DOMAIN-DRIVEN SOLVER (DDS) VERSION 2.0: 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 [19]. The current version of DDS accepts every combination of the following function/set constraints: (1) symmetric cones (LP, SOCP, and SDP); (2) quadratic constraints that are SOCP representable; (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) generalized power cone; (5) epigraphs of matrix norms (including as a special case minimization of nuclear norm over a linear subspace); (6) vector relative entropy; (7) epigraphs of quantum entropy and quantum relative entropy; and (8) constraints involving hyperbolic polynomials. DDS is a practical implementation of the infeasible-start primal-dual algorithm designed and analyzed in [19]. This manuscript contains the users' guide, as well as theoretical results needed for the implementation of the algorithms. To help the users, we included many examples. We also discussed some implementation details and techniques we used to improve the efficiency and further expansion of the software to cover the emerging classes of convex optimization problems.

Date: August 7, 2019 (arXiv: 1908.03075); revised: November 20, 2020.

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

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

$$\inf_x \{\langle c, x\rangle : Ax \in D\}, \tag{1}$$

where $x \mapsto Ax : \mathbb{R}^n \to \mathbb{R}^m$ 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 [27, 26]. 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 $i \in \{1, \ldots, \ell\}$. 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 to evaluate Φ_* and its first (and hopefully second) derivative. 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 [19], 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 [20].

Even though there are similarities between DDS and some modeling systems such as CVX [16] (such as the variety input constraints), there are major differences, including:

- DDS is not just a modeling system and it uses its own algorithm. The algorithm used in DDS is an infeasible-start primal-dual path-following algorithm, and is of predictor corrector type [19].
- The modeling systems for convex optimization that rely on SDP solvers have to use approximation for set constraints which are not efficiently representable by spectrahedra (for example epigraph of functions involving *exp* or ln functions). However, DDS uses a s.c. barrier specifically well-suited to each set constraint without such approximations. This enables DDS to return proper certificates for all the input problems.
- As far as we know, some set constraints such as hyperbolic ones, are not accepted by other modeling systems. Some other constraints such as epigraphs of matrix norms, and those

involving quantum entropy and quantum relative entropy are handled more efficiently than other existing options.

The main part of the article is organized to be more as a users' guide, and contains the minimum theoretical content required by the users. In Section 2, we explain how to install and use DDS. Sections 3-11 explain how to input different types of set/function constraints into DDS. Section 13 contains several numerical results and tables. Many theoretical foundations needed by DDS are included in the appendices. 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. Some of the constraints accepted by DDS are involving matrix functions and many techniques are used in DDS to efficiently evaluate these functions and their derivatives, see Appendices B and C. DDS heavily relies on efficient evaluation of LF conjugates of s.c. barrier functions. For some of the functions, we show in the main text how to efficiently calculate the LF conjugate. For some others, the calculations are shifted to the appendices, for example Appendix D. DDS uses interior-point methods, and gradient and Hessian of the s.c. barriers are building blocks of the underlying linear systems. Explicit formulas for the gradient and Hessian of some s.c. barriers and their LF conjugates are given in Appendix D.2. Appendix F contains a brief comparison of DDS with some popular solvers and modeling systems.

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

http://www.math.uwaterloo.ca/~m7karimi/DDS.html
https://github.com/mehdi-karimi-math/DDS

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`.
- (optional) run the prepared small examples `DDS_example_1.m` and `DDS_example_2.m`.

The prepared examples contain many set constraints accepted by DDS and running them without error indicates that DDS is ready to use. There is a directory `Text_Examples` in the DDS package which includes many examples on different classes of convex optimization problems.

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 and initial points.

Output Arguments:
`x`: Primal point.
`y`: Dual point which is a cell array. Each cell contains the dual solution for the constraints in the corresponding cell in `A`.
`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:

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

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

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

where $\text{rec}(D)$ is the *recession cone* of D. For the convenience of users, there is a built-in function in DDS package to calculate the dual objective value of the returned y vector:

```
z=dual_obj_value(y,b,cons);
```

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 [19] as

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

It is proved in [19] 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 ($\langle c, x\rangle \leq -1/tol$). 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 $D = D_1 \oplus \cdots \oplus D_\ell$, the ith 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 sections, 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. 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. `cons(k,1)` represents type of the kth block of constraints by using the abbreviations of Table 1. For example, `cons(2,1)='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. LP, SOCP, SDP

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

$$A_L^i x + b_L^i \geq 0, \quad i \in \{1, \ldots, \ell\}, \tag{5}$$

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
Generalized Power Cone	GPC
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
Quantum Relative Entropy	QRE
Relative Entropy	RE
Hyperbolic Polynomials	HB
Equality Constraints	EQ

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

$$\texttt{A\{k,1\}} = \begin{bmatrix} A_L^1 \\ \vdots \\ A_L^\ell \end{bmatrix}, \quad \texttt{b\{k,1\}} = \begin{bmatrix} b_L^1 \\ \vdots \\ b_L^\ell \end{bmatrix}$$

$$\texttt{cons\{k,1\}='LP'}, \quad \texttt{cons\{k,2\}} = [m_L^1, \ldots, m_L^\ell]. \tag{6}$$

Similarly to add ℓ SOCP constraints of the form

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

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

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

$$\texttt{cons\{k,1\}='SOCP'}, \quad \texttt{cons\{k,2\}} = [m_S^1,\ldots,m_S^\ell]. \tag{8}$$

Let us see an example:

Example 3.1. *Suppose we are given the problem:*

$$\begin{array}{ll} \min & c^\top x \\ s.t. & [-2,1]x \le 1, \\ & \left\| \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix} x + \begin{bmatrix} 3 \\ 4 \end{bmatrix} \right\|_2 \le 2. \end{array} \tag{9}$$

Then we define

$$\texttt{cons\{1,1\}='LP'}, \quad \texttt{cons\{1,2\}=[1]}, \quad \texttt{A\{1,1\}} = \begin{bmatrix} 2 & -1 \end{bmatrix}, \quad \texttt{b\{1,1\}} = \begin{bmatrix} 1 \end{bmatrix},$$

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

The s.c. barriers and their LF conjugates being used in DDS for these constraints are

$$\begin{array}{l} \Phi(z) = -\ln(z), \quad z \in \mathbb{R}_+, \quad \Phi_*(\eta) = -1 - \ln(-\eta), \\ \Phi(t,z) = -\ln(t^2 - z^\top z), \quad \Phi_*(\eta, w) = -2 + \ln(4) - \ln(\eta^2 - w^\top w). \end{array} \tag{10}$$

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

$$\{(z,t,s) \in \mathbb{R}^n \oplus \mathbb{R} \oplus \mathbb{R} : \|z\|^2 \le ts,\ t \ge 0,\ s \ge 0\}, \tag{11}$$

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

$$\begin{array}{l} \|A_S^i x + b_S^i\|_2^2 \le ((g_S^i)^\top x + d_S^i)((\bar{g}_S^i)^\top x + \bar{d}_S^i), \quad i \in \{1,\ldots,\ell\}, \\ (g_S^i)^\top x + d_S^i \ge 0, \quad (\bar{g}_S^i)^\top x + \bar{d}_S^i \ge 0, \end{array} \tag{12}$$

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

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

$$\texttt{cons\{k,1\}='SOCPR'}, \quad \texttt{cons\{k,2\}} = [m_S^1, \ldots, m_S^\ell]. \tag{13}$$

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

$$F_0^i + x_1 F_1^i + \cdots + x_n F_n^i \succeq 0, \quad i \in \{1, \ldots, \ell\}. \tag{14}$$

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

$$\texttt{vec2sm(sm2vec(X))=X}. \tag{15}$$

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

$$\langle X, Y \rangle = \texttt{sm2vec(X)}^\top \texttt{sm2vec(Y)}. \tag{16}$$

To give (14) to DDS as the kth input block, we define:

$$\texttt{A\{k,1\}} := \begin{bmatrix} \texttt{sm2vec}(F_1^1), \cdots, \texttt{sm2vec}(F_n^1) \\ \vdots \\ \texttt{sm2vec}(F_1^\ell), \cdots, \texttt{sm2vec}(F_n^\ell) \end{bmatrix}, \quad b\{k,1\} := \begin{bmatrix} \texttt{sm2vec}(F_0^1) \\ \vdots \\ \texttt{sm2vec}(F_0^\ell) \end{bmatrix},$$

$$\texttt{cons\{k,1\}='SDP'} \quad \texttt{cons\{k,2\}} = [n^1, \ \ldots \ , n^\ell]. \tag{17}$$

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*

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

We can write this problem as

$$\begin{aligned} \min \quad & t \\ s.t. \quad & -1 + x_1 + x_2 + x_3 \geq 0, \\ & tI - (A_0 + x_1A_1 + x_2A_2 + x_3A_3) \succeq 0. \end{aligned} \tag{18}$$

To solve this problem, we define:

$$\texttt{cons}\{1,1\}=\texttt{'LP'}, \quad \texttt{cons}\{1,2\} = [1], \quad \texttt{cons}\{2,1\}=\texttt{'SDP'}, \quad \texttt{cons}\{2,2\} = [3],$$

$$\texttt{A}\{1,1\} = \begin{bmatrix} 1 & 1 & 1 & 0 \end{bmatrix}, \quad \texttt{b}\{1,1\} = \begin{bmatrix} -1 \end{bmatrix},$$

$$\texttt{A}\{2,1\} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \texttt{b}\{2,1\} = \begin{bmatrix} -2 \\ 0.5 \\ 0.6 \\ 0.5 \\ -2 \\ -0.4 \\ 0.6 \\ -0.4 \\ -3 \end{bmatrix},$$

$$\texttt{c} = (0,0,0,1)^\top.$$

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

4. Quadratic constraints

Suppose we want to add the following constraints to DDS:

$$x^\top A_i^\top Q_i A_i x + b_i^\top x + d_i \leq 0, \quad i \in \{1, \ldots, \ell\}, \tag{19}$$

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 (19) and a shifted *hyperbolicity cone* defined by the quadratic inequality $y^\top Q_i y \leq 0$.

To give constraints in (19) as input to DDS as the kth block, we define

$$\texttt{A\{k,1\}} = \begin{bmatrix} b_1^\top \\ A_1 \\ \vdots \\ b_l^\top \\ A_\ell \end{bmatrix}, \quad \texttt{b\{k,1\}} = \begin{bmatrix} d_1 \\ 0 \\ \vdots \\ d_\ell \\ 0 \end{bmatrix}$$

$$\texttt{cons\{k,1\}='QC'} \quad \texttt{cons\{k,2\}} = [m_1, \ldots, m_\ell],$$

$$\texttt{cons\{k,3,i\}} = Q_i, \quad i \in \{1, \ldots, \ell\}. \tag{20}$$

If `cons{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 (19) can be written as

$$\begin{aligned} & u^\top u + w + d \leq 0 \\ & u := R_i A_i x, \quad w := b_i^\top x, \quad d := d_i, \end{aligned} \tag{21}$$

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:

$$\begin{aligned} \Phi(u, w) &= -\ln(-(u^\top u + w + d)), \\ \Phi_*(y, \eta) &= \frac{y^\top y}{4\eta} - 1 - d\eta - \ln(\eta). \end{aligned} \tag{22}$$

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 (19), 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.

5. Generalized Power Cone

We define the (m, n)-generalized power cone with parameter α as

$$K_\alpha^{(m,n)} := \left\{ (s, u) \in \mathbb{R}_+^m \oplus \mathbb{R}^n : \prod_{i=1}^m s_i^{\alpha_i} \geq \|u\|_2 \right\}, \tag{23}$$

where α belongs to the simplex $\{\alpha \in \mathbb{R}^m_+ : \sum_{i=1}^m \alpha_i = 1\}$. Note that the rotated second order cone is a special case where $m = 2$ and $\alpha_1 = \alpha_2 = \frac{1}{2}$. Different s.c. barriers for this cone or special cases of it have been considered [8, 35]. Chares conjectured a s.c. barrier for the cone in (23), which is proved in [31]. This function that we use in DDS is:

$$\Phi(s,u) = -\ln\left(\prod_{i=1}^m s_i^{2\alpha_i} - u^\top u\right) - \sum_{i=1}^m (1-\alpha_i)\ln(s_i). \tag{24}$$

To add generalized power cone constraints to DDS, we use the abbreviation 'GPC'. Therefore, if the kth block of constraints is GNC, we define `cons{k,1}='GPC'`. Assume that we want to input the following ℓ constraints to DDS:

$$(A_s^i x + b_s^i, A_u^i x + b_u^i) \in K_{\alpha^i}^{(m_i,n_i)}, \quad i \in \{1,\dots,\ell\}, \tag{25}$$

where A_s^i, b_s^i, A_u^i, and b_u^i are matrices and vectors of proper size. Then, to input these constraints as the kth block, we define `cons{k,2}` as a MATLAB cell array of size ℓ-by-2, each row represents one constraint. We then define:

$$\begin{aligned} \texttt{cons\{k,2\}\{i,1\}} &= [m_i \;\; n_i], \\ \texttt{cons\{k,2\}\{i,2\}} &= \alpha^i, \quad i \in \{1,\dots,\ell\}. \end{aligned} \tag{26}$$

For matrices A and b, we define:

$$\texttt{A\{k,1\}} = \begin{bmatrix} A_s^1 \\ A_u^1 \\ \vdots \\ A_s^\ell \\ A_u^\ell \end{bmatrix}, \quad \texttt{b\{k,1\}} = \begin{bmatrix} b_s^1 \\ b_u^1 \\ \vdots \\ b_s^\ell \\ b_u^\ell \end{bmatrix}. \tag{27}$$

Example 5.1. *Consider the following optimization problem with 'LP' and 'GPC' constraints:*

$$\begin{aligned} \min \quad & -x_1 - x_2 - x_3 \\ s.t. \quad & \|x\| \le (x_1+3)^{0.3}(x_2+1)^{0.3}(x_3+2)^{0.4}, \\ & x_1, x_2, x_3 \ge 3. \end{aligned} \tag{28}$$

Then we define:

$$
\begin{aligned}
&\texttt{cons\{1,1\}='GPC'}, \quad \texttt{cons\{1,2\}} = \{[3, \quad 3], \quad [0.3; 0.3; 0.4]\} \\
&\texttt{A\{1,1\}} = \begin{bmatrix} \texttt{eye(3)} \\ \texttt{eye(3)} \end{bmatrix}, \quad \texttt{b\{1,1\}} = [3; 1; 2; 0; 0; 0] \\
&\texttt{cons\{2,1\}='LP'}, \quad \texttt{cons\{2,2\}} = [3] \\
&\texttt{A\{2,1\}} = [-\texttt{eye(3)}], \quad \texttt{b\{2,1\}} = [3; 3; 3] \\
&c = [-1, -1, -1].
\end{aligned}
$$

6. Epigraphs of matrix norms

Assume that we have constraints of the form

$$
\begin{aligned}
& X - UU^\top \succeq 0, \\
& X = A_0 + \sum_{i=1}^{\ell} x_i A_i, \\
& U = B_0 + \sum_{i=1}^{\ell} x_i B_i,
\end{aligned} \tag{29}
$$

where A_i, $i \in \{1, \ldots, \ell\}$, are m-by-m symmetric matrices, and B_i, $i \in \{1, \ldots, \ell\}$, 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:

$$
\Phi(Z, U) := -\ln(\det(Z - UU^\top)), \tag{30}
$$

with LF conjugate

$$
\Phi_*(Y, V) = -m - \frac{1}{4}\mathrm{Tr}(V^\top Y^{-1} V) - \ln(\det(-Y)), \tag{31}
$$

where $Y \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{m \times n}$ [25]. 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. A special but very important application is minimizing the *nuclear norm* of a matrix, which we describe in a separate subsection in the following.

DDS has two internal functions `m2vec` and `vec2m` for converting matrices (not necessarily symmetric) to vectors and vice versa. For an m-by-n matrix Z, `m2vec(Z,n)` change the matrix into a vector. `vec2m(v,m)` reshapes a vector v of proper size to a matrix with m rows. The abbreviation we use for epigraph of a matrix norm is MN. If the kth 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 (29), the corresponding parts in A and b are defined as

$$\texttt{A\{k,1\}} = \begin{bmatrix} \texttt{m2vec}(B_1, n) & \cdots & \texttt{m2vec}(B_\ell, n) \\ \texttt{sm2vec}(A_1) & \cdots & \texttt{sm2vec}(A_\ell) \end{bmatrix}, \quad \texttt{b\{k,1\}} = \begin{bmatrix} \texttt{m2vec}(B_0, n) \\ \texttt{sm2vec}(A_0) \end{bmatrix}. \tag{32}$$

For implementation details involving epigraph of matrix norms, see Appendix B.2. Here is an example:

Example 6.1. *Assume that we have matrices*

$$U_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}, \quad U_1 = \begin{bmatrix} -1 & -1 & 1 \\ 0 & 0 & 1 \end{bmatrix}, \quad U_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \tag{33}$$

and our goal is to solve

$$\begin{array}{ll} \min & t \\ s.t. & UU^\top \preceq tI, \\ & U = U_0 + x_1 U_1 + x_2 U_2. \end{array} \tag{34}$$

Then the input to DDS is defined as

$$\texttt{cons\{1,1\}='MN'}, \quad \texttt{cons\{2,1\}} = [2 \;\; 3],$$
$$\texttt{A\{1,1\}} = \begin{bmatrix} \texttt{m2vec}(U_1, 3) & \texttt{m2vec}(U_2, 3) & \texttt{zeros}(6, 1) \\ \texttt{zeros}(4, 1) & \texttt{zeros}(4, 1) & \texttt{sm2vec}(I_{2\times 2}) \end{bmatrix}, \quad \texttt{b\{1,1\}} = \begin{bmatrix} \texttt{m2vec}(U_0, 3) \\ \texttt{zeros}(4, 1) \end{bmatrix},$$
$$\texttt{c} = [0, 0, 1].$$

6.1. **Minimizing nuclear norm.** The nuclear norm of a matrix Z is $\|Z\|_* := \mathrm{Tr}\left((ZZ^\top)^{1/2}\right)$. The dual norm of $\|\cdot\|_*$ is the operator 2-norm $\|\cdot\|$ of a matrix. Minimization of nuclear norm has application in machine learning and matrix sparsification. The following optimization problems are a primal-dual pair [29].

$$\begin{array}{llll} (P_N) & \min_X & \|X\|_* & \\ & s.t. & A(X) = b. & \end{array} \qquad \begin{array}{llll} (D_N) & \max_z & \langle b, z\rangle & \\ & s.t. & \|A^*(z)\| \le 1, & \end{array} \tag{35}$$

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 $\mathrm{rank}(X)$ subject to $A(X) = b$, with applications in machine learning and compressed sensing. The dual problem (D_N) is a special case of (29) 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).

More specifically, consider the optimization problem

$$\begin{array}{ll} \min & \|X\|_* \\ s.t. & \mathrm{Tr}(U_i X) = c_i, \quad i \in \{1, \ldots, \ell\}, \end{array} \tag{36}$$

where X is n-by-m. DDS can solve the dual problem by defining

$$\begin{array}{l} \texttt{A\{1,1\}} = \begin{bmatrix} \texttt{m2vec}(U_1, n) & \cdots & \texttt{m2vec}(U_\ell, n) \\ \texttt{zeros}(m^2, 1) & \cdots & \texttt{zeros}(m^2, 1) \end{bmatrix}, \quad \texttt{b\{1,1\}} = \begin{bmatrix} \texttt{zeros}(mn, 1) \\ \texttt{sm2vec}(I_{m\times m}) \end{bmatrix}, \\ \texttt{cons\{1,1\}='MN'}, \quad \texttt{cons\{1,2\}} = [m \;\; n]. \end{array} \tag{37}$$

Then, if we run `[x,y]=DDS(c,A,b,cons)` and define `V:=(vec2m(y{1}(1:m*n),m))'`, then V is an optimal solution for (36). In subsection 13.3, we present numerical results for solving problem (36) and show that for the cases that $n \gg m$, DDS can be more efficient than SDP based solvers. Here is an example:

Example 6.2. *We consider minimizing the nuclear norm over a subspace. Consider the following optimization problem:*

$$\begin{array}{ll} \min & \|X\|_* \\ s.t. & \mathrm{Tr}(U_1 X) = 1 \\ & \mathrm{Tr}(U_2 X) = 2, \end{array} \tag{38}$$

where

$$U_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \quad U_2 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{39}$$

By using (35), the dual of this problem is

$$\begin{array}{ll} \min & -u_1 - 2u_2 \\ s.t. & \|u_1 U_1 + u_2 U_2\| \leq 1. \end{array} \tag{40}$$

To solve this problem with our code, we define

$$\begin{array}{l} \texttt{cons\{1,1\}='MN'}, \quad \texttt{cons\{1,2\}} = [2 \;\; 4], \\ \texttt{A\{1,1\}} = \begin{bmatrix} \texttt{m2vec}(U_1, 4) & \texttt{m2vec}(U_2, 4) \\ \texttt{zeros}(4, 1) & \texttt{zeros}(4, 1) \end{bmatrix}, \quad \texttt{b\{1,1\}} = \begin{bmatrix} \texttt{zeros}(8, 1) \\ \texttt{sm2vec}(I_{2\times 2}) \end{bmatrix}, \\ \texttt{c} = [-1, -2]. \end{array}$$

If we solve the problem using `[x,y]=DDS(c,A,b,cons)`*, the optimal value is* -2.2360*. Now* `V:=(vec2m(y{1}(1:8),2))`$^\top$ *is the solution of* (38) *with objective value* 2.2360*. We have*

$$X^* := V = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}. \tag{41}$$

7. Epigraphs of convex univariate functions (geometric, entropy, and p-norm programming)

DDS accepts constraints of the form

$$\sum_{i=1}^{\ell} \alpha_i f_i(a_i^\top x + \beta_i) + g^\top x + \gamma \le 0, \quad a_i, g \in \mathbb{R}^n, \quad \beta_i, \gamma \in \mathbb{R}, \quad i \in \{1, \ldots, \ell\}, \tag{42}$$

where $\alpha_i \ge 0$ and $f_i(x)$, $i \in \{1, \ldots, \ell\}$, 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) [2] and entropy programming (EP) [12] with many applications in engineering are constructed with constraints of the form (42) when $f_i(z) = e^z$ for $i \in \{1, \ldots, \ell\}$ and $f_i(z) = z\ln(z)$ for $i \in \{1, \ldots, \ell\}$, respectively. The other functions with p powers let us solve optimization problems related to p-norm minimization. The corresponding s.c. barriers used in DDS are shown in Table

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

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

2. There is a closed form expression for the LF conjugate of the first two functions. For the last four, the LF conjugate can be calculated to high accuracy efficiently. In Appendix D, 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 from (42), for given $\gamma \in \mathbb{R}$ and $\beta_i \in \mathbb{R}$, $i \in \{1, \ldots, \ell\}$, we can define the corresponding convex set D as

$$D := \{(w, s_i, u_i) : w + \gamma \leq 0, \quad f_i(s_i + \beta_i) \leq u_i, \ \forall i\}, \tag{43}$$

and our matrix A represents $w = \sum_{i=1}^{\ell} \alpha_i u_i + g^\top x$ and $s_i = a_i^T x$, $i \in \{1, \ldots, \ell\}$. As can be seen, to represent our set as above, we introduce some auxiliary variables u_i's to our formulations. DDS code does this internally. Let us assume that we want to add the following s constraints to our model

$$\sum_{type} \sum_{i=1}^{\ell^j_{type}} \alpha_i^{j,type} f_{type}((a_i^{j,type})^\top x + \beta_i^{j,type}) + g_j^\top x + \gamma_j \leq 0, \qquad j \in \{1, \ldots, s\}. \tag{44}$$

From now on, *type* indexes the rows of Table 2. The abbreviation we use for these constraints is TD. Hence, if the kth input block are the constraints in (44), then we have `cons{k,1}='TD'`. `cons{k,2}` is a **cell array** of MATLAB with s rows, each row represents one constraint. For the jth constraint we have:

- `cons{k,2}{j,1}` is a matrix with two columns: the first column shows the type of a function from Table 2 and the second column shows the number of that function in the constraint. Let us say that in the jth constraint, we have l_2^j functions of type 2 and l_3^j functions of type 3, then we have

$$\texttt{cons\{k,2\}\{j,1\}} = \begin{bmatrix} 2 & l_2^j \\ 3 & l_3^j \end{bmatrix}.$$

  The types can be in any order, but the functions of the same type are consecutive and the order must match with the rows of A and b.
- `cons{k,2}{j,2}` is a vector with the coefficients of the functions in a constraint, i.e., $\alpha_i^{j,type}$ in (44). Note that the coefficients must be in the same order as their corresponding rows in A and b. If in the jth constraint we have 2 functions of type 2 and 1 function of type 3, it starts as

$$\texttt{cons\{k,2\}\{j,2\}} = [\alpha_1^{j,2}, \alpha_2^{j,2}, \alpha_1^{j,3}, \cdots].$$

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 `cons{k,2}`. 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 jth constraint is as follows if we have for example five types

$$\texttt{A} = \begin{bmatrix} g_j^\top \\ a_1^{j,1} \\ \vdots \\ a_{l_1^j}^{j,1} \\ \vdots \\ a_1^{j,5} \\ \vdots \\ a_{l_5^j}^{j,5} \end{bmatrix}, \quad \texttt{b} = \begin{bmatrix} \gamma_j \\ \beta_1^{j,1} \\ \vdots \\ \beta_{l_1^j}^{j,1} \\ \vdots \\ \beta_1^{j,5} \\ \vdots \\ \beta_{l_5^j}^{j5} \end{bmatrix}. \tag{45}$$

Let us see an example:

Example 7.1. *Assume that we want to solve*

$$\begin{aligned} \min \quad & c^\top x \\ s.t. \quad & -1.2\ln(x_2 + 2x_3 + 55) + 1.8e^{x_1+x_2+1} + x_1 - 2.1 \le 0, \\ & -3.5\ln(x_1 + 2x_2 + 3x_3 - 30) + 0.9e^{-x_3-3} - x_3 + 1.2 \le 0, \\ & x \ge 0. \end{aligned} \tag{46}$$

For this problem, we define:

$$\texttt{cons}\{1,1\}=\texttt{'LP'}, \quad \texttt{cons}\{1,2\} = [3],$$

$$\texttt{cons}\{2,1\}=\texttt{'TD'}, \quad \texttt{cons}\{2,2\} = \left\{ \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix}, [1.2 \quad 1.8] \; ; \quad \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix}, [3.5 \quad 0.9] \right\},$$

$$\texttt{A}\{1,1\} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \quad \texttt{b}\{1,1\} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix},$$

$$\texttt{A}\{2,1\} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 1 & 1 & 0 \\ 0 & 0 & -1 \\ 1 & 2 & 3 \\ 0 & 0 & -1 \end{bmatrix}, \quad \texttt{b}\{2,1\} = \begin{bmatrix} -2.1 \\ 55 \\ 1 \\ 1.2 \\ -30 \\ -3 \end{bmatrix}.$$

Note: *As we mentioned, modeling systems for convex optimization that are based on SDP solvers, such as CVX, have to use approximation for functions involving exp and* ln*. Approximation makes it hard to return dual certificates, specifically when the problem is infeasible or unbounded.*

7.1. **Constraints involving power functions.** The difference between these two types (4 and 5) and the others is that we also need to give the value of p for each function. To do that, we add another column to `cons{k,2}`.

Note: For TD constraints, `cons{k,2}` can have two or three columns. If we do not use types 4 and 5, it has two, otherwise three columns. `cons{k,2}{j,3}` is a vector which contains the powers p for functions of types 4 and 5. The powers are given in the same order as the coefficients in `cons{k,2}{j,2}`. If the constraint also has functions of other types, we must put 0 in place of the power.

Let us see an example:

Example 7.2.

$$\begin{array}{ll} \min & c^\top x \\ s.t. & 2.2\exp(2x_1+3)+|x_1+x_2+x_3|^2+4.5|x_1+x_2|^{2.5}+|x_2+2x_3|^3+1.3x_1-1.9\leq 0. \end{array}$$

For this problem, we define:

$$A\{1,1\}=\begin{bmatrix} 1.3 & 0 & 0\\ 2 & 0 & 0\\ 1 & 1 & 1\\ 1 & 1 & 0\\ 0 & 1 & 2 \end{bmatrix},\quad b\{1,1\}=\begin{bmatrix} -1.9\\ 3\\ 0\\ 0\\ 0 \end{bmatrix},$$

$$cons\{1,1\}=\text{'TD'},\quad cons\{1,2\}=\left\{\begin{bmatrix} 2 & 1\\ 4 & 3 \end{bmatrix},\ [2.2\ \ 1\ \ 4.5\ \ 1],\ [0\ \ 2\ \ 2.5\ \ 3]\right\}.$$

8. Vector Relative Entropy

Consider the relative entropy function $f:\mathbb{R}^\ell_{++}\oplus\mathbb{R}^\ell_{++}\to\mathbb{R}$ defined as

$$f(u,z):=\sum_{i=1}^{\ell}u_i\ln(u_i)-u_i\ln(z_i).$$

The epigraph of this function accepts the following $(2\ell+1)$-s.c. barrier (see Appendix E):

$$\Phi(t,u,z) := -\ln\left(t - \sum_{i=1}^{\ell} u_i \ln(u_i/z_i)\right) - \sum_{i=1}^{\ell}\ln(u_i) - \sum_{i=1}^{\ell}\ln(z_i). \tag{47}$$

This function covers many applications as discussed in [10]. The special case of $\ell = 1$ is used for handling EXP cone in the commercial solver MOSEK. We claim that the LF of Φ can be efficiently calculated using ℓ one-dimensional minimizations, i.e., having the function $\theta(r)$ as the solution of $\frac{1}{\theta} - \ln(\theta) = r$. We consider the case of $\ell = 1$ and the generalization is straightforward. We have

$$\Phi_*(\alpha, y_u, y_z) := \min_{t,z,u} \alpha t + y_z z + y_u u + \ln(t - u\ln(u) + u\ln(z)) + \ln(z) + \ln(u). \tag{48}$$

Writing the optimality conditions, we get (defining $T := t - u\ln(u) + u\ln(z)$)

$$\begin{aligned} \alpha + \tfrac{1}{T} &= 0 \\ y_z + \tfrac{u}{zT} + \tfrac{1}{z} &= 0 \\ y_u - \tfrac{1+\ln(u)-\ln(z)}{T} + \tfrac{1}{u} &= 0 \end{aligned} \tag{49}$$

We can see that at the optimal solution, $T = -1/\alpha$, and if we calculate u, we can easily get z. u is calculated by the following equation:

$$\frac{1}{T} + \frac{1}{u} = \frac{1}{\theta\left(-Ty_u + 2 + \ln(-y_z) + \ln(T)\right)} \tag{50}$$

Therefore, to calculate Φ_* at every point, we need to evaluate θ once. For the general ℓ, we evaluate θ by ℓ times.

The abbreviation we use for relative entropy is RE. So, for the kth block of s constraints of the form

$$f(A_u^i x + b_u^i, A_z^i x + b_z^i) + g_i^\top x + \gamma_i \le 0, \quad i \in \{1,\ldots,s\}, \tag{51}$$

we define `cons{k,1} = 'RE'` and `cons{k,2}` is a vector of length s with the ith element equal to $2\ell+1$. We also define:

$$\mathtt{A} = \begin{bmatrix} g_i^\top \\ A_u^1 \\ A_z^1 \\ \vdots \\ g_s^\top \\ A_u^s \\ A_z^s \end{bmatrix}, \quad \mathtt{b} = \begin{bmatrix} \gamma_1 \\ b_u^1 \\ b_z^1 \\ \vdots \\ \gamma_s \\ b_u^s \\ b_z^s \end{bmatrix}. \tag{52}$$

Example 8.1. *Assume that we want to minimize a relative entropy function under a linear constraint:*

$$\begin{array}{ll} \min & (0.8x_1+1.3)\ln\left(\frac{0.8x_1+1.3}{2.1x_1+1.3x_2+1.9}\right)+(1.1x_1-1.5x_2-3.8)\ln\left(\frac{1.1x_1-1.5x_2-3.8}{3.9x_2}\right) \\ s.t. & x_1+x_2\leq\beta. \end{array} \tag{53}$$

We add an auxiliary variable x_3 to model the objective function as a constraint. For this problem we define:

$$\begin{aligned} &\texttt{cons}\{1,1\}=\texttt{'RE'},\quad \texttt{cons}\{1,2\}=[5] \\ &\texttt{A}\{1,1\}=\begin{bmatrix} 0 & 0 & -1 \\ 0.8 & 0 & 0 \\ 1.1 & -1.5 & 0 \\ 2.1 & 1.3 & 0 \\ 0 & 3.9 & 0 \end{bmatrix},\quad \texttt{b}\{1,1\}=\begin{bmatrix} 0 \\ 1.3 \\ -3.8 \\ 1.9 \\ 0 \end{bmatrix}, \\ &\texttt{cons}\{2,1\}=\texttt{'LP'},\quad \texttt{cons}\{2,2\}=[1] \\ &\texttt{A}\{2,1\}=[-1\quad -1\quad 0],\quad \texttt{b}\{2,1\}=[\beta]. \end{aligned}$$

If we solve this problem by DDS, for $\beta=2$ the problem is infeasible, and for $\beta=7$ it returns an optimal solution $x^*:=(5.93,1.06)^\top$ with the minimum of function equal to -7.259.

9. Quantum entropy and Quantum relative entropy

Quantum entropy and quantum relative entropy functions are important in quantum information processing. DDS 2.0 accepts constraints involving these two functions. Let us start with the main definitions. Consider a function $f:\mathbb{R}\to\mathbb{R}\cup\{+\infty\}$ and let $X\in\mathbb{H}^n$ be a Hermitian matrix (with entries from $\mathbb{C}$) with a spectral decomposition $X=U\mathrm{Diag}(\lambda_1,\ldots,\lambda_n)U^*$, where Diag returns a diagonal matrix with the given entries on its diagonal and U^* is the conjugate transpose of a unitary matrix U. We define the *matrix extension* F of f as $F(X):=U\mathrm{Diag}(f(\lambda_1),\ldots,f(\lambda_n))U^*$. Whenever we write $\phi(X):=\mathrm{Tr}(F(X))$, we mean a function $\phi:\mathbb{H}^n\to\mathbb{R}\cup\{+\infty\}$ defined as

$$\phi(X):=\begin{cases} \mathrm{Tr}(U\mathrm{Diag}(f(\lambda_1),\ldots,f(\lambda_n))U^*) & \text{if } f(\lambda_i)\in\mathbb{R},\ \forall i, \\ +\infty & \text{o.w.} \end{cases} \tag{54}$$

Study of such matrix functions go back to the work of Löwner as well as Von-Neumann (see [11], [21], 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

$$F(tX + (1-t)Y) \preceq tF(X) + (1-t)F(Y), \quad \forall t \in (0, 1). \tag{55}$$

Faybusovich and Tsuchiya [14] 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 ϕ as (54). Then, the function

$$\Phi(t, X) := -\ln(t - \phi(X)) - \ln\det(X) \tag{56}$$

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* $qe : \mathbb{H}^n \to \mathbb{R} \cup \{+\infty\}$ which is defined as $qe(X) := \mathrm{Tr}(X\ln(X))$. If we consider $f(x) := x\ln(x)$, then $f'(x) = 1 + \ln(x)$ is matrix monotone on $(0, \infty)$ (see, for instance [18]-Example 4.2) and so we have a s.c. barrier for the set

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

We have to solve the optimization problem

$$\Phi_*(\eta, Y) = \sup_{t,X} t\eta + \langle X, Y\rangle + \ln(t - qe(X)) + \ln\det(X), \tag{57}$$

to calculate the LF conjugate of (56), which is done in Appendix C.

Another interesting function with applications in quantum information theory is *quantum relative entropy* $qre : \mathbb{H}^n \oplus \mathbb{H}^n \to \mathbb{R} \cup \{+\infty\}$ defined as

$$qre(X, Y) := X\ln(X) - X\ln(Y).$$

This function is convex as proved in [34]. The epigraph of qre is

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

DDS 2.0 uses the following barrier (not yet known to be s.c.) for solving problems involving quantum relative entropy constraints:

$$\Phi(t, X, Y) := \ln(t - qre(X, Y)) - \ln\det(X) - \ln\det(Y).$$

We can input constraints involving quantum entropy and quantum relative entropy into DDS as we explain in the following subsections. Appendix C contains some interesting theoretical results about quantum entropy functions and how to calculate the derivative and Hessian for the above s.c. barriers and also the LF conjugate given in (57).

9.1. **Adding quantum entropy based constraints.** Let $qe_i : \mathbb{S}^{n_i} \to \mathbb{R} \cup \{+\infty\}$ be quantum entropy functions and consider ℓ quantum entropy constraints of the form

$$qe_i(A_0^i + x_1 A_1^i + \cdots + x_n A_n^i) \leq g_i^\top x + d_i, \quad i \in \{1, \ldots, \ell\}. \tag{58}$$

A_j^i's are n_i-by-n_i symmetric matrices. To input (58) to DDS as the kth block, we define:

$$\begin{aligned} &\texttt{cons\{k,1\}='QE'}, \quad \texttt{cons\{k,2\}} = [n_1, \ldots, n_\ell], \\ &\texttt{A\{k,1\}} := \begin{bmatrix} g_1^\top \\ \texttt{sm2vec}(A_1^1), \cdots, \texttt{sm2vec}(A_n^1) \\ \vdots \\ g_\ell^\top \\ \texttt{sm2vec}(A_1^\ell), \cdots, \texttt{sm2vec}(A_n^\ell) \end{bmatrix}, \quad \texttt{b\{k,1\}} := \begin{bmatrix} d_1 \\ \texttt{sm2vec}(A_0^1) \\ \vdots \\ d_\ell \\ \texttt{sm2vec}(A_0^\ell) \end{bmatrix}. \end{aligned} \tag{59}$$

Example 9.1. *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*

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

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

$$\begin{aligned} \min \quad & t \\ s.t. \quad & qe(x_1A_1 + x_2A_2 + x_3A_3) \leq t, \\ & 2x_1 + 3x_2 - x_3 \leq 5, \\ & x_1A_1 + x_2A_2 + x_3A_3 \succeq 3I. \end{aligned} \tag{60}$$

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

$$\texttt{cons\{3,1\}='QE'}, \quad \texttt{cons\{3,2\}} = [3], \quad \texttt{b\{3,1\}} := \begin{bmatrix} 0 \\ \texttt{zeros}(9,1) \end{bmatrix},$$

$$\texttt{A\{3,1\}} := \begin{bmatrix} 0 & 0 & 0 & 1 \\ \texttt{sm2vec}(A1) & \texttt{sm2vec}(A2) & \texttt{sm2vec}(A3) & \texttt{sm2vec}(\texttt{zeros}(3)) \end{bmatrix}.$$

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

9.2. **Adding quantum relative entropy based constraints.** The abbreviation we use for quantum relative entropy is QRE. Let $qre_i : \mathbb{S}^{n_i} \oplus \mathbb{S}^{n_i} \to \mathbb{R} \cup \{+\infty\}$ be quantum relative entropy functions and consider ℓ quantum entropy constraints of the form

$$qre_i(A_0^i + x_1 A_1^i + \cdots + x_n A_n^i, B_0^i + x_1 B_1^i + \cdots + x_n B_n^i) \leq g_i^\top x + d_i, \quad i \in \{1, \ldots, \ell\}.$$

A_j^i's and B_j^i's are n_i-by-n_i symmetric matrices. To input (58) to DDS as the kth block, we define:

$$\texttt{cons\{k,1\}='QRE'}, \quad \texttt{cons\{k,2\}} = [n_1, \ldots, n_\ell],$$

$$\texttt{A\{k,1\}} := \begin{bmatrix} g_1^\top \\ \texttt{sm2vec}(A_1^1), \cdots, \texttt{sm2vec}(A_n^1) \\ \texttt{sm2vec}(B_1^1), \cdots, \texttt{sm2vec}(B_n^1) \\ \vdots \\ g_\ell^\top \\ \texttt{sm2vec}(A_1^\ell), \cdots, \texttt{sm2vec}(A_n^\ell) \\ \texttt{sm2vec}(B_1^\ell), \cdots, \texttt{sm2vec}(B_n^\ell) \end{bmatrix}, \quad \texttt{b\{k,1\}} := \begin{bmatrix} d_1 \\ \texttt{sm2vec}(A_0^1) \\ \texttt{sm2vec}(B_0^1) \\ \vdots \\ d_\ell \\ \texttt{sm2vec}(A_0^\ell) \\ \texttt{sm2vec}(B_0^\ell) \end{bmatrix}. \tag{61}$$

10. Hyperbolic polynomials

Hyperbolic programming problems form one of the largest classes of convex optimization problems which have the kind of special mathematical structure amenable to more robust and efficient computational approaches. DDS 2.0 accepts hyperbolic constraints. Let us first define a hyperbolic polynomial. A polynomial $p(x) \in \mathbb{R}[x_1, \ldots, x_m]$ is said to be *homogeneous* if every term has the same degree d. A homogeneous polynomial $p : \mathbb{R}^m \to \mathbb{R}$ 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.

Let $p(x) \in \mathbb{R}[x_1, \ldots, x_m]$ be hyperbolic in direction e. We define the *eigenvalue function* $\lambda : \mathbb{R}^m \to \mathbb{R}^d$ with respect to p and e such that for every $x \in \mathbb{R}^m$, the elements of $\lambda(x)$ are the roots of the univariate polynomial $p(x - te)$. The underlying *hyperbolicity cone* is defined as

$$\Lambda_+(p, e) := \{x \in \mathbb{R}^m : \lambda(x) \geq 0\}. \tag{62}$$

Example 10.1. *The polynomial $p(x) = x_1^2 - x_2^2 - \cdots - x_m^2$ is hyperbolic in the direction $e := (1, 0, \ldots, 0)^\top$ 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 10.1 (Güler [17]). *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, as we explain in Section 12.

10.1. **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 jth row, the first m entries are the powers 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

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

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 acyclic directed graph. 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 represents a simple operation. Assume that $p(x)$ has m variables, then we define

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

The ℓth 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

For example, if $p(x) = x_1^2 - x_2^2 - x_3^2$, we have the following representation:

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

Note that straight-line program is not unique for a polynomial.

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

$$p(x) = \det(H_0 + x_1 H_1 + x_2 H_2 + \cdots + x_m H_m), \tag{63}$$

where H_i, $i \in \{0, 1, \ldots, m\}$ are in $\mathbb{S}^m$. We define

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

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

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

Note: $H_1, \ldots, H_m$ must be nonzero symmetric matrices.

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

$$p(Ax + b) \geq 0. \tag{64}$$

To input this constraint to DDS as the kth 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 $p(z)$.
`cons{k,3}` is the `poly` that can be given in one of the three formats of Subsection 10.1.

`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 10.2. *Assume that we want to give constraint (64) to DDS for $p(x) = x_1^2 - x_2^2 - x_3^2$, using the monomial format. Then, `cons` part is defined as*

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

11. Equality constraints

In the main formulation of Domain-Driven form (1), equality constraints are not included. However, the current version of DDS accepts equality constraints. In other form, users can input a problem of the form

$$\inf_x \{\langle c, x\rangle : Bx = d,\ Ax \in D\}. \tag{65}$$

To input a block of equality constraints $Bx = d$, where B is a m_{EQ}-by-n matrix, as the kth block, we define

$$\texttt{cons\{k,1\}='EQ'}, \quad \texttt{cons\{k,2\}} = m_{EQ}, \quad \texttt{A\{k,1\}} = B, \quad \texttt{b\{k,1\}} = d.$$

Example 11.1. *If for a given problem with $x \in \mathbb{R}^3$, we have a constraint $x_2 - x_3 = 2$, then we can add it as the kth block by the following definitions:*

$$\texttt{cons\{k,1\}='EQ'}, \quad \texttt{cons\{k,2\}} = 1, \quad \texttt{A\{k,1\}} = [0 \ \ 1 \ \ -1], \quad \texttt{b\{k,1\}} = [2].$$

In the main Domain-Driven formulation (1), for some theoretical and practical considerations, we did not consider the equality constraints. From a mathematical point of view, this is not a problem. For any matrix Z whose columns form a basis for the null space of B, we may represent the solution set of $Bx = d$ as $x^0 + Zw$, where x^0 is any solution of $Bx = d$. Then the feasible region in (65) is equivalent to:

$$D_w := \{w : AZw \in (D - Ax^0)\}. \tag{66}$$

$D - Ax^0$ is a translation of D with the s.c. barrier $\Phi(z - Ax^0)$, where Φ is a s.c. barrier for D. Therefore, theoretically, we can reformulate (65) as (1), where AZ replaces A. Even though this procedure is straightforward in theory, there might be numerical challenges in application. For example, if we have a nice structure for A, such as sparsity, multiplying A with Z may ruin the structure. In DDS, we use a combination of heuristics, LU and QR factorizations to implement this transition efficiently.

12. 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 based on the relations between the derivatives of Φ and Φ_*: for every point $z \in \text{int}D$ we have

$$z = \Phi'_*(\Phi'(z)), \quad \Phi''_*(\Phi'(z)) = [\Phi''(z)]^{-1}. \tag{67}$$

Instead of the primal-dual proximity measure defined in [19], we use the primal-heavy version:

$$\left\| \frac{\tau y}{\mu} - \Phi'(u) \right\|_{[\Phi''(u)]^{-1}}, \tag{68}$$

where $u := Ax + \frac{1}{\tau} z^0$, τ is an artificial variable we use in the formulation of the central path (see [19] for details), and μ is the parameter of the central path. By [19]-Corollary 4.1, this proximity measures is "equivalent" to the primal-dual one, but (68) is less efficient computationally.

By using a primal-heavy version, we lose some of the desired 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 that 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 [19]. Note that when D is a shifted cone $D = K - b$, then we have

$$\delta_*(y|D) = -\langle b, y \rangle. \tag{69}$$

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 (69). This is the case for the current version of DDS as all the set constraints using primal-heavy version are based on a shifted convex cone. To make sure that the dual iterates are feasible, we choose our neighborhoods

to satisfy

$$\left\| \frac{\tau y}{\mu} - \Phi'(u) \right\|_{\Phi''(u)} < 1, \tag{70}$$

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;
```

13. Numerical Results

In this section, we present some numerical examples of running DDS 2.0. We performed computational experiments using the software MATLAB R2018b, on a 4-core 3.2 GHz Intel Xeon X5672 machine with 96GB of memory. Many of the examples in this section are given as mat-files in the DDS package; in the folder `Test_Examples`.

DDS pre-processes the input data and returns the statistics before and after the pre-processing. This processing includes checking the structure of the input data removing obviously redundant constraints. For example, if the size of A given for a block of constraints does not match the information in `cons`, DDS returns an error. The pre-processing also includes scaling the input data properly for each block of constraints.

As mentioned in Section 2, users have the option to give the starting point of the infeasible-start algorithm as input. Otherwise, DDS chooses an easy-to-find starting point for every input set constraint. The stopping criteria in DDS are based on the status determination result in [20], which studied how to efficiently certify different status of a given problem in Domain-Driven form, using duality theory. [20]-Section 9 discusses stopping criteria in a practical setting. We define the following parameters for the current point (x, τ, y):

$$gap := \frac{|\langle c, x\rangle + \frac{1}{\tau}\delta_*(y|D)|}{1 + |\langle c, x\rangle| + |\frac{1}{\tau}\delta_*(y|D)|}, \quad P_{feas} := \frac{1}{\tau}\|z^0\|, \quad D_{feas} := \frac{\|\frac{1}{\tau}A^\top y + c\|}{1 + \|c\|}, \tag{71}$$

where x and y are the primal and dual points, and τ is the artificial variable we use (see [19] for details). DDS stops the algorithm and returns the status as "solved" when we have

$$\max\{gap, P_{feas}, D_{feas}\} \leq tol. \tag{72}$$

DDS returns that status as "infeasible" if the returned certificate $\bar{y} := \frac{\tau}{\mu}y$ satisfies

$$\|A^\top \bar{y}\| \leq tol, \quad \delta_*(\bar{y}|D) < 0, \tag{73}$$

and returns the status as "unbounded" if

$$\langle c, x\rangle \leq -\frac{1}{tol}. \tag{74}$$

DDS has different criteria for returning "ill-conditioned" status, which can be because of numerical inaccuracy issues.

In the following, we see the numerical performance of DDS for different combinations of set constraints.

13.1. **LP-SOCP-SDP.** In this subsection, we consider LP-SOCP-SDP instances mostly from the Dimacs library [28]. 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. DDS has a built-in function `read_sedumi_DDS` to read problems in SeDuMi input format. You can use either of the following commands based on the SeDuMi format file

```
[c,A,b,cons]=read_sedumi_DDS(At,b,c,K);
[c,A,b,cons]=read_sedumi_DDS(A,b,c,K);
```

13.2. **EXP cone optimization problems from CBLIB.** Conic Benchmark Library (CBLIB) [15] is a collection of benchmark problems for conic mixed-integer and continuous optimization. It contains a collection of optimization problems involving exponential cone, they show as EXP. Exponential cone is a special case of vector relative entropy we discussed in Section 8 when $\ell = 1$. Table 4 show the results of running DDS to solve problems with EXP cone from CBLIB. The files converted into DDS format can be found in the `Test_Examples` folder.

13.3. **Minimizing Nuclear Norm.** In this subsection, we are going to use DDS to solve problem (36); minimizing nuclear norm of a matrix subject to linear constraints. We are interested in the case that X is an n-by-m matrix with $n \gg m$, which makes the s.c. barrier for the epigraph of a matrix norm more efficient than the SDP representation. We compare DDS with convex modeling system CVX, which has a function `norm_nuc` for nuclear norm. For numerical results, we assume (36) has two linear constraints and U_1 and U_2 are 0-1 matrices generated randomly. Let us fix the number of columns m and calculate the running time by changing the number of rows n. Figure 1 shows the plots of running time for both DDS and CVX. For epigraph of a matrix norm constraints, DDS does not form matrices of size $m + n$, and as can be seen in the figure, the running time is almost linear as a function of n. On the other hand, the CVX running time is clearly super-linear.

13.4. **Quantum Entropy and Quantum Relative Entropy.** In this subsection, we see optimization problems involving quantum entropy. As far as we know, there is no library for quantum

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	42
nb_L1	$3176 * 915$	SOCP-LP	37
nb_L2	$4195 * 123$	SOCP-LP	23
nb_L2_bessel	$2641 * 123$	SOCP-LP	27
filter48_socp	$3284 * 969$	SDP-SOCP-LP	80
filtinf1	$3395 * 983$	SDP-SOCP-LP	26
truss5	$3301 * 208$	SDP	66
truss8	$11914 * 496$	SDP	76
copo14	$3108 * 1275$	SDP-LP	24
copo23	$3108 * 1275$	SDP-LP	32
toruspm3-8-50	$262144 * 512$	SDP	20
torusg3-8	$262144 * 512$	SDP	24
sched_50_50_scaled	$4977 * 2526$	SOCP-LP	81
mater-3	$39448 * 1439$	SDP-LP	124
cnhil8	$14400 * 1716$	SDP	31
cnhil10	$48400 * 5005$	SDP	35
cphil10	$48400 * 5005$	SDP	12
ros_500	$17944 * 4988$	SDP	41
sensor_500	$245601 * 3540$	SDP	65
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	17

entropy and quantum relative entropy functions. We have created examples of combinations of quantum entropy constraints and other types such as LP, SOCP, and p-norm. Problems for

TABLE 4. Numerical results for some EXP cone problems from CBLIB.

Problem	size of A	size of A_{EQ}	Type of Constraints	Iterations
syn30m	$324 * 121$	$19 * 121$	EXP-LP	36
syn30h	$528 * 249$	$161 * 249$	EXP-LP	51
syn30m02h	$1326 * 617$	$381 * 617$	EXP-LP	46
syn05m02m	$177 * 67$	$11 * 67$	EXP-LP	36
syn10h	$171 * 84$	$55 * 84$	EXP-LP	37
syn15m	$177 * 67$	$12 * 67$	EXP-LP	31
syn40h	$715 * 331$	$213 * 331$	EXP-LP	48
synthes1	$30 * 13$	$1 * 13$	EXP-LP	13
synthes2	$41 * 16$	$1 * 12$	EXP-LP	26
synthes3	$71 * 26$	$3 * 26$	EXP-LP	20
ex1223a	$59 * 17$	$1 * 17$	EXP-LP	11
gp_dave_1	$985 * 705$	$453 * 705$	EXP-LP	37
fiac81a	$163 * 191$	$126 * 191$	EXP-LP	15
fiac81b	$65 * 87$	$57 * 87$	EXP-LP	15
batch	$175 * 58$	$13 * 58$	EXP-LP	160
demb761	$93 * 131$	$90 * 131$	EXP-LP	24
demb762	$93 * 131$	$90 * 131$	EXP-LP	27
demb781	$14 * 19$	$12 * 19$	EXP-LP	8
enpro48	$506 * 167$	$33 * 167$	EXP-LP	216
isi101	$467 * 393$	$261 * 33$	EXP-LP	52 (infeas)
jha88	$866 * 1131$	$825 * 1131$	EXP-LP	33
LogExpCR-n20-m400	$2023 * 2022$	$1200 * 2022$	EXP-LP	40
LogExpCR-n100-m400	$2103 * 2102$	$1200 * 2102$	EXP-LP	46
LogExpCR-n20-m1200	$6023 * 6022$	$3600 * 6022$	EXP-LP	44
LogExpCR-n500-m400	$2503 * 2502$	$1200 * 2502$	EXP-LP	68
LogExpCR-n500-m1600	$8503 * 8502$	$4800 * 8502$	EXP-LP	63

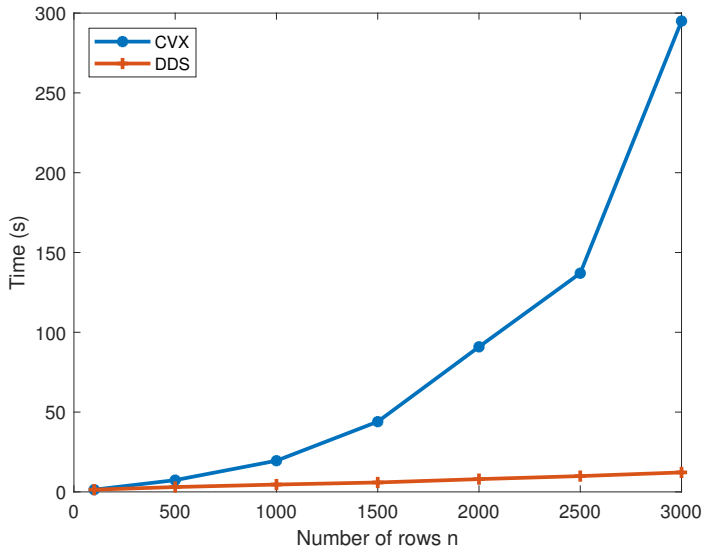

FIGURE 1. Average running time for minimizing nuclear norm versus the number of rows in the matrix, where the number of columns is fixed at $m = 20$.

quantum entropy are of the form

$$\begin{aligned} \min \quad & t \\ \text{s.t.} \quad & qe(A_0 + x_1A_1 + \cdots + x_nA_n) \leq t, \\ & \text{other constraints,} \end{aligned} \tag{75}$$

where $qe = \mathrm{Tr}X\ln(X)$ and A_i's are sparse symmetric matrices. The problems for qre are also in the same format. Our collection contains both feasible and infeasible cases. The results are shown in Tables 5 and 6 respectively for quantum entropy and quantum relative entropy. The MATLAB files in the DDS format for some of these problems are given in the `Test_Examples` folder. We compare our results to CVXQUAD, which is a collection of matrix functions to be used on top of CVX. The package is based on the paper [13], which approximates matrix logarithm with functions that an be described as feasible region of SDPs. Note that these approximations increase the dimension of the problem, which is why CVX fails for the larger problems in Table 5.

13.5. **Hyperbolic Polynomials.** We have created a library of hyperbolic cone programming problems for our experiments. Hyperbolic polynomials are defined in Section 10. We discuss three methods in this section for the creation of our library.

13.5.1. *Hyperbolic polynomials from matriods.* Let us first define a matroid.

TABLE 5. Results for problems involving Quantum Entropy optimization problems.

Problem	size of A	Type of Constraints	Itr/ time(s)	CVXQUAD
QuanEntr-10	$101 * 21$	QE	11/ 0.6	12/ 1.3
QuanEntr-20	$401 * 41$	QE	13/ 0.8	13/ 2.4
QuanEntr-50	$2501 * 101$	QE	18/ 1.8	12/ 23.2
QuanEntr-100	$10001 * 201$	QE	24/ 6.7	array size error
QuanEntr-200	$40001 * 401$	QE	32/ 53.9	array size error
QuanEntr-LP-10	$111 * 21$	QE-LP	19/ 1.1	16/ 3.4
QuanEntr-LP-20	$421 * 41$	QE-LP	23 / 1.6	20/ 9.1
QuanEntr-LP-50	$2551 * 101$	QE-LP	34/ 2.3	20/ 103.4
QuanEntr-LP-100	$10101 * 201$	QE-LP	44/ 14.5	array size error
QuanEntr-LP-SOCP-10-infea	$122 * 21$	QE-LP-SOCP	12/ 0.6 (infea)	16/ 5.1
QuanEntr-LP-SOCP-10	$122 * 21$	QE-LP-SOCP	26/ 1.0	24/ 2.0
QuanEntr-LP-SOCP-20-infea	$441 * 41$	QE-LP-SOCP	14/ 1.1 (infea)	14/ 5.8
QuanEntr-LP-SOCP-20	$441 * 41$	QE-LP-SOCP	33/ 1.8	22/ 4.1
QuanEntr-LP-SOCP-100	$10201 * 201$	QE-LP-SOCP	51/ 20.8	array size error
QuanEntr-LP-SOCP-200	$40402 * 401$	QE-LP-SOCP	69/ 196	array size error
QuanEntr-LP-3norm-10-infea	$121 * 21$	QE-LP-pnorm	16/ 0.6 (infea)	22/ 2.0
QuanEntr-LP-3norm-10	$121 * 21$	QE-LP-pnorm	21/ 1.2	19/ 2.0
QuanEntr-LP-3norm-20-infea	$441 * 41$	QE-LP-pnorm	16/ 0.9 (infea)	18/ 6.5
QuanEntr-LP-3norm-20	$441 * 41$	QE-LP-pnorm	29/ 1.1	20/ 7.3
QuanEntr-LP-3norm-100-infea	$10201 * 201$	QE-LP-pnorm	25/ 15.3 (infea)	array size error
QuanEntr-LP-3norm-100	$10201 * 201$	QE-LP-pnorm	66/ 40.5	array size error

Definition 13.1. *A* ground set E *and a collection of* independent sets $\mathcal{I} \subseteq 2^E$ *form a* matroid $M = (E, \mathcal{I})$ *if:*

- $\emptyset \in \mathcal{I}$,
- *if* $A \in \mathcal{I}$ *and* $B \subset A$, *then* $B \in \mathcal{I}$,
- *if* $A, B \in \mathcal{I}$, *and* $|B| > |A|$, *then there exists* $b \in B \setminus A$ *such that* $A \cup \{b\} \in \mathcal{I}$.

TABLE 6. Results for problems involving Quantum Relative Entropy optimization problems.

Problem	size of A	Type of Constraints	Itr/ time(s)	CVXQUAD
QuanReEntr-6	$73 * 13$	QRE	12/ 6.3	12/ 18.1
QuanReEntr-10	$201 * 21$	QRE	12/ 21.2	25/ 202
QuanReEntr-20	$801 * 41$	QRE	17/ 282	array size error
QuanReEntr-LP-6	$79 * 13$	QRE-LP	25/ 23.2	21/ 19.6
QuanReEntr-LP-6-infea	$79 * 13$	QRE-LP	30/ 11.8	28/ 21.5
QuanReEntr-LP-10	$101 * 21$	QRE-LP	26/ 49.3	24/ 223

The independent sets with maximum cardinality are called the bases of the matroid, we denote the set of bases of the matroid by $\mathcal{B}$. We can also assign a *rank* function $r_M : 2^E \to \mathbb{Z}_+$ as:

$$r_M(A) := \max\{|B| : B \subseteq A, B \in \mathcal{I}\}. \tag{76}$$

Naturally, the rank of a matroid is the cardinality of any basis of it. The *basis generating polynomial* of a matroid is defined as

$$p_M(x) := \sum_{B \in \mathcal{B}} \prod_{i \in B} x_i. \tag{77}$$

A class of hyperbolic polynomials are the basis generating polynomials of certain matroids with the *half-plane property* [9]. A polynomials has the half-plane property if it is nonvanishing whenever all the variables lie in the open right half-plane. We state it as the following lemma:

Lemma 13.1. *Assume that M is a matroid with half-plane property. Then its basis generating polynomial is hyperbolic in any direction in the positive orthant.*

Several classes of matroids have been proven to have the half-plane property [9, 3, 1]. As the first example, we introduce the Vamos matroid. The ground set has size 8 (can be seen as 8 vertices of the cube) and the rank of matroid is 4. All subsets of size 4 are independent except 5 of them. Vamos matroid has the half-plane property [37]. The basis generating polynomial is:

$$p_V(x) := E_4(x_1, \ldots, x_8) - x_1x_2x_3x_4 - x_1x_2x_5x_6 - x_1x_2x_6x_8 - x_3x_4x_5x_6 - x_5x_6x_7x_8,$$

where $E_d(x_1, \ldots, x_m)$ is the elementary symmetric polynomial of degree d with m variables. Note that elementary symmetric polynomials are hyperbolic in direction of the vector of all ones. Some extensions of Vamos matroid also satisfy the half-plane property [6]. These matroids give the following Vamos-like basis generating polynomials:

$$p_{VL}(x) := E_4(x_1, \ldots, x_{2m}) - \left(\sum_{i=2}^{m} x_1x_2x_{2k-1}x_{2k}\right) - \left(\sum_{i=2}^{m-1} x_{2k-1}x_{2k}x_{2k+1}x_{2k+2}\right). \tag{78}$$

Vamos like polynomials in (78) have an interesting property of being counter examples to one generalization of Lax conjecture [4, 6]. Explicitly, there is no power k and symmetric matrices $H_2, \ldots, H_{2m}$ such that

$$(p_{VL}(x))^k = \det(x_1 I + x_2 H_2 + \cdots + x_{2m} H_{2m}). \tag{79}$$

In the DDS package, we have created a function

```
poly = vamos(m)
```

that returns a Vamos like polynomial as in (78) in the `'monomial'` format, which can be given as `cons{k,3}` for inputting a hyperbolic constraint. `vamos(4)` returns the Vamos polynomial with 8 variables.

Graphic matroids also have the half-plane property [9]. However, the hyperbolicity cones arising from graphic matroids are isomorphic to positive semidefinite cones. This can be proved using a classical result that the characteristic polynomial of the bases of graphic matroids is the determinant of the Laplacian (with a row and a column removed) of the underlying graph [5]. Consider a graph $G = (V, E)$ and let $\mathcal{T}$ be the set of all spanning trees of G. Then the generating polynomial defined as

$$T_G(x) := \sum_{T \in \mathcal{T}} \prod_{e \in T} x_e \tag{80}$$

is a hyperbolic polynomial. Several matroid operations preserve the half-plane property as proved in [9], including taking minors, duals, and direct sums.

We have created a set of problems with combinations of hyperbolic polynomial inequalities and other types of constraints such as those arising from LP, SOCP, and entropy function.

TABLE 7. Results for problems involving hyperbolic polynomials.

Problem	size of A	var/deg of $p(z)$	Type of Constraints	Itr/ time(s)
Vamos-1	$8 * 4$	8/4	HB	7/ 1.5
Vamos-LP-1	$12 * 4$	8/4	HB-LP	8/ 2
Vamos-SOCP-1	$17 * 5$	8/4	HB-LP-SOCP	12/ 2
Vamos-Entr-1	$17 * 5$	8/4	HB-LP-Entropy	9/ 1.2
VL-Entr-1	$41 * 11$	20/4	HB-LP-Entropy	9/ 24
VL-Entr-1	$61 * 16$	30/4	HB-LP-Entropy	12/ 211

13.5.2. *Derivatives of products of linear forms.* Hyperbolicity of a polynomial is preserved under directional derivative operation (see [30]):

Theorem 13.1. *Let $p(x) \in \mathbb{R}[x_1, \ldots, x_m]$ be hyperbolic in direction e and of degree at least 2. Then the polynomial*

$$p'_e(x) := (\nabla p)(x)[e] \tag{81}$$

is also hyperbolic in direction e. Moreover,

$$\Lambda(p, e) \subseteq \Lambda(p'_e, e). \tag{82}$$

Assume that $p(x) := l_1(x) \cdots l_\ell(x)$, where $l_1(x), \ldots, l_\ell(x)$ are linear forms. Then, p is hyperbolic in the direction of any vector e such that $p(e) \neq 0$. As an example, consider $E_m(x_1, \ldots, x_m) = x_1 \cdots x_m$. Recursively applying Theorem 13.1 to such polynomials p leads to many hyperbolic polynomials, including all elementary symmetric polynomials. For some properties of hyperbolicity cones of elementary symmetric polynomials, see [38]. For some preliminary computational experiments on such hyperbolic programming problems, see [24].

For the polynomials constructed by the products of linear forms, it is more efficient to use the straight-line program. For a polynomial $p(x) := l_1(x) \cdots l_\ell(x)$, let us define a ℓ-by-m matrix L where the jth row contains the coefficients of l_j. In DDS, we have created a function

```
[poly,poly_grad] = lin2stl(M,d)
```

which returns two polynomials in "straight-line" form, one as the product of linear forms defined by M, and the other one its directional derivative in the direction of d. For example, if we want the polynomial $p(x) := (2x_1 - x_3)(x_1 - 3x_2 + 4x_3)$, then our M is defined as

$$M := \begin{bmatrix} 2 & 0 & -1 \\ 1 & -3 & 4 \end{bmatrix}. \tag{83}$$

Table 8 shows some results of problem where the hyperbolic polynomial is either a product of linear forms or their derivatives.

13.5.3. *Derivatives of determinant of matrix pencils.* Let $H_1, \ldots, H_m \in \mathbb{H}^n$ be Hermitian matrices and $e \in \mathbb{R}^m$ be such that $e_1H_1 + \cdots + e_mH_m$ is positive definite. Then, $p(x) := \det(x_1H_1 + \cdots + x_mH_m)$ is hyperbolic in direction e. Now, by using Theorem 13.1, we can generate a set of interesting hyperbolic polynomials by taking the directional derivative of this polynomial up to $n - 3$ times at direction e.

TABLE 8. Results for problems involving hyperbolic polynomials as product of linear forms and their derivatives.

Problem	size of A	var/deg of $p(z)$	Type of Constraints	Itr
HPLin-LP-1	$55 * 50$	5/3	HB-LP	8
HPLinDer-LP-1	$55 * 50$	5/3	HB-LP	7
HPLin-LP-2	$55 * 50$	5/7	HB-LP	19
HPLinDer-LP-2	$55 * 50$	5/7	HB-LP	22
HPLin-Entr-1	$111 * 51$	10/15	HB-LP-Entropy	19
HPLinDer-Entr-1	$111 * 51$	10/15	HB-LP-Entropy	18
HPLin-pn-1	$111 * 51$	10/15	HB-LP-pnorm	30
HPLinDer-pn-1	$111 * 51$	10/15	HB-LP-pnorm	32
Elem-unb-10	$10 * 5$	10/3	HB	12 (unb)
Elem-LP-10	$15 * 5$	10/3	HB-LP	9
Elem-LP-10-2	$1010 * 1000$	10/3	HB-LP	30
Elem-Entr-10-1	$211 * 101$	10/3	HB-LP-Entropy	17
Elem-Entr-10-2	$221 * 101$	10/4	HB-LP-Entropy	18

REFERENCES

[1] N. AMINI AND P. BRÄNDÉN, *Non-representable hyperbolic matroids*, Adv. Math., 334 (2018), pp. 417–449.

[2] S. BOYD, S.-J. KIM, L. VANDENBERGHE, AND A. HASSIBI, *A tutorial on geometric programming*, Optimization and Engineering, 8 (2007), pp. 67–127.

[3] P. BRÄNDÉN, *Polynomials with the half-plane property and matroid theory*, Advances in Mathematics, 216 (2007), pp. 302–320.

[4] ———, *Obstructions to determinantal representability*, Advances in Mathematics, 226 (2011), pp. 1202–1212.

[5] ———, *Hyperbolicity cones of elementary symmetric polynomials are spectrahedral*, Optimization Letters, 8 (2014), pp. 1773–1782.

[6] S. BURTON, C. VINZANT, AND Y. YOUM, *A real stable extension of the vamos matroid polynomial*, arXiv preprint arXiv:1411.2038, (2014).

[7] V. CHANDRASEKARAN AND P. SHAH, *Relative entropy optimization and its applications*, Mathematical Programming, 161 (2017), pp. 1–32.

[8] R. CHARES, *Cones and Interior-Point Algorithms for Structured Convex Optimization Involving Powers and Exponentials*, PhD thesis, Université Catholique de Louvain, Louvain-la-Neuve, 2008.

[9] Y.-B. CHOE, J. G. OXLEY, A. D. SOKAL, AND D. G. WAGNER, *Homogeneous multivariate polynomials with the half-plane property*, Advances in Applied Mathematics, 32 (2004), pp. 88–187.

[10] J. DAHL AND E. D. ANDERSEN, *A primal-dual interior-point algorithm for nonsymmetric exponential-cone optimization*, Optimization Online, (2019).

[11] C. DAVIS, *All convex invariant functions of hermitian matrices*, Archiv der Mathematik, 8 (1957), pp. 276–278.

[12] S.-C. FANG, J. R. RAJASEKERA, AND H.-S. J. TSAO, *Entropy optimization and Mathematical Programming*, vol. 8, Springer Science & Business Media, 1997.

[13] H. Fawzi, J. Saunderson, and P. A. Parrilo, *Semidefinite approximations of the matrix logarithm*, Foundations of Computational Mathematics, 19 (2019), pp. 259–296.
[14] L. Faybusovich and T. Tsuchiya, *Matrix monotonicity and self-concordance: how to handle quantum entropy in optimization problems*, Optimization Letters, (2017), pp. 1513–1526.
[15] H. A. Friberg, *Cblib 2014: a benchmark library for conic mixed-integer and continuous optimization*, Mathematical Programming Computation, 8 (2016), pp. 191–214.
[16] M. Grant, S. Boyd, and Y. Ye, *CVX: MATLAB software for disciplined convex programming*, 2008.
[17] O. Güler, *Hyperbolic polynomials and interior point methods for convex programming*, Mathematics of Operations Research, 22 (1997), pp. 350–377.
[18] F. Hiai and D. Petz, *Introduction to matrix analysis and applications*, Springer Science & Business Media, 2014.
[19] 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).
[20] ———, *Status determination by interior-point methods for convex optimization problems in domain-driven form*, arXiv preprint arXiv:1901.07084, (2019).
[21] A. S. Lewis, *The mathematics of eigenvalue optimization*, Mathematical Programming, 97 (2003), pp. 155–176.
[22] 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.
[23] MOSEK ApS, *The MOSEK optimization toolbox for MATLAB manual. Version 9.0.*, 2019. http://docs.mosek.com/9.0/toolbox/index.html.
[24] T. G. J. Myklebust, *On primal-dual interior-point algorithms for convex optimisation*, PhD thesis, Department of Combinatorics and Optimization, Faculty of Mathematics, University of Waterloo.
[25] 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.
[26] Y. Nesterov, *Lectures on Convex Optimization*, Springer, 2018.
[27] Y. Nesterov and A. Nemirovski, *Interior-Point Polynomial Algorithms in Convex Programming*, SIAM Series in Applied Mathematics, SIAM: Philadelphia, 1994.
[28] 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.
[29] 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.
[30] J. Renegar, *Hyperbolic programs, and their derivative relaxations*, Foundations of Computational Mathematics, 6 (2006), pp. 59–79.
[31] S. Roy and L. Xiao, *On self-concordant barriers for generalized power cones*, tech. rep., Tech. Report MSR-TR-2018-3, January 2018, https://www. microsoft. com/en-us . . . , 2018.
[32] J. F. Sturm, *Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones*, Optimization Methods and Software, 11 (1999), pp. 625–653.
[33] 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.
[34] J. Tropp, *An introduction to matrix concentration inequalities*, arXiv preprint arXiv:1501.01571, (2015).
[35] L. Tunçel and A. Nemirovski, *Self-concordant barriers for convex approximations of structured convex sets*, Foundations of Computational Mathematics, 10 (2010), pp. 485–525.
[36] R. H. Tütüncü, K.-C. Toh, and M. J. Todd, *Solving semidefinite-quadratic-linear programs using SDPT3*, Mathematical programming, 95 (2003), pp. 189–217.
[37] D. G. Wagner and Y. Wei, *A criterion for the half-plane property*, Discrete Mathematics, 309 (2009), pp. 1385–1390.
[38] Y. Zinchenko, *On hyperbolicity cones associated with elementary symmetric polynomials*, Optim. Lett., 2 (2008), pp. 389–402.

Appendix A. Calculating the predictor and corrector steps

As discussed in [19], for both the predictor and corrector steps, the theoretical algorithm solves a linear system with the LHS matrix of the form

$$U^\top \begin{bmatrix} \begin{bmatrix} H & h \\ h^\top & \zeta \end{bmatrix} & 0 \\ 0 & \begin{bmatrix} G + \eta_* h_* h_*^\top & -\eta_* h_* \\ -\eta_* h_*^\top & \eta_* \end{bmatrix} \end{bmatrix} U \tag{84}$$

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

$$U = \begin{bmatrix} A & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -c_A & -F^\top \\ c^\top & 0 & 0 \end{bmatrix}. \tag{85}$$

H and G are positive definite matrices based on the Hessians of the s.c. functions, F is a matrix whose rows form a basis for null(A), c_A is any vector that satisfies $A^\top c_A = c$, and η_*, ζ, h, and h_* are scalars and vectors defined in [19]. A practical issue of this system is that calculating F is not computationally efficient, and DDS uses a way around it. If we expand the system given in (84), the linear systems DDS needs to solve become of the form:

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

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

$$\begin{aligned} & \eta_* h_* c^\top \bar{d}_x + d_\tau (G + \eta_* h_* h_*^\top) c_A + (G + \eta_* h_* h_*^\top) F^\top d_v = r_3 + A w \\ \Rightarrow \quad & \eta_* \bar{G}^{-1} h_* c^\top \bar{d}_x + d_\tau c_A + F^\top d_v = \bar{G}^{-1} r_3 + \bar{G}^{-1} A w, \end{aligned} \tag{86}$$

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

$$\eta_* A^\top \bar{G}^{-1} h_* c^\top \bar{d}_x + d_\tau c = A^\top \bar{G}^{-1} r_3 + A^\top \bar{G}^{-1} A w. \tag{87}$$

By using the equations in (86) and (87), we can get a linear system without F as

$$\begin{bmatrix} A^\top HA + (\eta_* - \eta_*^2 h_*^\top \bar{G}^{-1} h_*) cc^\top & \eta_* c h_*^\top \bar{G}^{-1} A & A^\top h \\ \eta_* A^\top \bar{G}^{-1} h_* c^\top & -A^\top \bar{G}^{-1} A & c \\ h^\top A & c^\top & \zeta \end{bmatrix} \begin{bmatrix} \bar{d}_x \\ w \\ d_\tau \end{bmatrix} = \begin{bmatrix} r_1 - \eta_* c h_*^\top \bar{G}^{-1} r_3 \\ A^\top \bar{G}^{-1} r_3 \\ r_2 - c_A^\top r_3 \end{bmatrix}. \tag{88}$$

By Sherman–Morrison formula, we can write

$$\begin{aligned} \bar{G}^{-1} &= G^{-1} - \eta_* \frac{G^{-1} h_* h_*^\top G^{-1}}{1 + \eta_* h_*^\top G^{-1} h_*}, \\ \Rightarrow \quad \bar{G}^{-1} h_* &= \frac{1}{1 + \eta_* h_*^\top G^{-1} h_*} G^{-1} h_*. \end{aligned} \tag{89}$$

Let us define

$$\beta := \frac{1}{1 + \eta_* h_*^\top G^{-1} h_*}. \tag{90}$$

Then we can see that the LHS matrix in (88) can be written as

$$\underbrace{\begin{bmatrix} A^\top HA & 0 & A^\top h \\ 0 & -A^\top G^{-1} A & c \\ h^\top A & c^\top & \zeta \end{bmatrix}}_{\tilde{H}} + \eta_* \beta \begin{bmatrix} c \\ A^\top G^{-1} h_* \\ 0 \end{bmatrix} \begin{bmatrix} c \\ A^\top G^{-1} h_* \\ 0 \end{bmatrix}^\top. \tag{91}$$

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 $A^\top HA$ 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. Implementation details for SDP and generalized epigraphs of matrix norms

DDS accepts many constraints that involve matrices. These matrices are given into DDS as vectors. Calculating the derivatives and implementing the required linear algebra operations efficiently is critical in the performance of DDS. In DDS 2.0, many ideas and tricks have been used for matrix functions. In this section, we mention some of these techniques for SDP and matrix norm constraints.

B.1. **SDP.** The primal and dual barrier functions being used for SDP constraints (14) are:

$$\begin{aligned} &\Phi(Z) = -\ln(\det(F_0 + Z)), \\ &\Phi_*(Y) = -n - \langle F_0, Y\rangle - \ln(\det(-Y)). \end{aligned} \tag{92}$$

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

$$\begin{aligned} \langle f'(X), H\rangle &= -\mathrm{Tr}(X^{-1}H), \\ \langle f''(X)H, H\rangle &= \mathrm{Tr}(X^{-1}HX^{-1}H). \end{aligned} \tag{93}$$

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:

$$\begin{aligned} -\mathrm{Tr}(X^{-1}H) &= -g_X^\top \texttt{sm2vec}(H), \\ \mathrm{Tr}(X^{-1}HX^{-1}H) &= \texttt{sm2vec}(H)^\top H_X \texttt{sm2vec}(H). \end{aligned} \tag{94}$$

It can be shown that $g_X = \texttt{sm2vec}(X^{-1})$ and $H_X = X^{-1} \otimes X^{-1}$, where $\otimes$ stands for the Kronecker product of two matrices. Although this representation is theoretically nice, there are two important practical issues with it. First, it is not efficient to calculate the inverse of a matrix explicitly. Second, forming and storing H_X is not numerically efficient for large scale problems. DDS does not explicitly form the inverse of matrices. An important matrix in calculating the search direction is $A^\top \Phi''(u)A$, where Φ is the s.c. barrier for the whole input problem. In DDS, there exist an internal function `hessian_A` that directly returns $A^\top \Phi''(u)A$ in an efficient way, optimized for all the set constraints, including SDP. For transitions from matrices to vectors, we use the properties of Kronecker product that for matrices A, B, and X of proper size, we have

$$\begin{aligned} &(B^\top \otimes A)sm2vec(X) = sm2vec(AXB), \\ &(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}. \end{aligned} \tag{95}$$

Similar to `hessian_A`, there are other internal functions in DDS for efficiently calculating matrix-vector products, such as `hessian_V` for evaluating the product of Hessian with a given vector of proper size.

B.2. **Generalized epigraphs of matrix norms.** Let us see how to calculate the first and second derivatives of functions in (30) and (31).

Proposition B.1. *(a) Consider* $\Phi(X,U)$ *defined in* (30)*. Let, for simplicity,* $\bar{X} := X - UU^\top$*, then, we have*

$$\begin{aligned}
\Phi'(X,U)[(d_X,d_U)] &= \mathrm{Tr}(-\bar{X}^{-1}d_X + \bar{X}^{-1}(d_U U^\top + U d_U^\top)),\\
\Phi''(X,U)[(d_X,d_U),(\bar{d}_X,\bar{d}_U)] &= \mathrm{Tr}(\bar{X}^{-1}d_X\bar{X}^{-1}\bar{d}_X)\\
&\quad -\mathrm{Tr}(\bar{X}^{-1}\bar{d}_X\bar{X}^{-1}(d_U U^\top + U d_U^\top))\\
&\quad -\mathrm{Tr}(\bar{X}^{-1}d_X\bar{X}^{-1}(\bar{d}_U U^\top + U \bar{d}_U^\top))\\
&\quad +\mathrm{Tr}(\bar{X}^{-1}(d_U U^\top + U d_U^\top)\bar{X}^{-1}(\bar{d}_U U^\top + U \bar{d}_U^\top))\\
&\quad +2\mathrm{Tr}(\bar{X}^{-1}d_U\bar{d}_U^\top). \qquad (96)
\end{aligned}$$

(b) Consider $\Phi_*(Y,V)$ *defined in* (31)*, we have*

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

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

$$g(\alpha) := -\ln(\det(X + \alpha d_X - (U+\alpha d_U)(U+\alpha d_U)^\top)), \qquad (97)$$

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

$$f(\alpha) := \mathrm{Tr}((Y+\alpha d_Y)^{-1}VV^\top Y^{-1}d_Y), \qquad (98)$$

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

$$\begin{aligned}
f'(0) &:= \lim_{\alpha\to 0}\frac{f(\alpha)-f(0)}{\alpha}\\
&= \mathrm{Tr}\left(\lim_{\alpha\to 0}\frac{(Y+\alpha d_Y)^{-1}VV^\top Y^{-1}d_Y - Y^{-1}VV^\top Y^{-1}d_Y}{\alpha}\right)\\
&= \mathrm{Tr}\left(\lim_{\alpha\to 0}\frac{(Y+\alpha d_Y)^{-1}\left[VV^\top Y^{-1}d_Y - (I+\alpha d_Y Y^{-1})VV^\top Y^{-1}d_Y\right]}{\alpha}\right)\\
&= \mathrm{Tr}\left(\lim_{\alpha\to 0}(Y+\alpha d_Y)^{-1}\left[d_Y Y^{-1}VV^\top Y^{-1}d_Y\right]\right)\\
&= \mathrm{Tr}\left(Y^{-1}d_Y Y^{-1}VV^\top Y^{-1}d_Y\right). \qquad (99)
\end{aligned}$$

□

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 (96) we have

$$\begin{aligned}\Phi'(X,U)[(d_X,d_U)] &= \mathrm{Tr}(-\bar{X}^{-1}d_X)+\mathrm{Tr}(\bar{X}^{-1}d_UU^\top)+\mathrm{Tr}(X^{-1}Ud_U^\top)),\\ &= \mathrm{Tr}(-\bar{X}^{-1}d_X)+2\mathrm{Tr}(U^\top\bar{X}^{-1}d_U).\end{aligned} \tag{100}$$

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

$$g=\begin{bmatrix} 2\times m2vec(X^{-1}U,n)\\ -sm2vec(X^{-1})\end{bmatrix}. \tag{101}$$

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 (96) 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 jth 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 C. Quantum entropy and quantum relative entropy

The s.c. barrier DDS uses for quantum entropy is given in (56). To derive the LF conjugate, we solve the optimization problem in (57); let us write the first order optimality conditions for (57). Section 3.3 of the book [18] is about the derivation of matrix-values functions. For the first derivative, we have the following theorem:

Theorem C.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,*

$$\frac{d}{dt}\mathrm{Tr}f(X+\alpha H)\Big|_{\alpha=\alpha_0}=\mathrm{Tr}Hf'(X+\alpha_0H). \tag{102}$$

The first-order optimality conditions for (57) can be written as

$$\begin{aligned}\eta+\frac{1}{t-\phi(X)} &= 0\\ Y+\frac{-f'(X)}{t-\phi(X)}+X^{-1} &= 0.\end{aligned} \tag{103}$$

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

$$\frac{1}{\eta}Y+f'(X)+\frac{1}{\eta}X^{-1}=0. \tag{104}$$

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

$$\frac{1}{\eta}\lambda_i(Y) + f'(\lambda_i(X)) + \frac{1}{\eta\lambda_i(X)} = 0, \quad i \in \{1,\ldots,n\}. \tag{105}$$

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 [7] 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 (114). The LF conjugate of (57) is given in the following lemma:

Lemma C.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 (57) becomes*

$$\Phi_*(\eta, Y) = -\ln(-\eta) + \mathrm{Tr}(\Theta + \Theta^{-1}) - \mathrm{Tr}\left(\frac{1}{\eta}Y\right) - 1 - 2n, \tag{106}$$

where $\Theta := \Theta(\frac{1}{\eta}Y + (1 - \ln(-\eta))I)$. Θ is the matrix extension of θ defined in (114) as described in Section 9.

Proof. Assume that for a given (η, Y), (t, X) is the optimal solution for (57). If we use theorem C.1, we have $f'(X) = I + \ln(X)$. By substituting this in the first order optimality condition (104) we get

$$\eta X = -\Theta\left(\frac{1}{\eta}Y + (1 - \ln(-\eta))I\right). \tag{107}$$

By the first equation in (103) and using (107) we can write

$$\begin{aligned} \eta t = -1 + \mathrm{Tr}(\eta X \ln(X)) &= -1 + \mathrm{Tr}(-\Theta \cdot \ln(X)) \\ &= -1 + \mathrm{Tr}\left(\Theta \cdot \left(\frac{1}{\eta}Y + I - \Theta^{-1}\right)\right). \\ &= -1 - n + \mathrm{Tr}(\Theta Y/\eta) + \mathrm{Tr}(\Theta). \end{aligned} \tag{108}$$

If we substitute t and X in (57), we get the result. □

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 C.1 how to calculate the gradient. The following theorem gives us a tool to calculate the Hessian.

Theorem C.2 ([18]-Theorem 3.25)**.** *Assume that $f : (a, b) \mapsto \mathbb{R}$ is a $\mathcal{C}^1$-function and $T = \mathrm{Diag}(t_1,\ldots,t_n)$ with $t_i \in (a, b)$, $i \in \{1,\ldots,n\}$. Then, for a Hermitian matrix H, we have*

$$\frac{d}{dt} f(T + \alpha H)\Big|_{\alpha=0} = T_f \odot H, \tag{109}$$

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

$$T_f := \begin{cases} \frac{f(t_i) - f(t_j)}{t_i - t_j} & t_i \neq t_j \\ f'(t_i) & t_i = t_j \end{cases}. \tag{110}$$

T is diagonal in the statement of the theorem, which is without loss of generality. Note that by the definition of functional calculus in (54), for a Hermitian matrix X and a unitary matrix U, we have

$$f(UXU^*) = Uf(X)U^*. \tag{111}$$

Therefore, for a matrix $T = U\text{Diag}(t_1, \ldots, t_n)U^*$, we can update (109)

$$\frac{d}{dt} f(T + \alpha H)\Big|_{\alpha=0} = U\left(T_f \odot (U^*HU)\right)U^*, \tag{112}$$

where we extend the definition of T_f in (110) to non-diagonal matrices. Now we can use Theorems C.2 and C.1 to calculate the Hessian of a matrix function.

Corollary C.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 $X = U\text{Diag}(\lambda_1, \ldots, \lambda_n)U^*$. Then,*

$$f''(X)[H, \tilde{H}] = \text{Tr}\left(\left(X_f \odot (U^*HU)\right)U^*\tilde{H}U\right). \tag{113}$$

Let us calculate the gradient and Hessian for our functions for $\phi(x) = x\ln(x)$. Let $X = U\text{Diag}(\lambda_1, \ldots, \lambda_n)U^*$ 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 ($\zeta := \frac{1}{t - \text{Tr}(X\ln X)}$)

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

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

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

where $\bar{\Theta}$ is the matrix extension of $\left(\theta' - \frac{\theta'}{\theta^2}\right)$. For the second derivative, let us first define $\bar{Y}$ as in (110):

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

In the expression of the Hessian, we use $\bar{\bar{\Theta}}$ as the matrix extension of $\left(\theta'' - \frac{\theta''\theta - 2(\theta')^2}{\theta^3}\right)$. Then, we have

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

Appendix D. Univariate convex functions

DDS 2.0 uses the epigraph of six univariate convex function, given in Table 2, in the direct sum format to handle important constraints such as geometric and entropy programming. In these section, we show how to calculate the LF conjugate for 2-variable s.c. barriers. Some of these function are implicitly evaluated, but in a computationally efficient way. In the second part of this section, we show that gradient and Hessian of these functions can also be evaluated efficiently.

D.1. **Legendre-Fenchel conjugates of univariate convex functions.** The LF conjugates for the first three functions are given in Table 9. Finding the LF conjugates for the first two

Table 9. 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	$-\ln(t - z\ln(z)) - \ln(z)$	$-\ln(-\eta) + \theta\left(1 + \frac{y}{\eta} - \ln(-\eta)\right) - \frac{y}{\eta} + \frac{1}{\theta\left(1+\frac{y}{\eta}-\ln(-\eta)\right)} - 3$

functions can be handled with easy calculus. In the third row, $\theta(r)$, defined in [25], is the unique solution of

$$\frac{1}{\theta} - \ln(\theta) = r. \tag{114}$$

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:

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

The s.c. barrier DDS uses for the set $|z|^p \le t,\ p \ge 1$, is $\Phi(z,t) = -\ln(t^{\frac{2}{p}} - z^2) - 2\ln(t)$. To find the LF conjugate, we need to solve the following optimization problem:

$$\min_{z,t} \left\{ yz + \eta t + \ln(t^{\frac{2}{p}} - z^2) + 2\ln(t) \right\}. \tag{115}$$

By writing the first order optimality conditions we have:

$$y = \frac{2z}{t^{\frac{2}{p}} - z^2}, \quad \eta = -\frac{\frac{2}{p}t^{\frac{2}{p}-1}}{t^{\frac{2}{p}} - z^2} - \frac{2}{t}. \tag{116}$$

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

$$\begin{aligned} & y\left(\frac{2(\frac{1}{p}+1)+\frac{1}{p}yz}{-\eta}\right)^{\frac{2}{p}} - yz^2 - 2z = 0, \\ & t = \frac{2(\frac{1}{p}+1)+\frac{1}{p}yz}{-\eta}. \end{aligned} \tag{117}$$

Let us define $z(y,\eta)$ as the solution of the first equation in (117). 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 $-z^p \le t,\ 0 \le p \le 1, z > 0$, the corresponding s.c. barrier is $\Phi(z,t) = -\ln(z^p + t) - \ln(z)$. To calculate the LF conjugate, we need to solve the following optimization problem:

$$\min_{z,t} \left\{ yz + \eta t + \ln(z^p + t) + \ln(z) \right\}. \tag{118}$$

By writing the first order optimality conditions we have:

$$y = \frac{-pz^{(p-1)}}{z^p + t} - \frac{1}{z}, \quad \eta = -\frac{1}{z^p + t}. \tag{119}$$

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

$$y - \eta p z^{(p-1)} + \frac{1}{z} = 0. \tag{120}$$

Similar to the previous case, let us define $z(y, \eta)$ as the solution of the first equation in (120). For each pair (y, η), we can calculate $z(y, \eta)$ by few iterations of Newton method. Then, the first and second derivatives 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 10 is the continuation of Table 9.

TABLE 10. 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	$-\ln(t^{\frac{2}{p}} - z^2) - 2\ln(t)$	$-\left(\frac{2}{p} + (\frac{1}{p} - 1)yz(y,\eta)\right) - 2 + 2\ln\left(\frac{2(\frac{1}{p}+1)+\frac{1}{p}yz(y,\eta)}{-\eta}\right)$ $+\ln\left(\left(\frac{2(\frac{1}{p}+1)+\frac{1}{p}yz(y,\eta)}{-\eta}\right)^{\frac{2}{p}} - z^2(y,\eta)\right)$
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:

$$\min_{z,t} \{yz + \eta t + \ln(zt - 1)\}. \tag{121}$$

At the optimal solution, we must have

$$y = -\frac{t}{zt - 1}, \quad \eta = -\frac{z}{zt - 1}. \tag{122}$$

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

$$t = \frac{-1 - \sqrt{1 + 4y\eta}}{2\eta}, \quad z = \frac{-1 - \sqrt{1 + 4y\eta}}{2y},$$
$$\Rightarrow \quad \Phi_*(y,\eta) = -1 - \sqrt{1 + 4y\eta} + \ln\left(\frac{1 + \sqrt{1 + 4y\eta}}{2y\eta}\right). \tag{123}$$

D.2. **Gradient and Hessian.** In this subsection, we show how to calculate the gradient and Hessian for the s.c. barriers and their LF conjugates. Specifically for the implicit function, we show that the derivatives can also be calculated efficiently. First consider the three pairs of functions in Table 9. 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

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

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

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

$$y\left(\frac{2(\frac{1}{p}+1)+\frac{1}{p}yz}{-\eta}\right)^{\frac{2}{p}} - yz^2 - 2z = 0. \tag{124}$$

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

$$\begin{aligned} z'_y &= \frac{B^{\frac{2}{p}} - \frac{2y}{p^2\eta}xB^{\frac{2}{p}-1} - z^2 \quad =: S}{\frac{2y^2}{p^2\eta}B^{\frac{2}{p}-1} + 2yz + 2 \quad =: M} \\ z'_\eta &= \frac{\frac{-2y}{p\eta}B^{\frac{2}{p}} \quad =: T}{\frac{2y^2}{p^2\eta}B^{\frac{2}{p}-1} + 2yz + 2} \\ B &:= \frac{2(\frac{1}{p}+1)+\frac{1}{p}yz}{-\eta} \end{aligned} \tag{125}$$

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

$$\begin{aligned} B'_y &= \frac{z + yz'_y}{-p\eta}, \\ B'_\eta &= \frac{-\frac{\eta}{p}yz'_\eta + 2(\frac{1}{p}+1) + \frac{1}{p}yz}{\eta^2}. \end{aligned} \tag{126}$$

Then we have

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

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

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

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:

$$\nabla\Phi = \begin{bmatrix} \frac{2z}{t^{\frac{2}{p}}-z^2} \\ \\ \frac{-\frac{2}{p}t^{\frac{2}{p}-1}}{t^{\frac{2}{p}}-z^2} - \frac{2}{t} \end{bmatrix},$$

$$\nabla^2\Phi = \begin{bmatrix} \frac{2(t^{\frac{2}{p}}-z^2)+4z^2}{(t^{\frac{2}{p}}-z^2)^2} & \frac{-\frac{4}{p}t^{\frac{2}{p}-1}z}{(t^{\frac{2}{p}}-z^2)^2} \\ \\ \frac{-\frac{4}{p}t^{\frac{2}{p}-1}z}{(t^{\frac{2}{p}}-z^2)^2} & \frac{-\frac{2}{p}\left(\frac{2}{p}-1\right)t^{\frac{2}{p}-2}(t^{\frac{2}{p}}-z^2)+\left(\frac{2}{p}\right)^2 t^{\frac{4}{p}-2}}{(t^{\frac{2}{p}}-z^2)^2} + \frac{2}{t^2} \end{bmatrix}. \tag{127}$$

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

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

(128)

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

$$\nabla^2\Phi_* = \begin{bmatrix} f_{11} & f_{12} \\ f_{21} & f_{22} \end{bmatrix}, \tag{129}$$

where

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

$f_{21} = f_{12}$ 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

$$y - \eta p z^{(p-1)} + \frac{1}{z} = 0 \tag{130}$$

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

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

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:

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

The first derivative of Φ_* is equal to

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

(131)

and the second derivatives of Φ_* is equal to

$$\nabla^2\Phi_* = \begin{bmatrix} f_{11} & f_{12} \\ f_{21} & f_{22} \end{bmatrix}, \tag{132}$$

where

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

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

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 E. A s.c. barrier for vector relative entropy

In section, we prove that the function

$$\Phi(t,u,z) := -\ln\left(t - \sum_{i=1}^{\ell} u_i \ln(u_i/z_i)\right) - \sum_{i=1}^{\ell} \ln(z_i) - \sum_{i=1}^{\ell} \ln(u_i), \tag{133}$$

is a $(2\ell+1)$-LH-s.c. barrier for the epigraph of vector relative entropy function $f : \mathbb{R}^{\ell}_{++} \oplus \mathbb{R}^{\ell}_{++} \to \mathbb{R}$ defined as

$$f(z,u) := \sum_{i=1}^{\ell} u_i \ln(u_i) - u_i \ln(z_i).$$

First note that Φ is $(2\ell+1)$-LH, so we just need to prove self-concordance.

The proof is based on the compatibility results of [27]. We first consider the case $\ell = 1$ and then generalize it. Let us define the map $A : \mathbb{R}^2_{++} \to \mathbb{R}$ as

$$A(u,z) = -u\ln\left(\frac{u}{z}\right). \tag{134}$$

For a vector $d := (d_u, d_z)$, we can verify

$$\begin{aligned} A''[d,d] &= -\frac{(zd_u - ud_z)^2}{uz^2}, \\ A'''[d,d,d] &= \frac{(zd_u + 2ud_z)(zd_u - ud_z)^2}{u^2z^3}. \end{aligned} \tag{135}$$

(135) implies that A is concave with respect to $\mathbb{R}_+$. We claim that A is $(\mathbb{R}_+, 1)$-compatible with the barrier $-\ln(u) - \ln(z)$ ([27]- Definition 5.1.2). For this, we need to show

$$A'''[d,d,d] \le -3A''[d,d]\sqrt{\frac{z^2d_u^2 + u^2d_z^2}{u^2z^2}}. \tag{136}$$

By using (135) and canceling out the common terms from both sides, (136) reduces to

$$(zd_u + 2ud_z) \le 3\sqrt{z^2d_u^2 + u^2d_z^2}. \tag{137}$$

We can assume that the LHS is nonnegative, then by taking the square of both side and reordering, we get the obvious inequality

$$8z^2d_u^2 - 4zd_yud_z + 5u^2d_z^2 = 4z^2d_u^2 + (2zd_u - ud_z)^2 + 4u^2d_z^2 \ge 0. \tag{138}$$

Therefore $A(u,z)$ is $(\mathbb{R}_+, 1)$-compatible with the barrier $-\ln(u) - \ln(z)$. Also note that its summation with a linear term $t + A(u,z)$ is also $(\mathbb{R}_+, 1)$-compatible with the barrier $-\ln(z) - \ln(u)$. Hence, $-\ln(t + A) - \ln(u) - \ln(z)$ is a 3-s.c. barrier by [27]-Proposition 5.1.7.

For the general case, consider the map $\bar{A} : [\mathbb{R}^2_{++}]^{\ell} \to \mathbb{R}$ as

$$\bar{A}(u,z) := \sum_{i}^{\ell} A(u_i, z_i). \tag{139}$$

For a vector d of proper size, we have

$$\begin{aligned} \bar{A}''(u,z)[d,d] &= \sum_i^{\ell} A''(u_i,z_i)[d^i,d^i], \\ \bar{A}'''(u,z)[d,d,d] &= \sum_i^{\ell} A'''(u_i,z_i)[d^i,d^i,d^i]. \end{aligned} \tag{140}$$

We claim that $\bar{A}$ is $(\mathbb{R}_+,1)$-compatible with the barrier $-\sum_i \ln(u_i) - \sum_i \ln(z_i)$. First note that $\bar{A}$ is concave with respect to $\mathbb{R}_+$. We need to prove a similar inequality as (136) for $\bar{A}$. Clearly we have

$$\frac{z_j^2(d_u^j)^2 + u_j^2(d_z^j)^2}{u_j^2 z_j^2} \leq \sum_i^{\ell} \frac{z_i^2(d_u^i)^2 + u_i^2(d_z^i)^2}{u_i^2 z_i^2}, \quad \forall j \in \{1,\ldots,\ell\}. \tag{141}$$

Using inequality (136) and (141) for all j and adding them together yields the inequality we want for $\bar{A}$. Therefore, $\bar{A}$ is $(\mathbb{R}_+,1)$-compatible with the barrier $-\sum_i \ln(u_i) - \sum_i \ln(z_i)$, and by [27]-Proposition 5.1.7, (133) is a $(2\ell+1)$-s.c. barrier.

Appendix F. Comparison with some related solvers and modeling systems

In this section, we take a look at the input format for some other well-known solvers and modeling systems. [22] 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** [23]**.** 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, many convex cone constraints which arise in applications [10]. There are different options for the using platform that can seen in MOSEK's website [23].

F.2. **SDPT3** [36, 33]**.** SDPT3 is a MATLAB package for optimization over symmetric cones, and it solves a conic optimization problem in the equality form as

$$\begin{aligned} \min \quad & \langle c, x \rangle \\ \text{s.t.} \quad & Ax = b, \\ & x \in K, \end{aligned} \tag{142}$$

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 he form

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

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

F.3. **SeDuMi** [32]**.** SeDuMi is also a MATLAB package for optimization over symmetric cones in the format of (142). 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) >= norm(x(12:13))`, `x(14) >= norm(x(15:20))`, and `mat(x(21:36),4)` and `mat(x(37:45),3)` are positive semidefinite. To insert our problem into SeDuMi, we have to write it in the format of (142) . We also have the choice to solve the dual problem because all of the above cones are self-dual.

F.4. **CVX** [16]**.** 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 <= 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.