Proceedings of the 1996 IEEE International Symposium
cn Computer-Aided Control System Design
Dearborn, MI • September 15-18, 1996

Design and Implementation of a Parser/Solver for SDPs with Matrix Structure

Shao-Po Wu and Stephen Boyd

Information Systems Laboratory
Department of Electrical Engineering, Stanford University
Stanford, CA 94305
clive@isl.stanford.edu, boyd@isl.stanford.edu

Abstract

A wide variety of analysis and design problems arising in control, communication and information theory, statistics, computational geometry and many other fields can be expressed as *semidefinite programming* problems (SDPs) or *determinant maximization* problems (maxdet-problems). In engineering applications these problems usually have matrix structure, *i.e.*, the optimization variables are matrices. Recent interior-point methods can exploit such structure to gain huge efficiency.

In this paper, we describe the design and implementation of a parser/solver for SDPs and maxdet-problems with matrix structure. The parser/solver parses a problem specification close to its natural mathematical description, solves the compiled problem efficiently, and returns the results in a convenient form.

1. Introduction

We consider the design and implementation of a parser/solver for SDPs and maxdet-problems with matrix structure (see below). This parser/solver serves as a convenient and efficient tool, which reduces the complexity of specifying and solving the problems.

1.1. Maxdet-problems and SDPs

By *determinant maximization* (maxdet) problems we mean optimization problems of the form:

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

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:

$$G^{(i)}(x) = G_0^{(i)} + x_1 G_1^{(i)} + \cdots + x_m G_m^{(i)},\ i = 1,\dots,K$$
$$F^{(i)}(x) = F_0^{(i)} + x_1 F_1^{(i)} + \cdots + x_m F_m^{(i)},\ i = 1,\dots,L$$

where $G_j^{(i)}$ and $F_j^{(i)}$ are symmetric for $j = 0,\dots,m$. The inequality signs in (1) denote matrix inequalities. We call $G^{(i)}(x) > 0$ and $F^{(i)}(x) > 0$ *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 maxdet-problem (1) reduces to the *semidefinite programming* (SDP) problem:

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

The maxdet-problem (1) and the SDP (2) are convex optimization problems. Indeed, LMI constraints can represent many common convex constraints, including linear inequalities, convex quadratic inequalities, matrix norm and eigenvalue constraints. Conversely, many common convex optimization problems arising in combinatorial optimization, control, statistics, computational geometry, information and communication theory can be expressed as SDPs and maxdet-problems. See Alizadeh[1], Boyd, El Ghaoui, Feron and Balakrishnan[2], Lewis and Overton[6], Nesterov and Nemirovsky[7, §6.4], Vandenberghe and Boyd[11] and Vandenberghe, Boyd and Wu[12] for many examples. SDPs and maxdet-problems can be solved very efficiently, both in worst-case complexity theory and in practice, using interior-point methods (see [11] and [12]).

1.2. Matrix structure

In many SDPs and maxdet-problems, especially those arising in control, the optimization variables are matrices of various dimensions and structure, *e.g.*, row or column vectors, symmetric or diagonal matrices. In general, the variables of an SDP or a maxdet-problem 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 or diagonal). These variables can be vectorized and put into a single vector variable x as given in (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

$$x = \begin{bmatrix} x^{(1)T} & \cdots & x^{(M)T} \end{bmatrix}^T.$$

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

The following example illustrates how to convert an SDP with matrix variables into the standard form (2). Consider the problem:

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

where A, B are given matrices, 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

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

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

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

$$x = \begin{bmatrix} x_{11}^{(1)} & \cdots & x_{nn}^{(1)} & x_1^{(2)} & \cdots & x_k^{(2)} \end{bmatrix}^T \in \mathbf{R}^{n(n+1)/2+k}.$$

1.3. Implications of the matrix structure

Clearly it is straightforward but inconvenient to convert an SDP or a maxdet-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 to gain huge efficiency (see [10] and [3]). To illustrate the idea, consider the operation

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

that evaluates the $-A^T P - PA$ term in the first LMI 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

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

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 maxdet-problems with matrix structure. As an example, the SDP (3) can be specified in the specification language as shown in sdpsol specification 1

sdpsol specification 1

```
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 [8], which handles linear and integer programming problems.

LMITOOL [4] and the LMI Control Toolbox [5], provide convenient Matlab-interfaces that allow to specify SDPs with matrix structure, 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 the parser/solver, sdpsol version beta, is described in §3.

Three examples are given in §4 to illustrate various features of sdpsol. In §5, we give some future extensions of the current design and implementation.

2. Language design

The fundamental data structure of the specification language is matrix, since the optimization problems involve LMI constraints and have matrix structure. The language provides a Matlab-like grammar, including various facilities to manipulate matrix expressions. For example, the language provides commands that construct matrices, operations involving matrices such as matrix multiplication and transpose, and matrix functions such as trace and inner-product.

An important extension from Matlab is that the language provides matrix variables, *i.e.*, matrix expressions without specific values that serve as the optimization variables. Variables of various sizes 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 given 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 objective functions of SDPs and maxdet-problems.

Various types of linear constraints are supported by the language, including matrix inequalities, component-wise inequalities and equality constraints. Constraints are formed with expressions and relation operators, for example, the statements

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

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

The language supports assignments, *i.e.*, expressions can be assigned to internal variables that can later be used in the problem specification. Assignments can also be used to initialize various algorithm parameters, such as tolerance or maximum number of iterations.

The objective is specified via an assignment, for example, the statement

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

assigns $\mathbf{Tr}\, P$ to the internal variable objective and makes maximizing it the objective.

3. Implementation

We have implemented a preliminary version of the parser/solver, sdpsol version beta. The parser of sdpsol is implemented using BISON and FLEX. Two solvers, SP [9] and MAXDET [13], are used to solve SDPs and maxdet-problems. Both solvers exploit only block-diagonal structure, hence sdpsol is not particularly efficient. Of course the user can write very efficient, dedicated programs for their problems which outperforms sdpsol. However, for the cases when the coding time is significantly longer than the actual run-time, sdpsol wins the tradeoff because it takes almost no time to code using the sdpsol language. Nevertheless, we hope to improve the efficiency of sdpsol in the future; in particular, with an efficient solver that exploits sparse structure.

sdpsol uses the method described in [11, §6] to handle the feasibility phase of the problem, that is, sdpsol either finds a feasible solution 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.

4. Examples

In this section, we give three 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

$$\dot{x}(t) = Ax(t) \tag{6}$$

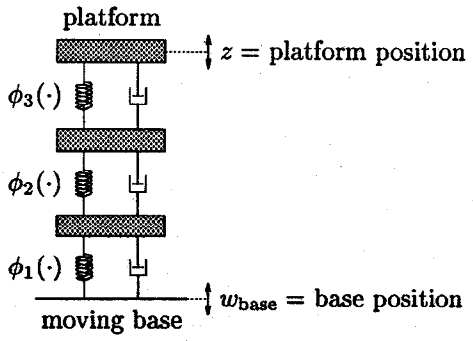

Figure 1: The mass-spring-damper system

where $A \in \mathbf{R}^{n\times n}$ is given. The linear system is stable (*i.e.*, all solutions of (6) converge to zero), if and only if there exists a symmetric, positive definite P such that the Lyapunov inequality

$$A^TP + PA < 0$$

is satisfied. The problem of finding such a P (if one exists) can be specified in the sdpsol language as given in sdpsol specification 2.

sdpsol specification 2

```
% 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 $A^TP + PA$ 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 $A^TP+PA+I = 0$ for the matrix P, which is positive definite if and only if (6) 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 of $\mathrm{RMS}(z)$.

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

$$\begin{aligned} \dot{x} &= Ax + B_p p + B_w w, \\ z &= C_z x, \\ q &= C_q x, \\ p_i(t) &= \tilde{\phi}_i(q_i(t)), \quad i = 1,2,3, \end{aligned} \tag{7}$$

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

$$V(x) = x^TPx + 2\sum_{i=1}^{3}\lambda_i \int_0^{C_{q_i}x} \tilde{\phi}_i(\sigma)d\sigma \tag{8}$$

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

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

for all w, z satisfying (7). The square root of γ^2 is an upperbound of the $\mathbf{L_2}$ gain of (7), hence it is also an upperbound of $\mathrm{RMS}(z)$ because $\mathrm{RMS}(w) \le 1$.

The problem of finding γ^2 and the Lyapunov function (8) can be further relaxed using the $\mathcal{S}$-procedure to the following SDP (see [2, §8.1.4]):

$$\begin{array}{ll} \text{minimize} & \gamma^2 \\ \text{subject to} & P > 0, \;\; L \ge 0, \;\; T \ge 0 \end{array} \tag{9}$$
$$\begin{bmatrix} A^TP+PA+C_z^TC_z & PB_p+A^TC_q^TL+C_q^TT & PB_w \\ B_p^TP+LC_qA+TC_q & LC_qB_p+B_p^TC_q^TL-2T & LC_qB_w \\ B_w^TP & B_w^TC_q^TL & -\gamma^2 \end{bmatrix} \le 0,$$

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 (9) is an upper bound of $\mathrm{RMS}(z)$ if there exists P, L, T that satisfy the constraints.

The SDP (9) can be described using the sdpsol language as shown in sdpsol specification 3.

Again, the specification in the sdpsol language is very close to the mathematical description (9). 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.

sdpsol specification 3

```
% 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;
```

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 $A^{\dagger}(A^{\dagger})^T = (A^T A)^{-1}$. We suppose that the rows of the matrix $A = [a_1 \ \dots \ a_q]^T$ can be chosen among M possible test vectors $v^{(i)} \in \mathbf{R}^p$, $i = 1, \dots, M$:

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

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.

The D-optimal design problem can be cast as the maxdet-problem:

$$\begin{array}{ll} \text{minimize} & \log\det\left(\sum_{i=1}^{M} \lambda_i v^{(i)} v^{(i)T}\right)^{-1} \\ \text{subject to} & \lambda_i \geq 0, \ i = 1, \dots, M \\ & \sum_{i=1}^{M} \lambda_i = 1. \end{array} \tag{10}$$

This problem can be described in the sdpsol language as shown in sdpsol specification 4

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.

sdpsol specification 4

```
% 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);
```

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. Extensions

Using a parser/solver for SDPs and maxdet-problems has the following advantages:

- problems with matrix structure can be conveniently specified and solved, and
- the problem structure can be fully exploited by sophisticated solvers.

Our rudimentary implementation, sdpsol, has the first advantage but does not fully exploit the problem structure yet. Nevertheless, sdpsol reduces the complexity of solving engineering design problems involving SDPs and maxdet-problems, or at least, make more effective use of the engineer's time.

The following features are considered to be implemented in the future design:

- a data structure that handles sparse matrices,
- a solver that exploits sparse structure,
- a parser/solver that handles non-strict LMIs, and
- a parser/solver that handles other extensions of the SDP, such as the generalized eigenvalue minimization problem.

Acknowledgments
The authors would like to thank Lieven Vandenberghe, Laurent El Ghaoui, Paul Dankoski and Michael Grant for very useful 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, Sept. 1994.

[4] L. El Ghaoui. LMITOOL: a package for LMI optimization. In *Proc. IEEE Conf. on Decision and Control*, pages 3096–3101, Dec. 1995.

[5] 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, Dec. 1994.

[6] A. S. Lewis and M. O. Overton. Eigenvalue optimization. *Acta Numerica*, pages 149–190, 1996.

[7] Y. Nesterov and A. Nemirovsky. *Interior-point polynomial methods in convex programming*, volume 13 of *Studies in Applied Mathematics*. SIAM, Philadelphia, PA, 1994.

[8] B. K. R. Fourer, D. Gay. AMPL–*a modeling language for mathematical programming*. Scientific Press, San Francisco, 1993.

[9] L. Vandenberghe and S. Boyd. SP*: Software for Semidefinite Programming. User's Guide, Beta Version*. K.U. Leuven and Stanford University, Oct. 1994.

[10] L. Vandenberghe and S. Boyd. A primal-dual potential reduction method for problems involving matrix inequalities. *Mathematical Programming*, 69(1):205–236, July 1995.

[11] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, Mar. 1996.

[12] L. Vandenberghe, S. Boyd, and S.-P. Wu. Determinant maximization with linear matrix inequality constraints. *submitted to SIMAX*, Feb. 1996.

[13] S.-P. Wu, L. Vandenberghe, and S. Boyd. MAXDET*: Software for Determinant Maximization Problems. User's Guide, Alpha Version*. Stanford University, Apr. 1996.