sdpt3r: Semidefinite Quadratic Linear Programming in R

by Adam Rahman

Abstract We present the package **sdpt3r**, an R implementation of the Matlab package SDPT3 (Toh et al., 1999). The purpose of the software is to solve semidefinite quadratic linear programming (SQLP) problems, which encompasses problems such as D-optimal experimental design, the nearest correlation matrix problem, and distance weighted discrimination, as well as problems in graph theory such as finding the maximum cut or Lovasz number of a graph.

Current optimization packages in R include **Rdsdp**, **Rcsdp**, **scs**, **cccp**, and **Rmosek**. Of these, **scs** and **Rmosek** solve a similar suite of problems. In addition to these solvers, the R packages **CXVR** and **ROI** provide sophisticated modelling interfaces to these solvers. As a point of difference from the current solvers in R, **sdpt3r** allows for log-barrier terms in the objective function, which allows for problems such as the D-optimal design of experiments to be solved with minimal modifications. The **sdpt3r** package also provides helper functions, which formulate the required input for several well-known problems, an additional perk not present in the other R packages.

Introduction

Convex optimization is a well traversed field with far reaching applications. While perhaps unfamiliar to those in the statistical sciences, many problems important to statisticians can be formulated as a convex optimization, perhaps the most well known of which would be the least squares problem. More specifically, many problems in statistics can be formulated as a subset of these convex optimization problems, known as *conic linear optimization problems*.

One such example would be the nearest correlation matrix problem (Higham, 2002), which was first considered when attempting to find correlations between stocks, where incomplete data on daily stock returns are not unusual. Pairwise correlations are only computed when data is available for both pairs of stocks under consideration, resulting in a correlation matrix that contains pairwise correlations, but is not necessarily positive semidefinite - an *approximate* correlation matrix. The goal is to then find the correlation matrix that is nearest to the approximate correlation matrix in some way.

Other examples of problems that can be formulated in terms of a conic linear optimization problem include D-optimal experimental design (Smith, 1918), classification using distance weighted discrimination (Marron et al., 2007), minimum volume ellipsoids (John, 2014), and problems in educational testing (Chu and Wright, 1995).

Problems in related fields can also be solved, including finding the maximum cut (or maximum k-cut) of a graph, finding the upper bound of the Shannon entropy of a graph, also known as the Lovasz number (Vandenberghe et al., 1998), as well as problems in control theory, Toeplitz matrix approximation, and Chebyshev approximation.

For the purpose of solving these conic linear optimization problems, we introduce the R package **sdpt3r**, an implementation of the Matlab package SDPT3 by Toh et al. (1999). Of the R packages available to perform conic optimization, **sdpt3r** is among the most general. **Rdsdp** (Zhu and Ye, 2016) & **Rcsdp** (Bravo, 2016) are capable of solving semidefinite conic optimization problems, while **cccp** (Pfaff, 2015) solves linear and quadratic conic optimization problems. The **sdpt3r** package allows for all of linear, quadratic, and semidefinite conic optimization to be solved simultaneously (i.e., a problem with any combination of semidefinite, quadratic, or linear cones can be solved). Two comparable packages, **scs** (O'Donoghue et al., 2017) and **Rmosek** (Friberg, 2012), solve a similar suite of problems. Additionally, the R packages **CXVR** (Fu et al., 2017) and **ROI** (Theußl et al., 2017) provide sophisticated modelling interfaces to these solvers.

As a point of difference, **scs** and **Rmosek** allow for the exponential and power cones to be included in the constraints, while **sdpt3r** handles log-barrier terms in the objective function directly. The inclusion of log-barrier terms allows for the D-optimal design of experiments and minimum volume ellipsoid problems to be solved with minimal modifications. In addition, **sdpt3r** provides helper functions which directly solve a number of well known problems (such as the "max cut" or nearest correlation matrix problems) with minimal input burden on the user. This additional functionality is not found in either **scs** or **Rmosek** (although **scs** can be used with the **CVXR** package).

This paper is structured as follows. In Section 52.2 we discuss in greater detail the mathematical formulation of the linear conic optimization problem, and introduce three examples to explore the increasing generality of the problem to be solved. Section 52.3 discusses the R implementation of

sdpt3r, and the main function by which conic linear optimization problems are solved, `sqlp`, including the required input, and the output generated. The same examples used in Section 52.2 will be used to demonstrate how a standard conic linear optimization problem can be converted to a form solvable by `sqlp`. Section 52.4 presents the classic form of several other well known problems that can be solved using **sdpt3r**, as well as the helper functions available to convert them to the appropriate form. Finally Section 52.5 provides some closing remarks.

Conic linear optimization

At its simplest, a conic linear optimization problem has the following standard form (Tütüncü et al., 2003):

$$\begin{array}{lrcl} \underset{\mathbf{X}}{\text{minimize}} & \langle \mathbf{C}, \mathbf{X} \rangle & & \\ \text{subject to} & & & \\ & \langle \mathbf{A}_k, \mathbf{X} \rangle & = & \mathbf{b}_k, \quad k = 1, ..., m \\ & \mathbf{X} & \in & \mathcal{K} \end{array} \tag{1}$$

where $\mathcal{K}$ is a cone. Generally, $\mathcal{K}$ is either a

- Semidefinite Cone - $\mathcal{S}^n = \{\mathbf{X} \in \mathcal{R}^{n \times n} : \mathbf{X} \succeq 0, \mathbf{X}_{ij} = \mathbf{X}_{ji} \; \forall \; i \neq j\}$
- Quadratic Cone - $\mathcal{Q}^n = \{\mathbf{x} = [x_0; \tilde{\mathbf{x}}] \in \mathcal{R}^n : x_0 \geq \sqrt{\tilde{\mathbf{x}}^\mathsf{T} \tilde{\mathbf{x}}}\}$
- Linear Cone - $\mathcal{L}^n$ - non-negative orthant of $\mathcal{R}^n$

Here, $\tilde{\mathbf{x}} = [x_1, \ldots, x_{n-1}]$, and $\langle \cdot, \cdot \rangle$ represents the standard inner product in the appropriate space. In the semidefinite cone the inner product is $\langle \mathbf{X}, \mathbf{Y} \rangle = vec(\mathbf{X})^\mathsf{T} vec(\mathbf{Y})$, where the operator *vec* is the by-column vector version of the matrix $\mathbf{X}$, that is, for the $n \times n$ matrix $\mathbf{X} = [x_{ij}]$, $vec(\mathbf{X})$ is the $n^2 \times 1$ vector $[x_{11}, x_{12}, x_{13}, \ldots, x_{(n-1)n}, x_{nn}]^\mathsf{T}$. Note that *vec* does not require a square matrix in general.

One of the simplest problems that can be formulated in terms of a conic linear optimization problem is finding the maximum cut of a graph. Let $\mathbf{G} = [\mathbf{V}, \mathbf{E}]$ be a graph with vertices $\mathbf{V}$ and edges $\mathbf{E}$. A *cut* of the graph $\mathbf{G}$ is a partition of the vertices of $\mathbf{G}$ into two disjoint subsets $\mathbf{G}_1 = [\mathbf{V}_1, \mathbf{E}_1]$, $\mathbf{G}_2 = [\mathbf{V}_2, \mathbf{E}_2]$, with $\mathbf{V}_1 \cap \mathbf{V}_2 = \varnothing$. The size of the cut is defined to be the number of edges connecting the two subsets. The *maximum cut* is defined to be the cut of a graph $\mathbf{G}$ whose size is at least as large as any other cut. For a weighted graph object, we can also define the maximum cut to be the cut with weight at least as large as any other cut.

Finding the maximum cut is referred to as the **Max-Cut Problem**, and was one of the first problems found to be NP-complete, and is also one of the 21 algorithms on Karp's 21 NP-complete problems (Karp, 1972). The Max-Cut problem is also known to be *APX hard* (Papadimitriou and Yannakakis, 1991), meaning in addition to there being no polynomial time solution, there is also no polynomial time approximation.

Using the semidefinite programming approximation formulation of Goemans and Williamson (1995), the Max-Cut problem can be approximated to within an *approximation constant*. For a weighted adjacency matrix $\mathbf{B}$, the objective function can be stated as

$$\begin{array}{lrcl} \underset{\mathbf{X}}{\text{minimize}} & \langle \mathbf{C}, \mathbf{X} \rangle & & \\ \text{subject to} & & & \\ & diag(\mathbf{X}) & = & \mathbf{1} \\ & \mathbf{X} & \in & \mathcal{S}^n \end{array}$$

where $\mathcal{S}^n$ is the cone of symmetric positive semidefinite matrices of size n, and $\mathbf{C} = -(diag(\mathbf{B1}) - \mathbf{B})/4$. Here, we define $diag(\mathbf{a})$ for an $n \times 1$ vector $\mathbf{a}$ to be the diagonal matrix $\mathbf{A} = [A_{ij}]$ of size $n \times n$ with $A_{ii} = a_i, \;\; i = 1, \ldots, n$. For a matrix $\mathbf{X}$, $diag(\mathbf{X})$ extracts the diagonal elements from $\mathbf{X}$ and places them in a column-vector.

To see that the Max-Cut problem is a conic linear optimization problem it needs to be written in the same form as Equation 1. The objective function is already in a form identical to that of Equation 1, with minimization occurring over $\mathbf{X}$ of its inner product with a constant matrix $\mathbf{C} = -(diag(\mathbf{B1}) - \mathbf{B})/4$. There are n equality constraints of the form $x_{kk} = 1, \;\; k = 1, ..., n$, where x_{kk} is the k^{th} diagonal element of $\mathbf{X}$, and $b_k = 1$ in Equation 1. To represent this in the form $\langle \mathbf{A}_k, \mathbf{X} \rangle = x_{kk}$, take $\mathbf{A}_k$ to be

$$\mathbf{A}_k = [a_{ij}] = \begin{cases} 1, & i = j = k \\ 0, & \text{otherwise} \end{cases}$$

Now $\langle \mathbf{A}_k, \mathbf{X} \rangle = vec(\mathbf{A}_k)^{\mathsf{T}} vec(\mathbf{X}) = x_{kk}$ as required, and the Max-Cut problem is specified as a conic linear optimization problem.

Allowing for optimization to occur over only one variable at a time is quite restrictive, as only a small number of problems can be formulated in this form. Allowing optimization to occur over multiple variables simultaneously would allow for a broader range of problems to be solved.

A separable set of variables

The conic linear optimization problem actually covers a much wider class of problems than those expressible as in Equation 1. Variables can be separated into those which are constrained to a semidefinite cone, $\mathcal{S}$, a quadratic cone, $\mathcal{Q}$, or a linear cone, $\mathcal{L}$. The objective function is a sum of the corresponding inner products of each set of variables. The linear constraint is simply a sum of variables of linear functions of each set. This more general version of the conic linear optimization problem is

$$\begin{array}{ll} \underset{\mathbf{X}^s, \mathbf{X}^q, \mathbf{X}^l}{\text{minimize}} & \sum_{j=1}^{n_s} \langle \mathbf{C}_j^s, \mathbf{X}_j^s \rangle + \sum_{i=1}^{n_q} \langle \mathbf{C}_i^q, \mathbf{X}_i^q \rangle + \langle \mathbf{C}^l, \mathbf{X}^l \rangle \\ \text{subject to} & \\ & \begin{array}{rcl} \sum_{j=1}^{n_s} (\mathbf{A}_j^s)^{\mathsf{T}} svec(\mathbf{X}_j^s) + \sum_{i=1}^{n_q} (\mathbf{A}_i^q)^{\mathsf{T}} \mathbf{X}_i^q + (\mathbf{A}^l)^{\mathsf{T}} \mathbf{X}^l & = & \mathbf{b} \\ \mathbf{X}_j^s & \in & \mathcal{S}^{s_j} \ \forall j \\ \mathbf{X}_i^q & \in & \mathcal{Q}^{q_i} \ \forall i \end{array} \end{array} \tag{2}$$

Here, *svec* takes the upper triangular elements of a matrix (including the diagonal) in a column-wise fashion and vectorizes them. In general for an $n \times p$ matrix $\mathbf{X} = [x_{ij}]$, $svec(\mathbf{X})$ will have the following form $[x_{11}, x_{12}, x_{22}, x_{13}, ..., x_{(n-1)p}, x_{np}]^{\mathsf{T}}$. Recall that matrices in $\mathcal{S}$ are symmetric, so it is sufficient to constrain only the upper triangular elements of the matrix $\mathbf{X}^s$. For this formulation, $\mathbf{A}_j^s$, $\mathbf{A}_i^q$ and $\mathbf{A}^l$ are the constraint matrices of the appropriate size.

Some important problems in statistics can be formulated to fit this form of the optimization problem.

The nearest correlation matrix

First addressed by Higham (2002) in dealing with correlations between stock prices, difficulty arises when data is not available for all stocks on each day, which is unfortunately a common occurrence. To help address this situation, correlations are calculated for pairs of stocks only when data is available for both stocks on any given day. The resulting correlation matrix is only approximate in that it is not necessarily positive semidefinite.

This problem was cast by Higham (2002) as

$$\begin{array}{ll} \underset{\mathbf{X}}{\text{minimize}} & ||\mathbf{R} - \mathbf{X}||_F \\ \text{subject to} & \\ & \begin{array}{rcl} diag(\mathbf{X}) & = & \mathbf{1} \\ \mathbf{X} & \in & \mathcal{S}^n \end{array} \end{array}$$

where $\mathbf{R}$ is the approximate correlation matrix and $|| \cdot ||_F$ denotes the Frobenius norm. Unfortunately, the Frobenius norm in the objective function prevents the problem being formatted as a conic linear optimization problem.

Since the matrix $\mathbf{X}$ is constrained to have unit diagonal and be symmetric, and the matrix $\mathbf{R}$ is an approximate correlation matrix, meaning it will also have unit diagonal and be symmetric, we can re-write the objective function as

$$||\mathbf{R} - \mathbf{X}||_F = 2 * ||svec(\mathbf{R}) - svec(\mathbf{X})|| = 2 * ||\mathbf{e}||$$

Now, introduce a variable e_0 such that $e_0 \geq ||\mathbf{e}||$, and define $\mathbf{e}^* = [e_0; \ \mathbf{e}]$. The vector $\mathbf{e}^*$ is now restricted to be in the quadratic cone $\mathcal{Q}^{n(n+1)/2+1}$. This work leads to the formulation of Toh et al. (1999)

$$\begin{array}{lrcl} \underset{\mathbf{e}^*,\,\mathbf{X}}{\text{minimize}} & e_0 & & \\ \text{subject to} & & & \\ & svec(\mathbf{R}) - svec(\mathbf{X}) & = & [\mathbf{0}, \mathbf{I}_{n(n+1)/2}]\ \mathbf{e}^* \\ & diag(\mathbf{X}) & = & \mathbf{1} \\ & \mathbf{X} & \in & \mathcal{S}^n \\ & \mathbf{e}^* & \in & \mathcal{Q}^{n(n+1)/2+1} \end{array}$$

Here, $[\mathbf{X}, \mathbf{Y}]$ denotes column binding of the two matrices $\mathbf{X}_{n\times p}$ and $\mathbf{Y}_{n\times m}$ to form a matrix of size $n \times (p+m)$. By minimizing e_0, we indirectly minimize $\mathbf{e} = svec(\mathbf{R}) - svec(\mathbf{X})$, since recall we have $e_0 \geq ||\mathbf{e}||$, which is the goal of the original objective function.

To see this as a conic linear optimization problem, notice that e_0 can be written as $\langle \mathbf{C}^q,\ \mathbf{X}^q \rangle$ by letting $\mathbf{C}^q = [1; \mathbf{0}_{n(n+1)/2}]$ and $\mathbf{X}^q = \mathbf{e}^*$. Since the matrix $\mathbf{X}$ (i.e., $\mathbf{X}^s$) does not appear in the objective function, the matrix $\mathbf{C}^s$ is an $n \times n$ matrix of zeros.

Re-writing the first constraint as

$$svec(\mathbf{X}) + [\mathbf{0}, \mathbf{I}_{n(n+1)/2}]\ \mathbf{e}^* = \ svec(\mathbf{R})$$

we can easily define the constraint matrices and right hand side of the first constraint as

$$\begin{array}{ll} \mathbf{A}_1^s & = \mathbf{I}_{n(n+1)/2} \\ \mathbf{A}_1^q & = [\mathbf{0}, \mathbf{I}_{n(n+1)/2}] \\ \mathbf{b}_1 & = svec(\mathbf{R}) \end{array}$$

The second constraint is identical to the constraint from the Max-Cut problem, where each diagonal element of $\mathbf{X}$ is constrained to be equal to 1. Define $\mathbf{b}_2 = \mathbf{1}$, and for the k^{th} diagonal element of $\mathbf{X}$, define the matrix $\mathbf{A}_k$ as

$$\mathbf{A}_k = [a_{ij}] = \begin{cases} 1, & i = j = k \\ 0, & \text{otherwise} \end{cases}$$

yielding $\langle \mathbf{A}_k, \mathbf{X} \rangle = x_{kk}$. To write this as $(\mathbf{A}_2^s)^\mathsf{T}\mathbf{X}^s$, define

$$\mathbf{A}_2^s = [svec(\mathbf{A}_1), ..., svec(\mathbf{A}_n)]$$

Since $\mathbf{e}^*$ does not appear in the second constraint, $\mathbf{A}_2^q = \mathbf{0}_{n(n+1)/2+1}$.

The final step is to combine the individual constraint matrices from each constraint to form one constraint matrix for each variable, which is done by defining $\mathbf{A}^s = [\mathbf{A}_1^s,\ \mathbf{A}_2^s]$, $\mathbf{A}^q = [\mathbf{A}_1^q,\ \mathbf{A}_2^q]$. We also concatenate both right hand side vectors to form a single vector by defining $\mathbf{b} = [\mathbf{b}_1;\ \mathbf{b}_2]$. Here, the notation $[\mathbf{X}; \mathbf{Y}]$ is used to denote two matrices $\mathbf{X}_{p\times m}$ and $\mathbf{Y}_{q\times m}$ bound vertically to form a matrix of size $(p+q) \times m$. With this, the nearest correlation matrix problem is written as a conic linear optimization.

Semidefinite quadratic linear programming

While Equation 2 allows for additional variables to be present, it can be made more general still to allow even more problems to be solved. We will refer to this general form as a *semidefinite quadratic linear programming* (SQLP) problem.

The first generality afforded by an SQLP is the addition of an unconstrained variable $\mathbf{X}^u$, which, as the name suggests, is not bound to a cone, but instead, it is "constrained" to the reals in the appropriate dimension. The second generalization is to allow for what are known as *log-barrier* terms to exist in the objective function. In general, a barrier function in an optimization problem is a term that approaches infinity as the point approaches the boundary of the feasible region. As we will see, these log-barrier terms appear as log terms in the objective function.

Recall that for any linear optimization problem, there exists two formulations - the primal formulation and the dual formulation. For the purposes of a semidefinite quadratic linear programming problem, the primal problem will always be defined as a minimization, and the associated dual problem will therefore be a maximization

The primal problem

The primal formulation of the SQLP problem is

$$\begin{array}{lrcll}
\underset{\mathbf{X}_j^s,\mathbf{X}_i^q,\mathbf{X}^l,\mathbf{X}^u}{\text{minimize}} & \multicolumn{4}{l}{\sum_{j=1}^{n_s}[\langle \mathbf{C}_j^s,\,\mathbf{X}_j^s\rangle - \mathbf{v}_j^s\, \mathit{log\ det}\, \mathbf{X}_j^s] \;+\; \sum_{i=1}^{n_q}[\langle \mathbf{C}_i^q,\,\mathbf{X}_i^q\rangle - \mathbf{v}_i^q\, \mathit{log}\, \gamma(\mathbf{X}_i^q)]} \\
 & \multicolumn{4}{l}{\quad +\, \langle \mathbf{C}^l,\,\mathbf{X}^l\rangle \;-\; \sum_{k=1}^{n_l} \mathbf{v}_k^l\, \mathit{log}\, \mathbf{X}_k^l \;+\; \langle \mathbf{C}^u,\,\mathbf{X}^u\rangle} \\
\text{subject to} & & & & \\
 & \sum_{j=1}^{n_s}\mathbf{A}_j^s(\mathbf{X}_j^s) + \sum_{i=1}^{n_q}\mathbf{A}_i^q\mathbf{X}_i^q + \mathbf{A}^l\mathbf{X}^l + \mathbf{A}^u\mathbf{X}^u & = & \mathbf{b} & \\
 & \mathbf{X}_j^s & \in & \mathcal{S}^{s_j} & \forall\, j \\
 & \mathbf{X}_i^q & \in & \mathcal{Q}^{q_i} & \forall\, i \\
 & \mathbf{X}^l & \in & \mathcal{L}^{n_l} & \\
 & \mathbf{X}^u & \in & \mathcal{R}^{n_u} &
\end{array} \tag{3}$$

For each j, $\mathbf{C}_j^s$ and $\mathbf{X}_j^s$ are symmetric matrices of dimension s_j, restricted to the cone of positive semidefinite matrices of the same dimension. Similarly, for all i, $\mathbf{C}_i^q$ and $\mathbf{X}_i^q$ are real vectors of dimension q_i, restricted to the quadratic cone of dimension q_i. For a vector $\mathbf{u} = [u_0; \tilde{\mathbf{u}}]$ in a second order cone, define $\gamma(u) = \sqrt{u_0^2 - \tilde{\mathbf{u}}^\mathsf{T}\tilde{\mathbf{u}}}$. Finally, $\mathbf{C}^l$ and $\mathbf{X}^l$ are vectors of dimension n_l, restricted to linear cone of the same dimension, and $\mathbf{C}^u$ and $\mathbf{X}^u$ are unrestricted real vectors of dimension n_u.

As before, the matrices $\mathbf{A}_i^q$, $\mathbf{A}^l$, and $\mathbf{A}^u$ are constraint matrices in q_i, n_l, and n_u dimensions respectively, each corresponding to their respective quadratic, linear, or unrestricted block. $\mathbf{A}_j^s$ is defined to be a linear map from $\mathcal{S}^{s_j}$ to $\mathcal{R}^m$ defined by

$$\mathbf{A}_j^{s_j}(\mathbf{X}_j^s) = [\langle \mathbf{A}_{j,1}^s, \mathbf{X}_j^s\rangle; \ldots; \langle \mathbf{A}_{j,m}^s, \mathbf{X}_j^s\rangle]$$

where $\mathbf{A}_{j,1}^s \ldots \mathbf{A}_{j,m}^s \in \mathcal{S}^{s_j}$ are constraint matrices associated with the j^{th} semidefinite variable $\mathbf{X}_j^s$.

The dual problem

The dual problem associated with the semidefinite quadratic linear programming formulation is

$$\begin{array}{llllll}
\underset{\mathbf{Z}_j^s,\mathbf{Z}_i^q,\mathbf{Z}^l,\mathbf{y}}{\text{maximize}} & \mathbf{b}^\mathsf{T}\mathbf{y} & \multicolumn{4}{l}{+\; \sum_{j=1}^{n_s}[\mathbf{v}_j^s\, \mathit{log\ det}\, \mathbf{Z}_j^s \;+\; s_j\, \mathbf{v}_j^s\, (1 - \mathit{log}\, \mathbf{v}_j^s)]} \\
 & & \multicolumn{4}{l}{+\; \sum_{i=1}^{n_q}[\mathbf{v}_i^q\, \mathit{log}\, \gamma(\mathbf{Z}_i^q) \;+\; \mathbf{v}_i^q\, (1 - \mathit{log}\, \mathbf{v}_i^q)]} \\
 & & \multicolumn{4}{l}{+\; \sum_{k=1}^{n_l}[\mathbf{v}_k^l\, \mathit{log}\, \mathbf{Z}_k^l \;+\; \mathbf{v}_k^l\, (1 - \mathit{log}\, \mathbf{v}_k^l)]} \\
\text{subject to} & & & & & \\
 & (\mathbf{A}_j^s)^\mathsf{T}\mathbf{y} & +\; \mathbf{Z}_j^s & = \mathbf{C}_j^s, & \mathbf{Z}_j^s \in \mathcal{S}^{s_j}, & j = 1, \ldots, n_s \\
 & (\mathbf{A}_i^q)^\mathsf{T}\mathbf{y} & +\; \mathbf{Z}_i^q & = \mathbf{C}_i^q, & \mathbf{Z}_i^q \in \mathcal{Q}^{q_i}, & i = 1, \ldots, n_q \\
 & (\mathbf{A}^l)^\mathsf{T}\mathbf{y} & +\; \mathbf{Z}^l & = \mathbf{C}^l, & \mathbf{Z}^l \in \mathcal{L}^{n_l} & \\
 & (\mathbf{A}^u)^\mathsf{T}\mathbf{y} & & = \mathbf{C}^u, & \mathbf{y} \in \mathcal{R}^m &
\end{array} \tag{4}$$

where $(\mathbf{A}_j^s)^T$ is defined to be the adjoint operator of $\mathbf{A}_j^s$, where $(\mathbf{A}_j^s)^\mathsf{T}\mathbf{y} = \sum_{k=1}^m \mathbf{y}_k \mathbf{A}_{j,k}^s$. Equations 3 and 4 represent the most general form of the linear conic optimization problem that can be solved using **sdpt3r**.

Optimal design of experiments

Consider the problem of estimating a vector $\mathbf{x}$ from measurements $\mathbf{y}$ given by the relationship

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0,1).$$

The variance-covariance matrix of such an estimator is proportional to $(\mathbf{A}^\mathsf{T}\mathbf{A})^{-1}$. A reasonable goal during the design phase of an experiment would therefore be to minimize $(\mathbf{A}^\mathsf{T}\mathbf{A})^{-1}$ in some way.

There are many different ways in which $(\mathbf{A}^\mathsf{T}\mathbf{A})^{-1}$ might be made minimal. For example, minimization of the trace of $(\mathbf{A}^\mathsf{T}\mathbf{A})^{-1}$ (A-Optimality), minimization of the maximum eigenvalue of $(\mathbf{A}^\mathsf{T}\mathbf{A})^{-1}$ (E-Optimality), minimization of the determinant of $(\mathbf{A}^\mathsf{T}\mathbf{A})^{-1}$ (D-Optimality), and maximization of the trace of $\mathbf{A}^\mathsf{T}\mathbf{A}$ (T-Optimality) all have their merits.

Perhaps the most commonly used of these optimality criteria is D-Optimality, which is equivalent to maximizing the determinant of $\mathbf{A}^\mathsf{T}\mathbf{A}$. Typically, the rows of $\mathbf{A} = [\mathbf{a}_1, ..., \mathbf{a}_q]^T$ are chosen from M possible test vectors $\mathbf{u}_i \in \mathcal{R}^p,\ i = 1, ...M$, which are known in advance. That is,

$$\mathbf{a}_i \in \{\mathbf{u}_1, ..., \mathbf{u}_M\}, \quad i = 1, ..., q$$

Given that the matrix $\mathbf{A}$ is made up of these test vectors $\mathbf{u}_i$, Vandenberghe et al. (1998) write the matrix $\mathbf{A}^\mathsf{T}\mathbf{A}$ as

$$\mathbf{A}^\mathsf{T}\mathbf{A} = q\sum_{i=1}^{M} \lambda_i \mathbf{u}_i \mathbf{u}_i^\mathsf{T} \tag{5}$$

where λ_i is the fraction of rows in $\mathbf{A}$ that are equal to the vector $\mathbf{u}_i$. Then, Vandenberghe et al. (1998) write the D-optimal experimental design problem as a minimum determinant problem

$$\begin{array}{lrcl}
\underset{\lambda}{\text{minimize}} & \multicolumn{3}{l}{\log\det\,(\sum_{i=1}^{M}\lambda_i\mathbf{u}_i\mathbf{u}_i^\mathsf{T})^{-1}} \\
\text{subject to} & & & \\
& \lambda_i & \geq & 0, \quad i = 1, ..., m \\
& \sum_{i=1}^{M}\lambda_i & = & 1
\end{array}$$

Due to the inequality constraint, this primal formulation cannot be interpreted as an SQLP of the form of Equation 3. By defining $\mathbf{Z} = \mathbf{u}\ diag(\lambda)\ \mathbf{u}^\mathsf{T}$, the dual problem is (Toh et al., 1999)

$$\begin{array}{lrcl}
\underset{\mathbf{Z},\, \mathbf{z}^l,\, \lambda}{\text{maximize}} & \multicolumn{3}{l}{\log\det\,(\mathbf{Z})} \\
\text{subject to} & & & \\
& -\sum_{i=1}^{p}\lambda_i(\mathbf{u}_i\mathbf{u}_i^\mathsf{T}) + \mathbf{Z} & = & 0, \quad \mathbf{Z} \in \mathcal{S}^n \\
& -\lambda + \mathbf{z}^l & = & 0, \quad \mathbf{z}^l \in \mathcal{R}^p_+ \\
& \mathbf{1}^T\lambda & = & 1, \quad \lambda \in \mathcal{R}^p
\end{array}$$

Keeping in mind that this is a dual configuration, and thus follows Equation 4, we proceed with writing the D-Optimal design problem as an SQLP by first considering the objective function. The objective function depends only on the determinant of the matrix variable $\mathbf{Z}$, which is the log-barrier. This indicates that the variable v^s in Equation 4 is equal to 1 in this formulation, while v^q and v^l are both zero. Since λ does not appear in the objective function, the vector $\mathbf{b}$ is equal to $\mathbf{0}$.

The constraint matrices $\mathbf{A}$ are easy to define in the case of the dual formulation, as they multiply the vector $\mathbf{y}$ in Equation 4, so therefore multiply λ in our case. In the first constraint, each λ_i is multiplied by the matrix formed by $-\mathbf{u}_i\mathbf{u}_i^\mathsf{T}$, so define $\mathbf{A}_i$ to be

$$\mathbf{A}_i = -\mathbf{u}_i\mathbf{u}_i^\mathsf{T},\ \ i = 1, ..., p.$$

Then, the constraint matrix is $\mathbf{A}^s = [svec(\mathbf{A}_1), ..., svec(\mathbf{A}_p)]$. In the second constraint containing the linear variable $\mathbf{z}^l$, the constraint matrix is $\mathbf{A}^l = -\mathbf{I}_p$, and in the third constraint containing only the unconstrained variable λ, the constraint matrix is $\mathbf{A}^u = \mathbf{1}^\mathsf{T}$. Since there is no quadratic variable, $A^q = \mathbf{0}$.

Finally, define the right hand side of each constraint

$$\begin{array}{ll}
\mathbf{C}^s & = \mathbf{0}_{n\times n} \\
\mathbf{C}^l & = \mathbf{0}_{p\times 1} \\
\mathbf{C}^u & = 1
\end{array}$$

which fully specifies the D-Optimal design problem as an SQLP.

In the next section, we will demonstrate using R how these definitions can be translated for use in the main function of **sdpt3r** so an SQLP problem can be solved.

Solving a conic linear optimization problem with sdpt3r

Each of the problems presented in Section 52.2 can be solved using the **sdpt3r** package, an R implementation of the Matlab program SDPT3. The algorithm is an infeasible primal-dual predictor-corrector path-following method, utilizing either an HKM (Helmberg et al., 1996) or NT (Nesterov and Todd, 1997) search direction. The interested reader is directed to Tütüncü et al. (2003) for further details surrounding the implementation.

The main function available in **sdpt3r** is `sqlp`, which takes a number of inputs (or an `sqlp_input` object) specifying the problem to be solved, and executes the optimization, returning both the primal and dual solution to the problem. This function will be thoroughly discussed in Section 52.3.1, and examples will be provided. In addition to `sqlp`, a prospective user will also have access to a number of helper functions for well known problems that can be solved using **sdpt3r**. For example, the function `maxcut` takes as input an adjacency matrix $\mathbf{B}$, and produces an S3 object containing all the input variables necessary to solve the problem using `sqlp`. These functions will be discussed in Sections 52.3.3, 52.3.4, 52.3.4, and 52.4.

For **sdpt3r**, each optimization variable will be referred to as a *block* in the space in which it is restricted. For instance, if we have an optimization variable $\mathbf{X} \in \mathcal{S}^n$, we will refer to this as a semidefinite block of size n. It is important to note that it is possible to have multiple blocks from the same space, that is, it is possible to have both $\mathbf{X} \in \mathcal{S}^n$ as well as $\mathbf{Y} \in \mathcal{S}^m$ in the same problem.

Input variables

The main function call in **sdpt3r** is `sqlp`, which takes the following input variables

`blk`	A named-list object describing the block structure of the optimization variables.
`At`	A list object containing constraint matrices $\mathbf{A}^s$, $\mathbf{A}^q$, $\mathbf{A}^l$, and $\mathbf{A}^u$ for the primal-dual problem.
`b`	A vector containing the right hand side of the equality constraints, $\mathbf{b}$, in the primal problem, or equivalently the constant vector in the dual.
`C`	A list object containing the constant $\mathbf{C}$ matrices in the primal objective function or equivalently the corresponding right hand side of the equality constraints in the dual problem.
`X0`, `y0`, `Z0`	Matrix objects containing an initial iterate for the $\mathbf{X}$, $\mathbf{y}$, and $\mathbf{Z}$ variables for the SQLP problem. If not provided, an initial iterate is computed internally.
`control`	A list object providing additional parameters for use in `sqlp`. If not provided, default values are used.

The input variable `blk` describes the block structure of the problem. Letting `L` be the total number of semidefinite, quadratic, linear, and unrestricted blocks in the SQLP problem, define `blk` to be a named-vector object of length L, with names describing the type of block, and values denoting the size of the optimization variable, summarized in Table 1.

Block type	Name	Value
Semidefinite	s	s_j
Quadratic	q	q_i
Linear	l	n_l
Unrestricted	u	n_u

Table 1: *Structure of* `blk`.

The input variable `At` corresponds to the constraint matrices in Equation 3, and `C` the constant matrices in the objective function. The size of these input variables depends on the block they are representing, summarized in Table 2 for each block type.

	Block type			
	Semidefinite	Quadratic	Linear	Unrestricted
`At`	$\bar{s}_j \times m$	$q_j \times m$	$n_l \times m$	$n_u \times m$
`C`	$s_j \times s_j$	$q_j \times 1$	$n_l \times 1$	$n_u \times 1$

Table 2: *Size of* `At` *and* `C` *for each block type.*

Note that in Table 2, $\bar{s}_j = s_j(s_j+1)/2$. The size of `At` in the semidefinite block reflects the upper-triangular input format that has been discussed previously. In a semidefinite block, the optimization variable $\mathbf{X}$ is necessarily symmetric and positive semidefinite, it is therefore more efficient to consider only the upper-triangular portion of the corresponding constraint matrix.

It is important to note that both input variables `At` and `C` are lists containing constraint and constant matrices for each optimization variable. In general, the user need not supply initial iterates `X0`, `y0`, and

Z0 for a solution to be found using sqlp. The infeasible starting point generated internally by sqlp tends to be sufficient to find a solution. If the user wishes to provide a starting point however, the size parameters in Table 3 must be met for each block.

	Block type			
	Semidefinite	Quadratic	Linear	Unrestricted
X0	$s_j \times s_j$	$q_j \times 1$	$n_l \times 1$	$n_u \times 1$
y0	$s_j \times 1$	$q_j \times 1$	$n_l \times 1$	$n_u \times 1$
Z0	$s_j \times s_j$	$q_j \times 1$	$n_l \times 1$	$n_u \times 1$

Table 3: *Required size for initial iterates* X0*,* y0*, and* Z0*.*

The user may choose to depart from the default values of several parameters which could affect the optimization by specifying alternative values in the control list. A complete list of all parameters that can be altered can be found in Appendix 52.6.

An important example is the specification of the parbarrier parameter in control, which specifies the presence of a log-barrier in the objective function. The default case in control assumes that the parameters $\mathbf{v}_j^s$, $\mathbf{v}_i^q$, $\mathbf{v}_k^l$ in Equation 3 are all $\mathbf{0}$. If this, however, is not the case, then the user must specify an $L \times 1$ matrix object in control$parbarrier to store the values of these parameters (including zeros). If the j^{th} block is a semidefinite block containing p variables, $parbarrier_j = [v_{j1}^s, ..., v_{jn}^s]$. If the j^{th} block is a quadratic block containing p variables, $parbarrier_j = [v_{j1}^q, ..., v_{jn}^q]$. If the j^{th} block is a linear block $parbarrier_j = [v_1^l, ..., v_{n_l}^l]$. Finally, if the j^{th} block is the unrestricted block, then $parbarrier_j = [0, ..., 0]$, where 0 is repeated n_u times.

When executed, sqlp simultaneously solves both the primal and dual problems, meaning solutions for both problems are returned. The relevance of each output therefore depends on the problem being solved. The following object of class sqlp_output is returned upon completion

pobj	the value of the primary objective function
dobj	the value of the dual objective function
X	A list object containing the optimal matrix **X** for the primary problem
y	A vector object containing the optimal vector **y** for the dual problem
Z	A list object containing the optimal matrix **Z** for the dual problem

The examples in subsequent subsubsections will demonstrate the output provided by sqlp.

Toy Examples

Before moving on to more complex problems, consider first some very simple example to illustrate the functionality of the **sdpt3r** package. First, consider the following simple linear programming problem:

$$\begin{array}{ll} \text{Minimize} & x_1 + x_2 \\ \text{subject to} & \\ & \begin{aligned} x_1 + 4x_2 &= 12 \\ 3x_1 - x_2 &= 10 \end{aligned} \end{array}$$

This problem can be solved using **sdpt3r** in very straightforward fashion. First, this is a linear programming problem with two variables, x_1 and x_2. This implies that blk = c("l" = 2). Next the objective function can be written as $1 * x_1 + 1 * x_2$, so C = matrix(c(1,1),nrow=1). The constraints can be summarized in matrix form as:

$$A = \begin{bmatrix} 1 & 4 \\ 3 & -1 \end{bmatrix}$$

so A = matrix(c(1,3,4,-1),nrow=2)) and At = t(A). Finally the right hand side can be written in vector form as $[12, 10]$, so b = c(12,10). Pulling these all together, the problem is solved using sqlp:

```
blk = c("l" = 2)
C = matrix(c(1,1),nrow=1)
A = matrix(c(1,3,4,-1), nrow=2)
At = t(A)
b = c(12,10)
```

```
out = sqlp(blk,list(At),list(C),b)
out

$X
$X[[1]]
2 x 1 Matrix of class "dgeMatrix"
     [,1]
[1,]    4
[2,]    2


$y
          [,1]
[1,] 0.3076923
[2,] 0.2307692

$Z
$Z[[1]]
2 x 1 Matrix of class "dgeMatrix"
             [,1]
[1,] 6.494441e-10
[2,] 1.234448e-09


$pobj
[1] 6

$dobj
[1] 6
```

which returns the solution $x_1 = 4$ and $x_2 = 2$, and the optimal primal solution of 6. Second, consider the following simple quadratic programming problem:

$$\begin{array}{lrcl} \text{Minimize} & \frac{1}{2}x_1^2 - x_2^2 & & \\ \text{subject to} & & & \\ & 2x_1 - x_2 & = & 5 \\ & x_1 + x_2 & = & 4 \end{array}$$

This problem can be solved using **sdpt3r** by formulating the input variables in a similar fashion as the linear programming problem:

```
blk = c("q" = 2)
C = matrix(c(0.5,-1),nrow=1)
A = matrix(c(2,1,-1,1), nrow=2)
At = t(A)
b = c(5,4)

out = sqlp(blk,list(At),list(C),b)
out

$X
$X[[1]]
2 x 1 Matrix of class "dgeMatrix"
     [,1]
[1,]    3
[2,]    1


$y
     [,1]
[1,]  0.5
[2,] -0.5

$Z
$Z[[1]]
```

```
2 x 1 Matrix of class "dgeMatrix"
              [,1]
[1,]  2.186180e-09
[2,] -3.522956e-10


$pobj
[1] 0.5

$dobj
[1] 0.5
```

which returns the solution $x_1 = 3$ and $x_2 = 1$, with optimal primal solution of 0.5. Finally, consider the following simple semidefinite programming problem (taken from Freund (2004)):

$$\begin{array}{ll}\text{Minimize} & \begin{bmatrix} 1 & 2 & 3 \\ 2 & 9 & 0 \\ 3 & 0 & 7 \end{bmatrix} \cdot \begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix} \\ \text{subject to} & \\ & \begin{bmatrix} 1 & 0 & 1 \\ 0 & 3 & 7 \\ 1 & 7 & 5 \end{bmatrix} \cdot \begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix} = 11 \\ & \begin{bmatrix} 0 & 2 & 8 \\ 2 & 6 & 0 \\ 8 & 0 & 4 \end{bmatrix} \cdot \begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix} = 9 \end{array}$$

This problem is written almost exactly in the language used by **sdpt3**, and so can be easily solved by taking:

```
blk = c("s" = 3)
C = list(matrix(c(1,2,3,2,9,0,3,0,7), nrow=3))
A1 = matrix(c(1,0,1,0,3,7,1,7,5), nrow=3)
A2 = matrix(c(0,2,8,2,6,0,8,0,4), nrow=3)
At = svec(blk,list(A1,A2))
b = c(11,9)

out = sqlp(blk,At,C,b)
out

$X
$X[[1]]
3 x 3 Matrix of class "dgeMatrix"
           [,1]      [,2]      [,3]
[1,] 0.08928297 0.1606827 0.2453417
[2,] 0.16068265 0.2891815 0.4415426
[3,] 0.24534167 0.4415426 0.6741785


$y
          [,1]
[1,] 0.5172462
[2,] 0.4262486

$Z
$Z[[1]]
3 x 3 Matrix of class "dsyMatrix"
           [,1]      [,2]       [,3]
[1,]  0.4827538  1.147503 -0.9272352
[2,]  1.1475028  4.890770 -3.6207235
[3,] -0.9272352 -3.620723  2.7087744


$pobj
[1] 9.525946
```

```
$dobj
[1] 9.525946
```

which provides the optimal matrix solution X, and the optimal value of the objective function 9.53. Note that the function svec is used since the problem is a semidefinite programming problem, and thus each A matrix is necessarily symmetric.

The Max-Cut problem

Recall that the maximum cut of a graph **G** with adjacency matrix **B** can be found as the solution to

$$\begin{array}{lrcl} \text{Minimize} & \langle \mathbf{C}, \mathbf{X} \rangle & & \\ \text{subject to} & & & \\ & diag(\mathbf{X}) & = & \mathbf{1} \\ & \mathbf{X} & \in & \mathcal{S}^n \end{array}$$

where $\mathbf{C} = -(diag(\mathbf{B1}) - \mathbf{B})/4$. In Section 52.2, we wrote this in the form of an SQLP

$$\begin{array}{lrclcl} \text{Minimize} & \langle \mathbf{C}, \mathbf{X} \rangle & & & & \\ \text{subject to} & & & & & \\ & \langle \mathbf{A}_k, \mathbf{X} \rangle & = & 1, & k & = 1, \ldots, n \\ & \mathbf{X} & \in & \mathcal{S}^n & & \end{array}$$

where we defined $\mathbf{A}_k$ as

$$\mathbf{A}_k = [a_{ij}] = \begin{cases} 1, & i = j = k \\ 0, & \text{otherwise} \end{cases}$$

To convert this to a form usable by sqlp, we begin by noting that we have one optimization variable, **X**, and therefore $L = 1$. For an adjacency matrix B of dimension n for which we would like to determine the Max-Cut, **X** is constrained to the space of semidefinite matrices of size n. Therefore, for a 10×10 matrix B (as in Figure 1), blk is specified as

```
B <- rbind(c(0, 0, 0, 1, 0, 0, 1, 1, 0, 0),
           c(0, 0, 0, 1, 0, 0, 1, 0, 1, 1),
           c(0, 0, 0, 0, 0, 0, 0, 1, 0, 0),
           c(1, 1, 0, 0, 0, 0, 0, 1, 0, 1),
           c(0, 0, 0, 0, 0, 0, 1, 1, 1, 1),
           c(0, 0, 0, 0, 0, 0, 0, 0, 1, 0),
           c(1, 1, 0, 0, 1, 0, 0, 1, 1, 1),
           c(1, 0, 1, 1, 1, 0, 1, 0, 0, 0),
           c(0, 1, 0, 0, 1, 1, 1, 0, 0, 1),
           c(0, 1, 0, 1, 1, 0, 1, 0, 1, 0))

n <- max(dim(B))

blk <- c("s" = n)
```

With the objective function in the form $\langle \mathbf{C}, \mathbf{X} \rangle$, we define the input C as

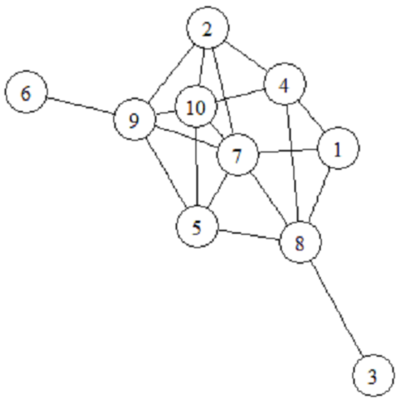
```
 one <- matrix(1, nrow = n, ncol = 1)
 C <- -(diag(c(B %*% one)) - B) / 4
```

where, again, **B** is the adjacency matrix for a graph on which we would like to find the maximum cut, such as the one in Figure 1.

The matrix At is constructed using the upper triangular portion of the $\mathbf{A}_k$ matrices. To do this in R, the function svec is made available in **sdpt3r**.

```
  A <- list()
  for(k in 1:n){
    A[[k]] <- Matrix(0,n,n)
    A[[k]][k,k] <- 1
  }

 At <- svec(blk[1],A,1)
```



$$\mathbf{B} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

Figure 1: *A graph object and associated adjacency matrix for which we would like to find the maximum cut.*

Having each of the diagonal elements of **X** constrained to be 1, b is a $n \times 1$ matrix of ones

```
b <- matrix(1, nrow = n, ncol = 1)
```

With all the input variables now defined, we can now call sqlp to solve the Max-Cut problem

```
sqlp(blk, At, list(C), b)
```

A numerical example and the maxcut function

The built-in function maxcut takes as input a (weighted) adjacency matrix B and returns the optimal solution directly. If we wish to find to the maximum cut of the graph in Figure 1, given the adjacency matrix **B** we can compute using maxcut as

```
out <- maxcut(B)
out

$pobj

[1] -14.67622

$X

      [,1]   [,2]   [,3]   [,4]   [,5]   [,6]   [,7]   [,8]   [,9]  [,10]
V1   1.000  0.987 -0.136 -0.858  0.480  0.857 -0.879  0.136 -0.857  0.597
V2   0.987  1.000  0.026 -0.763  0.616  0.929 -0.791 -0.026 -0.929  0.459
V3  -0.136  0.026  1.000  0.626  0.804  0.394  0.592 -1.000 -0.394 -0.876
V4  -0.858 -0.763  0.626  1.000  0.039 -0.469  0.999 -0.626  0.470 -0.925
V5   0.480  0.616  0.804  0.039  1.000  0.864 -0.004 -0.804 -0.864 -0.417
V6   0.857  0.929  0.394 -0.469  0.864  1.000 -0.508 -0.394 -1.000  0.098
V7  -0.879 -0.791  0.592  0.999 -0.004 -0.508  1.000 -0.592  0.508 -0.907
V8   0.136 -0.026 -1.000 -0.626 -0.804 -0.394 -0.592  1.000  0.394  0.876
V9  -0.857 -0.929 -0.394  0.470 -0.864 -1.000  0.508  0.394  1.000 -0.098
V10  0.597  0.459 -0.876 -0.925 -0.417  0.098 -0.907  0.876 -0.098  1.000
```

Note that the value of the primary objective function is negative as we have defined $\mathbf{C} = -(diag(\mathbf{B1}) - \mathbf{B})/4$ since we require the primal formulation to be a minimization problem. The original formulation given in Goemans and Williamson (1995) frames the Max-Cut problem as a maximization problem with $\mathbf{C} = (diag(\mathbf{B1}) - \mathbf{B})/4$. Therefore, the approximate value of the maximum cut for the graph in Figure 1 is 14.68 (recall we are solving a relaxation).

As an interesting aside, we can show that the matrix **X** is actually a correlation matrix by considering its eigenvalues - we can see it clearly is symmetric, with unit diagonal and all elements in [-1,1].

```
eigen(out$X[[1]])

$values
```

```
[1] 5.59e+00 4.41e+00 2.07e-07 1.08e-07 4.92e-08 3.62e-08 3.22e-08
[8] 1.90e-08 1.66e-08 9.38e-09
```

The fact that $\mathbf{X}$ is indeed a correlation matrix comes as no surprise. Goemans and Williamson (1995) show that the set of feasible solutions for the Max-Cut problem is in fact the set of correlation matrices. So while we may not be interested in $\mathbf{X}$ as an output for solving the Max-Cut problem, it is nonetheless interesting to see that it is in fact in the set of feasible solutions.

Nearest correlation matrix

Recall that the nearest correlation matrix is found as the solution to

$$\begin{array}{ll} \underset{\mathbf{e}^*,\,\mathbf{X}}{\text{minimize}} & e_0 \\ \text{subject to} & \\ & \begin{array}{rcl} svec(\mathbf{R}) - svec(\mathbf{X}) & = & [\mathbf{0}, \mathbf{I}_{n(n+1)/2}]\,\mathbf{e}^* \\ diag(\mathbf{X}) & = & \mathbf{1} \\ \mathbf{X} & \in & \mathcal{S}^n \\ \mathbf{e}^* & \in & \mathcal{Q}^{n(n+1)/2+1} \end{array} \end{array}$$

In Section 52.2.1 we wrote this as the following SQLP

$$\begin{array}{ll} \underset{\mathbf{e}^*,\,\mathbf{X}}{\text{minimize}} & \langle \mathbf{C}, \mathbf{e}^* \rangle \\ \text{subject to} & \\ & \begin{array}{rcl} (\mathbf{A}^s)^\mathsf{T} svec(\mathbf{X}) + (\mathbf{A}^q)^\mathsf{T} \mathbf{e}^* & = & \mathbf{b} \\ \mathbf{X} & \in & \mathcal{S}^n \\ \mathbf{e}^* & \in & \mathcal{Q}^{n(n+1)/2+1} \end{array} \end{array}$$

for $\mathbf{C} = [1, \mathbf{0}_{n(n+1)/2}]$, and

$$\begin{array}{rcl} \mathbf{A}^s & = & [\mathbf{A}_1^s,\ \mathbf{A}_2^s] \\ \mathbf{A}^q & = & [\mathbf{A}_1^q,\ \mathbf{A}_2^q] \end{array} \qquad \mathbf{b} = [\mathbf{b}_1;\ \mathbf{b}_2]$$

where

$$\begin{array}{rcl} \mathbf{A}_1^s & = & \mathbf{I}_{n_2} \\ \mathbf{A}_1^q & = & [\mathbf{0}, \mathbf{I}_{n_2}] \\ & & \\ \mathbf{A}_2^s & = & [svec(\mathbf{A}_1), \ldots, svec(\mathbf{A}_n)] \\ \mathbf{A}_2^q & = & \mathbf{0}_{n_2} \end{array} \qquad \begin{array}{rcl} \mathbf{b}_1 & = & svec(\mathbf{R}) \\ \mathbf{b}_2 & = & \mathbf{1}^\mathsf{T} \end{array}$$

and $\mathbf{A}_1, \ldots, \mathbf{A}_n$ are given by

$$\mathbf{A}_k = [a_{ij}] = \begin{cases} 1, & i = j = k \\ 0, & \text{otherwise} \end{cases}$$

To be solved using `sqlp`, we first define `blk`. There are two optimization variables in the formulation of the nearest correlation matrix - $\mathbf{X}$ is an $n \times n$ matrix constrained to be in a semidefinite cone, and $\mathbf{y}$ is an $n(n+1)/2 + 1$ length vector constrained to be in a quadratic cone, so

```
data(Hnearcorr)

X = Hnearcorr
n = max(dim(X))
n2 = n * (n + 1) / 2

blk <- c("s" = n, "q" = n2+1)
```

Note that $\mathbf{X}$ does not appear in the objective function, so the `C` entry corresponding to the block variable $\mathbf{X}$ is an $n \times n$ matrix of zeros, which defines `C` as

```
C1 <- matrix(0, nrow = n, ncol = n)
C2 <- rbind(1, matrix(0, nrow = n2, ncol = 1))
C <- list(C1,C2)
```

Next comes the constraint matrix for **X**

```
Aks <- matrix(list(), nrow = 1, ncol = n)
for(k in 1:n){
  Aks[[k]] <- matrix(0, nrow = n, ncol = n)
  diag(Aks[[k]])[k] <- 1
}

A1s <- svec(blk[1], Aks)[[1]]
A2s <- diag(1, nrow = n2, ncol = n2)

At1 <- cbind(A1s,A2s)
```

then the constraint matrix for $\mathbf{e}^*$.

```
A1q <- matrix(0, nrow = n, ncol = n2 + 1)

A2q1 <- matrix(0, nrow = n2, ncol = 1)
A2q2 <- diag(1, nrow = n2, ncol = n2)
A2q <- cbind(A2q1, A2q2)

At2 <- rbind(A1q, A2q)
```

and the right hand side vector b

```
b <- rbind(matrix(1, n, 1),svec(blk[1], X))
```

The nearest correlation matrix problem is now solved by

```
sqlp(blk, list(At1,At2), C, b)
```

A numerical example and the nearcorr function

To demonstrate the nearest correlation matrix problem, we will modify an existing correlation matrix by exploring the effect of changing the sign of just one of the pairwise correlations. In the context of stock correlations, we make use of tools available in the R package **quantmod** (Ryan and Ulrich, 2017) to access stock data from five tech firms (Microsoft, Apple, Amazon, Alphabet/Google, and IBM) beginning in 2007.

```
library("quantmod")

getSymbols(c("MSFT", "AAPL", "AMZN", "GOOGL", "IBM"))
stock.close <- as.xts(merge(MSFT, AAPL, AMZN,
   GOOGL, IBM))[, c(4, 10, 16, 22, 28)]
```

The correlation matrix for these five stocks is

```
stock.corr <- cor(stock.close)
stock.corr
```

```
           MSFT.Close AAPL.Close AMZN.Close GOOGL.Close IBM.Close
MSFT.Close  1.0000000 -0.2990463  0.9301085   0.5480033 0.2825698
AAPL.Close -0.2990463  1.0000000 -0.1514348   0.3908624 0.6887127
AMZN.Close  0.9301085 -0.1514348  1.0000000   0.6228299 0.3870390
GOOGL.Close  0.5480033  0.3908624  0.6228299   1.0000000 0.5885146
IBM.Close   0.2825698  0.6887127  0.3870390   0.5885146 1.0000000
```

Next, consider the effect of having a positive correlation between Microsoft and Apple

```
stock.corr[1, 2] <- -1 * stock.corr[1, 2]
stock.corr[2, 1] <- stock.corr[1, 2]
stock.corr
```

```
           MSFT.Close AAPL.Close AMZN.Close GOOGL.Close IBM.Close
MSFT.Close  1.0000000  0.2990463  0.9301085   0.5480033 0.2825698
AAPL.Close  0.2990463  1.0000000 -0.1514348   0.3908624 0.6887127
```

```
AMZN.Close   0.9301085 -0.1514348  1.0000000   0.6228299 0.3870390
GOOGL.Close  0.5480033  0.3908624  0.6228299   1.0000000 0.5885146
IBM.Close    0.2825698  0.6887127  0.3870390   0.5885146 1.0000000
```

Unfortunately, this correlation matrix is not positive semidefinite

```
eigen(stock.corr)$values
```

```
[1]  2.8850790  1.4306393  0.4902211  0.3294150 -0.1353544
```

Given the approximate correlation matrix `stock.corr`, the built-in function `nearcorr` solves the nearest correlation matrix problem using `sqlp`

```
out <- nearcorr(stock.corr)
```

Since this is a minimization problem, and thus a primal formulation of the SQLP, the output `X` from `sqlp` will provide the optimal solution to the problem - that is, `X` will be the nearest correlation matrix to `stock.corr`.

```
out$X
```

```
          [,1]        [,2]        [,3]      [,4]      [,5]
[1,] 1.0000000  0.25388359  0.86150833 0.5600734 0.3126420
[2,] 0.2538836  1.00000000 -0.09611382 0.3808981 0.6643566
[3,] 0.8615083 -0.09611382  1.00000000 0.6115212 0.3480430
[4,] 0.5600734  0.38089811  0.61152116 1.0000000 0.5935021
[5,] 0.3126420  0.66435657  0.34804303 0.5935021 1.0000000
```

The matrix above is symmetric with unit diagonal and all entries in $[-1, 1]$. By checking the eigenvalues,

```
eigen(out$X)
```

```
$values
```

```
[1] 2.846016e+00 1.384062e+00 4.570408e-01 3.128807e-01 9.680507e-11
```

we can see that `X` is indeed a correlation matrix.

D-optimal experimental design

Recall from Section 52.2.2 that the D-Optimal experimental design problem was stated as the following dual SQLP

$$\begin{array}{ll} \underset{\mathbf{Z},\,\mathbf{z}^l,\,\boldsymbol{\lambda}}{\text{maximize}} & \log\det(\mathbf{Z}) \\ \text{subject to} & \\ & \begin{array}{rcll} -\sum_{i=1}^{p} \lambda_i(\mathbf{u}_i\mathbf{u}_i^{\mathsf{T}}) + \mathbf{Z} & = & 0, & \mathbf{Z} \in \mathcal{S}^n \\ -\boldsymbol{\lambda} + \mathbf{z}^l & = & 0, & \mathbf{z}^l \in \mathcal{R}^p_+ \\ \mathbf{1}^T\boldsymbol{\lambda} & = & 1, & \boldsymbol{\lambda} \in \mathcal{R}^p \end{array} \end{array}$$

which we wrote as

$$\begin{array}{ll} \underset{\mathbf{Z},\,\mathbf{z}^l,\,\boldsymbol{\lambda}}{\text{maximize}} & \log\det(\mathbf{Z}) \\ \text{subject to} & \\ & \begin{array}{rcll} (\mathbf{A}^s)^{\mathsf{T}}\boldsymbol{\lambda} \;+\; \mathbf{Z} & = & \mathbf{C}^s, & \mathbf{Z} \in \mathcal{S}^n \\ (\mathbf{A}^l)^{\mathsf{T}}\boldsymbol{\lambda} \;+\; \mathbf{z}^l & = & \mathbf{C}^q, & \mathbf{z}^l \in \mathcal{R}^p_+ \\ (\mathbf{A}^u)^{\mathsf{T}}\boldsymbol{\lambda} & = & \mathbf{C}^u, & \boldsymbol{\lambda} \in \mathcal{R}^p \end{array} \end{array}$$

where $\mathbf{b} = \mathbf{0}$, and

$$\begin{array}{llcll} \mathbf{A}^s & = -[svec(\mathbf{A}_1), \ldots, svec(\mathbf{A}_p)] & \qquad & \mathbf{C}^s & = \mathbf{0}_{n\times n} \\ \mathbf{A}^l & = -\mathbf{I}_p & & \mathbf{C}^l & = \mathbf{0}_{p\times 1} \\ \mathbf{A}^u & = \mathbf{1}^{\mathsf{T}} & & \mathbf{C}^u & = 1 \end{array}$$

Here, $\mathbf{A}_1, \ldots, \mathbf{A}_p$ are given by

$$\mathbf{A}_i = \mathbf{u}_i \mathbf{u}_i^\mathsf{T}, \quad i = 1, \ldots, p$$

To convert this to a form usable by **sdpt3r**, we first declare the three blocks in `blk`. For a matrix The first block is semidefinite containing the matrix $\mathbf{Z}$, the second a linear block containing $\mathbf{z}^l$, and the third an unrestricted block containing $\boldsymbol{\lambda}$

```
data(DoptDesign)
V = DoptDesign
n = nrow(V)
p = ncol(V)

blk = c("s" = n, "l" = p, "u" = 1)
```

Next, by noting the variable $\boldsymbol{\lambda}$ does not appear in the objective function, we specify `b` as a vector of zeros

```
b <- matrix(0, nrow = p, ncol = 1)
```

Next, looking at the right-hand side of the constraints, we define the matrices `C`

```
C1 <- matrix(0, nrow = n, ncol = n)
C2 <- matrix(0, nrow = p, ncol = 1)
C3 <- 1

C = list(C1,C2,C3)
```

Finally, we construct `At` for each variable

```
A <- matrix(list(), nrow = p, ncol = 1)

for(k in 1:p){
  A[[k]] <- -V[,k] %*% t(V[,k])
}

At1 <- svec(blk[1], A)[[1]]
At2 <- diag(-1, nrow = p, ncol = p)
At3 <- matrix(1, nrow = 1, ncol = p)

At = list(At1,At2,At3)
```

The final hurdle necessary to address in this problem is the existence of the log-barrier. Recall that it is assumed that v^s, v^q, and v^l in Equation 4 are all zero in `control`. In this case, we can see that is not true, as we have a log term containing $\mathbf{Z}$ in the objective function, meaning v^s is equal to one. To pass this to `sqlp`, we define the `control$parbarrier` variable as

```
control <- list(parbarrier = matrix(list(),3,1))
control$parbarrier[[1]] <- 1
control$parbarrier[[2]] <- 0
control$parbarrier[[3]] <- 0
```

The D-Optimal experimental design problem can now be solved using `sqlp`

```
sqlp(blk, At, C, b, control)
```

A numerical example and the doptimal function

To demonstrate the output generated from a D-optimal experimental design problem, we consider a simple 3×25 matrix containing the known test vectors $\mathbf{u}_1, ..., \mathbf{u}_{25}$ (the data is available in the `sqlp` package). To solve the problem using`sqlp`, we use the function `doptimal`, which takes as input an $n \times p$ matrix $\mathbf{U}$ containing the known test vectors, and returns the optimal solution. The output we are interested in is `y`, corresponding to $\boldsymbol{\lambda}$ in our formulation, the percentage of each $\mathbf{u}_i$ necessary to achieve maximum information in the experiment.

```
data("DoptDesign")

out <- doptimal(DoptDesign)
```

```
out$y
       [,1]
 [1,] 0.000
 [2,] 0.000
 [3,] 0.000
 [4,] 0.000
 [5,] 0.000
 [6,] 0.000
 [7,] 0.154
 [8,] 0.000
 [9,] 0.000
[10,] 0.000
[11,] 0.000
[12,] 0.000
[13,] 0.319
[14,] 0.000
[15,] 0.000
[16,] 0.240
[17,] 0.000
[18,] 0.000
[19,] 0.000
[20,] 0.000
[21,] 0.000
[22,] 0.000
[23,] 0.287
[24,] 0.000
[25,] 0.000
```

The information matrix $\mathbf{A}^\top\mathbf{A}$ is a linear combination of the test vectors $\mathbf{u}_i$, weighted by the optimal vector y above.

Additional problems

The **sdpt3r** package considerably broadens the set of optimization problems that can be solved in R. In addition to those problems presented in detail in Section 52.3, there are a large number of well known problems that can also be formulated as an SQLP.

Each problem presented will be described briefly, with appropriate references for the interested reader, and presented mathematically in its classical form, not as an SQLP as in Equation 3 or 4. Accompanying each problem will be an R helper function, which will solve the corresponding problem using `sqlp`. Each helper function in **sdpt3r** (including those for the max-cut, D-optimal experimental design, and nearest correlation matrix) is an R implementation of the helper functions that are available to the user in the Matlab **SDPT3** package (Toh et al., 1999).

Minimum volume ellipsoids

The problem of finding the ellipsoid of minimum volume containing a set of points $\mathbf{v}_1, ..., \mathbf{v}_n$ is stated as the following optimization problem (Vandenberghe et al., 1998)

$$\begin{array}{ll} \underset{\mathbf{B},\,\mathbf{d}}{\text{maximize}} & log\ det(\mathbf{B}) \\ \text{subject to} & \\ & ||\mathbf{Bx}+\mathbf{d}|| \quad \leq \quad 1, \quad \forall]vex \ \in \ [\mathbf{v}_1, ..., \mathbf{v}_n] \end{array}$$

The function `minelips` takes as input an $n \times p$ matrix $\mathbf{V}$ containing the points around which we would like to find the minimum volume ellipsoid, and returns the optimal solution using `sqlp`.

```
data(Vminelips)
out <- minelips(Vminelips)
```

Distance weighted discrimination

Given two sets of points in a matrix $\mathbf{X} \in \mathcal{R}^n$ with associated class variables [-1,1] in $\mathbf{Y} = diag(\mathbf{y})$, distance weighted discrimination (Marron et al., 2007) seeks to classify the points into two distinct subsets by finding a hyperplane between the two sets of points. Mathematically, the distance weighted discrimination problem seeks a hyperplane defined by a normal vector, $\boldsymbol{\omega}$, and position, β, such that each element in the residual vector $\bar{\mathbf{r}} = \mathbf{Y}\mathbf{X}^{\mathsf{T}}\omega + \beta\mathbf{y}$ is positive and large. Since the class labels are either 1 or -1, having the residuals be positive is equivalent to having the points on the proper side of the hyperplane.

Of course, it may be impossible to have a perfect separation of points using a linear hyperplane, so an error term ξ is introduced. Thus, the perturbed residuals are defined to be

$$\mathbf{r} = \mathbf{Y}\mathbf{X}^{\mathsf{T}}\omega + \beta\mathbf{y} + \xi$$

Distance weighted discrimination (Marron et al., 2007) solves the following optimization problem to find the optimal hyperplane.

$$\begin{array}{ll} \underset{\mathbf{r},\,\omega,\,\beta,\,\xi}{\text{minimize}} & \sum_{i=1}^{n}(1/r_i) + C\mathbf{1}^{\mathsf{T}}\xi \\ \text{subject to} & \\ & \begin{array}{rcl} \mathbf{r} & = & \mathbf{Y}\mathbf{X}^{\mathsf{T}}\omega + \beta\mathbf{y} + \xi \\ \omega^{\mathsf{T}}\omega & \leq & 1 \\ \mathbf{r} & \geq & \mathbf{0} \\ \xi & \geq & \mathbf{0} \end{array} \end{array}$$

where $C > 0$ is a penalty parameter to be chosen.

The function `dwd` takes as input two $n \times p$ matrices `X1` and `X2` containing the points to be separated, as well as a penalty term `C` ≥ 0 penalizing the movement of a point on the wrong side of the hyperplane to the proper side, and returns the optimal solution using `sqlp`.

```
data(Andwd)
data(Apdwd)
C <- 0.5

out <- dwd(Apdwd,Andwd,penalty)
```

Max-kCut

Similar to the Max-Cut problem, the Max-kCut problem asks, given a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ and an integer k, does a cut exist of at least size k. For a given (weighted) adjacency matrix $\mathbf{B}$ and integer k, the Max-kCut problem is formulated as the following primal problem

$$\begin{array}{ll} \underset{\mathbf{X}}{\text{minimize}} & \langle \mathbf{C},\, \mathbf{X} \rangle \\ \text{subject to} & \\ & \begin{array}{rcll} diag(\mathbf{X}) & = & \mathbf{1} & \\ X_{ij} & \geq & 1/(k-1) & \forall\, i \neq j \\ \mathbf{X} & \in & \mathcal{S}_n & \end{array} \end{array}$$

Here, $\mathbf{C} = -(1-1/k)/2 * (diag(\mathbf{B1}) - \mathbf{B})$. The Max-kCut problem is slightly more complex than the Max-Cut problem due to the inequality constraint. In order to turn this into a standard SQLP, we must replace the inequality constraints with equality constraints, which we do by introducing a slack variable $\mathbf{x}^l$, allowing the problem to be restated as

$$\begin{array}{ll} \underset{\mathbf{X}}{\text{minimize}} & \langle \mathbf{C},\, \mathbf{X} \rangle \\ \text{subject to} & \\ & \begin{array}{rcll} diag(\mathbf{X}) & = & \mathbf{1} & \\ X_{ij} - x^l & = & 1/(k-1) & \forall\, i \neq j \\ \mathbf{X} & \in & \mathcal{S}^n & \\ \mathbf{x}^l & \in & \mathcal{L}^{n(n+1)/2} & \end{array} \end{array}$$

The function `maxkcut` takes as input an adjacency matrix `B` and an integer `k`, and returns the optimal solution using `sqlp`.

```
data(Bmaxkcut)
k = 2

out <- maxkcut(Bmaxkcut,k)
```

Graph partitioning problem

The graph partitioning problem can be formulated as the following primal optimization problem

$$\begin{array}{ll} \underset{\mathbf{X}}{\text{minimize}} & tr(\mathbf{CX}) \\ \text{subject to} & \\ & \begin{array}{rcl} tr(\mathbf{11}^{\mathsf{T}}\mathbf{X}) & = & \alpha \\ diag(\mathbf{X}) & = & \mathbf{1} \end{array} \end{array}$$

Here, $\mathbf{C} = -(diag(\mathbf{B1}) - \mathbf{B})$, for an adjacency matrix $\mathbf{B}$, and α is any real number.

The function gpp, takes as input a weighted adjacency matrix B and a real number alpha and returns the optimal solution using sqlp.

```
data(Bgpp)
alpha <- nrow(Bgpp)

out <- gpp(Bgpp, alpha)
```

The Lovasz number

The Lovasz Number of a graph $\mathbf{G}$, denoted $\vartheta(\mathbf{G})$, is the upper bound on the Shannon capacity of the graph. For an adjacency matrix $\mathbf{B} = [B_{ij}]$ the problem of finding the Lovasz number is given by the following primal SQLP problem

$$\begin{array}{ll} \underset{\mathbf{X}}{\text{minimize}} & tr(\mathbf{CX}) \\ \text{subject to} & \\ & \begin{array}{rcll} tr(\mathbf{X}) & = & 1 & \\ X_{ij} & = & 0 & \text{if } B_{ij} = 1 \\ \mathbf{X} & \in & \mathcal{S}^n & \end{array} \end{array}$$

The function lovasz takes as input an adjacency matrix $\mathbf{B}$, and returns the optimal solution using sqlp.

```
data(Glovasz)

out <- lovasz(Glovasz)
```

Toeplitz approximation

Given a symmetric matrix $\mathbf{F}$, the Toeplitz approximation problem seeks to find the nearest symmetric positive definite Toeplitz matrix. In general, a Toeplitz matrix is one with constant descending diagonals, i.e.,

$$\mathbf{T} = \begin{bmatrix} a & b & c & d & e \\ f & a & b & c & d \\ g & f & a & b & c \\ h & g & f & a & b \\ i & h & g & f & a \end{bmatrix}$$

is a general Toeplitz matrix. The problem is formulated as the following optimization problem

$$\begin{array}{ll} \underset{\mathbf{X}}{\text{maximize}} & -y_{n+1} \\ \text{subject to} & \\ & \begin{array}{rcl} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & -\beta \end{bmatrix} + \sum_{k=1}^{n} y_k \begin{bmatrix} \mathbf{0} & \gamma_k \mathbf{e}_k \\ \gamma_k \mathbf{e}_k^T & -2q_k \end{bmatrix} + y_{n+1}\mathbf{B} & \geq & \mathbf{0} \\ [y_1, ..., y_n]^{\mathsf{T}} + y_{n+1}\mathbf{B} & \geq & \mathbf{0} \end{array} \end{array}$$

where $\mathbf{B}$ is an $(n+1) \times (n+1)$ matrix of zeros, and $\mathbf{B}_{(n+1)(n+1)} = 1$, $q_1 = -tr(\mathbf{F})$, q_k = sum of k^{th} diagonal upper and lower triangular matrix, $\gamma_1 = \sqrt{n}$, $\gamma_k = \sqrt{2 * (n-k+1)}$, $k = 2, ..., n$, and $\beta = ||\mathbf{F}||_F^2$.

The function toep takes as input a symmetric matrix F for which we would like to find the nearest Toeplitz matrix, and returns the optimal solution using sqlp.

```
data(Ftoep)

out <- toep(Ftoep)
```

The educational testing problem

The educational testing problem arises in measuring the reliability of a student's total score in an examination consisting of a number of sub-tests (Fletcher, 1981). In terms of formulation as an optimization problem, the problem is to determine how much can be subtracted from the diagonal of a given symmetric positive definite matrix $\mathbf{S}$ such that the resulting matrix remains positive semidefinite (Chu and Wright, 1995).

The Educational Testing Problem (ETP) is formulated as the following dual problem

$$\begin{array}{ll} \underset{\mathbf{d}}{\text{maximize}} & \mathbf{1}^{\mathsf{T}}\mathbf{d} \\ \text{subject to} & \\ & \begin{array}{rcl} \mathbf{A} - diag(\mathbf{d}) & \succeq & \mathbf{0} \\ \mathbf{d} & \geq & \mathbf{0} \end{array} \end{array}$$

where $\mathbf{d} = [d_1, d_2, ..., d_n]$ is a vector of size n and $diag(\mathbf{d})$ denotes the corresponding $n \times n$ diagonal matrix. In the second constraint, having each element in $\mathbf{d}$ be greater than or equal to 0 is equivalent to having $diag(\mathbf{d}) \succeq 0$.

The corresponding primal problem is

$$\begin{array}{ll} \underset{\mathbf{X}}{\text{minimize}} & tr(\mathbf{AX}) \\ \text{subject to} & \\ & \begin{array}{rcl} diag(\mathbf{X}) & \geq & \mathbf{1} \\ \mathbf{X} & \succeq & 0 \end{array} \end{array}$$

The function `etp` takes as input an $n \times n$ positive definite matrix `A`, and returns the optimal solution using `sqlp`.

```
data(Betp)

out <- etp(Betp)
```

Logarithmic Chebyshev approximation

For a $p \times n$ $(p > n)$ matrix $\mathbf{B}$ and $p \times 1$ vector $\mathbf{f}$, the Logarithmic Chebyshev Approximation problem is stated as the following optimization problem (Vandenberghe et al., 1998)

$$\begin{array}{ll} \underset{\mathbf{x},\, t}{\text{minimize}} & t \\ \text{subject to} & \\ & 1/t \quad \leq \quad (\mathbf{x}^{\mathsf{T}}\mathbf{B}_{i\cdot})/\mathbf{f}_i \quad \leq \quad t, \quad i = 1, ..., p \end{array}$$

where $\mathbf{B}_{i\cdot}$ denotes the i^{th} row of the matrix $\mathbf{B}$. Note that we require each element of $\mathbf{B}_{\cdot j}/\mathbf{f}$ to be greater than or equal to 0 for all j.

The function `logcheby` takes as input a matrix `B` and vector `f`, and returns the optimal solution to the Logarithmic Chebyshev Approximation problem using `sqlp`.

```
data(Blogcheby)
data(flogcheby)

out <- logcheby(Blogcheby, flogcheby)
```

Linear matrix inequality problems

We consider three distinct linear matrix inequality problems, all written in the form of a dual optimization problem. The first linear matrix inequality problem we will consider is defined by the following optimization equation for some $n \times p$ matrix $\mathbf{B}$ known in advance

$$
\begin{array}{ll}
\underset{\eta,\,\mathbf{Y}}{\text{maximize}} & -\eta \\
\text{subject to} & \\
& \begin{array}{rcl}
\mathbf{BY} + \mathbf{YB}^{\mathsf{T}} & \preceq & 0 \\
-\mathbf{Y} & \preceq & -\mathbf{I} \\
\mathbf{Y} - \eta\mathbf{I} & \preceq & 0 \\
Y_{11} & = & 1, \quad \mathbf{Y} \in \mathcal{S}^n
\end{array}
\end{array}
$$

The function `lmi1` takes as input a matrix **B**, and returns the optimal solution using `sqlp`.

```
B <- matrix(c(-1,5,1,0,-2,1,0,0,-1), nrow=3)

out <- lmi1(B)
```

The second linear matrix inequality problem is

$$
\begin{array}{ll}
\underset{\mathbf{P},\,\mathbf{d}}{\text{maximize}} & -tr(\mathbf{P}) \\
\text{subject to} & \\
& \begin{array}{rcl}
\mathbf{A}_1\mathbf{P} + \mathbf{P}\mathbf{A}_1{}^{\mathsf{T}} + \mathbf{B} * diag(\mathbf{d}) * \mathbf{B}^{\mathsf{T}} & \preceq & 0 \\
\mathbf{A}_2\mathbf{P} + \mathbf{P}\mathbf{A}_2{}^{\mathsf{T}} + \mathbf{B} * diag(\mathbf{d}) * \mathbf{B}^{\mathsf{T}} & \preceq & 0 \\
-\mathbf{d} & \preceq & 0 \\
\sum_i^p d_i & = & 1
\end{array}
\end{array}
$$

Here, the matrices **B**, $\mathbf{A}_1$, and $\mathbf{A}_2$ are known in advance.

The function `lmi2` takes the matrices `A1`, `A2`, and `B` as input, and returns the optimal solution using `sqlp`.

```
A1 <- matrix(c(-1,0,1,0,-2,1,0,0,-1),3,3)
A2 <- A1 + 0.1*t(A1)
B  <- matrix(c(1,3,5,2,4,6),3,2)

out <- lmi2(A1,A2,B)
```

The final linear matrix inequality problem originates from a problem in control theory (Boyd et al., 1994) and requires three matrices be known in advance, **A**, **B**, and **G**

$$
\begin{array}{ll}
\underset{\eta,\,\mathbf{P}}{\text{maximize}} & \eta \\
\text{subject to} & \\
& \begin{bmatrix} \mathbf{AP} + \mathbf{PA}^{\mathsf{T}} & \mathbf{0} \\ \mathbf{BP} & \mathbf{0} \end{bmatrix} + \eta \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \preceq \begin{bmatrix} -\mathbf{G} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}
\end{array}
$$

The function `lmi3` takes as input the matrices `A`, `B`, and `G`, and returns the optimal solution using `sqlp`.

```
A <- matrix(c(-1,0,1,0,-2,1,0,0,-1),3,3)
B <- matrix(c(1,2,3,4,5,6), 2, 3)
G <- matrix(1,3,3)

out <- lmi3(A,B,G)
```

Summary

In Section 52.2, we introduced the problem of conic linear optimization. Using the Max-Cut, Nearest Correlation Matrix, and D-Optimal Experimental Design problems as examples, we demonstrated the increasing generality of the problem, culminating in a general form of the conic linear optimization problem, known as the semidefinite quadratic linear program, in Section 52.2.2.

In Section 52.3, we introduced the R package **sdpt3r**, and the main function call available in the package, `sqlp`. The specifics of the necessary input variables, the optional input variables, and the output variables provided by `sqlp` were presented. Using the examples from Section 52.2, we showed how a problem written as a semidefinite quadratic linear program could be solved in R using **sdpt3r**.

Finally, in Section 52.4, we presented a number of additional problems that can be solved using the **sdpt3r** package, and presented the helper functions available so these problems could be easily solved using `sqlp`.

The **sdpt3r** package broadens the range of problems that can be solved using R. Here, we discussed a number of problems that can be solved using **sdpt3r**, including problems in the statistical sciences, graph theory, classification, control theory, and general matrix theory. The `sqlp` function in **sdpt3r** is in fact even more general, and users may apply it to any other conic linear optimization problem that can be written in the form of Equation 3 or 4 by specifying the input variables `blk`, `At`, `C`, and `b` for their particular problem.

control

`vers`	specifies the search direction
	0, HKM if semidefinite blocks present, NT otherwise (default)
	1, HKM direction
	2, NT direction
`predcorr`	TRUE, use Mehrotra prediction-correction (default)
	FALSE, otherwise
`gam`	step-length (default 0)
`expon`	exponent used to decrease sigma (default 1)
`gaptol`	tolerance for duality gap as a fraction of the objective function (default $1e-8$)
`inftol`	tolerance for stopping due to infeasibility (default 1e-8)
`steptol`	tolerance for stopping due to small steps (default 1e-6)
`maxit`	maximum number of iterations (default 100)
`stoplevel`	0, continue until successful completion, maximum iteration, or numerical failure
	1, automatically detect termination, restart if small steps is cause (default)
	2, automatically detect termination
`scale_data`	TRUE, scale data prior to solving
	FALSE, otherwise (default)
`rmdepconstr`	TRUE, remove nearly dependent constraints
	FALSE, otherwise (default)
`parbarrier`	declare the existence of a log barrier term
	default value is 0 (i.e., no log barrier)

Bibliography

S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*. SIAM, 1994. [p391]

H. C. Bravo. *Rcsdp: R Interface to the CSDP Semidefinite Programming Library*, 2016. URL http://CRAN.R-project.org/package=Rcsdp. R package version 0.1.55. [p371]

M. T. Chu and J. W. Wright. The educational testing problem revisited. *IMA journal of numerical analysis*, 15(1):141–160, 1995. [p371, 390]

R. Fletcher. A nonlinear programming problem in statistics (educational testing). *SIAM Journal on Scientific and Statistical Computing*, 2(3):257–267, 1981. [p390]

R. M. Freund. Introduction to semidefinite programming (sdp). *Massachusetts Institute of Technology*, 2004. [p380]

H. Friberg. Rmosek: The r-to-mosek optimization interface. *R package version*, 1(3), 2012. [p371]

A. Fu, B. Narasimhan, and S. Boyd. CVXR: An R Package for Disciplined Convex Optimization. *arXiv*, 2017. URL https://arxiv.org/abs/1711.07582v1. [p371]

M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995. [p372, 382, 383]

C. Helmberg, F. Rendl, R. J. Vanderbei, and H. Wolkowicz. An interior-point method for semidefinite programming. *SIAM Journal on Optimization*, 6(2):342–361, 1996. [p376]

N. J. Higham. Computing the nearest correlation matrix-a problem from finance. *IMA Journal of Numerical Analysis*, 22(3):329–343, 2002. [p371, 373]

F. John. Extremum problems with inequalities as subsidiary conditions. In *Traces and Emergence of Nonlinear Programming*, pages 197–215. Springer-Verlag, 2014. [p371]

R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer-Verlag, 1972. [p372]

J. S. Marron, M. J. Todd, and J. Ahn. Distance-weighted discrimination. *Journal of the American Statistical Association*, 102(480):1267–1271, 2007. [p371, 387, 388]

Y. E. Nesterov and M. J. Todd. Self-scaled barriers and interior-point methods for convex programming. *Mathematics of Operations research*, 22(1):1–42, 1997. [p376]

B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. SCS: Splitting conic solver, version 2.0.2. https://github.com/cvxgrp/scs, 2017. [p371]

C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of computer and system sciences*, 43(3):425–440, 1991. [p372]

B. Pfaff. *Cccp: Cone Constrained Convex Problems*, 2015. URL http://CRAN.R-project.org/package=cccp. R package version 0.2-4. [p371]

J. A. Ryan and J. M. Ulrich. *Quantmod: Quantitative Financial Modelling Framework*, 2017. URL http://CRAN.R-project.org/package=quantmod. R package version 0.4-9. [p384]

K. Smith. On the standard deviations of adjusted and interpolated values of an observed polynomial function and its constants and the guidance they give towards a proper choice of the distribution of observations. *Biometrika*, 12(1-2):1–85, 1918. [p371]

S. Theußl, F. Schwendinger, and K. Hornik. Roi: The r optimization infrastructure package. Research Report Series / Department of Statistics and Mathematics 133, WU Vienna University of Economics and Business, Vienna, 2017. URL http://epub.wu.ac.at/5858/. [p371]

K.-C. Toh, M. J. Todd, and R. H. Tütüncü. Sdpt3 - a matlab software package for semidefinite programming, version 1.3. *Optimization methods and software*, 11(1-4):545–581, 1999. [p371, 373, 376, 387]

R. H. Tütüncü, K.-C. Toh, and M. J. Todd. Solving semidefinite-quadratic-linear programs using sdpt3. *Mathematical programming*, 95(2):189–217, 2003. [p372, 376]

L. Vandenberghe, S. Boyd, and S.-P. Wu. Determinant maximization with linear matrix inequality constraints. *SIAM journal on matrix analysis and applications*, 19(2):499–533, 1998. [p371, 376, 387, 390]

Z. Zhu and Y. Ye. *Rdsdp: R Interface to DSDP Semidefinite Programming Library*, 2016. URL http://CRAN.R-project.org/package=Rdsdp. R package version 1.0.4-2. [p371]

Adam Rahman
University of Waterloo
200 University Ave W, Waterloo, Ontario
Canada
a45rahma@uwaterloo.ca