Noname manuscript No.
(will be inserted by the editor)

Domain-Driven Solver (DDS) Version 2.1: A MATLAB-based Software Package for Convex Optimization Problems in Domain-Driven Form

Mehdi Karimi · Levent Tunçel

Received: date / Accepted: date

Abstract Domain-Driven Solver (DDS) is a MATLAB-based software package for convex optimization. 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 Koecher (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. The infeasible-start primal-dual algorithms used for DDS rely heavily on duality theory and properties of Legendre-Fenchel conjugate functions, and are designed to rigorously determine the status of a given problem. We discuss some important implementation details and techniques we used to improve the robustness and efficiency of the software. The appendix contains many examples.

Keywords convex optimization · software package · self-concordant functions · interior-point methods · primal-dual algorithms · duality theory

Mathematics Subject Classification (2020) 90C25 · 90C51 · 49N15 · 90C90

Mehdi Karimi
Department of Mathematics, Illinois State University, IL, United States.
E-mail: mkarim3@ilstu.edu

Levent Tunçel
Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada.
E-mail: ltuncel@math.uwaterloo.ca

1 Introduction

The code DDS (Domain-Driven Solver) [24] 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, $A \in \mathbb{R}^{m\times n}$ 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* [32,31]. DDS is a practical implementation of the primal-dual algorithm designed and analyzed in [25], which extended many advantages of primal-dual interior-point techniques available for conic formulations, such as the current best complexity bounds, and more robust certificates of approximate optimality, unboundedness, and infeasibility, to the Domain-Driven formulations (covering both conic and non-conic constraints). 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 [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 $i \in \{1, \ldots, \ell\}$. Each 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 listed in Table 1 with their abbreviations. If a user has a nonlinear convex objective function $f(x)$ to minimize, one can

Table 1: Function/set constraints the current version of DDS accepts.

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

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. Some important features of DDS are:

– The algorithm underlying the code fully exploits the properties of Legendre-Fenchel (LF) conjugate Φ_* of Φ defined as

$$\Phi_*(y) := \sup_w \{\langle y, w\rangle - \Phi(w)\}, \tag{2}$$

if it is computationally efficient to evaluate Φ_* and its first (and hopefully second) derivative. For all the underlying sets accepted by DDS, we use established s.c. barriers. For the epigraph of vector relative entropy, we introduce a s.c. barrier (and prove that it is a s.c. barrier). Many of the LF conjugates we derive in this article and efficient evaluation of their gradients and Hessians are original in DDS. We also discuss that using LF conjugates has many advantages such as letting DDS to take longer steps (than the current theoretical guarantees based on the primal barrier alone) and facilitating more robust and efficient computation of proximity measures. MOSEK, the current best commercial solver for conic optimization [28], recently implemented exponential-cone optimization (being accepted by DDS), using the LF conjugate of the barrier function.

– DDS uses a same core algorithm for any combination of the constraints listed in Table 1. Therefore, expanding DDS is easy and a new type of constraint can be added to DDS by the discovery of a corresponding s.c. barrier. The performance enhancement of DDS has been twofold: (1) improving the core algorithm parts such as solving the linear systems for the search directions, and (2) improving the oracles for each function/set constraint such as Hessian-vector products.

Even though there are similarities between DDS and some modeling systems such as CVX [21] (such as the variety of 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 [25].
– The modeling systems for convex optimization that rely on SDP solvers have to use approximations 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 those based on hyperbolic polynomials 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 by DDS than other existing options.

There have been other works to practically apply primal-dual interior-point algorithms based on s.c. barriers to non-symmetric cones. Closed convex cones attain s.c. barriers that are *logarithmically homogeneous* (LH) [32, 31]. [39] proposed an algorithm that requires to evaluate only the primal barrier and its derivatives, similar to the first-order algorithms in [42], and reported interesting computational results for non-symmetric conic optimization. The authors in [34] corrected the analysis in [39] and implemented the algorithms to create a MATLAB-based software package Alfonso [34], and they used it for some classes of optimization problems, for example sum-of-squares [33]. In a recent work, another software package Hypatia has been created in the Julia language [10, 11]. Hypatia is based on the modification of the algorithms in [39] and solves problems involving a direct sum of cones where an efficient LH-s.c. barrier is available for the cone or its dual. All the above three works have established that for many problems, using the *natural formulation* of the problem has advantages over approximating or reformulating the underlying sets with symmetric cones (or standard cones as defined in [10]). There are major differences between our Domain-Driven approach and the others above:

– In the Domain-Driven approach, we allow that the input problem may have both conic and non-conic constraints, whereas the above three approaches are based on conic formulations. [25] has a comprehensive discussion of this issue.
– DDS heavily relies on the theory of duality and exploiting the LF conjugates of the barriers. The algorithms based on [39] avoid using the LF conjugates arguing that their calculation is not numerically efficient. However, in this paper, we show how to calculate the LF conjugate for many of our functions. As we mentioned, MOSEK also uses LF conjugate for exponential-cone optimization [12].
– DDS focuses on the robustness of the status determination and the returned certificate as theoretically analyzed in [26]. This requires utilizing the duality theory to its full extent.

The main part of the article contains the theoretical foundations of DDS. In Section 2, we explain how to use DDS and introduce the list of function/set constraints accepted by DDS. The users' guide on how to add these different types of constraints is included in Appendix A. Calculating the predictor and corrector step efficiently and accurately are critical to the performance of DDS. In Section 3, we explain how such linear systems are being solved for DDS. In Sections 4-12, we discuss the function/set constraints DDS accepts. These sections include the s.c. barrier used for each set and how to evaluate its LF conjugate (except for a few of the constraint types). We also discuss how to efficiently evaluate the gradient and Hessian for complicated functions, and how to calculate the Hessian-vector products. For a few of the constraint types that we do not have efficient ways of computing LF conjugates, DDS uses a primal-heavy version of the algorithms, as discussed in Section 13. Section 14 contains several numerical results.

1.1 Installation

The current version of DDS is written in MATLAB. The link to this version is given in [24]. DDS is also available in the website:

`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 small examples `DDS_example_1.m` and `DDS_example_2.m`.

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

1.2 Notations

Matrices are shown by capital letters (e.g., A, X) and vectors are shown by lower case letters (e.g., c, x). $\mathbb{R}^n$, $\mathbb{R}^n_+$ and $\mathbb{R}^n_{++}$ represent Euclidean n-space, non-negative orthant, and positive orthant, respectively. $\mathbb{S}^m$ is the set of m-by-m symmetric matrices and for $X \in \mathbb{S}^m$, $X \succeq 0$ means that X is positive semidefinite. For a vector or matrix function f, the gradient and Hessian are represented by f' and f''. For a function $f(x)$ and vectors (or matrices) $h, \bar{h}$ from suitable spaces, the first and second order directional derivatives are shown as $f'(x)[h]$ and $f''(x)[h, \bar{h}]$.

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 $D = \mathbb{R}^m_+ + b$, where $b \in \mathbb{R}^m$ is given as part of the input data, and $-\sum_{i=1}^m \ln(x_i)$ is a s.c. barrier for $\mathbb{R}^m_+$. The command in MATLAB that calls DDS to solve problem (1) 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:

$$\inf_y \{\delta_*(y|D) : A^\top y = -c, y \in D_*\}, \tag{3}$$

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

$$D_* := \{y : \langle y, h\rangle \leq 0, \quad \forall h \in \operatorname{rec}(D)\}, \tag{4}$$

where $\operatorname{rec}(D)$ 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:

```
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 [25] as

$$\langle c, x\rangle + \delta_*(y|D). \tag{5}$$

It is proved in [25] that the duality gap is well-defined and zero duality gap implies optimality. If DDS returns `info.status=1` or the status "solved" for a problem, 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 $\operatorname{int} D_i$. All the sets D_i is DDS have an obvious canonical interior point, for example the identity matrix I in $\mathbb{S}^m_+$.

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 $Ax^0 + z^0 \in \operatorname{int} D$. If only `OPTIONS.x0` is given, then x^0 must satisfy $Ax^0 \in \operatorname{int} D$. 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 Framework for the Algorithms in DDS

DDS is based on the algorithms designed and analyzed in [25,26]. Figure 1 shows the diagram of internal process in DDS. The main step is the interior-point algorithm; however, there are multiple pre- and post-processing to prepare data for the algorithm and determine the status of the problem at the end.

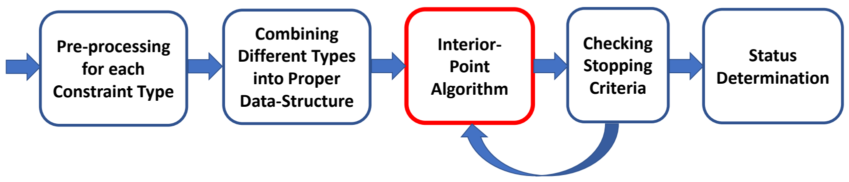

Pre-processing for each Constraint Type
Combining Different Types into Proper Data-Structure
Interior-Point Algorithm
Checking Stopping Criteria
Status Determination

Fig. 1: Diagram for the algorithms in DDS.

A critical step in the algorithm at every iteration is calculating the predictor and corrector directions. At every iteration, the current point is (x, τ, y), where x is the primal vector, y is the dual vector, and τ is an auxiliary variable that DDS uses for status determination. As discussed in [25,26], for both the predictor and corrector steps, we need to solve the following linear system at the point (x, τ, y) with a proper r_{RHS} vector:

$$U^\top \begin{bmatrix} \begin{bmatrix} H & h \\ h^\top & \zeta \end{bmatrix} & 0 \\ 0 & \begin{bmatrix} G + \eta_* h_* h_*^\top & -\eta_* h_* \\ -\eta_* h_*^\top & \eta_* \end{bmatrix} \end{bmatrix} U \begin{bmatrix} \bar{d}_x \\ d_\tau \\ d_v \end{bmatrix} = r_{RHS},$$

$$d_x := \bar{d}_x - d_\tau x, \qquad d_y := -d_\tau c_A - F^\top d_v, \tag{6}$$

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

$$U = \begin{bmatrix} A & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -c_A & -F^\top \\ c^\top & 0 & 0 \end{bmatrix}. \tag{7}$$

A and c are defined in (1), H and G are positive definite matrices based on the Hessians of the primal and dual s.c. functions, F is a matrix whose rows form a basis for null$(A^\top)$, c_A is any vector that satisfies $A^\top c_A = c$, and η_*, ζ, h, and h_* are scalars and vectors defined in [25]. η_*, ζ, h, and h_* are functions of the derivatives and Hessians of Φ and Φ_* defined to guarantee the polynomial time convergence of the algorithm. The following pseudocode gives a framework for the algorithm:

Framework for the Interior-Point Algorithm in DDS

```
INPUT: Internally modified matrices and vectors $A$, $b$, $c$. The oracles for calculating $\Phi$ and its derivatives. If available, the oracles for calculating $\Phi_*$ and its derivatives. A proximity measure $\Omega$ and the constants $0 < \delta_1 < \delta_2$.
while (the stopping criteria are not met)
    Predictor Step
    Calculate the predictor search direction $(d_x, d_\tau, d_y)$ by solving (6) with the proper $r_{RHS}$. Update $(x, \tau, y)$ using the search direction and a step size that guarantees $\Omega(x, \tau, y) \le \delta_2$.
    Corrector Step
    Modify the system and the $r_{RHS}$ in (6) and calculate the corrector search direction $(d_x, d_\tau, d_y)$. Apply the corrector step at least once so that the updated $(x, \tau, y)$ satisfy $\Omega(x, \tau, y) < \delta_1$.
end while
```

A practical issue of the linear system (6) is that calculating the matrix F is not computationally efficient, and DDS uses a way around it. If we expand the system given in (6), the left-hand-side of the linear system reduces to:

$$\begin{bmatrix} A^\top HA + \eta_* cc^\top & A^\top h + \eta_* h_*^\top c_A c & \eta_* ch_*^\top F^\top \\ h^\top A + \eta_* c_A^\top h_* c^\top & \zeta + c_A^\top G c_A + \eta_*(c_A^\top h_*)^2 & c_A^\top GF^\top + \eta_* c_A^\top h_* h_*^\top F^\top \\ \eta_* F h_* c^\top & FGc_A + \eta_* F h_* h_*^\top c_A & FGF^\top + \eta_* F h_* h_*^\top F^\top \end{bmatrix} \begin{bmatrix} \bar{d}_x \\ d_\tau \\ d_v \end{bmatrix}$$

and r_{RHS} gets the form $r_{RHS}^\top = [r_1^\top \; r_2^\top \; (Fr_3)^\top]$. At the end, we need $F^\top d_v$ 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

$$\begin{aligned} &\eta_* h_* c^\top \bar{d}_x + d_\tau (G + \eta_* h_* h_*^\top) c_A + (G + \eta_* h_* h_*^\top) F^\top d_v = r_3 + Aw \\ \Rightarrow\ &\eta_* \bar{G}^{-1} h_* c^\top \bar{d}_x + d_\tau c_A + F^\top d_v = \bar{G}^{-1} r_3 + \bar{G}^{-1} Aw, \qquad (8) \end{aligned}$$

where $\bar{G} := G + \eta_* h_* h_*^\top$. Now, we multiply the last equation by $A^\top$ from the left and use the fact that $A^\top F^\top = 0$ to eliminate the term with d_v and get

$$\eta_* A^\top \bar{G}^{-1} h_* c^\top \bar{d}_x + d_\tau c = A^\top \bar{G}^{-1} r_3 + A^\top \bar{G}^{-1} Aw. \qquad (9)$$

By using the equations in (8) and (9), we can get a linear system without F as

$$\begin{bmatrix} \bar{H}_{11} & \eta_* ch_*^\top \bar{G}^{-1} A & A^\top h \\ \eta_* A^\top \bar{G}^{-1} h_* c^\top & -A^\top \bar{G}^{-1} A & c \\ h^\top A & c^\top & \zeta \end{bmatrix} \begin{bmatrix} \bar{d}_x \\ w \\ d_\tau \end{bmatrix} = \begin{bmatrix} r_1 - \eta_* ch_*^\top \bar{G}^{-1} r_3 \\ A^\top \bar{G}^{-1} r_3 \\ r_2 - c_A^\top r_3 \end{bmatrix},$$

where $\bar{H}_{11} := A^\top HA + (\eta_* - \eta_*^2 h_*^\top \bar{G}^{-1} h_*) cc^\top$. By Sherman–Morrison formula, we can write

$$\bar{G}^{-1} = G^{-1} - \eta_* \frac{G^{-1} h_* h_*^\top G^{-1}}{1 + \eta_* h_*^\top G^{-1} h_*} \quad \Rightarrow \quad \bar{G}^{-1} h_* = \frac{1}{1 + \eta_* h_*^\top G^{-1} h_*} G^{-1} h_*. \tag{10}$$

Let us define

$$\beta := \frac{1}{1 + \eta_* h_*^\top G^{-1} h_*}. \tag{11}$$

Then we can see that the LHS matrix in the last linear system can be written as

$$\underbrace{\begin{bmatrix} A^\top HA & 0 & A^\top h \\ 0 & -A^\top G^{-1} A & c \\ h^\top A & c^\top & \zeta \end{bmatrix}}_{\tilde{H}} + \eta_* \beta \begin{bmatrix} c \\ A^\top G^{-1} h_* \\ 0 \end{bmatrix} \begin{bmatrix} c \\ A^\top G^{-1} h_* \\ 0 \end{bmatrix}^\top. \tag{12}$$

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.

4 Optimization over Symmetric Cones

Optimization problems over symmetric cones, i.e., linear programming (LP), second-order cone programming (SOCP), and semidefinite programming (SDP), are the most important and well-studied classes of convex optimization problems.

4.1 Linear programming (LP) and second-order cone programming (SOCP)

A linear programming (LP) constraint is of the form

$$A_L x + b_L \geq 0, \tag{13}$$

where A_L is an m_L-by-n matrix. An SOCP constraint has the form

$$\|A_S x + b_S\| \leq (g_S)^\top x + d_S, \tag{14}$$

where A_S is an m_S-by-n matrix. The s.c. barriers and their LF conjugates being used in DDS for these constraints are

$$\begin{aligned} &\Phi(z) = -\ln(z), \quad z \in \mathbb{R}_+, \quad \Phi_*(\eta) = -1 - \ln(-\eta), \\ &\Phi(t, z) = -\ln(t^2 - z^\top z), \quad \Phi_*(\eta, w) = -2 + \ln(4) - \ln(\eta^2 - w^\top w). \end{aligned} \tag{15}$$

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

$$\{(z,t,s) \in \mathbb{R}^n \oplus \mathbb{R} \oplus \mathbb{R} : \|z\|^2 \leq ts,\ t \geq 0,\ s \geq 0\}, \tag{16}$$

which is handled by the s.c. barrier $-\ln(ts - z^\top z)$.

4.2 Semidefinite programming (SDP)

An SDP constraint in standard inequality (linear matrix inequality (LMI)) form is of the form

$$F_0 + x_1 F_1 + \cdots + x_n F_n \succeq 0. \tag{17}$$

F_j's are symmetric matrices. The above constraint 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

$$\texttt{vec2sm(sm2vec(X))=X}. \tag{18}$$

It is easy to check that for any pair of symmetric matrices X and Y we have

$$\langle X, Y \rangle := \mathrm{Tr}(XY) = \texttt{sm2vec(X)}^\top \texttt{sm2vec(Y)}. \tag{19}$$

The primal and dual barrier functions being used for the SDP constraint (17) are:

$$\begin{aligned} \Phi(Z) &= -\ln(\det(F_0 + Z)), \\ \Phi_*(Y) &= -n - \langle F_0, Y \rangle - \ln(\det(-Y)). \end{aligned} \tag{20}$$

For function $f(X) = -\ln(\det(X))$ and a symmetric matrix H, we have:

$$\begin{aligned} \langle f'(X), H \rangle &= -\mathrm{Tr}(X^{-1}H), \\ \langle f''(X)H, H \rangle &= \mathrm{Tr}(X^{-1}HX^{-1}H). \end{aligned} \tag{21}$$

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:

$$\begin{aligned} -\mathrm{Tr}(X^{-1}H) &= -g_X^\top \texttt{sm2vec}(H), \\ \mathrm{Tr}(X^{-1}HX^{-1}H) &= \texttt{sm2vec}(H)^\top H_X \texttt{sm2vec}(H). \end{aligned} \tag{22}$$

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 $A^\top \Phi'' A$, where Φ is the s.c. barrier for the whole input problem. In DDS, there exist an internal function `hessian_A` that directly returns $A^\top \Phi'' A$ 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

$$\begin{aligned}&(B^\top \otimes A)\texttt{sm2vec}(X) = \texttt{sm2vec}(AXB),\\ &(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}.\end{aligned} \tag{23}$$

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.

5 Quadratic constraints

Suppose we want to add the following constraints to DDS:

$$x^\top A_Q^\top Q A_Q x + b^\top x + d \leq 0, \tag{24}$$

where A_Q is m_Q-by-n with rank n, and $Q \in \mathbb{S}^{m_Q}$. In general, this constraint may be non-convex and difficult to handle. Currently, DDS handles two cases:

- Q is positive semidefinite,
- Q has exactly one negative eigenvalue. In this case, DDS considers the intersection of the set of points satisfying (24) and a shifted *hyperbolicity cone* defined by the quadratic inequality $y^\top Q y \leq 0$.

If Q is positive semidefinite, then the corresponding constraint in (24) can be written as

$$\begin{aligned}&u^\top u + w + d \leq 0\\ &u := RA_Q x, \quad w := b^\top x, \quad d := d,\end{aligned} \tag{25}$$

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

$$\begin{aligned}\Phi(u,w) &= -\ln(-(u^\top u + w + d)),\\ \Phi_*(y,\eta) &= \frac{y^\top y}{4\eta} - 1 - d\eta - \ln(\eta).\end{aligned} \tag{26}$$

If Q has exactly one negative eigenvalue with eigenvector v, then $-y^\top Q y$ is a hyperbolic polynomial with respect to v. The hyperbolicity cone is the connected component of $y^\top Q y \leq 0$ which contains v and $-\ln(-y^\top Q y)$ is a s.c. barrier for this cone. Therefore, if Q has exactly one negative eigenvalue while $b = 0$ and $d = 0$, DDS considers the hyperbolicity cone defined by the inequality.

6 Generalized Koecher (Power) Cone

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

$$K_\alpha^{(m,n)} := \left\{ (s,u) \in \mathbb{R}_+^m \oplus \mathbb{R}^n : \prod_{i=1}^m s_i^{\alpha_i} \geq \|u\|_2 \right\}, \tag{27}$$

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 [8,43]. The s.c. barrier DDS uses for this set is

$$\Phi(s,u) = -\ln\left(\prod_{i=1}^m s_i^{2\alpha_i} - u^\top u\right) - \sum_{i=1}^m (1-\alpha_i)\ln(s_i), \tag{28}$$

which was conjectured by Chares to be s.c. in [8] and recently proved to be s.c. by Roy and Xiao [38].

7 Epigraphs of matrix norms

Consider a matrix inequality of the form

$$X - UU^\top \succeq 0, \tag{29}$$

where X is a m-by-m symmetric matrix and U is a m-by-n matrix. The set $\{(Z,U) \in \mathbb{S}^m \oplus \mathbb{R}^{m\times n} : Z - UU^\top \succeq 0\}$ accepts 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 $V \in \mathbb{R}^{m\times n}$ [30]. 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)` changes the matrix into a vector. `vec2m(v,m)` reshapes a vector v of proper size to a matrix with m rows. Let us see how to calculate the first and second derivatives of functions in (30) and (31).

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

$$\begin{aligned}
\Phi'(X,U)[(d_X,d_U)] &= \mathrm{Tr}(-\bar{X}^{-1}d_X + \bar{X}^{-1}(d_U U^\top + U d_U^\top)),\\
\Phi''(X,U)[(d_X,d_U),(\bar{d}_X,\bar{d}_U)] &= \mathrm{Tr}(\bar{X}^{-1}d_X\bar{X}^{-1}\bar{d}_X)\\
&\quad -\mathrm{Tr}(\bar{X}^{-1}\bar{d}_X\bar{X}^{-1}(d_U U^\top + U d_U^\top))\\
&\quad -\mathrm{Tr}(\bar{X}^{-1}d_X\bar{X}^{-1}(\bar{d}_U U^\top + U\bar{d}_U^\top))\\
&\quad +\mathrm{Tr}(\bar{X}^{-1}(d_U U^\top + U d_U^\top)\bar{X}^{-1}(\bar{d}_U U^\top + U\bar{d}_U^\top))\\
&\quad +2\mathrm{Tr}(\bar{X}^{-1}d_U\bar{d}_U^\top).
\end{aligned} \tag{32}$$

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

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

Proof For gradients with respect to X and Y, 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

$$g(\alpha) := -\ln(\det(X + \alpha d_X - (U + \alpha d_U)(U + \alpha d_U)^\top)), \tag{33}$$

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 $\Phi_*(Y,V)$. In other words, we can use the definition of directional derivative. For example, let us define

$$f(\alpha) := \mathrm{Tr}((Y + \alpha d_Y)^{-1}VV^\top Y^{-1}d_Y), \tag{34}$$

and we want to calculate $f'(0)$. We have simplify $f'(0) := \lim_{\alpha\to 0}\frac{f(\alpha)-f(0)}{\alpha}$ as

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

□

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 (32) we have

$$\begin{aligned}\Phi'(X,U)[(d_X,d_U)] &= \mathrm{Tr}(-\bar{X}^{-1}d_X) + \mathrm{Tr}(\bar{X}^{-1}d_U U^\top) + \mathrm{Tr}(X^{-1}U d_U^\top)),\\ &= \mathrm{Tr}(-\bar{X}^{-1}d_X) + 2\mathrm{Tr}(U^\top \bar{X}^{-1}d_U). \qquad (35)\end{aligned}$$

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

$$g = \begin{bmatrix} -\texttt{sm2vec}(X^{-1}) \\ 2\times \texttt{m2vec}(X^{-1}U,n)\end{bmatrix}. \qquad (36)$$

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 (32) 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.

7.1 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 [36].

$$\begin{array}{llll}(P_N)\ \min_X & \|X\|_* & (D_N)\ \max_z & \langle b,z\rangle \\ \quad s.t. & A(X)=b. & \quad s.t. & \|A^*(z)\|\le 1,\end{array} \qquad (37)$$

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 rank(X) subject to $A(X)=b$, with applications in machine learning and compressed sensing. The inequality constraint in the dual problem (D_N) is a special case of (29) where $Z=I$ and $U=A^*(z)$. We have $A^*(z)(A^*(z))^\top \preceq I$, which means all the singular values of $A^*(z)$ are at most 1, equivalent to $\|A^*(z)\|\le 1$. Solving (D_N) by `[x,y]=DDS(c,A,b,Z)` leads us to y, which gives a solution for (P_N).

8 Epigraphs of convex univariate functions (geometric, entropy, and *p*-norm programming)

DDS accepts constraints of the form

$$\sum_{i=1}^{\ell} \alpha_i f_i(a_i^\top x + \beta_i) + g^\top x + \gamma \le 0, \qquad (38)$$
$$a_i, g \in \mathbb{R}^n, \quad \beta_i,\gamma\in\mathbb{R}, \quad i\in\{1,\dots,\ell\},$$

where $\alpha_i \geq 0$ and $f_i(x)$, $i \in \{1,\ldots,\ell\}$, 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) [15] with many applications in engineering are constructed with constraints of the form (38) 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. To represent a constraint of the from (38), for given $\gamma \in \mathbb{R}$ and $\beta_i \in \mathbb{R}$,

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

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

$i \in \{1,\ldots,\ell\}$, we can define the corresponding convex set D as

$$D := \{(w, s_i, u_i) : w+\gamma \leq 0,\ \ f_i(s_i+\beta_i) \leq u_i,\ \forall i\}, \tag{39}$$

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. The corresponding s.c. barriers used in DDS are shown in Table 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 as we explain in this section. For these functions, implicit formulas are given using auxiliary functions that can be calculated efficiently by numerical methods. The LF conjugates for the first three functions are given in Table 3. Finding the LF

Table 3: 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	$-\ln(t-z\ln(z))-\ln(z)$	$-\ln(-\eta)+\theta(\bar{y})-\frac{y}{\eta}+\frac{1}{\theta(\bar{y})}-3$ $\bar{y} := 1+\frac{y}{\eta}-\ln(-\eta)$

conjugates for the first two functions can be handled with easy calculus. In the third row, $\theta(r)$, defined in [30], is the unique solution of

$$\frac{1}{\theta} - \ln(\theta) = r. \tag{40}$$

It is easy to check by implicit differentiation that

$$\theta'(r) = -\frac{\theta^2(r)}{\theta(r)+1}, \quad \theta''(r) = -\frac{\theta^2(r)+2\theta(r)}{[\theta(r)+1]^2}\theta'(r). \tag{41}$$

We can calculate $\theta(r)$ with accuracy 10^{-15} in a few steps with the following Newton iterations [30]:

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

The s.c. barrier DDS uses for the set $|z|^p \leq t,\ p \geq 1$, 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 as defined in (2):

$$\max_{z,t} \left\{ yz + \eta t + \ln(t^{\frac{2}{p}} - z^2) + 2\ln(t) \right\}. \tag{42}$$

By writing the first order optimality conditions we have:

$$y = \frac{2z}{t^{\frac{2}{p}} - z^2}, \quad \eta = -\frac{\frac{2}{p}t^{\frac{2}{p}-1}}{t^{\frac{2}{p}} - z^2} - \frac{2}{t}. \tag{43}$$

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

$$\begin{aligned} & y\left(\frac{2(\frac{1}{p}+1) + \frac{1}{p}yz}{-\eta}\right)^{\frac{2}{p}} - yz^2 - 2z = 0, \\ & t = \frac{2(\frac{1}{p}+1) + \frac{1}{p}yz}{-\eta}. \end{aligned} \tag{44}$$

Let us define $z(y,\eta)$ as the solution of the first equation in (44). For each pair (y,η), we can calculate $z(y,\eta)$ by a few iterations of Newton method. Then, the first and second derivatives can be calculated in terms of $z(y,\eta)$.

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

$$\max_{z,t} \left\{ yz + \eta t + \ln(z^p + t) + \ln(z) \right\}. \tag{45}$$

By writing the first order optimality conditions we have:

$$y = \frac{-pz^{(p-1)}}{z^p + t} - \frac{1}{z}, \quad \eta = -\frac{1}{z^p + t}. \tag{46}$$

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

$$y - \eta p z^{(p-1)} + \frac{1}{z} = 0. \tag{47}$$

Similar to the previous case, let us define $z(y, \eta)$ as the solution of the first equation in (47). For each pair (y, η), we can calculate $z(y, \eta)$ by a few iterations of Newton method. Then, the first and second derivatives can be calculated in terms of $z(y, \eta)$. In other words, if we have an oracle to efficiently compute $z(y, \eta)$, then the derivatives can be calculated explicitly by simple algebraic formulas. Table 4 is the continuation of Table 3.

Table 4: 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	$-\ln(t^{\frac{2}{p}} - z^2) - 2\ln(t)$	$-\left(\frac{2}{p} + (\frac{1}{p} - 1) y z(y,\eta)\right) - 2 + 2\ln\left(\frac{2(\frac{1}{p}+1)+\frac{1}{p} y z(y,\eta)}{-\eta}\right) + \ln\left(\left(\frac{2(\frac{1}{p}+1)+\frac{1}{p} y z(y,\eta)}{-\eta}\right)^{\frac{2}{p}} - z^2(y,\eta)\right)$
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:

$$\max_{z,t} \left\{ yz + \eta t + \ln(zt - 1) \right\}. \tag{48}$$

At the optimal solution, we must have

$$y = -\frac{t}{zt - 1}, \quad \eta = -\frac{z}{zt - 1}. \tag{49}$$

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

$$t = \frac{-1 - \sqrt{1 + 4y\eta}}{2\eta}, \quad z = \frac{-1 - \sqrt{1 + 4y\eta}}{2y},$$
$$\Rightarrow \Phi_*(y, \eta) = -1 - \sqrt{1 + 4y\eta} + \ln\left(\frac{1 + \sqrt{1 + 4y\eta}}{2y\eta}\right). \tag{50}$$

8.1 Evaluating the gradient and Hessian for bivariate functions and their LF conjugates

For four LF conjugates, we derived implicit formulas which depend on efficient evaluations of univariate $\theta(r)$ or two-variable $z(y, \eta)$ functions. We claim that by one evaluation of these functions, we can calculate the gradient and Hessian in addition to the function itself. By (41), both θ' and θ'' are functions of θ. Therefore, we can calculate the gradient and Hessian of $\Phi_*(y,\eta)$ if $\theta(r)$ is calculated. For the other three functions that depend on $z(y,\eta)$, we can show a similar result. For example, for the type 4 function, the conjugate function is based on $z(y,\eta)$ as the solution of

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

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

$$z'_y = \frac{B^{\frac{2}{p}} - \frac{2y}{p^2\eta}xB^{\frac{2}{p}-1} - z^2}{\frac{2y^2}{p^2\eta}B^{\frac{2}{p}-1} + 2yz + 2}, \quad z'_\eta = \frac{\frac{-2y}{p\eta}B^{\frac{2}{p}}}{\frac{2y^2}{p^2\eta}B^{\frac{2}{p}-1} + 2yz + 2},$$

where $B := \frac{2(\frac{1}{p}+1)+\frac{1}{p}yz}{-\eta}$. Similarly, we can derive z''_{yy}, $z''_{\eta y}$, and $z''_{\eta\eta}$ for the second derivatives, all as the function of z. Therefore, with one accurate evaluation of $z(y,\eta)$, we can explicitly calculate the gradient and Hessian of the corresponding LF conjugate.

9 Vector Relative Entropy

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

$$f(u,z) := \sum_{i=1}^{\ell} u_i \ln(u_i) - u_i \ln(z_i). \tag{51}$$

This function covers many applications as discussed in [12]. The special case of $\ell = 1$ is used for handling EXP cone in the commercial solver MOSEK. We prove that the epigraph of this function accepts the following $(2\ell+1)$-s.c. barrier:

$$\Phi(t,u,z) := -\ln\left(t - \sum_{i=1}^{\ell} u_i \ln(u_i/z_i)\right) - \sum_{i=1}^{\ell} \ln(u_i) - \sum_{i=1}^{\ell} \ln(z_i). \tag{52}$$

Theorem 1 *The function defined in* (52) *is a* $(2\ell+1)$*-LH-s.c. barrier for the epigraph of vector relative entropy function* f *defined in* (51).

Proof Since Φ is $(2\ell+1)$-LH, we just need to prove self-concordance. The proof is based on the compatibility results of [32]. We first consider the case $\ell = 1$ and then generalize it. Let us define the map $A : \mathbb{R}^2_{++} \to \mathbb{R}$ as

$$A(u,z) = -u\ln\left(\frac{u}{z}\right). \tag{53}$$

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

$$A''[d,d] = -\frac{(zd_u - ud_z)^2}{uz^2},$$
$$A'''[d,d,d] = \frac{(zd_u + 2ud_z)(zd_u - ud_z)^2}{u^2z^3}. \tag{54}$$

(54) implies that A is concave with respect to $\mathbb{R}_+$. We claim that A is $(\mathbb{R}_+, 1)$-compatible with the barrier $-\ln(u) - \ln(z)$ ([32]- Definition 5.1.2). For this, we need to show

$$A'''[d,d,d] \leq -3A''[d,d]\sqrt{\frac{z^2d_u^2 + u^2d_z^2}{u^2z^2}}. \tag{55}$$

By using (54) and canceling out the common terms from both sides, (55) reduces to

$$(zd_u + 2ud_z) \leq 3\sqrt{z^2d_u^2 + u^2d_z^2}. \tag{56}$$

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

$$8z^2d_u^2 - 4zd_yud_z + 5u^2d_z^2 = 4z^2d_u^2 + (2zd_u - ud_z)^2 + 4u^2d_z^2 \geq 0. \tag{57}$$

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 [32]-Proposition 5.1.7.

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

$$\bar{A}(u,z) := \sum_i^\ell A(u_i, z_i). \tag{58}$$

For a vector d of proper size, we have

$$\bar{A}''(u,z)[d,d] = \sum_i^\ell A''(u_i,z_i)[d^i,d^i],$$
$$\bar{A}'''(u,z)[d,d,d] = \sum_i^\ell A'''(u_i,z_i)[d^i,d^i,d^i]. \tag{59}$$

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 (55) for $\bar{A}$. Clearly we have

$$\frac{z_j^2(d_u^j)^2 + u_j^2(d_z^j)^2}{u_j^2z_j^2} \leq \sum_i^\ell \frac{z_i^2(d_u^i)^2 + u_i^2(d_z^i)^2}{u_i^2z_i^2}, \quad \forall j \in \{1,\ldots,\ell\}. \tag{60}$$

Using inequality (55) and (60) 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 [32]-Proposition 5.1.7, (52) is a $(2\ell + 1)$-s.c. barrier. $\square$

We claim that the LF of Φ can be efficiently calculated using ℓ 1-dim 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

$$\begin{aligned}&\Phi_*(\alpha, y_u, y_z) := \\ &\min_{t,z,u}\{\alpha t + y_z z + y_u u + \ln(t - u\ln(u) + u\ln(z)) + \ln(z) + \ln(u)\}.\end{aligned}$$

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

$$\begin{array}{ll}\alpha + \frac{1}{T} & = 0 \\ y_z + \frac{u}{zT} + \frac{1}{z} & = 0 \\ y_u - \frac{1+\ln(u)-\ln(z)}{T} + \frac{1}{u} & = 0.\end{array} \tag{61}$$

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:

$$\frac{1}{T} + \frac{1}{u} = \frac{1}{\theta\left(-Ty_u + 2 + \ln(-y_z) + \ln(T)\right)}. \tag{62}$$

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

10 Quantum entropy and Quantum relative entropy

Quantum entropy and quantum relative entropy functions are important in quantum information processing. DDS 2.1 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 $X \in \mathbb{H}^n$ ($\mathbb{H}^n$ is the set of n-by-n Hermitian matrices) 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

$$\phi(X) := \begin{cases}\mathrm{Tr}(U\mathrm{Diag}(f(\lambda_1), \ldots, f(\lambda_n))U^*) & \text{if } f(\lambda_i) \in \mathbb{R},\ \forall i, \\ +\infty & \text{o.w.}\end{cases} \tag{63}$$

Study of such matrix functions go back to the work of Löwner as well as Von-Neumann (see [13], [27], and the references therein). A function $f : (a, b) \mapsto \mathbb{R}$ is said to be matrix monotone if for any two self-adjoint matrices 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

$$F(tX + (1-t)Y) \preceq tF(X) + (1-t)F(Y), \quad \forall t \in (0, 1). \tag{64}$$

Faybusovich and Tsuchiya [18] 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 (63). Then, the function

$$\varPhi(t, X) := -\ln(t - \phi(X)) - \ln\det(X) \tag{65}$$

is a $(n+1)$-s.c. barrier for the epigraph of $\phi(X)$. Many optimization problems arising in quantum information theory and some other areas require dealing with the so-called *quantum* or *von Neumann entropy* $qe : \mathbb{H}^n \to \mathbb{R} \cup \{+\infty\}$ which is defined as $qe(X) := \text{Tr}(X\ln(X))$. If we consider $f(x) := x\ln(x)$, then $f'(x) = 1+\ln(x)$ is matrix monotone on $(0, \infty)$ (see, for instance [23]-Example 4.2) and so we have a s.c. barrier for the set

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

We have to solve the optimization problem

$$\varPhi_*(\eta, Y) = \sup_{t,X} t\eta + \langle X, Y\rangle + \ln(t - \phi(X)) + \ln\det(X), \tag{66}$$

to calculate the LF conjugate of (65), which we will do in a following subsection.

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

$$qre(X, Y) := \text{Tr}(X\ln(X) - X\ln(Y)).$$

This function is convex as proved in [41]. The epigraph of qre is

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

DDS has been using the following barrier function since 2019 for solving problems involving quantum relative entropy constraints:

$$\varPhi(t, X, Y) := \ln(t - qre(X, Y)) - \ln\det(X) - \ln\det(Y), \tag{67}$$

which was proved to be self-concordant very recently by Fawzi and Saunderson [16]. DDS accepts constraints involving quantum entropy and quantum relative entropy. In the following, we see how to calculate the derivatives and Hessians for the above s.c. barriers and also the LF conjugate given in (66).

10.1 Evaluating the LF conjugate and derivatives for quantum entropy

The s.c. barrier DDS uses for quantum entropy is given in (65). To derive the LF conjugate, we solve the optimization problem in (66); let us write the first order optimality conditions for (66). For the first derivative, we have the following theorem:

Theorem 2 ([23]-Section 3.3) *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,*

$$\frac{d}{d\alpha}\mathrm{Tr}F(X+\alpha H)\bigg|_{\alpha=\alpha_0} = \mathrm{Tr}HF'(X+\alpha_0 H). \tag{68}$$

The first-order optimality conditions for (66) can be written as

$$\begin{aligned} \eta + \frac{1}{t-\phi(X)} &= 0 \\ Y + \frac{-\nabla\phi(X)}{t-\phi(X)} + X^{-1} &= 0. \end{aligned} \tag{69}$$

Note that by (68), we have $\nabla\phi(X) = F'(X)$, where F' is the matrix extension of f'. If we substitute the first equation in the second one, we get

$$\frac{1}{\eta}Y + F'(X) + \frac{1}{\eta}X^{-1} = 0. \tag{70}$$

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

$$\frac{1}{\eta}\lambda_i(Y) + f'(\lambda_i(X)) + \frac{1}{\eta\lambda_i(X)} = 0, \quad i \in \{1, \dots, n\}. \tag{71}$$

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 (40). The LF conjugate of (66) is given in the following lemma:

Lemma 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 (66) becomes*

$$\Phi_*(\eta, Y) = -\ln(-\eta) + \mathrm{Tr}(\Theta + \Theta^{-1}) - \mathrm{Tr}\left(\frac{1}{\eta}Y\right) - 1 - 2n, \tag{72}$$

where $\Theta := \Theta(\frac{1}{\eta}Y + (1-\ln(-\eta))I)$. Θ is the matrix extension of θ defined in (40) as described in Section 10.

Proof Assume that for a given (η, Y), (t, X) is the optimal solution for (66). If we use Theorem 2, we have $F'(X) = I + \ln(X)$. By substituting this in the first order optimality condition (70) we get

$$\eta X = -\Theta\left(\frac{1}{\eta}Y + (1-\ln(-\eta))I\right). \tag{73}$$

By the first equation in (69) and using (73) we can write

$$\begin{aligned}\eta t = -1 + \mathrm{Tr}(\eta X \ln(X)) &= -1 + \mathrm{Tr}(-\Theta \cdot \ln(X)) \\ &= -1 + \mathrm{Tr}\left(\Theta \cdot \left(\frac{1}{\eta} Y + I - \Theta^{-1}\right)\right). \\ &= -1 - n + \mathrm{Tr}(\Theta Y/\eta) + \mathrm{Tr}(\Theta). \end{aligned} \tag{74}$$

If we substitute t and X in (66), we get the result. □

For the rest of our discussion, we need two definitions for univariate functions similar to the derivative. For a continuously differentiable function $f : (a,b) \mapsto \mathbb{R}$, we define the first and second divided differences as

$$\begin{aligned} f^{[1]}(\alpha,\beta) &= \begin{cases} \frac{f(\alpha)-f(\beta)}{\alpha-\beta} & \alpha \neq \beta \\ f'(\alpha) & \alpha = \beta \end{cases} \\ f^{[2]}(\alpha,\beta,\gamma) &= \begin{cases} \frac{f^{[1]}(\alpha,\beta)-f^{[1]}(\alpha,\gamma)}{\beta-\gamma} & \beta \neq \gamma \\ \frac{f^{[1]}(\alpha,\beta)-f'(\beta)}{\alpha-\beta} & \beta = \gamma \neq \alpha \\ -\frac{1}{2} f''(\alpha) & \beta = \gamma = \alpha \end{cases} \end{aligned} \tag{75}$$

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

Theorem 3 ([23]-Theorem 3.25) *Assume that $f : (a,b) \mapsto \mathbb{R}$ is a $\mathcal{C}^1$-function and $T = \mathrm{Diag}(t_1,\dots,t_n)$ with $t_i \in (a,b)$, $i \in \{1,\dots,n\}$. Then, for a Hermitian matrix H, we have*

$$\frac{d}{d\alpha} F(T+\alpha H)\bigg|_{\alpha=0} = T_f \odot H, \tag{76}$$

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

$$[T_f]_{ij} := f^{[1]}(t_i,t_j), \quad \forall i,j \in \{1,\dots,n\}. \tag{77}$$

T is diagonal in the statement of the theorem, which is without loss of generality. Note that by the definition of functional calculus in (63), for a Hermitian matrix X and a unitary matrix U, we have

$$F(UXU^*) = UF(X)U^*. \tag{78}$$

Therefore, for a matrix $T = U\mathrm{Diag}(t_1,\dots,t_n)U^*$, we can update (76)

$$\frac{d}{d\alpha} F(T+\alpha H)\bigg|_{\alpha=0} = U\left(T_f \odot (U^*HU)\right)U^*, \tag{79}$$

where we extend the definition of T_f in (77) to non-diagonal matrices. Now we can use Theorems 3 and 2 to calculate the Hessian of a matrix function.

Corollary 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 $X = U\mathrm{Diag}(\lambda_1, \dots, \lambda_n)U^*$. Then,*

$$\nabla^2 \mathrm{Tr}(F(X))[H, \tilde{H}] = \mathrm{Tr}\left((X_f \odot (U^*HU))\, U^*\tilde{H}U \right). \tag{80}$$

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

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

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 ($\zeta := \frac{1}{t - \mathrm{Tr}(X \ln X)}$)

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

Now let us compute the gradient and Hessian for the conjugate function. Let $Y = U\lambda(Y)U^*$, by using Theorem 2, the gradient of $\Phi_*(\eta, Y)$ is

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

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 (77):

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

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

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

10.2 Evaluating the derivatives for quantum relative entropy

In this subsection, we discuss how to calculate the gradient and Hessian for the function in (67). This reduces to calculating the gradient and Hessian for the $qre(X,Y)$ function. Calculating the derivatives of $X\ln(X)$ has been discussed for the quantum entropy and we need to handle $-X\ln(Y)$. For a fixed matrix X and a continuously differentiable function $f : (a,b) \mapsto \mathbb{R}$, let us define

$$F_X(Y) := \operatorname{Tr}(XF(Y)). \tag{81}$$

The gradient and Hessian of $F_X(Y)$ can be calculated as follows [19]. Assume that the spectral decomposition of Y is $Y = U\operatorname{Diag}(\lambda_1, \ldots, \lambda_n)U^*$. Then we have

$$F_X'(Y) = U\left(Y_f \odot (U^*XU)\right)U^*. \tag{82}$$

For the Hessian of F_X, we have:

$$F_X''(Y) = (U \otimes U)S(U^* \otimes U^*), \tag{83}$$

where S is the n^2-by-n^2 second divided difference matrix. If we assume that S is a block matrix of size n-by-n where each block is again a matrix of size n-by-n, then we can show the entries of S as $S_{ij,kl}$ where ij denotes the place of the block, and kl denotes the rows and columns inside the block. We have:

$$S_{ij,kl} = \delta_{kl}\tilde{X}_{ij}f^{[2]}(\lambda_i, \lambda_j, \lambda_l) + \delta_{ij}\tilde{X}_{kl}f^{[2]}(\lambda_j, \lambda_k, \lambda_l), \tag{84}$$

where $\tilde{X} := U^*XU$ and δ_{ij} is an indicator function which is 1 if $i = j$ and 0 otherwise.

11 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 2.1 accepts constraints based on hyperbolic polynomials. Let us first define a hyperbolic polynomial. A polynomial $p(x) \in \mathbb{R}[x_1, \dots, x_m]$ is said to be *homogeneous* if every term has the same degree d. A homogeneous polynomial $p : \mathbb{R}^m \to \mathbb{R}$ 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 $p(x) \in \mathbb{R}[x_1, \dots, x_m]$ 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

$$\Lambda_+(p, e) := \{x \in \mathbb{R}^m : \lambda(x) \geq 0\}. \tag{85}$$

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

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

Theorem 4 (Güler [22]) *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 13.

11.1 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` 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 may use the following representation:

$$\begin{bmatrix} 1 & 1 & 1 & 33 \\ -1 & 2 & 2 & 33 \\ -1 & 3 & 3 & 33 \\ 1 & 4 & 5 & 11 \\ 1 & 6 & 7 & 11 \end{bmatrix} \begin{array}{l} \rightarrow\; f_4 = (1)(f_1 \times f_1) = x_1^2 \\ \quad\rightarrow\; f_5 = (-1)(f_2 \times f_2) = -x_2^2 \\ \qquad\rightarrow\; f_6 = (-1)(f_3 \times f_3) = -x_3^2 \\ \qquad\quad\rightarrow\; f_7 = (1)(f_4 + f_5) = x_1^2 - x_2^2 \\ \qquad\qquad\rightarrow\; f_8 = (1)(f_6 + f_7) = x_1^2 - x_2^2 - x_3^2 \end{array}$$

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

$$p(x) = \det(H_0 + x_1 H_1 + x_2 H_2 + \cdots + x_m H_m), \tag{86}$$

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.

12 Linear 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 an alternative form, users can input a problem of the form

$$\inf_x \{\langle c, x\rangle : Bx = d,\ Ax \in D\}. \tag{87}$$

In the main Domain-Driven formulation (1), due to 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 (87) is equivalent to:

$$D_w := \{w : AZw \in (D - Ax^0)\}. \tag{88}$$

$D - Ax^0$ 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 (87) 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.

13 Primal-heavy version of the algorithm

For some classes of problems, such as hyperbolic programming, a computable s.c. barrier Φ is available for the set D, while the LF of it may not be 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

$$z = \Phi'_*(\Phi'(z)), \quad \Phi''_*(\Phi'(z)) = [\Phi''(z)]^{-1}. \tag{89}$$

Instead of the primal-dual proximity measure defined in [25], we use the primal-heavy version:

$$\left\| \frac{\tau y}{\mu} - \Phi'(u) \right\|_{[\Phi''(u)]^{-1}}, \tag{90}$$

where $u := Ax + \frac{1}{\tau}z^0$, τ is an artificial variable we use in the formulation of the central path (see [25] for details), and μ is the parameter of the central path. By [25]-Corollary 4.1, this proximity measure is "equivalent" to the primal-dual one, but (90) is less efficient computationally.

By using a primal-heavy version, we lose some of the desired properties of the 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 (5). For a general convex domain D, we need Φ'_* to accurately calculate $\delta_*(y|D)$ as explained in [25]. Note that when D is a shifted cone $D = K - b$, then we have

$$\delta_*(y|D) = -\langle b, y\rangle. \tag{91}$$

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

$$\left\| \frac{\tau y}{\mu} - \Phi'(u) \right\|_{\Phi''(u)} < 1, \tag{92}$$

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.`

14 Numerical Results

In this section, we present some numerical examples of running DDS 2.1. We performed computational experiments using the software MATLAB R2022a, on a 1.7 GHz 12th Gen Intel Core i7 personal computer with 32GB 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 [26], which studied how to efficiently certify different status of a given problem in Domain-Driven form, using duality theory. [26]-Section 9 discusses stopping criteria in a practical setting. We define the following parameters for the current point (x, τ, y):

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

where x and y are the primal and dual points, and τ is the artificial variable we use (see [25] for details). z^0 is the initial point such that $Ax + \frac{1}{\tau}z^0 \in D$ at every iteration for A and D defined in (1). DDS stops the algorithm and returns the status as "solved" when we have

$$\max\{gap, P_{feas}, D_{feas}\} \leq tol. \tag{93}$$

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

$$\|A^\top \bar{y}\| \leq tol, \quad \delta_*(\bar{y}|D) < 0, \tag{94}$$

and returns the status as "unbounded" if

$$\langle c, x\rangle \leq -\frac{1}{tol}. \tag{95}$$

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.

14.1 Robustness in Status Determination

As mentioned, DDS highly exploits the duality theory to rigorously determine the problem's status. To achieve robustness, DDS takes more conservative steps to stay closer to the central path, compared to other solvers using interior-point methods, such as MOSEK and Hypatia. This approach sacrifices some speed to gain some more robustness. For example, consider the following optimization problem:

$$\begin{array}{lr} \min \sum_{i=1}^{\ell} u_i \ln(u_i) - u_i \ln(z_i) & \\ s.t. \qquad \sum_{i=1}^{\ell} u_i \ln(u_i) \leq 3 & \\ \qquad\quad \sum_{i=1}^{\ell} z_i \ln(z_i) \leq 4 & \\ \qquad\qquad u_1 \leq -0.01. & \end{array} \tag{96}$$

This problem can be input into DDS, CVX, and Hypatia. For DDS, we can use RE, TD, and LP type constraints. For CVX, we can use functions `rel_entr` and `entr` to input this problem. For Hypatia, we can handle the objective and the constraints by using the cone `RelativeEntropyCone`. Without the constraint $u_1 \leq -0.01$, the problem has an optimal solution. For $\ell = 100$, all three solvers solve the problem with optimal value -38.23. With the linear constraint $u_1 \leq -0.01$, the problem is infeasible. DDS returns a certificate of infeasibility in 52 iterations. As this problem can not be represented by symmetric cones exactly, CVX uses successive approximation. The returned status is infeasible, but with no certificate. Hypatia cannot detect the status and returns "SLOW_PROGRESS". Hypatia is a fast solver comparable in many cases to MOSEK [11], and this speed seems to come with some cost of losing some robustness in a few boundary cases as we will see in more examples.

Table 5: Numerical results for some problem from the Dimacs library

Problem	size of A	Type of Constraints	Iter/time(sec)
nb	$2383 * 123$	SOCP-LP	41/16.1
nb_L1	$3176 * 915$	SOCP-LP	38/92.0
nb_L2	$4195 * 123$	SOCP-LP	25/22.2
nb_L2_bessel	$2641 * 123$	SOCP-LP	27/13.0
filter48_socp	$3284 * 969$	SDP-SOCP-LP	81/6.25
filtinf1	$3395 * 983$	SDP-SOCP-LP	26/2.31
truss5	$3301 * 208$	SDP	66/5.49
truss8	$11914 * 496$	SDP	76/17.0
copo14	$3108 * 1275$	SDP-LP	21/2.53
copo23	$13938 * 5820$	SDP-LP	31/38.5
toruspm3-8-50	$262144 * 512$	SDP	20/25.3
torusg3-8	$262144 * 512$	SDP	29/38.3
sched_50_50_scaled	$4977 * 2526$	SOCP-LP	77/67.1
mater-3	$39448 * 1439$	SDP-LP	124/255
cnhil8	$14400 * 1716$	SDP	31/6.58
cnhil10	$48400 * 5005$	SDP	35/40.1
cphil10	$48400 * 5005$	SDP	12/11.5
ros_500	$17944 * 4988$	SDP	39/309
sensor_500	$245601 * 3540$	SDP	66/175
taha1a	$231672 * 3002$	SDP	24/284
G40mc	$4000000 * 2000$	SDP	34/1080
1tc.1024	$1048576 * 7937$	SDP	45/504
yalsdp	$30000 * 5051$	SDP	17/21.0

14.2 LP-SOCP-SDP

This subsection considers LP-SOCP-SDP instances mostly from the Dimacs library [35]. Note that the problems in the library are for the standard equality form and we solve the duals of the given problems. Table 5 shows the results. DDS has a built-in function `read_sedumi_DDS` to read problems in SeDuMi input format. One may 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);
```

14.3 EXP cone optimization problems from CBLIB

Conic Benchmark Library (CBLIB) [20] 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 9 when $\ell = 1$. Table 6 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. This table also contains the iteration count and running time for Hypatia [1]. As can be seen, on "well-conditioned" problems, Hypatia takes fewer iterations, since it takes more aggressive steps than DDS. However, there are four cases where it fails, specifically for the only infeasible problem `isil01`.

14.4 Minimizing Nuclear Norm

In this subsection, we use DDS to solve the problem of minimizing nuclear norm of a matrix subject to linear constraints:

$$\begin{aligned} &\min \|X\|_* \\ &s.t.\ \operatorname{Tr}(U_i X) = c_i, \quad i \in \{1, \ldots, \ell\}, \end{aligned} \tag{97}$$

where X is n-by-m. We are interested in the case that $n \gg m$, which makes the s.c. barrier for the epigraph of a matrix norm more efficient than the SDP representation. We compare DDS with the convex modeling system CVX Version 2.2 [21] with SDPT3 [40] as the solver, which has a function `norm_nuc` for nuclear norm. For numerical results, we assume (97) 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 2 shows the plots of running time for both DDS and CVX for different values of m. For the epigraph of the 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.

14.5 Quantum Entropy and Quantum Relative Entropy

In this subsection, we focus on optimization problems involving quantum entropy. As far as we know, there is no library for quantum 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 quantum entropy are of the form

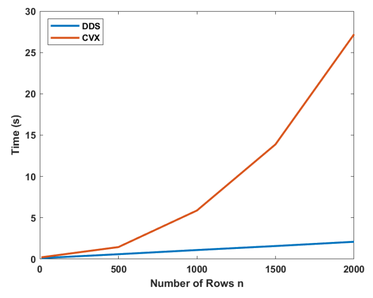
$$\begin{aligned} \min\ & t \\ \text{s.t.}\ & qe(A_0 + x_1 A_1 + \cdots + x_n A_n) \le t, \\ & \text{other constraints}, \end{aligned} \tag{98}$$

[1] For these results, the code provided in the Hypatia package for CBLIB using JuMP [14] is used, with the parameter `default_tol_relax =1`. By changing the parameter to `default_tol_relax =1000`, Hypatia solves the problems `batch` and `enpro48` approximately, but fails again for `isil01` and `LogExpCR-n500-m1600`.

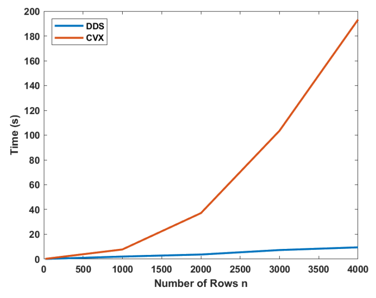
Table 6: Numerical results for some EXP cone problems from CBLIB. Times are in seconds.

Problem	size of A	size of A_{EQ}	Iter/time DDS	Iter/time Hypatia (JuMP)
syn30m	$324 * 121$	$19 * 121$	36/1.1	19/0.5
syn30h	$528 * 249$	$161 * 249$	54/1.1	26/0.6
syn30m02h	$1326 * 617$	$381 * 617$	46/1.4	28/0.7
syn05m02m	$177 * 67$	$11 * 67$	38/0.3	20/0.5
syn10h	$171 * 84$	$55 * 84$	37/0.2	19/0.4
syn15m	$174 * 67$	$12 * 67$	33/0.3	18/0.6
syn40h	$715 * 331$	$213 * 331$	49/0.9	24/0.5
synthes1	$30 * 13$	$1 * 13$	13/0.1	15/0.4
synthes2	$41 * 16$	$2 * 16$	26/0.1	17/0.4
synthes3	$71 * 26$	$3 * 26$	21/0.2	17/0.4
ex1223a	$59 * 17$	$1 * 17$	12/0.1	21/0.5
gp_dave_1	$988 * 705$	$453 * 705$	39/2.8	22/3.4
fiac81a	$163 * 191$	$126 * 191$	17/0.3	14/0.5
fiac81b	$65 * 87$	$57 * 87$	16/0.2	13/0.4
batch	$175 * 58$	$13 * 58$	161/0.9	58/ fail
demb761	$93 * 131$	$90 * 131$	25/0.4	16/0.4
demb762	$93 * 131$	$90 * 131$	27/0.4	15/0.4
demb781	$14 * 19$	$12 * 19$	9/0.1	11/0.4
enpro48	$506 * 167$	$33 * 167$	223/1.6	96/ fail
isil01 (infeasible)	$468 * 393$	$261 * 393$	58/1.9	44/ fail
jha88.mat	$866 * 1131$	$825 * 1131$	33/3.7	17/0.8
LogExpCR-n20-m400	$2023 * 2022$	$1200 * 2022$	41/7.9	30/3.5
LogExpCR-n100-m400	$2103 * 2102$	$1200 * 2102$	50/9.1	32/3.5
LogExpCR-n20-m1200	$6023 * 6022$	$3600 * 6022$	58/80.0	38/45.1
LogExpCR-n500-m400	$2503 * 2502$	$1200 * 2502$	75/20.5	59/9.5
LogExpCR-n500-m1600	$8503 * 8502$	$4800 * 8502$	82/228.1	4/ fail

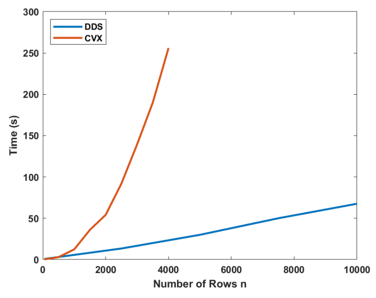
where $qe = \text{Tr}X\ln(X)$ and A_i's are sparse symmetric matrices. The problems for qre are also in the same format. Our collection contains both feasible and infeasible instances. The results are shown in Tables 7 and 8 respectively for quantum entropy and quantum relative entropy. The MATLAB files 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 [17], which approximates matrix logarithm with functions that can be described by SDPs. Note that these approximations increase the dimension of the problem, which is why CVX fails for the larger problems in Table 7. One application of minimizing qre function is calculating the rate of quantum key distribution



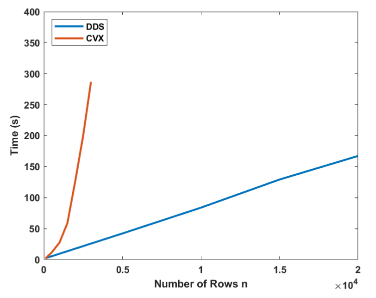
(a) $m = 10$



(b) $m = 20$



(c) $m = 50$



(d) $m = 100$

Fig. 2: Average running time for minimizing nuclear norm versus the number of rows in the matrix, where the number of columns m is fixed at four different values. The plots are created by linear interpolation of running times for five or eight values of n. Each point is the average time for five randomly created instances.

(QKD) channels. QKD is a secure communication method between two parties involving components of quantum mechanics [46]. The security of the QKD channel depends on the exact calculation of its key rate. There are different protocols for QKD and for many of them, the main non-trivial component of calculating the key rate is solving an optimization problem of the form:

$$\begin{aligned} \min\ & qre(\mathcal{G}(\rho), \mathcal{Z}(\mathcal{G}(\rho))) \\ \text{s.t.}\ & A(\rho) = b, \\ & \rho \succeq 0, \end{aligned} \tag{99}$$

where A is a linear map on Hermitian matrices and $\mathcal{G}$ and $\mathcal{Z}$ are Kraus operators. OpenQKDSecurity is a platform for numerical key rate calculation of QKD [45], where several examples for different regimes can be created. One protocol for QKD is Entanglement-Based BB84. The parameters of the problem $\mathcal{G}$, $\mathcal{Z}$, and Γ are specified by two parameters: the probability of performing measurement in one of the two possible basis p_z, and the error rate e. As we use natural logarithm in DDS, the key rate R for this protocol is calculated

Table 7: Results for problems involving Quantum Entropy using DDS and CVXQUAD (with SPDT3 as the solver). Times are in seconds.

Problem	size of A	Iter/time DDS	Iter/time CVXQUAD
QuanEntr-10	$101 * 21$	11/ 0.2	12/ 0.4
QuanEntr-20	$401 * 41$	12/ 0.2	13/ 0.7
QuanEntr-50	$2501 * 101$	19/ 0.6	12/ 6.1
QuanEntr-100	$10001 * 201$	23/ 3.1	13, 80
QuanEntr-200	$40001 * 401$	34/ 25	size error
QuanEntr-LP-10	$111 * 21$	20/ 0.2	16/ 0.5
QuanEntr-LP-20	$421 * 41$	22 / 0.3	20/ 1.7
QuanEntr-LP-50	$2551 * 101$	34/ 1.3	20/ 29.1
QuanEntr-LP-100	$10101 * 201$	45/ 7.2	21/ 479.7
QuanEntr-LP-SOCP-10-infea	$122 * 21$	12/ 0.6	16/ 1.3
QuanEntr-LP-SOCP-10	$122 * 21$	27/ 0.6	24/ 0.8
QuanEntr-LP-SOCP-20-infea	$441 * 41$	14/ 0.6	15/ 1.9
QuanEntr-LP-SOCP-20	$441 * 41$	34/ 0.6	22/ 1.8
QuanEntr-LP-SOCP-100	$10201 * 201$	51/ 14.7	28/ 2218.7
QuanEntr-LP-SOCP-200	$40402 * 401$	69/ 49	size error
QuanEntr-LP-3norm-10-infea	$121 * 21$	16/ 0.9	22/ 1.6
QuanEntr-LP-3norm-10	$121 * 21$	23/ 0.6	19/ 0.8
QuanEntr-LP-3norm-20-infea	$441 * 41$	16/ 0.7	18/ 2.0
QuanEntr-LP-3norm-20	$441 * 41$	28/ 0.9	20/ 3.1
QuanEntr-LP-3norm-100-infea	$10201 * 201$	25/ 9.6	17/ 618.3
QuanEntr-LP-3norm-100	$10201 * 201$	68/ 21.8	23/ 329.0

Table 8: Results for problems involving Quantum Relative Entropy using DDS and CVXQUAD (with SPDT3 as the solver). Times are in seconds.

Problem	size of A	Type of Constraints	Iter/time DDS	Iter/time CVXQUAD
QuanReEntr-6	$73 * 13$	QRE	9/ 1.0	20/ 1.7
QuanReEntr-10	$201 * 21$	QRE	12/ 11.2	25/ 95
QuanReEntr-20	$801 * 41$	QRE	15/ 34.4	size error
QuanReEntr-LP-6	$79 * 13$	QRE-LP	29/ 1.7	24/ 4.8
QuanReEntr-LP-6-infea	$79 * 13$	QRE-LP	30/ 1.7	28/ 3.6
QuanReEntr-LP-10	$101 * 21$	QRE-LP	27/ 4.6	38/ 678.9

by the formula

$$R = \frac{p}{\ln(2)} - \delta_{EC},$$

where p is the optimal value of (99) and δ_{EC} is a constant caused by performing error-correction. To input the problem into DDS, after removing the equality constraints, the matrix A is rank deficient. Therefore, we have to perform some column operations to make A full column rank. Table 9 contains the results of some examples for different (p_z, e) values.

Table 9: Results for problems involving Quantum Relative Entropy optimization problems.

Parameters (p_z, e)	Size of matrices	Type of Constraints	Optimal value p of (99)
$(0.5, .01)$	16	QRE	0.3186
$(0.5, .1)$	16	QRE	0.1840
$(0.9, .01)$	16	QRE	0.5225
$(0.9, .1)$	16	QRE	0.3018

14.6 Hyperbolic Polynomials

We have created a library of hyperbolic cone programming problems for our experiments. Hyperbolic polynomials are defined in Section 11. We discuss three methods in this section for the creation of our library.

14.6.1 Hyperbolic polynomials from matriods

Let us first define a matroid.

Definition 1 A *ground set* E and a collection of *independent sets* $\mathcal{I} \subseteq 2^E$ form a *matroid* $M = (E, \mathcal{I})$ if all of the following hold:

– $\emptyset \in \mathcal{I}$,
– 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 $b \in B \backslash A$ such that $A \cup \{b\} \in \mathcal{I}$.

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:

$$r_M(A) := \max\{|B| : B \subseteq A, B \in \mathcal{I}\}. \tag{100}$$

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

$$p_M(x) := \sum_{B \in \mathcal{B}} \prod_{i \in B} x_i. \tag{101}$$

A class of hyperbolic polynomials are the basis generating polynomials of certain matroids with the *half-plane property* [9]. A polynomial 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 2 *Assume that M is a matroid with the 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 the matroid is 4. All subsets of size 4 are independent except 5 of them. Vamos matroid has the half-plane property [44]. The basis generating polynomial is:

$$\begin{aligned}&p_V(x) :=\\ &E_4(x_1,\dots,x_8) - x_1x_2x_3x_4 - x_1x_2x_5x_6 - x_1x_2x_6x_8 - x_3x_4x_5x_6 - x_5x_6x_7x_8,\end{aligned}$$

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

$$\begin{aligned}&p_{VL}(x) :=\\ &E_4(x_1,\dots,x_{2m}) - \left(\sum_{i=2}^{m} x_1x_2x_{2k-1}x_{2k}\right) - \left(\sum_{i=2}^{m-1} x_{2k-1}x_{2k}x_{2k+1}x_{2k+2}\right).\end{aligned}$$

These polynomials have an interesting property of being counter examples to one generalization of the Lax conjecture [4,6]. Explicitly, there is no power k and symmetric matrices $H_2,\dots,H_{2m}$ such that

$$(p_{VL}(x))^k = \det(x_1I + x_2H_2 + \cdots + x_{2m}H_{2m}). \tag{102}$$

In the DDS package, we have created a function
`poly = vamos(m)`
that returns a Vamos like polynomial 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

$$T_G(x) := \sum_{T\in\mathcal{T}} \prod_{e\in T} x_e \tag{103}$$

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 10: Results for problems involving hyperbolic polynomials. The times are in seconds.

Problem	size of A	var/deg of $p(z)$	Type of Constraints	Iter/time
Vamos-1	$8*4$	8/4	HB	7/ 0.7
Vamos-LP-1	$12*4$	8/4	HB-LP	8/ 0.3
Vamos-SOCP-1	$17*5$	8/4	HB-LP-SOCP	11/ 0.5
Vamos-Entr-1	$17*5$	8/4	HB-LP-Entropy	9/ 10.8
VL-Entr-1	$41*11$	20/4	HB-LP-Entropy	11/ 7.2
VL-Entr-2	$61*16$	30/4	HB-LP-Entropy	12/ 60.2

14.6.2 Derivatives of products of linear forms

Hyperbolicity of a polynomial is preserved under directional derivative operation (see [37]):

Theorem 5 *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*

$$p'_e(x) := (\nabla p)(x)[e] \tag{104}$$

is also hyperbolic in direction e. Moreover,

$$\Lambda(p, e) \subseteq \Lambda(p'_e, e). \tag{105}$$

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 5 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 [47]. For some preliminary computational experiments on such hyperbolic programming problems, see [29]. 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 an 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

$$M := \begin{bmatrix} 2 & 0 & -1 \\ 1 & -3 & 4 \end{bmatrix}. \tag{106}$$

Table 11 shows some results of problems where the hyperbolic polynomial is either a product of linear forms or their derivatives.

Table 11: Results for problems involving hyperbolic polynomials as product of linear forms and their derivatives. Times are in seconds.

Problem	size of A	var/deg of $p(z)$	Type of Constraints	Iter/time
HPLin-LP-1	$55 * 50$	5/3	HB-LP	13/ 1.0
HPLinDer-LP-1	$55 * 50$	5/3	HB-LP	12/ 9.6
HPLin-LP-2	$55 * 50$	5/7	HB-LP	19/ 1.8
HPLin-Entr-1	$111 * 51$	10/15	HB-LP-Entropy	19/ 12.8
HPLinDer-Entr-1	$111 * 51$	10/15	HB-LP-Entropy	18/ 36.2
HPLin-pn-1	$111 * 51$	10/15	HB-LP-pnorm	30/ 15.6
HPLinDer-pn-1	$111 * 51$	10/15	HB-LP-pnorm	32/ 61.6
Elem-unb-10 (unb)	$10 * 5$	10/3	HB	12/ 0.8
Elem-LP-10	$15 * 5$	10/3	HB-LP	9/ 0.4
Elem-LP-10-2	$1010 * 1000$	10/3	HB-LP	37 / 5.0
Elem-Entr-10-1	$211 * 101$	10/3	HB-LP-Entropy	17/ 1.0
Elem-Entr-10-2	$221 * 101$	10/4	HB-LP-Entropy	18 / 11.3

14.6.3 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 $e_1 H_1 + \cdots + e_m H_m$ is positive definite. Then, $p(x) := \det(x_1 H_1 + \cdots + x_m H_m)$ is hyperbolic in the direction of e. Now, by using Theorem 5, 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.

References

1. Amini, N., Brändén, P.: Non-representable hyperbolic matroids. Adv. Math. **334**, 417–449 (2018)
2. Boyd, S., Kim, S.J., Vandenberghe, L., Hassibi, A.: A tutorial on geometric programming. Optimization and Engineering **8**(1), 67–127 (2007)
3. Brändén, P.: Polynomials with the half-plane property and matroid theory. Advances in Mathematics **216**(1), 302–320 (2007)
4. Brändén, P.: Obstructions to determinantal representability. Advances in Mathematics **226**(2), 1202–1212 (2011)
5. Brändén, P.: Hyperbolicity cones of elementary symmetric polynomials are spectrahedral. Optimization Letters **8**(5), 1773–1782 (2014)
6. Burton, S., Vinzant, C., Youm, Y.: A real stable extension of the Vamos matroid polynomial. arXiv preprint arXiv:1411.2038 (2014)
7. Chandrasekaran, V., Shah, P.: Relative entropy optimization and its applications. Mathematical Programming **161**(1-2), 1–32 (2017)
8. Chares, R.: Cones and interior-point algorithms for structured convex optimization involving powers and exponentials. Ph.D. thesis, Université Catholique de Louvain, Louvain-la-Neuve (2008)
9. Choe, Y.B., Oxley, J.G., Sokal, A.D., Wagner, D.G.: Homogeneous multivariate polynomials with the half-plane property. Advances in Applied Mathematics **32**(1-2), 88–187 (2004)
10. Coey, C., Kapelevich, L., Vielma, J.P.: Performance enhancements for a generic conic interior point algorithm. Mathematical Programming Computation pp. 1–49 (2022)

11. Coey, C., Kapelevich, L., Vielma, J.P.: Solving natural conic formulations with hypatia.jl. INFORMS Journal on Computing **34**(5), 2686–2699 (2022)
12. Dahl, J., Andersen, E.D.: A primal-dual interior-point algorithm for nonsymmetric exponential-cone optimization. Mathematical Programming **194**(1-2), 341–370 (2022)
13. Davis, C.: All convex invariant functions of Hermitian matrices. Archiv der Mathematik **8**(4), 276–278 (1957)
14. Dunning, I., Huchette, J., Lubin, M.: Jump: A modeling language for mathematical optimization. SIAM Review **59**(2), 295–320 (2017). DOI 10.1137/15M1020575
15. Fang, S.C., Rajasekera, J.R., Tsao, H.S.J.: Entropy Optimization and Mathematical Programming, vol. 8. Springer Science & Business Media (1997)
16. Fawzi, H., Saunderson, J.: Optimal self-concordant barriers for quantum relative entropies. arXiv preprint arXiv:2205.04581 (2022)
17. Fawzi, H., Saunderson, J., Parrilo, P.A.: Semidefinite approximations of the matrix logarithm. Foundations of Computational Mathematics **19**(2), 259–296 (2019)
18. Faybusovich, L., Tsuchiya, T.: Matrix monotonicity and self-concordance: how to handle quantum entropy in optimization problems. Optimization Letters pp. 1513–1526 (2017)
19. Faybusovich, L., Zhou, C.: Long-step path-following algorithm in quantum information theory: Some numerical aspects and applications. arXiv preprint arXiv:1906.00037 (2020)
20. Friberg, H.A.: CBLIB 2014: a benchmark library for conic mixed-integer and continuous optimization. Mathematical Programming Computation **8**(2), 191–214 (2016)
21. Grant, M., Boyd, S.: CVX: Matlab software for disciplined convex programming, version 2.2. `http://cvxr.com/cvx` (2020)
22. Güler, O.: Hyperbolic polynomials and interior point methods for convex programming. Mathematics of Operations Research **22**(2), 350–377 (1997)
23. Hiai, F., Petz, D.: Introduction to matrix analysis and applications. Springer Science & Business Media (2014)
24. Karimi, M., Tunçel, L.: mehdi-karimi-math/DDS: DDS 2.1 (2023). DOI 10.5281/zenodo.8339473. URL `https://doi.org/10.5281/zenodo.8339473`
25. Karimi, M., Tunçel, L.: Primal–dual interior-point methods for domain-driven formulations. Mathematics of Operations Research **45**(2), 591–621 (2020)
26. Karimi, M., Tunçel, L.: Status determination by interior-point methods for convex optimization problems in Domain-Driven form. Mathematical Programming **194**(1-2), 937–974 (2022)
27. Lewis, A.S.: The mathematics of eigenvalue optimization. Mathematical Programming **97**(1-2), 155–176 (2003)
28. MOSEK ApS: The MOSEK optimization toolbox for MATLAB manual. Version 9.0. (2019). `http://docs.mosek.com/9.0/toolbox/index.html`
29. Myklebust, T.G.J.: On primal-dual interior-point algorithms for convex optimisation. Ph.D. thesis, Department of Combinatorics and Optimization, Faculty of Mathematics, University of Waterloo
30. Nemirovski, A., Tunçel, L.: Cone-free primal-dual path-following and potential reduction polynomial time interior-point methods. Mathematical Programming **102**, 261–294 (2005)
31. Nesterov, Y.: Lectures on Convex Optimization. Springer (2018)
32. Nesterov, Y., Nemirovski, A.: Interior-Point Polynomial Algorithms in Convex Programming. SIAM Series in Applied Mathematics, SIAM: Philadelphia (1994)
33. Papp, D., Yildiz, S.: Sum-of-squares optimization without semidefinite programming. SIAM Journal on Optimization **29**(1), 822–851 (2019)
34. Papp, D., Yıldız, S.: Alfonso: Matlab package for nonsymmetric conic optimization. INFORMS Journal on Computing (2021)
35. Pataki, G., Schmieta, S.: The DIMACS library of semidefinite-quadratic-linear programs. Tech. rep., Preliminary draft, Computational Optimization Research Center, Columbia University, New York (2002)
36. Recht, B., Fazel, M., Parrilo, P.A.: Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. SIAM Review **52**(3), 471–501 (2010)
37. Renegar, J.: Hyperbolic programs, and their derivative relaxations. Foundations of Computational Mathematics **6**(1), 59–79 (2006)

38. Roy, S., Xiao, L.: On self-concordant barriers for generalized power cones. Optimization Letters **16**(2), 681–694 (2022)
39. Skajaa, A., Ye, Y.: A homogeneous interior-point algorithm for nonsymmetric convex conic optimization. Mathematical Programming **150**(2), 391–422 (2015)
40. Toh, K.C., Todd, M.J., Tütüncü, R.H.: SDPT3– a MATLAB software package for semidefinite programming, version 1.3. Optimization Methods and Software **11**(1-4), 545–581 (1999)
41. Tropp, J.A.: An introduction to matrix concentration inequalities. Foundations and Trends® in Machine Learning **8**(1-2), 1–230 (2015)
42. Tunçel, L.: Generalization of primal-dual interior-point methods to convex optimization problems in conic form. Foundations of Computational Mathematics **1**(3), 229–254 (2001)
43. Tunçel, L., Nemirovski, A.: Self-concordant barriers for convex approximations of structured convex sets. Foundations of Computational Mathematics **10**(5), 485–525 (2010)
44. Wagner, D.G., Wei, Y.: A criterion for the half-plane property. Discrete Mathematics **309**(6), 1385–1390 (2009)
45. Wang, W., Lütkenhaus, N.: OpenQKDSecurity platform. https://github.com/nlutkenhaus/openQKDsecurity (2021)
46. Winick, A., Lütkenhaus, N., Coles, P.J.: Reliable numerical key rates for quantum key distribution. Quantum **2**, 77 (2018)
47. Zinchenko, Y.: On hyperbolicity cones associated with elementary symmetric polynomials. Optim. Lett. **2**(3), 389–402 (2008)

Acknowledgements The authors wish to thank the associate editor and the anonymous reviewers, whose insightful comments and careful reading helped improve the presentation. The first author, Mehdi Karimi, extends heartfelt gratitude to his wife, Mehrnoosh, for her loving support throughout the five-year journey of developing this code.

Funding

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.

Conflict of interest

The authors declare that they have no conflict of interest.

Code availability

The full code was made available for review. Reference [24] in this published article is the link to the publicly available code.

A How to add different function/set constraints

A.1 Linear programming (LP) and second-order cone programming (SOCP)

Suppose we want to add ℓ LP constraints of the form

$$A_L^i x + b_L^i \geq 0, \quad i \in \{1, \dots, \ell\}, \tag{107}$$

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

$$\texttt{A\{k,1\}} = \begin{bmatrix} A_L^1 \\ \vdots \\ A_L^\ell \end{bmatrix}, \quad \texttt{b\{k,1\}} = \begin{bmatrix} b_L^1 \\ \vdots \\ b_L^\ell \end{bmatrix}$$
$$\texttt{cons\{k,1\}='LP'}, \quad \texttt{cons\{k,2\}} = [m_L^1, \ldots, m_L^\ell]. \tag{108}$$

Similarly to add ℓ SOCP constraints of the form

$$\|A_S^i x + b_S^i\| \le (g_S^i)^\top x + d_S^i, \quad i \in \{1, \ldots, \ell\}, \tag{109}$$

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

$$\texttt{A\{k,1\}} = \begin{bmatrix} (g_S^1)^\top \\ A_S^1 \\ \vdots \\ (g_S^\ell)^\top \\ A_S^\ell \end{bmatrix}, \quad \texttt{b\{k,1\}} = \begin{bmatrix} d_S^1 \\ b_S^1 \\ \vdots \\ d_S^\ell \\ b_S^\ell \end{bmatrix}$$
$$\texttt{cons\{k,1\}='SOCP'}, \quad \texttt{cons\{k,2\}} = [m_S^1, \ldots, m_S^\ell]. \tag{110}$$

Let us see an example:

Example 2 Suppose we are given the problem:

$$\begin{aligned} &\min\ c^\top x \\ &\text{s.t. } [-2, 1]x \le 1, \\ &\left\| \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix} x + \begin{bmatrix} 3 \\ 4 \end{bmatrix} \right\|_2 \le 2. \end{aligned} \tag{111}$$

Then we define

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

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

$$\{(z, t, s) \in \mathbb{R}^n \oplus \mathbb{R} \oplus \mathbb{R} : \|z\|^2 \le ts, \ t \ge 0, \ s \ge 0\}. \tag{112}$$

The abbreviation we use is `'SOCPR'`. To add ℓ rotated SOCP constraints of the form

$$\begin{aligned} &\|A_S^i x + b_S^i\|_2^2 \le ((g_S^i)^\top x + d_S^i)((\bar{g}_S^i)^\top x + \bar{d}_S^i), \quad i \in \{1, \ldots, \ell\}, \\ &(g_S^i)^\top x + d_S^i \ge 0, \ \ (\bar{g}_S^i)^\top x + \bar{d}_S^i \ge 0, \end{aligned} \tag{113}$$

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

$$\texttt{A\{k,1\}} = \begin{bmatrix} (g_S^1)^\top \\ (\bar{g}_S^1)^\top \\ A_S^1 \\ \vdots \\ (g_S^\ell)^\top \\ (\bar{g}_S^\ell)^\top \\ A_S^\ell \end{bmatrix}, \quad \texttt{b\{k,1\}} = \begin{bmatrix} d_S^1 \\ \bar{d}_S^1 \\ b_S^1 \\ \vdots \\ d_S^\ell \\ \bar{d}_S^\ell \\ b_S^\ell \end{bmatrix}$$
$$\texttt{cons\{k,1\}='SOCPR'}, \quad \texttt{cons\{k,2\}} = [m_S^1, \ldots, m_S^\ell]. \tag{114}$$

A.2 Semidefinite programming (SDP)

Consider ℓ SDP constraints in standard inequality (linear matrix inequality (LMI)) form:

$$F_0^i + x_1 F_1^i + \cdots + x_n F_n^i \succeq 0, \quad i \in \{1, \ldots, \ell\}. \tag{115}$$

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

$$\texttt{vec2sm(sm2vec(X))=X}. \tag{116}$$

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

$$\langle X, Y \rangle = \texttt{sm2vec(X)}^\top \texttt{sm2vec(Y)}. \tag{117}$$

To give (115) to DDS as the kth input block, we define:

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

$$\texttt{cons\{k,1\}='SDP'} \quad \texttt{cons\{k,2\}} = [n^1, \ \ldots \ , n^\ell]. \tag{118}$$

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 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 $A_0 + x_1 A_1 + x_2 A_2 + x_3 A_3$ 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

$$\begin{aligned} &\min t \\ &s.t. \ -1 + x_1 + x_2 + x_3 \geq 0, \\ &\quad tI - (A_0 + x_1 A_1 + x_2 A_2 + x_3 A_3) \succeq 0. \end{aligned} \tag{119}$$

To solve this problem, we define:

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

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

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

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

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.

A.3 Quadratic constraints

Suppose we want to add the following constraints to DDS:

$$x^\top A_i^\top Q_i A_i x + b_i^\top x + d_i \le 0, \quad i \in \{1,\dots,\ell\}, \tag{120}$$

where each A_i is m_i-by-n with rank n, and $Q_i \in \mathbb{S}^{m_i}$. To give constraints in (120) as input to DDS as the kth block, we define

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

$$\texttt{cons\{k,1\}='QC'} \;\; \texttt{cons\{k,2\}} = [m_1,\dots,m_\ell],$$
$$\texttt{cons\{k,3,i\}} = Q_i, \quad i \in \{1,\dots,\ell\}. \tag{121}$$

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

A.4 Generalized Power Cone

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

$$(A_s^i x + b_s^i, A_u^i x + b_u^i) \in K_{\alpha^i}^{(m_i,n_i)}, \quad i \in \{1,\dots,\ell\}, \tag{122}$$

where A_s^i, b_s^i, A_u^i, and b_u^i are matrices and vectors of proper size, and $K_\alpha^{(m,n)}$ is defined in (27). 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:

$$\texttt{cons\{k,2\}\{i,1\}} = [m_i \;\; n_i],$$
$$\texttt{cons\{k,2\}\{i,2\}} = \alpha^i, \quad i \in \{1,\dots,\ell\}. \tag{123}$$

For matrices A and b, we define:

$$\texttt{A\{k,1\}} = \begin{bmatrix} A_s^1 \\ A_u^1 \\ \vdots \\ A_s^\ell \\ A_u^\ell \end{bmatrix}, \quad \texttt{b\{k,1\}} = \begin{bmatrix} b_s^1 \\ b_u^1 \\ \vdots \\ b_s^\ell \\ b_u^\ell \end{bmatrix}. \tag{124}$$

Example 4 Consider the following optimization problem with 'LP' and 'GPC' constraints:

$$\begin{aligned} &\min\; -x_1 - x_2 - x_3 \\ &s.t.\;\; \|x\| \le (x_1+3)^{0.3}(x_2+1)^{0.3}(x_3+2)^{0.4}, \\ &\quad\;\; x_1, x_2, x_3 \ge 3. \end{aligned} \tag{125}$$

Then we define:

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

A.5 Epigraphs of matrix norms

Assume that we have constraints of the form

$$\begin{aligned} &X - UU^\top \succeq 0, \\ &X = A_0 + \sum_{i=1}^{\ell} x_i A_i, \\ &U = B_0 + \sum_{i=1}^{\ell} x_i B_i, \end{aligned} \tag{126}$$

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. 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 an 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 (126), the corresponding parts in A and b are defined as

$$\texttt{A\{k,1\}} = \begin{bmatrix} \texttt{m2vec}(B_1,n) & \cdots & \texttt{m2vec}(B_\ell,n) \\ \texttt{sm2vec}(A_1) & \cdots & \texttt{sm2vec}(A_\ell) \end{bmatrix}, \quad \texttt{b\{k,1\}} = \begin{bmatrix} \texttt{m2vec}(B_0,n) \\ \texttt{sm2vec}(A_0) \end{bmatrix}. \tag{127}$$

Example 5 Assume that we have matrices

$$U_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}, \quad U_1 = \begin{bmatrix} -1 & -1 & 1 \\ 0 & 0 & 1 \end{bmatrix}, \quad U_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \tag{128}$$

and our goal is to solve

$$\begin{aligned} &\min\ t \\ &s.t.\ \ UU^\top \preceq tI, \\ &\qquad U = U_0 + x_1 U_1 + x_2 U_2. \end{aligned} \tag{129}$$

Then the input to DDS is defined as

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

A.6 Minimizing nuclear norm

Consider the optimization problem

$$\begin{aligned} &\min\ \|X\|_* \\ &s.t.\ \ \mathrm{Tr}(U_i X) = c_i, \quad i \in \{1,\ldots,\ell\}, \end{aligned} \tag{130}$$

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

$$\begin{aligned} &\texttt{A\{1,1\}} = \begin{bmatrix} \texttt{m2vec}(U_1,n) & \cdots & \texttt{m2vec}(U_\ell,n) \\ \texttt{zeros}(m^2,1) & \cdots & \texttt{zeros}(m^2,1) \end{bmatrix}, \quad \texttt{b\{1,1\}} = \begin{bmatrix} \texttt{zeros}(mn,1) \\ \texttt{sm2vec}(I_{m\times m}) \end{bmatrix}, \\ &\texttt{cons\{1,1\}='MN'}, \quad \texttt{cons\{1,2\}} = [m \;\; n]. \end{aligned} \tag{131}$$

Then, if we run `[x,y]=DDS(c,A,b,cons)` and define `V:=(vec2m(y{1}(1:m*n),m))`$^\top$, then V is an optimal solution for (130). In subsection 14.4, we present numerical results for solving problem (130) 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 We consider minimizing the nuclear norm over a subspace. Consider the following optimization problem:

$$\begin{aligned} &\min \ \|X\|_* \\ &s.t. \ \ \mathrm{Tr}(U_1X) = 1 \\ &\qquad \mathrm{Tr}(U_2X) = 2, \end{aligned} \tag{132}$$

where

$$U_1 = \begin{bmatrix} 1 \ 0 \ 0 \ 0 \\ 0 \ 1 \ 0 \ 0 \end{bmatrix}, \quad U_2 = \begin{bmatrix} 0 \ 0 \ 1 \ 0 \\ 0 \ 0 \ 0 \ 1 \end{bmatrix}. \tag{133}$$

By using (37), the dual of this problem is

$$\begin{aligned} &\min \ -u_1 - 2u_2 \\ &s.t. \ \ \|u_1U_1 + u_2U_2\| \le 1. \end{aligned} \tag{134}$$

To solve this problem with our code, we define

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

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 (132) with objective value 2.2360. We have

$$X^* := V = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}. \tag{135}$$

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

In this subsection, we show how to add constraints of the form (38). Let us assume that we want to add the following s constraints to our model

$$\sum_{type} \sum_{i=1}^{\ell^j_{type}} \alpha_i^{j,type} f_{type}((a_i^{j,type})^\top x + \beta_i^{j,type}) + g_j^\top x + \gamma_j \le 0, \qquad j \in \{1, \dots, s\}. \tag{136}$$

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 (136), 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 (136). 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

$$\texttt{A} = \begin{bmatrix} g_j^\top \\ a_1^{j,1} \\ \vdots \\ a_{l_1^j}^{j,1} \\ \vdots \\ a_1^{j,5} \\ \vdots \\ a_{l_5^j}^{j,5} \end{bmatrix}, \quad \texttt{b} = \begin{bmatrix} \gamma_j \\ \beta_1^{j,1} \\ \vdots \\ \beta_{l_1^j}^{j,1} \\ \vdots \\ \beta_1^{j,5} \\ \vdots \\ \beta_{l_5^j}^{j5} \end{bmatrix}. \tag{137}$$

Let us see an example:

Example 7 Assume that we want to solve

$$\begin{aligned} &\min\ c^\top x \\ &\text{s.t.}\ \ -1.2\ln(x_2 + 2x_3 + 55) + 1.8e^{x_1+x_2+1} + x_1 - 2.1 \le 0, \\ &\qquad -3.5\ln(x_1 + 2x_2 + 3x_3 - 30) + 0.9e^{-x_3-3} - x_3 + 1.2 \le 0, \\ &\qquad x \ge 0. \end{aligned} \tag{138}$$

For this problem, we define:

$$\begin{aligned} &\texttt{cons\{1,1\}='LP'},\ \ \texttt{cons\{1,2\}} = [3], \\ &\texttt{cons\{2,1\}='TD'},\ \ \texttt{cons\{2,2\}} = \left\{ \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix}, [1.2\ \ 1.8]\ ;\ \ \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix}, [3.5\ \ 0.9] \right\}, \\ &\texttt{A\{1,1\}} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \quad \texttt{b\{1,1\}} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \\ &\texttt{A\{2,1\}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 1 & 1 & 0 \\ 0 & 0 & -1 \\ 1 & 2 & 3 \\ 0 & 0 & -1 \end{bmatrix}, \quad \texttt{b\{2,1\}} = \begin{bmatrix} -2.1 \\ 55 \\ 1 \\ 1.2 \\ -30 \\ -3 \end{bmatrix}. \end{aligned}$$

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.

A.8 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 8

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

For this problem, we define:

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

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

A.9 Vector Relative Entropy

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

$$f(A_u^i x + b_u^i, A_z^i x + b_z^i) + g_i^\top x + \gamma_i \le 0, \quad i \in \{1,\dots,s\}, \tag{139}$$

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:

$$\texttt{A} = \begin{bmatrix} g_i^\top \\ A_u^1 \\ A_z^1 \\ \vdots \\ g_s^\top \\ A_u^s \\ A_z^s \end{bmatrix}, \quad \texttt{b} = \begin{bmatrix} \gamma_1 \\ b_u^1 \\ b_z^1 \\ \vdots \\ \gamma_s \\ b_u^s \\ b_z^s \end{bmatrix}. \tag{140}$$

Example 9 Assume that we want to minimize a relative entropy function under a linear constraint:

$$\begin{aligned} &\min\ (0.8x_1+1.3)\ln\left(\tfrac{0.8x_1+1.3}{2.1x_1+1.3x_2+1.9}\right)+(1.1x_1-1.5x_2-3.8)\ln\left(\tfrac{1.1x_1-1.5x_2-3.8}{3.9x_2}\right) \\ &\textit{s.t.} \qquad\qquad x_1+x_2\le\beta. \end{aligned} \tag{141}$$

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

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

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.

A.10 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

$$qe_i(A_0^i + x_1 A_1^i + \cdots + x_n A_n^i) \le g_i^\top x + d_i, \quad i \in \{1, \ldots, \ell\}. \tag{142}$$

A_j^i's are n_i-by-n_i symmetric matrices. To input (142) to DDS as the kth block, we define:

$$\begin{aligned}
&\texttt{cons\{k,1\}='QE'},\quad \texttt{cons\{k,2\}} = [n_1, \ldots, n_\ell],\\
&\texttt{A\{k,1\}} := \begin{bmatrix} g_1^\top \\ \texttt{sm2vec}(A_1^1), \cdots, \texttt{sm2vec}(A_n^1) \\ \vdots \\ g_\ell^\top \\ \texttt{sm2vec}(A_1^\ell), \cdots, \texttt{sm2vec}(A_n^\ell) \end{bmatrix}, \quad \texttt{b\{k,1\}} := \begin{bmatrix} d_1 \\ \texttt{sm2vec}(A_0^1) \\ \vdots \\ d_\ell \\ \texttt{sm2vec}(A_0^\ell) \end{bmatrix}.
\end{aligned} \tag{143}$$

Example 10 Assume that we want to find scalars x_1, x_2, and x_3 such that $2x_1+3x_2-x_3 \le 5$ 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

$$\begin{aligned}
&\min \; t\\
&s.t. \;\; qe(x_1A_1 + x_2A_2 + x_3A_3) \le t,\\
&\qquad 2x_1 + 3x_2 - x_3 \le 5,\\
&\qquad x_1A_1 + x_2A_2 + x_3A_3 \succeq 3I.
\end{aligned} \tag{144}$$

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 $f(H) = 14.63$.

A.11 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

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

A_j^i's and B_j^i's are n_i-by-n_i symmetric matrices. To input (142) to DDS as the kth block, we define:

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

$$\texttt{A\{k,1\}} := \begin{bmatrix} g_1^\top \\ \texttt{sm2vec}(A_1^1), \cdots, \texttt{sm2vec}(A_n^1) \\ \texttt{sm2vec}(B_1^1), \cdots, \texttt{sm2vec}(B_n^1) \\ \vdots \\ g_\ell^\top \\ \texttt{sm2vec}(A_1^\ell), \cdots, \texttt{sm2vec}(A_n^\ell) \\ \texttt{sm2vec}(B_1^\ell), \cdots, \texttt{sm2vec}(B_n^\ell) \end{bmatrix}, \quad \texttt{b\{k,1\}} := \begin{bmatrix} d_1 \\ \texttt{sm2vec}(A_0^1) \\ \texttt{sm2vec}(B_0^1) \\ \vdots \\ d_\ell \\ \texttt{sm2vec}(A_0^\ell) \\ \texttt{sm2vec}(B_0^\ell) \end{bmatrix}. \tag{145}$$

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

A.12 Adding constraints involving hyperbolic polynomials

Consider a hyperbolic polynomial constraint of the form

$$p(Ax + b) \ge 0. \tag{146}$$

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 11.1.
`cons{k,4}` is the format of polynomial: `'monomial'`, `'straight_line'`, or `'determinant'`.
`cons{k,5}` is the direction of hyperbolicity or a vector in the interior of the hyperbolicity cone.

Example 11 Assume that we want to give constraint (146) to DDS for $p(x) = x_1^2 - x_2^2 - x_3^2$, using the monomial format. Then, `cons` part is defined as

$$\texttt{cons\{k,1\}='HB'}, \;\; \texttt{cons\{k,2\}} = [3],$$

$$\texttt{cons\{k,3\}} = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 0 & 2 & 0 & -1 \\ 0 & 0 & 2 & -1 \end{bmatrix},$$

$$\texttt{cons\{k,4\}='monomial'}, \;\; \texttt{cons\{k,5\}} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}.$$

A.13 Equality constraints

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

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

Example 12 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:

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