\{k, 1\} &:= \begin{bmatrix} \text{sm2vec}(F_0^1) \\ \vdots \\ \text{sm2vec}(F_0^\ell) \end{bmatrix}, \\ \text{cons}\{k, 1\} &= \text{'SDP'} & \text{cons}\{k, 2\} &= [n^1, \dots, n^\ell]. \end{aligned}$$

The s.c. barrier used in DDS for SDP is the well-known function $-\ln(\det(X))$ defined on the convex cone of symmetric positive definite matrices.

Example 3.2. Assume that we want to find scalars x_1 , x_2 , and x_3 such that $x_1 + x_2 + x_3 \geq 1$ and the maximum eigenvalue of $A_0 + x_1A_1 + x_2A_2 + x_3A_3$ is minimized, where

$$A_0 = \begin{bmatrix} 2 & -0.5 & -0.6 \\ -0.5 & 2 & 0.4 \\ -0.6 & 0.4 & 3 \end{bmatrix}, \quad A_1 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad A_2 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \quad A_3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

We can write this problem as

$$(17) \quad \begin{aligned} \min \quad & t \\ \text{s.t.} \quad & -1 + x_1 + x_2 + x_3 \geq 0, \\ & tI - (A_0 + x_1A_1 + x_2A_2 + x_3A_3) \succeq 0. \end{aligned}$$

To solve this problem, we define:

$$\begin{aligned}
\text{cons}\{1,1\} &= \text{'LP'}, & \text{cons}\{1,2\} &= [1], & \text{cons}\{2,1\} &= \text{'SDP'}, & \text{cons}\{2,2\} &= [3], \\
A\{1,1\} &= \begin{bmatrix} 1 & 1 & 1 & 0 \end{bmatrix}, & b\{1,1\} &= \begin{bmatrix} -1 \end{bmatrix}, \\
A\{2,1\} &= \begin{bmatrix} 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, & b\{2,1\} &= \begin{bmatrix} -2 \\ 0.5 \\ 0.6 \\ 0.5 \\ -2 \\ -0.4 \\ 0.6 \\ -0.4 \\ -3 \end{bmatrix}, \\
& & c &= (0, 0, 0, 1)^\top.
\end{aligned}$$

Then $\text{DDS}(c, A, b, \text{cons})$ gives the answer $x = (1.1265, 0.6, -0.4, 3)$, which means the minimum largest eigenvalue is 3.

3.3. Quadratic constraints. Assume that we want to add the following constraints to DDS:

$$(18) \quad x^\top A_i^\top Q_i A_i x + b_i^\top x + d_i \leq 0, \quad i = \{1, \dots, \ell\},$$

where each A_i is m_i -by- n with rank n , and $Q_i \in \mathbb{S}^{m_i}$. In general, this type of constraints may be non-convex and difficult to handle. Currently, DDS handles two cases:

- Q_i is positive semidefinite,
- Q_i has exactly one negative eigenvalue. In this case, DDS considers the intersection of the set of points satisfying (18) and a shifted *hyperbolicity cone* defined by the quadratic inequality $y^\top Q_i y \leq 0$.

To give constraints in (18) as input to DDS as the k th block, we define

$$(19) \quad \begin{aligned} \mathbf{A}\{\mathbf{k}, 1\} &= \begin{bmatrix} b_1^\top \\ A_1 \\ \vdots \\ b_\ell^\top \\ A_\ell \end{bmatrix}, \quad \mathbf{b}\{\mathbf{k}, 1\} = \begin{bmatrix} d_1 \\ 0 \\ \vdots \\ d_\ell \\ 0 \end{bmatrix} \\ \text{cons}\{\mathbf{k}, 1\} &= \text{'QC'}, \quad \text{cons}\{\mathbf{k}, 2\} = [m_1, \dots, m_\ell], \\ \text{cons}\{\mathbf{k}, 3, i\} &= Q_i, \quad i \in \{1, \dots, \ell\}. \end{aligned}$$

If $\text{cons}\{\mathbf{k}, 3\}$ is not given as the input, DDS takes all Q_i 's to be identity matrices.

If Q_i is positive semidefinite, then the corresponding constraint in (18) can be written as

$$(20) \quad \begin{aligned} u^\top u + w + d &\leq 0 \\ u &:= R_i A_i x, \quad w := b_i^\top x, \quad d := d_i, \end{aligned}$$

where $Q_i = R_i^\top R_i$ is a Cholesky factorization of Q_i . We associate the following s.c. barrier and its LF conjugate to such quadratic constraints:

$$(21) \quad \begin{aligned} \Phi(u, w) &= -\ln(-(u^\top u + w + d)), \\ \Phi_*(y, \eta) &= \frac{y^\top y}{4\eta} - 1 - d\eta - \ln(\eta). \end{aligned}$$

If Q_i has exactly one negative eigenvalue with eigenvector v , then $-y^\top Q_i y$ is a hyperbolic polynomial with respect to v . The hyperbolicity cone is the connected component of $y^\top Q_i y \leq 0$ which contains v and $-\ln(-y^\top Q_i y)$ is a s.c. barrier for this cone.

If for any of the inequalities in (18), Q_i has exactly one negative eigenvalue while $b_i = 0$ and $d_i = 0$, DDS considers the hyperbolicity cone defined by the inequality as the set constraint.

3.4. Constraints involving epigraphs of matrix norms. Assume that we have constraints of the form

$$(22) \quad \begin{aligned} X - UU^\top &\succeq 0, \\ X &= A_0 + \sum_{i=1}^{\ell} x_i A_i, \\ U &= B_0 + \sum_{i=1}^{\ell} x_i B_i, \end{aligned}$$

where A_i , $i \in \{0, \dots, \ell\}$, are m -by- m symmetric matrices, and B_i , $i \in \{0, \dots, \ell\}$, are m -by- n matrices. The set $\{(Z, U) \in \mathbb{S}^m \oplus \mathbb{R}^{m \times n} : Z - UU^\top \succeq 0\}$ is handled by the following s.c. barrier:

$$(23) \quad \Phi(Z, U) := -\ln(\det(Z - UU^\top)),$$

with LF conjugate

$$(24) \quad \Phi_*(Y, V) = -m - \frac{1}{4}\text{Tr}(V^\top Y^{-1}V) - \ln(\det(-Y)),$$

where $Y \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{m \times n}$ [16]. This constraint can be reformulated as an SDP constraint using a Schur complement. However, $\Phi(Z, U)$ is a m -s.c. barrier while the size of SDP reformulation is $m + n$. For the cases that $m \ll n$, using the Domain-Driven form may be advantageous. We also mentioned that a special but very important application is minimizing the *nuclear norm* of the matrix. The nuclear norm of a matrix Z is $\|Z\|_* := \text{Tr}((ZZ^\top)^{1/2})$. The dual norm of $\|\cdot\|_*$ is the operator 2-norm $\|\cdot\|$ of a matrix. The following optimization problems are a primal-dual pair [20].

$$(25) \quad \begin{array}{ll} (P_N) & \min_X \quad \|X\|_* \\ & \text{s.t.} \quad A(X) = b. \end{array} \quad \begin{array}{ll} (D_N) & \max_z \quad \langle b, z \rangle \\ & \text{s.t.} \quad \|A^*(z)\| \leq 1, \end{array}$$

where A is a linear transformation on matrices and A^* is its adjoint. (P_N) is a very popular relaxation of the problem of minimizing $\text{rank}(X)$ subject to $A(X) = b$, with applications in machine learning and compressed sensing. The dual problem (D_N) is a special case of (22) where $Z = I$ and $U = A^*(z)$. As we will show on an example, solving (D_N) by `[x,y]=DDS(c,A,b,Z)` leads us to y , which gives a solution for (P_N) .

DDS has two internal functions `m2vec` and `vec2m` for converting matrices (not necessarily symmetric) to vectors and vice versa. The abbreviation we use for epigraph of a matrix norm is MN. If the k th input block is of this type, `cons{k,2}` is a ℓ -by-2 matrix, where ℓ is the number of constraints of this type, and each row is of the form $[m \ n]$. For each constraint of the form (22), the corresponding parts in A and b are defined as

$$(26) \quad A\{k,1\} = \begin{bmatrix} \text{m2vec}(B_1, n) & \cdots & \text{m2vec}(B_\ell, n) \\ \text{sm2vec}(A_1) & \cdots & \text{sm2vec}(A_\ell) \end{bmatrix}, \quad b\{k,1\} = \begin{bmatrix} \text{m2vec}(B_0, n) \\ \text{sm2vec}(A_0) \end{bmatrix}.$$

Example 3.3. *Assume that we have matrices*

$$(27) \quad U_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}, \quad U_1 = \begin{bmatrix} -1 & -1 & 1 \\ 0 & 0 & 1 \end{bmatrix}, \quad U_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix},$$

and our goal is to solve

$$(28) \quad \begin{aligned} & \min t \\ & \text{s.t. } UU^\top \preceq tI, \\ & U = U_0 + x_1U_1 + x_2U_2. \end{aligned}$$

Then the input to DDS is defined as

$$\begin{aligned} & \text{cons}\{1,1\} = 'MN', \quad \text{cons}\{2,1\} = [2 \ 3], \\ & A\{1,1\} = \begin{bmatrix} m2\text{vec}(U_1,3) & m2\text{vec}(U_2,3) & \text{zeros}(6,1) \\ \text{zeros}(4,1) & \text{zeros}(4,1) & sm2\text{vec}(I_{2 \times 2}) \end{bmatrix}, \quad b\{1,1\} = \begin{bmatrix} m2\text{vec}(U_0,3) \\ \text{zeros}(4,1) \end{bmatrix}, \\ & c = (0,0,1). \end{aligned}$$

CVX does not accept a constraint of the form $X - UU^\top \succeq 0$ and we need to give an SDP representation. By doing that, both codes give the solution 0.407105. If we change c to $c = (0,0,-1)$, the problem is unbounded and DDS returns unboundedness certificate.

Example 3.4. We consider minimizing the nuclear norm over a subspace. Consider the following optimization problem:

$$(29) \quad \begin{aligned} & \min \|X\|_* \\ & \text{s.t. } \text{Tr}(U_1X) = 1 \\ & \text{Tr}(U_2X) = 2, \end{aligned}$$

where

$$(30) \quad U_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \quad U_2 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

By using (25), the dual of this problem is

$$(31) \quad \begin{aligned} & \min -u_1 - 2u_2 \\ & \text{s.t. } \|u_1U_1 + u_2U_2\| \leq 1. \end{aligned}$$

To solve this problem with our code, we define

$$\begin{aligned} & \text{cons}\{1,1\} = 'MN', \quad \text{cons}\{1,2\} = [2 \ 4], \\ & A\{1,1\} = \begin{bmatrix} m2\text{vec}(U_1,4) & m2\text{vec}(U_2,4) \\ \text{zeros}(4,1) & \text{zeros}(4,1) \end{bmatrix}, \quad b\{1,1\} = \begin{bmatrix} \text{zeros}(8,1) \\ sm2\text{vec}(I_{2 \times 2}) \end{bmatrix}, \\ & c = (-1, -2), \end{aligned}$$

If we solve the problem using $[\mathbf{x}, \mathbf{y}] = \text{DDS}(\mathbf{c}, \mathbf{A}, \mathbf{b}, \text{cons})$, the optimal value is -2.2360 . Now the dual solution \mathbf{y} is $(\text{m2vec}(V^\top, 4), \text{sm2vec}(Y))$ and V is the solution of (29) with objective value 2.2360. We have

$$(32) \quad X^* := V = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

What we did in the last example can be done in general. For the optimization problem

$$(33) \quad \begin{aligned} \min \quad & \|X\|_* \\ \text{s.t.} \quad & \text{Tr}(U_i X) = c_i, \quad i \in \{1, \dots, \ell\}, \end{aligned}$$

where X is n -by- m , we solve the dual problem by defining

$$(34) \quad \begin{aligned} \mathbf{A}\{1,1\} &= \begin{bmatrix} \text{m2vec}(U_1, n) & \cdots & \text{m2vec}(U_\ell, n) \\ \text{zeros}(m^2, 1) & \cdots & \text{zeros}(m^2, 1) \end{bmatrix}, \quad \mathbf{b}\{1,1\} = \begin{bmatrix} \text{zeros}(mn, 1) \\ \text{sm2vec}(I_{m \times m}) \end{bmatrix}, \\ \text{cons}\{1,1\} &= 'MN', \quad \text{cons}\{1,2\} = [m \quad n]. \end{aligned}$$

Then, if we run $[\mathbf{x}, \mathbf{y}] = \text{DDS}(\mathbf{c}, \mathbf{A}, \mathbf{b}, \text{cons})$ and define $V := (\text{vec2m}(\mathbf{y}(1:\mathbf{m}*\mathbf{n}), \mathbf{m}))^\top$, then V is an optimal solution for (33).

For implementation details involving epigraph of matrix norms, see Appendix C.2.

4. CONSTRAINTS INVOLVING EPIGRAPHS OF UNIVARIATE FUNCTIONS (GEOMETRIC, ENTROPY, AND p -NORM PROGRAMMING)

DDS accepts constraints of the form

$$(35) \quad \sum_{i=1}^{\ell} \alpha_i f_i(a_i^\top x + \beta_i) + g^\top x + \gamma \leq 0, \quad a_i, g \in \mathbb{R}^n, \quad \beta_i, \gamma \in \mathbb{R}, \quad i \in \{1, \dots, \ell\},$$

where $\alpha_i \geq 0$ and $f_i(x)$, $i \in \{1, \dots, \ell\}$, can be any function from Table 2. Note that every univariate convex function can be added to this table in the same fashion. By using this simple structure, we can model many interesting optimization problems. Geometric programming (GP) [1] and entropy programming (EP) [6] with many applications in engineering are constructed with constraints of the form (35) when $f_i(z) = e^z$ for $i \in \{1, \dots, \ell\}$ and $f_i(z) = z \ln(z)$ for $i \in \{1, \dots, \ell\}$, respectively. The other functions with p powers let us solve optimization problems related to p -norm minimization. The corresponding s.c. barriers are shown in Table 2, which are the functions being used in DDS. There is a closed form expression for the LF conjugate of

TABLE 2. Some 2-dimensional convex sets and their s.c. barriers.

	set (z, t)	s.c. barrier $\Phi(z, t)$
1	$-\ln(z) \leq t, z > 0$	$-\ln(t + \ln(z)) - \ln(z)$
2	$e^z \leq t$	$-\ln(\ln(t) - z) - \ln(t)$
3	$z \ln(z) \leq t, z > 0$	$-\ln(t - z \ln(z)) - \ln(z)$
4	$ z ^p \leq t, p \geq 1$	$-\ln(t^{\frac{2}{p}} - z^2) - 2 \ln(t)$
5	$-z^p \leq t, z > 0, 0 \leq p \leq 1$	$-\ln(z^p + t) - \ln(z)$
6	$\frac{1}{z} \leq t, z > 0$	$-\ln(zt - 1)$

the first two functions. For the last four, the LF conjugate can be calculated to high accuracy efficiently. In Appendix B, we show how to calculate the LF conjugates for the functions in Table 2 and the internal functions we have in DDS.

To represent a constraint of the form (35), for given $\gamma \in \mathbb{R}$ and $\beta_i \in \mathbb{R}, i \in \{1, \dots, \ell\}$, we can define the corresponding convex set D as

$$(36) \quad D := \{(w, s_i, u_i) : w + \gamma \leq 0, f_i(s_i + \beta_i) \leq u_i, \forall i\},$$

and our matrix A represents $w = \sum_{i=1}^{\ell} \alpha_i u_i + g^\top x$ and $s_i = a_i^\top x, i \in \{1, \dots, \ell\}$. As can be seen, to show our set as above, we need to add some artificial variables u_i 's to our formulations. DDS code does it internally and we do not need to insert them. Let us assume that we want to add the following s constraints to our code

$$(37) \quad \sum_{type} \sum_{i=1}^{\ell_{type}^j} -\alpha_i^{j,type} f_{type}((a_i^{j,type})^\top x + \beta_i^{j,type}) + g_j^\top x + \gamma_j \leq 0, \quad j \in \{1, \dots, s\}.$$

From now on, *type* indexes the rows of Table 2. The abbreviation we use for these constraints is TD. Hence, if the k th input block are the constraints in (37), then we have $\mathbf{cons}\{\mathbf{k}, 1\} = \text{'TD'}$. $\mathbf{cons}\{\mathbf{k}, 2\}$ is a matrix with 3 columns. In each row, the first entry is the index of constraint, the second entry is the type, and the third entry is the number of functions of that type we have in that constraint. Let us say that in the j th constraint, we have l_2^j functions of type 2 and l_3^j functions of type 3, then the corresponding columns in $\mathbf{cons}\{\mathbf{k}, 2\}$ are as follows

$$(38) \quad \mathbf{cons}\{\mathbf{k}, 2\} = \begin{bmatrix} \vdots \\ j & 2 & l_2^j \\ j & 3 & l_3^j \\ \vdots \end{bmatrix}.$$

Note: The indices of constraints must:

- start from 1,
- all the rows corresponding to each constraint must be consecutive,
- indices must be consecutive and in ascending order.

The types can be in any order, but they must match with the rows of A and b .

We also add $\text{cons}\{\mathbf{k}, 3\}$ which is a row or column vector that contains all the coefficients in each constraint. Note that the coefficients must be in the same order as their corresponding rows in A . If in the first constraint we have 2 functions of type 2 and 1 function of type 3, it starts as

$$(39) \quad \text{cons}\{\mathbf{k}, 3\} = [\alpha_1^{1,2}, \alpha_2^{1,2}, \alpha_1^{1,3}, \dots].$$

To add the rows to A , for each constraint j , we first add g_j , then $a_i^{j,type}$'s in the order that matches $\text{cons}\{\mathbf{k}, 2\}$ and $\text{cons}\{\mathbf{k}, 3\}$. We do the same thing for vector b (first γ_j , then $\beta_i^{j,type}$'s). The part of A and b corresponding to the j th constraint is as follows if we have for example five types

$$(40) \quad \mathbf{A} = \begin{bmatrix} g_j^\top \\ a_1^{j,1} \\ \vdots \\ a_1^{j,1} \\ \vdots \\ a_1^{j,5} \\ \vdots \\ a_1^{j,5} \\ \vdots \\ a_1^{j,5} \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} \gamma_j \\ \beta_1^{j,1} \\ \vdots \\ \beta_1^{j,1} \\ \vdots \\ \beta_1^{j,5} \\ \vdots \\ \beta_1^{j,5} \\ \vdots \\ \beta_1^{j,5} \end{bmatrix}.$$

Let us see an example:

Example 4.1. Assume that we want to solve

$$(41) \quad \begin{aligned} \min \quad & c^\top x \\ \text{s.t.} \quad & -\ln(x_2 + 2x_3 + 55) + 2e^{x_1+x_2+1} + x_1 - 2 \leq 0, \\ & -3\ln(x_1 + 2x_2 + 3x_3 - 30) + e^{-x_3-3} - x_3 + 1 \leq 0, \\ & x \geq 0. \end{aligned}$$

For this problem, we define:

$$\begin{aligned} \text{cons}\{1,1\} &= \text{'LP'}, & \text{cons}\{1,2\} &= [3], \\ \text{cons}\{2,1\} &= \text{'TD'}, & \text{cons}\{2,2\} &= \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 2 & 1 & 1 \\ 2 & 2 & 1 \end{bmatrix}, & \text{cons}\{2,3\} &= [1 \ 2 \ 3 \ 1], \\ A\{1,1\} &= \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}, & b\{1,1\} &= \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \\ A\{2,1\} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 1 & 1 & 0 \\ 0 & 0 & -1 \\ 1 & 2 & 3 \\ 0 & 0 & -1 \end{bmatrix}, & b\{2,1\} &= \begin{bmatrix} -2 \\ 55 \\ 1 \\ 1 \\ -30 \\ -3 \end{bmatrix}. \end{aligned}$$

The first three rows of A and b are for linear constraints. CVX uses successive approximation method for these kinds of problems. When $c = (1, 1, 1)$, both codes return the correct answer with objective value of 10.0165 (See Appendix F for an introduction to input formats for CVX and some other solvers). When we put $c = (1, 1, -1)$, the problem is unbounded. CVX does not return a meaningful solution, but DDS returns $(0, 0, 1.23 \times 10^6)$ as a certificate of unboundedness.

Let us add a function $z\ln(z)$ (type 3) to the first constraint and change sign constraints:

$$\begin{aligned} \min \quad & c^\top x \\ \text{s.t.} \quad & -\ln(x_2 + 2x_3 + 55) + 2e^{x_1 + x_2 + 1} + (x_2 - 3x_3)\ln(x_2 - 3x_3) + x_1 - 2 \leq 0, \\ & -3\ln(x_1 + 2x_2 + 3x_3 - 30) + e^{-x_3 - 3} - x_3 + 1 \leq 0, \\ (42) \quad & x_2, x_3 \geq 0, \quad x_1 \leq 0. \end{aligned}$$

For CVX to recognize it as a convex optimization problem, we use the entropy function $\text{entr}(z) = -z\ln(z)$ from its library. For $c = (0, 1, 1)$, both codes return $x = (-13.2167, 14.4958, 4.8322)$ as the optimal solution. If we change $c = (0, 1, 1)$, the problem becomes unbounded. DDS returns $10^8 \times (-4.9050, 1.6350, 0.5450)$ as a certificate of unboundedness, but CVX does not return a meaningful solution.

4.1. Constraints involving power functions. The difference between these two types and the others is that we also need to give the value of p for each function. To do that, we need to add $\text{cons}\{\mathbf{k}, 4\}$. $\text{cons}\{\mathbf{k}, 4\}$ has the same length as $\text{cons}\{\mathbf{k}, 3\}$ and it has zero for functions of types 1 to 3 and also 6. For functions of type 4 and 5, we put the power p in exactly the same place we put the coefficient of the function in $\text{cons}\{\mathbf{k}, 3\}$. Let us see an example:

Example 4.2.

$$\begin{aligned} \min \quad & c^\top x \\ \text{s.t.} \quad & 2\exp(2x_1 + 3) + |x_1 + x_2 + x_3|^2 + 4.5|x_1 + x_2|^{2.5} + |x_2 + 2x_3|^3 + x_1 - 2 \leq 0. \end{aligned}$$

For this problem, we define:

$$\begin{aligned} A\{1, 1\} &= \begin{bmatrix} 1 & 0 & 0 \\ 2 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 2 \end{bmatrix}, \quad b\{1, 1\} = \begin{bmatrix} -2 \\ 3 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \\ \text{cons}\{1, 1\} &= 'TD', \quad \text{cons}\{1, 2\} = \begin{bmatrix} 1 & 2 & 1 \\ 1 & 4 & 3 \end{bmatrix}, \\ \text{cons}\{1, 3\} &= [2 \ 1 \ 4.5 \ 1], \quad \text{cons}\{1, 4\} = [0 \ 2 \ 2.5 \ 3]. \end{aligned}$$

DDS solves this problem and returns objective value -2.87198 . CVX also solves the problem by using successive approximation method and returns the same solution.

5. MATRIX MONOTONICITY, QUANTUM ENTROPY

Consider a function $f : \mathbb{R} \rightarrow \mathbb{R} \cup \{+\infty\}$ and let $X \in \mathbb{H}^n$ be a Hermitian matrix (with entries from \mathbb{C}) with a spectral decomposition $X = U\text{Diag}(\lambda_1, \dots, \lambda_n)U^*$, where Diag returns a diagonal matrix with the given entries on its diagonal and U^* is the conjugate transpose of a unitary matrix U . Then, $F : \mathbb{H}^n \rightarrow \mathbb{R} \cup \{+\infty\}$ is defined as

$$F(X) := \text{Tr}(U\text{Diag}(f(\lambda_1), \dots, f(\lambda_n))U^*).$$

Study of such matrix functions go back to the work of Löwner as well as Von-Neumann (see [5], [13], and the references therein). A function $f : (a, b) \mapsto \mathbb{R}$ is said to be matrix monotone if for any two self-adjoint matrix X and Y with eigenvalues in (a, b) that satisfy $X \succeq Y$, we have $f(X) \succeq f(Y)$. A function $f : (a, b) \mapsto \mathbb{R}$ is said to be *matrix convex* if for any pair of self-adjoint

matrices X and Y with eigenvalues in (a, b) , we have

$$(43) \quad f(tX + (1-t)Y) \preceq tf(X) + (1-t)f(Y), \quad \forall t \in (0, 1).$$

Faybusovich and Tsuchiya [7] utilized the connection between the matrix monotone functions and self-concordant functions. Let f be a continuously differentiable function whose derivative is matrix monotone on the positive semi-axis and let us define the function $\phi : \mathbb{S}^n \mapsto \mathbb{R}$ as $\phi(X) := \text{Tr}(f(X))$. Then, the function

$$(44) \quad \Phi(t, X) := -\ln(t - \phi(X)) - \ln \det(X)$$

is a $(n+1)$ -s.c. barrier for the epigraph of $\phi(X)$. This convex set has many applications. Many optimization problems arising in quantum information theory and some other areas requires dealing with the so-called *quantum* or von Neumann entropy which is the function of the form $\text{Tr}(X \ln(X))$ where X is a positive semidefinite complex Hermitian matrix. If we consider $f(x) = x \ln(x)$, then $f'(x) = 1 + \ln(x)$ is matrix monotone on $(0, \infty)$ (see, for instance [10]-Example 4.2) and so we have a s.c. barrier for the set

$$\{(t, X) \in \mathbb{R} \oplus \mathbb{S}^n : X \ln(X) \leq t\}.$$

We have to solve the optimization problem

$$(45) \quad \Phi_*(\eta, Y) = \sup_{t, X} t\eta + \langle X, Y \rangle + \ln(t - \phi(X)) + \ln \det(X),$$

to calculate the LF conjugate of (44), which is done in Appendix D. Appendix D also contains some results on the derivative of the quantum entropy function.

5.1. Adding quantum entropy based constraints. Let $f(X) := \text{Tr}(X \ln(X))$ and consider ℓ quantum entropy constraints of the form

$$(46) \quad f(F_0^i + x_1 F_1^i + \dots + x_n F_n^i) \leq g_i^\top x + d_i, \quad i \in \{1, \dots, \ell\}.$$

F_j^i 's are n_i -by- n_i symmetric matrices. To input (46) to DDS as the k th block, we define:

$$(47) \quad \text{cons}\{\mathbf{k}, 1\} = \text{'QE'}, \quad \text{cons}\{\mathbf{k}, 2\} = [n_1, \dots, n_\ell],$$

$$\mathbf{A}\{\mathbf{k}, 1\} := \begin{bmatrix} g_1^\top \\ \text{sm2vec}(F_1^1), \dots, \text{sm2vec}(F_n^1) \\ \vdots \\ g_\ell^\top \\ \text{sm2vec}(F_1^\ell), \dots, \text{sm2vec}(F_n^\ell) \end{bmatrix}, \quad \mathbf{b}\{\mathbf{k}, 1\} := \begin{bmatrix} d_1 \\ \text{sm2vec}(F_0^1) \\ \vdots \\ d_\ell \\ \text{sm2vec}(F_0^\ell) \end{bmatrix}.$$

Example 5.1. Assume that we want to find scalars x_1 , x_2 , and x_3 such that $2x_1 + 3x_2 - x_3 \leq 5$ and all the eigenvalues of $H := x_1A_1 + x_2A_2 + x_3A_3$ are at least 3, for

$$A_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad A_2 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \quad A_3 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

such that the quantum entropy f of H is minimized. We can write this problem as

$$(48) \quad \begin{aligned} & \min \quad t \\ & \text{s.t.} \quad f(x_1A_1 + x_2A_2 + x_3A_3) \leq t, \\ & \quad \quad 2x_1 + 3x_2 - x_3 \leq 5, \\ & \quad \quad x_1A_1 + x_2A_2 + x_3A_3 \succeq 3I. \end{aligned}$$

For the objective function we have $c = (0, 0, 0, 1)^\top$. Assume that the first and second blocks are LP and SDP as before. We define the third block of constraints as:

$$\begin{aligned} \text{cons}\{\mathcal{3}, 1\} &= 'QE', \quad \text{cons}\{\mathcal{3}, 2\} = [3], \quad \text{b}\{\mathcal{3}, 1\} := \begin{bmatrix} 0 \\ \text{zeros}(9, 1) \end{bmatrix}, \\ A\{\mathcal{3}, 1\} &:= \begin{bmatrix} 0 & 0 & 0 & 1 \\ \text{sm2vec}(A_1) & \text{sm2vec}(A_2) & \text{sm2vec}(A_3) & \text{sm2vec}(\text{zeros}(3)) \end{bmatrix}. \end{aligned}$$

If we run DDS, the answer we get is $(x_1, x_2, x_3) = (4, -1, 0)$ with $f(H) = 14.63$.

6. CONSTRAINTS INVOLVING HYPERBOLIC POLYNOMIALS

A polynomial $p(x) \in \mathbb{R}[x_1, \dots, x_m]$ is said to be *homogeneous* if every term has the same degree d . A homogeneous polynomial p is hyperbolic in direction $e \in \mathbb{R}^m$ if

- $p(e) > 0$.
- for every $x \in \mathbb{R}^m$, the univariate polynomial $p(x + te)$ has only real roots.

The *hyperbolicity cone* defined by a hyperbolic polynomial p in direction e , $\Lambda(p, e)$, is defined as

$$(49) \quad \Lambda(p, e) := \{x \in \mathbb{R}^m : p(x + \lambda e) \geq 0, \forall \lambda \geq 0\}.$$

Example 6.1. The polynomial $p(x) = x_1^2 - x_2^2 - \dots - x_m^2$ is hyperbolic in the direction $e = (1, 0, \dots, 0)^\top$ and the hyperbolicity cone with respect to e is the second-order cone. The polynomial $p(X) = \det(X)$ defined on \mathbb{S}^n is hyperbolic in the direction I , and the hyperbolicity cone with respect to I is the positive-semidefinite cone.

By the above example, optimization over hyperbolicity cone is an extension of SOCP and SDP. The following theorem by Güler gives a s.c. barrier for the hyperbolicity cone.

Theorem 6.1 (Güler [9]). *Let $p(x)$ be a homogeneous polynomial of degree d , which is hyperbolic in direction e . Then, the function $-\ln(p(x))$ is a d -LH s.c. barrier for $\Lambda(p, e)$.*

DDS handles optimization problems involving hyperbolic polynomials using the above s.c. barrier. A computational problem is that, currently, we do not have a practical, efficient, algorithm to evaluate the LF conjugate of $-\ln(p(x))$. Therefore, DDS uses a primal-heavy version of the algorithm for these problems.

6.1. Different formats for inputting multivariate polynomials. To input constraints involving hyperbolic polynomials, we use a matrix named `poly`. In DDS, there are different options to input a multivariate polynomial:

Using monomials: In this representation, if $p(x)$ is a polynomial of m variables, then `poly` is an k -by- $(m + 1)$ matrix, where k is the number of monomials. In the j th row, the first m entries are the power of the m variables in the monomial, and the last entry is the coefficient of the monomial in $p(x)$. For example, if $p(x) = x_1^2 - x_2^2 - x_3^2$, then

$$\text{poly} := \begin{bmatrix} 2 & 0 & 0 & 1 \\ 0 & 2 & 0 & -1 \\ 0 & 0 & 2 & -1 \end{bmatrix}.$$

Note: In many applications, the above matrix is very sparse. DDS recommends that in the monomial format, `poly` should be defined as a sparse matrix.

Using straight-line program: Another way to represent a polynomial is by a *straight-line program*, which can be seen as a rooted directed graph with no cycle. The leaves represent the variables or constants. Each node is a simple binary operation (such as addition or multiplication), and the root is the result of the polynomial. In this case, `poly` is a k -by-4 matrix, where each row represent a simple operation. Assume that $p(x)$ has m variables, then we define

$$f_0 = 1, \quad f_i := x_i, \quad i \in \{1, \dots, m\}.$$

The ℓ th row of `poly` is of the form $[\alpha_j \ i \ j \ \square]$, which means that

$$f_{m+j} = \alpha_j(f_i \square f_j).$$

Operations are indexed by 2-digit numbers as the following table:

operation \square	index
+	11
-	22
\times	33
$f_i^{f_j}$	44

Straight-line program is not unique for a polynomial. For example, if $p(x) = x_1^2 - x_2^2 - x_3^2$, we have at least the following two representations:

$$\begin{bmatrix} 1 & 1 & 1 & 33 \\ -1 & 2 & 2 & 33 \\ -1 & 3 & 3 & 33 \\ 1 & 4 & 5 & 11 \\ 1 & 6 & 7 & 11 \end{bmatrix}, \quad \begin{bmatrix} 1 & 1 & 2 & 44 \\ -1 & 2 & 2 & 44 \\ -1 & 3 & 2 & 44 \\ 1 & 4 & 5 & 11 \\ 1 & 6 & 7 & 11 \end{bmatrix}.$$

Determinantal representation: In this case, if possible, the polynomial $p(x)$ is written as

$$(50) \quad p(x) = \det(H_0 + x_1 H_1 + x_2 H_2 + \cdots + x_m H_m),$$

where $H_i, i \in \{0, 1, \dots, m\}$ are in \mathbb{S}^m . In this case, we define

$$\text{poly} := [\text{sm2vec}(H_0) \quad \text{sm2vec}(H_1) \quad \cdots \quad \text{sm2vec}(H_m)].$$

For example, for $p(x) = x_1^2 - x_2^2 - x_3^2$, we can have

$$H_0 := \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, \quad H_1 := \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad H_2 := \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \quad H_3 := \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

6.2. Adding constraints involving hyperbolic polynomials. Consider a hyperbolic polynomial constraint of the form

$$(51) \quad p(Ax + b) \geq 0.$$

To input this constraint to DDS as the k th block, A and b are defined as before, and different parts of `cons` are defined as follows:

`cons{k,1}`= 'HB',

`cons{k,2}`= number of variables in $p(x)$.

`cons{k,3}` is the `poly` that can be given in one of the three formats of Subsection 6.1.

`cons{k,4}` is the format of polynomial that can be 'monomial', 'straight_line', or 'determinant'.

`cons{k,5}` is the direction of hyperbolicity or a vector in the interior of the hyperbolicity cone.

Example 6.2. Assume that we want to give constraint (51) to DDS for $p(x) = x_1^2 - x_2^2 - x_3^2$, using the monomial format. Then, *cons* part is defined as

$$\begin{aligned} \mathit{cons}\{k, 1\} &= \text{'HB'}, & \mathit{cons}\{k, 2\} &= [3], \\ \mathit{cons}\{k, 3\} &= \begin{bmatrix} 2 & 0 & 0 & 1 \\ 0 & 2 & 0 & -1 \\ 0 & 0 & 2 & -1 \end{bmatrix}, \\ \mathit{cons}\{k, 4\} &= \text{'monomial'}, & \mathit{cons}\{k, 5\} &= \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}. \end{aligned}$$

7. PRIMAL-HEAVY VERSION OF THE ALGORITHM

For some class of problems, such as hyperbolic programming, a computable s.c. barrier Φ is available for the set D , while the LF of it is not available. For these classes, DDS uses a primal-heavy version of the algorithm. In the primal-heavy version, we approximate the primal-dual system of equations for computing the search directions by approximating the gradient and Hessian of Φ_* . The approximations are by using the relations between the derivatives of Φ and Φ_* : for every point $z \in \text{int}D$ we have

$$(52) \quad z = \Phi'_*(\Phi'(z)), \quad \Phi''_*(\Phi'(z)) = [\Phi''(z)]^{-1}.$$

Instead of the primal-dual proximity measure defined in [11], we use the primal-heavy version:

$$(53) \quad \left\| \frac{\tau y}{\mu} - \Phi'(u) \right\|_{\Phi''(u)},$$

where $u := Ax + \frac{1}{\tau}z^0$, τ is an artificial variable we use in the formulation of the central path (see [11] for details), and μ is the parameter of the central path. By [11]-Corollary 4.1, this proximity measure is “equivalent” to the primal-dual one, but (53) is less efficient computationally.

By using a primal-heavy version, we lose some of good properties of primal-dual setup, such as the ability to move in a wider neighbourhood of the central path. Moreover, in the primal-heavy version, we have to somehow make sure the dual iterates y are feasible (or at least the final dual iterate is). Another issue is with calculating the duality gap (4). For a general convex domain D , we need Φ'_* to accurately calculate $\delta_*(y|D)$ as explained in [11]. Note that when D is a shifted cone $D = K - b$, then we have

$$(54) \quad \delta_*(y|D) = -\langle b, y \rangle.$$

To calculate the duality gap, we can write it as the summation of separate terms for the domains, and if a domain with only the primal barrier is a shifted cone, we can use (54). This is the case for the current version of DDS as hyperbolicity cones are the only domains without an efficient dual function.

To make sure that the dual iterates are feasible, we choose our neighborhoods to satisfy

$$(55) \quad \left\| \frac{\tau y}{\mu} - \Phi'(u) \right\|_{\Phi''(u)} < 1,$$

and by the Dikin ellipsoid property of s.c. functions, y iterates stay feasible. We can specify in OPTIONS if we want to use a primal-heavy version of the algorithm in DDS by

```
OPTIONS.primal=1;
```

8. MORE NUMERICAL EXAMPLES

In this section, we present some numerical examples of running our code. We performed computational experiments using the software MATLAB R2018b, on a 4-core 3.2 GHz Intel Xeon X5672 machine with 96GB of memory.

Here is a typical output of DDS for the problem copo-14 from Dimacs Library.

Iteration:	mu	Rel-Duality-Gap	Pfeas	Dfeas	Obj-Val
1:	1.13e+00	9.92e-01	8.62e-01	9.34e+01	5.368195e-03
2:	1.27e+00	9.88e-01	7.92e-01	8.36e+01	1.178187e-01
3:	1.58e+00	9.79e-01	6.61e-01	6.73e+01	3.430985e-01
25:	9.64e+08	1.97e-08	8.79e-10	1.12e-07	4.522724e-07
26:	1.49e+09	1.52e-08	5.68e-10	7.21e-08	2.934715e-07
27:	1.88e+09	1.21e-08	4.51e-10	5.73e-08	2.339206e-07
28:	5.67e+09	3.99e-09	1.49e-10	1.90e-08	7.924154e-08

Status: Solved; returned vector x is an optimal solution.

Primal feasibility = 1.49e-10.

Dual feasibility = 1.90e-08,

Relative duality gap = 3.99e-09,

Optimal objective value = 7.92415377e-08.

8.1. **LP-SOCP-SDP.** In this subsection, we consider LP-SOCP-SDP instances mostly from the Dimacs library [19]. Note that the problems in the library are for the standard equality form and we solve the dual of the problems. Table 3 shows the results.

TABLE 3. Numerical results for some problem from the Dimacs library for $tol = 10^{-8}$.

Problem	size of A	Type of Constraints	Iterations
nb	2383 * 123	SOCP-LP	41
nb_L1	3176 * 915	SOCP-LP	35
nb_L2	4195 * 123	SOCP-LP	26
nb_L2_bessel	2641 * 123	SOCP-LP	26
filter48_socp	3284 * 969	SDP-SOCP-LP	80
filtinf1	3395 * 983	SDP-SOCP-LP	21
truss5	3301 * 208	SDP	68
truss8	11914 * 496	SDP	80
copo14	3108 * 1275	SDP-LP	28
copo23	3108 * 1275	SDP-LP	45
toruspm3-8-50	262144 * 512	SDP	19
torusg3-8	262144 * 512	SDP	24
sched_50_50_scaled	4977 * 2526	SOCP-LP	81
mater-3	39448 * 1439	SDP-LP	130
cnhil8	14400 * 1716	SDP	31
cnhil10	48400 * 5005	SDP	37
cphil10	48400 * 5005	SDP	9
ros_500	17944 * 4988	SDP	44
sensor_500	245601 * 3540	SDP	56
taha1a	231672 * 3002	SDP	23
taha1a	231672 * 3002	SDP	42
G40mc	4000000 * 2000	SDP	33
1tc.1024	1048576 * 7937	SDP	41
yalsdp	30000 * 5051	SDP	26

8.2. LP-SOCP-SDP constraints combined with geometric and entropy ones. In this subsection, we solve instances that have LP-SOCP-SDP constraints as well as constraints defined by epigraph of univariate functions. Consider problems of the form

$$\begin{aligned}
 & \min \quad c^\top x \\
 & \text{s.t.} \quad A_0 + x_1 A_1 + \cdots + x_n A_n \succeq 0, \\
 & \quad \sum_{i=1}^k \exp((a_i^1)^\top x + b_i^1) + \sum_{i=1}^k \text{entr}((\bar{a}_i^1)^\top x + \bar{b}_i^1) + (g^1)^\top x + \gamma^1 \leq 0, \\
 (56) \quad & \quad \sum_{i=1}^k \exp((a_i^2)^\top x + b_i^2) + \sum_{i=1}^k \text{entr}((\bar{a}_i^2)^\top x + \bar{b}_i^2) + (g^2)^\top x + \gamma^2 \leq 0,
 \end{aligned}$$

where $c \in \mathbb{R}^n$, A_i 's are m -by- m symmetric matrices, and $\text{entr}(t) = \ln(t)$. This problem has one block of SDP constraints plus two constraints involving exponential and entropy functions. We compare running time of DDS with CVX. CVX uses successive approximation method and calls an SDP solver several times. The numbers are the average for 10 examples with random data.

Note that CVX does a reformulation to feed the problem to an SDP solver. We input the problem into CVX as the obvious way of writing it using *exp* and *entr* functions.

TABLE 4.

dim of c , (n)	Size of SDP m	number of exp and entr	running time for DDS	running time for CVX
200	40	100	13 sec	48 sec
200	40	100	16 sec	96 sec
200	40	500	30 sec	225 sec
500	40	100	81 sec	366 sec

REFERENCES

- [1] S. BOYD, S.-J. KIM, L. VANDENBERGHE, AND A. HASSIBI, *A tutorial on geometric programming*, Optimization and Engineering, 8 (2007), pp. 67–127.
- [2] P. BRÄNDÉN, *Obstructions to determinantal representability*, Advances in Mathematics, 226 (2011), pp. 1202–1212.
- [3] V. CHANDRASEKARAN AND P. SHAH, *Relative entropy optimization and its applications*, Mathematical Programming, 161 (2017), pp. 1–32.
- [4] J. DAHL AND E. D. ANDERSEN, *A primal-dual interior-point algorithm for nonsymmetric exponential-cone optimization*, Optimization Online, (2019).
- [5] C. DAVIS, *All convex invariant functions of hermitian matrices*, Archiv der Mathematik, 8 (1957), pp. 276–278.
- [6] S.-C. FANG, J. R. RAJASEKERA, AND H.-S. J. TSAO, *Entropy optimization and Mathematical Programming*, vol. 8, Springer Science & Business Media, 1997.
- [7] L. FAYBUSOVICH AND T. TSUCHIYA, *Matrix monotonicity and self-concordance: how to handle quantum entropy in optimization problems*, Optimization Letters, (2017), pp. 1513–1526.

- [8] M. GRANT, S. BOYD, AND Y. YE, *CVX: MATLAB software for disciplined convex programming*, 2008.
- [9] O. GÜLER, *Hyperbolic polynomials and interior point methods for convex programming*, Mathematics of Operations Research, 22 (1997), pp. 350–377.
- [10] F. HIAI AND D. PETZ, *Introduction to matrix analysis and applications*, Springer Science & Business Media, 2014.
- [11] M. KARIMI AND L. TUNÇEL, *Primal-dual interior-point methods for Domain-Driven formulations*, Mathematics of Operations Research (to appear), arXiv preprint arXiv:1804.06925, (2018).
- [12] ———, *Status determination by interior-point methods for convex optimization problems in domain-driven form*, arXiv preprint arXiv:1901.07084, (2019).
- [13] A. S. LEWIS, *The mathematics of eigenvalue optimization*, Mathematical Programming, 97 (2003), pp. 155–176.
- [14] H. D. MITTELMANN, *The state-of-the-art in conic optimization software*, in Handbook on Semidefinite, Conic and Polynomial Optimization, Springer, 2012, pp. 671–686.
- [15] MOSEK ApS, *The MOSEK optimization toolbox for MATLAB manual. Version 9.0.*, 2019. <http://docs.mosek.com/9.0/toolbox/index.html>.
- [16] A. NEMIROVSKI AND L. TUNÇEL, *Cone-free primal-dual path-following and potential reduction polynomial time interior-point methods*, Mathematical Programming, 102 (2005), pp. 261–294.
- [17] Y. NESTEROV, *Lectures on Convex Optimization*, Springer, 2018.
- [18] Y. NESTEROV AND A. NEMIROVSKI, *Interior-Point Polynomial Algorithms in Convex Programming*, SIAM Series in Applied Mathematics, SIAM: Philadelphia, 1994.
- [19] G. PATAKI AND S. SCHMIETA, *The DIMACS library of semidefinite-quadratic-linear programs*, tech. rep., Preliminary draft, Computational Optimization Research Center, Columbia University, New York, 2002.
- [20] B. RECHT, M. FAZEL, AND P. A. PARRILO, *Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization*, SIAM Review, 52 (2010), pp. 471–501.
- [21] J. F. STURM, *Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones*, Optimization Methods and Software, 11 (1999), pp. 625–653.
- [22] K.-C. TOH, M. J. TODD, AND R. H. TÜTÜNCÜ, *On the implementation and usage of SDPT3—a MATLAB software package for semidefinite-quadratic-linear programming, version 4.0*, in Handbook on semidefinite, conic and polynomial optimization, Springer, 2012, pp. 715–754.
- [23] R. H. TÜTÜNCÜ, K.-C. TOH, AND M. J. TODD, *Solving semidefinite-quadratic-linear programs using SDPT3*, Mathematical programming, 95 (2003), pp. 189–217.

APPENDIX A. CALCULATING THE PREDICTOR AND CORRECTOR STEPS

As discussed in [11], for both the predictor and corrector steps, the matrix in the LHS of the linear system is

$$\begin{aligned}
 & U^\top \left[\begin{array}{c} \left[\begin{array}{cc} H & h \\ h^\top & \zeta \end{array} \right] & 0 \\ 0 & \left[\begin{array}{cc} G + \eta_* h_* h_*^\top & -\eta_* h_* \\ -\eta_* h_*^\top & \eta_* \end{array} \right] \end{array} \right] U \\
 (57) \quad & = \left[\begin{array}{ccc} A^\top H A + \eta_* c c^\top & A^\top h + \eta_* h_*^\top c_A c & \eta_* c h_*^\top F^\top \\ h^\top A + \eta_* c_A^\top h_* c^\top & \zeta + c_A^\top G c_A + \eta_* (c_A^\top h_*)^2 & c_A^\top G F^\top + \eta_* c_A^\top h_* h_*^\top F^\top \\ \eta_* F h_* c^\top & F G c_A + \eta_* F h_* h_*^\top c_A & F G F^\top + \eta_* F h_* h_*^\top F^\top \end{array} \right],
 \end{aligned}$$

where U is a matrix that contains the linear transformations we need:

$$(58) \quad U = \begin{bmatrix} A & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -c_A & -F^\top \\ c^\top & 0 & 0 \end{bmatrix},$$

where F is a matrix whose rows form a basis for $\text{null}(A)$ and c_A is any vector that satisfies $A^\top c_A = c$. The problem of this system in practice is that calculating F is not computationally efficient. Assume that we are solving the system

$$\begin{bmatrix} A^\top H A + \eta_* c c^\top & A^\top h + \eta_* h_*^\top c_A c & \eta_* c h_*^\top F^\top \\ h^\top A + \eta_* c_A^\top h_* c^\top & \zeta + c_A^\top G c_A + \eta_* (c_A^\top h_*)^2 & c_A^\top G F^\top + \eta_* c_A^\top h_* h_*^\top F^\top \\ \eta_* F h_* c^\top & F G c_A + \eta_* F h_* h_*^\top c_A & F G F^\top + \eta_* F h_* h_*^\top F^\top \end{bmatrix} \begin{bmatrix} \bar{d}_x \\ d_\tau \\ d_v \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ F r_3 \end{bmatrix}.$$

At the end, we are interested in $F^\top d_v$ to calculate our search directions. If we consider the last equation, we can remove the matrix F multiplied from the left to all the terms as

$$(59) \quad \begin{aligned} & \eta_* h_* c^\top \bar{d}_x + d_\tau (G + \eta_* h_* h_*^\top) c_A + (G + \eta_* h_* h_*^\top) F^\top d_v = r_3 + A w \\ \Rightarrow & \eta_* \bar{G}^{-1} h_* c^\top \bar{d}_x + d_\tau c_A + F^\top d_v = \bar{G}^{-1} r_3 + \bar{G}^{-1} A w, \end{aligned}$$

where $\bar{G} := G + \eta_* h_* h_*^\top$. Now, we multiply the last equation by A^\top from the left and eliminate d_v as

$$(60) \quad \eta_* A^\top \bar{G}^{-1} h_* c^\top \bar{d}_x + d_\tau c = A^\top \bar{G}^{-1} r_3 + A^\top \bar{G}^{-1} A w.$$

By using the equations in (59) and (60), we can get the system

$$\begin{bmatrix} A^\top H A + (\eta_* - \eta_*^2 h_*^\top \bar{G}^{-1} h_*) c c^\top & \eta_* c h_*^\top \bar{G}^{-1} A & A^\top h \\ \eta_* A^\top \bar{G}^{-1} h_* c^\top & -A^\top \bar{G}^{-1} A & c \\ h^\top A & c^\top & \zeta \end{bmatrix} \begin{bmatrix} \bar{d}_x \\ w \\ d_\tau \end{bmatrix} = \begin{bmatrix} r_1 - \eta_* c h_*^\top \bar{G}^{-1} r_3 \\ A^\top \bar{G}^{-1} r_3 \\ r_2 - c_A^\top r_3 \end{bmatrix}.$$

Note that we have

$$(61) \quad \begin{aligned} \bar{G}^{-1} &= G^{-1} - \eta_* \frac{G^{-1} h_* h_*^\top G^{-1}}{1 + \eta_* h_*^\top G^{-1} h_*}, \\ \Rightarrow \bar{G}^{-1} h_* &= \frac{1}{1 + \eta_* h_*^\top G^{-1} h_*} G^{-1} h_*. \end{aligned}$$

Let us define

$$(62) \quad \beta := \frac{1}{1 + \eta_* h_*^\top G^{-1} h_*}.$$

Then we can see that the LHS matrix in (61) can be written as

$$(63) \quad \underbrace{\begin{bmatrix} A^\top H A & 0 & A^\top h \\ 0 & -A^\top G^{-1} A & c \\ h^\top A & c^\top & \zeta \end{bmatrix}}_{\tilde{H}} + \eta_* \beta \begin{bmatrix} c \\ A^\top G^{-1} h_* \\ 0 \end{bmatrix} \begin{bmatrix} c \\ A^\top G^{-1} h_* \\ 0 \end{bmatrix}^\top.$$

This matrix is a $(2n+1)$ -by- $(2n+1)$ matrix \tilde{H} plus a rank one update. If we have the Cholesky or LU factorization of $A^\top H A$ and $A^\top G^{-1} A$ (in the case that $G := \mu^2 H$, we need just one such factorization), then we have such a factorization for the $2n$ -by- $2n$ leading minor of \tilde{H} and we can easily extend it to a factorization for the whole \tilde{H} . To solve the whole system, we can then use Sherman-Morrison formula.

APPENDIX B. LEGENDRE-FENCHEL CONJUGATES OF UNIVARIATE CONVEX FUNCTIONS

Table 5 shows the LF conjugate for the first three functions of Table 2. Finding the LF

TABLE 5. LF conjugates for the first three s.c. barriers in Table 2.

	$\Phi(z, t)$	$\Phi_*(y, \eta)$
1	$-\ln(t + \ln(z)) - \ln(z)$	$-1 + (-\eta + 1) \left[-1 + \ln \frac{-(-\eta+1)}{y} \right] - \ln(-\eta)$
2	$-\ln(\ln(t) - z) - \ln(t)$	$-1 + (y + 1) \left[-1 + \ln \frac{-(y+1)}{\eta} \right] - \ln(y)$
3	$-\ln(t - z \ln(z)) - \ln(z)$	$-\ln(-\eta) + \theta \left(1 + \frac{y}{\eta} - \ln(-\eta) \right) - \frac{y}{\eta} + \frac{1}{\theta \left(1 + \frac{y}{\eta} - \ln(-\eta) \right)} - 3$

conjugates for the first two functions can be handled with easy calculus. In the third row, $\theta(r)$, defined in [16], is the unique solution of

$$(64) \quad \frac{1}{\theta} - \ln(\theta) = r.$$

It is easy to check by implicit differentiation that

$$\theta'(r) = -\frac{\theta^2(r)}{\theta(r) + 1}, \quad \theta''(r) = \frac{\theta^2(r) + 2\theta(r)}{[\theta(r) + 1]^2} \theta'(r).$$

We can calculate $\theta(r)$ with accuracy 10^{-15} in few steps with the following Newton iterations:

$$\theta_k = \frac{\theta_{k-1}^2}{\theta_{k-1} + 1} \left[1 + \frac{2}{\theta_{k-1}} - \ln(\theta_{k-1}) - r \right], \quad \theta_0 = \begin{cases} \exp(-r), & r \leq 1 \\ \frac{1}{r - \ln(r)}, & r > 1 \end{cases}.$$

Now we want to add the sets defined by constraints $|z|^p \leq t$, $p \geq 1$. These functions are of type 4. The corresponding s.c. barrier is $\Phi(z, t) = -\ln(t^{\frac{2}{p}} - z^2) - 2\ln(t)$. Let us first see how to calculate the LF conjugate. We need to solve the following optimization problem:

$$(65) \quad \min_{z,t} \left\{ yz + \eta t + \ln(t^{\frac{2}{p}} - z^2) + 2\ln(t) \right\}.$$

The optimal solution satisfies:

$$(66) \quad y = \frac{2z}{t^{\frac{2}{p}} - z^2}, \quad \eta = -\frac{\frac{2}{p}t^{\frac{2}{p}-1}}{t^{\frac{2}{p}} - z^2} - \frac{2}{t}.$$

By doing some algebra, we can see that z and t satisfy:

$$(67) \quad \begin{aligned} y \left(\frac{2(\frac{1}{p} + 1) + \frac{1}{p}yz}{-\eta} \right)^{\frac{2}{p}} - yz^2 - 2z &= 0, \\ t &= \frac{2(\frac{1}{p} + 1) + \frac{1}{p}yz}{-\eta}. \end{aligned}$$

Let us define $z(y, \eta)$ as the solution of the first equation in (67). For each pair (y, η) , we can calculate $z(y, \eta)$ by few iterations of Newton method. Then, the first and second derivative can be calculated in terms of $z(y, \eta)$. In DDS, we have two functions for these derivatives.

```
p1_TD(y,eta,p)    % returns z
p1_TD_der(y,eta,p) % returns [z_y z_eta z_y,y z_y,eta z_eta,eta]
```

For the set defined by $-z^p \leq t$, $0 \leq p \leq 1$, $z > 0$, the corresponding s.c. barrier is $\Phi(z, t) = -\ln(z^p + t) - \ln(z)$. Let us see how to calculate the LF conjugate. We need to solve the following optimization problem:

$$(68) \quad \min_{z,t} \{ yz + \eta t + \ln(z^p + t) + \ln(z) \}.$$

The optimal solution satisfies:

$$(69) \quad y = \frac{-pz^{(p-1)}}{z^p + t} - \frac{1}{z}, \quad \eta = -\frac{1}{z^p + t}.$$

By doing some algebra, we can see that z satisfies:

$$(70) \quad y - \eta pz^{(p-1)} + \frac{1}{z} = 0.$$

Similar to the previous case, let us define $z(y, \eta)$ as the solution of the first equation in (70). For each pair (y, η) , we can calculate $z(y, \eta)$ by few iterations of Newton method. Then, the first and second derivative can be calculated in terms of $z(y, \eta)$. The important point is that when we calculate $z(y, \eta)$, then the derivatives can be calculated by explicit formulas. In our code, we have two functions

```
p2_TD(y,eta,p)    % returns z
p2_TD_der(y,eta,p) % returns [z_y z_eta z_y,y z_y,eta z_eta,eta]
```

The inputs to the above functions can be vectors. Table 6 is the continuation of Table 5.

TABLE 6. s.c. barriers and their LF conjugate for rows 4 and 5 of Table 2

	s.c. barrier $\Phi(z, t)$	$\Phi_*(y, \eta)$
4	$-\ln(t^{\frac{2}{p}} - z^2) - 2\ln(t)$	$-\left(\frac{2}{p} + \left(\frac{1}{p} - 1\right)yz(y, \eta)\right) - 2 + 2\ln\left(\frac{2\left(\frac{1}{p}+1\right)+\frac{1}{p}yz(y,\eta)}{-\eta}\right)$ $+ \ln\left(\left(\frac{2\left(\frac{1}{p}+1\right)+\frac{1}{p}yz(y,\eta)}{-\eta}\right)^{\frac{2}{p}} - z^2(y, \eta)\right)$
5	$-\ln(z^p + t) - \ln(z)$	$\eta(p - 1)z^p(y, \eta) - 2 - \ln(-\eta) + \ln(z(y, \eta))$

For the set defined by $\frac{1}{z} \leq t, z > 0$, the corresponding s.c. barrier is $\Phi(z, t) = -\ln(zt - 1)$. To calculate the LF conjugate, we need to solve the following optimization problem:

$$(71) \quad \min_{z,t} \{yz + \eta t + \ln(zt - 1)\}.$$

At the optimal solution, we must have

$$(72) \quad y = -\frac{t}{zt - 1}, \quad \eta = -\frac{z}{zt - 1}.$$

Since we have $z, t > 0$, then we must have $y, \eta < 0$ at a dual feasible point. By solving these systems we get

$$(73) \quad t = \frac{-1 - \sqrt{1 + 4y\eta}}{2\eta}, \quad z = \frac{-1 - \sqrt{1 + 4y\eta}}{2y},$$

$$\Rightarrow \Phi_*(y, \eta) = -1 - \sqrt{1 + 4y\eta} + \ln\left(\frac{1 + \sqrt{1 + 4y\eta}}{2y\eta}\right).$$

APPENDIX C. IMPLEMENTATION DETAILS FOR SDP AND GENERALIZED EPIGRAPHS OF MATRIX NORMS

For implementing SDP and generalized epigraph of a matrix norm that we have to represent symmetric matrices in term of vectors, there are some changes that we discuss in this section.

C.1. SDP. Let us elaborate more on calculating the gradient and Hessian for SDP part. For SDP (13), we have:

$$(74) \quad \Phi(Z) = -\ln(\det(F_0 + Z)),$$

$$\Phi_*(Y) = -n - \langle F_0, Y \rangle - \ln(\det(-Y)).$$

For function $f = -\ln(\det(X))$, we have:

$$(75) \quad \begin{aligned} \langle f'(X), H \rangle &= -\text{Tr}(X^{-1}H), \\ \langle f''(X)H, H \rangle &= \text{Tr}(X^{-1}HX^{-1}H). \end{aligned}$$

To implement our algorithm, for each matrix X , we need to find the corresponding gradient g_X and Hessian H_X , such that for any symmetric positive semidefinite matrix X and symmetric matrix H we have:

$$(76) \quad \begin{aligned} -\text{Tr}(X^{-1}H) &= -g_X^\top \text{sm2vec}(H), \\ \text{Tr}(X^{-1}HX^{-1}H) &= \text{sm2vec}(H)^\top H_X \text{sm2vec}(H). \end{aligned}$$

It can be shown that $g_X = \text{sm2vec}(X^{-1})$ and $H_X = X^{-1} \otimes X^{-1}$, where \otimes stands for the Kronecker product of two matrices. Although this representation is theoretically nice, it is not efficient to calculate the inverse of a matrix explicitly. As we explain, we do not explicitly form inverses of such matrices in our code. Consider forming $A^\top \Phi''(u)A$ for calculating the search directions. Usually in practice, matrix A is tall and thin in our setup. Hence, it may not be efficient to form $\Phi''(u)$ as its size may be much bigger than $A^\top \Phi''(u)A$. In our code, we do not form $\Phi''(u)$ and we have a function

`hessian_A(b,Z,w,A)`

that directly returns $A^\top \Phi''(u)A$. Note that $\Phi''(u)$ has a block diagonal structure, and each block of the SDP part is equal to H_X defined in (76) for a properly chosen X . Hence, calculating $A^\top \Phi''(u)A$ for the SDP part reduces to calculating $v^\top H_X w$ for two vectors v and w . Using (76), we have

$$(77) \quad v^\top H_X w = \text{Tr}(X^{-1} \text{vec2sm}(v) X^{-1} \text{vec2sm}(w)).$$

Now, we calculate $X^{-1} \text{vec2sm}(v)$ and $X^{-1} \text{vec2sm}(w)$ by solving linear systems of equations instead of explicitly forming X^{-1} .

Other numerical difficulties happen for calculating the corrector step, specially when the iterates are getting close to the boundary. In DDS, we have different functions to calculate each part of the predictor and corrector steps. In our functions, we use the properties of Kronecker product that for matrices A , B , and X of proper size, we have

$$(78) \quad \begin{aligned} (B^\top \otimes A) \text{sm2vec}(X) &= \text{sm2vec}(AXB), \\ (A \otimes B)^{-1} &= A^{-1} \otimes B^{-1}. \end{aligned}$$

In the code, we do not use $[\Phi_*''(y)]^{-1}$ explicitly, but we could avoid it by using the following formula

$$(79) \quad [\Phi_*''(y)]^{-1} = \Phi''(\Phi_*'(y)).$$

For the blocks of the SDP part, we do not even need to use (79), because by using the second property in (78), for $H_X = X^{-1} \otimes X^{-1}$ we have

$$(80) \quad (H_X)^{-1} = (X^{-1} \otimes X^{-1})^{-1} = X \otimes X.$$

In our code, as we explained above, we do not explicitly form $[\Phi_*''(y)]^{-1}$ and the function

```
hessian_Leg_inv_A(b,Z,y,A)
```

returns $A^\top [\Phi_*''(y)]^{-1} A$ directly. We also have the following functions to calculate different parts of the dual damped Newton step. The structure of the functions are almost the same, but there are small differences to make DDS more efficient.

```
hessian_Leg_A(Z,y,A)
hessian_Leg_inv_V_grad(b,Z,y)
hessian_Leg_inv_V(b,Z,y,v)
```

C.2. Generalized epigraphs of matrix norms. Let us see how to calculate the first and second derivatives of functions in (23) and (24).

Proposition C.1. (a) Consider $\Phi(X, U)$ defined in (23). Let, for simplicity, $\bar{X} := X - UU^\top$, then, we have

$$(81) \quad \begin{aligned} \Phi'(X, U)[(d_X, d_U)] &= \text{Tr}(-\bar{X}^{-1}d_X + \bar{X}^{-1}(d_U U^\top + U d_U^\top)), \\ \Phi''(X, U)[(d_X, d_U), (\bar{d}_X, \bar{d}_U)] &= \text{Tr}(\bar{X}^{-1}d_X \bar{X}^{-1}\bar{d}_X) \\ &\quad - \text{Tr}(\bar{X}^{-1}\bar{d}_X \bar{X}^{-1}(d_U U^\top + U d_U^\top)) \\ &\quad - \text{Tr}(\bar{X}^{-1}d_X \bar{X}^{-1}(\bar{d}_U U^\top + U \bar{d}_U^\top)) \\ &\quad + \text{Tr}(\bar{X}^{-1}(d_U U^\top + U d_U^\top) \bar{X}^{-1}(\bar{d}_U U^\top + U \bar{d}_U^\top)) \\ &\quad + 2\text{Tr}(\bar{X}^{-1}d_U \bar{d}_U^\top). \end{aligned}$$

(b) Consider $\Phi_*(Y, V)$ defined in (24), we have

$$\begin{aligned} \Phi'_*(Y, V)[(d_Y, d_V)] &= -\frac{1}{2}\text{Tr}(V^\top Y^{-1}d_V) + \frac{1}{4}\text{Tr}(Y^{-1}V V^\top Y^{-1}d_Y) - \text{Tr}(Y^{-1}d_Y), \\ \Phi''_*(Y, V)[(d_Y, d_V), (\bar{d}_Y, \bar{d}_V)] &= -\frac{1}{2}\text{Tr}(d_V^\top Y^{-1}\bar{d}_V) \\ &\quad + \frac{1}{2}\text{Tr}(Y^{-1}d_V V^\top Y^{-1}\bar{d}_Y) + \frac{1}{2}\text{Tr}(Y^{-1}\bar{d}_V V^\top Y^{-1}d_Y) \\ &\quad - \frac{1}{2}\text{Tr}(Y^{-1}d_Y Y^{-1}\bar{d}_Y Y^{-1}V V^\top) + \text{Tr}(Y^{-1}d_Y Y^{-1}\bar{d}_Y). \end{aligned}$$

Proof. For the proof we use the fact that if $g = -\ln(\det(X))$, then $g'(X)[H] = \text{Tr}(X^{-1}H)$. Also note that if we define

$$(82) \quad g(\alpha) := -\ln(\det(X + \alpha d_X - (U + \alpha d_U)(U + \alpha d_U)^\top)),$$

then

$$g'(0) = \Phi'(X, U)[(d_X, d_U)], \quad g''(0) = \Phi''(X, U)[(d_X, d_U), (d_X, d_U)],$$

and similarly for $\Phi_*(Y, V)$. We do not provide all the details, but we show how the proof works. For example, let us define

$$(83) \quad f(\alpha) := \text{Tr}((Y + \alpha d_Y)^{-1} V V^\top Y^{-1} d_Y),$$

and we want to calculate $f'(0)$. We have

$$\begin{aligned} f'(0) &:= \lim_{\alpha \rightarrow 0} \frac{f(\alpha) - f(0)}{\alpha} \\ &= \text{Tr} \left(\lim_{\alpha \rightarrow 0} \frac{(Y + \alpha d_Y)^{-1} V V^\top Y^{-1} d_Y - Y^{-1} V V^\top Y^{-1} d_Y}{\alpha} \right) \\ &= \text{Tr} \left(\lim_{\alpha \rightarrow 0} \frac{(Y + \alpha d_Y)^{-1} [V V^\top Y^{-1} d_Y - (I + \alpha d_Y Y^{-1}) V V^\top Y^{-1} d_Y]}{\alpha} \right) \\ &= \text{Tr} \left(\lim_{\alpha \rightarrow 0} (Y + \alpha d_Y)^{-1} [d_Y Y^{-1} V V^\top Y^{-1} d_Y] \right) \\ (84) \quad &= \text{Tr} \left(Y^{-1} d_Y Y^{-1} V V^\top Y^{-1} d_Y \right). \end{aligned}$$

□

Note that all the above formulas for the derivatives are in matrix form. Let us explain briefly how to convert them to the vector form for the code. We explain it for the derivatives of $\Phi(X, U)$ and the rest are similar. From (81) we have

$$\begin{aligned} \Phi'(X, U)[(d_X, d_U)] &= \text{Tr}(-\bar{X}^{-1} d_X) + \text{Tr}(\bar{X}^{-1} d_U U^\top) + \text{Tr}(X^{-1} U d_U^\top), \\ (85) \quad &= \text{Tr}(-\bar{X}^{-1} d_X) + 2\text{Tr}(U^\top \bar{X}^{-1} d_U). \end{aligned}$$

Hence, if g is the gradient of $\Phi(X, U)$ in the vector form, we have

$$(86) \quad g = \begin{bmatrix} 2 \times m2vec(X^{-1}U, n) \\ -sm2vec(X^{-1}) \end{bmatrix}.$$

The second derivatives are trickier. Assume that for example we want the vector form h for $\Phi''(X, U)[(d_X, d_U)]$. By using (81) we can easily get each entry of h ; consider the identity matrix of size $m^2 + mn$. If we choose (\bar{d}_X, \bar{d}_U) to represent the j th column of this identity matrix, we get $h(j)$. Practically, this can be done by a *for* loop, which is not efficient. What we did in the code is to implement this using matrix multiplication.

APPENDIX D. IMPLEMENTATION OF QUANTUM ENTROPY FUNCTION AND RELATED CONSTRAINTS

We want to continue by using first order optimality conditions for (45). Section 3.3 of the book [10] is about the derivation of matrix-values functions. For the first derivative, we have the following theorem:

Theorem D.1. *Let X and H be self-adjoint matrices and $f : (a, b) \mapsto \mathbb{R}$ be a continuously differentiable function defined on an interval. Assume that the eigenvalues of $X + \alpha H$ are in (a, b) for an interval around $\alpha_0 \in \mathbb{R}$. Then,*

$$(87) \quad \left. \frac{d}{dt} \text{Tr} f(X + \alpha H) \right|_{\alpha=\alpha_0} = \text{Tr} H f'(X + \alpha_0 H).$$

If we use the first-order optimality condition for (45) we get

$$(88) \quad \begin{aligned} \eta + \frac{1}{t - \phi(X)} &= 0 \\ Y + \frac{-f'(X)}{t - \phi(X)} + X^{-1} &= 0. \end{aligned}$$

If we substitute the first equation in the second one, we get

$$(89) \quad \frac{1}{\eta} Y + f'(X) + \frac{1}{\eta} X^{-1} = 0.$$

This equation implies that Y and X are simultaneously diagonalizable and if we have $Y = U \text{Diag}(\lambda_1(Y), \dots, \lambda_n(Y))$, then we have $X = U \text{Diag}(\lambda_1(X), \dots, \lambda_n(X))$ and so

$$(90) \quad \frac{1}{\eta} \lambda_i(Y) + f'(\lambda_i(X)) + \frac{1}{\eta \lambda_i(X)} = 0, \quad i \in \{1, \dots, n\}.$$

Here, we focus on the case that $f(x) = x \ln(x)$. This matrix function is related to quantum relative entropy and Von-Neumann entropy optimization problems (see [3] for a review of the applications). In this case, we can use results for type 3 univariate function in Table 2 and use the θ function we defined in (64). The LF conjugate of (45) is given in the following lemma:

Lemma D.1. *Assume that $f(x) = x \ln(x)$. For a given $\eta < 0$ and a symmetric matrix $Y \in \mathbb{S}^n$, the function defined in (45) becomes*

$$(91) \quad \Phi_*(\eta, Y) := -\ln(-\eta) + \text{Tr}(\theta + \theta^{-1}) - \text{Tr}\left(\frac{1}{\eta} Y\right) - 1 - 2n,$$

where $\theta := \theta\left(\frac{1}{\eta} Y + (1 - \ln(-\eta))I\right)$ for the function θ defined in (64).

Proof. Assume that for a given (η, Y) , (t, X) is the optimal solution for (45). If we use theorem D.1, we have $f'(X) = I = \ln(X)$. By substituting this in the first order optimality condition (89) we get

$$(92) \quad \eta X = \theta\left(\frac{1}{\eta} Y + (1 - \ln(-\eta))I\right).$$

Note that we have

$$\begin{aligned}
\eta t = -1 + \text{Tr}(\eta X \ln(X)) &= -1 + \text{Tr}(-\theta \cdot \ln(X)) \\
&= -1 + \text{Tr}\left(\theta \cdot \left(\frac{1}{\eta} Y - I + \theta^{-1}\right)\right). \\
(93) \qquad \qquad \qquad &= -1 - n + \text{Tr}(Y/\eta) + \text{Tr}(\theta).
\end{aligned}$$

If we substitute t and X in (45), we get the result. \square

To implement our primal-dual techniques, we need the gradient and Hessian of $\Phi(t, X)$ and $\Phi_*(\eta, Y)$. We already saw in Theorem D.1 how to calculate the gradient. The following theorem gives us a tool to calculate the Hessian.

Theorem D.2 ([10]-Theorem 3.25). *Assume that $f : (a, b) \mapsto \mathbb{R}$ is a \mathcal{C}^1 -function and $T = \text{Diag}(t_1, \dots, t_n)$ with $t_i \in (a, b)$, $i \in \{1, \dots, n\}$. Then, for a Hermitian matrix H , we have*

$$(94) \qquad \qquad \qquad \left. \frac{d}{dt} f(T + \alpha H) \right|_{\alpha=0} = T_f \odot H,$$

where \odot is the Hadamard product and T_f is the divided difference matrix:

$$(95) \qquad \qquad \qquad T_f := \begin{cases} \frac{f(t_i) - f(t_j)}{t_i - t_j} & t_i \neq t_j \\ f'(t_i) & t_i = t_j \end{cases}.$$

Note that T is diagonal in the statement of the theorem, which is without loss of generality. Note that by the definition of functional calculus in (43), for a Hermitian matrix X and a unitary matrix U , we have

$$(96) \qquad \qquad \qquad f(UXU^*) = Uf(X)U^*.$$

Therefore, for a matrix $T = U\text{Diag}(t_1, \dots, t_n)U^*$, we can update (94)

$$(97) \qquad \qquad \qquad \left. \frac{d}{dt} f(T + \alpha H) \right|_{\alpha=0} = U(T_f \odot (U^* H U))U^*,$$

where we extend the definition of T_f in (95) to non-diagonal matrices. Now we can use Theorems D.2 and D.1 to calculate the Hessian of a matrix function.

Corollary D.1. *Let X , H , and \tilde{H} be self-adjoint matrices and $f : (a, b) \mapsto \mathbb{R}$ be a continuously differentiable function defined on an interval. Assume that the eigenvalues of $X + tH$ and $X + t\tilde{H}$ are in (a, b) for an interval around $t = 0$. Assume that $X = U\text{Diag}(\lambda_1, \dots, \lambda_n)U^*$. Then,*

$$(98) \qquad \qquad \qquad f''(X)[H, \tilde{H}] = \text{Tr}\left((X_f \odot (U^* H U))U^* \tilde{H} U\right).$$

Let us calculate the gradient and Hessian for our functions for $\phi(x) = x \ln(x)$. Let $X = U\text{Diag}(\lambda_1, \dots, \lambda_n)U^*$ in the following.

$$\Phi'(t, X)[(h, H)] = -\frac{h}{t - \text{Tr}(X \ln X)} + \frac{1}{t - \text{Tr}(X \ln X)} \text{Tr}((I + \ln(X))H) - \text{Tr}(X^{-1}H).$$

For the second derivative, we can use the fact that

$$\Phi''(t, X)[(\tilde{h}, \tilde{H}), (h, H)] = \Phi'(t + \alpha\tilde{h}, X + \alpha\tilde{H}) \Big|_{\alpha=0} [(h, H)].$$

Using this formula, we have ($\zeta := \frac{1}{t - \text{Tr}(X \ln X)}$)

$$\begin{aligned} \Phi''(t, X)[(\tilde{h}, \tilde{H}), (h, H)] &= \zeta^2 h \tilde{h} \\ &\quad - \zeta^2 \tilde{h} \text{Tr}((I + \ln(X))H) - \zeta h \text{Tr}((I + \ln(X))\tilde{H}) \\ &\quad + \zeta^2 \text{Tr}((I + \ln(X))\tilde{H}) \text{Tr}((I + \ln(X))H) \\ &\quad + \zeta \text{Tr} \left(U \left(X_{\ln} \odot (U^* \tilde{H} U) \right) U^* H \right) \\ &\quad + \text{Tr}(X^{-1} \tilde{H} X^{-1} H). \end{aligned}$$

Now let us compute the gradient and Hessian for the conjugate function. Let $Y = U \lambda(Y) U^*$, by using Theorem D.1, the gradient of $\Phi_*(\eta, Y)$ is

$$\begin{aligned} \Phi'_*(\eta, Y)[(h, H)] &= h \left[-\frac{1}{\eta} + \text{Tr} \left(\left(-\frac{1}{\eta^2} Y - \frac{1}{\eta} I \right) \left(\theta' - \frac{\theta'}{\theta^2} \right) + \frac{1}{\eta^2} Y \right) \right] \\ &\quad + \text{Tr} \left(H \left(\frac{1}{\eta} \left(\theta' - \frac{\theta'}{\theta^2} \right) - \frac{1}{\eta} I \right) \right). \end{aligned}$$

For the second derivative, let us first define

$$\bar{Y} := \left(\frac{1}{\eta} Y + (1 - \ln(-\eta)) I \right)_{\left(\theta' - \frac{\theta'}{\theta^2} \right)}.$$

Then, we have

$$\begin{aligned} \Phi''_*(\eta, Y)[(\tilde{h}, \tilde{H}), (h, H)] &= \\ &h \tilde{h} \left[\frac{1}{\eta^2} + \text{Tr} \left(\left(\frac{2}{\eta^3} Y + \frac{1}{\eta^2} I \right) \left(\theta' - \frac{\theta'}{\theta^2} \right) + \left(-\frac{1}{\eta^2} Y - \frac{1}{\eta} I \right)^2 \left(\theta'' - \frac{\theta'' \theta - 2(\theta')^2}{\theta^3} \right) - \frac{2}{\eta^3} Y \right) \right] \\ &+ \tilde{h} \text{Tr} \left(H \left[\frac{-1}{\eta^2} \left(\theta' - \frac{\theta'}{\theta^2} \right) + \frac{1}{\eta} U \left(\bar{Y} \odot \left(\frac{-1}{\eta^2} \lambda(Y) - \frac{1}{\eta} I \right) \right) U^* + \frac{1}{\eta^2} I \right] \right) \\ &+ h \text{Tr} \left(\tilde{H} \left[\frac{-1}{\eta^2} \left(\theta' - \frac{\theta'}{\theta^2} \right) + \frac{1}{\eta} U \left(\bar{Y} \odot \left(\frac{-1}{\eta^2} \lambda(Y) - \frac{1}{\eta} I \right) \right) U^* + \frac{1}{\eta^2} I \right] \right) \\ &+ \frac{1}{\eta^2} \text{Tr} \left(U \left(\bar{Y} \odot (U^* \tilde{H} U) \right) U^* H \right). \end{aligned}$$

APPENDIX E. CALCULATING LEGENDRE-FENCHEL CONJUGATE AND DERIVATIVES

In this section, we elaborate more on calculating the Legendre-Fenchel conjugate of s.c. barriers we discussed. We have implicit formulas for some of the conjugate functions. In this section, we also give the explicit formulas for some of the first and second derivatives. Let us start by the following lemma:

Lemma E.1. *Assume that $\Phi(z)$ is a s.c.b. and let $\Phi_*(y)$ be its Legendre-Fenchel conjugate. Then the Legendre-Fenchel conjugate of $\Phi(z+b)$ is $-\langle y, b \rangle + \Phi_*(y)$.*

For LP and SOCP we have

$$(99) \quad \begin{aligned} \Phi(z) &= -\ln(z), \quad z \in \mathbb{R}_+, & \Phi_*(\eta) &= -1 - \ln(-\eta), \\ \Phi(t, z) &= -\ln(t^2 - z^\top z), & \Phi_*(\eta, w) &= -2 + \ln(4) - \ln(\eta^2 - w^\top w). \end{aligned}$$

For quadratic constraints, we explained how to use the following pair of functions:

$$(100) \quad \begin{aligned} \Phi(u, w) &= -\ln(-(u^\top u + w + d)), \\ \Phi_*(y, \eta) &= \frac{y^\top y}{4\eta} - 1 - d\eta - \ln(\eta). \end{aligned}$$

For the first and second derivatives of Φ we have

$$\nabla\Phi = \frac{1}{u^\top u + w + d} \begin{bmatrix} -2u \\ -1 \end{bmatrix}, \quad \nabla^2\Phi = \frac{1}{(u^\top u + w + d)^2} \begin{bmatrix} -2(u^\top u + w + d) + 4uu^\top & 2u^\top \\ 2u & 1 \end{bmatrix},$$

and for the first and second derivatives of Φ_* we have

$$\nabla\Phi_* = \begin{bmatrix} \frac{y}{2\eta} \\ -d - \frac{1}{\eta} - \frac{1}{4\eta^2} y^\top y \end{bmatrix}, \quad \nabla^2\Phi_* = \begin{bmatrix} \frac{1}{2\eta} I & -\frac{1}{2\eta^2} y^\top \\ -\frac{1}{2\eta^2} y & \frac{1}{\eta^2} + \frac{1}{2\eta^3} y^\top y \end{bmatrix}.$$

E.1. Constraints defined by the 2-dimensional convex sets. We discussed five types of constraints based on 2-dimensional convex sets. In this section, we show more details about the formulas. First consider the three pairs of functions in Table 5. Here are the explicit formulas for the first and second derivatives:

$\Phi(z, t)$	$\Phi_*(y, \eta)$
$-\ln(t + \ln(z)) - \ln(z)$	$-1 + (-\eta + 1) \left[-1 + \ln \frac{-(-\eta+1)}{y} \right] - \ln(-\eta)$

For the primal function we have

$$\nabla\Phi = \begin{bmatrix} -\frac{1}{z} \left(\frac{1}{t+\ln(z)} + 1 \right) \\ -\frac{1}{t+\ln(z)} \end{bmatrix}, \quad \nabla^2\Phi = \begin{bmatrix} \frac{1}{z^2} \left(\frac{1}{t+\ln(z)} + \frac{1}{(t+\ln(z))^2} + 1 \right) & \frac{1}{z(t+\ln(z))^2} \\ \frac{1}{z(t+\ln(z))^2} & \frac{1}{(t+\ln(z))^2} \end{bmatrix},$$

and for the dual function we have

$$\nabla\Phi_* = \begin{bmatrix} -\frac{-\eta+1}{y} \\ -\ln \left(-\frac{-\eta+1}{y} \right) - \frac{1}{\eta} \end{bmatrix}, \quad \nabla^2\Phi_* = \begin{bmatrix} \frac{-\eta+1}{y^2} & \frac{1}{y} \\ \frac{1}{y} & \frac{1}{-\eta+1} + \frac{1}{\eta^2} \end{bmatrix}.$$

$\Phi(z, t)$	$\Phi_*(y, \eta)$
$-\ln(\ln(t) - z) - \ln(t)$	$-1 + (y + 1) \left[-1 + \ln \frac{-(y+1)}{\eta} \right] - \ln(y)$

For the primal function we have

$$\nabla \Phi = \begin{bmatrix} \frac{1}{\ln(t)-z} \\ \frac{1}{t} \left(\frac{1}{\ln(t)-z} + 1 \right) \end{bmatrix}, \quad \nabla^2 \Phi = \frac{1}{(\ln(t) - z)^2} \begin{bmatrix} 1 & -\frac{1}{t} \\ -\frac{1}{t} & \frac{1 + (\ln(t)-z) + (\ln(t)-z)^2}{t^2} \end{bmatrix},$$

and for the dual function we have

$$\nabla \Phi_* = \begin{bmatrix} \ln \left(\frac{-(y+1)}{\eta} \right) - \frac{1}{y} \\ -\frac{y+1}{\eta} \end{bmatrix}, \quad \nabla^2 \Phi_* = \begin{bmatrix} \frac{1}{y+1} + \frac{1}{y^2} & -\frac{1}{\eta} \\ -\frac{1}{\eta} & \frac{y+1}{\eta^2} \end{bmatrix}.$$

$\Phi(z, t)$	$\Phi_*(y, \eta)$
$-\ln(t - z \ln(z)) - \ln(z)$	$-\ln(-\eta) + \theta \left(1 + \frac{y}{\eta} - \ln(-\eta) \right) - \frac{y}{\eta} + \frac{1}{\theta \left(1 + \frac{y}{\eta} - \ln(-\eta) \right)} - 3$

For the primal function we have

$$\nabla \Phi = \begin{bmatrix} \frac{\ln(z)+1}{t-z\ln(z)} - \frac{1}{z} \\ \frac{-1}{t-z\ln(z)} \end{bmatrix}, \quad \nabla^2 \Phi = \begin{bmatrix} \frac{(t-z\ln(z)) + (\ln(z)+1)^2}{z(t-z\ln(z))^2} + \frac{1}{z^2} & \frac{-(\ln(z)+1)}{(t-z\ln(z))^2} \\ \frac{-(\ln(z)+1)}{(t-z\ln(z))^2} & \frac{1}{(t-z\ln(z))^2} \end{bmatrix}.$$

For the dual function, since the argument of the function $\theta(\cdot)$ is always $1 + \frac{y}{\eta} - \ln(-\eta)$, we ignore that in the following formulas and use θ , θ' , and θ'' for the function and its derivative.

$$\nabla \Phi_* = \begin{bmatrix} \frac{\theta'-1}{\eta} - \frac{\theta'}{\eta\theta^2} \\ -\frac{1}{\eta} + \frac{y}{\eta^2} - \left(\frac{y}{\eta^2} + \frac{1}{\eta} \right) \theta' \left(1 + \frac{1}{\theta^2} \right) \end{bmatrix}, \quad \nabla^2 \Phi_* = \begin{bmatrix} f_{11} & f_{12} \\ f_{21} & f_{22} \end{bmatrix},$$

where

$$\begin{aligned} f_{11} &= \frac{1}{\eta^2} \theta'' - \frac{\theta''\theta - 2(\theta')^2}{\eta^2 \theta^3}, \\ f_{21} = f_{12} &= -\frac{1}{\eta^2} \theta' + \frac{1}{\eta} \left(-\frac{y}{\eta^2} - \frac{1}{\eta} \right) \theta'' + \frac{1}{\eta^2} - \frac{\left[-\frac{1}{\eta^2} \theta' + \frac{1}{\eta} \left(-\frac{y}{\eta^2} - \frac{1}{\eta} \right) \theta'' \right] \theta - \frac{2}{\eta} \left(-\frac{y}{\eta^2} - \frac{1}{\eta} \right) (\theta')^2}{\theta^3}, \\ f_{22} &= \frac{1}{\eta^2} - \frac{2y}{\eta^3} + \left[\left(\frac{2y}{\eta^3} + \frac{1}{\eta^2} \right) \theta' + \left(-\frac{y}{\eta^2} - \frac{1}{\eta} \right)^2 \theta'' \right] \left(1 + \frac{1}{\theta^2} \right) + \left(-\frac{y}{\eta^2} - \frac{1}{\eta} \right)^2 \frac{2(\theta')^2}{\theta^3} \end{aligned}$$

$\Phi(z, t)$	$\Phi_*(y, \eta)$
$-\ln(t^{\frac{2}{p}} - z^2) - 2\ln(t)$	$-\left(\frac{2}{p} + \left(\frac{1}{p} - 1 \right) yz \right) - 2 + 2\ln \left(\frac{2(\frac{1}{p}+1) + \frac{1}{p} yz}{-\eta} \right) + \ln \left(\left(\frac{2(\frac{1}{p}+1) + \frac{1}{p} yz}{-\eta} \right)^{\frac{2}{p}} - z^2 \right)$

where $z(y, \eta)$ is the solution of

$$(101) \quad y \left(\frac{2(\frac{1}{p} + 1) + \frac{1}{p}yz}{-\eta} \right)^{\frac{2}{p}} - yz^2 - 2z = 0.$$

For simplicity, we drop the arguments of $z(y, \eta)$ and denote it as z . We denote the first derivatives with respect to y and η as z'_y and z'_η , respectively. Similarly, we use z''_{yy} , $z''_{\eta y}$, and $z''_{\eta\eta}$ for the second derivatives. We have

$$(102) \quad \begin{aligned} z'_y &= \frac{B^{\frac{2}{p}} - \frac{2y}{p^2\eta} x B^{\frac{2}{p}-1} - z^2}{\frac{2y^2}{p^2\eta} B^{\frac{2}{p}-1} + 2yz + 2} =: S \\ z'_\eta &= \frac{\frac{-2y}{p\eta} B^{\frac{2}{p}}}{\frac{2y^2}{p^2\eta} B^{\frac{2}{p}-1} + 2yz + 2} =: T \\ B &:= \frac{2(\frac{1}{p} + 1) + \frac{1}{p}yz}{-\eta} \end{aligned}$$

For calculating the second derivatives of Φ_* , we need the derivatives of B :

$$(103) \quad \begin{aligned} B'_y &= \frac{z + yz'_y}{-p\eta}, \\ B'_\eta &= \frac{-\frac{1}{p}yz'_\eta + 2(\frac{1}{p} + 1) + \frac{1}{p}yz}{\eta^2}. \end{aligned}$$

Then we have

$$S'_y = \frac{2}{p} B'_y B^{\frac{2}{p}-1} + \left(\frac{-2z}{p^2\eta} - \frac{2y}{p^2\eta} z'_y \right) B^{\frac{2}{p}-1} + \frac{-2yz}{p^2\eta} \left(\frac{2}{p} - 1 \right) B'_y B^{\frac{2}{p}-2} - 2zz'_y,$$

$$S'_\eta = \frac{2}{p} B'_\eta B^{\frac{2}{p}-1} + \frac{-2y}{p^2} \left(\frac{\eta z'_\eta - z}{\eta^2} \right) B^{\frac{2}{p}-1} + \frac{-2yz}{p^2\eta} \left(\frac{2}{p} - 1 \right) B'_\eta B^{\frac{2}{p}-2} - 2zz'_\eta,$$

$$M'_y = \frac{4y}{p^2\eta} B^{\frac{2}{p}-1} + \frac{2y^2}{p^2\eta} \left(\frac{2}{p} - 1 \right) B'_y B^{\frac{2}{p}-2} + 2z + 2yz'_y,$$

$$M'_\eta = -\frac{2y^2}{p^2\eta^2} B^{\frac{2}{p}-1} + \frac{2y^2}{p^2\eta} \left(\frac{2}{p} - 1 \right) B'_\eta B^{\frac{2}{p}-2} + 2yz'_\eta,$$

$$T'_\eta = \frac{2y}{p\eta^2} B^{\frac{2}{p}} + \frac{-4y}{p^2\eta} B'_\eta B^{\frac{2}{p}-1}.$$

By the above definitions of S , M , and T , we have

$$z''_{yy} = \frac{S'_y M - M'_y S}{M^2}, \quad z''_{\eta y} = \frac{S'_\eta M - M'_\eta S}{M^2}, \quad z''_{\eta\eta} = \frac{T'_\eta M - M'_\eta T}{M^2}.$$

The first and second derivatives of Φ are calculated as follows:

$$(104) \quad \begin{aligned} \nabla\Phi &= \begin{bmatrix} \frac{2z}{t^{\frac{2}{p}} - z^2} \\ \frac{-\frac{2}{p}t^{\frac{2}{p}-1}}{t^{\frac{2}{p}} - z^2} - \frac{2}{t} \end{bmatrix}, \\ \nabla^2\Phi &= \begin{bmatrix} \frac{2(t^{\frac{2}{p}} - z^2) + 4z^2}{(t^{\frac{2}{p}} - z^2)^2} & \frac{-\frac{4}{p}t^{\frac{2}{p}-1}z}{(t^{\frac{2}{p}} - z^2)^2} \\ \frac{-\frac{4}{p}t^{\frac{2}{p}-1}z}{(t^{\frac{2}{p}} - z^2)^2} & \frac{-\frac{2}{p}(\frac{2}{p}-1)t^{\frac{2}{p}-2}(t^{\frac{2}{p}} - z^2) + (\frac{2}{p})^2 t^{\frac{4}{p}-2}}{(t^{\frac{2}{p}} - z^2)^2} + \frac{2}{t^2} \end{bmatrix}. \end{aligned}$$

The first and second derivatives of Φ_* are messier. For the first derivative we have

$$(105) \quad \nabla\Phi_* = \begin{bmatrix} -\left(\frac{1}{p} - 1\right)(z + yz'_y) + \frac{\frac{2}{p}B'_y B^{\frac{2}{p}-1} - 2zz'_y}{B^{\frac{2}{p}} - z^2} + \frac{2B'_y}{B} \\ -\left(\frac{1}{p} - 1\right)(yz'_\eta) + \frac{\frac{2}{p}B'_\eta B^{\frac{2}{p}-1} - 2zz'_\eta}{B^{\frac{2}{p}} - z^2} + \frac{2B'_\eta}{B} \end{bmatrix},$$

For calculating the second derivative, we also need the second derivatives of B :

$$\begin{aligned} B''_{yy} &= \frac{2z'_y + z''_{yy}}{-p\eta}, \\ B''_{y\eta} &= \frac{-p\eta(z'_\eta + yz''_{\eta y}) + p(z + yz'_y)}{(p\eta)^2}, \\ B''_{\eta\eta} &= -\frac{yz''_{\eta\eta}\eta - yz'_\eta}{\eta^2} - \left(\frac{1}{p} + 1\right)\frac{4}{\eta^3} + \frac{1}{p}\frac{yz'_\eta\eta - zy}{\eta^2}. \end{aligned}$$

Using the second derivatives of B , we have

$$(106) \quad \nabla^2\Phi_* = \begin{bmatrix} f_{11} & f_{12} \\ f_{21} & f_{22} \end{bmatrix},$$

where

$$\begin{aligned} f_{11} &= -\left(\frac{1}{p} - 1\right)(2z'_y + yz''_{yy}) + \frac{\left[\frac{2}{p}\left[B''_{yy}B^{\frac{2}{p}-1} + \left(\frac{2}{p} - 1\right)(B'_y)^2B^{\frac{2}{p}-2}\right] - 2((z'_y)^2 + z z''_{yy})\right](B^{\frac{2}{p}} - z^2)}{(B^{\frac{2}{p}} - z^2)^2} \\ &\quad - \frac{\left[\frac{2}{p}B'_y B^{\frac{2}{p}-1} - 2zz'_y\right]^2}{(B^{\frac{2}{p}} - z^2)^2} + \frac{2B''_{yy}B - 2(B'_y)^2}{B^2}. \end{aligned}$$

$f_{21} = f_{12}$ and f_{22} have similar formulations.

$\Phi(z, t)$	$\Phi_*(y, \eta)$
$-\ln(z^p + t) - \ln(z)$	$\eta(p-1)z^p(y, \eta) - 2 - \ln(-\eta) + \ln(z(y, \eta))$

where z is the solution of

$$(107) \quad y - \eta p z^{(p-1)} + \frac{1}{z} = 0$$

Similar to the previous case, for simplicity, we drop the arguments of $z(y, \eta)$ and denote it as z . We denote the first derivatives with respect to y and η as z'_y and z'_η , respectively. By implicit differentiation, we have

$$\begin{aligned} z'_y &= \frac{1}{\eta p(p-1)z^{p-2} + z^{-2}} =: B, \\ z'_\eta &= \frac{-pz^{p-1}}{\eta p(p-1)z^{p-2} + z^{-2}}. \end{aligned}$$

For the second derivatives of z , by using

$$\begin{aligned} B'_y &= \eta p(p-1)(p-2)z'_y z^{p-3} - 2z'_y z^{-3}, \\ B'_\eta &= p(p-1)z^{p-2} + \eta p(p-1)(p-2)z'_\eta z^{p-3} - 2z'_\eta z^{-3}. \end{aligned}$$

we have

$$z''_{yy} = \frac{-B'_y}{B^2}, \quad z''_{\eta y} = \frac{-B'_\eta}{B^2}, \quad z''_{\eta\eta} = \frac{-p(p-1)z'_\eta z^{p-2} B + pz^{p-1} B'_\eta}{B^2}.$$

The first and second derivatives of Φ are calculated as follows:

$$\begin{aligned} \nabla \Phi &= \begin{bmatrix} -\frac{pz^{p-1}}{z^p+t} - \frac{1}{x} \\ -\frac{1}{z^p+t} \end{bmatrix}, \\ \nabla^2 \Phi &= \begin{bmatrix} -p(p-1)z^{p-2}(z^p+t) + p^2 z^{2p-2} + \frac{1}{z^2} & \frac{pz^{p-1}}{(z^p+t)^2} \\ \frac{pz^{p-1}}{(z^p+t)^2} & \frac{1}{(z^p+t)^2} \end{bmatrix}. \end{aligned}$$

The first derivative of Φ_* is equal to

$$(108) \quad \nabla \Phi_* = \begin{bmatrix} \eta p(p-1)z'_y x^{p-1} + \frac{z'_y}{z} \\ (p-1)z^p + \eta p(p-1)z'_\eta z^{p-1} - \frac{1}{\eta} + \frac{z'_\eta}{z} \end{bmatrix},$$

and the second derivatives of Φ_* is equal to

$$(109) \quad \nabla^2 \Phi_* = \begin{bmatrix} f_{11} & f_{12} \\ f_{21} & f_{22} \end{bmatrix},$$

where

$$\begin{aligned}
f_{11} &= \eta(p-1)p [z'_{yy}x^{p-1} + (p-1)(z'_y)^2z^{p-2}] + \frac{z'_{yy}z - (z'_y)^2}{z^2}, \\
f_{12} = f_{21} &= (p-1)p [z'_yz^{p-1} + \eta z'_{y\eta}z^{p-1} + \eta(p-1)z'_yz'_\eta z^{p-2}] + \frac{z'_{y\eta}z - z'_\eta z'_y}{z^2}, \\
f_{22} &= (p-1)p z'_\eta z^{p-1} + (p-1)p [z'_\eta z^{p-1} + \eta z'_{\eta\eta}z^{p-1} + \eta(z'_\eta)^2(p-1)z^{p-2}] + \frac{1}{\eta^2} + \frac{z'_{\eta\eta}z - (z'_\eta)^2}{z^2}.
\end{aligned}$$

$\Phi(z, t)$	$\Phi_*(y, \eta)$
$-\ln(zt - 1)$	$-1 - \sqrt{1 + 4y\eta} + \ln\left(\frac{1 + \sqrt{1 + 4y\eta}}{2y\eta}\right)$

For the primal function we have

$$\nabla\Phi = \begin{bmatrix} -\frac{t}{zt-1} \\ -\frac{z}{zt-1} \end{bmatrix}, \quad \nabla^2\Phi = \begin{bmatrix} \frac{t^2}{(zt-1)^2} & \frac{1}{(zt-1)^2} \\ \frac{1}{(zt-1)^2} & \frac{z^2}{(zt-1)^2} \end{bmatrix},$$

and for the dual function we have

$$\nabla\Phi_* = \begin{bmatrix} -\frac{2\eta}{1 + \sqrt{1 + 4y\eta}} - \frac{1}{y} \\ -\frac{2y}{1 + \sqrt{1 + 4y\eta}} - \frac{1}{\eta} \end{bmatrix}, \quad \nabla^2\Phi_* = \begin{bmatrix} \frac{4\eta^2}{\sqrt{1 + 4y\eta}(1 + \sqrt{1 + 4y\eta})^2} + \frac{1}{y^2} & \frac{-2(\sqrt{1 + 4y\eta} + 1 + 4y\eta) + 4y\eta}{\sqrt{1 + 4y\eta}(1 + \sqrt{1 + 4y\eta})^2} \\ \frac{-2(\sqrt{1 + 4y\eta} + 1 + 4y\eta) + 4y\eta}{\sqrt{1 + 4y\eta}(1 + \sqrt{1 + 4y\eta})^2} & \frac{4y^2}{\sqrt{1 + 4y\eta}(1 + \sqrt{1 + 4y\eta})^2} + \frac{1}{\eta^2} \end{bmatrix}.$$

APPENDIX F. FORMAT OF THE INPUT FOR SOME OTHER SOLVERS

In this section, we take a look at the input format for some other well-known solvers. [14] is a survey by Mittelmann about solvers for conic optimization, which gives an overview of the major codes available for the solution of linear semidefinite (SDP) and second-order cone (SOCP) programs. Many of these codes also solve linear programs (LP). We mention the leaders MOSEK, SDPT3, and SeDuMi from the list. We also look at CVX, a very user-friendly interface for convex optimization. CVX is not a solver, but is a modeling system that (by following some rules) detects if a given problem is convex and remodels it as a suitable input for solvers such as SeDuMi.

F.1. MOSEK [15]. MOSEK is a leading commercial solver for not just optimization over symmetric cones, but also many other convex optimization problems. The most recent version, MOSEK 9.0, for this state-of-the-art convex optimization software handles, in a primal-dual framework, all convex cone constraints which arise in applications [4]. There are different options for the using platform that can seen in MOSEK's website [15].

F.2. **SDPT3** [23, 22]. SDPT3 is a MATLAB package for optimization over symmetric cones, and it solves a conic optimization problem in the equality form as

$$(110) \quad \begin{aligned} \min \quad & \langle c, x \rangle \\ \text{s.t.} \quad & Ax = b, \\ & x \in K, \end{aligned}$$

where our cone K can be a direct sum of nonnegative rays (leading to LP problems), second-order cones or semidefinite cones.

The Input for SDPT3 is given in the cell array structure of MATLAB. The command to solve SDPT3 is of the form

```
[obj,X,y,Z,info,runhist] = sglp(blk,At,C,b,OPTIONS,X0,y0,Z0).
```

The input data is given in different blocks, where for the k th block, `blk{k,1}` specifies the type of the constraint. Letters 'l', 'q', and 's' are representing linear, quadratic, and semidefinite constraints. In the k th block, `At{k}`, `C{k}`, ... contain the part of the input related to this block.

F.3. **SeDuMi** [21]. SeDuMi is also a MATLAB package for optimization over symmetric cones in the format of (110). For SeDuMi, we give as the input A , b and c and a structure array K . The vector of variables has a "direct sum" structure. In other words, the set of variables is the direct sum of free, linear, quadratic, or semidefinite variables. The fields of the structure array K contain the number of constraints we have from each type and their sizes. SeDuMi can be called in MATLAB by the command

```
[x,y] = sedumi(A,b,c,K);
```

and the variables are distinguished by K as follows:

- (1) $K.f$ is the number of free variables, i.e., in the variable vector x , $x(1:K.f)$ are free variables.
- (2) $K.l$ is the number of nonnegative variables.
- (3) $K.q$ lists the dimension of Lorentz constraints.
- (4) $K.s$ lists the dimensions of positive semidefinite constraints.

For example, if $K.l=10$, $K.q=[3 \ 7]$ and $K.s=[4 \ 3]$, then $x(1:10)$ are non-negative. Then we have $x(11) \geq \text{norm}(x(12:13))$, $x(14) \geq \text{norm}(x(15:20))$, and $\text{mat}(x(21:36), 4)$ and $\text{mat}(x(37:45), 3)$ are positive semidefinite. To insert our problem into SeDuMi, we have to write it in the format of (110). We also have the choice to solve the dual problem because all of the above cones are self-dual.

F.4. **CVX** [8]. CVX is an interface that is more user-friendly than solvers like SeDuMi. It provides many options for giving the problem as an input, and then translates them to an eligible format for a solver such as SeDuMi. We can insert our problem constraint-by-constraint into CVX, but they must follow a protocol called *Disciplined convex programming* (DCP). DCP has a rule-set that the user has to follow, which allows CVX to verify that the problem is convex and convert it to a solvable form. For example, we can write a \leq constraint only when the left side is convex and the right side is concave, and to do that, we can use a large class of functions from the library of CVX.

Another advantage of CVX is that we do not have to be worried about the structure of the variables, and instead we can input our problem in a more natural way. For example, consider the following problem:

$$\min\{\|Ax - b\|_2 : l \leq x \leq u\}.$$

We can insert this problem in CVX as:

```
cvx_begin
    variable x(n);
    minimize( norm(A*x-b) );
    subject to
        x >= l;
        x <= u;
cvx_end
```

However, to feed it into SeDuMi, we have to do some modification, for example:

$$(111) \quad \begin{aligned} \min \quad & t \\ \text{s.t.} \quad & y = Ax - b, \\ & \bar{x} = x - l, \\ & \hat{x} = u - x, \\ & \|y\| \leq t, \quad \bar{x}, \hat{x} \geq 0. \end{aligned}$$