VSDP 2020 manual

C. Jansson et al.

Jan 12, 2022

CONTENTS

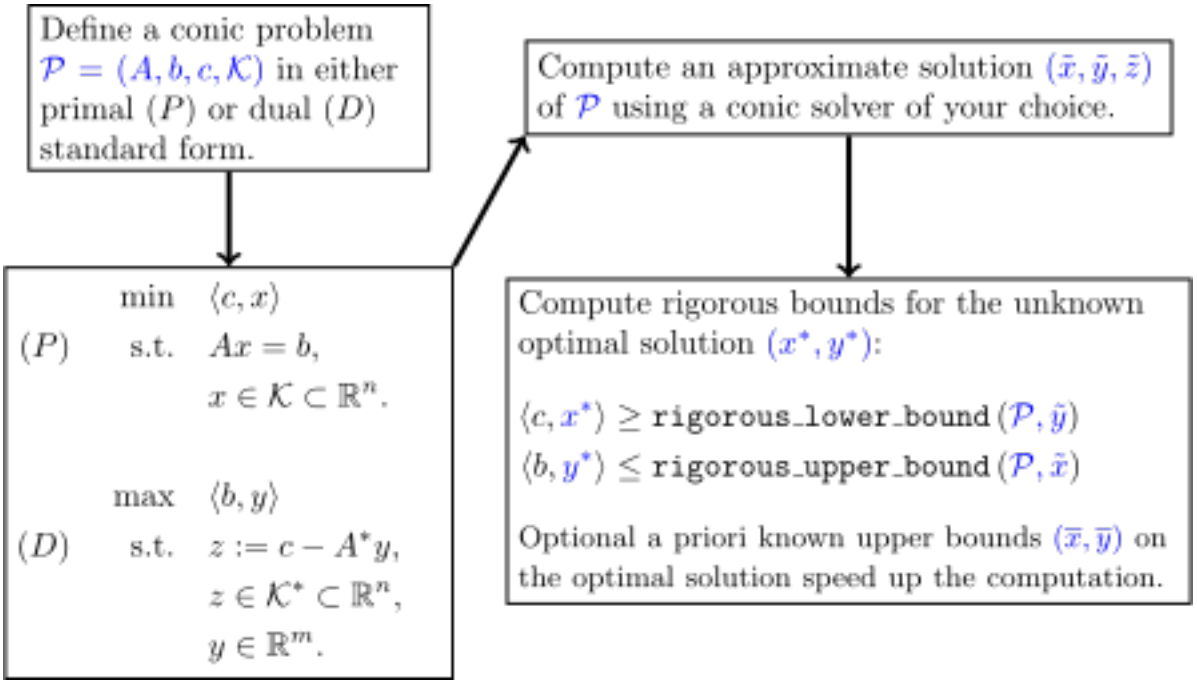
Latest version of this manual https://vsdp.github.io or https://vsdp.github.io/vsdp-2020-manual.pdf.

Abstract

VSDP (Verified SemiDefinite-quadratic-linear Programming) is a software package for the computation of verified results in conic programming. It supports the constraint cone consisting of the product of semidefinite cones, second-order cones, and the non-negative orthant. VSDP provides functions for computing rigorous error bounds of the true optimal value, verified enclosures of epsilon-optimal solutions, and verified certificates of infeasibility. All rounding errors due to floating-point arithmetic are taken into account.

For theoretical details of the implemented algorithms, we refer to [11, 12, 13, 14].

The software is completely written in MATLAB / GNU Octave and requires the interval toolbox INTLAB. Thus interval input is supported as well.



The latest version of VSDP provides easy access to the *conic solvers*:

- CSDP, GLPK, LINPROG, lp_solve, MOSEK, SDPA, SDPT3, and SeDuMi.

Available VSDP versions

- The VSDP versions numbers reflect the release date:
  - VSDP 2020
    - Improvements: solver support and detection, workflow, testing, and documentation.
  - VSDP 2012
    - Improvements: additional support of second-order cones, linear cones, and free variables, vectorized internal structure. See [9].
  - VSDP 2006

    * Support for large scale semidefinite problems, comprehensible code. See [10].

Contributors

- Christian Jansson (mailto:jansson@tuhh.de)
- Marko Lange (mailto:m.lange@tuhh.de)
- Viktor Härter
- Kai Torben Ohlhus (mailto:k.ohlhus@gmail.com)

Part I

Manual

CHAPTER

ONE

INSTALLATION

1.1 Requirements

To run VSDP, the following requirements have to be fulfilled:

- A recent version of GNU Octave or MATLAB has to be installed.
- The interval toolbox INTLAB is required.
- At least one of the following approximate solvers has to be installed:
  - CSDP,
  - GLPK,
  - LINPROG,
  - lp_solve,
  - MOSEK,
  - SDPA,
  - SDPT3, or
  - SeDuMi.

1.2 Obtaining VSDP

1.2.1 ZIP-File

The most recent version of VSDP and this manual are available at https://vsdp.github.io. There you can download a ZIP-file `vsdp-2020-master.zip` and extract it to an arbitrary location.

Legacy versions of VSDP are available from https://www.tuhh.de/ti3/software/.

1.2.2 Using git

If you have git installed and about 700 MB of disk space available, you can easily obtain a full bundle of VSDP 2006, 2012, 2020, including some aforementioned approximate solvers, and some benchmark libraries by the command

```
git clone --recurse-submodules https://github.com/vsdp/vsdp-bundle
```

In the cloned directory `vsdp-bundle/vsdp/2020` you find the latest version of VSDP.

1.3 Installing VSDP

If all requirements are fulfilled, just call from the MATLAB or GNU Octave command prompt inside the VSDP directory

```
install_vsdp;
```

and all necessary paths are set and VSDP is fully functional. To test the latter, you can run the small builtin test suite from MATLAB via

```
runtests ('testVSDP')

  Totals:
    5 Passed, 0 Failed, 0 Incomplete.
    8.2712 seconds testing time.
```

or from GNU Octave via

```
testVSDP;
```

```
Test summary
------------

testSINDEX            PASSED        0.322136
testSVEC_SMAT         PASSED        0.549652
testLP                PASSED        3.108967
testSOCP              PASSED        2.231160
testSDP               PASSED        16.238318
```

CHAPTER

TWO

CONIC PROGRAMMING

2.1 Primal and dual form

VSDP can handle three self dual convex cones, that often occur in practical applications. These are:

- The non-negative orthant:

$$\mathbb{R}^n_+ := \{x \in \mathbb{R}^n \colon\ x_i \geq 0,\ i = 1, \dots, n\}.$$

- The Lorentz cone (see [1]):

$$\mathbb{L}^n := \{x \in \mathbb{R}^n \colon x_1 \geq \|x_{2:n}\|_2\}.$$

- The cone of symmetric positive semidefinite matrices:

$$\mathbb{S}^n_+ := \left\{X \in \mathbb{R}^{n\times n} \colon\ X = X^T,\ v^T X v \geq 0,\ \forall v \in \mathbb{R}^n\right\}.$$

If a quantity is in the interior of one of the above cones, the definitions above must hold with strict inequalities.

By $\langle c, x\rangle := c^T x$ the usual Euclidean inner product of vectors in $\mathbb{R}^n$ is denoted. For symmetric matrices $X, Y \in \mathbb{S}^n$ the inner product is given by $\langle X, Y\rangle := \text{trace}(XY)$.

Let A^f and A^l be a $m \times n_f$ and a $m \times n_l$ matrix, respectively, and let A^q_i be $m \times q_i$ matrices for $i = 1, \dots, n_q$. Let $c^f \in \mathbb{R}^{n_f}$, $c^l \in \mathbb{R}^{n_l}$, $c^q_i \in \mathbb{R}^{q_i}$, and $b \in \mathbb{R}^m$. Moreover, let $A^s_{1,j}, \dots, A^s_{m,j}$, C^s_j, X^s_j be symmetric $s_j \times s_j$ matrices for $j = 1, \dots, n_s$.

Now we can define the conic semidefinite-quadratic-linear programming problem in primal standard form

$$\begin{array}{ll}
\text{minimize} & \langle c^f, x^f\rangle + \langle c^l, x^l\rangle + \sum_{i=1}^{n_q} \langle c^q_i, x^q_i\rangle + \sum_{j=1}^{n_s} \langle C^s_j, X^s_j\rangle \\
\text{subject to} & A^f x^f + A^l x^l + \sum_{i=1}^{n_q} A^q_i x^q_i + \sum_{j=1}^{n_s} \mathcal{A}^s_j(X^s_j) = b, \\
& x^f \in \mathbb{R}^{n_f}, \\
& x^l \in \mathbb{R}^{n_l}_+, \\
& x^q_i \in \mathbb{L}^{q_i}, \quad i = 1, \dots, n_q, \\
& X^s_j \in \mathbb{S}^{s_j}_+, \quad j = 1, \dots, n_s,
\end{array}$$

where x^f are "free variables", x^l are "non-negative variables", x^q_i are "second-order cone (SOCP) variables", and finally X^s_j are "positive semidefinite (SDP) variables". The linear operator

$$\mathcal{A}^s_j(X^s_j) := \begin{pmatrix} \langle A^s_{1j}, X^s_j\rangle \\ \vdots \\ \langle A^s_{mj}, X^s_j\rangle \end{pmatrix}$$

maps the symmetric matrices X^s_j to $\mathbb{R}^m$. The adjoint linear operator is

$$(\mathcal{A}^s_j)^* y := \sum_{k=1}^{m} A^s_{kj} y_k.$$

The dual problem associated with the primal standard form is

$$\begin{array}{ll} \text{maximize} & b^T y \\ \text{subject to} & (A^f)^T y + z^f = c^f, \\ & (A^l)^T y + z^l = c^l, \\ & (A_i^q)^T y + z_i^q = c_i^q, \\ & (\mathcal{A}_j^s)^* y + Z_j^s = C_j^s, \end{array}$$

where $z^f \in \{0\}^{n_f}$, $z^l \in \mathbb{R}_+^{n_l}$, $z_i^q \in \mathbb{L}^{q_i}$, $i = 1, \ldots, n_q$, and $Z_j^s \in \mathbb{S}_+^{s_j}$, $j = 1, \ldots, n_s$.

The objective functions and equality constraints of the primal and dual problem are linear. Thus conic programming can be seen as an extension of linear programming with additional conic constraints.

By definition the vector x^f contains all unconstrained or free variables, whereas all other variables are bounded by conic constraints. In several applications some solvers (for example SDPA or CSDP) require that free variables are converted into the difference of non-negative variables. Besides the major disadvantage that this transformation is numerical unstable, it also increases the number of variables of the particular problems. In VSDP *free variables* can be handled in a numerical stable manner.

2.2 Condensed form

Occasionally, it is useful to represent the conic programming problem in a more compact form by using the symmetric vectorization operator. This operator maps a symmetric matrix $X \in \mathbb{S}^n$ to an $n(n+1)/2$-dimensional vector

$$svec(X, \alpha) := \begin{pmatrix} X_{11} & \alpha X_{12} & X_{22} & \alpha X_{13} & \cdots & X_{nn} \end{pmatrix}^T,$$

where α is a scaling factor for the off diagonal elements. The inverse operation is denoted by $smat$ such that $smat(svec(X, \alpha), 1/\alpha) = X$. For a vectorized quantity x, scaled by α, we often write $x \in \mathbb{S}^n$ as abbreviation for $smat(x, 1/\alpha) \in \mathbb{S}^n$.

By using $svec$ it is possible to map each symmetric matrix to a vector quantity. Additionally, by using the scaling factor $\alpha = 1$ for all A_{kj}^s, C_j^s, and Z_j^s and the scaling factor $\alpha = 2$ for all X_j^s, the inner product of symmetric matrices reduces to a simple scalar product of two vectors. Thus all cones can be treated equally.

The condensed quantities c, x, and z are $n \times 1$-vectors:

$$c := \begin{pmatrix} c^f \\ c^l \\ c_1^q \\ \vdots \\ c_{n_q}^q \\ svec(C_1^s, 1) \\ \vdots \\ svec(C_{n_s,1}^s) \end{pmatrix}, x := \begin{pmatrix} x^f \\ x^l \\ x_1^q \\ \vdots \\ x_{n_q}^q \\ svec(X_1^s, 2) \\ \vdots \\ svec(X_{n_s,2}^s) \end{pmatrix}, z := \begin{pmatrix} z^f \\ z^l \\ z_1^q \\ \vdots \\ z_{n_q}^q \\ svec(Z_1^s, 1) \\ \vdots \\ svec(Z_{n_s}^s, 1) \end{pmatrix},$$

where $n = n_f + n_l + \sum_{i=1}^{n_q} q_i + \sum_{j=1}^{n_s} s_j(s_j+1)/2$. A^T becomes an $n \times m$ matrix

$$A^T := \begin{pmatrix} & A^f & \\ & A^l & \\ & A_1^q & \\ & \vdots & \\ & A_{n_q}^q & \\ svec(A_{11}^s, 1) & \cdots & svec(A_{1m}^s, 1) \\ \vdots & & \vdots \\ svec(A_{n_s 1}^s, 1) & \cdots & svec(A_{n_s m}^s, 1) \end{pmatrix}$$

Let the constraint cone K and its dual cone K^* be

$$\begin{aligned}\mathcal{K} &:= \quad \mathbb{R}^{n_f} \times \mathbb{R}^{n_l}_+ \times \mathbb{L}^{q_1} \times ... \times \mathbb{L}^{q_{n_q}} \times \mathbb{S}^{s_1}_+ \times ... \times \mathbb{S}^{s_{n_s}}_+, \\ \mathcal{K}^* &:= \{0\}^{n_f} \times \mathbb{R}^{n_l}_+ \times \mathbb{L}^{q_1} \times ... \times \mathbb{L}^{q_{n_q}} \times \mathbb{S}^{s_1}_+ \times ... \times \mathbb{S}^{s_{n_s}}_+.\end{aligned}$$

With these abbreviations we obtain the following block form of the conic problem:

$$\begin{array}{ll}\text{minimize} & c^T x, \\ \text{subject to} & Ax = b, \\ & x \in \mathcal{K},\end{array}$$

with optimal value $\hat{f}_p$ and the corresponding dual problem

$$\begin{array}{ll}\text{maximize} & b^T y, \\ \text{subject to} & z = c - (A)^T y \in \mathcal{K}^*,\end{array}$$

with optimal value $\hat{f}_d$. In VSDP each conic problem is fully described by the four variables `(A, b, c, K)`. The first two quantities represent the affine constraints $Ax = b$. The third is the primal objective vector `c`, and the last describes the underlying cone. The cone `K` is a structure with four fields: `K.f`, `K.l`, `K.q`, and `K.s`. The field `K.f` stores the number of free variables n_f, the field `K.l` stores the number of non-negative variables n_l, the field `K.q` stores the dimensions $q_1, ..., q_{n_q}$ of the second order cones, and similarly `K.s` stores the dimensions $s_1, ..., s_{n_s}$ of the semidefinite cones. If a component of `K` is empty, then it is assumed that the corresponding cone does not occur.

It is well known that for linear programming problems strong duality $\hat{f}_p = \hat{f}_d$ holds without any constraint qualifications. General conic programs satisfy only the weak duality condition $\hat{f}_d \leq \hat{f}_p$. Strong duality requires additional constraint qualifications, such as *Slater's constraint qualifications* (see [24] and [28]).

Strong Duality Theorem

- If the primal problem is strictly feasible (i.e. there exists a primal feasible point x in the interior of K) and $\hat{f}_p$ is finite, then $\hat{f}_p = \hat{f}_d$ and the dual supremum is attained.
- If the dual problem is strictly feasible (i.e. there exists some y such that $z = c-(A)^T y$ is in the interior of K^*) and $\hat{f}_d$ is finite, then $\hat{f}_d = \hat{f}_p$, and the primal infimum is attained.

In general, the primal or dual problem formulation may have optimal solutions while its respective dual problem is infeasible, or the duality gap may be positive at optimality.

Duality theory is central to the study of optimization. Firstly, algorithms are frequently based on duality (like primal-dual interior-point methods), secondly, they enable one to check whether or not a given feasible point is optimal, and thirdly, it allows one to compute verified results efficiently.

2.3 Interval arithmetic

For the usage of VSDP a knowledge of interval arithmetic [23] is not required, but some error bound outputs are intervals. Therefore we give a very brief introduction to interval formats. An interval vector or an interval matrix is defined as a set of vectors or matrices that vary between a lower and an upper vector or matrix, respectively. In other words, these are quantities with interval components.

In INTLAB [22] these interval quantities can be initialized with the `infsup`-function.

```
format infsup short
x = infsup (-1, 2)
```

```
intval x =
[   -1.0000,    2.0000]
```

Equivalently, these quantities can be defined by a midpoint-radius representation, using the `midrad`-function.

```
format midrad short
y = midrad (0.5, 1.5)
format infsup short
y
```

```
intval y =
<    0.5000,   1.5000>
intval y =
[   -1.0000,    2.0000]
```

CHAPTER

THREE

LINEAR PROGRAMMING

In this section we describe how linear programming problems can be solved with VSDP. In particular, two linear programs are considered in detail.

3.1 First example

Consider the linear program in primal standard form

$$\begin{array}{ll} \text{minimize} & 2x_2 + 3x_4 + 5x_5, \\ \text{subject to} & \begin{pmatrix} -1 & 2 & 0 & 1 & 1 \\ 0 & 0 & -1 & 0 & 2 \end{pmatrix} x = \begin{pmatrix} 2 \\ 3 \end{pmatrix}, \\ & x \in \mathbb{R}^5_+, \end{array}$$

with its corresponding dual problem

$$\begin{array}{ll} \text{maximize} & 2y_1 + 3y_2, \\ \text{subject to} & z = \begin{pmatrix} 0 \\ 2 \\ 0 \\ 3 \\ 5 \end{pmatrix} - \begin{pmatrix} -1 & 0 \\ 2 & 0 \\ 0 & -1 \\ 1 & 0 \\ 1 & 2 \end{pmatrix} y \in \mathbb{R}^5_+. \end{array}$$

The unique exact optimal solution is given by $x^* = (0, 0.25, 0, 0, 1.5)^T$, $y^* = (1, 2)^T$ with $\hat{f}_p = \hat{f}_d = 8$.

The input data of the problem in VSDP are:

```
A = [-1, 2,  0, 1, 1;
       0, 0, -1, 0, 2];
b = [2; 3];
c = [0; 2; 0; 3; 5];
K.l = 5;
```

To create a VSDP object of the linear program data above, we call the VSDP class constructor and do not suppress the output by terminating the statement with a semicolon `;`.

```
obj = vsdp (A, b, c, K)
```

```
obj =
  VSDP conic programming problem with dimensions:

    [n,m] = size(obj.At)
```

(continues on next page)

(continued from previous page)

```
     n    = 5 variables
       m  = 2 constraints

  and cones:

     K.l = 5

  Compute an approximate solution:

    'obj = obj.solve()'


  Detailed information:  'obj.info()'
```

The output contains all relevant information about the conic problem and includes the command `obj.solve` to proceed.

By calling the `obj.solve` method on the VSDP object `obj`, we can compute an approximate solution `x`, `y`, and `z`, for example by using SDPT3. When calling `obj.solve` without any arguments, the user is asked to choose one of the supported solvers.

```
obj.solve ('sdpt3');
```

```
 num. of constraints =  2
 dim. of linear var  =  5
*****************************************************************
   SDPT3: Infeasible path-following algorithms
*****************************************************************
 version  predcorr  gam  expon  scale_data
    NT      1      0.000   1        0
it pstep dstep pinfeas dinfeas  gap      prim-obj      dual-obj    cputime
-----------------------------------------------------------------
 0|0.000|0.000|6.3e+00|2.6e+00|5.0e+02| 1.000000e+02  0.000000e+00| 0:0:00| chol ␣
 ↪1  1
 1|1.000|0.867|9.5e-07|3.7e-01|8.7e+01| 4.535853e+01  2.191628e+00| 0:0:00| chol ␣
 ↪1  1
 2|1.000|1.000|1.9e-06|3.1e-03|1.1e+01| 1.670044e+01  5.453562e+00| 0:0:00| chol ␣
 ↪1  1
 3|0.928|1.000|1.6e-07|3.1e-04|1.1e+00| 8.503754e+00  7.407909e+00| 0:0:00| chol ␣
 ↪1  1
 4|1.000|0.591|1.0e-07|1.5e-04|7.9e-01| 8.626424e+00  7.841794e+00| 0:0:00| chol ␣
 ↪1  1
 5|0.971|0.984|3.0e-09|5.4e-06|2.2e-02| 8.015560e+00  7.993623e+00| 0:0:00| chol ␣
 ↪1  1
 6|0.988|0.988|7.1e-10|3.7e-07|2.6e-04| 8.000185e+00  7.999926e+00| 0:0:00| chol ␣
 ↪1  1
 7|0.989|0.989|1.1e-10|4.3e-09|2.9e-06| 8.000002e+00  7.999999e+00| 0:0:00| chol ␣
 ↪1  1
 8|0.997|1.000|9.4e-13|2.2e-11|3.9e-08| 8.000000e+00  8.000000e+00| 0:0:00|
  stop: max(relative gap, infeasibilities) < 1.00e-08
-----------------------------------------------------------------
 number of iterations   =  8
 primal objective value =  8.00000003e+00
 dual   objective value =  7.99999999e+00
```

(continues on next page)

(continued from previous page)

```
 gap := trace(XZ)       = 3.88e-08
 relative gap           = 2.28e-09
 actual relative gap    = 2.27e-09
 rel. primal infeas (scaled problem)   = 9.39e-13
 rel. dual     "        "       "      = 2.19e-11
 rel. primal infeas (unscaled problem) = 0.00e+00
 rel. dual     "        "       "      = 0.00e+00
 norm(X), norm(y), norm(Z) = 1.5e+00, 2.2e+00, 3.0e+00
 norm(A), norm(b), norm(C) = 4.5e+00, 4.6e+00, 7.2e+00
 Total CPU time (secs)  = 0.21
 CPU time per iteration = 0.03
 termination code       =  0
 DIMACS: 1.1e-12  0.0e+00  2.6e-11  0.0e+00  2.3e-09  2.3e-09
------------------------------------------------------------------
```

The solver output is often quite verbose. Especially for large problems it is recommended to display the solver progress. To suppress solver messages, the following option can be set:

```
obj.options.VERBOSE_OUTPUT = false;
```

To permanently assign an approximate solver to a VSDP object, use the following option:

```
obj.options.SOLVER = 'sdpt3';
```

By simply typing the VSDP object's name, the user gets a short summary of the solution state.

```
obj
```

```
obj =
  VSDP conic programming problem with dimensions:

    [n,m] = size(obj.At)
     n    = 5 variables
       m  = 2 constraints

  and cones:

     K.l = 5

  obj.solutions.approximate:

      Solver 'sdpt3': Normal termination, 0.3 seconds.

        c'*x = 8.000000025993693e+00
        b'*y = 7.999999987362061e+00


  Compute a rigorous lower bound:

    'obj = obj.rigorous_lower_bound()'

  Compute a rigorous upper bound:

    'obj = obj.rigorous_upper_bound()'
```

(continues on next page)

(continued from previous page)

```
  Detailed information:  'obj.info()'
```

On success, one can obtain the approximate solutions for further processing.

```
format short
x = obj.solutions.approximate.x
y = obj.solutions.approximate.y
```

```
x =
   0.0000000092324
   0.2500000014452
   0.0000000040905
   0.0000000042923
   1.5000000020453

y =
   1.00000
   2.00000
```

The approximate solution is close to the optimal solution $x^* = (0, 0.25, 0, 0, 1.5)^T$, $y^* = (1, 2)^T$.

With this approximate solution, a rigorous lower bound `fL` of the primal optimal value $\hat{f}_p = 8$ can be computed by calling:

```
format long
obj.rigorous_lower_bound ();
fL = obj.solutions.rigorous_lower_bound.f_objective(1)
```

```
fL =  7.999999987362061
```

Similarly, a rigorous upper bound `fU` of the dual optimal value $\hat{f}_d$ can be computed by calling:

```
obj.rigorous_upper_bound ();
fU = obj.solutions.rigorous_upper_bound.f_objective(2)
```

```
fU =  8.000000025997927
```

The summary output of the VSDP object contains the information about the rigorous error bounds, as well. It can be extracted if necessary.

```
obj
```

```
obj =
  VSDP conic programming problem with dimensions:

    [n,m] = size(obj.At)
     n    = 5 variables
       m  = 2 constraints

  and cones:
```

(continues on next page)

(continued from previous page)

```
    K.l = 5

  obj.solutions.approximate:

      Solver 'sdpt3': Normal termination, 0.3 seconds.

        c'*x = 8.000000025993693e+00
        b'*y = 7.999999987362061e+00


  obj.solutions.rigorous_lower_bound:

      Normal termination, 0.0 seconds, 0 iterations.

          fL = 7.999999987362061e+00

  obj.solutions.rigorous_upper_bound:

      Normal termination, 0.0 seconds, 0 iterations.

          fU = 8.000000025997927e+00



  Detailed information:  'obj.info()'
```

Despite the rigorous lower bound `fL`, the solution object `obj.solutions.rigorous_lower_bound` contains more information:

1. `Y` is a rigorous interval enclosure of a dual feasible near optimal solution and
2. `Zl` a lower bound of each cone in $z = c - A^*y$. For a linear program this is a lower bound on each component of `z`.

```
format short
format infsup
 Y = obj.solutions.rigorous_lower_bound.y
Zl = obj.solutions.rigorous_lower_bound.z
```

```
intval Y =
[    0.9999,    1.0000]
[    2.0000,    2.0001]
Zl =
   0.9999999873621
   0.0000000252759
   2.0000000042126
   2.0000000126379
   0.0000000042126
```

Since `Zl` is positive, the dual problem is strictly feasible, and the rigorous interval vector `Y` contains a dual interior solution. Here only some significant digits of this interval vector are displayed. The upper and lower bounds of the interval `Y` can be obtained by using the `sup` and `inf` routines of INTLAB. For more information about the `intval` data type see [22].

The information returned by `vsdp.rigorous_upper_bound` is similar:

1. `X` is a rigorous interval enclosure of a primal feasible near optimal solution and
2. `Xl` a lower bound of each cone in `X`. Again, for a linear program this is a lower bound on each component of `X`.

```
X  = obj.solutions.rigorous_upper_bound.x
Xl = obj.solutions.rigorous_upper_bound.z
```

```
intval X =
[    0.0000,    0.0001]
[    0.2500,    0.2501]
[    0.0000,    0.0001]
[    0.0000,    0.0001]
[    1.5000,    1.5001]
Xl =
   0.0000000092324
   0.2500000014474
   0.0000000040905
   0.0000000042923
   1.5000000020452
```

Since `Xl` is a positive vector, `X` is contained in the positive orthant and the primal problem is strictly feasible.

Summarizing, we have obtained a primal-dual interval solution pair with an accuracy measured by

$$\mu(a,b) = \frac{a-b}{\max\{1.0, (|a|+|b|)/2\}},$$

see [10].

```
format shorte
mu = (fU - fL) / max (1, (abs (fU) + abs(fL)) / 2)
```

```
mu =    4.8295e-09
```

This means, that the computed rigorous upper and lower error bounds have an accuracy of eight to nine decimal digits.

3.2 Second example with free variables

How a linear program with free variables can be rigorously solved by VSDP is demonstrated by the following example with one free variable x_3:

$$\begin{array}{ll} \text{minimize} & \begin{pmatrix} 1 & 1 & -0.5 \end{pmatrix} x, \\ \text{subject to} & \begin{pmatrix} 1 & -1 & 2 \\ 1 & 1 & -1 \end{pmatrix} x = \begin{pmatrix} 0.5 \\ 1 \end{pmatrix} \\ & x_1, x_2 \in \mathbb{R}^2_+, \\ & x_3 \in \mathbb{R}. \end{array}$$

The optimal solution pair of this problem is $x^* = (\frac{5}{6}, 0, -\frac{1}{6})^T$, $y^* = (\frac{1}{6}, \frac{5}{6})^T$ with $\hat{f}_p = \hat{f}_d = \frac{11}{12} \approx 9.166\ldots$.

When entering a conic problem the order of the variables is important:

1. free variables,
2. non-negative variables,
3. second-order cone variables,

4. semidefinite variables.

All involved VSDP quantities, the constraint matrix `A`, the primal objective `c`, the primal solution `x`, as well as `z`, follow this order. In the second linear programming example, the free variable is x_3 and the non-negative variables are x_1 and x_2, respectively. Second-order cone variables and semidefinite variables are not present.

Therefore, the problem data are:

```
K.f = 1;  % number of free variables
K.l = 2;  % number of non-negative variables
A = [ 2, 1, -1;   % first column corresponds to free variable x3
     -1, 1,  1];  % second and third to bounded x1, x2
c = [-0.5; 1; 1]; % the same applies to c
b = [0.5; 1];
```

The whole VSDP computation can be done in a few lines of code:

```
obj = vsdp (A, b, c, K);
obj.options.VERBOSE_OUTPUT = false;
obj.solve ('sdpt3') ...
   .rigorous_lower_bound () ...
   .rigorous_upper_bound ();
```

Yielding

```
obj
```

```
obj =
  VSDP conic programming problem with dimensions:

    [n,m] = size(obj.At)
     n    = 3 variables
       m  = 2 constraints

  and cones:

     K.f = 1
     K.l = 2

  obj.solutions.approximate:

      Solver 'sdpt3': Normal termination, 0.3 seconds.

        c'*x = 9.166666669227741e-01
        b'*y = 9.166666662221519e-01


  obj.solutions.rigorous_lower_bound:

      Normal termination, 0.0 seconds, 0 iterations.

          fL = 9.166666662221495e-01

  obj.solutions.rigorous_upper_bound:

      Normal termination, 0.0 seconds, 0 iterations.
```

(continues on next page)

(continued from previous page)

```
          fU = 9.166666669227844e-01


  Detailed information:  'obj.info()'
```

CHAPTER

FOUR

SECOND-ORDER CONE PROGRAMMING

Consider a least squares problem from [7]:

$$\|b_{data} - A_{data}\,\hat{y}\|_2 = \min_{y_{3:5}\in\mathbb{R}^3} \|b_{data} - A_{data}\,y_{3:5}\|_2$$

with a matrix of rank two

```
A_data = [ 3 1 4 ;
           0 1 1 ;
          -2 5 3 ;
           1 4 5 ];
```

and right-hand side

```
b_data = [ 0 ;
           2 ;
           1 ;
           3 ];
```

This problem can be formulated as second-order cone program in dual standard form:

$$\begin{array}{ll} \text{maximize} & -y_1 - y_2, \\ \text{subject to} & y_1 \geq \|(b_{data} - A_{data}\,y_{3:5})\|_2, \\ & y_2 \geq \left\| \begin{pmatrix} 1 \\ y_{3:5} \end{pmatrix} \right\|_2, \\ & y \in \mathbb{R}^5. \end{array}$$

The two inequality constraints can be written as second-order cone vectors

$$\begin{pmatrix} y_1 \\ b_{data} - A_{data}\,y_{3:5} \end{pmatrix} \in \mathbb{L}^5 \quad \text{and} \quad \begin{pmatrix} y_2 \\ 1 \\ y_{3:5} \end{pmatrix} \in \mathbb{L}^5.$$

Both vectors can be expressed as matrix-vector product of y

$$\underbrace{\begin{pmatrix} 0 \\ b_{data} \end{pmatrix}}_{=c_1^q} - \underbrace{\begin{pmatrix} -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & (& A_{data} &) \end{pmatrix}}_{=(A_1^q)^T} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{pmatrix} \in \mathbb{L}^5$$

and

$$\underbrace{\begin{pmatrix}0\\1\\0\\0\\0\end{pmatrix}}_{=c_2^q} - \underbrace{\begin{pmatrix}0&-1&0&0&0\\0&0&0&0&0\\0&0&-1&0&0\\0&0&0&-1&0\\0&0&0&0&-1\end{pmatrix}}_{=(A_2^q)^T}\begin{pmatrix}y_1\\y_2\\y_3\\y_4\\y_5\end{pmatrix} \in \mathbb{L}^5.$$

With these formulations, the dual problem takes the form

$$\begin{aligned}\text{maximize} \quad & \underbrace{\begin{pmatrix}-1&-1&0&0&0\end{pmatrix}}_{=b^T} y,\\ \text{subject to} \quad & z = \underbrace{\begin{pmatrix}0\\b_{data}\\0\\1\\0\\0\\0\end{pmatrix}}_{=c} - \underbrace{\begin{pmatrix}-1&0&0&0&0\\0&0&(&A_{data}&)\\0&-1&0&0&0\\0&0&0&0&0\\0&0&-1&0&0\\0&0&0&-1&0\\0&0&0&0&-1\end{pmatrix}}_{=A^T} y \in K^*,\\ & y \in \mathbb{R}^5.\end{aligned}$$

where $K^* = \mathbb{L}^5 \times \mathbb{L}^5$.

We want to solve this problem with SeDuMi and enter the problem data of the primal problem.

```
At = zeros (10, 5);
At(1,1) = -1;
At(2:5, 3:5)  = A_data;
At(6,2) = -1;
At(8:10, 3:5) = -eye(3);
b = [-1 -1 0 0 0]';
c = [ 0 b_data' 0 0 0 0 0]';
```

Apart from the data `(At,b,c)`, the vector `q = [5;5]` of the second-order cone block sizes must be forwarded to the structure `K`:

```
K.q = [5; 5];
```

Now we compute approximate solutions by using `obj.solve` and then rigorous error bounds by using `obj.rigorous_lower_bound` and `obj.rigorous_upper_bound`:

```
obj = vsdp (At, b, c, K);
obj.options.VERBOSE_OUTPUT = false;
obj.solve('sedumi') ...
   .rigorous_lower_bound() ...
   .rigorous_upper_bound();
```

Finally, we get an overview about all the performed computations:

```
obj
```

```
obj =
  VSDP conic programming problem with dimensions:
```

(continues on next page)

(continued from previous page)

```
  [n,m] = size(obj.At)
   n    = 10 variables
     m  =  5 constraints

and cones:

   K.q = [ 5, 5 ]

obj.solutions.approximate:

    Solver 'sedumi': Normal termination, 0.2 seconds.

      c'*x = -2.592163303832843e+00
      b'*y = -2.592163302997335e+00


obj.solutions.rigorous_lower_bound:

    Solver 'sedumi': Normal termination, 0.2 seconds, 1 iterations.

        fL = -2.592163303541427e+00

obj.solutions.rigorous_upper_bound:

    Solver 'sedumi': Normal termination, 0.3 seconds, 1 iterations.

        fU = -2.592163296674677e+00



Detailed information:  'obj.info()'
```

Now we analyze the resulting regularized least squares solution

```
y_SOCP = obj.solutions.approximate.y(3:5)
```

```
y_SOCP =
  -0.022817
   0.218532
   0.195715
```

and compare it to a naive least squares solution `y_LS`, which takes extreme values in this example

```
y_LS = A_data \ b_data
```

```
y_LS =
   8.0353e+14
   8.0353e+14
  -8.0353e+14
```

Displaying the norms of the results side-by-side reveals, that `y_SOCP` is better suited for numerical computations.

```
[                  norm(y_SOCP)                    norm(y_LS);
 norm(b_data - A_data * y_SOCP)  norm(b_data - A_data * y_LS)]
```

```
ans =
   2.9425e-01   1.3918e+15
   2.2979e+00   2.5125e+00
```

Conic programming allows to mix constraints of different types. For instance, one can add the linear inequality $\sum_{i=1}^{5} y_i \leq 3.5$ to the previous dual problem. We extend the input data as follows:

```
At = [1 1 1 1 1; At];
c =  [3.5      ; c ];
K.l = 1;
```

Remember that the order of the cone variables matters for `At` and `c`.

```
obj = vsdp (At, b, c, K);
obj.options.VERBOSE_OUTPUT = false;
obj.solve('sedumi') ...
   .rigorous_lower_bound() ...
   .rigorous_upper_bound();
```

Finally, one obtains:

```
obj
```

```
obj =
  VSDP conic programming problem with dimensions:

    [n,m] = size(obj.At)
     n    = 11 variables
       m  =  5 constraints

  and cones:

     K.l = 1
     K.q = [ 5, 5 ]

  obj.solutions.approximate:

      Solver 'sedumi': Normal termination, 0.3 seconds.

        c'*x = -2.592163292348387e+00
        b'*y = -2.592163288374707e+00


  obj.solutions.rigorous_lower_bound:

      Solver 'sedumi': Normal termination, 0.3 seconds, 1 iterations.

          fL = -2.592163308023824e+00

  obj.solutions.rigorous_upper_bound:
```

(continues on next page)

(continued from previous page)

```
     Normal termination, 0.0 seconds, 0 iterations.

         fU = -2.592163292358022e+00


 Detailed information:  'obj.info()'
```

CHAPTER

FIVE

SEMIDEFINITE PROGRAMMING

The primal standard form of a conic program with n_s symmetric positive semidefinite cones

$$\mathbb{S}_+^{s_j} := \left\{X \in \mathbb{R}^{s_j \times s_j} : \ X = X^T, \ v^T X v \geq 0, \ \forall v \in \mathbb{R}^{s_j}\right\}, \quad j = 1, \ldots, n_s.$$

is

$$\begin{array}{ll} \text{minimize} & \sum_{j=1}^{n_s} \langle C_j, X_j \rangle \\ \text{subject to} & \sum_{j=1}^{n_s} \langle A_{ij}, X_j \rangle = b_i, \quad i = 1, \ldots, m, \\ & X_j \in \mathbb{S}_+^{s_j}, \qquad\qquad\quad j = 1, \ldots, n_s, \end{array}$$

with symmetric $s_j \times s_j$ matrices A_{ij} and C_j. The dual problem form is

$$\begin{array}{ll} \text{maximize} & b^T y \\ \text{subject to} & Z_j := C_j - \sum_{i=1}^{m} y_i A_{ij} \in \mathbb{S}_+^{s_j}, \quad j = 1, \ldots, n_s. \end{array}$$

5.1 First SDP-Example

We consider an example from the CSDP User's Guide [4]:

$$\begin{array}{ll} \text{minimize} & \sum_{j=1}^{3} \langle C_j, X_j \rangle \\ \text{subject to} & \sum_{j=1}^{3} \langle A_{ij}, X_j \rangle = b_i, \quad i = 1, 2, \\ & X_1 \in \mathbb{S}_+^2, \\ & X_2 \in \mathbb{S}_+^3, \\ & X_3 \in \mathbb{S}_+^2, \end{array}$$

where $b = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$,

$$C_1^{s_1} = \begin{pmatrix} -2 & -1 \\ -1 & -2 \end{pmatrix}, \quad C_2^{s_2} = \begin{pmatrix} -3 & 0 & -1 \\ 0 & -2 & 0 \\ -1 & 0 & -3 \end{pmatrix}, \quad C_3^{s_3} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix},$$

$$A_{1,1}^{s_1} = \begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix}, \qquad A_{1,2}^{s_2} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \qquad A_{1,3}^{s_3} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix},$$

$$A_{2,1}^{s_1} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, \qquad A_{2,2}^{s_2} = \begin{pmatrix} 3 & 0 & 1 \\ 0 & 4 & 0 \\ 1 & 0 & 5 \end{pmatrix}, \qquad A_{2,3}^{s_3} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}.$$

In the vectorized format the corresponding coefficient matrix `At` and the primal objective vector `c` are

```
At{1} = [ 3; 1;
          1; 3;
          0; 0; 0;
          0; 0; 0;
          0; 0; 0;
          1; 0;
          0; 0 ];
At{2} = [ 0; 0;
          0; 0;
          3; 0; 1;
          0; 4; 0;
          1; 0; 5;
          0; 0;
          0; 1 ];
At = [At{:}];

b = [ 1;
      2 ];

c = [ -2; -1;
      -1; -2;
      -3;  0; -1;
       0; -2;  0;
      -1;  0; -3;
       0;  0;
       0;  0];
```

And the cone structure `K` for this problem is

```
K.s = [2 3 2];
obj = vsdp (At, b, c, K);
```

Before one starts with approximately solving the SDP, one can check for diagonal only SDP cones and convert them to linear cones. This is beneficial for two reasons: Firstly, storing linear cones requires less memory, and secondly, VSDP does not have to compute eigenvalues for the cone verification.

```
obj = obj.analyze (true);
```

```
warning: analyze: K.s(3) seems to only have diagonal elements.
warning: called from
    analyze>pattern1 at line 72 column 7
    analyze at line 50 column 5
 --> Convert it to LP block.
```

When calling `vsdp.analyze` with the argument `true`, all possible optimization are applied. Note that in the original example by Borchers [4] the last cone was already marked as diagonal only. This was only changed for the sake of demonstration.

Now we compute approximate solutions by using `vsdp.solve` and then rigorous error bounds by using `vsdp.rigorous_lower_bound` and `vsdp.rigorous_upper_bound`:

```
obj.options.VERBOSE_OUTPUT = false;
obj.solve('sdpt3') ...
   .rigorous_lower_bound() ...
   .rigorous_upper_bound()
```

```
ans =
  VSDP conic programming problem with dimensions:

    [n,m] = size(obj.At)
     n    = 11 variables
       m  =  2 constraints

  and cones:

     K.l = 2
     K.s = [ 2, 3 ]

  obj.solutions.approximate:

      Solver 'sdpt3': Normal termination, 0.5 seconds.

        c'*x = -2.749999966056186e+00
        b'*y = -2.750000014595577e+00


  obj.solutions.rigorous_lower_bound:

      Normal termination, 0.0 seconds, 0 iterations.

          fL = -2.750000014595577e+00

  obj.solutions.rigorous_upper_bound:

      Normal termination, 0.0 seconds, 0 iterations.

          fU = -2.749999966061941e+00



  Detailed information:  'obj.info()'
```

Those approximations match the true primal and dual optimal objective function value $\hat{f}_d = \hat{f}_d = -2.75$.

To compare the approximate solution `X`, `y`, and `Z` with the unique solution $\hat{X}$, $\hat{y}$, and $\hat{Z}$ from [4], the vectorized solution quantities `x` and `z` have to be transformed back to matrices by using `vsdp.smat` and the appropriate scaling factor `alpha`:

$$\hat{X} = \begin{pmatrix} 0.125 & 0.125 & & & & & \\ 0.125 & 0.125 & & & & & \\ & & 2/3 & 0 & 0 & & \\ & & 0 & 0 & 0 & & \\ & & 0 & 0 & 0 & & \\ & & & & & 0 & \\ & & & & & & 0 \end{pmatrix}$$

```
format short
alpha = 1/2;  % Invert scaling by "vdsp.svec"

x  = vsdp_indexable (full (obj.solutions.approximate.x), obj);
X1 = vsdp.smat ([], x.s(1), alpha) % SDP Block 1
X2 = vsdp.smat ([], x.s(2), alpha) % SDP Block 2
X3 = x.l                           % LP  Block
```

```
X1 =
   0.12500   0.12500
   0.12500   0.12500

X2 =
   0.66668   0.00000  -0.00002
   0.00000   0.00000   0.00000
  -0.00002   0.00000   0.00000

X3 =
   0.0000000090495
   0.0000000067871
```

$$\hat{y} = \begin{pmatrix} -0.75 \\ -1 \end{pmatrix},$$

```
y = obj.solutions.approximate.y
```

```
y =
  -0.75000
  -1.00000
```

$$\hat{Z} = \begin{pmatrix} 0.25 & -0.25 & & & & & \\ -0.25 & 0.25 & & & & & \\ & & 0 & 0 & 0 & & \\ & & 0 & 2 & 0 & & \\ & & 0 & 0 & 2 & & \\ & & & & & 0.75 & \\ & & & & & & 1 \end{pmatrix}$$

```
alpha = 1;  % Invert scaling by "vdsp.svec"

z  = vsdp_indexable (full (obj.solutions.approximate.z), obj);
Z1 = vsdp.smat ([], z.s(1), alpha) % SDP Block 1
Z2 = vsdp.smat ([], z.s(2), alpha) % SDP Block 2
Z3 = x.l                           % LP  Block
```

```
Z1 =
   0.25000  -0.25000
  -0.25000   0.25000

Z2 =
   0.00000   0.00000   0.00000
   0.00000   2.00000   0.00000
   0.00000   0.00000   2.00000

Z3 =
   0.0000000090495
   0.0000000067871
```

The computation of the rigorous lower bounds involves the computation of the smallest eigenvalues `Zl(j)` $= \lambda_{\min}([Z_j])$ for $j = 1, 2, 3$.

```
Zl = obj.solutions.rigorous_lower_bound.z'
```

```
Zl =
   0.75000   1.00000   0.00000   0.00000
```

```
Y  = obj.solutions.rigorous_lower_bound.y
```

```
intval Y =
   -0.7500
   -1.0000
```

Since all `Zl >= 0` it is proven that all matrices Z_j are in the interior of the cone $\mathcal{K}$ and `Y` is a rigorous enclosure of a dual strict feasible (near optimal) solution.

Analogous computations are performed for the rigorous upper bound. Here lower bounds on the smallest eigenvalue of the primal solution are computed `Xl(j)` $= \lambda_{\min}([X_j])$ for $j = 1, 2, 3$.

```
Xl = obj.solutions.rigorous_upper_bound.z'
```

```
Xl =
   0.0000000090495   0.0000000067871   0.0000000135747   0.0000000029910
```

The matrix `X` is a rigorous enclosure of a primal strict feasible (near optimal) solution and can be restored from the vectorized quantity `obj.solutions.rigorous_upper_bound.x` as shown for the approximate solution. We omit the display of the interval matrix `X` for brevity.

Since all `Xl` are non-negative, strict feasibility for the primal problem is proved. Thus strong duality holds for this example.

```
clear all
```

5.2 Second SDP-Example

Now we consider the following example (see [11]):

$$\begin{array}{ll} \text{minimize} & \langle C(\delta), X\rangle \\ \text{subject to} & \langle A_1, X\rangle = 1, \\ & \langle A_2, X\rangle = \varepsilon, \\ & \langle A_3, X\rangle = 0, \\ & \langle A_4, X\rangle = 0, \\ & X \in \mathbb{S}^3_+, \end{array}$$

with Lagrangian dual

$$\begin{array}{ll} \text{maximize} & y_1 + \varepsilon y_2 \\ \text{subject to} & Z(\delta) := C(\delta) - \sum_{i=1}^4 A_i y_i \in \mathbb{S}^3_+, \\ & y \in \mathbb{R}^4, \end{array}$$

where

```
c = @(DELTA) ...
    [  0;   1/2;    0;
      1/2; DELTA;   0;
       0;    0;   DELTA ];

At = {};
At{1} = [ 0; -1/2; 0;
         -1/2;  0;  0;
           0;   0;  0 ];
At{2} = [ 1; 0; 0;
          0; 0; 0;
          0; 0; 0 ];
At{3} = [ 0; 0; 1;
          0; 0; 0;
          1; 0; 0 ];
At{4} = [ 0; 0; 0;
          0; 0; 1;
          0; 1; 0 ];
At = [At{:}];

b = @(EPSILON) [1; EPSILON; 0; 0];

K.s = 3;
```

The linear constraints of the primal problem form imply

$$X(\varepsilon) = \begin{pmatrix} \varepsilon & -1 & 0 \\ -1 & X_{22} & 0 \\ 0 & 0 & X_{33} \end{pmatrix} \in \mathbb{S}^3_+$$

iff $X_{22} \geq 0$, $X_{33} \geq 0$, and $\varepsilon X_{22} - 1 \geq 0$. The conic constraint of the dual form is

$$Z(\delta) = \begin{pmatrix} -y_2 & \frac{1+y_1}{2} & -y_3 \\ \frac{1+y_1}{2} & \delta & -y_4 \\ -y_3 & -y_4 & \delta \end{pmatrix} \in \mathbb{S}^3_+.$$

Hence, for

- $\varepsilon \leq 0$: the problem is **primal infeasible** $\hat{f}_p = +\infty$.
- $\delta < 0$: the problem is **dual infeasible** $\hat{f}_d = -\infty$.
- $\varepsilon = \delta = 0$: the problem is **ill-posed** and there is a duality gap with $\hat{f}_p = +\infty$ and $\hat{f}_d = -1$.
- $\varepsilon > 0$ and $\delta > 0$: the problem is **feasible** with $\hat{f}_p = \hat{f}_d = -1 + \delta/\varepsilon$.

To obtain a feasible solution, we set $\delta = 10^{-2}$ and $\varepsilon = 2\delta$. Thus the primal and dual optimal objective function value is $\hat{f}_p = \hat{f}_d = -0.5$ and one can start the computations with VSDP.

```
DELTA   = 1e-4;
EPSILON = 2 * DELTA;

obj = vsdp (At, b(EPSILON), c(DELTA), K);
obj.options.VERBOSE_OUTPUT = false;
obj.solve('sdpt3') ...
   .rigorous_lower_bound() ...
   .rigorous_upper_bound()
```

```
ans =
  VSDP conic programming problem with dimensions:

    [n,m] = size(obj.At)
     n    = 6 variables
       m  = 4 constraints

  and cones:

     K.s = [ 3 ]

  obj.solutions.approximate:

      Solver 'sdpt3': Normal termination, 0.6 seconds.

        c'*x = -4.999999947476755e-01
        b'*y = -5.000000065021177e-01


  obj.solutions.rigorous_lower_bound:

      Normal termination, 0.0 seconds, 0 iterations.

          fL = -5.000000065021177e-01

  obj.solutions.rigorous_upper_bound:

      Solver 'sdpt3': Normal termination, 0.7 seconds, 1 iterations.

          fU = -4.999999933708725e-01



  Detailed information:  'obj.info()'
```

Everything works as expected. VSDP computes finite rigorous lower and upper bounds `fU` and `fL`. Weak duality, e.g. $\hat{f}_p \geq \hat{f}_d$ and `fU >= fL`, holds for the approximate and rigorous solutions. The accuracy of rigorous the error bounds can again be measured by

```
format shorte
fL = obj.solutions.rigorous_lower_bound.f_objective(1);
fU = obj.solutions.rigorous_upper_bound.f_objective(2);
mu = (fU - fL) / max (1, (abs (fU) + abs(fL)) / 2)
```

```
mu =    1.3131e-08
```

Nevertheless, successful termination reported by an approximate solver gives no guarantee on the quality of the computed solution. Only `fU` and `fL` are reliable results, which are computed by the functions `vsdp.rigorous_lower_bound` and `vsdp.rigorous_upper_bound`, respectively.

To emphasize this, one can apply SeDuMi to the same problem:

```
obj.options.SOLVER = 'sedumi';
obj.solve() ...
   .rigorous_lower_bound () ...
   .rigorous_upper_bound ()
```

```
ans =
  VSDP conic programming problem with dimensions:

    [n,m] = size(obj.At)
     n    = 6 variables
       m  = 4 constraints

  and cones:

     K.s = [ 3 ]

  obj.solutions.approximate:

      Solver 'sedumi': Normal termination, 0.5 seconds.

        c'*x = -4.999990761443555e-01
        b'*y = -4.999968121457571e-01


  obj.solutions.rigorous_lower_bound:

      Solver 'sedumi': Normal termination, 0.5 seconds, 1 iterations.

          fL = -5.000035760394096e-01

  obj.solutions.rigorous_upper_bound:

      Solver 'sedumi': Normal termination, 0.5 seconds, 1 iterations.

          fU = -4.999953448740588e-01



  Detailed information:  'obj.info()'
```

SeDuMi terminates without any warning, but the approximate results are poor. Since the approximate primal optimal objective function value is smaller than the dual one. Weak duality is not satisfied.

```
f_obj = obj.solutions.approximate.f_objective;

f_obj(1) >= f_obj(2)
```

```
ans = 0
```

As already mentioned, weak duality holds for the rigorous error bounds by VSDP:

```
fL = obj.solutions.rigorous_lower_bound.f_objective(1);
fU = obj.solutions.rigorous_upper_bound.f_objective(2);

fU >= fL
```

```
ans = 1
```

In general the quality of the rigorous error bounds strongly depends on the computed approximate solution and therefore on the used approximate conic solver. For example compare the accuracy of SeDuMi below with SDPT3 above:

```
format short e
acc_mu = (fU - fL) / max(1.0, (abs(fU) + abs(fL)) / 2)
```

```
acc_mu =    8.2312e-06
```

CHAPTER

SIX

A PRIORI BOUNDS

In many practical applications the order of the magnitude of a primal or dual optimal solution is known a priori. This is the case in many combinatorial optimization problems, or, for instance, in truss topology design where the design variables such as bar volumes can be roughly bounded. If such bounds are available they can speed up the computation of rigorous error bounds for the optimal value substantially, see [10].

For linear programming problems the upper bound for the variable x^l is a vector $\bar{x}$ such that $x^l \leq \bar{x}$. For second-order cone programming the upper bounds for block variables x_i^q with $i = 1, \ldots, n_q$ can be entered as a vector of upper bounds $\overline{\lambda}_i$ of the largest eigenvalues

$$\lambda_{\max}(x_i^q) = (x_i^q)_1 + ||(x_i^q)_{:}||_2.$$

Similarly, in semidefinite programs upper bounds for the primal variables X_j^s can be entered as a vector of upper bounds of the largest eigenvalues $\lambda_{\max}(X_j^s)$, $j = 1, \ldots, n_s$. An upper bound $\bar{y}$ for the dual optimal solution y is a vector which is elementwise larger than y. Analogously, for conic programs with free variables the upper bound can be entered as a vector $\bar{x}$ such that $|x^f| \leq \bar{x}$.

As an example, we consider the *Second SDP-Example* with an upper bound $xu = 10^5$ for $\lambda_{\max}(X)$.

```
DELTA   = 1e-4;
EPSILON = 2 * DELTA;

c = [  0;   1/2;    0;
      1/2; DELTA;   0;
       0;    0;   DELTA ];

At = {};
At{1} = [ 0; -1/2; 0;
         -1/2;  0;  0;
           0;   0;  0 ];
At{2} = [ 1; 0; 0;
           0; 0; 0;
           0; 0; 0 ];
At{3} = [ 0; 0; 1;
           0; 0; 0;
           1; 0; 0 ];
At{4} = [ 0; 0; 0;
           0; 0; 1;
           0; 1; 0 ];
At = [At{:}];

b = [1; EPSILON; 0; 0];

K.s = 3;
```

(continues on next page)

(continued from previous page)

```
obj = vsdp (At, b, c, K);
obj.options.VERBOSE_OUTPUT = false;
```

Now we compute approximate solutions by using `vsdp.solve` and then rigorous error bounds by using `vsdp.rigorous_lower_bound` and `vsdp.rigorous_upper_bound`:

```
xu = 1e5;
yu = 1e5 * [1 1 1 1]';

obj.solve('sedumi') ...
   .rigorous_lower_bound(xu) ...
   .rigorous_upper_bound(yu)
```

```
ans =
  VSDP conic programming problem with dimensions:

    [n,m] = size(obj.At)
     n    = 6 variables
       m  = 4 constraints

  and cones:

     K.s = [ 3 ]

  obj.solutions.approximate:

      Solver 'sedumi': Normal termination, 0.5 seconds.

        c'*x = -4.999990761443555e-01
        b'*y = -4.999968121457571e-01


  obj.solutions.rigorous_lower_bound:

      Normal termination, 0.0 seconds, 0 iterations.

          fL = -5.000869997554969e-01

  obj.solutions.rigorous_upper_bound:

      Normal termination, 0.0 seconds, 0 iterations.

          fU = -4.997496499568883e-01



  Detailed information:  'obj.info()'
```

yielding rigorous error bounds with reasonable accuracy:

```
format shorte
fL = obj.solutions.rigorous_lower_bound.f_objective(1);
fU = obj.solutions.rigorous_upper_bound.f_objective(2);
mu = (fU - fL) / max (1, (abs (fU) + abs(fL)) / 2)
```

```
mu =    3.3735e-04
```

The advantage of rigorous error bounds computed with a priori bounds on the solution is, that the computational effort can be neglected.

CHAPTER

SEVEN

RIGOROUS CERTIFICATES OF INFEASIBILITY

The functions `vsdp.rigorous_lower_bound` and `vsdp.rigorous_upper_bound` prove strict feasibility and compute rigorous error bounds. For the verification of infeasibility the functions `vsdp.check_primal_infeasible` and `vsdp.check_dual_infeasible` can be applied. In this section we show how to use these functions.

7.1 Theorems of alternatives

Both functions are based upon a theorem of alternatives [13]. Such a theorem states that for two systems of equations or inequalities, one or the other system has a solution, but not both. A solution of one of the systems is called a certificate of infeasibility for the other which has no solution.

For a conic program those two theorems of alternatives are as follows:

Primal Infeasibility Theorem

Suppose that some $\tilde{y}$ satisfies $-A^T y \in \mathcal{K}^*$ and $b^T \tilde{y} > 0$. Then the system of primal constraints $Ax = b$ with $x \in \mathcal{K}$ has no solution.

Dual Infeasibility Theorem

Suppose that some $\tilde{x} \in \mathcal{K}$ satisfies $A\tilde{x} = 0$ and $c^T \tilde{x} < 0$. Then the system of dual constraints $c - A^T y \in \mathcal{K}^*$ has no solution.

For a proof, see [13]. The first theorem is the foundation of `vsdp.check_primal_infeasible` and the second of `vsdp.check_dual_infeasible`.

7.2 Example: primal infeasible SOCP

We consider a slightly modified second-order cone problem from [28] (Example 2.4.2)

$$\begin{aligned} \text{minimize} \quad & \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} x, \\ \text{subject to} \quad & \begin{pmatrix} 1 & 0 & 0.5 \\ 0 & 1 & 0 \end{pmatrix} x = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \\ & x \in \mathbb{L}^3, \end{aligned}$$

with its dual problem

$$\begin{aligned} \text{maximize} \quad & \begin{pmatrix} 0 & 1 \end{pmatrix} y, \\ \text{subject to} \quad & \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} - \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0.5 & 0 \end{pmatrix} y \in \mathbb{L}^3. \end{aligned}$$

The primal problem is infeasible, while the dual problem is unbounded. One can easily prove this fact by assuming that there exists a primal feasible point x. This point has to satisfy $x_3 = -2x_1$ and therefore $x_1 \geq \sqrt{x_2^2 + (-2x_1)^2}$. From the second equality constraint we get $x_2 = 1$ yielding the contradiction $x_1 \geq \sqrt{1 + 4x_1^2}$. Thus, the primal problem has no feasible solution.

The set of dual feasible points is given by $y_1 \leq 0$ and $y_2 = -\frac{\sqrt{3}}{2} y_1$. Thus the maximization of the dual problem yields $\hat{f}_p = +\infty$ for $y = \alpha \begin{pmatrix} -1 & \sqrt{3}/2 \end{pmatrix}^T$ with $\alpha \to +\infty$.

To show primal infeasibility using VSDP, one first has to specify the input data:

```
A = [1, 0, 0.5;
     0, 1, 0];
b = [0; 1];
c = [0; 0; 0];
K.q = 3;
```

Using the approximate solver SDPT3, we obtain a rigorous certificate of infeasibility with the routine `vsdp.check_primal_infeasible`:

```
obj = vsdp(A,b,c,K).solve ('sdpt3') ...
                   .check_primal_infeasible () ...
                   .check_dual_infeasible ();
```

```
 num. of constraints =  2
 dim. of socp   var  =  3,   num. of socp blk  =  1
*****************************************************************
   SDPT3: Infeasible path-following algorithms
*****************************************************************
 version  predcorr  gam  expon  scale_data
    NT      1      0.000   1        0
it pstep dstep pinfeas dinfeas  gap      prim-obj      dual-obj    cputime
-----------------------------------------------------------------
 0|0.000|0.000|1.0e+00|2.1e+00|3.7e+00| 0.000000e+00  0.000000e+00| 0:0:00| chol ␣
 ↪1  1
 1|0.640|1.000|3.6e-01|1.0e-01|3.0e-01| 0.000000e+00  1.979648e+00| 0:0:00| chol ␣
 ↪1  1
 2|0.066|0.638|3.4e-01|4.3e-02|2.4e+00| 0.000000e+00  2.346369e+02| 0:0:00| chol ␣
 ↪1  1
 3|0.004|1.000|3.4e-01|1.0e-03|2.9e+04| 0.000000e+00  2.170266e+06| 0:0:00| chol ␣
 ↪2  2
 4|0.005|1.000|3.4e-01|9.9e-05|2.1e+08| 0.000000e+00  1.165746e+10| 0:0:00| chol ␣
 ↪2  2
 5|0.004|1.000|3.4e-01|0.0e+00|5.3e+11| 0.000000e+00  3.618635e+13| 0:0:00| chol ␣
 ↪1  1
 6|0.002|1.000|3.4e-01|0.0e+00|2.2e+15| 0.000000e+00  1.736991e+17| 0:0:00|
  sqlp stop: primal or dual is diverging, 9.1e+16
-----------------------------------------------------------------
 number of iterations   =  6
 Total CPU time (secs)  = 0.22
 CPU time per iteration = 0.04
 termination code       =  3
 DIMACS: 3.4e-01  0.0e+00  0.0e+00  0.0e+00  -1.0e+00  1.3e-02
-----------------------------------------------------------------
```

The output of the solver is quite verbose and can be suppressed by setting `obj.options.VERBOSE_OUTPUT` to `false`. Important is the message of the SDPT3 solver:

> sqlp stop: primal or dual is diverging

which supports the theoretical consideration about the unboundedness of the dual problem. As expected, `vsdp.check_primal_infeasible` proves the infeasiblity of the primal problem

```
obj.solutions.certificate_primal_infeasibility
```

```
ans =
      Normal termination, 0.0 seconds.

      A certificate of primal infeasibility 'y' was found.
      The conic problem is primal infeasible.
```

while dual infeasibility cannot be shown:

```
obj.solutions.certificate_dual_infeasibility
```

```
ans =
      Normal termination, 0.0 seconds.

      NO certificate of dual infeasibility was found.
```

The interval quantity `y`, the rigorous certificate of primal infeasiblity, matches the theoretical considerations. It diverges to infinite values:

```
y = obj.solutions.certificate_primal_infeasibility.y
```

```
intval y =
  1.0e+017 *
   -2.0060
    1.7369
```

and the first entry of `y` multiplied by $-\sqrt{3}/2$ is almost the second entry of `y`:

```
y(1) * -sqrt(3)/2
```

```
intval ans =
  1.0e+017 *
    1.7372
```

The following check is already done by `vsdp.check_primal_infeasible`, but for illustration we evaluate the conditions to prove primal infeasibility from the first theorem of alternatives $-A^T y \in \mathcal{K}^*$ and $b^T \tilde{y} > 0$:

```
z = -A' * y;
z(1) >= norm (z(2:end))  % Check z to be in the Lorentz-cone.
```

```
ans = 1
```

```
b' * y > 0
```

```
ans = 1
```

Note that the rigorous certificate of infeasiblity is not necessarily unique. Thus VSDP might proof a different `y`, when used with another approximate solver. Compare for example the certificate computed by SeDuMi:

```
obj = vsdp(A,b,c,K);
obj.options.VERBOSE_OUTPUT = false;
obj.solve ('sedumi') ...
   .check_primal_infeasible ();
y = obj.solutions.certificate_primal_infeasibility.y
```

```
intval y =
   -2.3924
    1.0000
```

```
z = -A' * y;
z(1) >= norm (z(2:end))  % Check z to be in the Lorentz-cone.
```

```
ans = 1
```

```
b' * y > 0
```

```
ans = 1
```

which is also perfectly valid.

7.3 Example: primal infeasible SDP

In the following we consider another conic optimization problem from [10]. The two SDP constraints of that problem depend on two arbitrary fixed chosen parameters $\delta = 0.1$ and $\varepsilon = -0.01$.

\begin{equation} \begin{array}{ll} \text{minimize} & \left\langle \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, X \right\rangle \\ \text{subject to} & \left\langle \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, X \right\rangle = \varepsilon, \\ & \left\langle \begin{pmatrix} 0 & 1 \\ 1 & \delta \end{pmatrix}, X \right\rangle = 1, \\ & X \in \mathbb{S}_{+}^{2}. \end{array} \end{equation}

The problem data is entered in the VSDP 2006 format:

```
clear all;
EPSILON = -0.01;
DELTA = 0.1;
blk(1,:) = {'s'; 2};
C{1,1} = [0 0; 0 0];
A{1,1} = [1 0; 0 0];
A{2,1} = [0 1; 1 DELTA];
b = [EPSILON; 1];

obj = vsdp (blk, A, C, b);
obj.options.VERBOSE_OUTPUT = false;
```

The first constraint yields $x_1 = \varepsilon < 0$. This is a contradiction to $X \in \mathbb{S}^2_+$, thus the problem is primal infeasible. The dual problem is

$$\begin{aligned} &\text{maximize} && \varepsilon y_1 + y_2 \\ &\text{subject to} && \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} - y_1 \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} - y_2 \begin{pmatrix} 0 & 1 \\ 1 & \delta \end{pmatrix} = \begin{pmatrix} -y_1 & -y_2 \\ -y_2 & -\delta y_2 \end{pmatrix} \in \mathbb{S}^2_+, \\ & && y_1, y_2 \in \mathbb{R}. \end{aligned}$$

For dual feasibility the first principal minor of the dual constraint must fulfill $-y_1 \geq 0$ and the entire matrix $y_2(\delta y_1 - y_2) \geq 0$. The objective function goes to $+\infty$ for $y_1 \to -\infty$ and $y_2 = 0$. Thus the dual problem is unbounded and each point $\hat{y} = (y_1, 0)$ with $y_1 \leq 0$ is a certificate of primal infeasibility.

To compute a rigorous certificate of primal infeasiblity using VSDP, one can make use of the `vsdp.check_primal_infeasible`-function:

```
obj = obj.solve ('sdpt3') ...
         .rigorous_upper_bound () ...
         .check_primal_infeasible () ...
         .check_dual_infeasible ()
```

```
warning: rigorous_upper_bound: Conic solver could not find a solution for␣
 ↪perturbed problem
warning: called from
    rigorous_upper_bound>rigorous_upper_bound_infinite_bounds at line 191 column 5
    rigorous_upper_bound at line 58 column 7
obj =
  VSDP conic programming problem with dimensions:

    [n,m] = size(obj.At)
     n    = 3 variables
       m  = 2 constraints

  and cones:

     K.s = [ 2 ]

  obj.solutions.approximate:

      Solver 'sdpt3': Primal infeasible, 0.4 seconds.

        c'*x = 0.000000000000000e+00
        b'*y = 1.000000000000000e+00


  Compute a rigorous lower bound:

    'obj = obj.rigorous_lower_bound()'
  obj.solutions.rigorous_upper_bound:

      Solver 'sdpt3': Unknown, 0.4 seconds, 1 iterations.

          fU = Inf


  obj.solutions.certificate_primal_infeasibility:

      Normal termination, 0.0 seconds.
```

(continues on next page)

(continued from previous page)

```
    A certificate of primal infeasibility 'y' was found.
    The conic problem is primal infeasible.


  obj.solutions.certificate_dual_infeasibility:

    Normal termination, 0.0 seconds.

    NO certificate of dual infeasibility was found.



  Detailed information:  'obj.info()'
```

While computing an approximate solution to this problem, SDPT3 already detects potential primal infeasibility. Trying to compute a rigorous upper error bound by `vsdp.rigorous_upper_bound` fails. This emphasizes the warning at the beginning of the output and the upper error bound is set to infinity (`fU = Inf`).

Using the approximate dual solution

```
yt = obj.solutions.approximate.y
```

```
yt =
  -100.007983163
    -0.000079832
```

the VSDP-function `vsdp.check_primal_infeasible` tries to prove a rigorous certificate of primal infeasibility. This is done by a rigorous evaluation of the theorem of alternatives using interval arithmetic:

```
format infsup
yy = obj.solutions.certificate_primal_infeasibility.y
```

```
intval yy =
[ -100.0080, -100.0079]
[   -0.0001,   -0.0000]
```

According to the Primal Infeasibility Theorem (cf. *Theorems of alternatives*) $\langle \tilde{y}, b \rangle$ is positive:

```
obj.b' * yy
```

```
intval ans =
[    1.0000,    1.0000]
```

and $-A^* \tilde{y}$ lies in the cone of symmetric positive semidefinite matrices $\mathbb{S}^2_+$:

```
-yy(1) * A{1,1} - yy(2) * A{2,1}
```

```
intval ans =
[  100.0079,  100.0080] [    0.0000,    0.0001]
[    0.0000,    0.0001] [    0.0000,    0.0001]
```

It was shown, that the problem is unbounded, but not infeasible. Therefore it is clear, that VSDP cannot prove a rigorous certificate of dual infeasiblity by `vsdp.check_dual_infeasible`:

```
obj.solutions.certificate_dual_infeasibility
```

```
ans =
      Normal termination, 0.0 seconds.

      NO certificate of dual infeasibility was found.
```

CHAPTER

EIGHT

FREE VARIABLES

Free variables often occur in practice. Handling free variables in interior-point algorithms is a pending issue (see for example [2, 3, 15, 17]). Frequently problems with free variables are converted into one with restricted variables by representing the free variables as a difference of two non-negative variables. This approach increases the problem size and introduces ill-posedness, which may lead to numerical difficulties.

For example we consider the test problem *nb_L1* from the DIMACS test library [20]. The problem originates from side lobe minimization in antenna engineering. This is a second-order cone programming problem with 915 equality constraints, 793 SOCP blocks each of size 3, and 797 non-negative variables. Moreover, the problem has two free variables that are described as the difference of four non-negative variables. This problem can be loaded from the `test` directory of VSDP.

SDPT3 solves the problem without warnings, although it is ill-posed according to Renegar's definition (see [21]):

```
load (fullfile ('..', 'test', 'nb_L1.mat'));
obj = vsdp (A, b, c, K);
obj.options.VERBOSE_OUTPUT = false;
obj.solve('sdpt3') ...
   .rigorous_lower_bound () ...
   .rigorous_upper_bound ()
```

```
warning: rigorous_lower_bound: Conic solver could not find a solution for␣
 ↪perturbed problem
warning: called from
    rigorous_lower_bound at line 174 column 5
ans =
  VSDP conic programming problem with dimensions:

    [n,m] = size(obj.At)
     n    = 3176 variables
       m  =  915 constraints

  and cones:

     K.l = 797
     K.q = [ 793 cones (length = 2379) ]

  obj.solutions.approximate:

      Solver 'sdpt3': Normal termination, 10.6 seconds.

        c'*x = -1.301227063328808e+01
        b'*y = -1.301227079584390e+01
```

(continues on next page)

(continued from previous page)

```
  obj.solutions.rigorous_lower_bound:

      Solver 'sdpt3': Unknown, 12.9 seconds, 1 iterations.

          fL = -Inf

  obj.solutions.rigorous_upper_bound:

      Solver 'sdpt3': Normal termination, 32.7 seconds, 2 iterations.

          fU = -1.301227062881248e+01


  The rigorous lower bound is infinite, check dual infeasibility:

    'obj = obj.check_dual_infeasible()'


  Detailed information:  'obj.info()'
```

These results reflect that the interior of the dual feasible solution set is empty. An ill-posed problem has the property that the distance to primal or dual infeasibility is zero. As above, if the distance to dual infeasibility is zero, then there are sequences of dual infeasible problems with input data converging to the input data of the original problem. Each problem of the sequence is dual infeasible and thus has the dual optimal solution $-\infty$. Hence, the result $-\infty$ of `vsdp.rigorous_lower_bound` is exactly the limit of the optimal values of the dual infeasible problems and reflects the fact that the distance to dual infeasibility is zero. This demonstrates that the infinite bound computed by VSDP is sharp, when viewed as the limit of a sequence of infeasible problems. We have a similar situation if the distance to primal infeasibility is zero.

If the free variables are not converted into restricted ones, then the problem is well-posed and a rigorous finite lower bound can be computed:

```
load (fullfile ('..', 'test', 'nb_L1free.mat'));
obj = vsdp (A, b, c, K);
obj.options.VERBOSE_OUTPUT = false;
obj.solve('sdpt3') ...
   .rigorous_lower_bound () ...
   .rigorous_upper_bound ()
```

```
ans =
  VSDP conic programming problem with dimensions:

    [n,m] = size(obj.At)
     n    = 3174 variables
       m  =  915 constraints

  and cones:

     K.f = 2
     K.l = 793
     K.q = [ 793 cones (length = 2379) ]

  obj.solutions.approximate:
```

(continues on next page)

(continued from previous page)

```
    Solver 'sdpt3': Normal termination, 11.1 seconds.

      c'*x = -1.301227062100383e+01
      b'*y = -1.301227081903705e+01


obj.solutions.rigorous_lower_bound:

    Solver 'sdpt3': Normal termination, 14.9 seconds, 1 iterations.

        fL = -1.301227081922505e+01

obj.solutions.rigorous_upper_bound:

    Normal termination, 2.7 seconds, 0 iterations.

        fU = -1.301227062000819e+01



Detailed information:  'obj.info()'
```

Therefore, without splitting the free variables, we get rigorous finite lower and upper bounds of the exact optimal value with an accuracy of about eight decimal digits. Moreover, verified interior solutions are computed for both the primal and the dual problem, proving strong duality.

Part II

Back matter

CHAPTER

NINE

NUMERICAL RESULTS

In this section, we present statistics for the numerical results obtained by VSDP for conic programming problems. The tests were performed using approximations computed by the conic solvers: CSDP, MOSEK, SDPA, SDPT3, and SeDuMi. For second-order cone programming problems only MOSEK, SDPT3, and SeDuMi were used. The solvers have been called with their default parameters. Almost all of the problems that could not be solved with a guaranteed accuracy of about 10^{-7} are known to be ill-posed (cf. [19]).

In particular, the results were obtained by using the following two systems:

- **System 1**
  - GNU Octave (4.4.1)
  - CPU: Intel(R) Xeon(R) E3-1220 (4 cores)
  - RAM: 12 GB
  - OS: Linux (openSUSE 15.0)
  - Interval arithmetic: INTLAB 11
  - Conic Solver: CSDP (6.2.0), SDPA (7.3.8), SDPT3 (4.0), SeDuMi (1.32)
- **System 2**
  - MATLAB(R) (R2018b)
  - CPU: Intel(R) Xeon(R) E5-2640v3 (8 cores)
  - RAM: 128 GB
  - OS: Linux (Ubuntu 18.04)
  - Interval arithmetic: INTLAB 11
  - Conic Solver: MOSEK (8.1.0.62), SDPT3 (4.0)

The relative accuracy of two numbers is measured by $\$\mu(a,b) := \frac{a-b}{\max\{1.0, (|a|+|b|)/2\}}.\$$

Notice that we do not use the absolute value of $a-b$. Hence, a negative sign implies that $a < b$.

9.1 SDPLIB

In the following, we describe the numerical results for 92 problems from the SDPLIB suite of Borchers [5]. In [8] it is shown that four problems are infeasible and 32 problems are ill-posed.

VSDP could compute rigorous bounds of the optimal values for all feasible well-posed problems and verify the existence of strictly primal and dual feasible solutions. Hence, strong duality is proved. For the 32 ill-posed problems VSDP has computed the upper bound $\overline{f_d} = \text{Inf}$, which reflects the fact that the distance to the next primal infeasible problem is zero. For the four infeasible problems VSDP could compute rigorous certificates of infeasibility. Detailed numerical results can be found in the tables for System 1 and System 2, where the computed rigorous upper bound $\overline{f_d}$, the rigorous lower bound $\underline{f_p}$, and the rigorous error bound $\mu(\overline{f_d}, \underline{f_p})$ are displayed. We have set $\mu(\overline{f_d}, \underline{f_p}) = \text{NaN}$ if the upper or the lower bound is infinite. Both tables also contain running times in seconds, where t_s is the time for computing the approximations and $\overline{t}$ and $\underline{t}$ are the times for computing the upper and the lower rigorous error bounds, respectively.

Some major characteristics the numerical results for the SDPLIB are summarized by the following figures.

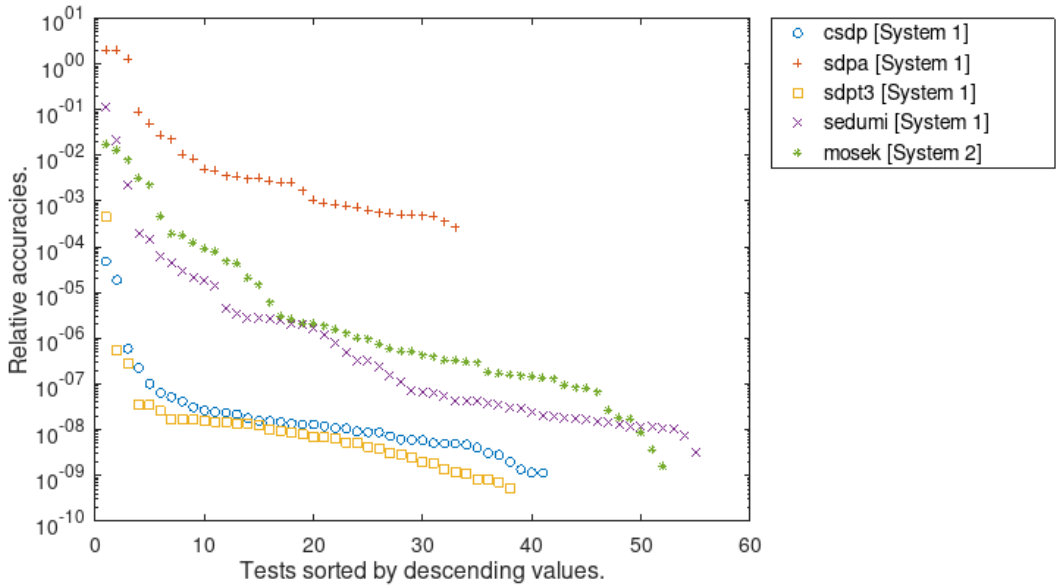


Figure 1: Relative accuracies $\mu(\overline{f_d}, \underline{f_p})$. Only results for which both rigorous error bounds were computed are taken into account. With the exception of SDPA, all approximate conic solvers can compute rigorous error bounds with 7 or 8 significant decimal digits.

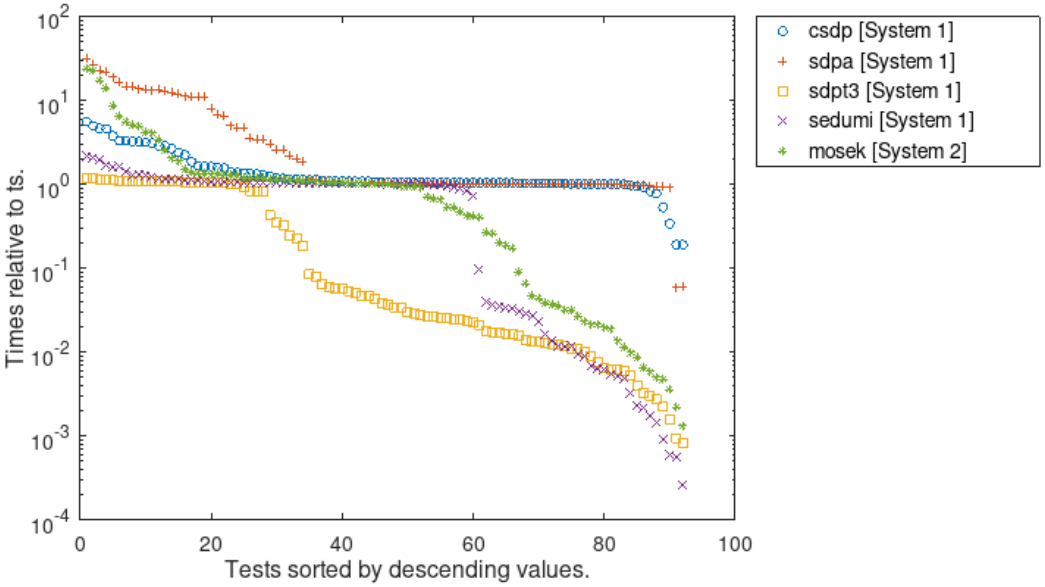


Figure 2: Computation times for $\underline{t}$ relative to t_s.

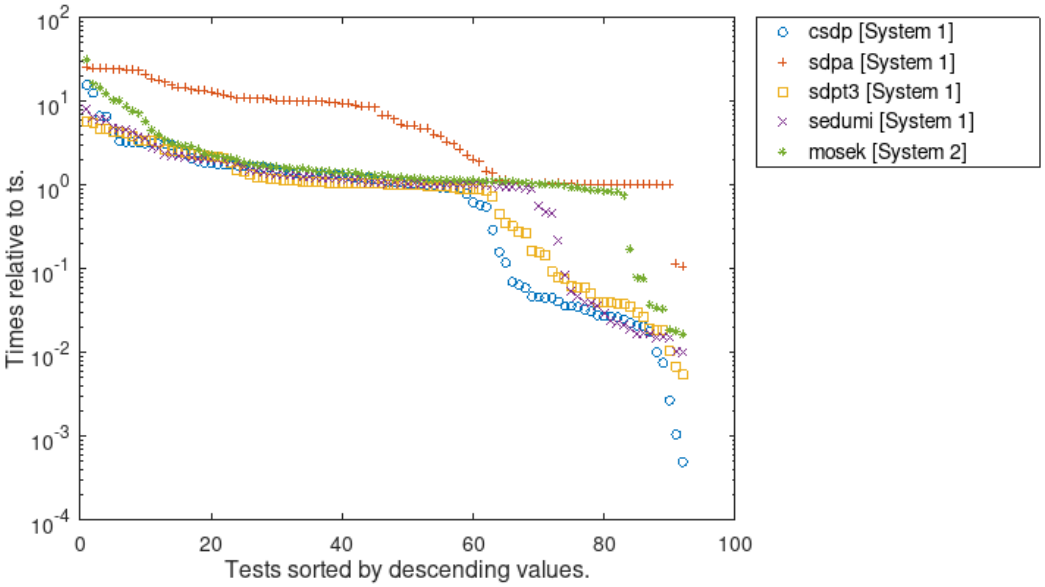


Figure 3: Computation times for $\bar{t}$ relative to t_s.

Furthermore, the figures show, that the error bounds as well as the time ratios depend significantly on the used conic solver. Even the largest problem *MaxG60* with about 24 million variables and 7000 constraints can be solved rigorously by VSDP with high accuracy and in a reasonable time.

9.2 SPARSE_SDP

In this section a statistic of the numerical results for problems from structural and topological optimization is presented. Structural and especially free material optimization gained more and more interest in the recent years. The most prominent example is the design of ribs in the leading edge of the Airbus A380. We performed tests on problems from the test library collected by Kočvara. This is a collection of 26 sparse semidefinite programming problems. More details on these problems can be found in [16, 26, 27]. For 22 problems out of this collection VSDP could compute a rigorous primal and dual ε-optimal solution, using SeDuMi as approximate solver. The largest problem that was rigorously solved by VSDP is *shmup5*. This problem has 1800 equality constraints and 13 million variables.

Detailed results can be found in the tables for System 1 and System 2. A statistic of these numerical experiments is given in the following figures.

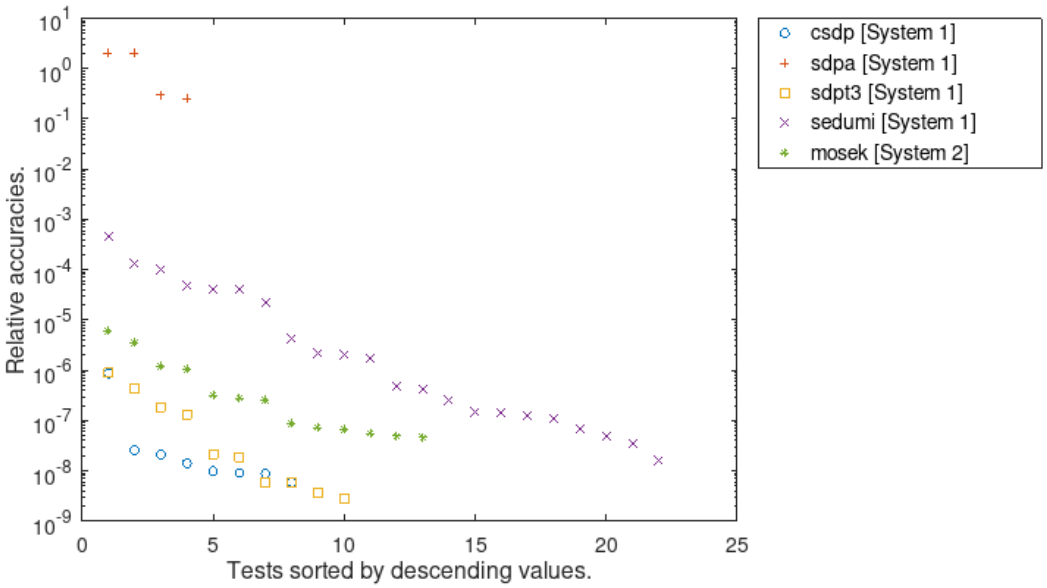


Figure 4: Relative accuracies $\mu(\overline{f_d}, \underline{f_p})$. Only results for which both rigorous error bounds were computed are taken into account.

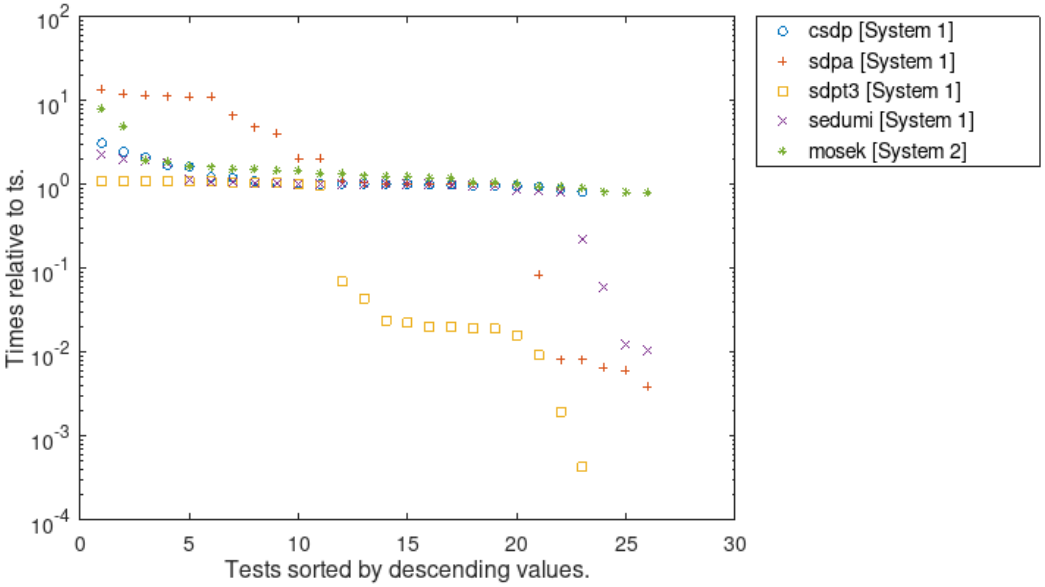


Figure 5: Computation times for $\underline{t}$ relative to t_s.

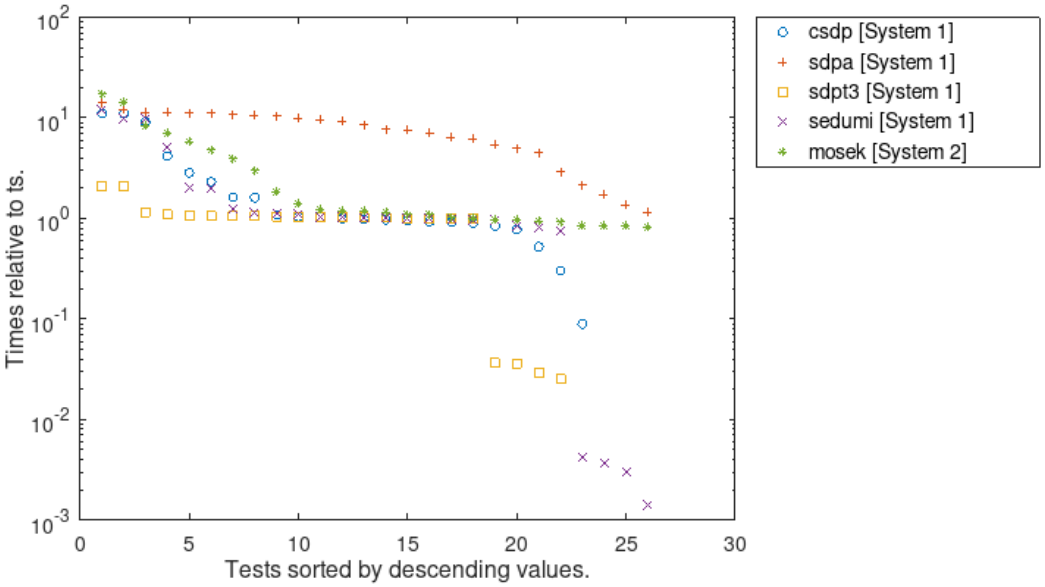


Figure 6: Computation times for $\bar{t}$ relative to t_s.

9.3 DIMACS

We present some statistics of numerical results for the DIMACS test library of semidefinte-quadratic-linear programs. This library was assembled for the purposes of the 7-th DIMACS Implementation Challenge. There are 47 challenging problems that are divided into 12 groups. For details see [20]. In each group there are about five instances, from routinely solvable ones to those at or beyond the capabilities of current solvers. Due to the large problem sizes this test library was only run on System 2 using MOSEK and the problem *fap25* had to be omitted in our test.

One of the largest problems which could be solved by VSDP is the problem *torusg3-15*, with 3375 equality constraints and about 5 million variables.

Detailed results can be found in the table for System 2. A statistic of these numerical experiments is given in the following figures.

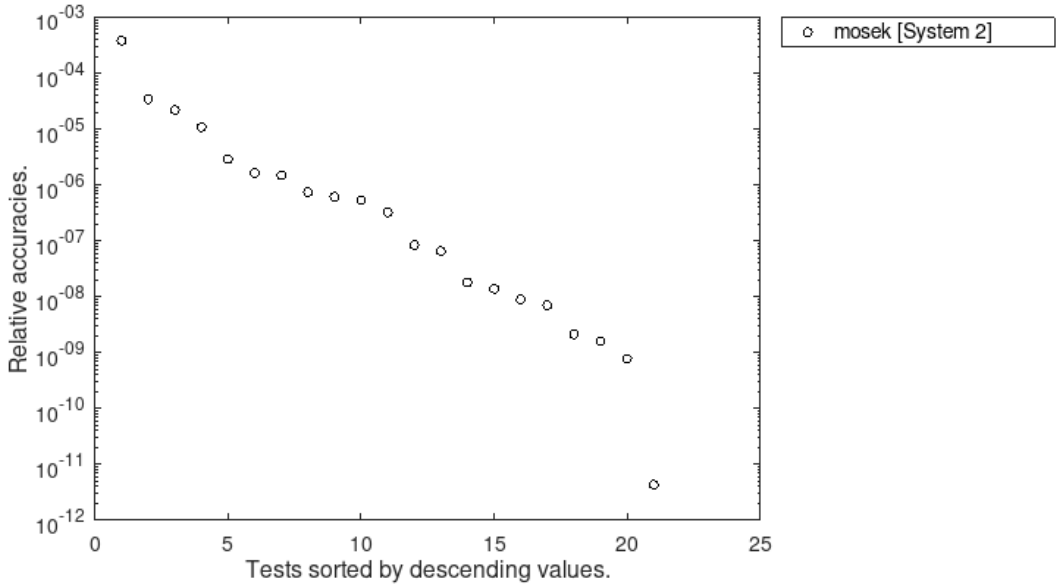


Figure 7: Relative accuracies $\mu(\overline{f_d}, \underline{f_p})$. Only results for which both rigorous error bounds were computed are taken into account.

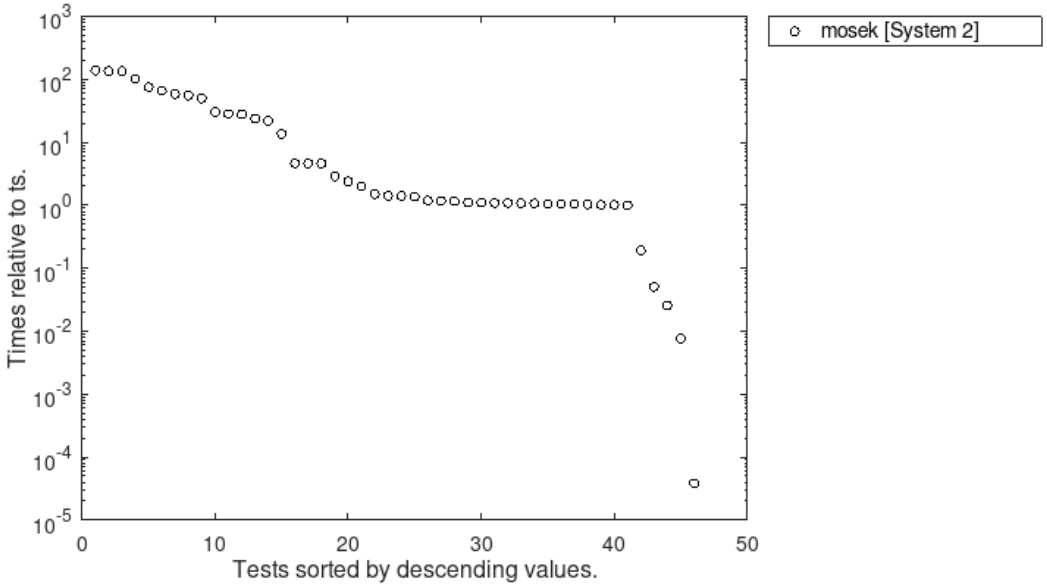


Figure 8: Computation times for $\underline{t}$ relative to t_s.

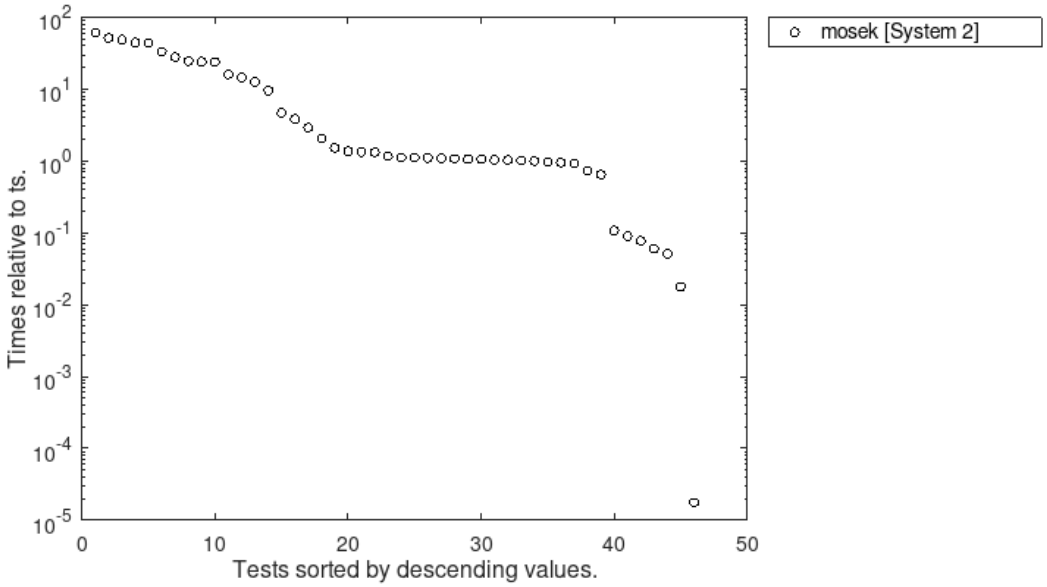


Figure 9: Computation times for $\bar{t}$ relative to t_s.

9.4 ESC

The ESC library contains 47 semidefinite programs from electronic structure calculations in quantum chemistry. In particular, the semidefinite programs are a relaxation of a variational approach for computing the ground state energy of N-electron molecules. For more details see [25]. The size of the resulting problems ranges between 100,000 and 2 million variables, and between 948 and 7230 constraints.

Approximate solutions and rigorous bounds were obtained for all problem instances, with the exception of the test problem *CF*, where the problem data is inconsistent.

Detailed results can be found in the table for System 2. All energy values are given in Hartree units and are the negative computed values plus the nuclear repulsion energy, see [25] for details.

In the table, the rigorous upper and lower error bounds of the optimal value are denoted by $\overline{E}$, $\underline{E}$, and $\underline{E}_2$, respectively. The value $\underline{E}_2$ is the rigorous lower energy error bound calculated by using our a priori eigenvalue bounds as derived in [6]. The quantities $\tilde{t}$, $\overline{t}$, $\underline{t}$, and $\underline{t}_2$ denote the running times in seconds for E_p and E_d, $\overline{E}$, $\underline{E}$, and $\underline{E}_2$, respectively.

A statistic of these numerical experiments is given in the following figures.

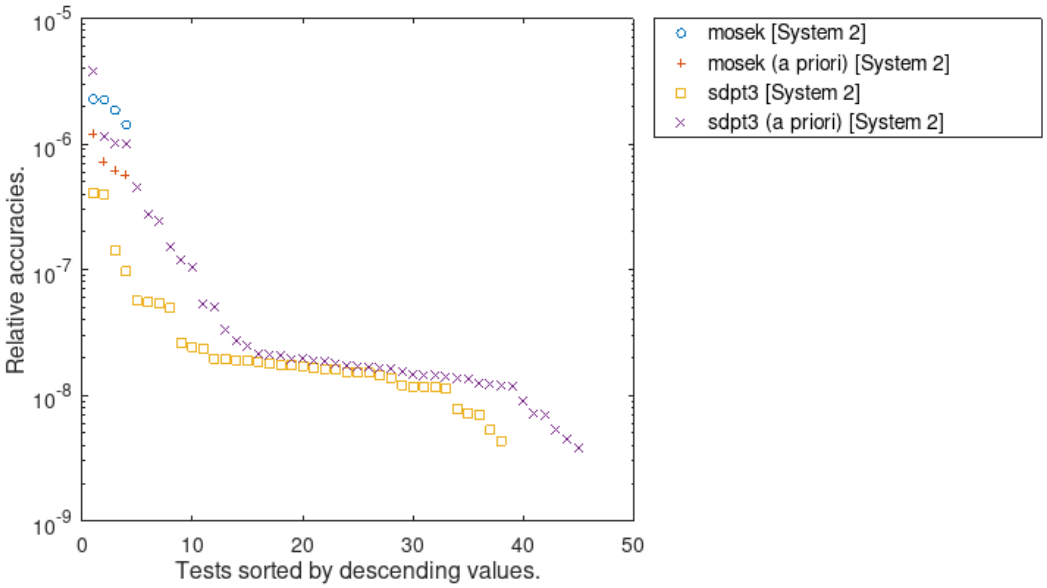


Figure 10: Relative accuracies $\mu(\overline{f_d}, \underline{f_p})$. Only results for which both rigorous error bounds were computed are taken into account.

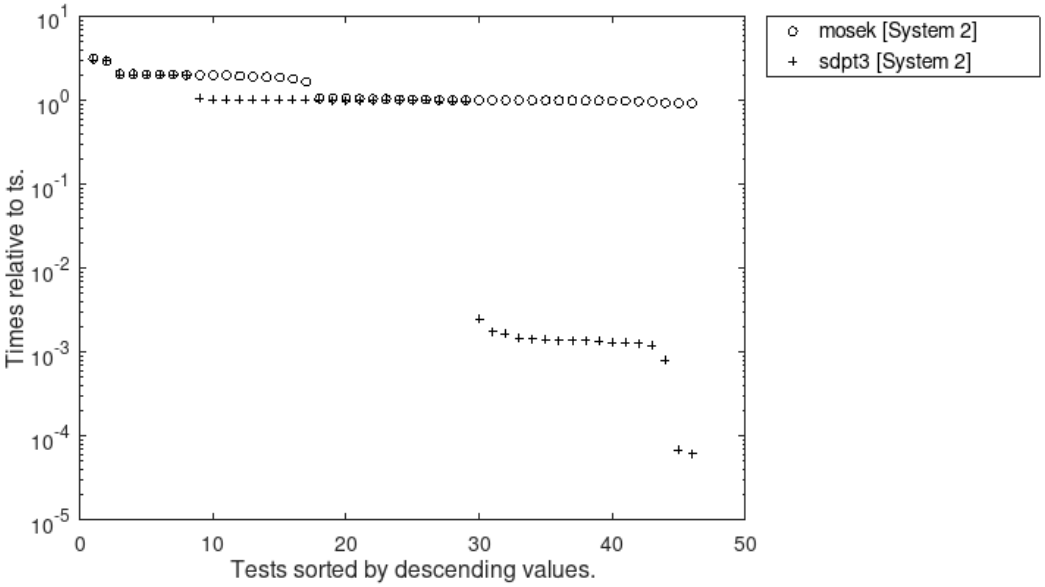


Figure 11: Computation times for $\underline{t}$ relative to t_s.

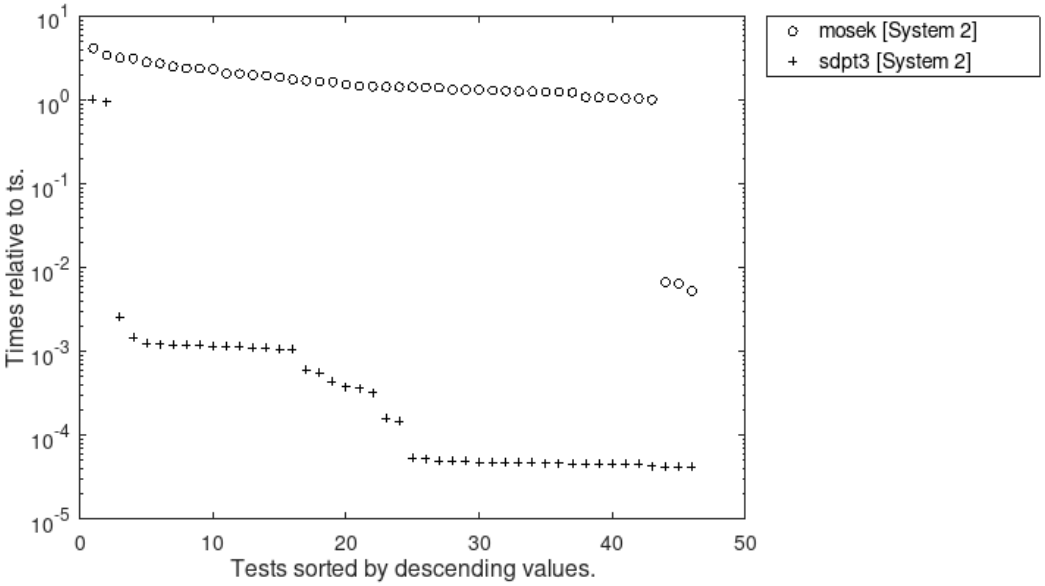


Figure 12: Computation times for $\bar{t}$ relative to t_s.

9.5 RDM

The RDM library [18] contains eight additional and larger problem instances, compared to the ESC benchmark library. The size of the resulting semidefinite problems ranges between 2 million and 19 million variables, and between 7230 and 27888 constraints.

Using the same notation as for the ESC table, the detailed results can be found for System 2.

CHAPTER

TEN

CONIC SOLVERS

In this section we provide some notes about our usage experience with VSDP and the following approximate conic solvers.

For our tests we tried the following four combinations to run the conic solver:

```
+---------------------------+
|      <conic solver>       |
+---------------------------+
|       VSDP & INTLAB       |
+------------+--------------+
|   MATLAB   |  GNU Octave  |
+------------+--------------+
```

10.1 CSDP

- Website: https://github.com/coin-or/Csdp
- Documentation: https://github.com/coin-or/Csdp/blob/master/doc/csdpuser.pdf
- *Cones*: (Free variables), LP, SDP
- *Installation*: Binary distributions for Windows and Linux on the website. Extract binary distribution to arbitrary location and use `addpath` within Octave or MATLAB to add the `bin` (solver executables) and `matlab` (interface routines) subdirectories.
- *Invocation*: Call `csdp` from the Octave or MATLAB command prompt.
- *Notes*: Free variables are only supported as difference of LP variables. The resulting problem is ill-posed.

10.2 GLPK

- Website: https://www.gnu.org/software/glpk
- Documentation: Part of the source code archive available from the website.
- *Cones*: Free variables, LP
- *Installation*: Built-in solver of GNU Octave.
- *Invocation*: Call `glpk` from the Octave command prompt.
- *Notes*: Not available for MATLAB.

10.3 LINPROG

- Website: https://www.mathworks.com/help/optim/ug/linprog.html
- Documentation: See website.
- *Cones*: Free variables, LP
- *Installation*: Built-in solver of MATLAB.
- *Invocation*: Call `linprog` from the MATLAB command prompt.
- *Notes*: Not available for GNU Octave.

10.4 lp_solve

- Website: http://lpsolve.sourceforge.net/5.5/index.htm
- Documentation: See website.
- *Cones*: Free variables, LP
- *Installation*: Binary distributions for Windows and Linux are available from https://sourceforge.net/projects/lpsolve/files/lpsolve/5.5.2.5. For Linux it is easier to install the binary solver files from the respective distribution package manager. The Octave and MATLAB interface is available from https://github.com/vsdp/lp_solve or https://sourceforge.net/projects/lpsolve/files/lpsolve/5.5.2.5. Inside the `extra/octave/lpsolve` or `extra/matlab/lpsolve` subdirectory one has to follow the build instructions and has to use `addpath` within Octave or MATLAB to make it work.
- *Invocation*: Call `lp_solve` from the Octave or MATLAB command prompt.

10.5 MOSEK

- Website: https://www.mosek.com/
- Documentation: https://www.mosek.com/documentation/
- *Cones*: Free variables, LP, SOCP, SDP
- *Installation*: Binary distributions for Windows and Linux on the website. Very good description is given at https://docs.mosek.com/8.1/install/installation.html
- *Invocation*: Call `mosekopt` from the Octave or MATLAB command prompt.
- *Notes*: One has to obtain a *license*, which is gratis for personal and academic use. *No Octave.* There is some no longer maintained octmosek Octave package, that does notes work for recent Octave versions.

10.6 SDPA

- Website: http://sdpa.sourceforge.net
- GitHub: https://github.com/vsdp/sdpa
- Documentation: https://sourceforge.net/projects/sdpa/files/sdpa/sdpa.7.1.1.manual.20080618.pdf
- Cones: LP, SDP
- Installation (Windows): Binary distributions can be obtained from http://sdpa.sourceforge.net/download.html (use the *SDPA-M* version). Extract binary distribution to arbitrary location and use `addpath` within MATLAB to add the solver and interface routines.
- *Invocation*: Call `mexsdpa` from the Octave or MATLAB command prompt.
- *Notes*: The "native" installation for Windows and MATLAB is the most reliable. The other three combinations worked partially or not at all. Some of the Linux efforts are reflected in the GitHub repository.

10.7 SDPT3

- Website: https://blog.nus.edu.sg/mattohkc/softwares/sdpt3/
- GitHub: https://github.com/sqlp/sdpt3 (preferred) or https://github.com/Kim-ChuanToh/SDPT3
- Documentation: https://blog.nus.edu.sg/mattohkc/files/2019/10/guide4-0-draft.pdf
- Cones: Free variables, LP, SOCP, SDP
- Installation: Download the files from the preferred GitHub repository extracted to an arbitrary location and run `install_sdpt3` from the Octave or MATLAB command prompt.
- Invocation: Call `sqlp` from the Octave or MATLAB command prompt.
- Notes: In case of errors with the MEX-Interface, run `install_sdpt3 -rebuild` from the Octave or MATLAB command prompt. The SDPT3 function `randmat` is in conflict with the INTLAB function `randmat`. To resolve that conflict, we chose to rename the INTLAB function.

10.8 SeDuMi

- Website: http://sedumi.ie.lehigh.edu
- GitHub: https://github.com/sqlp/sedumi (preferred)
- Documentation: http://sedumi.ie.lehigh.edu/sedumi/files/sedumi-downloads/SeDuMi_Guide_11.pdf and http://sedumi.ie.lehigh.edu/sedumi/files/sedumi-downloads/usrguide.ps
- Cones: Free variables, LP, SOCP, SDP
- Installation: Download the files from the preferred GitHub repository extracted to an arbitrary location and run `install_sedumi` from the Octave or MATLAB command prompt.
- Invocation: Call `sedumi` from the Octave or MATLAB command prompt.
- Notes: In case of errors with the MEX-Interface, run `install_sedumi -rebuild` from the Octave or MATLAB command prompt.

BIBLIOGRAPHY

[1] F. Alizadeh and D. Goldfarb. Second-order cone programming. *Mathematical Programming*, 95(1):3–51, 2003. doi:10.1007/s10107-002-0339-5.

[2] E. D. Andersen. Handling free variables in primal-dual interior-point methods using a quadratic cone. 2002. URL: https://meetings.siam.org/sess/dsp_talk.cfm?p=3815.

[3] M. Anjos and S. Burer. On Handling Free Variables in Interior-Point Methods for Conic Linear Optimization. *SIAM Journal on Optimization*, 18(4):1310–1325, 2007. doi:10.1137/06066847X.

[4] B. Borchers. CSDP 6.2.0 User's Guide. 2017. URL: https://github.com/coin-or/Csdp/blob/master/doc/csdpuser.pdf.

[5] Brian Borchers. CSDP 2.3 user's guide. *Optimization Methods and Software*, 11(1-4):597–611, 1999. doi:10.1080/10556789908805764.

[6] D. Chaykin, C. Jansson, F. Keil, M. Lange, K. T. Ohlhus, and S. M. Rump. Rigorous results in electronic structure calculations. 2016. URL: http://www.optimization-online.org/DB_HTML/2016/11/5730.html.

[7] L. El Ghaoui and H. Lebret. Robust Solutions to Least-Squares Problems with Uncertain Data. *SIAM Journal on Matrix Analysis and Applications*, 18(4):1035–1064, 1997. doi:10.1137/S0895479896298130.

[8] Robert M. Freund, Fernando Ordóñez, and Kim-Chuan Toh. Behavioral measures and their correlation with IPM iteration counts on semi-definite programming problems. *Mathematical Programming*, 109(2):445–475, 2007. doi:10.1007/s10107-006-0035-y.

[9] V. Härter, C. Jansson, and M. Lange. VSDP: A Matlab toolbox for verified semidefinite-quadratic-linear programming. 2012. URL: http://www.optimization-online.org/DB_HTML/2013/01/3724.html.

[10] C. Jansson. VSDP: Verified SemiDefinite Programming. 2006. URL: http://www.optimization-online.org/DB_HTML/2006/12/1547.html.

[11] C. Jansson, D. Chaykin, and C. Keil. Rigorous Error Bounds for the Optimal Value in Semidefinite Programming. *SIAM Journal on Numerical Analysis*, 46(1):180–200, 2007. doi:10.1137/050622870.

[12] Christian Jansson. A Rigorous Lower Bound for the Optimal Value of Convex Optimization Problems. *Journal of Global Optimization*, 28(1):121–137, 2004. doi:10.1023/B:JOGO.0000006720.68398.8c.

[13] Christian Jansson. Guaranteed Accuracy for Conic Programming Problems in Vector Lattices. *arXiv:0707.4366 [math]*, 2007. arXiv:0707.4366.

[14] Christian Jansson. On verified numerical computations in convex programming. *Japan Journal of Industrial and Applied Mathematics*, 26(2):337–363, 2009. doi:10.1007/BF03186539.

[15] Kazuhiro Kobayashi, Kazuhide Nakata, and Masakazu Kojima. A conversion of an SDP having free variables into the standard form SDP. *Computational Optimization and Applications*, 36(2):289–307, 2007. doi:10.1007/s10589-006-9002-z.

[16] M. Kočvara. On the modelling and solving of the truss design problem with global stability constraints. *Structural and Multidisciplinary Optimization*, 23(3):189–203, 2002. doi:10.1007/s00158-002-0177-3.

[17] Csaba Mészáros. On free variables in interior point methods. *Optimization Methods and Software*, 9(1-3):121–139, 1998. doi:10.1080/10556789808805689.

[18] Maho Nakata, Bastiaan J. Braams, Katsuki Fujisawa, Mituhiro Fukuda, Jerome K. Percus, Makoto Yamashita, and Zhengji Zhao. Variational calculation of second-order reduced density matrices by strong N-representability conditions and an accurate semidefinite programming solver. *The Journal of Chemical Physics*, 128(16):164113, 2008. doi:10.1063/1.2911696.

[19] F. Ordóñez and R. Freund. Computational Experience and the Explanatory Value of Condition Measures for Linear Optimization. *SIAM Journal on Optimization*, 14(2):307–333, 2003. doi:10.1137/S1052623402401804.

[20] G. Pataki and S. H. Schmieta. The DIMACS library of mixed semidefinite-quadratic-linear programs. Techreport, Columbia University, 2002. URL: http://dimacs.rutgers.edu/archive/Challenges/Seventh/.

[21] James Renegar. Some perturbation theory for linear programming. *Mathematical Programming*, 65(1):73–91, 1994. doi:10.1007/BF01581690.

[22] Siegfried M. Rump. INTLAB — INTerval LABoratory. In Tibor Csendes, editor, *Developments in Reliable Computing*, pages 77–104. Springer Netherlands, 1999. doi:10.1007/978-94-017-1247-7_7.

[23] Siegfried M. Rump. Verification methods: Rigorous results using floating-point arithmetic. *Acta Numerica*, 19:287–449, 2010. doi:10.1017/S096249291000005X.

[24] L. Vandenberghe and S. Boyd. Semidefinite Programming. *SIAM Review*, 38(1):49–95, 1996. doi:10.1137/1038003.

[25] Zhengji Zhao, Bastiaan J. Braams, Mituhiro Fukuda, Michael L. Overton, and Jerome K. Percus. The reduced density matrix method for electronic structure calculations and the role of three-index representability conditions. *The Journal of Chemical Physics*, 120(5):2095–2104, 2004. doi:10.1063/1.1636721.

[26] Jochem Zowe, Michal Kočvara, and Martin P. Bendsøe. Free material optimization via mathematical programming. *Mathematical Programming*, 79(1):445–466, 1997. doi:10.1007/BF02614328.

[27] A. Ben-Tal, M. Kovara, A. Nemirovski, and J. Zowe. Free Material Design via Semidefinite Programming: The Multiload Case with Contact Conditions. *SIAM Review*, 42(4):695–715, 2000. doi:10.1137/S0036144500372081.

[28] A. Ben-Tal and A. Nemirovski. *Lectures on Modern Convex Optimization*. MOS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics, 2001. ISBN 978-0-89871-491-3. doi:10.1137/1.9780898718829.