# SP

# Software for Semidefinite Programming

# User's Guide

Beta Version November 1994

Lieven Vandenberghe, K. U. Leuven vandenbe@esat.kuleuven.ac.be
Stephen Boyd, Stanford University boyd@isl.stanford.edu

# 1 Introduction

## Purpose

This package contains software for solving the *semidefinite programming* problem

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & F(x) \triangleq F_0 + x_1 F_1 + \cdots + x_m F_m \geq 0. \end{array} \tag{1}$$

The problem data are the vector $c \in \mathbf{R}^m$ and $m+1$ and symmetric matrices $F_0, \ldots, F_m \in \mathbf{R}^{n \times n}$; the inequality sign in $F(x) \geq 0$ means that $F(x)$ is positive semidefinite. For background information and applications, see the survey paper [VB94], which is included as part of the software distribution and also available via anonmyous ftp at `isl.stanford.edu` in `pub/boyd/reports/semidef_prog.ps.Z`. We follow the notation of [VB94], with one addition: the matrices $F_i$ are block diagonal, with $L$ diagonal blocks, with dimensions $n_1$, ..., $n_L$ (hence, $n = n_1 + \cdots + n_L$).

## Overview

The package contains:

- C-source: `sp_src.c` and `sp.h` contain a C-function `sp` that solves the general semidefinite programming problem (see §2 for further details).
- Matlab4 interface (which allows the user to call the code from matlab4): Compiled mex-files `sp.mex4` (for Sun4), `sp.mexds` (for DECstation), and `sp.mexhp7` (for HP9000, 700 series) are available in the ftp directory `pub/boyd/semidef_prog` at `isl.stanford.edu` (they are not included in the tar-file). If your machine is one of these types, just copy the appropriate mex-file to your machine. For other platforms you will need to compile the interface yourself using the source code `sp.c`. See §2 for more details.
- Matlab routines: `sp.m`, `bigM.m`, `phase1.m`. These routines allow the user to solve semidefinite programming problems from within matlab, using the mex-files. See §3.
- Matlab examples: we provide simple matlab routines that use `sp.m` to solve various problems (some of which are described in the survey paper [VB94]): linear programming (`lin_prog.m`), matrix norm minimization (`mtrx_nrm.m`), logarithmic Chebychev approximation (`log_cheb.m`), and the educational testing problem (`etp.m`). See §4.
- The survey paper [VB94], in compressed postscript format (`semidef_prog.ps.Z`).

## How to use the package

You can use the package in different ways.

- Write a C-program that calls the function `sp` in `sp_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.

- **The easiest method** (provided you have matlab4) is to use the executable mex-files for Sun4, DECstation, or HP9000/700 workstation, which allow you to just use matlab functions `sp.m`, `bigM.m`, and `phase1.m` from within matlab. All you have to do is retrieve the appropriate mex-file, *i.e.*, `sp.mex4` (for Sun4), `sp.mexds` (for DECstation), or `sp.mexhp7`[1] (for HP9000/700 workstation) and copy it to the directory `semidef` on your machine. See §3.

- Create a mex-file for use in matlab4 by compiling `sp.c` using `cmex`, and linking it with LAPACK and BLAS (see §2 for more details). If you use a Sun4, DECstation, or HP9000/700 workstation, you can just use the binary mex-files we provide.

## Who can/should use the code?

The code is intended for researchers in control theory or other disciplines who want to apply semidefinite programming. Our aim is to provide a program that is easy to use, reasonably efficient, and useful for small and medium-sized problems (say, up to a few hundred variables). People who are active in research on interior-point methods probably have more sophisticated code of their own and may find this software less useful. People who solve large-scale problems (say, several hundreds or thousands of variables) should use interior-point methods that exploit problem structure; see [VB95] and [BVG94] (available via anonmyous ftp at `isl.stanford.edu` in `pub/boyd/reports`).

## Related software

`lmitool`, developed by Laurent El Ghaoui at ENSTA in Paris (`elghaoui@ensta.fr`), is a user-friendly front end to this code. `lmitool` allows users to specify semidefinite programming problems interactively from within matlab. It is especially well suited for problems with matrix variables.

Franz Rendl (`rendl@fmathbds01.tu-graz.ac.at`) at the Technical University of Graz has developed a Matlab toolbox for semidefinite programming.

Another related package is `LIPSOL`, developed by Y. Zhang [Zha94]. `LIPSOL` solves linear programs via an interior point method, and exploits sparsity, and so can be used for large problems. `LIPSOL` is available via anonymous ftp into `ftp.math.umbc.edu` in `pub/zhang/lipsol`.

## Caveats

- The *only* structure our code exploits is block-diagonal structure of $F$ and the dual matrix $Z$. For many practical problems, additional structure can be exploited to great benefit; see the papers [VB95] and [BVG94]. As one example, `sp` can be used to solve multiple Lyapunov inequalities, but the running time will grow as $O(p^6)$, where $p$ is the dimension of the matrices involved. Specialized algorithms that exploit structure

[1] `sp.mexhp7` has been compiled using standard, non-optimized BLAS routines. For optimal performance, we recommend recompiling the code using your manufacturers' BLAS library.

can solve such problems in $O(p^4)$ (see [VB95]). In many cases the available memory will determine the maximum sizes of the problems that can be solved using sp. The required memory for the multiple Lyapunov problem, for example, grows as $O(p^4)$.

- sp is a simple, general code; it does not use any special handling of particular problem types (*e.g.*, linear programming, matrix norm minimization, etc.) for efficiency. As an example, when sp is used to solve an LP, it does a huge amount of silly computation (Cholesky factorization of $1\times 1$ matrices, generalized eigenvalue problems with $1\times 1$ matrices, ...). It does work, though!

- A wide variety of nonlinear convex problems can be recast as semidefinite programs. Expressing a convex optimization problem as a semidefinite program yields conceptual advantages (*e.g.*, we can readily work out a dual, give optimality criteria) and some practical advantages (*e.g.*, the simple code sp can handle many convex problems). In many cases, embedding a problem in the semidefinite programming framework does not yield the fastest interior-point method. In the language and framework developed by Nesterov and Nemirovsky [NN94], embedding a problem class into semidefinite programming yields *a* self-concordant barrier, but not necessarily one with a small parameter $\theta$ (which plays a role in the complexity analysis). As one example, we can solve quadratically constrained quadratic programs

$$\begin{array}{ll} \text{minimize} & (A_0x+b_0)^T(A_0x+b_0)-c_0^Tx-d_0 \\ \text{subject to} & (A_ix+b_i)^T(A_ix+b_i)-c_i^Tx-d_i \leq 0, \quad i=1,\ldots,L, \end{array}$$

by expressing them as semidefinite programs (see [VB94]), or as optimization problems over the second-order cone (see [NN94, §6.2.3]). The semidefinite programming formulation will be less efficient, especially when the matrices $A_i$ have high rank.

# 2 C routine

The main routine is a C-function sp:

```
int sp( int m, int L, double *F, int *blck_szs, double *c, double *x,
        double *Z, double *ul, double nu, double abstol, double reltol,
        double tv, int *iters, double *work, int lwork, int *iwork,
        int *info )
```

## Purpose

sp solves the semidefinite program

$$\begin{array}{ll} \text{minimize} & c^Tx \\ \text{subject to} & F_0+x_1F_1+\cdots+x_mF_m \geq 0 \end{array}$$

and its dual

$$\begin{array}{ll} \text{maximize} & -\mathbf{Tr}F_0Z \\ \text{subject to} & Z=Z^T\geq 0 \\ & \mathbf{Tr}F_iZ=c_i, \quad i=1,\ldots,m, \end{array}$$

given strictly feasible primal and dual initial points.

## Storage convention

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

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

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

For example, if `blck_szs` 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` } .

## 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 $F_i$, $i = 1, \ldots, m$, must be linearly independent.

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

5. `c` (pointer to double): Array of length $m$. Vector that specifies primal objective.

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

7. `Z` (pointer to double): Array of length $\sum_{i=1}^{L} n_i(n_i+1)/2$, giving the dual variable, using the storage scheme described above. On entry, the matrix $Z$ must be strictly dual feasible, *i.e.*, must satisfy $\mathbf{Tr} F_i Z = c_i$, $i = 1, \ldots, m$ and $Z > 0$. On exit, $Z$ is the last dual iterate. The meaning of the last dual iterate depends on the stopping criterion.

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

9. `nu` (double). The parameter $\nu$ controls the rate of convergence (see [VB94] or §5 for the meaning of $\nu$). $\nu \geq 1.0$; recommended values: 10.0–50.0.

10. `abstol` (double). Absolute tolerance. For the precise interpretation of `abstol`, see the section on convergence criteria below. If `abstol` $<$ `1e-8`, a value of `1e-8` is used instead of `abstol`.

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

12. `tv` (double). Target value. `tv` is only referenced if `reltol` $< 0.0$ (see the section on convergence criteria below).

13. `iters` (pointer to integer). On entry: `*iters` is the maximum number of iterations. If `*iters` $< 0$ the maximum number of iteration is set to 100. On exit, `*iters` is the number of iterations taken.

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

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

$$(m+2)N_{\mathrm{pd}} + N_{\mathrm{up}} + 2n + \max\{m + N_{\mathrm{pd}} N_B, 3n_{\max} + n_{\max}(n_{\max}+1), 3m\}$$

with

$$N_{\mathrm{pd}} = \sum_{i=1}^{L} n_i(n_i+1)/2, \quad N_{\mathrm{up}} = \sum_{i=1}^{L} n_i^2, \quad n_{\max} = \max_i n_i.$$

$N_B \geq 1$. 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]).

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

17. `info` (pointer to integer). On exit, `*info` $= 1$ if the maximum number of iterations is exceeded; `*info` $= 2$ if the absolute accuracy has been reached; `*info` $= 3$ if the relative accuracy has been reached; `*info` $= 4$ if the target value has been reached; `*info` $= 5$ if the target value is not achievable. Negative values of `*info` indicate errors: `*info` $= -i$ means that the $i$th argument has an illegal value; `*info` $= -18$ stands for all other errors.

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

## Convergence criteria

On exit, `sp` 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 upper bound `ul[0]` is the primal objective $c^T x$ evaluated at the primal feasible solution stored in `x`; the lower bound `ul[1]` is the dual objective $-\mathbf{Tr} F_0 Z$ evaluated at the dual feasible solution stored in `Z`. The quantity `ul[0] - ul[1]` is called the *duality gap* associated with $x$ and $Z$ (See [VB94] for the underlying theory).

The program exits normally under five possible conditions:

- *The maximum number of iterations is exceeded.* We note that it is possible to call `sp` with `*iters = 0` , in which case `sp` simply verifies primal and dual feasibility and evaluates the objective functions.
- *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

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

- *The target value is reached.* `reltol` is negative and the primal objective `ul[0]` is less than `tv`.
- *The target value is unachievable.* `reltol` is negative and the dual objective `ul[1]` is greater than `tv`.

The target value stopping condition is useful for feasibility problems: `sp` stops when it has either found a point with objective less than the given `tv`, or has found a dual point with objective value greater than `tv` (which proves the optimal objective is greater than `tv`). The target value stopping condition is enabled by passing a negative relative tolerance argument, *e.g.*, $\texttt{reltol} = -1.0$.

## Caveats

- The code returns with an error message if the matrices $F_i$, $i = 1, \ldots, m$, are linearly dependent. If the matrices $F_i$ are dependent, *i.e.*, if there exists $\lambda_1, \ldots, \lambda_m$, not all zero, such that

$$\lambda_1 F_1 + \cdots + \lambda_m F_m = 0$$

  then either the problem is not dual feasible (if $\sum_i \lambda_i c_i \neq 0$) or it can be reduced to a problem with fewer variables.

- The code requires *strictly* feasible primal and dual points, which means that standard trick of adding equality constraints (writing $Ax = b$ as the two vector inequalities $Ax - b \geq 0$, $b - Ax \geq 0$) will not work. You have to explicitly eliminate equality constraints.

### Compiling

The source code for the function `sp` is in the files `sp_src.c` and `sp.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 matlab4 by compiling `sp.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 HP9000/700, you can just use the executable mex-files provided in the ftp directory `pub/boyd/semidef_prog` at `isl.stanford.edu`.

# 3 Matlab routines

The main routines are `sp.m`, `phase1.m`, and `bigM.m`. To solve a given semidefinite program, you can proceed as follows:

1. if you know a primal and dual feasible point, use `sp.m`,
2. if you know a primal feasible point, but no dual feasible point, use `bigM.m`,
3. if you do not know primal or dual feasible points, use `phase1.m` to find a primal feasible point, followed by a call to `bigM.m`.

The other m-files (`etp.m`, `mtrx_nrm.m`, `lin_prog.m`, and `log_cheb.m`) are included as illustrations of the main routines.

## 3.1 sp.m

```
[x,Z,ul,info,time] = sp(F,blck_szs,c,x0,Z0,nu,abstol,reltol,tv,maxiters);
```

### Purpose

`sp` solves the semidefinite program

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & F_0 + x_1 F_1 + \cdots + x_m F_m \geq 0 \end{array}$$

and its dual

$$\begin{array}{ll} \text{maximize} & -\mathbf{Tr} F_0 Z \\ \text{subject to} & Z = Z^T \geq 0 \\ & \mathbf{Tr} F_i Z = c_i, i = 1, \ldots, m, \end{array}$$

given strictly feasible primal and dual initial points.

## Storage convention

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

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

and similarly for $Z$. The dimensions $n_i$ of the blocks are given in the (row or column) vector `blck_szs` of length $L$: `blck_szs` $= [n_1 \ldots n_L]$. 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

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

For example, if `blck_szs` 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 $i$th column contains $F_i$ stored in the format described above.

## Input 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 $F_i$, $i = 1, \ldots, m$, must be linearly independent.

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

3. `c`: $m$-vector that specifies primal objective.

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

5. `Z0`: vector of length $\sum_{i=1}^L n_i^2$, which contains the dual starting point $Z^0$ stored as described above. The matrix $Z^0$ is assumed to be symmetric, and only its lower triangular part is accessed. $Z^0$ must be strictly dual feasible, *i.e.*, satisfy $\mathbf{Tr} Z^0 F_i = c_i$ and $Z^0 > 0$.

6. `nu`: scalar parameter that controls the rate of of convergence (see [VB94] or §5 for the meaning of $\nu$). $\nu \geq 1.0$; recommended values: 10.0–50.0.

7. `abstol`: absolute tolerance.

8. `reltol`: relative tolerance.

9. `tv`: target value.

10. `maxiters`: maximum number of iterations. `maxiters` $\geq 0$.

For the meaning of `abstol`, `reltol`, and `tv`, see the section on convergence criteria on page 7.

## 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 $Z$ stored as described above.

3. `ul`: 2-vector. `ul(1)` is the primal objective value $c^T x$, `ul(2)` is the dual objective value $-\mathbf{Tr} F_0 Z$.

4. `info`: string. `info = 'maxiters exceeded'`, `'target reached'`, `'unachievable target'`, `'relative accuracy reached'`, or `'absolute accuracy reached'`.

5. `time`: 3-vector.

$$\texttt{time} = [\text{user time (sec.)},\ \text{system time (sec.)},\ \text{number of iterations}].$$

   The CPU-time used is the sum of user time and sytem time.

### Caveats

- The matrices $F_1, \ldots, F_m$ must be linearly independent. If the matrices $F_i$ are dependent, *i.e.*, if there exists $\lambda_1, \ldots, \lambda_m$, not all zero, such that

$$\lambda_1 F_1 + \cdots + \lambda_m F_m = 0$$

  then either the problem is not dual feasible (if $\sum_i \lambda_i c_i \neq 0$) or it can be reduced to a problem with fewer variables.

- The matlab variable `F` will be modified if `sp.m` is terminated by an interrupt.

- All variables must be full. No provision is made for sparse matrices.

- The code requires *strictly* feasible primal and dual points, which means that standard trick of adding equality constraints (writing $Ax = b$ as the two vector inequalities $Ax - b \geq 0$, $b - Ax \geq 0$) will not work. You have to explicitly eliminate equality constraints.

## 3.2 `bigM.m`

```
[x,Z,z,ul,iters] = bigM(F,c,blck_szs,x0,M,nu,abstol,reltol,tv,maxiters);
```

### Purpose

`bigM` solves the semidefinite program

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & F_0 + x_1 F_1 + \cdots + x_m F_m \geq 0 \\ & \mathbf{Tr}\,(F_0 + x_1 F_1 + \cdots + x_m F_m) \leq M \end{array}$$

and its dual

$$\begin{array}{ll} \text{maximize} & -\mathbf{Tr} F_0 (Z - zI) - Mz \\ \text{subject to} & \mathbf{Tr} F_i (Z - zI) = c_i, \quad i = 1, \ldots, m \\ & Z \geq 0, \quad z \geq 0, \end{array}$$

given a strictly feasible primal point. See [VB94, §6] for discussion.

### Input 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 scheme for `sp.m`. The matrices $F_i$, $i = 1, \ldots, m$, must be linearly independent.

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

3. `c`: $m$-vector that specifies primal objective.

4. `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 $\mathbf{Tr}\,(F_0 + \sum_{i=1}^m x_i^0 F_i) < M$.

5. `M`: scalar parameter. $M > \mathbf{Tr}\,(F_0 + \sum_{i=1}^m x_i^0 F_i)$.

6. `nu`: scalar parameter that controls the rate of of convergence (see [VB94] or §5 for the meaning of $\nu$). $\nu \geq 1.0$; recommended values: 10.0–50.0.

7. `abstol`: absolute tolerance.

8. `reltol`: relative tolerance.

9. `tv`: target value.

10. `maxiters`: maximum number of iterations. `maxiters` $\geq 0$.

For the meaning of `abstol`, `reltol`, and `tv`, see the section on convergence criteria on page 7.

## Output arguments

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

2. `Z`: vector of length $\sum_{i=1}^L n_i^2$. The block diagonal part $Z$ of the last dual iterate, stored as described above.

3. `z`: scalar. The scalar part $z$ of the last dual iterate.

4. `ul`: 2-vector. `ul(1)` is the primal objective value $c^T x$, `ul(2)` is the dual objective $-\mathbf{Tr} F_0(Z - zI) - Mz$.

5. `iters`: number of iterations.

## Caveats

- Not much can be said about the right choice of $M$. The constraint $\mathbf{Tr} F(x) \leq M$ is needed to bound the feasible set; its purpose is to change the dual problem to one with an obvious feasible solution. In most cases, however, there are more natural ways to achieve the same effect, and it is always worth your while to try to find a dual feasible point directly, without adding the big-$M$ constraint. As an example, consider the multiple Lyapunov problem

$$\begin{array}{ll} \text{minimize} & t \\ \text{subject to} & -A_k^T P - P A_k + tI \geq 0, \quad k = 1, \ldots, L \\ & P \geq 0. \end{array}$$

The left hand sides of the inequalities are homogeneous, and a constraint $P \leq I$ can be added without loss of generality. The dual of this modified problem is:

$$\begin{array}{ll} \text{maximize} & -\mathbf{Tr} Z_{L+2} \\ \text{subject to} & -Z_{L+2} + Z_{L+1} = \sum_{k=1}^{L} A_k Z_k + Z_k A_k^T \\ & \sum_{k=1}^{L} \mathbf{Tr} Z_k = 1 \\ & Z_k \geq 0, \quad k = 1, \ldots, L+2. \end{array}$$

For this modified problem it is not difficult to find dual feasible solutions.

- `bigM` calls `sp`, and therefore the caveats on page 11 are valid for `bigM` as well.

### 3.3 `phase1.m`

```
[x,Z,z,ul,iters] = phase1(F,blck_szs,M,nu,abstol,maxiters);
```

## Purpose

`phase1` computes a vector $x$ that satisfies

$$F_0 + \sum_{i=1}^{m} x_i F_i > 0 \quad \text{and} \quad \mathbf{Tr}\left(F_0 + \sum_{i=1}^{m} x_i F_i\right) < M, \tag{2}$$

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

$$\begin{array}{ll} \text{minimize} & t \\ \text{subject to} & F_0 + x_1 F_1 + \cdots + x_m F_m + tI \geq 0 \\ & \mathbf{Tr}\left(F_0 + \sum_{i=1}^{m} x_i F_i\right) \leq M \end{array} \tag{3}$$

is positive or negative. It either provides a feasible pair $x$, $t$ with $t < 0$ or a feasible pair $Z$, $z$ for the dual problem

$$\begin{array}{ll} \text{maximize} & -\mathbf{Tr} F_0 (Z - zI) - Mz \\ \text{subject to} & \mathbf{Tr} F_i (Z - zI) = 0, \; i = 1, \ldots, m \\ & \mathbf{Tr} Z = 1 \\ & Z \geq 0, \quad z \geq 0 \end{array}$$

with objective value $-\mathbf{Tr} F_0 (Z - zI) - Mz > 0$.

More details can be found in [VB94, §6].

## Input 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 scheme for `sp.m`. The matrices $F_i$, $i = 1, \ldots, m$, must be linearly independent.

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

3. `M`: scalar. $M > \mathbf{Tr} F_0$.

4. `nu`: scalar parameter that controls the rate of of convergence (see [VB94] or §5 for the meaning of $\nu$). $\nu \geq 1.0$; recommended values: 10.0–50.0.

5. `abstol`: absolute tolerance (see below).

6. `maxiters`: maximum number of 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 block diagonal part $Z$ of the last dual iterate stored as described above.

3. `z`: scalar part $z$ of the last dual iterate.

4. `ul`: 2-vector. `ul(1)` is the final value of $t$. `ul(2)` is the dual objective $-\mathbf{Tr} F_0(Z - zI) - Mz$.

5. `iters`: number of iterations.

## Convergence criteria

`phase1` exits normally under four possible conditions.

- *The maximum number of iterations is exceeded.*
- *The absolute tolerance is reached:* `ul(1)` $-$ `ul(2)` $<$ `abstol`.
- *A feasible solution to (2) has been found:* `ul(1)` $< 0.0$.
- *The inequalities (2) are infeasible:* `ul(2)` $> 0.0$.

## Caveats

- The constraint $\mathbf{Tr} F(x) \leq M$ can be active on exit, *i.e.*, one may have $\mathbf{Tr} F(x) = M$ for the $x$ returned by `phase1`. If you use this $x$-value as input to `bigM.m`, you will have to increase $M$ (*e.g.*, replace $M$ by $1.1M$).

- `phase1.m` calls `sp.m` and therefore the caveats on p. 11 are valid here as well.

# 4 Example matlab files

## 4.1 Linear programming

```
[x,z,info] = lin_prog(A,b,c)
```

### Purpose

`lin_prog.m` solves the linear program

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax + b \geq 0 \end{array}$$

and its dual

$$\begin{array}{ll} \text{maximize} & -b^T z \\ \text{subject to} & A^T z = c \\ & z \geq 0, \end{array}$$

using the semidefinite programming reformulation (see [VB94, §1.1]). Note that this dual problem has the standard form commonly used in linear programming literature.

### Input arguments

1. `A`: $n \times m$-matrix. Must have rank $m$.
2. `b`: $n$-vector.
3. `c`: $m$-vector.

### Output arguments

1. `x`: $m$-vector. `x` is the primal solution if `info = 'optimal'`, and `x = []` otherwise.
2. `z`: $n$-vector. `z` is the dual solution if `info = 'optimal'`, and `z = []` otherwise.
3. `info`: string. `info = 'optimal'` or `'maxiters exceeded'` or `'may be infeasible'` or `'may be unbounded below'`.

### Caveats

- `lin_prog.m` requires strict feasibility, and does not handle equality constraints expressed in the inequalities, *e.g.*, writing $Fx = g$ as the two vector inequalities $Fx - g \geq 0$, $g - Fx \geq 0$. You have to explicitly eliminate equality constraints.
- `sp` is not optimized for solving linear programs; see general caveats above.
- `lin_prog.m` calls `phase1.m` and `bigM.m` with a default setting for $M$. The code will exit with `info = 'may be infeasible'` or `'may be unbounded below'` when $M$ is too small.

## 4.2 Matrix norm minimization

`[x,Q]= mtrx_nrm(A,q)`

### Purpose

`mtrx_nrm.m` solves the matrix norm minimization problem

$$\text{minimize } \|A_0 + x_1 A_1 + \cdots + x_k A_k\|$$

and its dual

$$\begin{array}{ll} \text{maximize} & -\mathbf{Tr} A_0 Q \\ \text{subject to} & \mathbf{Tr} A_i Q = 0, \ i = 1, \ldots, k \\ & \|Q\|_* \leq 1. \end{array}$$

The matrices $A_i$ have size $p \times q$. $\|Q\|_*$ denotes the *nuclear norm* of $Q$, *i.e.*, the sum of the singular values of $Q$ (see [VB94, §3.1]).

### Input arguments

1. `A`: matrix of size $p \times (k+1)q$. `A` $= [\ A_0 \ A_1 \ldots A_k\ ]$.
2. `q`: scalar. The number of columns of each $A_i$.

### Output arguments

1. `x`: $k$-vector. Primal solution.
2. `Q`: $p \times q$-matrix. Dual solution.

## 4.3 Logarithmic Chebychev approximation

`[x,ul] = log_cheb(A)`

### Purpose

`log_cheb.m` solves the *logarithmic Chebychev approximation* problem (see [VB94, §2]):

$$\begin{array}{ll} \text{minimize} & t \\ \text{subject to} & (1/t)e \leq Ax \leq te \end{array}$$

where $A$ is a $p \times k$ matrix, and $e$ denotes the vector with all components one.

### Input arguments

1. `A`: $p \times k$-matrix of rank $k$. The set of inequalities $Ax > 0$ must be feasible.

### Output arguments

1. `x`: $k$-vector. The optimal solution.
2. `ul`: 2-vector. The minimum value of $t$ lies in the interval [`ul(2)`, `ul(1)`].

## 4.4 Educational testing problem

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

### Purpose

`etp.m` solves the *educational testing problem* (see [VB94, §2]):

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

and its dual

$$\begin{array}{ll} \text{minimize} & \mathbf{Tr} AQ \\ \text{subject to} & Q \geq 0 \\ & Q_{ii} \geq 1, \quad i = 1, \ldots, n. \end{array}$$

### Input arguments

1. `A`: $n \times n$-matrix. $A$ must be positive definite.

### Output arguments

1. `d`: $n$-vector. Solution of the primal problem.
2. `Q`: $n \times n$-matrix. Solution of the dual problem.

# 5 Algorithm

The algorithm is due to Nesterov and Todd [NT94], and is also described in [VB94, §5.5]. It is a particular primal-dual potential reduction method. These methods are based on the potential function

$$\varphi(x, Z) = (n + \nu\sqrt{n}) \log(c^T x + \mathbf{Tr} F_0 Z) - \log\det F(x) - \log\det Z - n \log n,$$

and have the general form:

**Primal-dual potential reduction algorithm**

**given** strictly feasible $x$ and $Z$.

**repeat**

1. Find a suitable direction $\delta x$, and a suitable dual feasible direction $\delta Z$.
2. Find $p$, $q \in \mathbf{R}$ that minimize $\varphi(x + p\delta x, Z + q\delta Z)$.
3. Update: $x := x + p\delta x$ and $Z := Z + q\delta Z$.

**until** stopping condition.

The directions $\delta x$ and $\delta Z$ are obtained as follows:

1. Compute the generalized eigenvalue decomposition $ZF(x)V = V\Lambda$, and normalize $V$ s.t. $V^T F(x) V = \Lambda^{1/2}$.

2. Find $\delta x$ by solving the least-squares problem

$$\delta x = \underset{v \in \mathbf{R}^m}{\operatorname{argmin}} \left\| \rho\Lambda^{1/2} - \Lambda^{-1/2} + V^T \left( \sum_{i=1}^m v_i F_i \right) V \right\|_F ,$$

   with $\rho = (n + \nu\sqrt{n})/\mathbf{Tr} F(x)Z$.

3. Compute $\delta Z$ as

$$\delta Z = -V \left( \rho\Lambda^{1/2} - \Lambda^{-1/2} + \sum_{i=1}^m \delta x_i V^T F_i V \right) V^T .$$

The plane search (and much more) is described in [VB94].

# 6 Future improvements

Please send e-mail to `vandenbe@esat.kuleuven.ac.be` 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. In particular, we hope to include some of the following features:

- Techniques to exploit structure in the equations. This should increase the efficiency by orders of magnitude.
- Possibility to start from infeasible starting points.
- Predictor steps to accelerate the local convergence.
- A front end that simplifies the handling of variables that are matrices.

# Acknowledgment

We thank Laurent El Ghaoui and Istvan Kollar for their comments on a preliminary version of the software and documentation.

L. Vandenberghe is postdoctoral researcher of the Belgian National Fund for Scientific Research (NFWO). His research was supported in part by the Belgian program on interuniversity attraction poles (IUAP 17 and 50) initiated by the Belgian State, Prime Minister's Office, Science Policy Programming. The research of S. Boyd was supported in part by AFOSR (under F49620-92-J-0013), NSF (under ECS-9222391), and ARPA (under F49620-93-1-0085).


# 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.

[BVG94] 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.

[NN94] Yu. Nesterov and A. Nemirovsky. *Interior-point polynomial methods in convex programming*, volume 13 of *Studies in Applied Mathematics*. SIAM, Philadelphia, PA, 1994.

[NT94] Yu. E. Nesterov and M. J. Todd. Self-scaled cones and interior-point methods in nonlinear programming. Technical Report 1091, Cornell University, April 1994.

[VB94] L. Vandenberghe and S. Boyd. Semidefinite programming. Submitted to *SIAM Review*, July 1994.

[VB95] L. Vandenberghe and S. Boyd. Primal-dual potential reduction method for problems involving matrix inequalities. To be published in *Math. Programming*, 1995.

[Zha94] Y. Zhang. *User's guide to LIPSOL: a matlab toolkit for linear programming interior-point solvers*. Math. & Stat. Dept., Univ. Md. Baltimore County, October 1994. Beta release.