MAXDET

Software for Determinant Maximization Problems

User's Guide

Alpha Version May 24, 1996

Shao-Po Wu `clive@isl.stanford.edu`
Lieven Vandenberghe `vandenbe@isl.stanford.edu`
Stephen Boyd `boyd@isl.stanford.edu`

Information Systems Laboratory
Electrical Engineering Department
Stanford University

1 Introduction

Purpose

This package contains software for solving the optimization problem

$$\begin{array}{ll} \text{minimize} & c^T x + \log\det G(x)^{-1} \\ \text{subject to} & G(x) > 0 \\ & F(x) \geq 0, \end{array} \tag{1}$$

where the optimization variable is the vector $x \in \mathbf{R}^m$. The functions $G : \mathbf{R}^m \to \mathbf{R}^{l\times l}$ and $F : \mathbf{R}^m \to \mathbf{R}^{n\times n}$ are affine:

$$\begin{aligned} G(x) &= G_0 + x_1 G_1 + \cdots + x_m G_m, \\ F(x) &= F_0 + x_1 F_1 + \cdots + x_m F_m, \end{aligned}$$

where $G_i = G_i^T$ and $F_i = F_i^T$ for $i = 0, \ldots, m$. The inequality signs in (1) denote matrix inequalities, *i.e.*, $G(x)$ is positive definite and $F(x)$ is positive semi-definite. We will refer to (1) as a maxdet-program.

For background information and applications, see the manuscript [VBW96], which is included as part of the software distribution. We follow the notation of [VBW96], with one addition: the matrices F_i and G_i are block diagonal: F_i has L diagonal blocks, with dimensions $n_1, \ldots, n_L$ (hence, $n = n_1 + \cdots + n_L$); G_i has K diagonal blocks, with dimensions $l_1, \ldots, l_K$ ($l = l_1 + \cdots + l_K$).

Overview

The package is available via anonymous ftp at `isl.stanford.edu` in `pub/boyd/maxdet`. It contains:

- C-source: `maxdet_src.c` and `maxdet.h` contain a C-function `maxdet` that solves the maxdet-program. See §2 for further details.
- Matlab interface: Compiled mex-files `maxdet.mex4` (for Sun4, *e.g.*, Sun SparcStation), `maxdet.mexds` (for DECstation) and `maxdet.mexhp7` (for HP 9000/700 series). These mex-files allow the user to call the C-function `maxdet` from within matlab as if it is a built-in matlab function `maxdet.m`. See §3 for details.
- Matlab routine: `phase1.m`. See §3.
- Matlab examples: we provide matlab scripts that solve various maxdet-programs as examples: (some of them are described in the manuscript [VBW96]): positive definite matrix completion with partially specified inverse (`mat_completion.m`), minimum-volume ellipsoid containing points (`minVe_pts.m`), minimum-volume ellipsoid containing ellipsoids (`minVe_ell.m`), D-optimal experiment design (`exp_design.m`), and the educational testing problem (`etp.m`). See §4.
- The manuscript [VBW96], in compressed postscript format (`maxdet.ps.Z`).

How to use the package

You can use the package in different ways.

- **The easiest method** is to solve maxdet-programs indirectly using `sdpsol`. `sdpsol` is a parser/solver that simplifies the specification and solution of optimization problems involving matrices such as maxdet-programs and semidefinite programs [BW95].
- The easiest method to solve maxdet-programs directly (provided you have matlab) is to use the executable mex-files for Sun4, DECstation, or HP 9000/700 series workstation. If your machine is one of these types, all you have to do is retrieve the appropriate mex-file, *i.e.*, `maxdet.mex4` (for Sun4), `maxdet.mexds` (for DECstation), or `maxdet.hp7` (for HP 9000/700), and copy it to the appropriate directory on your machine. See §3.

- Create a mex-file for use in matlab by compiling `maxdet.c` using `cmex`, and linking it with LAPACK and BLAS (see §2 for more details).
- Write a C-program that calls the function `maxdet` in `maxdet_src.c`, and link the code with LAPACK and BLAS. You should be able to do this on any machine with an ansi-C compiler; see §2 for more details.

Caveat

Our goal is to provide a program that is easy to use, reasonably efficient, and useful for small to medium-sized problems (say, up to a few hundred variables). If your problem is large scale (say, several hundreds or thousands of variables), you probably should use an implementation that exploits problem structure. The only structure `maxdet` exploits is the block-diagonal structure of F and G.

2 C routine

The main routine is a C-function `maxdet`:

```
int maxdet(int m, int L, double *F, int *F_blkszs, int K, double *G, int *G_blkszs,
           double *c, double *x, double *Z, double *W,
           double *ul, double *hist, double gamma, double abstol, double reltol,
           int *NTiters, double *work, int lwork, int *iwork, int *info)
```

Purpose

`maxdet` solves the maxdet-program

$$\begin{array}{ll} \text{minimize} & c^T x + \log\det G(x)^{-1} \\ \text{subject to} & G_0 + x_1 G_1 + \cdots + x_m G_m > 0 \\ & F_0 + x_1 F_1 + \cdots + x_m F_m \geq 0 \end{array}$$

and its dual

$$\begin{array}{ll} \text{maximize} & \log\det W - \mathbf{Tr} G_0 W - \mathbf{Tr} F_0 Z + l \\ \text{subject to} & \mathbf{Tr} G_i W + \mathbf{Tr} F_i Z = c_i, \quad i = 1, ..., m, \\ & W = W^T > 0, \quad Z = Z^T \geq 0, \end{array}$$

given a strictly primal feasible initial point x, and, optionally, a strictly dual feasible initial point (Z, W).

Storage convention

Warning: different storage conventions are used in the matlab and C functions.
The matrices F_i and Z have L diagonal blocks:

$$F_i = \begin{bmatrix} F_i^{(1)} & 0 & \cdots & 0 \\ 0 & F_i^{(2)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & F_i^{(L)} \end{bmatrix} \qquad Z = \begin{bmatrix} Z^{(1)} & 0 & \cdots & 0 \\ 0 & Z^{(2)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & Z^{(L)} \end{bmatrix}$$

with block dimensions $n_1, \ldots, n_L$. The dimensions n_i are given in the array `F_blkszs` of length L: `F_blkszs` $= \{n_1, \ldots, n_L\}$. We use packed storage for every diagonal block, *i.e.*, we store the lower triangular part columnwise.

For example, if `F_blkszs` is {`2, 3, 1`}, the matrix

$$Z = \begin{bmatrix} 1 & 2 & 0 & 0 & 0 & 0 \\ 2 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 5 & 6 & 0 \\ 0 & 0 & 5 & 7 & 8 & 0 \\ 0 & 0 & 6 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 \end{bmatrix}$$

would be stored as

{ `1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0` } .

Similar storage convention is used for G_i and W, with K diagonal blocks and the block dimensions stored in `G_blkszs` $= \{l_1, \ldots, l_K\}$.

Arguments

1. `m` (integer): The number of variables. $m \geq 1$.

2. `L` (integer): The number of diagonal blocks in F_i and Z. $L \geq 1$.

3. `F` (pointer to double): An array of length $(m+1)\sum_{i=1}^{L} n_i(n_i+1)/2$, containing the matrices F_i, $i = 0, \ldots m$, using the storage scheme described above. The matrices $\mathbf{diag}(F_i, G_i)$, $i = 1, \ldots, m$, must be linearly independent.

4. `F_blkszs` (pointer to integer): An array of L integers `F_blkszs` $= \{n_1, \ldots, n_L\}$. n_j is the dimension of the jth block of F and Z.

5. `K` (integer): The number of diagonal blocks in G_i and W. $K \geq 1$.

6. `G` (pointer to double): An array of length $(m+1)\sum_{i=1}^{K} l_i(l_i+1)/2$, containing the matrices G_i, $i = 0, \ldots m$, using the storage scheme described above. The matrices $\mathbf{diag}(F_i, G_i)$, $i = 1, \ldots, m$, must be linearly independent.

7. `G_blkszs` (pointer to integer): An array of K integers `G_blkszs` $= \{l_1, \ldots, l_K\}$. l_j is the dimension of the jth block of G and W.

8. `c` (pointer to double): Array of length m. Vector that specifies the linear part of the primal objective.

9. `x` (pointer to double): Array of length m. On entry, a strictly primal feasible point, *i.e.*, we must have $F_0 + \sum_{i=1}^{m} x_i F_i > 0$ and $G_0 + \sum_{i=1}^{m} x_i G_i > 0$. On exit, the last iterate for x. The meaning of the last iterate depends on the stopping criterion.

10. `Z` (pointer to double): Array of length $\sum_{i=1}^{L} n_i(n_i+1)/2$, using the storage scheme described above. On entry, if Z and W (see below) are strictly dual feasible, *i.e.*, satisfy $\mathbf{Tr} Z F_i + \mathbf{Tr} W G_i = c_i$, $i = 1, \ldots, m$, $Z > 0$ and $W > 0$, Z is used in the preliminary phase; otherwise, Z is not used at all. On exit, Z is the last dual iterate. The meaning of the last dual iterate depends on the stopping criterion.

11. `W` (pointer to double): Array of length $\sum_{i=1}^{K} l_i(l_i+1)/2$, using the storage scheme described above. On entry, if Z and W are strictly dual feasible, W is used in the preliminary phase; otherwise, W is not used at all. On exit, W is the last dual iterate. The meaning of the last dual iterate depends on the stopping criterion.

12. `ul` (pointer to double): Array of length two. On exit, `ul[0]` is the primal objective value $c^T x + \log\det G(x)^{-1}$ (*i.e.*, an upper bound on the optimal value); `ul[1]` is the dual objective value $\log\det W - \mathbf{Tr} G_0 W - \mathbf{Tr} F_0 Z + l$ (*i.e.*, a lower bound on the optimal value).

13. `hist` (pointer to double): Array of length three times the maximum number of Newton iterations (see `NTiters` below). On exit, `hist[3*i-3]`, `hist[3*i-2]` and `hist[3*i-1]` contain the primal objective value, the duality gap and the number of Newton iterations at the ith *outer* iteration, respectively.

14. `gamma` (double). The algorithm parameter γ, which affects convergence rate (see [VBW96, §5 – §6]). `gamma` must be positive. Typical value: 10–1000.

15. `abstol` (double). Absolute tolerance. For the precise interpretation of `abstol`, see the section on convergence criteria below. If `abstol` $<$ `1e-10`, a value of `1e-10` is used instead.

16. `reltol` (double). Relative tolerance. For the precise interpretation of `reltol`, see the section on convergence criteria below.

17. `NTiters` (pointer to integer). On entry, the maximum number of *total* Newton (and predictor) iterations. If `*NTiters` < 1, it is set to 100. On exit, the number of total Newton (and predictor) iterations.

18. `work` (pointer to double). Work space. Array of length `lwork`.

19. `lwork` (integer). The size of the array `work`. `lwork` must be at least equal to

$$(2m+5)N_{\mathrm{pd}} + 2(n+l) + \max\{m + N_{\mathrm{pd}}N_{\mathrm{B}}, 3(b_{\max} + N_{\max}), 3(m + m^2 + N_{\max})\}$$

with

$$F_{\mathrm{pd}} = \sum_{i=1}^{L} n_i(n_i+1)/2, \quad G_{\mathrm{pd}} = \sum_{i=1}^{K} l_i(l_i+1)/2, \quad N_{\mathrm{pd}} = F_{\mathrm{pd}} + G_{\mathrm{pd}},$$
$$N_{\max} = \max\{F_{\mathrm{pd}}, G_{\mathrm{pd}}\}, \quad n_{\max} = \max_i n_i, \quad l_{\max} = \max_i l_i, \quad b_{\max} = \max\{n_{\max}, l_{\max}\},$$

and $N_{\mathrm{B}} \geq 1$. For best performance, N_{B} should be at least equal to the optimal block size required by LAPACK routine `dgels` (see [ABB+92, §6.2]).

20. `iwork` (pointer to integer). Work space. Array of length m.

21. `info` (pointer to integer). On exit, `*info` $= 1$ if the maximum number of Newton iterations is exceeded; `*info` $= 2$ if the absolute accuracy has been reached; `*info` $= 3$ if the relative accuracy has been reached; Negative values of `*info` indicate errors: `*info` $= -i$ means that the ith argument has an illegal value; `*info` $= -23$ means `diag`(F_i, G_i) are not linearly independent. `*info` $= -24$ stands for all other errors.

`maxdet` returns 0 for normal exit, and 1 if an error occurs.

Convergence criteria

On exit, `maxdet` returns an *interval* in which the optimal value has been located. The two-vector `ul` gives an upper bound `ul[0]` and a lower bound `ul[1]` for the optimal value. The quantity `ul[0] - ul[1]` is called the *duality gap* (See [VBW96] for the underlying theory).

The program exits normally under four possible conditions:

- *The maximum number of Newton iterations is exceeded.*
- *The absolute tolerance is reached:*

$$\texttt{ul[0]} - \texttt{ul[1]} \leq \texttt{abstol}.$$

- *The relative tolerance is reached.* The primal objective `ul[0]` and the dual objective `ul[1]` are both positive and

$$\texttt{ul[0]} - \texttt{ul[1]} \leq \texttt{reltol*ul[1]},$$

or the primal and dual objective are both negative and

$$\texttt{ul[0]} - \texttt{ul[1]} \leq -\texttt{reltol*ul[0]}.$$

Caveats

- The code returns with an error message if the matrices $\mathtt{diag}(F_i, G_i)$, $i = 1, \ldots, m$, are linearly dependent. If the matrices are dependent, the problem is unbounded below or can be reduced to one with fewer variables.
- The code requires a *strictly* primal feasible initial point, which means that the standard trick of writing equality constraints $Ax = b$ as the two vector inequalities $Ax - b \geq 0$, $b - Ax \geq 0$, will not work. You have to explicitly eliminate equality constraints.
- The algorithm will sometimes fail for problems that are not strictly dual feasible. For such problems, maxdet will use the maximum number of Newton iterations attempting to solve the first centering problem. Conversely, if maxdet uses the maximum number of iterations and fails to complete even one outer iteration, it may be that the problem is not strictly dual feasible.

Compiling

The source code for the function `maxdet` is in the files `maxdet_src.c` and `maxdet.h`. It is written in ansi-C with calls to LAPACK and BLAS.

LAPACK can be obtained via Netlib (`http://www.netlib.org` or anonymous ftp at `ftp.netlib.org`). A set of optimized BLAS-routines should be supplied by your computer vendor. A non-optimized version can also be obtained from Netlib.

You can also create a mex-file for use in matlab by compiling `maxdet.c` using `cmex`, and linking it with LAPACK and BLAS. The details of compiling the code vary with the compiler used and other factors such as where various libraries are located. We have provided a sample Makefile in the source directory. In order to compile the mex-interface, edit this Makefile as indicated, and type `make`.

If you use a Sun4, DECstation, or HP 9000/700 series workstation, you can just use the executable mex-files provided in the ftp directory `pub/boyd/maxdet` at `isl.stanford.edu`.

3 Matlab routines

To solve a maxdet-program from within matlab, you can proceed as follows:

1. if you know a primal feasible point, use `maxdet.m`,
2. if you do not know a primal feasible point, use `phase1.m` to find one, followed by a call to `maxdet.m`.

3.1 `maxdet.m`

`[x,Z,W,ul,hist,infostr]=maxdet(F,F_blkszs,G,G_blkszs,c,x0,Z0,W0,abstol,reltol,gam,NTiters)`

Purpose

`maxdet` solves the maxdet-program and its dual, given a strictly feasible primal initial point, and optionally, a strictly dual feasible initial point.

Storage convention

Warning: different storage conventions are used in the matlab and C functions.
The matrices F_i and Z have L diagonal blocks:

$$F_i = \begin{bmatrix} F_i^{(1)} & 0 & \cdots & 0 \\ 0 & F_i^{(2)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & F_i^{(L)} \end{bmatrix} \qquad Z = \begin{bmatrix} Z^{(1)} & 0 & \cdots & 0 \\ 0 & Z^{(2)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & Z^{(L)} \end{bmatrix}$$

with block dimensions $n_1, \ldots, n_L$. The dimensions n_i are given in the (row or column) vector `F_blkszs` of length L: `F_blkszs` $= [n_1 \ldots n_L]$. We store these block-diagonal matrices as column vectors, by converting the blocks to vectors and simply concatenating the resulting vectors. Thus the matrix F_i above is stored as the vector

$$\left[F_i^{(1)}(:); \quad F_i^{(2)}(:); \quad \cdots \quad F_i^{(L)}(:) \right].$$

For example, if `F_blkszs` is `[2, 3, 1]`, the matrix

$$Z = \begin{bmatrix} 1 & 2 & 0 & 0 & 0 & 0 \\ 2 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 5 & 6 & 0 \\ 0 & 0 & 5 & 7 & 8 & 0 \\ 0 & 0 & 6 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 \end{bmatrix}$$

would be stored as the column vector

```
[1.0, 2.0, 2.0, 3.0, 4.0, 5.0, 6.0, 5.0, 7.0, 8.0, 6.0, 8.0, 9.0, 10.0 ]' .
```

The matrices F_i, $i = 0, \ldots, m$ are stored in a single matrix with $m+1$ columns and $\sum_{i=1}^{L} n_i^2$ rows. The ith column contains F_i stored in the format described above.

Similar storage convention is used for G_i and W, with K diagonal blocks and the block dimensions stored in `G_blkszs` $= [l_1, \ldots, l_K]$.

Input arguments

For more details, see the description of the C function arguments.

1. `F`: matrix with $m+1$ columns and $\sum_{i=1}^{L} n_i^2$ rows containing the block diagonal matrices $F_0, \ldots, F_m$, using the storage convention described above. The matrices stored in `F` are assumed to be symmetric, and only their lower triangular parts are accessed. The matrices $\mathtt{diag}(F_i, G_i)$, $i = 1, \ldots, m$, must be linearly independent.

2. `F_blkszs`: L-vector with dimensions of the diagonal blocks in F and Z.

3. `G`: matrix with $m+1$ columns and $\sum_{i=1}^{K} l_i^2$ rows containing the block diagonal matrices $G_0, \ldots, G_m$, using the storage convention described above. The matrices stored in `G` are assumed to be symmetric, and only their lower triangular parts are accessed. The matrices $\mathtt{diag}(F_i, G_i)$, $i = 1, \ldots, m$, must be linearly independent.

4. `G_blkszs`: K-vector with dimensions of the diagonal blocks in G and W.

5. `c`: m-vector that specifies the linear part of the primal objective.

6. `x0`: m-vector. The primal starting point x^0. x^0 must be strictly primal feasible, *i.e.*, satisfy $F_0 + \sum_{i=1}^{m} x_i^0 F_i > 0$ and $G_0 + \sum_{i=1}^{m} x_i^0 G_i > 0$.

7. `Z0`: vector of length $\sum_{i=1}^{L} n_i^2$, which contains an n-by-n symmetric matrix Z^0 using the storage described above. Only the lower triangular part of Z^0 is accessed. If Z^0 and W^0 (see below) are strictly dual feasible, *i.e.*, satisfy $\mathtt{Tr} Z^0 F_i + \mathtt{Tr} W^0 G_i = c_i$, $i = 1, \ldots, m$, $Z^0 > 0$ and $W^0 > 0$, they are used in the preliminary phase [VBW96, §7]; otherwise, Z^0 and W^0 are not used at all.

8. `W0`: vector of length $\sum_{i=1}^{K} l_i^2$, which contains an l-by-l symmetric matrix W^0 using the storage described above. Only the lower triangular part of W^0 is accessed.

9. `abstol`: absolute tolerance.

10. `reltol`: relative tolerance.

11. `gam`: algorithm parameter γ, which affects convergence rate. `gam` must be positive. Typical value: 10–1000.

12. `NTiters`: maximum number of total Newton (and predictor) iterations. `maxNTiters` ≥ 1.

Output arguments

1. `x`: m-vector. The last primal iterate x.

2. `Z`: vector of length $\sum_{i=1}^{L} n_i^2$, the last dual iterate Z in the storage described.

3. `W`: vector of length $\sum_{i=1}^{K} l_i^2$, the last dual iterate W in the storage described.

4. `ul`: 2-vector. `ul(1)` is the primal objective value $c^T x + \log\det G(x)^{-1}$; `ul(2)` is the dual objective value $\log\det W - \mathbf{Tr} G_0 W - \mathbf{Tr} F_0 Z + l$.

5. `hist`: matrix with 3 rows and the number of columns equal to the number of outer iterations. The ith column contains the primal objective value, the duality gap and the number of Newton iterations at the ith outer iteration.

6. `infostr`: string. `infostr = 'maximum iterations exceeded'`, `'absolute accuracy reached'`, or `'relative accuracy reached'`.

Caveats

See caveats in C function description.

- The matlab variables `F` and `G` will be modified if `maxdet.m` is terminated by an interrupt.
- All input arguments are full matrices. No provision is made (in this version) for sparse matrices.

3.2 `phase1.m`

```
[x,Z,W,ul,infostr] = phase1(F,F_blkszs,G,G_blkszs,gam,abstol,reltol,NTiters);
```

Purpose

`phase1` computes x that satisfies

$$F_0 + \sum_{i=1}^{m} x_i F_i > 0 \quad \text{and} \quad G_0 + \sum_{i=1}^{m} x_i G_i > 0, \tag{2}$$

or proves that no such x exists. More precisely, `phase1` examines whether the optimal value of the semidefinite program

$$\begin{array}{ll} \text{minimize} & t \\ \text{subject to} & F_0 + x_1 F_1 + \cdots + x_m F_m + tI \geq 0 \\ & G_0 + x_1 G_1 + \cdots + x_m G_m + tI > 0 \end{array} \tag{3}$$

is negative or not. It either provides a strictly feasible pair x, t with $t < 0$ or a strictly feasible pair Z, W for the dual problem

$$\begin{array}{ll} \text{maximize} & -\mathbf{Tr} F_0 Z - \mathbf{Tr} G_0 W \\ \text{subject to} & \mathbf{Tr} F_i Z + \mathbf{Tr} G_i W = 0, \quad i = 1, \ldots, m \\ & \mathbf{Tr} Z + \mathbf{Tr} W = 1 \\ & Z \geq 0, \quad W > 0 \end{array}$$

with objective value $-\mathbf{Tr} F_0 Z - \mathbf{Tr} G_0 W \geq 0$. More details can be found in [VB96, §6].

Input arguments

For more details, see the description of the matlab routine arguments.

1. `F`: matrix with $m+1$ columns and $\sum_{i=1}^{L} n_i^2$ rows containing the block diagonal matrices $F_0, \ldots, F_m$, using the storage convention described above. The matrices stored in `F` are assumed to be symmetric, and only their lower triangular parts are accessed. The matrices $\mathtt{diag}(F_i, G_i)$, $i = 1, \ldots, m$, must be linearly independent.
2. `F_blkszs`: L-vector with dimensions of the diagonal blocks in F and Z.
3. `G`: matrix with $m+1$ columns and $\sum_{i=1}^{K} l_i^2$ rows containing the block diagonal matrices $G_0, \ldots, G_m$.
4. `G_blkszs`: K-vector with dimensions of the diagonal blocks in G and W.
5. `gam`: algorithm parameter γ. `gam` > 0. Typical value: 10–1000.
6. `abstol`: absolute tolerance.
7. `reltol`: relative tolerance.
8. `NTiters`: maximum number of Newton iterations.

Output arguments

1. `x`: m-vector. The last primal iterate. (t is returned in `ul(1)`.)
2. `Z`: vector of length $\sum_{i=1}^{L} n_i^2$. The last dual iterate of Z.
3. `W`: vector of length $\sum_{i=1}^{K} l_i^2$. The last dual iterate of W.
4. `ul`: 2-vector. `ul(1)` is the final value of t. `ul(2)` is the dual objective $-\mathtt{Tr}F_0 Z - \mathtt{Tr}G_0 W$.
5. `infostr`: information string. `infostr = 'feasible'`, `'infeasible'`, or `'feasibility cannot be determined'`.

Caveat

If the maximum number of iterations is reached before a primal feasible point can be found, or infeasibility cannot be proved, then `phase1` returns `infostr = 'feasibility cannot be determined'`. This outcome often means the problem is feasible, but not strictly feasible.

4 Example matlab files

4.1 Positive definite matrix completion with partially specified inverse

`A = mat_completion(Ainit,indices,C)`

Purpose

In this example, we are given a positive definite matrix A_{init} and a symmetric matrix C; $A_{\text{init}}, C \in \mathbf{R}^{n \times n}$. Our goal is to adjust certain off-diagonal entries of A_{init} such that the inverse of the resulting matrix A matches C in every adjustable entry, while the rest (fixed) entries of A are given by A_{init}. Let the free entries of the lower triangular part of A be given by the index pairs (i_k, j_k), $i_k > j_k$, $k = 1, \ldots, m$. This is sufficient to specify the adjustable entries of A since A is symmetric. The problem of positive definite matrix completion with partially specified inverse can then be expressed as the maxdet-program [VBW96, §2.4]

$$\begin{array}{ll} \text{minimize} & \mathtt{Tr}CA(x) + \log\det A(x)^{-1} \\ \text{subject to} & A(x) > 0 \end{array}$$

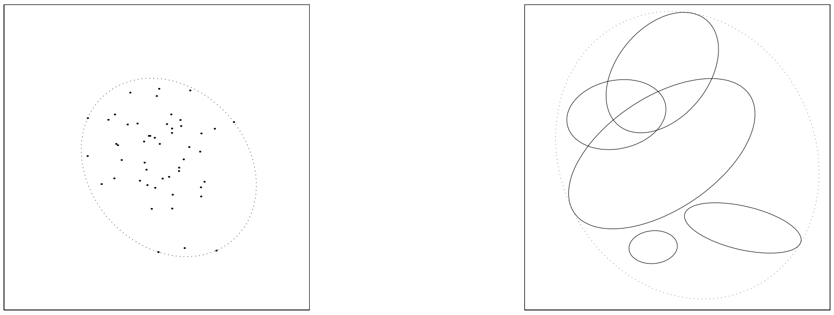

Figure 1: Left: minimum-volume ellipsoid containing 50 points. Right: minimum-volume ellipsoid containing 5 ellipsoids.

with

$$A(x) = A_{\text{init}} + \sum_{k=1}^{m} x_k \left(E_{i_k j_k} + E_{j_k i_k}\right),$$

where E_{ij} denotes the matrix with (i,j) element equal to one and all other elements zero.

Input and output arguments

- `Ainit`: n-by-n positive definite matrix A_{init}.
- `indices`: 2-by-m matrix. The kth column of `indices` gives the index pair (i_k, j_k).
- `C`: n-by-n symmetric matrix C. If `C` is not given, it is set to zero.
- `A`: n-by-n positive definite matrix, the desired completion of A.

4.2 Minimum-volume ellipsoid containing points

`[A,b,c] = minVe_pts(X)`

Purpose

`minVe_pts.m` determines the minimum volume ellipsoid which contains a given set of points in $\mathbf{R}^2$, $X = [x^1, \ldots, x^K]$ [VBW96, §2.1]. Figure 1 shows an instance of the problem.

Input and output arguments

- `X`: 2-by-K matrix, $[x^1, \ldots, x^K]$, containing K points in $\mathbf{R}^2$.
- `A`, `b`, `c`: the minimum volume ellipsoid $\{\ x \mid x^T A x + 2b^T x + c \leq 0,\ x \in \mathbf{R}^2\ \}$.

4.3 Minimum-volume ellipsoid containing ellipsoids

`[A,b,c] = minVe_ell(As,bs,cs)`

Purpose

`minVe_ell.m` determines the minimum volume ellipsoid which contains K given ellipsoids in $\mathbf{R}^2$, $\{\ x \mid x^T A_i x + 2b_i^T x + c_i\ \}$ for $i = 1, \ldots, K$ [VBW96, §2.1]. Figure 1 shows an instance of the problem.

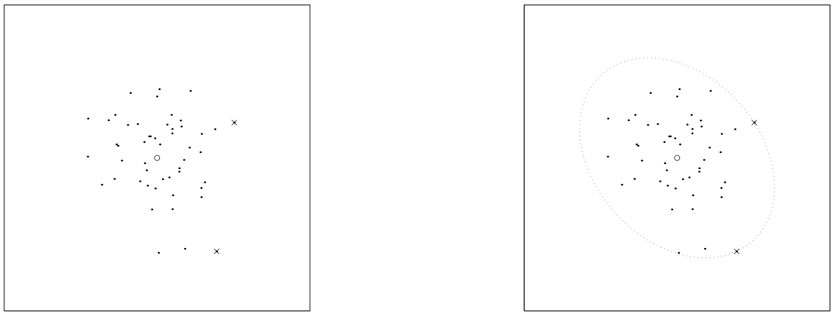

Figure 2: A D-optimal experiment design involving 50 test vectors in $\mathbf{R}^2$. Left: the test vectors used (shown as crosses) and not used (shown as dots). The circle indicates the origin. Right: the previous plot with the minimum-volume ellipsoid, centered at origin, containing the test vectors.

Input and output arguments

- `As`: 2-by-$2K$ matrix, $[A_1, \dots, A_K]$.
- `bs`: 2-by-K matrix, $[b_1, \dots, b_K]$.
- `cs`: K-vector, $[c_1, \dots, c_K]$.
- `A`, `b`, `c`: the minimum volume ellipsoid $\{ \ x \mid x^T A x + 2b^T x + c \le 0, \ x \in \mathbf{R}^2 \ \}$.

4.4 D-optimal experiment design

`[lambda,S] = exp_design(V)`

Purpose

Consider the problem of estimating a vector x from a measurement $y = Ax + w$, where $w \sim N(0, I)$ is measurement noise [VBW96, §2.6]. 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 D-optimal experiment design is to choose the vectors a_i so that the determinant of error covariance $\det(A^T A)^{-1}$ is minimized. We can write $A^T A = q \sum_{i=1}^M \lambda_i v^i {v^i}^T$, 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. `exp_design.m` determines the optimal distribution λ, given M possible test vectors, $V = [v^1, \dots, v^M]$.

Figure 2 shows an instance in $\mathbf{R}^2$ involving 50 test vectors.

In the dual of the D-optimal experiment design problem we compute the minimum volume ellipsoid, centered at origin, that contains the test vectors [VBW96, §3]. The test vectors with a non-zero weight lie on the boundary of the optimal ellipsoid. This is illustrated in Figure 2.

Input and output arguments

- `V`: p-by-M matrix, $[v^1, \dots, v^M]$, containing M possible test vectors in $\mathbf{R}^p$.
- `lambda`: M-vector, $[\lambda_1, \dots, \lambda_M]$.
- `S`: p-by-p matrix, $\sum_{i=1}^M \lambda_i v^i {v^i}^T$.

4.5 Educational testing problem

```
[d,Q] = etp(A)
```

Purpose

`etp.m` solves the educational testing problem [VB96, §2]:

$$
\begin{array}{ll}
\text{maximize} & \sum_{i=1}^{n} d_i \\
\text{subject to} & A - \mathtt{diag}(d) \geq 0 \\
& d \geq 0.
\end{array}
$$

This example illustrates how to solve semidefinite programs using `maxdet`.

Input and output arguments

- `A`: n-by-n matrix. A must be positive definite.
- `d`: n-vector. Solution of the primal problem.
- `Q`: n-by-n matrix. Solution of the dual problem.

5 Future improvements

Please send e-mail to `clive@isl.stanford.edu`, `vandenbe@isl.stanford.edu` or `boyd@isl.stanford.edu` if you wish to be informed about future updates of this software. The present version is rudimentary, and we expect important improvements in the near future, including an improved interface with `sdpsol` [BW95], a parser/solver that simplifies the specification and solution of optimization problems involving matrices, and ability to handle problems with sparse matrices.

Acknowledgment

The research was supported in part by AFOSR (under F49620-95-1-0318), NSF (under ECS-9222391 and EEC-9420565), and MURI (under F49620-95-1-0525).

References

[ABB+92] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide.* SIAM, 1992.

[BW95] S. Boyd and S.-P. Wu. *SDPSOL: A Parser/Solver for Semidefinite Programs with Matrix Structure. User's Guide, Version Alpha.* Stanford University, March 1995.

[VB96] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, March 1996.

[VBW96] L. Vandenberghe, S. Boyd, and S.-P. Wu. Determinant maximization with linear matrix inequality constraints. *submitted to SIMAX*, February 1996.