`sdpsol`: A Parser/Solver for Semidefinite Programs with Matrix Structure

Shao-Po Wu and Stephen Boyd

Information Systems Laboratory
Stanford University, Stanford CA 94305, USA

Abstract. A variety of analysis and design problems in control, communication and information theory, statistics, combinatorial optimization, computational geometry, circuit design, and other fields can be expressed as *semidefinite programming* problems (SDPs) or *determinant maximization* problems (max-det problems). These problems often have matrix structure, *i.e.*, some of the optimization variables are matrices. This matrix structure has two important practical ramifications: first, it makes the job of translating the problem into a standard SDP or max-det format tedious, and, second, it opens the possibility of exploiting the structure to speed up the computation.
In this paper we describe the design and implementation of `sdpsol`, a parser/solver for SDPs and max-det problems. `sdpsol` allows problems with matrix structure to be described in a simple, natural, and convenient way. Although the current implementation of `sdpsol` does not exploit matrix structure in the solution algorithm, the language and parser were designed with this goal in mind.

1 Introduction

1.1 Maxdet-problems and SDPs

A *determinant maximization* (max-det) problem has the form

$$\begin{array}{ll} \text{minimize} & c^T x + \sum_{i=1}^K \log\det G^{(i)}(x)^{-1} \\ \text{subject to} & G^{(i)}(x) \succ 0, \quad i = 1,\ldots,K \\ & F^{(i)}(x) \succ 0, \quad i = 1,\ldots,L \\ & Ax = b, \end{array} \tag{1}$$

where the optimization variable is the vector $x \in \mathbf{R}^m$. The matrix functions $G^{(i)} : \mathbf{R}^m \to \mathbf{R}^{l_i \times l_i}$ and $F^{(i)} : \mathbf{R}^m \to \mathbf{R}^{n_i \times n_i}$ are affine:

$$\begin{array}{l} G^{(i)}(x) = G_0^{(i)} + x_1 G_1^{(i)} + \cdots + x_m G_m^{(i)}, \quad i = 1,\ldots,K \\ F^{(i)}(x) = F_0^{(i)} + x_1 F_1^{(i)} + \cdots + x_m F_m^{(i)}, \quad i = 1,\ldots,L \end{array}$$

where $G_j^{(i)}$ and $F_j^{(i)}$ are symmetric for $j = 0,\ldots,m$. The inequality signs in (1) denote matrix inequalities. We call $G^{(i)}(x) \succ 0$ and $F^{(i)}(x) \succ 0$ *linear matrix inequalities* (LMIs) in the variable x. Of course the LMI constraints in (1) can be combined into one large block-diagonal LMI with diagonal blocks $G^{(i)}(x)$ and $F^{(i)}(x)$.

When $K = 1$ and $G^{(1)}(x) = 1$, the max-det problem (1) reduces to the *semidefinite programming* (SDP) problem:

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & F^{(i)}(x) \succ 0, \quad i = 1, \ldots, L \\ & Ax = b. \end{array} \tag{2}$$

The max-det problem (1) and the SDP (2) are convex optimization problems. In fact, LMI constraints can represent many common convex constraints, including linear inequalities, convex quadratic inequalities, and matrix norm and eigenvalue constraints. SDPs and max-det problems arise in many applications; wee Alizadeh [1], Boyd, El Ghaoui, Feron and Balakrishnan [2], Lewis and Overton [10], Nesterov and Nemirovsky [11, §6.4], Vandenberghe and Boyd [15] and Vandenberghe, Boyd and Wu [16] for many examples.

SDPs and max-det problems can be solved very efficiently, both in worst-case complexity theory and in practice, using interior-point methods (see [15] and [16]). Current general purpose solvers such as SP [13], LMITOOL [5], SDPT3 [12], SDPA [8], and SDPMATH [4] use the standard format (2), or a simple variation (*e.g.*, its dual form).

1.2 Matrix structure

In many SDPs and max-det problems the optimization variables are matrices of various dimensions and structure, *e.g.*, row or column vectors, or symmetric or diagonal matrices. In general, the optimization variables can be collected and expressed as $(X^{(1)}, \ldots, X^{(M)})$, where $X^{(i)} \in \mathbf{R}^{p_i \times q_i}$ and $X^{(i)}$ may have structure (*e.g.*, symmetric, diagonal, upper traingular, etc.). These matrix variables can be vectorized into the single vector variable x that appears in the standard format problems (1) and (2). To vectorize $X^{(i)}$, we find a basis $E_1^{(i)}, \ldots, E_{m_i}^{(i)}$ such that

$$X^{(i)} = \sum_{j=1}^{m_i} x_j^{(i)} E_j^{(i)}$$

with $x^{(i)} \in \mathbf{R}^{m_i}$ denotes the vectorized $X^{(i)}$. For example, if $X^{(i)} \in \mathbf{R}^{p_i \times q_i}$ has no structure, we have $x^{(i)} = \mathbf{vec}(X^{(i)})$ and $m_i = p_i q_i$; if $X^{(i)} \in \mathbf{R}^{p_i \times q_i}$ is diagonal ($p_i = q_i$), we have $x^{(i)} = \mathbf{diag}(X^{(i)})$ and $m_i = p_i$. Doing this for $i = 1, \ldots, M$, we obtain the vectorized variable $x \in \mathbf{R}^m, m = m_1 + \cdots + m_M$, and

$$x = \left[x^{(1)T} \cdots x^{(M)T} \right]^T.$$

Note that each variable $X^{(i)}$ of the problem corresponds to part of the vectorized variable x. With this correspondence, we can convert the problem to the standard form (1) or (2), and vice versa.

The following example illustrates this vectorization process. We consider the problem

$$\begin{array}{ll} \text{minimize} & \mathbf{Tr}\, P \\ \text{subject to} & \begin{bmatrix} -A^TP-PA & -PB \\ -B^TP & R \end{bmatrix} \succ 0 \\ & P = P^T \succ 0 \\ & R \succ 0, R \text{ diagonal}, \ \mathbf{Tr}\, R = 1, \end{array} \tag{3}$$

where A, B are given matrices (*i.e.*, problem data), and the symmetric $P \in \mathbf{R}^{n \times n}$ and diagonal $R \in \mathbf{R}^{k \times k}$ are the optimization variables.

We vectorize P and R as

$$P = \sum_{i=1}^{n} \sum_{j=i}^{n} x_{ij}^{(1)} P_{ij}, \tag{4}$$

$$R = \sum_{i=1}^{k} x_i^{(2)} R_i, \tag{5}$$

where P_{ij} denotes an $n \times n$ zero matrix except the (i,j) and (j,i) entries are 1, R_i denotes a $k \times k$ zero matrix except the (i,i) entry is 1. Substituting (4) and (5) for P and R everywhere in the SDP (3), we obtain the problem of the form (2) in the optimization variable

$$x = \left[x_{11}^{(1)} \cdots x_{nn}^{(1)} \ x_1^{(2)} \cdots x_k^{(2)} \right]^T \in \mathbf{R}^{n(n+1)/2+k}.$$

1.3 Implications of the matrix structure

Clearly it is straightforward but inconvenient to convert an SDP or a max-det problem with matrix structure into the standard form (2) or (1). This conversion obscures the problem structure and the correspondence between the variables $X^{(i)}$ and the vectorized variable x, which makes it hard to interpret the results after the problem is solved.

Moreover, the problem structure can be exploited by interior-point methods for substantial efficiency gain (see [14] and [3] for examples). To illustrate the idea with a simple example, consider the operation

$$\mathcal{L}(P) = -A^TP - PA$$

that evaluates the $-A^TP - PA$ term in the first matrix inequality of (3). $\mathcal{L}(P)$ is an $\mathcal{O}(n^3)$ operation because it involves matrix multiplications of $n \times n$ matrices. However, if we vectorize P as shown in (4), then $\mathcal{L}(P)$ becomes

$$\mathcal{L}(P) = \sum_{i=1}^{n} \sum_{j=i}^{n} x_{ij}^{(1)} (-A^TP_{ij} - P_{ij}A),$$

which is an $\mathcal{O}(n^4)$ operation.

1.4 sdpsol and related work

In this paper, we describe the design and implementation of sdpsol, a parser/solver for SDPs and max-det problems with matrix structure. Problems can be specified in the sdpsol language in a form very close to their natural mathematical descriptions. As an example, the SDP (3) can be specified as

```
variable P(n,n) symmetric;
variable R(k,k) diagonal;
[-A'*P-P*A, -P*B;
 -B'*P,     R    ] > 0;
P > 0;
R > 0;  Tr(R) == 1;
minimize  objective = Tr(P);
```

There exist several similar tools that use a specification language to describe and solve certain mathematical programming problems. A well-known example is AMPL [6], which handles linear and integer programming problems.

LMITOOL [5] and the LMI Control Toolbox [7] provide convenient Matlab-interfaces that allow the user to specify SDPs with matrix structure (specifically the ones arising in control), using a different approach from the parser/solver described in this paper.

In §2, we describe the design of the specification language. A preliminary implementation of sdpsol (version beta), is described in §3. In §4, we give several examples to illustrate various features of sdpsol.

2 Language design

2.1 Matrix variables and affine expressions

The fundamental data structure of the sdpsol language is, as in Matlab [19], the matrix. The language provides a Matlab-like grammar, including various facilities to manipulate matrix expressions. For example, it provides commands that construct matrices, operations such as matrix multiplication and transpose, and matrix functions such as trace and inner-product. The language also supports assignments, *i.e.*, expressions can be assigned to internal variables that can later be used in the problem specification. Various algorithm parameters, such as tolerance or maximum number of iterations, can be initialized using assignments.

An important extension beyond Matlab is that the sdpsol language provides matrix variables, and, by extension, matrix expressions formed from matrix variables and matrix constants. Matrix variables of various dimensions and structure can be declared using variable declaration statements. For example, the statements

```
variable P(7,7) symmetric;
variable x(k,1), y;
```

declare a 7×7 symmetric variable P, a $k \times 1$ variable x and a scalar variable y.

Variables can be used to form *affine expressions*, *i.e.*, expressions that depend affinely on the optimization variables. For example, the expression (assuming P is a variable and A, D are constant square matrices)

```
A'*P + P*A + D
```

is an affine expression that depends affinely on P. Affine expressions can be used to construct LMI constraints and the objective function of SDPs and max-det problems.

As each affine expression is parsed, `sdpsol` keeps track of its dependence on the variables, adding the dependency information to an internal table for later reference, *e.g.*, by a solver.

2.2 Constraints and objective

Various types of (affine) constraints are supported by the language, including matrix inequalities, component-wise inequalities and equality constraints. Constraints are formed with affine expressions and relation operators, as in

```
A'*P + P*A + D < -1;
diag(P) .> 0;
Tr(P) == 1;
```

which specify the matrix inequality $A^TP + PA + D \prec -I$, the component-wise inequality $\mathbf{diag}\, P > 0$, and the equality $\mathbf{Tr}\, P = 1$.

The objective function is specified via an assignment, as in

```
maximize  objective = Tr(P);
```

which assigns $\mathbf{Tr}\, P$ to the internal variable `objective` and makes maximizing it the objective. A feasibility problem (*i.e.*, a problem that determines whether a set of constraints is feasible or not) is specified without an objective statement.

2.3 Duality

We associate with the max-det problem (1) the *dual* problem (see [16] for details):

$$\begin{array}{ll} \text{maximize} & \displaystyle\sum_{i=1}^{K} \left(\log\det W^{(i)} - \mathbf{Tr}\, G_0^{(i)} W^{(i)} + l_i\right) - \sum_{i=1}^{L} \mathbf{Tr}\, F_0^{(i)} Z^{(i)} + b^T z \\ \text{subject to} & \displaystyle\sum_{i=1}^{K} \mathbf{Tr}\, G_j^{(i)} W^{(i)} + \sum_{i=1}^{L} \mathbf{Tr}\, F_j^{(i)} Z^{(i)} + (A^T z)_j = c_j, \quad j = 1, \ldots, m \\ & W^{(i)} \succ 0, \quad i = 1, \ldots, K \\ & Z^{(i)} \succ 0, \quad i = 1, \ldots, L. \end{array} \tag{6}$$

where the *dual variables* are symmetric matrices $W^{(i)} \in \mathbf{R}^{l_i \times l_i}$ for $i = 1, \ldots, K$, symmetric matrices $Z^{(i)} \in \mathbf{R}^{n_i \times n_i}$ for $i = 1, \ldots, L$, and a vector $z \in \mathbf{R}^p$ (assuming $A \in \mathbf{R}^{p \times m}$ and $b \in \mathbf{R}^p$ in (1)). We will refer to (1) as the *primal* problem of (6). Note that (6) is itself an max-det problem.

Similarly, we can associate with the SDP (2) the dual:

$$\begin{array}{ll} \text{maximize} & -\sum_{i=1}^{L} \mathbf{Tr}\, F_0^{(i)} Z^{(i)} + b^T z \\ \text{subject to} & \sum_{i=1}^{L} \mathbf{Tr}\, F_j^{(i)} Z^{(i)} + (A^T z)_j = c_j, \quad j = 1, \ldots, m \\ & Z^{(i)} \succ 0, \quad i = 1, \ldots, L, \end{array} \tag{7}$$

which is also an SDP.

The dual problem has several practical uses, *e.g.*, the optimal dual variables give a sensitivity analysis of the primal problem. The most efficient SDP and max-det solvers are primal-dual, *i.e.*, they simultaneously solve the primal and dual problems. `sdpsol` automatically forms the dual from the primal problem specification. The dual problem is used by the solver, and also, if requested, to return to the user optimal dual variables.

There is a simple relation between the form (size and structure) of the primal constraints and the dual variables (and vice versa). Observe that for each constraint in (1) (or (2)), there is a dual variable of the same dimensions and structure associated with it. The variables associated with the primal LMI constraints are constrained to be positive definite in the dual problem, while the variables associated with the equality constraints are unconstrained.

Taking the SDP (3) as an example, `sdpsol` associates the dual variables

$$\begin{array}{l} Z^{(1)} = Z^{(1)T} \in \mathbf{R}^{(n+k)\times(n+k)} \\ Z^{(2)} = Z^{(2)T} \in \mathbf{R}^{n\times n} \\ Z^{(3)} \in \mathbf{R}^{k\times k}, \quad Z^{(3)} \text{ diagonal} \end{array}$$

with the three matrix inequalities respectively, and $z \in \mathbf{R}$ with the equality constraint. After some manipulation (which can be easily performed by `sdpsol`), we have the dual problem:

$$\begin{array}{ll} \text{maximize} & z \\ \text{subject to} & \mathbf{Tr} \begin{bmatrix} -A^T P_{ij} - P_{ij} A & -P_{ij} B \\ -B^T P_{ij} & 0 \end{bmatrix} Z^{(1)} + \mathbf{Tr}\, P_{ij} Z^{(2)} = \delta_{ij} \\ & \qquad \text{for all } i = 1, \ldots, n \text{ and } j = i, \ldots, n \\ & \mathbf{Tr} \begin{bmatrix} 0 & 0 \\ 0 & R_i \end{bmatrix} Z^{(1)} + \mathbf{Tr}\, R_i Z^{(3)} + z = 0, \quad i = 1, \ldots, k \\ & Z^{(1)} \succ 0, \quad Z^{(2)} \succ 0, \quad Z^{(3)} \succ 0, \end{array} \tag{8}$$

where P_{ij}, R_i are given by (4) and (5), $\delta_{ij} = 1$ if $i = j$, otherwise zero.

`sdpsol` provides an alternate form of constraint specifications, in which the user associates a name with each constraint, which is then used by `sdpsol` to refer to the dual variables. For example, the statements (assuming $P, A, D \in \mathbf{R}^{n\times n}$)

```
constraint lyap   A'*P + P*A + D < -1;
constraint equ    Tr(P) == 1;
```

specifies (and names) the constraints $A^TP+PA+D \prec -I$ and $\mathbf{Tr}\, P = 1$. sdpsol associates with these constraints the dual variables with names

$$\begin{array}{l} \mathit{lyap\_dual} \in \mathbf{R}^{n\times n}, \quad \mathit{lyap\_dual} \text{ symmetric} \\ \mathit{equ\_dual} \in \mathbf{R}. \end{array}$$

After the solution phase, sdpsol returns the optimal values of these dual variables.

3 Implementation

We have implemented a preliminary version of the parser/solver, sdpsol version beta. The parser is implemented using BISON and FLEX. Two solvers, SP [13] and MAXDET [18], are used to solve the resulting SDPs and max-det problems. Both solvers exploit only block-diagonal structure, so the current version of sdpsol is not particularly efficient.

Both SP and MAXDET are primal-dual methods that require a feasible starting primal and dual point. sdpsol uses the method described in [15, §6] to handle the feasibility phase of the problem, that is, sdpsol either finds a feasible starting point (to start the optimization phase), or proves that the problem is infeasible. If there is no objective in the problem specification, sdpsol simply solves the feasibility problem only.

A Matlab interface is provided by sdpsol to import problem data and export the results. sdpsol can also be invoked from within Matlab interactively.

3.1 Handling equality constraints

Neither SP nor MAXDET handles problems with equality constraints, so the current implementation of sdpsol eliminates them, uses SP or MAXDET to solve the resulting problem (which has no equality constraints), and finally reassembles the variables before reporting the results.

The equality constraints given in the problem specification can be collected and put into the form of $Ax = b$ ($A \in \mathbf{R}^{p\times m}, b \in \mathbf{R}^p$) as given in (1) and (2). We assume that the rank of A, say r, is strictly less than m (otherwise the problem has a trivial solution). The matrix A is often not full rank because sdpsol does not detect and remove redundancy in the equality constraints specified. For example, a 2×2 matrix variable X can be constrained to be symmetric by the constraint specification

```
X == X';
```

sdpsol interprets the above statement as the following four equality constraints

$$X_{11} = X_{11}, \quad X_{12} = X_{21}, \quad X_{21} = X_{12}, \quad X_{22} = X_{22}.$$

Evidently one constraint $X_{21} = X_{12}$ is necessary. This will show up as two zero rows in A, and two identical rows of A; in particular A will not be full rank.

Instead of keeping track of such situations, `sdpsol` builds up the matrix A paying no attention to rank. `sdpsol` then performs a complete orthogonal decomposition (see [9]) on A such that

$$AP = Q \begin{bmatrix} R_{11} & 0 \\ 0 & 0 \end{bmatrix} U^T, \tag{9}$$

where P is a column permutation matrix (for column pivoting), $R_{11} \in \mathbf{R}^{r \times r}$ is upper-triangular, $Q, U \in \mathbf{R}^{p \times p}$ are orthogonal matrices. All $x \in \mathbf{R}^m$ satisfying the equality constraints can be expressed as

$$x = PU \begin{bmatrix} R_{11}^{-1} & 0 \\ 0 & 0 \end{bmatrix} Q^T b + PU \begin{bmatrix} 0 \\ \tilde{x} \end{bmatrix} \triangleq x_p + PU \begin{bmatrix} 0 \\ \tilde{x} \end{bmatrix}$$

for all $\tilde{x} \in \mathbf{R}^{\tilde{m}}$ with $\tilde{m} = m - r$. Defining $B \in \mathbf{R}^{m \times \tilde{m}}$ by

$$PU \begin{bmatrix} 0 \\ I_{\tilde{m} \times \tilde{m}} \end{bmatrix} = \begin{bmatrix} 0 & B \end{bmatrix},$$

we can express the max-det problem (1) as

$$\begin{array}{ll} \text{minimize} & c^T(x_p + B\tilde{x}) + \sum_{i=1}^{K} \log\det \widetilde{G}^{(i)}(\tilde{x})^{-1} \\ \text{subject to} & G^{(i)}(x_p) + \sum_{j=1}^{\tilde{m}} \tilde{x}_j \left(\sum_{k=1}^{m} B_{kj} G_k^{(i)} \right) \succ 0, \quad i = 1, \ldots, K \\ & F^{(i)}(x_p) + \sum_{j=1}^{\tilde{m}} \tilde{x}_j \left(\sum_{k=1}^{m} B_{kj} F_k^{(i)} \right) \succ 0, \quad i = 1, \ldots, L, \end{array} \tag{10}$$

where $\tilde{x} \in \mathbf{R}^{\tilde{m}}$ is the optimization variables and B_{kj} denotes the (k, j) entry of B. Evidently, the above process reduces the optimization problem (1) in $\mathbf{R}^m$ with equality constraints to (10) in $\mathbf{R}^{\tilde{m}}$ without any equality constraint. Similar methods apply to the SDP (2).

We should add that eliminating variables destroys the matrix structure that could be exploited in a solver. A future version of `sdpsol`, would directly pass the equality constraints to an SDP/max-det solver that handles equality constraints.

4 Examples

In this section, we give several examples to illustrate various features of `sdpsol`.

4.1 Lyapunov inequality

As a very simple example, consider a linear system described by the differential equation

$$\dot{x}(t) = Ax(t) \tag{11}$$

where $A \in \mathbf{R}^{n \times n}$ is given. The linear system is stable (*i.e.*, all solutions of (11) converge to zero), if and only if there exists a symmetric, positive definite P such that the Lyapunov inequality

$$A^T P + PA < 0$$

is satisfied. The problem of finding such a P (if one exists) can be specified in the `sdpsol` language as follows

```
% Lyapunov inequality
variable P(n,n) symmetric;

A'*P + P*A < 0;
P > 0;
```

In the problem specification, an $n \times n$ symmetric variable P is declared. An affine expression $A^T P + PA$ is formed and is used to specify the Lyapunov inequality. Another LMI, $P > 0$, constrains P to be positive definite. Since no objective is given, `sdpsol` solves the feasibility problem that either finds a feasible P or shows that the problem is infeasible.

Note that this problem can be solved analytically. Indeed, we can solve the linear equation $A^T P + PA + I = 0$ for the matrix P, which is positive definite if and only if (11) is stable.

4.2 Popov analysis

Consider the mass-spring-damper system with a vibrating base shown in Figure 1. The masses are 1kg each, the dampers have the damper constant 0.5nt/m/s, and the springs are nonlinear: $F_i = \phi_i(x_i)$ where

$$0.7 \le \frac{\phi_i(a)}{a} \le 1.3, \quad i = 1, 2, 3.$$

The base vibration is described by

$$w_{\text{base}} = \frac{1}{1 + s/0.3}\, w,$$

where $\mathrm{RMS}(w) \le 1$ but is otherwise unknown. Our goal is to find an upper bound on $\mathrm{RMS}(z)$.

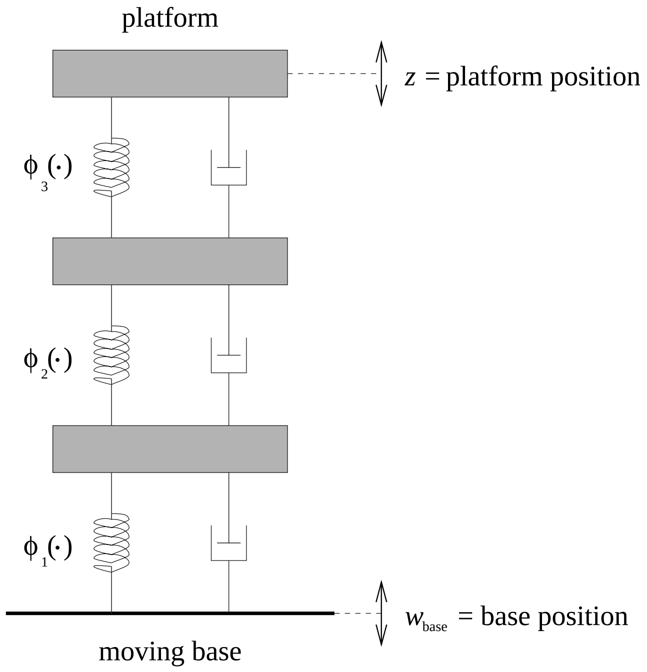

Fig. 1. The mass-spring-damper system

The mass-spring-damper system can be described by the Lur'e system (see [2, §8.1]) with 7 states and 3 nonlinearities

$$\begin{aligned} \dot{x} &= Ax + B_p p + B_w w, \\ z &= C_z x, \\ q &= C_q x, \\ p_i(t) &= \tilde{\phi}_i(q_i(t)), \quad i = 1, 2, 3, \end{aligned} \tag{12}$$

where $x \in \mathbf{R}^7$, $p \in \mathbf{R}^3$ and the functions $\tilde{\phi}_i$ satisfy the $[0, 1]$ sector condition, *i.e.*,

$$0 \le q_i \tilde{\phi}_i(q_i) \le q_i^2$$

for $i = 1, 2, 3$. One method to find an upper bound is to find a γ^2 and a Lyapunov function of the form

$$V(x) = x^T P x + 2 \sum_{i=1}^{3} \lambda_i \int_0^{C_{q_i} x} \tilde{\phi}_i(\sigma) d\sigma \tag{13}$$

where C_{q_i} denotes the ith row of C_q, such that

$$\frac{d}{dt} V(x) \le \gamma^2 w^T w - z^T z$$

for all w, z satisfying (12). The square root of γ^2 yields an upper bound on $\mathrm{RMS}(z)$.

The problem of finding γ^2 and the Lyapunov function (13) can be further relaxed using the $\mathcal{S}$-procedure to the following SDP (see [2, §8.1.4]):

$$\begin{array}{l} \text{minimize } \gamma^2 \\ \text{subject to } P > 0, \;\; L \geq 0, \;\; T \geq 0 \\ \qquad \begin{bmatrix} A^TP+PA+C_z^TC_z & PB_p+A^TC_q^TL+C_q^TT & PB_w \\ B_p^TP+LC_qA+TC_q & LC_qB_p+B_p^TC_q^TL-2T & LC_qB_w \\ B_w^TP & B_w^TC_q^TL & -\gamma^2 \end{bmatrix} \leq 0, \end{array} \qquad (14)$$

where $P = P^T \in \mathbf{R}^{7\times 7}$, $L, T \in \mathbf{R}^{3\times 3}$ diagonal, and $\gamma^2 \in \mathbf{R}_+$ are the optimization variables. The optimal γ^2 of (14) is an upper bound of $\mathrm{RMS}(z)$ if there exists P, L, T that satisfy the constraints.

The SDP (14) can be described using the `sdpsol` language as follows

```
% Popov analysis of a mass-spring-damper system
variable P(7,7) symmetric;
variable L(3,3), T(3,3) diagonal;
variable gamma_sqr;

P > 0;
L > 0;
T > 0;
[A'*P+P*A+Cz'*Cz,  P*Bp+A'*Cq'*L+Cq'*T,  P*Bw;
 Bp'*P+L*Cq*A+T*Cq,L*Cq*Bp+Bp'*Cq'*L-2*T,L*Cq*Bw;
 Bw'*P,            Bw'*Cq'*L,            -gamma_sqr]<0;

minimize RMS_bound_sqr = gamma_sqr;
```

Again, the specification in the `sdpsol` language is very close to the mathematical description (14). The objective of the problem is to minimize `RMS_bound_sqr`, the square of the RMS bound, which is equal to the variable `gamma_sqr`.

Unlike the previous example, this problem of finding an upperbound on $\mathrm{RMS}(z)$ has *no* analytical solution.

4.3 *D*-optimal experiment design

Consider the problem of estimating a vector x from a measurement $y = Ax + w$, where $w \sim N(0, I)$ is the measurement noise. The error covariance of the minimum-variance estimator is equal to $A^\dagger(A^\dagger)^T = (A^TA)^{-1}$. We suppose that the rows of the matrix $A = [a_1 \ \ldots \ a_q]^T$ can be chosen among M possible test vectors $v^{(i)} \in \mathbf{R}^p$, $i = 1, \ldots, M$:

$$a_i \in \{v^{(1)}, \ldots, v^{(M)}\}, \;\; i = 1, \ldots, q.$$

The goal of experiment design is to choose the vectors a_i so that the determinant of the error covariance $(A^T A)^{-1}$ is minimized. This is called the D-optimal experiment design.

We can write $A^T A = \sum_{i=1}^{M} \lambda_i v^{(i)} {v^{(i)}}^T$, where λ_i is the fraction of rows a_k equal to the vector $v^{(i)}$. We ignore the fact that the numbers λ_i are integer multiples of $1/q$, and instead treat them as continuous variables, which is justified in practice when q is large.

In D-optimal experiment design, λ_i are chosen to minimize the determinant of the error covariance matrix, which can be cast as the max-det problem

$$\begin{array}{ll} \text{minimize} & \log\det \left(\sum_{i=1}^{M} \lambda_i v^{(i)} {v^{(i)}}^T \right)^{-1} \\ \text{subject to} & \lambda_i \geq 0,\ i = 1, \ldots, M \\ & \sum_{i=1}^{M} \lambda_i = 1. \end{array} \tag{15}$$

This problem can be described in the sdpsol language as

```
% D-optimal experiment design
variable lambda(M,1);

lambda .> 0;
sum(lambda) == 1;

Cov_inv = zeros(p,p);
for i=1:M;
  Cov_inv = Cov_inv + lambda(i,1)*v(:,i)*v(:,i)';
end;

minimize log_det_Cov = -logdet(Cov_inv);
```

In the specification, an M-vector `lambda` is declared to be the optimization variable. The (component-wise) inequality constraint specifies that each entry of `lambda` is positive, and the equality constraint says the summation of all entries of `lambda` is 1.

A for-loop is used to construct `Cov_inv`, the inverse of the covariance matrix, to be $\sum_{i=1}^{M} \lambda_i v^{(i)} {v^{(i)}}^T$. The objective of the optimization problem is to minimize the log-determinant of the inverse of `Cov_inv`. An implicit LMI constraint, `Cov_inv` > 0 is added to the problem as soon as the objective is specified. This LMI corresponds to the $G(x) > 0$ term in (1), and it ensures that the log-determinant term is well-defined.

5 Summary

Using a parser/solver (such as `sdpsol` described in this paper) for SDPs and max-det problems has the following advantages:

– Problems with matrix structure can be conveniently specified and solved.
– Problem structure can be fully exploited by sophisticated solvers.

Our first implementation, `sdpsol`, has the first advantage but does not exploit the problem structure.

Acknowledgments

This research supported in part by AFOSR (under F49620-95-1-0318), NSF (under ECS-9222391 and EEC-9420565), and MURI (under F49620-95-1-0525). The authors would also like to thank Lieven Vandenberghe, Laurent El Ghaoui, Paul Dankoski and Michael Grant for very helpful discussions, comments and suggestions.

References

1. F. Alizadeh. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM Journal on Optimization*, 5(1):13–51, February 1995.
2. S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*, volume 15 of *Studies in Applied Mathematics*. SIAM, Philadelphia, PA, June 1994.
3. S. Boyd, L. Vandenberghe, and M. Grant. Efficient convex optimization for engineering design. In *Proceedings IFAC Symposium on Robust Control Design*, pages 14–23, September 1994.
4. N. Brixius, F. A. Potra, and R. Sheng. Solving semidefinite programs with Mathematica. Technical Report 97/1996, Department of Mathematics, University of Iowa, 1996.
5. L. El Ghaoui, R. Nikoukha, and F. Delebecque. LMITOOL*: a front-end for LMI optimization, user's guide*, 1995. Available via anonymous ftp to `ftp.ensta.fr` under `/pub/elghaoui/lmitool`.
6. B. K. R. Fourer, D. Gay. AMPL*–a modeling language for mathematical programming.* Scientific Press, San Francisco, 1993.
7. P. Gahinet, A. Nemirovskii, A. J. Laub, and M. Chilali. The LMI Control Toolbox. In *Proc. IEEE Conf. on Decision and Control*, pages 2083–2041, December 1994.
8. K. Fujisawa and M. Kojima. SDPA (semidefinite programming algorithm) user's manual. Technical Report B-308, Department of Mathematical and Computing Sciences. Tokyo Institute of Technology, 1995.
9. G. Golub and C. Van Loan. *Matrix Computations.* Johns Hopkins Univ. Press, Baltimore, second edition, 1989.
10. A. S. Lewis and M. L. Overton. Eigenvalue optimization. *Acta Numerica*, pages 149–190, 1996.
11. Yu. Nesterov and A. Nemirovsky. *Interior-point polynomial methods in convex programming*, volume 13 of *Studies in Applied Mathematics.* SIAM, Philadelphia, PA, 1994.
12. K. C. Toh, M. J. Todd, and R. H. Tütüncü. SDPT3*: a* MATLAB *software package for semidefinite programming*, 1996.
    Available at `http://www.math.nus.sg/~mattohkc/index.html`.

13. L. Vandenberghe and S. Boyd. SP*: Software for Semidefinite Programming. User's Guide, Beta Version.* K.U. Leuven and Stanford University, Oct. 1994.
14. L. Vandenberghe and S. Boyd. A primal-dual potential reduction method for problems involving matrix inequalities. *Mathematical Programming*, 69(1):205–236, July 1995.
15. L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, March 1996.
16. L. Vandenberghe, S. Boyd, and S.-P. Wu. Determinant maximization with linear matrix inequality constraints. *Submitted to SIMAX*, February 1996.
17. S.-P. Wu and S. Boyd. sdpsol: *A Parser/Solver for Semidefinite Programming and Determinant Maximization Problems with Matrix Structure. User's Guide, Version Beta.* Stanford University, June 1996.
18. S.-P. Wu, L. Vandenberghe, and S. Boyd. MAXDET*: Software for Determinant Maximization Problems. User's Guide, Alpha Version.* Stanford University, Apr. 1996.
19. MATLAB*: High-performance Numeric Computation and Visualization Software.* Version 4.1, MathWorks Inc., Natlick, MA, 1993.