Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

C. HELMBERG[1]

SBmethod

A C++ Implementation of the Spectral Bundle Method

http://www.zib.de/helmberg/SBmethod

Manual to Version 1.1

[1]Konrad-Zuse-Zentrum für Informationstechnik Berlin, Takustraße 7, D-14195 Berlin, Germany, helmberg@zib.de, http://www.zib.de/helmberg

ZIB-Report 00-35 (October 2000)

Abstract. `SBmethod`, Version 1.1, is an implementation of the spectral bundle method for eigenvalue optimization problems of the form

$$\min_{y\in\mathbb{R}^m} \; a\,\lambda_{\max}(C-\sum_{i=1}^{m} A_i y_i)+b^T y.$$

The design variables y_i may be sign constrained, C and and A_i are given real symmetric matrices, $b \in \mathbb{R}^m$ allows to specify a linear cost term, and $a > 0$ is a constant multiplier for the maximum eigenvalue function $\lambda_{\max}(\cdot)$. The code is intended for large scale problems and allows to exploit structural properties of the matrices such as sparsity and low rank structure.

The manual contains instructions for installation and use of the program. It describes in detail input format, options, and output. The meaning of the variables and parameters is made precise by relating them to a mathematical description of the algorithm in pseudocode.

Mathematics subject classification (MSC 2000). Primary 90-04; secondary 90C22, 90C06

Keywords. semidefinite programming, semidefinite relaxations, spectral bundle method, eigenvalue optimization, large-scale problems.

SBmethod

A C++ Implementation of the Spectral Bundle Method

http://www.zib.de/helmberg/SBmethod

Manual to Version 1.1

Christoph Helmberg
(helmberg@zib.de)

Konrad-Zuse-Zentrum für Informationstechnik Berlin
Takustraße 7
14195 Berlin
Germany

Contents

1 Introduction

`SBmethod` implements the spectral bundle method of Helmberg and Rendl [2000]; Helmberg and Kiwiel [1999] (see Helmberg [2000] for a self contained introduction) for large scale eigenvalue optimization problems of the form

$$\min_{y\in Y} \quad a\,\lambda_{\max}(C-\mathcal{A}^T y)+b^T y. \tag{1}$$

The function $\lambda_{\max}(\cdot)$ denotes the maximal eigenvalue and $C\in S_n$, $\mathcal{A}^T:\mathbb{R}^m\to S_n$, $b\in\mathbb{R}^m$, and $a\in\mathbb{R}$ with $a>0$ are given data (S_n denotes the set of symmetric matrices of order n). The matrix $C\in S_n$ is called the *cost matrix* and the linear operator $\mathcal{A}^T:\mathbb{R}^m\to S_n$ is defined by

$$\mathcal{A}^T y=\sum_{i=1}^m y_i A_i,$$

where the $A_i\in S_n$ for $i=1,\dots,m$ are given symmetric matrices of order n. The matrices C and A_i should be well structured (sparsity and various low rank structures are supported by the code, see Section 2).

The set $Y\subset\mathbb{R}^m$, over which is optimized, is a Cartesian product of real numbers $\mathbb{R}$, nonnegative real numbers $\mathbb{R}_+=\{x\in\mathbb{R}:x\geq 0\}$, and nonpositive real numbers $\mathbb{R}_-=\{x\in\mathbb{R}:x\leq 0\}$, *i.e.*, there is a partition $(J_=,J_\geq,J_\leq)$ of the index set $\{1,\dots,m\}$ with

$$Y=\{y\in\mathbb{R}^m:\quad y_i\geq 0 \text{ for } i\in J_\geq,\ y_i\leq 0 \text{ for } i\in J_\leq\}.$$

The somewhat peculiar choice of Y was motivated by the following connection to semidefinite programs with constant trace $a>0$. The eigenvalue optimization problem (1) is equivalent to the dual of the semidefinite program

$$\begin{array}{rll} \max & \langle C,X\rangle & \\ \text{s.t.} & \langle I,X\rangle = a & \\ & \langle A_i,X\rangle = b_i & i\in J_= \\ & \langle A_i,X\rangle \leq b_i & i\in J_\geq \\ & \langle A_i,X\rangle \geq b_i & i\in J_\leq \\ & X\succeq 0, & \end{array} \tag{2}$$

where $\langle A,B\rangle=\sum_{i,j}a_{ij}b_{ij}$ denotes the inner product for matrices $A,B\in\mathbb{R}^{m\times n}$, and $X\succeq 0$ is short for requiring X to be positive semidefinite. Alternatively, we write $X\in S_n^+$ with S_n^+ the set of symmetric positive semidefinite matrices of order n. If the equality constraints for $i\in J_=$ already imply $\langle I,X\rangle=a$ then the equivalence of (1) to the dual of (2) is true without the constraint $\langle I,X\rangle=a$.

Semidefinite programs of type (2) arise frequently in semidefinite relaxations of combinatorial optimization problems (see, *e.g.*, the survey of Goemans [1997]). In this context any feasible solution of (1) yields an upper bound on the combinatorial optimization problem. The desire to compute such bounds quickly via approximate solutions to (1) for well structured matrices A_i has been the main motivation for writing this software. The combinatorial optimization background had a heavy influence on many design decisions. This is maybe best visible in the input format, which is tailored for (2) rather than for (1) (see Section 2).

Although the distribution-package contains the full source code, its end is only to provide a compiled program `sb` for solving (1). Therefore, this manual concentrates on explaining how to compile and use the program `sb`.

The source code is, unfortunately, the result of many years of experimentation in various directions without proper documentation and cleaning, a fact, that I am not at all proud of. Therefore, for the time being, the source code is best ignored.

For installation, get the gzipped tar-file `SBm_v1.1.tar.gz` from the URL

```
http://www.zib.de/helmberg/SBmethod
```

Execute

```
gunzip SBm_v1.1.tar.gz
tar xvf SBm_v1.1.tar          (creates subdirectory SBmethod and fills it)
cd SBmethod
more README
```

Follow the instructions of the `README` file in order to make the program `sb` (if you are using gnu-make and g++ then typing `make` should suffice with high probability). Installation is complete if you find an executable program `sb` in the directory `SBmethod`.

The program `sb` reads the problem description either from the standard input or a file. The format of the problem description is explained in Section 2. The code offers the possibility (for better or for worse) to tune most of the algorithmic parameters to specific needs via options or a parameter file, see Section 4. In order to explain the meaning of these parameters I give a rough description of the algorithm in Section 3, but I sometimes assume familiarity with Kiwiel [1990]; Helmberg and Rendl [2000]; Helmberg and Kiwiel [1999].

Acknowledgment. I thank K. C. Kiwiel who not only contributed many ideas but also improved the reliability of my code significantly. He pointed out many mistakes in an earlier version of the code by miraculously spotting tiny inconsistencies in tons of logfiles that I sent to him by email.

2 Input Format

The format of the problem description for (1) (either read from standard input or from a file) is based on the primal problem (2) and is as follows

a
C
m

$A_1 \quad = | < | > \quad b_1$

$\vdots$

$A_m \quad = | < | > \quad b_m$

The lines "$A_i \quad = | < | > \quad b_i$" represent constraints of the form $\langle A_i, X\rangle = b_i$ (for =), $\langle A_i, X\rangle \le b_i$ (for <), or $\langle A_i, X\rangle \ge b_i$ (for >). The matrices C and A_i are specified by one of the matrix formats listed below (CAUTION: the code uses the convention that indices start at zero!). Before describing these formats in detail we give a tiny example.

Example: Max-cut relaxation for a graph on 5 nodes.

$$\begin{array}{ll} \max & \langle C, X\rangle \\ \text{s.t.} & \operatorname{diag}(X) = \mathbf{1} \\ & X \succeq 0, \end{array} \qquad \min_y a\lambda_{\max}(C - \operatorname{Diag}(y)) + \mathbf{1}^T y$$

$a = \quad 5 \quad$ (multiplier for $\lambda_{\max}$)

$$C = \begin{bmatrix} 0.5 & -0.25 & -0.25 & 0 & -0.25 \\ -0.25 & 0.5 & 0 & -0.25 & -0.25 \\ -0.25 & 0 & 0.5 & 0 & 0 \\ 0 & -0.25 & 0 & 0.5 & 0 \\ -0.25 & -0.25 & 0 & 0 & 0.5 \end{bmatrix}$$

$m = \quad 5 \quad$ (#constraints)

$$A_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$=$ (equation) $\quad 1 \quad (= b_1)$

$$A_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$=$ (equation) $\quad 1 \quad (= b_2)$

$$A_3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$=$ (equation) $\quad 1 \quad (= b_3)$

$$A_4 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$=$ (equation) $\quad 1 \quad (= b_4)$

$$A_5 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$=$ (equation) $\quad 1 \quad (= b_5)$

Input:

```
5
SYMMETRIC_SPARSE
5 10
0 0 0.5
1 1 0.5
2 2 0.5
3 3 0.5
4 4 0.5
0 1 -0.25
0 2 -0.25
0 4 -0.25
1 3 -0.25
1 4 -0.25
5
SINGLETON
5 0 0 1
= 1
SINGLETON
5 1 1 1
= 1
SINGLETON
5 2 2 1
= 1
SINGLETON
5 3 3 1
= 1
SINGLETON
5 4 4 1
= 1
```

2.1 Sparse Symmetric Matrices

Suppose the matrix $A \in S_n$ has k nonzero entries in the upper triangle (a_{ij} with $i \le j$). Then the matrix is described by (NOTE: indices start at zero!)

SYMMETRIC_SPARSE

$n \; k$

i_1 j_1 $a_{i_1 j_1}$
$\vdots$
i_k j_k $a_{i_k j_k}$

It does not matter if $i \leq j$ or $i \geq j$. If several values are given for the same index pair then these values are summed up.

Example:

$$A = \begin{bmatrix} 0 & 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0 & 0.2 & 0 \\ 0.1 & 0 & 0 & 0 & 0 \\ 0 & 0.2 & 0 & 0.3 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

```
SYMMETRIC_SPARSE
5 3
0 2 0.1
1 3 0.2
3 3 0.3
```

2.2 Gram Matrices of Dense Matrices

Here, a Gram matrix has the form AA^T with $A \in \mathbb{R}^{n \times k}$ a dense matrix (k should be small to make this representation useful).

`GRAM_DENSE`
⟨Matrix⟩

The format of ⟨Matrix⟩ is explained in Section 2.9 below.

Example:

$$A = \begin{bmatrix} 1 & 0 & 0 & 2 & 0 \\ 0 & 9 & 0 & 0 & 12 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 4 & 0 \\ 0 & 12 & 0 & 0 & 16 \end{bmatrix}$$

```
GRAM_DENSE
5 2
1. 0.
0. 3.
0. 0.
2. 0.
0. 4.
```

2.3 Gram Matrices of Sparse Matrices

Matrices AA^T with $A \in \mathbb{R}^{n \times k}$ a sparse matrix.

`GRAM_SPARSE`
⟨Sparsemat⟩

The format of ⟨Sparsemat⟩ is explained in Section 2.9 below (NOTE: indices start at zero!).

Example:

$$A = \begin{bmatrix} 1 & 0 & 0 & 2 & 0 \\ 0 & 9 & 0 & 0 & 12 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 4 & 0 \\ 0 & 12 & 0 & 0 & 16 \end{bmatrix}$$

```
GRAM_SPARSE
5 2 4
0 0 1.
3 0 2.
1 1 3.
4 1 4.
```

2.4 Low Rank Matrices Formed by Two Dense Matrices

Matrices $AB^T + BA^T$ with $A, B \in \mathbb{R}^{n\times k}$ dense matrices.

```
LOWRANK_DENSE_DENSE
```
⟨Matrix A⟩
⟨Matrix B⟩

The format of ⟨Matrix⟩ is explained in Section 2.9 below.

Example:

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 3 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \end{bmatrix}$$

```
LOWRANK_DENSE_DENSE
5 1
0.
1.
0.
0.
0.
5 1
0.
0.
1.
2.
3.
```

2.5 Low Rank Matrices Formed by Sparse and Dense Matrices

Matrices $AB^T + BA^T$ with $A \in \mathbb{R}^{n\times k}$ a sparse matrix and $B \in \mathbb{R}^{n\times k}$ a dense matrix.

```
LOWRANK_SPARSE_DENSE
```
⟨Sparsemat A⟩
⟨Matrix B⟩

The format of ⟨Matrix⟩ and ⟨Sparsemat⟩ is explained in Section 2.9 below (NOTE: indices start at zero!).

Example:

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 3 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \end{bmatrix}$$

```
LOWRANK_SPARSE_DENSE
5 1 1
0 1 1.
5 1
0.
0.
1.
2.
3.
```

2.6 Low Rank Matrices Formed by Two Sparse Matrices

Matrices $AB^T + BA^T$ with $A, B \in \mathbb{R}^{n\times k}$ sparse matrices.

LOWRANK_SPARSE_SPARSE
⟨Sparsemat A⟩
⟨Sparsemat B⟩

The format of ⟨Sparsemat⟩ is explained in Section 2.9 below (NOTE: indices start at zero!).

Example:

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 3 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \end{bmatrix}$$

```
LOWRANK_SPARSE_SPARSE
5 1 1
0 1 1.
5 1 3
0 2 1.
0 3 2.
0 4 3.
```

2.7 Matrices with a Single Nonzero Element

A Matrix $A \in S_n$ with only one index pair i and j satisfying $a_{ij} = a_{ji} \neq 0$ may be described by (NOTE: indices start at zero!)

SINGLETON
$n\ i\ j\ a_{ij}$

Example:

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

```
SINGLETON
5 1 3 0.5
```

2.8 Dense Symmetric Matrices

This class is only useful for testing purposes, it is not recommended to use the spectral bundle method for dense problems. A dense symmetric matrix $A \in S_n$ is given by specifying the upper triangle row-wise (or the lower triangle column-wise).

SYMMETRIC_DENSE
n
$a_{11}\ a_{12}\ \ldots a_{1(n-1)}\ a_{1n}$
$a_{22}\ a_{23}\ \ldots a_{2n}$
$\vdots$
a_{nn}

Example:

$$A = \begin{bmatrix} 0.1 & 0.2 & 0.3 & 0.4 & 0.5 \\ 0.2 & 0.6 & 0.7 & 0.8 & 0.9 \\ 0.3 & 0.7 & 0.10 & 0.11 & 0.12 \\ 0.4 & 0.8 & 0.11 & 0.13 & 0.14 \\ 0.5 & 0.9 & 0.12 & 0.14 & 0.15 \end{bmatrix}$$

```
SYMMETRIC_DENSE
5
0.1  0.2  0.3  0.4  0.5
0.6  0.7  0.8  0.9
0.10 0.11 0.12
0.13 0.14
0.15
```

2.9 Building Blocks ⟨Matrix⟩ and ⟨Sparsemat⟩

Now we explain the format of the two basic matrix classes ⟨Matrix⟩ and ⟨Sparsemat⟩.

A ⟨Matrix⟩ $A \in \mathbb{R}^{n \times k}$ is specified by giving its elements row-wise.

$$\begin{array}{l} n\ k \\ a_{11}\ a_{12}\ \ldots a_{1k} \\ a_{21}\ a_{22}\ \ldots a_{2k} \\ \vdots \\ a_{n1}\ a_{n2}\ \ldots a_{nk} \end{array}$$

Example:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0.11 & 0.12 \\ 0 & 0 & 0 \\ 0.3 & 0 & 0 \\ 0 & 0.41 & 0 \end{bmatrix}$$

```
5 3
0.   0.   0.
0.   0.11 0.12
0.   0.   0.
0.3  0.   0.
0.   0.41 0.
```

A ⟨Sparsemat⟩ $A \in \mathbb{R}^{n \times k}$ is specified by giving its h nonzero elements in any order (NOTE: indices start at zero!).

$$\begin{array}{l} n\ k\ h \\ i_1\ j_1\ a_{i_1 j_1} \\ \vdots \\ i_h\ j_h\ a_{i_h j_h} \end{array}$$

If several values are given for the same index pair then these values are summed up.

Example:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0.11 & 0.12 \\ 0 & 0 & 0 \\ 0.3 & 0 & 0 \\ 0 & 0.41 & 0 \end{bmatrix}$$

```
5 3 4
1 1 0.11
1 2 0.12
3 0 0.3
4 1 0.41
```

3 The Algorithm

3.1 Verbal Description

In order to explain the meaning of the parameters to be described in Section 4, we have to introduce the algorithm in some detail. We start by extending the cost function of (1) to $\mathbb{R}^m$ by

$$f(y) := a\,\lambda_{\max}(C - \mathcal{A}^T y) + b^T y + \imath_Y(y),$$

where $\imath_Y$ denotes the indicator function whose value is 0 for $y \in Y$ and ∞ otherwise. Instead of minimizing f directly we work with sequences of simpler cutting surface models of f. We construct these model functions by the following two steps.

In the first step we minorize $a\,\lambda_{\max}(C - \mathcal{A}^T y)$. It is well known that, for $a \geq 0$,

$$a\lambda_{\max}(C - \mathcal{A}^T y) = \max\{\langle C - \mathcal{A}^T y, W\rangle : \operatorname{tr} W = a, W \succeq 0\}.$$

Therfore, any $W \in \mathcal{W} := \{W \succeq 0 : \operatorname{tr} W = a\}$ gives rise to a linear function $\langle C - \mathcal{A}^T \cdot, W\rangle$ minorizing the function $a\lambda_{\max}(C - \mathcal{A}^T \cdot)$. A subset $\widehat{\mathcal{W}} \subset \mathcal{W}$ yields, as the pointwise maximum of the linear functions, a convex function minorizing $a\lambda_{\max}(C - \mathcal{A}^T \cdot)$. `SBmethod` uses a subset of the form

$$\widehat{\mathcal{W}} := \{PVP^T + \alpha\overline{W} : \operatorname{tr} V + \alpha = a, V \succeq 0, \alpha \geq 0\}, \tag{3}$$

where P is a given orthonormal matrix of size $n \times r$ and $\overline{W} \in S_n^+$ is a given positive semidefinite matrix of trace 1 ($\operatorname{tr} \overline{W} = 1$). We refer to P as the *bundle*, the number of columns r of P as the *size* of the bundle, and to $\overline{W}$ as the *aggregate*; within a single iteration of the algorithm P and $\overline{W}$ may be regarded as constant but both will be updated at the end of each iteration.

In the second step we minorize $\imath_Y$ by a linear function $-\eta^T y$, where η is any element of the dual cone Y^* to Y, *i.e.*,

$$\begin{aligned} \eta \in Y^* &= \{\eta \in \mathbb{R}^m : \eta^T y \geq 0 \ \ \forall y \in Y\} \\ &= \{\eta \in \mathbb{R}^m : \eta_i = 0 \ \ \forall i \in J_=, \ \ \eta_i \geq 0 \ \ \forall i \in J_\geq, \ \ \eta_i \leq 0 \ \ \forall i \in J_\leq\}. \end{aligned}$$

Thus, for any fixed $W \in \mathcal{W}$ and $\eta \in Y^*$ we obtain a linear minorant of f by

$$f_{W,\eta}(y) := \langle C - \mathcal{A}^T y, W\rangle + (b - \eta)^T y \ \ \leq f(y) \qquad \forall y \in Y.$$

A subset $\widehat{\mathcal{W}} \subset \mathcal{W}$ together with a fixed $\eta \in Y^*$ yields a convex minorant of f,

$$f_{\widehat{\mathcal{W}},\eta}(y) := \max_{W \in \widehat{\mathcal{W}}} \langle C - \mathcal{A}^T y, W\rangle + (b - \eta)^T y \ \ \leq f(y) \qquad \forall y \in Y.$$

Since we will form $\widehat{\mathcal{W}}$ from accumulated local information, the model $f_{\widehat{\mathcal{W}},\eta}$ can be expected to be of reasonable quality only locally. Therefore we determine the next candidate as the minimizer of

$$\min_{y \in \mathbb{R}^m} \max_{\eta \in Y^*} f_{\widehat{\mathcal{W}},\eta}(y) + \frac{u}{2}||y - \hat{y}||_H^2, \tag{4}$$

where $\hat{y}$ is our current *center of stability* (the starting point or last successful iterate) and the *weight* u is a parameter that allows some indirect influence on the distance of the minimizer of (4) to $\hat{y}$. In the algorithm we restrict $H \in S_m^{++}$ to be a diagonal matrix, so that it is equivalent to a scaling of the primal constraints. H is usually the identity but the algorithm offers a scaling heuristic setting $h_{ii} = 1/\max\{1, \hat{y}_i^2\}$ for $i \in \{1, \ldots, m\}$. Instead of solving (4) we solve its dual,

$$\max_{W \in \widehat{\mathcal{W}}, \eta \in Y^*} \min_{y \in \mathbb{R}^m} \left\langle C - \mathcal{A}^T y, W \right\rangle + (b - \eta)^T y + \frac{u}{2} ||y - \hat{y}||_H^2. \tag{5}$$

The inner minimization over y is an unconstrained convex quadratic problem and can be solved explicitly for any choice of W and η,

$$y_{\min}(W, \eta) := \hat{y} + \frac{1}{u} H^{-1}(\mathcal{A}W - b + \eta). \tag{6}$$

Substituting this for y into (5) we obtain

$$\max_{W \in \widehat{\mathcal{W}}, \eta \in Y^*} \left\langle C - \mathcal{A}^T \hat{y}, W \right\rangle + (b - \eta)^T \hat{y} - \frac{1}{2u} ||\mathcal{A}W - b + \eta||_H^2. \tag{7}$$

In order to further reduce the complexity of solving this subproblem we solve it by a sequence of coordinatewise optimization steps. In particular, we first fix $\hat{\eta}$ and solve (7) over $W \in \widehat{\mathcal{W}}$ yielding an optimal W^+, then we fix W^+ and solve (7) over $\eta \in Y^*$ and iterate this if necessary. Optimizing over $W \in \widehat{\mathcal{W}}$ for fixed $\hat{\eta}$ results in a small dimensional ("small" depending on the bundle size r) convex quadratic semidefinite programming problem in V and α (see the definition of $\widehat{\mathcal{W}}$ (3)) that can be solved efficiently by interior point methods,

$$W^+ \in \operatorname*{Argmax}_{W \in \widehat{\mathcal{W}}} \left\langle C - \mathcal{A}^T \hat{y}, W \right\rangle + (b - \hat{\eta})^T \hat{y} - \frac{1}{2u} ||\mathcal{A}W - b + \hat{\eta}||_H^2. \tag{8}$$

See Helmberg and Rendl [2000]; Helmberg and Kiwiel [1999] for a description of the interior point code. Optimizing (7) over η for some fixed W is separable convex; the optimal argument $\eta_{\max}(W)$ is determined by

$$[\eta_{\max}(W)]_i = \begin{cases} \max\left\{0, -h_{ii} u \left[\hat{y} + \frac{1}{h_{ii} u}(\mathcal{A}W - b)\right]_i\right\} & i \in J_{\geq} \\ \min\left\{0, -h_{ii} u \left[\hat{y} + \frac{1}{h_{ii} u}(\mathcal{A}W - b)\right]_i\right\} & i \in J_{\leq} \\ 0 & i \in J_{=}. \end{cases} \tag{9}$$

Therfore we set

$$\eta^+ := \eta_{\max}(W^+). \tag{10}$$

Observe that η^+ also ensures feasibility and complementarity of $y_{\min}(W^+, \eta^+)$ (6),

$$y^+ := y_{\min}(W^+, \eta^+) \in Y \qquad \text{and} \qquad (\eta^+)^T y^+ = 0. \tag{11}$$

The pair (W^+, η^+) is, in general, not an optimal solution of (7) but a only an approximation. We regard this approximation as *not* sufficiently accurate if

$$f_{\widehat{\mathcal{W}},\eta^+}(y^+) - f_{W^+,\eta^+}(y^+) > \kappa_M[f(\hat{y}^k) - f_{W^+,\eta^+}(y^+)] \tag{12}$$

for a *model-precision parameter* $\kappa_M \in (0, \infty)$, *i.e.*, if the gap at y^+ between the model value $f_{\widehat{\mathcal{W}}^k,\eta^+}(y^+)$ and its linear minorant $f_{W^+,\eta^+}(y^+)$ is too big in comparison to the gap between the old function value $f(\hat{y}^k)$ and the linear minorant. In this case, the two coordinatewise steps are repeated, now fixing $\hat{\eta}$ to the new η^+.

Otherwise the solution of the model is considered sufficiently accurate and y^+ is the new candidate at which the function is evaluated. We do this by a Lanczos method (see also Section 3.3) which generates a sequence of normalized vectors v whose Ritz-values $v^T(C - \mathcal{A}^T y^+)v$ yield successively better estimates of $\lambda_{\max}(C - \mathcal{A}^T y^+)$ till either the function values turn out to be too high for sufficient decrease in objective value or the vectors have converged to an eigenvector of $\lambda_{\max}$. More formally, the Lanczos process continues to generate better and better $W_S := avv^T \in \mathcal{W}$ till

(a) $f(\hat{y}) - f_{W_S,\eta^+}(y^+) \leq \bar{\kappa}\,[f(\hat{y}) - f_{W^+,\eta^+}(y^+)]$, or

(b) $f_{W_S,\eta^+}(y^+) = f(y^+)$ and $f(\hat{y}) - f(y^+) \geq \kappa\,[f(\hat{y}) - f_{W^+,\eta^+}(y^+)]$.

where the *descent parameter* $\kappa \in (0, 1)$ controls the necessary progress relative to the gap $f(\hat{y}) - f_{W^+,\eta^+}(y^+)$ for accepting *descent steps* to y^+, and the *null step parameter* $\bar{\kappa} \in [\kappa, 1)$ controls the level for accepting a *null step*. At a descent step the algorithm moves the center of stability $\hat{y}$ to y^+, at a null step $\hat{y}$ remains unchanged. In both cases the model $\widehat{\mathcal{W}}$ is updated so that W_S and W^+ are both contained in $\widehat{\mathcal{W}}^+$ and the algorithm iterates.

The algorithm stops as soon as

$$f(\hat{y}) - f_{W^+,\eta^+}(y^+) \leq \varepsilon(|f(\hat{y})| + 1) \tag{13}$$

for a *termination precision parameter* $\varepsilon > 0$, *i.e.*, when the maximal progress of the next step ($f_{W^+,\eta^+}(y^+)$ may be regarded as a lower bound for this) is small in comparison to the absolute value of the function.

We now give the algorithm in detail, superscripts k are used to indicate iteration indices.

3.2 The Main Loop in Pseudo-Code

Algorithm 3.1 *(Spectral Bundle Method with Bounds)*

Input: $y^0 \in \mathbb{R}^n$, $\varepsilon > 0$, $\kappa \in (0, 1)$, $\bar{\kappa} \in [\kappa, 1)$, $\kappa_M \in (0, \infty]$, $\kappa_\eta \in [0, 1]$, *a weight* $u^0 > 0$.

1. *Set* $k = 0$, $\hat{y}^0 = y^0$, $\eta^0 = 0$, *compute* $f(y^0)$ *and* $\widehat{\mathcal{W}}^0$, *choose* $W^0 \in \widehat{\mathcal{W}}^0$.

2. *(Trial point finding). Compute* $\hat{\eta} = (1 - \kappa_\eta)\eta^k + \kappa_\eta \eta^k_{\max}(W^k)$, *see* (9).

   (a) Find W^+, η^+, y^+ *by* (8), (10), *and* (11) *in this sequence.*

(b) *(Stopping criterion). If* $f(\hat{y}^k) - f_{W^+,\eta^+}(y^+) \le \varepsilon(|f(\hat{y}^k)| + 1)$ *then* **stop**.

(c) *If* $f_{\widehat{\mathcal{W}}^k,\eta^+}(y^+) - f_{W^+,\eta^+}(y^+) > \kappa_M[f(\hat{y}^k) - f_{W^+,\eta^+}(y^+)]$ *then set* $\hat{\eta} = \eta^+$ *and* **goto** *(a).*

(d) *Set* $y^{k+1} = y^+$, $W^{k+1} = W^+$, $\eta^{k+1} = \eta^+$.

3. *(Descent test). Find* $W_S^{k+1} \in \mathcal{W}$ *such that either*

   (a) $f(\hat{y}^k) - f_{W_S^{k+1},\eta^{k+1}}(y^{k+1}) \le \bar{\kappa}\left[f(\hat{y}^k) - f_{W^{k+1},\eta^{k+1}}(y^{k+1})\right]$, *or*

   (b) $f_{W_S^{k+1},\eta^{k+1}}(y^{k+1}) = f(y^{k+1})$ *and* $f(\hat{y}^k) - f(y^{k+1}) \ge \kappa\left[f(\hat{y}^k) - f_{W^{k+1},\eta^{k+1}}(y^{k+1})\right]$.

   In case (a), set $\hat{y}^{k+1} = \hat{y}^k$ *(null step), otherwise set* $\hat{y}^{k+1} = y^{k+1}$ *(descent step).*

4. *(Weight updating). Choose a new weight* u^{k+1} *(not explained here).*

5. *(Model updating). Choose a* $\widehat{\mathcal{W}}^{k+1} \supset \left\{W^{k+1}, W_S^{k+1}\right\}$ *of the form* (3).

6. *Increase* k *by one and* **goto** *2.*

The following theorem holds for $\varepsilon = 0$ and $H^k = I$.

Theorem 3.2 *Helmberg and Kiwiel [1999]*
Either $\hat{y}^k \to \bar{y} \in \operatorname{Argmin} f$, *or* $\operatorname{Argmin} f = \emptyset$ *and* $\|\hat{y}^k\| \to \infty$. *In both cases* $f(\hat{y}^k) \downarrow \inf f$.

The efficiency of the algorithm is governed by the parameters controlling the eigenvalue computation in the Lanczos method and the parameters determining the size and composition of the bundle in $\widehat{\mathcal{W}}$. Therefore we will describe these two aspects briefly in the following.

3.3 Eigenvalue Computation

We implement step 3 of Algorithm 3.1 by computing, by means of the Lanczos method, an eigenvector v to the maximal eigenvalue of the matrix $C - \mathcal{A}^T y^k$ and setting $W_S^k = avv^T$.

The Lanczos process is an iterative method that generates, from a series of matrix vector multiplications, a partial tridiagonalization of the matrix. Each iteration, called a *Lanczos step*, increases the order of the tridiagonal matrix by one. The eigenvectors and eigenvalues of the tridiagonal matrix are used to generate approximations to the eigenvectors and eigenvalues of the original matrix; slightly deviating from the usual terminology we call these approximations *Lanczos vectors* and their *Ritz values*. See Golub and van Loan [1989] for an introduction and Parlett [1998]; Saad [1992] for a detailed description of the Lanczos method.

Our algorithm uses complete orthogonalization and restarts the Lanczos process after n_L Lanczos steps using the Lanczos vector corresponding to the maximal eigenvalue of the tridiagonal matrix as new starting vector. Before each restart we check the null step criterion of step 3(a) of Algorithm 3.1.

A large parameter n_L enhances the convergence of the Lanczos process, but also increases computation time significantly (as well as memory consumption) because of the complete orthogonalization of the vectors. If the matrix vector multiplication is computationally expensive, then we prefer to choose a large parameter n_L, say 150.

If the matrix vector multiplication is very cheap (extremely sparse matrices), then it may be worth to apply at each Lanczos step a spectral transformation to the matrix. The code offers the possibility to compute a Chebychev polynomial transformation by a series of matrix vector multiplications. A parameter n_C allows to specify the degree of the Chebychev polynomial (we use odd degrees only to preserve the ordering of the eigenvalues outside the Chebychev interval). In our experience this parameter should by either 0 (to be read as no spectral transformation) or above 21. If $n_C > 0$ each restart of the Lanczos process requires roughly $n_L \cdot n_C$ matrix vector multiplications. Therefore, we recommend not to choose n_L very large in this case, *e.g.*, between 20 and 50.

To the best of our knowledge no satisfactory heuristics are known for choosing the parameters n_L and n_C in an automatic way. In lack of a reasonable approach, `SBmethod` provides the following very naive heuristic as default. Based on a rough estimate of the flops of the matrix vector operation it decides whether to use $n_C = 0$ or $n_C > 0$. If it decides in favor of $n_C = 0$ then (assuming n is sufficiently large) it sets $n_L = \min\{50\lfloor \#\text{restarts}/2 + 1\rfloor, 200\}$. Otherwise, for $n_C > 0$, it first computes a guess of the interval $[\lambda_{\max}, \lambda_{\min}]$ by ten Lanczos steps for block size two (the interval is needed to set up the Chebychev polynomial; starting vectors are generated randomly or from previous vectors). It then uses blocksize one and begins with $n_L = 15$ and $n_C = 20$ so as to quickly eliminate poor candidates at the end of this cycle by criterion 3(a) of Algorithm 3.1. After this second restart it continues with $n_L = 25$ and $n_C = \min\{20 + 10\lfloor \#\text{restarts}/2\rfloor, 200\}$.

3.4 Model Updating

The model $\widehat{\mathcal{W}}$ of (3) is completely determined by the choice of the bundle P and the aggregate $\overline{W}$. In updating P and $\overline{W}$ to P^+ and $\overline{W}^+$ we have to ensure that the new model $\widehat{\mathcal{W}}^+$ contains $\{W^+, W_S = vv^T\}$ (v denoting the new Lanczos vector), see step 5 of Algorithm 3.1. In `SBmethod` the construction of the new bundle P^+ for $\widehat{\mathcal{W}}^+$ from P of $\widehat{\mathcal{W}}$ and the new information obtained from Lanczos vectors is controlled by four parameters $n_K, n_{\min} \in \mathbb{N}_0$, $n_K \ge n_{\min}$ (maximum and minimum subspace dimension to keep), $n_A \in \mathbb{N}$ (maximum number of Lanczos vectors to add), and $t_a \in (0,1)$ (aggregation tolerance). These are used as follows.

Let $W^+ = PV^+P^T + \alpha^+\overline{W}$ denote the computed solution of (8) and let $V^+ = Q\Lambda Q^T$ be an eigenvalue decomposition of V^+ with $Q^TQ = I_r$ and $\Lambda = \operatorname{Diag}(\lambda_1, \ldots, \lambda_r)$ a diagonal matrix with $\lambda_1 \ge \lambda_2 \ge \cdots \ge \lambda_r$. For

$$r_1 = \begin{cases} 0 & \text{if } n_K = 0 \\ \max\{i \in \{1, \ldots, n_K\} : \lambda_i \ge t_a\lambda_1\} \cup \{n_{\min}\} & \text{otherwise} \end{cases}$$

let Q_1 contain the r_1 first columns of Q and Q_2 the last $r - r_1$ columns and let Λ_1 and Λ_2 denote the corresponding diagonal matrices. Furthermore, let n_L denote the number of

Lanczos vectors returned by the eigenvalue computation routine and let L be the matrix formed by the $\min(n_L, n_A)$ Lanczos vectors with largest Ritz-values. Then the new model $\widehat{\mathcal{W}}^+$ is determined by

$$\begin{aligned} P^+ &= \operatorname{orth}([PQ_1, L]) \\ \overline{W}^+ &= \frac{PQ_2\Lambda_2(PQ_2)^T + \alpha^+\overline{W}}{\operatorname{tr}\Lambda_2 + \alpha^+}, \end{aligned}$$

where orth() constructs a matrix whose columns form an orthonormal basis of the space spanned by the columns of the argument. This construction ensures $\widehat{\mathcal{W}}^+ \supset \{W^+, W_S\}$ (see Helmberg and Rendl [2000]).

Remark 3.3 *As shown in Helmberg and Rendl [2000] it is not necessary to store and update $\overline{W}$ itself, but only $\mathcal{A}\overline{W}$ and $\langle C, \overline{W}\rangle$. Note, however, that the matrix W^+ may be interpreted as an approximate primal solution X (and the η variables as primal slack variables). If full knowledge about W^+ is desired, then $\overline{W}$ should be stored explicitly and* `SBmethod` *offers this possibility.*

Again no reasonable guidelines are known for choosing these parameters. In the hope to make first experiments with the code easier, `SBmethod` offers a heuristic that adapts these parameters dynamically (in this case the input parameters should be understood as upper bounds on the possible values). It starts with $n_{\min} = 0$. In each iteration it adds up to n_A new vectors if their Ritz value is above a bound that is determined relative to the largest Ritz value and its gap to estimates of the Ritz values of the vectors already in P. P is then updated as explained above. As long as the accumulated time required for solving the quadratic semidefinite model is less than half the time spent in the eigenvalue routine, $n_{\min}$ is increased in each iteration by one; this rule is intended for situations where t_a is too restrictive and not enough columns are maintained in P. If, however, an iteration is encountered where $\alpha > \frac{3}{10}a$ and solving the augmented model takes ten times as long as computing the eigenvalues, then evidence is high that a sufficiently large bundle would be inefficient and, in this case, n_K is set to zero and n_A to seven for all further iterations. This heuristic cannot be expected to deliver optimal choices for the bundle parameters; In addition, its dependence on time accounting information has the undesirable effect that running the same algorithm with the same input twice may result in different output (even though convergence to an [existing] optimal solution is guaranteed). Yet, it should perform reasonably well if no better setting from related problems is at hand.

4 Options and Parameters

When `sb` is called with the help option `-h` or an unknown option it prints (an error message and) a short description of all options (see below) and exits:

```
usage: sb [options]        default input for problem description: stdin
```

```
options:  [-te Real] [-tt Integer] [-td Integer]
    [-bu Integer] [-bk Integer] [-ba Integer] [-bm Integer] [-bt Real]
    [-ln Integer] [-lc Integer]
    [-sh 0/1] [-si 0/1] [-sc 0/1]
    [-k Real] [-ki Real] [-km Real] [-ke Real] [-e Integer] [-c 0/1]
    [-u Integer] [-uk Real] [-uf Real] [-umin Real] [-umax Real]
    [-f filename] [-fp filename] [-fs filename]
    [-oy filename] [-ol filename] [-om filename] [-oP filename]
    [-oas filename] [-oad filename] [-oXs filename] [-oXd filename]
    [-oprob filename]
    [-op 0/1] [-ob 0/1] [-ot Integer]
    [-rf filename]
  (for repeated or contradicting options the last specification is used)

termination paramters:
  -te ... relative precision (termination epsilon) [default=1e-5]
  -tt ... timelimit in seconds [default=-1 for no limit]
  -td ... maximum number of descent steps [default=-1 for no limit]
bundle parameters:
  -bu ... method for bundle update [default heuristic=0]
  -bk ... maximum number of vectors kept [default=45]
  -ba ... maximum number of vectors added [default=10]
  -bm ... maximum for minimum number of vectors kept [default=30]
  -bt ... relative aggregation threshold [default=.01]
Lanczos eigenvalue computation:
  -ln ... number of Lanczos steps per Lanczos restart [default=-1]
  -lc ... degree of Chebychev polynomial in matrix multiplication [default=-1]
starting point and scaling heuristics:
  -sh ... use the starting point heuristic [default=0 no]
  -si ... scale constraint matrices to norm one on input [default=0 no]
  -sc ... use a scaling heuristic during computation [default=0 no]
step acceptance/rejection parameters:
  -k  ... factor for descent steps [0.1]
  -ki ... factor for inexact null steps [0.1]
  -km ... model precision (for inequalities only) [default=.6]
  -ke ... convex combination factor for guessing new multipliers [1.]
  -e  ... compute 'number' exact eigenvalue(s) for null steps [default=0 no]
  -c  ... use cutval instead of linval for acceptance criteria [default=0 no]
weight updating:
  -u  ... choice of u: 0 Kiwiel/1 Helmberg/4 HelmbergKiwiel [4]
  -uk ... factor for criterion for decreasing u [0.5]
  -uf ... fix value of u for entire algorithm
  -umin.. min value of u for entire algorithm, <=0 for default bound
  -umax.. max value of u for entire algorithm, <=0 for default bound
input files:
  -f  ... file for problem description
  -fp ... parameter file (format: list of command line options)
  -fs ... starting point file (feasible!), format: m 1 y11 y21 ... ym1
output files:
  -oy ... store best y at termination in filename
  -ol ... store computed Lanczosvalues/vectors of this y in filename
```

```
  -om ... store last Lagrange multipliers for y in filename
  -oXs... store sparse (pattern of cost matrix) approx of primal X in filename
  -oXd... store dense approximation of primal X in filename
  -oP ... store eigenvalues/vectors of last QSDP solution in filename
  -oas... store sparse aggregate matrix of last solution in filename
  -oad... store aggregate matrix of last solution in filename
  -oprob..write problem description to file
detail of log output:
  -ob ... write log info for bundle method [default=0 no]
  -op ... write log info for problem specific routines [default=0 no]
  -ot  ... termination: output at 0 final precision/1 precision levels [0]
resume file name:
  -rf ... name of the 'save'-file for resuming in case of interruption
or
  sb -resume filename
      ... resumes an old process that has been terminated prematurely
          filename has to refer to the corresponding 'save'-file

   SIGTERM  (15)   save resume info, output summary and option files, exit
   SIGUSR1  (10)   output summary and option files, exit
   SIGUSR2  (12)   output summary and option files and continue
   SIGKILL  (9)    kill without any further output
```

In order to facilitate locating the description of an option I stick to this order in the following. Yet I would like to emphasize that the order is not related to the importance of the parameters. In my experience the most important parameters are, in this sequence,

1. the relative precision for termination (`-te`, Section 4.1.1),

2. the size of the bundle (the number of columns in P) controlled by setting the maximum number of vectors kept (`-bk`, Section 4.2.2) and the maximum number of vectors added (`-ba`, Section 4.2.3) (sometimes it is helpful to also specify a minimum number of vectors to keep,`-bm`, Section 4.2.4).

3. the parameters guiding the eigenvalue computation by the Lanczos method, *i.e.*, the number of Lanczos steps within each restarting cycle (`-ln`, Section 4.3.1) and the degree of the Chebychev polynomial applied to the matrix for each matrix-vector multiplication (`-lc`, Section 4.3.2).

4. unfortunately, the time limit in seconds (`-tt`, Section 4.1.2).

We now explain the options in the sequence given by the code.

4.1 Termination Parameters

4.1.1 `[-te Real]`

specifies the relative precision ε required for termination, see (13) and Algorithm 3.1, step 2(b), page 10, for the stopping criterion.

4.1.2 `[-tt Integer]`

specifies the number of seconds (measured in user time) after which the algorithm stops the first time it checks the termination criterion (Algorithm 3.1, step 2(b), page 10). A negative number is interpreted as no time restriction; this is also the default.

4.1.3 `[-td Integer]`

specifies an upper bound on the number of descent steps. If this number is reached, the algorithm terminates. A negative number is interpreted as no bound; this is also the default.

4.2 Bundle Parameters

For an overview of the bundle updating process see Section 3.4.

4.2.1 `[-bu Integer]`

specifies the method used for updating the bundle. The default choice zero uses the heuristic described at the end of Section 3.4, the value one is used for the version described in in the beginning of Section 3.4, without dynamic update of n_K, n_A and $n_{\min}$.

4.2.2 `[-bk Integer]`

gives the maximum number n_K of columns to be kept in the bundle P (see Section 3.4).

4.2.3 `[-ba Integer]`

gives the maximum number n_A of new Lanczos vectors to be added to the bundle P (see Section 3.4).

4.2.4 `[-bm Integer]`

gives $n_{\min}$, the minimum number of columns kept in the bundle P (see Section 3.4). The number is less or equal to n_K (set by `-bk`, Section 4.2.2). The algorithm keeps $n_{\min}$ columns in P even if the aggregation tolerance t_a (set by `-bt`, Section 4.2.5) would lead to fewer columns. This parameter may come in handy if the size of the eigenvalues of V^+ (see Section 3.4) decreases too fast. In the case of the default dynamic parameter update rule `-bu 0` (see Section 4.2.1) the number only acts as an upper bound on the actual minimum number.

4.2.5 `[-bt Real]`

specifies the relative aggregation tolerance t_a for keeping and aggregating columns (see Section 3.4). The default value is 0.01.

4.3 Lanczos Eigenvalue Computation

For a rough description of the Lanczos algorithm and its parameters see Section 3.3.

4.3.1 `[-ln Integer]`

specifies the number n_L of Lanczos steps before the algorithm is restarted (see Section 3.3).

My current implementation of the Lanczos method uses complete orthogonalization, so a large number of steps does not only consume more memory, it is also computationally expensive. On the other hand, for a large parameter convergence is significantly faster. Thus, if matrix-vector multiplications are very expensive, a value of up to 200 may be worth trying (consult Golub and van Loan [1989] for an introduction and Saad [1992]; Parlett [1998] for an in-depth treatment of Lanczos methods).

4.3.2 `[-lc Integer]`

specifies the degree n_C of the Chebychev polynomial applied to the matrix in each matrix-vector multiplication (see Section 3.3). This spectral transformation helps to enlarge the spectral separation (the gap between largest and second largest eigenvalue) in order to speed up convergence of the Lanczos method. Since it requires *degree* many matrix-vector multiplications it is of use only if matrix-vector multiplications are cheap in comparison to the dense linear algebra employed inside the Lanczos method (in principle the Lanczos method produces the best polynomial for a given number of iterative matrix-vector multiplications and, thus, the fastest convergence).

4.4 Starting Point and Scaling Heuristics

4.4.1 `[-sh 0/1]`

When this option is set (value 1, this is the default), then the algorithm initially performs up to five steepest descent steps as long as the gap between maximal and second largest eigenvalue is "large". The stepsize is determined by linearly interpolating the increase of the second smallest eigenvalue. This can be regarded as a starting point heuristic.

4.4.2 `[-si 0/1]`

When this option is set (value 1; default is 0), then the constraints are scaled on input so that the constraint matrices have Frobenius norm 1. In some cases this may yield a better scaling of the problem.

4.4.3 `[-sc 0/1]`

When this option is set (value 1; default is 0), the algorithm employs a simple diagonal scaling heuristic. Essentially, the y_i with $|y_i| > 1$ are scaled to $|y_i| = 1$, see the description of H after (4) on page 8.

4.5 Step Acceptance/Rejection Parameters

4.5.1 [-k Real]

This sets the factor $\kappa \in (0,1)$ for accepting descent steps in step 3(b) of Algorithm 3.1 on page 10. The default value of 0.1 should be fine.

4.5.2 [-ki Real]

This sets the factor $\bar{\kappa} \in [\kappa, 1)$ for accepting null steps without computation of the exact maximal eigenvalue, see step 3(a) of Algorithm 3.1 on page 10. The default value of $\bar{\kappa} = \kappa$ should be fine. To force exact eigenvalue computations for null steps, use option `-e` (Section 4.5.5).

4.5.3 [-km Real]

This sets the factor $\kappa_M \in (0, \infty]$ for accepting an inexact model value, see step 3(c) of Algorithm 3.1 on page 10 and (12) on page 10. A small value requires high accuracy. A large value reduces the number of inner iterations (without endangering convergence), but may lead to unnecessary additional function evaluations. This cannot happen if $\kappa_M < 1 - \kappa$. The default value of 0.6 should be fine.

4.5.4 [-ke Real]

This sets the factor $\kappa_\eta \in [0,1]$ for computing a new initial guess $\hat{\eta}$ after a descent step in step 2 of Algorithm 3.1 on page 10. It forms a convex combination of the old η^k and the new maximizing choice $\eta^k_{\max}(W^k)$ (see (9)) for the new $\hat{y}^k$. For a null step, $\hat{y}^k = \hat{y}^{k-1}$, both are equal and the result is independent of κ_η. The current default setting $\kappa_\eta = 1$ (take the new value) should be fine.

4.5.5 [-e Integer]

This option specifies the number of exact eigenvalues to be computed at each evaluation and admits inexact evaluation for null steps if the value is zero (see step 3(a) of Algorithm 3.1 on page 10). If the value is at least one then inexact evaluation is switched off. Although the maximal eigenvalue is among these eigenvalues with probability one, it may happen that not all eigenvalues are largest possible (the starting vectors for additional eigenvalues are no longer random in general). The default value is zero.

4.5.6 [-c 0/1]

If this option is set (value 1, default is 0) then the exact value of the cutting model $f_{\widehat{W}^k,\eta^+}(y^+)$ is used instead of the linear minorant $f_{W^+,\eta^+}(y^+)$ in the stopping criterion of step 2(b) and the descent test of step 3 of Algorithm 3.1 on page 10. In the case of sign constraints on y the value of the cutting model is usually somewhat larger than the latter value. The option seems to be of little relevance.

4.6 Weight Updating

4.6.1 `[-u Integer]`

specifies one of several rules for updating the weight u in step 4 of Algorithm 3.1 on page 10, see also (4) on page 8 and the explanations thereafter. If an option `-uf` (Section 4.6.3) is specified afterwards, then this option is ignored.

In the following g is the initial subgradient, $g = b - a\mathcal{A}vv^T$, with v a normalized eigenvector to the maximal eigenvalue in the starting point.

0 rule of Kiwiel [1990] with starting value $u_0 = ||g||$
1 rule of Helmberg with starting value $u_0 = ||g||/\sqrt{m}$
2 rule of Helmberg with starting value $u_0 = ||g||$
3 rule of Kiwiel [1990] with starting value $u_0 = ||g||/\sqrt{m}$
4 rule of Kiwiel [1990] with level updates, $u_0 = ||g||$
5 rule of Kiwiel [1990] with level updates, $u_0 = ||g||/\sqrt{m}$

The level updates are described in Helmberg and Kiwiel [1999], but the "Helmberg"-rule has not appeared (and most likely will not appear) in the literature. It is a modification of the rule of Kiwiel [1990] that is a bit more aggressive in decreasing u, but less likely to increase u in a series of null steps. My personal favorite is 4 but sometimes it increases u too much which may lead to premature termination. In this case, rule 1 might be an alternative, or even setting an upper bound with `-umax`, see Section 4.6.5.

All current strategies seem to run astray if the starting point is bad, *i.e.*, the maximal eigenvalue is well separated from the remaining eigenvalues (consider, *e.g.*, starting the spectral bundle method on the Lovász ϑ function with $C = ee^T$ and all variables zero). In this case the u-value is constantly decreased in the first few iterations and usually does not increase sufficiently afterwards. The starting point heuristic `-sh` (Section refS-sh) should help to skip this initial phase and was included for this very purpose.

4.6.2 `[-uk Real]`

We call this factor $\kappa_R \in (\kappa, 1)$ (it is called m_R in Kiwiel [1990]) or $\kappa_R \in (\kappa, 1)$. It determines whether after a descent step and some additional conditions the weight u is decreased in step 4 of Algorithm 3.1 on page 10. The central condition (see Kiwiel [1990] for a precise description and additional restrictions) is that after a descent step model and function value are close in the sense of

$$f(\hat{y}^k) - f(y^{k+1}) \geq \kappa_R[f(\hat{y}^k) - f_{W^{k+1},\eta^{k+1}}(y^{k+1})].$$

The default value of 0.5 should be fine.

4.6.3 [-uf Real]

This option fixes the weight u at the given value for the entire algorithm. If an option -u (Section 4.6.1) is specified afterwards, then this option is ignored.

4.6.4 [-umin Real]

This option sets a lower bound on the weight u for the entire algorithm and any choice of a dynamic weight updating strategy. A value less or equal to zero results in the default choice.

4.6.5 [-umax Real]

This option sets an upper bound on the weight u for the entire algorithm and any choice of a dynamic weight updating strategy. A value less or equal to zero results in the default choice.

4.7 Input Files

4.7.1 [-f filename]

specifies the file from where the problem description should be read (see Section 2 for the format description). If this option is not set, the description is read from standard input.

4.7.2 [-fp filename]

specifies a file that contains personal preferences for standard parameter settings. Simply collect your favorite command line options in an ASCII-text file in arbitrary order and include it with this option. Any options specified afterwards will override these settings.

4.7.3 [-fs filename]

specifies a file that contains the starting point. The starting point must be a vector given in the format ⟨Matrix⟩ (see Section 2.9) and must be feasible! If no starting point is specified the algorithm starts at zero.

4.8 Output Files

4.8.1 [-oy filename]

specifies a file for writing the center of stability $\hat{y}$ at termination in step 2(b) of Algorithm 3.1 on page 10. Without this option, $\hat{y}$ is lost on termination.

4.8.2 `[-ol filename]`

specifies a file for writing the Lanczos values and Lanczos vectors corresponding to the center of stability $\hat{y}$ at termination in step 2(b) of Algorithm 3.1 on page 10. Without this options, these values are lost on termination.

4.8.3 `[-om filename]`

specifies a file for writing the multipliers η^+ at termination in step 2(b) of Algorithm 3.1 on page 10. Without this option, η^+ is lost on termination.

4.8.4 `[-oXs filename]`

specifies a file for writing a sparse subset of matrix elements of W^+ (an approximation to the primal solution matrix X) at termination in step 2(b) of Algorithm 3.1 on page 10. The sparsity pattern is taken from the cost matrix and the output format is as used in the description of `SYMMETRIC_SPARSE`, Section 2.1. This option implies that the aggregate $\overline{W}$ is also updated in sparse form and thus induces some overhead per iteration. If the cost matrix is not sparse, the option switches automatically to `-oXd` (Section 4.8.5) using the same filename. Without this option and without `-oas` (Section 4.8.7) the aggregate $\overline{W}$ is not stored in sparse form and the data is not available.

4.8.5 `[-oXd filename]`

specifies a file for writing the matrix W^+ (an approximation to the primal solution matrix X) at termination in step 2(b) of Algorithm 3.1 on page 10. The output format is dense symmetric as used in the description of `SYMMETRIC_DENSE`, Section 2.8. This option implies that the aggregate $\overline{W}$ is also updated in dense form which may slow down the algorithm substantially and may entail a significant increase in memory consumption. Without this option and without `-oad` (Section 4.8.8) (or possibly `-oas` for dense cost matrices, see Section 4.8.7) the aggregate $\overline{W}$ is not stored in dense form and the data is not available.

4.8.6 `[-oP filename]`

specifies a file for writing the eigenvalue and eigenvector matrices diag(Λ) and P of the matrix $W^+ = P\Lambda P^T + \alpha^+\overline{W}$ at termination in step 2(b) of Algorithm 3.1 on page 10. The output format is in dense ⟨Matrix⟩ form (see Section 2.9) in this sequence. This information always exists (see `-bk`, Section S-bk) but is lost on termination without this option.

4.8.7 `[-oas filename]`

specifies a file for writing a sparse subset of matrix elements of $\overline{W}$ and its multiplier α^+ at termination in step 2(b) of Algorithm 3.1 on page 10 ($W^+ = PV^+P^T\alpha^+\overline{W}$). The sparsity pattern is taken from the cost matrix and the output format is as used in the description of SYMMETRIC_SPARSE, Section 2.1. This option implies that the aggregate $\overline{W}$ is updated in sparse form and thus induces some overhead per iteration. If the cost matrix is not sparse, the option switches automatically to -oad (Section 4.8.8) using the same filename. Without this option and without -oXs (Section 4.8.4) the aggregate $\overline{W}$ is not stored in sparse form and the data is not available.

4.8.8 [-oad filename]

specifies a file for writing the dense aggregate $\overline{W}$ and its multiplier α^+ at termination in step 2(b) of Algorithm 3.1 on page 10 ($W^+ = PV^+P^T\alpha^+\overline{W}$). The output format is dense symmetric as used in the description of SYMMETRIC_DENSE, Section 2.8. This option implies that the aggregate $\overline{W}$ is updated in dense form which may slow down the algorithm substantially and may entail a significant increase in memory consumption. Without this option and without -oXd (Section 4.8.5) (or possibly -oas for dense cost matrices, see Section 4.8.7) the aggregate $\overline{W}$ is not stored in dense form and the data is not available.

4.8.9 [-oprob filename]

specifies a file for writing the problem description. In the current setting this is useful only for debugging purposes.

4.9 Detail of Log Output

4.9.1 [-ob 0/1]

flag for printing detailed log information on the progress of the spectral bundle algorithm, switched on with 1 and off with 0. This is not intended for general use. If in trouble, please send me the problem description, your parameter files, and a log file with this option and -op (Section 4.9.2) switched on. I will do my best, but I can give no guarantee on my reply time nor on success.

4.9.2 [-op 0/1]

flag for printing detailed log information on the eigenvalue computation, switched on with 1 and off with 0. This is not intended for general use. If in trouble, please send me the problem description, your parameter files, and a log file with this option and -ob (Section 4.9.1) switched on. I will do my best, but I can give no guarantee on my reply time nor on any success.

4.9.3 [-ot Integer]

specifies the routine for checking termination. Value 0 gives a summary output only at termination, value 1 provides intermediate summaries at the respective first iteration when the next precision level is reached with respect to the termination criterion of step 2(b) of Algorithm 3.1 on page 10. The precision levels are $\varepsilon = 10^{-2}$, $5{\cdot}10^{-3}$, 10^{-3}, ...

4.10 Resuming after SIGTERM

4.10.1 `[-rf filename]`

specifies a file for writing a complete dump of all variables and accounting information for continuing some later time. These files may get huge, so be sure to have an appropriate file size limit installed on your system. The saving process is invoked by the first termination check after sending SIGTERM to the job. To resume, execute `sb -resume filename` without any other parameters (see next Section 4.10.2).

4.10.2 `[-resume filename]`

specifies a file that contains a dump of all variables as created by `-rf` (see previous Section 4.10.1) and may only be used in the form `sb -resume filename` without any parameters. It then continues the run saved in this file.

4.11 Signals

4.11.1 `SIGTERM`

Sending SIGTERM to `sb` (*e.g.*, by executing `kill %job`) results in saving a default resume file named `sb_resume.dat` at the first termination check after the job received the signal. Furthermore the code prints a summary of the performance so far to standard out and writes all information requested by options to the appropriate files.

4.11.2 `SIGUSR1`

Upon the first termination check after receiving the signal SIGUSR1 (*e.g.*, after `kill -10 %job`), the code prints a summary of the performance so far to standard out, writes all information requested by options to the appropriate files, and *exits*.

4.11.3 `SIGUSR2`

Upon the first termination check after receiving the signal SIGUSR2 (*e.g.*, after `kill -12 %job`), the code prints a summary of the performance so far to standard out, writes all information requested by options to the appropriate files, and *continues*.

4.11.4 `SIGKILL`

Since this signal cannot be caught, the process may be terminated without any output by means of `kill -12 %job`.

5 Explanation of the Output

The basic structure of the log output of `sb` is

1. actual parameter settings,
2. the time the clock is started (after the problem has been read),
3. problem dimensions n and m,
4. initial eigenvalue and objective value (line `endit 0`)
5. one line per descent step,
6. Ritz-values obtained in the eigenvalue computation for the last serious step before termination
7. final line repeated with "`>>>`" in front
8. some time accounting information and possibly warnings
9. the time the clock is stopped (before writing output files)

We illustrate this by means of the following example.

```
SBmethod version 1.1, Copyright (C) 2000 Christoph Helmberg
SBmethod comes with ABSOLUTELY NO WARRANTY

Parameter settings:
-te : termeps   = 1e-05
-tt : timelimit = -1 (no timelimit)
-td : maxiter   = -1 (no limit on descent steps)
-bu : bundleupd = 0 -> heuristic choice of bundle size, uses dual cost elimination
-bk : maxkeepv  = 45
-ba : maxaddv   = 10
-bk : minkeepv  = 30
-bt : aggregtol = 0.01
-ln : lanczosst = -1
-lc : chebits   = -1
-sh : usestarth = 1
-si : initialsc = 0
-sc : do_scaling= 0
-k  : kappa     = 0.1
-ki : bar kappa = 0.1
-km : kappa_M   = 0.6
-ke : kappa_eta = 1
-e  : exacteigs = 0
-c  : usecutval = 0
-u  : u method  = 4 -> Kiwiel90 with  bar f_k^*
-uk : kappa_R   = 0.5
-umin: umin     = -1
-umax: umax     = -1
-f  : prob-file =  stdin
-fs : start-y   =  0-vector
-ot : terminat  = 0 -> Kiwiel90
elapsed time: 00:00:00.00   ----   Wed Oct  4 23:51:10 2000


dim=800  nr_constraints=800

00:00:00.27 endit  0   1  0.86545 (692.36386)
00:00:00.42 endit  1   3   171.   46.5  0.84762 675.39134 (678.09842)
00:00:00.51 endit  2   4   171.   36.7  0.83987 668.20836 (671.89476)
```

```
00:00:00.63 endit  3   5   128.   29.6  0.82994 662.56035 (663.94839)
00:00:00.72 endit  4   6   37.9   24.0  0.82732 647.15163 (661.85486)
00:00:00.82 endit  5   7   37.9   19.7  0.81539 640.15321 (652.31038)
00:00:00.96 endit  6   8   37.9   16.5  0.81150 637.02990 (649.20232)
00:00:01.12 endit  7   9   37.9   13.9  0.80448 634.76097 (643.58574)
00:00:01.48 endit  8  11   19.0   9.40  0.79801 631.78281 (638.40834)
00:00:01.90 endit  9  13   19.0   6.93  0.79263 630.53142 (634.10560)
00:00:02.30 endit 10  15   19.0   5.31  0.79197 629.64586 (633.57903)
00:00:03.06 endit 11  18   19.0   3.88  0.78983 629.62585 (631.86629)
00:00:03.63 endit 12  20   19.0   3.03  0.78853 629.38053 (630.82206)
00:00:04.87 endit 13  25   9.49   2.01  0.78772 629.27400 (630.17604)
00:00:06.03 endit 14  30   9.49   1.52  0.78753 629.17847 (630.02066)
00:00:07.02 endit 15  34   9.49   1.20  0.78728 629.11700 (629.82706)
00:00:07.94 endit 16  37   9.49  0.992  0.78714 629.08999 (629.71024)
00:00:08.75 endit 17  40   9.49  0.857  0.78704 629.07695 (629.62960)
00:00:10.14 endit 18  46   9.49  0.630  0.78687 629.11406 (629.49898)
00:00:10.97 endit 19  50   9.49  0.502  0.78679 629.12518 (629.43186)
00:00:11.40 endit 20  52   9.49  0.459  0.78675 629.11969 (629.39954)
00:00:12.64 endit 21  59   9.49  0.385  0.78664 629.13792 (629.31373)
00:00:13.30 endit 22  62   9.49  0.340  0.78660 629.13459 (629.27610)
00:00:15.08 endit 23  74   9.49  0.276  0.78657 629.15129 (629.25767)
00:00:16.83 endit 24  88   4.74  0.169  0.78654 629.15146 (629.23065)
00:00:19.05 endit 25 105   4.74  0.121  0.78652 629.15589 (629.21896)
00:00:20.98 endit 26 120   4.74 0.0992  0.78651 629.15813 (629.21066)
00:00:23.55 endit 27 139   4.74 0.0768  0.78649 629.16005 (629.19464)
00:00:26.29 endit 28 160   4.74 0.0634  0.78649 629.16228 (629.18990)
00:00:29.01 endit 29 182   4.74 0.0541  0.78647 629.16336 (629.17826)
00:00:35.79 endit 30 237   4.74 0.0517  0.78647 629.16510 (629.17256)
00:00:41.35 endit 31 272   4.74 0.0502  0.78646 629.16528 (629.17157)
00:00:43.31 endit 32 281   4.74 0.0494  0.78646 629.16492 (629.17069)
00:00:43.35 endit 33 281   4.74 0.0508  0.78646 629.16445 (629.17069)
matrix 1 (10,1)
columns 0 to 0
 0.78646336
 0.78646155
 0.78645772
 0.78644290
 0.78643372
 0.78641804
 0.78632739
 0.78597501
 0.78581918
 0.78542066
>>>00:00:43.35 endit 33 281   4.74 0.0508  0.78646 629.16445 (629.17069)
  eigvaltime=00:00:22.78 reltime=52.55 maxPrank=19
  qspsolves  :  281  2930 00:00:07.66 00:00:08.61
  qspresolves:    0     0 00:00:00.00 00:00:00.00
  augmod=00:00:17.25  eval=00:00:24.24  update=00:00:01.49  choose=00:00:00.09
termination status: 1, relative precision criterion satisfied
elapsed time: 00:00:43.35   ----   Wed Oct  4 23:52:37 2000
```

We explain the entries of the following sample line

```
00:00:26.29 endit 28 160   4.74 0.0634  0.78649 629.16228 (629.18990)
```

`00:00:26.29`: User time since the clock was started in hours:minutes:seconds.hundreds.

`endit` if the log includes options `-ob` or `-op` (sections 4.9.1 and 4.9.2), then `endit` is convenient for `grep`.

28 number of completed descent steps.

`160` number of function evaluations (currently equal to the number of iterations of the bundle method plus one for the initial computation; the start heuristic is not contained in the count).

`4.74` value of the weight u (not shown in line `endit 0`).

`0.0634` Euclidean norm of the subgradient $g = b - \eta - \mathcal{A}W$ that gave rise to this candidate via $y^+ = \hat{y} - g/u$ (not shown in line `endit 0`).

`0.78649` maximal eigenvalue at this descent step

`629.16228` model value used (see option `-c`, Section 4.5.6) (not shown in line `endit 0`).

`(629.18990)` objective value.

Finally, consider the summary giving the accounting information.

```
  eigvaltime=00:00:22.78 reltime=52.55 maxPrank=19
  qspsolves  :  281  2930 00:00:07.66 00:00:08.61
  qspresolves:    0     0 00:00:00.00 00:00:00.00
  augmod=00:00:17.25  eval=00:00:24.24  update=00:00:01.49  choose=00:00:00.09
termination status: 1, relative precision criterion satisfied
```

`eigvaltime` time spent in the Lanczos routine (the time for building the matrix is not included).

`reltime` percentage of `eigvaltime` with respect to overall computation time.

`maxPrank` gives the maximum number of columns used in P for model computations (not during updating).

`qspsolves` This line gives, in this sequence: total number of calls to the interior point code for solving the quadratic semidefinite subproblem, total number of interior point iterations over all calls, overall time needed to compute the cost coefficients of the quadratic semidefinite subproblem, overall time spent in actually solving the quadratic semidefinite subproblem for precomputed cost coefficients.

`qspresolves` This line is nonzero only if there were additional updates of η in the inner loop (see Helmberg and Kiwiel [1999] and option `-km`, Section 4.5.3). The line displays, in this sequence: total number of calls to the interior point code for solving the quadratic semidefinite subproblem with restart heuristic, total number of interior point iterations over all such calls, overall time needed to update the linear cost coefficients of the quadratic semidefinite subproblem, overall time spent in actually solving the quadratic semidefinite subproblem for updated cost coefficients. (Every 50 iterations, within a direct sequence of inner iterations, the full coefficient computation and interior point code without restarting is called).

`augmod` time spent in solving the augmented model, *i.e.*, in computing the next candidate.

`eval` time spent in function evaluation.

`update` time spent in updating the bundle.

`choose` time spent in choosing the weight u.

`termination status` gives the termination code and its translation.

References

Goemans, M. X. (1997). Semidefinite programming in combinatorial optimization. *Math. Programming*, 79:143–161.

Golub, G. H. and van Loan, C. F. (1989). *Matrix Computations*. The Johns Hopkins University Press, 2^{nd} edition.

Helmberg, C. (2000). Semidefinite programming for combinatorial optimization. Habilitationsschrift TU Berlin, ZIB-Report ZR-00-34, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Takustraße 7, 14195 Berlin, Germany.

Helmberg, C. and Kiwiel, K. C. (1999). A spectral bundle method with bounds. ZIB Preprint SC-99-37, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Takustraße 7, 14195 Berlin, Germany.

Helmberg, C. and Rendl, F. (2000). A spectral bundle method for semidefinite programming. *SIAM J. Optim.*, 10(3):673–696.

Kiwiel, K. C. (1990). Proximity control in bundle methods for convex nondifferentiable minimization. *Math. Programming*, 46:105–122.

Parlett, B. N. (1998). *The Symmetric Eigenvalue Problem*. SIAM.

Saad, Y. (1992). *Numerical Methods for Large Eigenvalue Problems*. Halsted Press, New York.

A General Notation

$\mathbb{N}$ ($\mathbb{N}_0$)	nonnegative integers (with zero)
$\mathbb{R}$	real numbers
$\mathbb{R}^n$	real column vector of dimension n
S_n	$n \times n$ symmetric real matrices
S_n^+, $A \succeq 0$	$n \times n$ symmetric positive semidefinite matrices
S_n^{++}, $A \succ 0$	$n \times n$ symmetric positive definite matrices
I, I_n	identity of appropriate size or of size n
e	vector of all ones of appropriate dimension
E_{ij}	symmetric matrix with ij-entry equal to one and zero otherwise
$\lambda_i(A)$	i-th eigenvalue of $A \in M_n$, usually $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$
$\lambda_{\min}(A)$, $\lambda_{\max}(A)$	minimal and maximal eigenvalue of A
Λ_A	diagonal matrix with $(\Lambda_A)_{ii} = \lambda_i(A)$
A^T	transpose of A
$\mathrm{tr}(A)$	trace of $A \in M_n$, $\mathrm{tr}(A) = \sum_{i=1}^n a_{ii} = \sum_{i=1}^n \lambda_i(A)$
$\langle A, B \rangle$	inner product in $M_{m,n}$, $\langle A, B \rangle = \mathrm{tr}(B^T A)$
$\|A\|_{\mathrm{F}}$	Frobenius norm of A, $\|A\|_{\mathrm{F}} = \sqrt{\langle A, A \rangle}$
$\|\|y\|\|_H$	H-norm of $y \in \mathbb{R}^m$ for $H \in S_m^{++}$; $\|\|y\|\|_H^2 = \langle y, Hy \rangle$
$\mathrm{Diag}(v)$	diagonal matrix with v on its main diagonal
$\mathrm{diag}(A)$	the diagonal of $A \in M_n$ as a column vector
argmin	minimizing argument
Argmin	set of minimizing arguments

B Variables, Functions, Definitions

n	size of the matrix variable
m	number of primal constraints/dual design variables
C	cost matrix $\in S_n$
A_i	primal constraint matrix i, $A_i \in S_n$
$\mathcal{A}^T$	linear combination of constraint matrices, $\mathcal{A}^T y = \sum A_i y_i$
$\mathcal{A}$	primal constraint operator, $[\mathcal{A}X]_i = \langle A_i, X \rangle$
b	primal right hand side, dual linear cost coefficients
a	constant trace size, $a > 0$, dual factor of $\lambda_{\max}$
$J_=, J_\leq, J_\geq$	index sets of equality and inequality constraints, partition $\{1, \ldots, m\}$
y	dual design variables, $y \in Y \subset \mathbb{R}^m$
Y, Y^*	feasible set for y and its dual, $Y = Y^* = \{y \in \mathbb{R}^m : \ y_i \geq 0 \text{ for } i \in J_\geq,\ y_i \leq 0 \text{ for } i \in J_\leq\}$
η	Lagrange multipliers for sign constraints on y, primal slacks
X	primal matrix variable, $X \in S_n$
$\mathcal{W}$	positive semidefinite matrices with trace a, $\mathcal{W} = \{W \succeq 0 : \operatorname{tr} W = a\}$
$\widehat{\mathcal{W}}$	subset of $\mathcal{W}$, $\widehat{\mathcal{W}} = \{PVP^T + \alpha\overline{W} : \operatorname{tr} V + \alpha = a, V \succeq 0, \alpha \geq 0\}$
P	orthonormal $n \times r$ matrix, the *bundle*
$\overline{W}$	element of $\mathcal{W}$, the *aggregate*
V, α	variables from S_r^+ and $\mathbb{R}_+$ spanning $\widehat{\mathcal{W}}$
r	number of columns of P, the *size of the bundle*
$\imath_Y$	indicator function for Y, $\imath_Y(y) = 0$ for $y \in Y$, ∞ for $y \notin Y$
f	eigenvalue/objective function extended to $\mathbb{R}^m$, $f(y) = a\lambda_{\max}(C - \mathcal{A}^T y) + b^T y + \imath_Y(y) = \sup\limits_{W \in \mathcal{W}, \eta \in Y^*} \left\langle C - \mathcal{A}^T y, W \right\rangle + (b - \eta)^T y$
$f_{W,\eta}$	linear minorant of f, $f_{W,\eta}(y) = \left\langle C - \mathcal{A}^T y, W \right\rangle + (b - \eta)^T y$
$f_{\widehat{\mathcal{W}},\eta}$	convex minorant of f, $f_{\widehat{\mathcal{W}},\eta}(y) = \max_{W \in \widehat{\mathcal{W}}} \left\langle C - \mathcal{A}^T y, W \right\rangle + (b - \eta)^T y$
$\hat{y}$	*center of stability* for *augmented model* $\max_{\eta \in Y^*} f_{\widehat{\mathcal{W}},\eta}(y) + \frac{u}{2}\|\|y - \hat{y}\|\|_H^2$
u	*weight* of augmenting/prox term
H	diagonal scaling matrix $\in S_m^{++}$, usually $H = I$
$y_{\min}(W, \eta)$	optimal y for given W and η, $y_{\min}(W, \eta) = \hat{y} + \frac{1}{u} H^{-1}(\mathcal{A}W - b + \eta)$
$\eta_{\max}(W)$	optimal η for given W
$\cdot^k$	iteration index